

SQL Intermediate

Prepared by



Global Business Intelligence Consulting and Training

Destiny Corporation
100 Great Meadow Rd Suite 601
Wethersfield, CT 06109-2379
Phone: (860) 721-1684 1-800-7TRAINING
Fax: (860) 721-9784
Email: info@destinycorp.com
Web: www.destinycorp.com

Objectives

The SQL Intermediate workshop is designed to take attendees into the intermediate uses of SQL, including Having, Full Joins and creation of Views, Indexes and Data Sets.

Joins

Joins are used to combine tables in a side-by-side fashion, analogous to a merge in a Data Step.

There are two types of Joins:

- Inner Joins
- Outer Joins

Inner Joins

An Inner Join (or more simply a Join) is the product of all the rows from one table with all the rows from another table or tables. Up to 32 tables can be joined at the same time.

The syntax for an Inner Join consists of a Select statement and a list of two or more table names in the From clause.

Cartesian Products

Example: Create Cartesian products for Tables A and B

Use these sample tables for illustration purposes.

Table A

Key	Agg1
aa	123
aa	345
bb	234
cc	789

Table B

Key	Agg2
aa	999
cc	888
cc	777
dd	666
dd	555

Submit the following code.

```
Command ==>
00001 data work.a;
00002 input key $ agg1;
00003 cards;
00004 aa      123
00005 aa      345
00006 bb      234
00007 cc      789
00008 run;
00009 data work.b;
00010 input key $ agg2;
00011 cards;
00012 aa      999
00013 cc      888
00014 cc      777
00015 dd      666
00016 dd      555
00017 run;
00018 proc sql;
00019 select *
00020 from a,b;
00021 quit;
```

This will produce the output displayed below.

key	agg1	key	agg2
aa	123	aa	999
aa	123	cc	888
aa	123	cc	777
aa	123	dd	666
aa	123	dd	555
aa	345	aa	999
aa	345	cc	888
aa	345	cc	777
aa	345	dd	666
aa	345	dd	555
bb	234	aa	999
bb	234	cc	888
bb	234	cc	777
bb	234	dd	666
bb	234	dd	555
cc	789	aa	999
cc	789	cc	888
cc	789	cc	777
cc	789	dd	666
cc	789	dd	555

This join is known as a Cartesian product.

Every row from the first table has been combined with every row from the second table.

Inner Join on Matching Key Fields

Example: Inner Join on matching key fields for Tables A and B.

Often a full Cartesian product is not required.

To restrict how rows are Joined a Where clause is added.

The Where clause is used to select rows where the key fields match.

It is useful to construct a chart listing the variables in the tables to be joined together. This identifies key variables.

For this example the chart would look like the following.

work.a
key
agg1

The following code performs the Inner Join.

```
Command ==> |
00001 proc sql;
00002   select *
00003   from a,b
00004   where a.key = b.key;
00005 quit;
```

key	agg1	key	agg2
aa	123	aa	999
aa	345	aa	999
cc	789	cc	888
cc	789	cc	777

Notes

- Compare this output with that shown above for the full Cartesian product, noting the values of the key column.

A Join with a Where clause may be considered as being constructed via an intermediate Cartesian product, from which the selection of rows is then made according to the Where criteria.

This is not the way it happens physically. It does give a good mental image to help predict the results from joins.

- Since there are two columns with the same name (key) the ambiguity as to which table this variable is to be taken from is resolved by appending the table name to the column name.

The syntax is table-name.column-name. The following code illustrates this concept.

```
Command ==> |
00001 proc sql;
00002   select a.key, agg1, agg2
00003   from a,b
00004   where a.key=b.key;
00005 quit;
```

key	agg1	agg2
aa	123	999
aa	345	999
cc	789	888
cc	789	777

The following code performs a traditional match-merge on the data sets.

```
Command ==> |
00001 data c;
00002   merge a b;
00003   by key;
00004 run;
00005
00006 title 'Match Merge';
00007 proc print data=c;
00008 run;
```

Match Merge

Obs	key	agg1	agg2
1	aa	123	999
2	aa	345	999
3	bb	234	.
4	cc	789	888
5	cc	789	777
6	dd	.	666
7	dd	.	555

Note that the match-merge takes all unique observations from each data set as well as combining observations with the same common variable.

Compare the results of the match-merge with that of the SQL Inner Join.

How could the match-merge be changed to obtain the same results as the SQL Inner Join?

Try the following code.

```
Command ==> |
00001 data c;
00002   merge a(in=one)
00003         b(in=two);
00004   by key;
00005   if one and two;
00006 run;
```

The log will display the following.

```
Log - (Untitled)
Command ==>
92 data c;
93   merge a(in=one)
94         b(in=two);
95   by key;
96   if one and two;
97 run;
```

NOTE: There were 4 observations read from the data set WORK.A.
NOTE: There were 5 observations read from the data set WORK.B.
NOTE: The data set WORK.C has 4 observations and 3 variables.
NOTE: DATA statement used:
real time 0.02 seconds
cpu time 0.02 seconds

Example: Inner Join on matching key fields for Garden Supply Company.

Combine the information from two tables, Saved.Products and Saved.Orders.

Saved.Products contains information on all products produced at a gardening company while Saved.Orders contains information on all products currently on order at this company.

The following chart identifies the variables contained in each table.

Saved.Products	Saved.Orders
Prodno	Custno
Stock	Prodno
Proddesc	Orderno
Unitwt	Packsord
Packsz	Dateord
Price	
Carrier	

The key or variable common to both data sets is Prodno.

The two data sets are displayed below.

Saved.Products has the following values.

VIEWTABLE: saved.products	PRODNO	STOCK	PRODDESC	UNITWT	PACKSZ	PRICE	CARRIER
1	F40019	0	TABLE	24	1	24	OVERNITE
2	F40020	9	CHAIR	14	4	9.99	OVERNITE
3	F40021	15	LOUNGER	19	1	14	OVERNITE
4	F40022	3	UMBRELLA	12	1	13	OVERNITE
5	H10098	0	15 METER DOMESTIC	6.5	5	11	ERS
6	H10099	2	30 METER DOMESTIC	12.5	2	21	ERS
7	H20048	0	15 METER COMMERCIAL	8.6	1	15	ERS
8	H20049	4	30 METER COMMERCIAL	16	1	27	OVERNITE
9	H20054	3	45 METER COMMERCIAL	23	1	33	STATE EXPRESS
10	H20055	2	60 METER COMMERCIAL	31.5	1	46	STATE EXPRESS
11	H20056	0	30 METER HEAVY DUTY	22	1	55	BANG
12	H20057	2	60 METER HEAVY DUTY	44	1	99	BANG
13	T30023	0	BALL WHEEL BARROW	32.5	1	17.5	ERS
14	T30024	6	HOE	5.4	4	13	SPEEDY
15	T30025	5	TROWEL	2.3	10	7.99	SPEEDY
16	T30026	10	SPADE	12.6	4	19	SPEEDY
17	T30027	10	FORK	10	4	17	BANG
18	T30028	10	RAKE	6.3	4	15	BANG

Saved.Orders has the following values.

VIEWTABLE: saved.orders	CUSTNO	PRODNO	ORDERNO	PACKSORD	DATEORD
1	199	H10098	12	6	11036
2	199	H10098	13	6	11037
3	199	H10098	14	6	11038
4	199	H10098	15	6	11039
5	199	H10099	10	6	11038
6	199	T30023	109	5	11042
7	202	F40019	55	3	11035
8	202	F40019	56	3	11036
9	202	F40019	134	3	11036
10	308	F40019	29	20	11030
11	308	F40019	30	4	11031
12	308	F40020	31	3	11030
13	308	F40021	66	2	11030
14	308	F40021	126	2	11031
15	401	T30025	54	2	11030
16	401	T30025	56	2	11031
17	401	T30025	57	2	11032
18	401	T30026	30	2	11031
19	401	T30027	22	8	11031

Note

- A customer can place multiple orders.
- Customers place orders in packs while stock is held as individual items.

What method can be used to calculate the weight of each order?

To accomplish this, the product in Saved.Orders needs to be combined with the following variables from Saved.Products:

- Packsz (package size)
- Packsord (number of packages)
- Unitwt (weight of each unit)

The code is displayed below.

```
Command ==> |
00001 proc sql;
00002     select p.prodno label='Product Number',
00003            custno label='Customer Number',
00004            packs*packsord*unitwt as orderwt
00005            label='Consignment Weight'
00006            format=5.1
00007     from saved.products p, saved.orders o
00008     where p.prodno = o.prodno;
00009 quit;
```

Notes

- The Where clause selects only those observations with matching values of Prodno.
- An Alias is a table nickname, which can be used to reference a table in place of the entire table name. An Alias is assigned to a table following the table name in the From clause. In this example, the Alias p was assigned for the table Saved.Products and the Alias o was assigned for the table Saved.Orders.
- Since there are two columns with the same name (Prodno) the ambiguity as to which table this variable is to be taken from is resolved by appending the table Alias to the column name. Notice that the table Alias can be used in the Select statement before it is defined in the From clause.

The output is displayed below.

Product Number	Customer Number	Consignment Weight
F40019	202	72.0
F40019	202	72.0
F40019	202	72.0
F40019	308	480.0
F40019	308	96.0
F40020	308	168.0
F40021	308	38.0
F40021	308	38.0
H10098	199	195.0
H10098	199	195.0
H10098	199	195.0
H10098	199	195.0
H10099	199	150.0
T30023	199	162.5
T30025	401	46.0
T30025	401	46.0
T30025	401	46.0
T30026	401	100.8
T30027	401	320.0

The SQL result is the same as that from a traditional match merge with an In= option.

Submit the following code.

```
Command ==>
00001 proc sort data=saved.products out=prods;
00002   by prodno;
00003 run;
00004 proc sort data=saved.orders out=ords;
00005   by prodno;
00006 run;
00007 data new;
00008   merge prods ords(in = inords);
00009   by prodno;
00010   if inords;
00011   orderwt = packs*packsord*unitwt;
00012   keep prodno custno orderwt;
00013 run;
00014 proc print data = new label noobs;
00015   label prodno = 'Product Number'
00016         custno = 'Customer Number'
00017         orderwt = 'Consignment Weight';
00018   format orderwt 5.1;
00019 run;
```

This gives the following output.

Product Number	Customer Number	Consignment Weight
F40019	202	72.0
F40019	202	72.0
F40019	202	72.0
F40019	308	480.0
F40019	308	96.0
F40020	308	168.0
F40021	308	38.0
F40021	308	38.0
H10098	199	195.0
H10098	199	195.0
H10098	199	195.0
H10098	199	195.0
H10099	199	150.0
T30023	199	162.5
T30025	401	46.0
T30025	401	46.0
T30025	401	46.0
T30026	401	100.8
T30027	401	320.0

Outer Joins

An Outer join is used to select matching and non-matching rows from tables

Outer Joins are performed on two tables at a time.

There are three types of Outer Joins:

- Left Outer Join.
- Right Outer Join.
- Full Outer Join.

The following sections will illustrate the different types of Outer Joins.

Left Outer Join

Consider Table A and Table B.

Table A

Key	Agg1
aa	123
aa	345
bb	234
cc	789

Table B

Key	Agg2
aa	999
cc	888
cc	777
dd	666
dd	555

The following code performs a Left Outer Join using the above tables.

```
Command ==>
00001 proc sql;
00002   title 'Left Outer Join';
00003   select *
00004   from a left join b
00005   on a.key=b.key;
00006 quit;
```

Left Outer Join

key	agg1	key	agg2
aa	123	aa	999
aa	345	aa	999
bb	234		
cc	789	cc	777
cc	789	cc	888

Notes

- The output contains all rows from the left hand table as well as rows matching on the key variable from the right hand table.
- The syntax consists of a Join operator in between the table names in the From clause.

For a Left Outer Join the operator is Left.

There are three possible operators corresponding to the three possible Outer Joins.

- Instead of a Where clause an On clause is used to select matching rows.

Right Outer Join

Table A

Key	Agg1
aa	123
aa	345
bb	234
cc	789

Table B

Key	Agg2
aa	999
cc	888
cc	777
dd	666
dd	555

The following code performs a Right Outer Join using the above tables.

```
Command ==>
00001 proc sql;
00002   title 'Right Outer Join';
00003   select *
00004   from a right join b
00005   on a.key=b.key;
00006 quit;
```

Right Outer Join

key	agg1	key	agg2
aa	123	aa	999
aa	345	aa	999
cc	789	cc	777
cc	789	cc	888
		dd	555
		dd	666

Notes

- The output contains all rows from the right hand table as well as rows matching on the key variable from the left hand table.
- The Join operator used is Right.

Full Outer Join

Table A

Key	Agg1
aa	123
aa	345
bb	234
cc	789

Table B

Key	Agg2
aa	999
cc	888
cc	777
dd	666
dd	555

The following code performs a Full Outer Join using the preceding tables.

```
Command ==> |
00001 proc sql;
00002   title 'Full Outer Join';
00003   select *
00004   from a full join b
00005   on a.key=b.key;
00006 quit;
```

Full Outer Join

key	agg1	key	agg2
aa	123	aa	999
aa	345	aa	999
bb	234		
cc	789	cc	777
cc	789	cc	888
		dd	555
		dd	666

Notes

- The output contains non-matching rows from both tables as well as the matching rows.
- For a Full Outer Join the Join operator appearing between the table names in the From clause is Full Join.
- Notice that the variable key is displayed twice. This occurs because both data sets contain the variable. The next example introduces the Coalesce function which can be used to address this issue.

Incorporating the Coalesce Function

Consider the tables displayed below.

Table A

Key	Agg1
aa	123
aa	345
bb	234
cc	789

Table B

Key	Agg2
aa	999
cc	888
cc	777
dd	666
dd	555

The following code compares a Proc SQL Full Outer Join using a Coalesce function with a data step match-merge.

Submit the code below.

```
Command ==> |
00001 proc sql;
00002   title 'Full Outer Join';
00003   select coalesce(a.key,b.key) label = 'Key',agg1,agg2
00004   from a full join b
00005   on a.key=b.key;
00006 run;
00007 data new;
00008   merge a b;
00009   by key;
00010   title 'Match Merge';
00011 run;
00012 proc print data=new;
00013 run;
```

The following output is displayed.

Full Outer Join

Key	agg1	agg2
aa	123	999
aa	345	999
bb	234	
cc	789	777
cc	789	888
dd		555
dd		666

Match Merge

Obs	key	agg1	agg2
1	aa	123	999
2	aa	345	999
3	bb	234	
4	cc	789	888
5	cc	789	777
6	dd		666
7	dd		555

Notes

- The output contains non-matching rows and matching rows, from both tables.
- The Coalesce function used in Proc SQL eliminates the second key column from the output.
- The syntax for the Coalesce function is Coalesce(col1,col2). All arguments must be of the same data type.
- The Coalesce function overlays two columns. It displays the column with the first non-missing value.

Appending the Table Name to the Column Name in the Select Statement

A second way to display a single column for the variable key is to append the table name to the column name in the select statement.

Compare the previous output to that from the following code.

```
Command ==> |
00001 proc sql;
00002   select a.key,agg1,agg2
00003   from a full join b
00004        on a.key=b.key;
00005 quit;
```

Match Merge		
key	agg1	agg2
aa	123	999
aa	345	999
bb	234	.
cc	789	777
cc	789	888
		555
		666

Example: Left Outer Join for Garden Supply Company

A garden supplies company maintains a data set called Saved.Apcusts, which contains information about its customers including address and credit rating data.

The company also has a data set called Saved.Orders, which contains information on all products currently on order.

The common or key variable between these two data sets is Custno (customer number).

The company wishes to create a report showing order information for all customers, including those customers who currently have no orders.

The following code performs a Left Outer Join to create the desired report.

```
Command ==> |
00001 proc sql;
00002   select c.customer label = 'Customer Name' ,
00003         orderno  label = 'Order Number' ,
00004         prodno   label = 'Product Number' ,
00005         packsord label = 'Packs Ordered'
00006   from saved.apcusts c left join saved.orders o
00007        on c.custno = o.custno;
00008 quit;
```

Customer Name	Order Number	Product Number	Packs Ordered
GREATBILLS	109	T30023	5
GREATBILLS	14	H10098	6
GREATBILLS	15	H10098	6
GREATBILLS	12	H10098	6
GREATBILLS	10	H10099	6
GREATBILLS	13	H10098	6
DILKES	55	F40019	3
DILKES	134	F40019	3
DILKES	56	F40019	3
HOMEPAD			.
PRICECUT			.
DOOLITTLE	126	F40021	2
DOOLITTLE	66	F40021	2
DOOLITTLE	31	F40020	3
DOOLITTLE	30	F40019	4
DOOLITTLE	29	F40019	20
BODGECITY			.
ARIZONA	54	T30025	2
ARIZONA	22	T30027	8
ARIZONA	30	T30026	2
ARIZONA	57	T30025	2
ARIZONA	56	T30025	2

Complex Queries

Instead of explicitly indicating a data set name in the From clause, a temporary data set can be specified by using an inner query.

This is known as an In-Line View.

The data set created by the inner query is temporary and exists only during the query execution.

Example: In-Line View for Garden Supply Company.

Consider the Left Outer Join example in the previous section.

Use the table resulting from this query to determine the total packs ordered per product per customer.

Use the derived table to be sure to include those customers that currently have no orders.

```
Command ==> |
00001 proc sql;
00002   select max(customer)label = 'Customer',
00003         prodno,
00004         sum(packsord)
00005         label='Total Order Quantity per Product'
00006   from (
00007     select c.customer as customer label = 'Customer',
00008           c.custno as custno  label = 'Customer Number',
00009           orderno  label = 'Order Number',
00010           prodno   label = 'Product Number',
00011           packsord label = 'Packs Ordered'
00012   from saved.apcusts c left join saved.orders o
00013        on c.custno = o.custno
00014   )
00015   group by custno,prodno;
00016 quit;
```

The output is displayed below.

Customer	Product Number	Total Order Quantity per Product
HOMEPAD		.
PRICECUT		.
BODGECITY		.
DILKES	F40019	9
DOOLITTLE	F40019	24
DOOLITTLE	F40020	3
DOOLITTLE	F40021	4
GREATBILLS	H10098	24
GREATBILLS	H10099	6
GREATBILLS	T30023	5
ARIZONA	T30025	6
ARIZONA	T30026	2
ARIZONA	T30027	8

Notes

- The Max function was used to eliminate repeats of customer when the same customer has multiple orders for the same product.
- Column names referenced in the inner query must apply in the outer.

For example, the reference c.customer cannot be made in the outer query because the alias does not exist for the in-line view, nor can the column be left without a name.

```
select c.customer
label='Customer';
```

This would not give a column name for the outer query to use.

Combining Three Tables

Data for the Garden Supply Company will be used for this example.

The goal is to determine the number of items ordered per product per customer, not just the number of packs.

The pack size is held on the Products table in a column called Packsz.

First, join the Orders table with the Products table, excluding those products for which there is no order.

Then join it with the Customer table but this time include customers with no orders.

```
Command ==> |
00001 proc sql;
00002 select customer label='Customer', /* (3) */
00003 orderno label='Order Number',
00004 proddesc label='Product Description',
00005 (packsord*packsz) as unitord
00006 label='Individual Order',
00007 sum(packsord*packsz) as totord
00008 label='Total Order per Item'
00009 from (
00010 select * /* (1) */
00011 from saved.products p, saved.orders o
00012 where p.prodno = o.prodno
00013 as inquiry right join saved.apcusts c
00014 on inquiry.custno = c.custno /* (2) */
00015 group by customer, proddesc;
00016 quit;
```

The output is displayed below.

Customer	Order Number	Product Description	Individual Order	Total Order per Item
ARIZONA	22	FORK	32	32
ARIZONA	30	SPADE	8	8
ARIZONA	56	TROWEL	20	60
ARIZONA	57	TROWEL	20	60
ARIZONA	54	TROWEL	20	60
BODGECITY			.	.
DILKES	55	TABLE	3	9
DILKES	56	TABLE	3	9
DILKES	134	TABLE	3	9
DOOLITTLE	31	CHAIR	12	12
DOOLITTLE	66	LOUNGER	2	4
DOOLITTLE	126	LOUNGER	2	4
DOOLITTLE	29	TABLE	20	24
DOOLITTLE	30	TABLE	4	24
GREATBILLS	14	15 METER DOMESTIC	30	120
GREATBILLS	15	15 METER DOMESTIC	30	120
GREATBILLS	13	15 METER DOMESTIC	30	120
GREATBILLS	12	15 METER DOMESTIC	30	120
GREATBILLS	10	30 METER DOMESTIC	12	12
GREATBILLS	109	BALL WHEEL BARROW	5	5
HOMEPAD			.	.
PRICECUT			.	.

Notes

- Getting the product description requires a join with the Products table. Getting the customer name requires a join with the Customer table.
- The code as inquiry gives an alias to the In-Line view.
- The right join ensures that customers with no current orders are included in the report.

The following code gives a summary report of the above output.

The code is the same other than the deletion of the selection of those columns, which define unique rows, specifically Orderno and Unitord.

```
Command ==> |
00001 proc sql;
00002 select customer label='Customer', /* (3) */
00003 proddesc label='Product Description',
00004 sum(packsord*packsz) as totord
00005 label='Total Order per Item'
00006 from (
00007 select * /* (1) */
00008 from saved.products p, saved.orders o
00009 where p.prodno = o.prodno
00010 as inquiry right join saved.apcusts c
00011 on inquiry.custno = c.custno /* (2) */
00012 group by customer, proddesc;
00013 quit;
```

Customer	Product Description	Total Order per Item
ARIZONA	FORK	32
ARIZONA	SPADE	8
ARIZONA	TROWEL	60
BODGEICITY		.
DILKES	TABLE	9
DOOLITTLE	CHAIR	12
DOOLITTLE	LOUNGER	4
DOOLITTLE	TABLE	24
GREATBILLS	15 METER DOMESTIC	120
GREATBILLS	30 METER DOMESTIC	12
GREATBILLS	BALL WHEEL BARROW	5
HOMEPAD		.
PRICECUT		.

Set Operations

Proc SQL can be used to combine two tables vertically. This is analogous to concatenation of data sets in traditional data step processing.

The syntax for Set operations is to have one of four keywords between two Select statements.

Set Operators

Set operators are key words that control how the tables are combined.

They are placed between the two Select statements.

The four Set operators are listed below:

- Except
- Intersect
- Union
- Outer Union

Modifying the Action of Set Operators

The following two modifiers can be used to alter the actions of a Set Operator.

- Corresponding (Corr)
- All

Notes

- Corresponding overlays columns by name, and not by position.
- All allows duplicate rows to be output.
- The All and Corresponding keywords are placed after the Set operator keyword.

Except

- Select rows from the first table that are not found in the second table.
The output produces rows that are unique to the first table.
- Column names are taken from the first table.
- Columns are matched by position not column names.

Use the following two tables.

Table A

Key	Agg1
aa	123
aa	999
bb	234
cc	789
cc	888

Table B

Key	Agg2
aa	999
cc	888
cc	777
dd	666
dd	555

Example 1: Except Operator to combine tables A and B.

```

Command ==>
00001 data work.a;
00002 input key $ agg1;
00003 cards;
00004 aa 123
00005 aa 999
00006 bb 234
00007 cc 789
00008 cc 888
00009 run;
00010 data work.b;
00011 input key $ agg2;
00012 cards;
00013 aa 999
00014 cc 888
00015 cc 777
00016 dd 666
00017 dd 555
00018 run;
00019 proc sql;
00020 title 'Except Operator';
00021 select *
00022 from a
00023 EXCEPT
00024 select *
00025 from b;
00026 quit;

```

Except Operator	
key	agg1
aa	123
bb	234
cc	789

Notes

- The output has rows unique to table A.
- Column names have been taken from the first table.
- The columns have been matched by position.

Example 2: Except operator to combine tables for Garden Supply Company.

The Products table has information on all products sold by a gardening company.

By comparing this with the Orders table, which has data about products currently on order, products for which there are no orders can be determined.

```

Command ==>
00001 proc sql;
00002   title 'Products for which there are no Orders';
00003   select *
00004   from saved.products
00005   except corresponding
00006   select *
00007   from saved.orders;
00008 quit;

```

Products for which there are no Orders	
PRODNO	
F40022	
H20048	
H20049	
H20054	
H20055	
H20056	
H20057	
T30024	
T30028	

Notes

- The output has rows unique to table Saved.Products; that is, products not on order.
- The Corresponding option overlays columns by name, not position.
- Corresponding removes columns not found in both tables when used with the Except operator.
- This particular query can be performed more efficiently with a Sub-query.

```

Command ==>
00001 proc sql;
00002   select prodno label= 'Product Number'
00003   from saved.products
00004   where prodno not in
00005   (select prodno from saved.orders);
00006 quit;

```

Intersect

- Select rows from the first table that are also found in the second table.
- The output produces rows common to both tables.
- Column names are taken from the first table.
- Columns are matched by position not column names.

Table A

Key	Agg1
aa	123
aa	999
bb	234
cc	789
cc	888

Table B

Key	Agg2
aa	999
cc	888
cc	777
dd	666
dd	555

Example 3: Intersect Operator to combine tables A and B.

```

Command ==>
00001 proc sql;
00002   title 'Intersect Operator';
00003   select *
00004   from a
00005   intersect
00006   select *
00007   from b;
00008 quit;

```

Intersect Operator	
key	agg1
aa	999
cc	888

Notes

- The output has rows common to Table A and Table B.
- Column names have been taken from the first table.
- The columns have been matched by position.

Example 4: Intersect operator to combine tables for Garden Supply Company.

Show those products with orders as listed in the Orders table, which have product numbers listed in the Products table.

```

Program Editor - sql099
Command ==>
00001 proc sql;
00002   title 'Intersect Operator';
00003   select *
00004   from saved.orders
00005   intersect all corresponding
00006   select *
00007   from saved.products;
00008 quit;

```

Intersect Operator	
PRODNO	
F40019	
F40020	
F40021	
H10098	
H10099	
T30023	
T30025	
T30026	
T30027	

Notes

- The output has rows common to Saved.Orders and Saved.Products.
- Corresponding overlays columns by name, not position.
- Corresponding removes columns not found in both tables when used with the Intersect operator.
- All allows duplicate rows to be output. Because Saved.Products has one row per product, there are no duplicate rows in this example. As a result, the output will be the same with or without the All keyword. System performance is improved however, because SAS does not check for duplicates when the All option is used.

Union

- Selects all rows from both tables; excludes duplicate rows.
- Column names are taken from the first table.
- Columns are matched by position not column names.

Use the two tables displayed below.

Table A

Key	Agg1
aa	123
aa	999
bb	234
cc	789
cc	888

Table B

Key	Agg2
aa	999
cc	888
cc	777
dd	666
dd	555

Example 5: Union operator to combine tables A and B.

```
Command ==>
00001 proc sql;
00002   title 'Union Operator';
00003   select *
00004   from a
00005   union
00006   select *
00007   from b;
00008 quit;
```

Union Operator

key	agg1
aa	123
aa	999
bb	234
cc	777
cc	789
cc	888
dd	555
dd	666

The All and Corresponding options can be used with the Union operator.

Using the All Option with the Union Operator

```
Command ==>
00001 proc sql;
00002   title 'Union Operator - All';
00003   select * from a
00004   union all
00005   select * from b;
00006 quit;
```

Union Operator - All

key	agg1
aa	123
aa	999
bb	234
cc	789
cc	888
aa	999
cc	888
cc	777
dd	666
dd	555

Notes

- With the All option, two duplicate rows with a key value of cc and an agg1 value of 888 are output.

Using Corresponding Option with the Union Operator

```
Command ==>
00001 proc sql;
00002   title 'Union Operator - Corr';
00003   select * from a
00004   union corr
00005   select * from b;
00006 quit;
```

Union Operator - Corr

key
aa
bb
cc
dd

Notes

- The Corresponding option overlaid columns by name.
- Corresponding removed columns not found in both tables.

Outer Union

The Outer Union selects all rows from both tables, including duplicate rows.

It also selects all columns from both tables. If two tables have columns with the same name, both columns will be selected.

The All option cannot be used as this operation is implied.
Duplicate rows are included in the output, by default.

Table A

Key	Agg1
aa	123
aa	999
bb	234
cc	789
cc	888

Table B

Key	Agg2
aa	999
cc	888
cc	777
dd	666
dd	555

Example 6: Outer Union to combine tables A and B.

```
Command ==> |
00001 proc sql;
00002   title 'Outer Union Operator';
00003   select * from work.a
00004   outer union
00005   select * from work.b;
00006 quit;
```

Outer Union Operator

key	agg1	key	agg2
aa	123		
aa	999		
bb	234		
cc	789		
cc	888		
		aa	999
		cc	888
		cc	777
		dd	666
		dd	555

Example 7: Outer Union Operator with Corr Option.

If the Corr option is used with the Outer Union operator, common columns are overlaid, resulting in the same output as concatenation within the data step.

```
Command ==> |
00001 proc sql;
00002   title 'Outer Union Operator - Corr';
00003   select * from a
00004   outer union corr
00005   select * from b;
00006 quit;
```

Outer Union Operator - Corr

key	agg1	agg2
aa	123	.
aa	999	.
bb	234	.
cc	789	.
cc	888	.
aa	.	999
cc	.	888
cc	.	777
dd	.	666
dd	.	555

Compare the previous output with the following data step concatenation.

```
Command ==> |
00001 data new;
00002   set a b;
00003   title 'Comparative Data Step Concatenation';
00004 run;
00005 proc print data=new;
00006 run;
```

Comparative Data Step Concatenation

Obs	key	agg1	agg2
1	aa	123	.
2	aa	999	.
3	bb	234	.
4	cc	789	.
5	cc	888	.
6	aa	.	999
7	cc	.	888
8	cc	.	777
9	dd	.	666
10	dd	.	555

Notes on Combining Tables with Proc SQL Set Operators

- Set operators usually require more resources than the corresponding data/proc step processing.
- Multiple data/proc steps can often be combined in one SQL query.
- SQL Set operators work on two tables at a time although it is possible to chain multiple SQL Set operators together.
- With Proc SQL, a table cannot be opened for input and output simultaneously.
- With SQL Set operators, use the All option where possible if outputting duplicate rows is acceptable.

Creating Tables, Views and Indexes

Creating Tables

There are three ways to create tables:

- As a result of a query expression
- Using another table as a template
- By defining the structure.

Method 1: As a result of a query expression

```
Command ==>
00001 proc sql;
00002   create table bigcust as
00003   select custno,
00004          prodno,
00005          packsord
00006   from saved.orders
00007   group by custno
00008   having count(*) > 3;
00009   select * from bigcust;
00010 quit;
```

```
Log - (Untitled)
Command ==> |
NOTE: Table WORK.BIGCUST created, with 16 rows and 3 columns.
```

Note: This method transfers the data as well as the structure of the table.

Method 2: Using another table as a template

```
Command ==> |
00001 proc sql;
00002   create table stock like
00003   saved.products(keep=prodno stock);
00004   select * from stock;
00005 quit;
```

```
Log - (Untitled)
Command ==> |
NOTE: Table WORK.STOCK created, with 0 rows and 2 columns.
103   select * from stock;
NOTE: Writing HTML Body file: sashtml.htm
NOTE: No rows were selected.
```

Method 3: By defining the structure.

```
Command ==>
00001 proc sql;
00002   create table perform
00003   (prodno char(6) label='Product Number',
00004    diameter num informat=5.2 format=5.2,
00005    lit_min num informat=3. format=3.);
00006 quit;
```

Note the specification of variables and their attributes.

We see how to load up a table with data in the next section.

```
Log - (Untitled)
Command ==> |
NOTE: Table WORK.PERFORM created, with 0 rows and 3 columns.
```

Creating Views

A View deriving from Proc SQL (a native View as opposed to an interface View derived from Proc Access) is simply a Query expression that is given a name.

```
Command ==> |
00001 proc sql;
00002   create view work.totwt as
00003   select prodno,
00004          proddesc,
00005          (unitwt*packsz) as packwt label='Pack Weight'
00006   from saved.products;
00007 quit;
00008 proc print data=work.totwt noobs label;
00009 run;
```

PRODNO	PRODDISC	Pack Weight
F40019	TABLE	24.0
F40020	CHAIR	56.0
F40021	LOUNGER	19.0
F40022	UMBRELLA	12.0
H10098	15 METER DOMESTIC	32.5
H10099	30 METER DOMESTIC	25.0
H20048	15 METER COMMERCIAL	8.6
H20049	30 METER COMMERCIAL	16.0
H20054	45 METER COMMERCIAL	23.0
H20055	60 METER COMMERCIAL	31.5
H20056	30 METER HEAVY DUTY	22.0
H20057	60 METER HEAVY DUTY	44.0
T30023	BALL WHEEL BARROW	32.5
T30024	HOE	21.6
T30025	TROWEL	23.0
T30026	SPADE	50.4
T30027	FORK	40.0
T30028	RAKE	25.2

Views:

- Can be used wherever a SAS data file/table can be used.
- Are stored Query Expressions, not stored data and so cannot be updated. We cannot insert rows, delete rows or update the values in a View, although a future release will allow updating of the original base tables via the view.
- Can be used to:
 - Save disk space.
 - Reduce the complexity of a complex query.
 - Be sure that live data is used.
 - Provide different users with different perspectives on the same data.
- Are executed internally to any query or procedure to which they provide input, building a virtual table in the process.
- Can be stored permanently by naming them with a Libref (just as for permanent tables).

Proc Contents Listing

Running Proc Contents on a View shows a missing value for the number of observations.

Note the Access Engine.

```
Command ==> |
00001 proc contents data = totwt;
00002 run;
```

The CONTENTS Procedure

Data Set Name:	WORK.TOTWT	Observations:	.
Member Type:	VIEW	Variables:	3
Engine:	SQLVIEW	Indexes:	0
Created:	15:44 Wednesday, December 27, 2000	Observation Length:	40
Last Modified:	15:44 Wednesday, December 27, 2000	Deleted Observations:	0
Protection:		Compressed:	NO
Data Set Type:		Sorted:	NO
Label:			

-----Alphabetic List of Variables and Attributes-----						
#	Variable	Type	Len	Pos	Flags	Label
2	PRODESC	Char	20	6	---	
1	PRODNO	Char	6	0	---	
3	packwt	Num	8	32	P--	Pack Weight

Describing a View

Once a View has been defined, it may be necessary to return and browse the query expression.

This can be done with the Describe statement.

```
Command ==>
00001 proc sql;
00002 describe view work.totwt;
00003 quit;
```



```
Log - (Untitled)
Command ==>
NOTE: SQL view WORK.TOTWT is defined as:

select prodno, proddesc, unitwt*packsz as packwt label='Pack Weight'
from SAVED.PRODUCTS;
```

Creating an Index

```
Command ==>
00001 proc sql;
00002 create index prodno on saved.products(prodno);
00003 create index ordno on saved.orders(custno,prodno,orderno);
00004 quit;
```



```
Log - (Untitled)
Command ==>
341 create index prodno on saved.products(prodno);
NOTE: Simple index prodno has been defined.
342 create index ordno on saved.orders(custno,prodno,orderno);
NOTE: Composite index ordno has been defined.
343 quit;
```

Notes on Indexes

- Simple Indexes are created using a single column, normally a unique key.
- For simple Indexes, the name of the Index must be the same as the name of the column.
- Composite Indexes are created using multiple columns.
- A table may have many Indexes defined on it. There is a processing overhead in maintaining the Indexes as the values in the table change.

- The Index stores information on the index columns as to where the actual observation lives inside of the SAS data set. Direct access to those observations is quick.
- Once an index has been defined, it is regarded as part of the table and the internal costing algorithm will determine its use. You cannot insist that an Index will be used, but can see whether it has been used, as follows:

```
options msglevel = 'I';
```

If you delete or insert rows into the table, or change any of the Index variable values in the table, the Index(es) will be automatically updated

- The Unique keyword can be used with both simple and composite Indexes. Such an Index can only be created when the index column or columns form a unique value.
- If this is not the case, the Index cannot be created. Such an Index will prevent the creation of duplicate rows.

```
Command ==>
00001 proc sql;
00002 create unique index prodno on saved.products(prodno);
00003 quit;
00004
```



```
Log - (Untitled)
Command ==>
233 create unique index prodno on saved.products(prodno);
NOTE: Simple index prodno has been defined.
234 quit;
```

Deletion

Tables can be deleted with the Drop statement.

```
Command ==>
00001 proc sql;
00002 drop table stock;
00003 drop view totwt;
00004 drop index ordno from saved.orders;
00005 quit;
```

Changing Data in Tables

Adding Rows

Adding rows to a table is done with the Insert statement.

Since a table has no concept of order there is no way to tell Proc SQL where to insert the data.

Review the previous example where a table outline was defined.

```
Command ==>
00001 proc sql;
00002 create table perform
00003 (prodno char(6) label='Product Number',
00004 diameter num informat=5.2 format=5.2,
00005 lit_min num informat=3. format=3.);
00006 insert into perform
00007 values('H10098',10.45,125);
00008 values('H10099',15.85,235);
00009 insert into perform
00010 set prodno = 'H20048',
00011 diameter=12.50,
00012 lit_min =180;
00013 quit;
```

Data can be inserted in two ways:

1. Using a Values clause where the values are listed in the order of the columns.
2. Using set column=value where order can be ignored.

```
Log - (Untitled)
Command ==>
212 create table perform
213 (prodno char(6) label='Product Number',
214 diameter num informat=5.2 format=5.2,
215 lit_min num informat=3. format=3.);
NOTE: Table WORK.PERFORM created, with 0 rows and 3 columns.
216 insert into perform
217 values('H10098',10.45,125)
218 values('H10099',15.85,235);
NOTE: 2 rows were inserted into WORK.PERFORM.
219 insert into perform
220 set prodno = 'H20048',
221 diameter=12.50,
222 lit_min =180;
NOTE: 1 row was inserted into WORK.PERFORM.
```

Deleting Rows

Rows are deleted with the Delete statement.

```
Command ==>
00001 proc sql;
00002 create table stock as
00003 select *
00004 from saved.stock;
00005 delete from stock
00006 where code like 'FX';
00007 quit;

Log - (Untitled)
Command ==>
228 delete from stock
229 where code like 'FX';
NOTE: 16 rows were deleted from WORK.STOCK.
```

Updating Data

In this example, the prices in the products table have been increased by 15% using the Update statement.

```
Command ==>
00001 proc sql;
00002 update saved.products
00003 set price = price*1.15;
00004 select prodno label = 'Product Number',
00005 proddesc label = 'Product Description',
00006 price label = 'Price' format=6.2
00007 from saved.products;
00008 quit;

Log - (Untitled)
Command ==>
4 update saved.products
5 set price = price*1.15;
NOTE: 18 rows were updated in SAVED.PRODUCTS.
6 select prodno label = 'Product Number',
7 proddesc label = 'Product Description',
8 price label = 'Price' format=6.2
9 from saved.products;
```

Product Number	Product Description	Price
F40019	TABLE	62.15
F40020	CHAIR	25.88
F40021	LOUNGER	36.24
F40022	UMBRELLA	33.65
H10098	15 METER DOMESTIC	29.84
H10099	30 METER DOMESTIC	56.99
H20048	15 METER COMMERCIAL	40.70
H20049	30 METER COMMERCIAL	73.29
H20054	45 METER COMMERCIAL	89.58
H20055	60 METER COMMERCIAL	124.88
H20056	30 METER HEAVY DUTY	149.32
H20057	60 METER HEAVY DUTY	268.79
T30023	BALL WHEEL BARROW	52.80
T30024	HOE	39.19
T30025	TROWEL	24.11
T30026	SPADE	57.29
T30027	FORK	51.26
T30028	RAKE	45.23

Conditional Updating

This can be done using a Case...When...Then...End expression, as illustrated below.

```
Command ==>
00001 proc copy in=saved out=work;
00002 select products;
00003 run;
00004 proc sql;
00005 update work.products
00006 set price = price *
00007 case substr(prodno,1,1)
00008 when 'F' then 1.12
00009 when 'H' then 1.08
00010 when 'T' then 1.20
00011 else 1.00
00012 end;
00013 select prodno label = 'Product Number',
00014 proddesc label = 'Product Description',
00015 price label = 'Price' format=6.2
00016 from work.products;
00017 quit;
```

Conditions that are more specific can be met, as displayed below.

```
Command ==>
00001 proc sql;
00002 update work.products
00003 set price = price *
00004 case
00005 when substr(prodno,1,1) = 'F'
00006 and price gt 15 then 1.15
00007 when substr(prodno,1,1) = 'F'
00008 and price le 15 then 1.10
00009 else 1.25
00010 end;
00011 select prodno label = 'Product Number',
00012 proddesc label = 'Product Description',
00013 price label = 'Price' format=6.2
00014 from work.products;
00015 quit;
```

Product Number	Product Description	Price
F40019	TABLE	27.59
F40020	CHAIR	10.99
F40021	LOUNGER	15.39
F40022	UMBRELLA	14.29
H10098	15 METER DOMESTIC	13.74
H10099	30 METER DOMESTIC	26.24
H20048	15 METER COMMERCIAL	18.74
H20049	30 METER COMMERCIAL	33.74
H20054	45 METER COMMERCIAL	41.24
H20055	60 METER COMMERCIAL	57.49
H20056	30 METER HEAVY DUTY	68.74
H20057	60 METER HEAVY DUTY	123.74
T30023	BALL WHEEL BARROW	21.88
T30024	HOE	16.24
T30025	TROWEL	9.99
T30026	SPADE	23.74
T30027	FORK	21.24
T30028	RAKE	18.74

Using Case in a Query

```

Command ==> |
00001 proc sql;
00002   select prodno label = 'Product Number',
00003          proddesc label = 'Product Description',
00004          case substr(prodno,1,1)
00005             when 'F' then 'Furniture'
00006             when 'H' then 'Hose'
00007             when 'T' then 'Tool'
00008             else ''
00009          end as prodtype label = 'Product Type'
00010   from saved.products;
00011 quit;

```

Product Number	Product Description	Product Type
F40019	TABLE	Furniture
F40020	CHAIR	Furniture
F40021	LOUNGER	Furniture
F40022	UMBRELLA	Furniture
H10098	15 METER DOMESTIC	Hose
H10099	30 METER DOMESTIC	Hose
H20048	15 METER COMMERCIAL	Hose
H20049	30 METER COMMERCIAL	Hose
H20054	45 METER COMMERCIAL	Hose
H20055	60 METER COMMERCIAL	Hose
H20056	30 METER HEAVY DUTY	Hose
H20057	60 METER HEAVY DUTY	Hose
T30023	BALL WHEEL BARROW	Tool
T30024	HOE	Tool
T30025	TROWEL	Tool
T30026	SPADE	Tool
T30027	FORK	Tool
T30028	RAKE	Tool

The Case expression is another way of doing the following in the Data step:

- If...Then...Else
- Select...When...Otherwise...End

Changing the Structure of Tables

The Alter statement can be used to change the attributes of the table's columns.

The following clauses can be used with this statement:

- **Modify:** Changes the attributes of the columns.
- **Add:** Adds columns to the table.
- **Drop:** Deletes columns from the table.

```

Command ==> |
00001 proc sql;
00002   create table perform
00003     (prodno char(6) label='Product Number',
00004      diameter num informat=5.2 format=5.2,
00005      lit_min num informat=3. format=3.);
00006   insert into perform
00007     values('H10098',10.45,125);
00008     values('H10099',15.85,235);
00009   insert into perform
00010     set prodno='H20048',
00011        diameter =12.50,
00012        lit_min =180;
00013   alter table perform
00014     modify lit_min num format=3.1 label='Litres/Minute',
00015            diameter num label='Nozzle Diameter',
00016            add gall_min num format=3.1 label='Gallons/Minute';
00017   update perform
00018     set gall_min=lit_min*.22;
00019   select * from perform;
00020 quit;

```

```

Log - (Untitled)
Command ==> |
78   alter table perform
79     modify lit_min num format=3.1 label='Litres/Minute',
80     diameter num label='Nozzle Diameter',
81     add gall_min num format=3.1 label='Gallons/Minute';
NOTE: Table WORK.PERFORM has been modified, with 4 columns.
82   update perform
83     set gall_min=lit_min*.22;
NOTE: 3 rows were updated in WORK.PERFORM.
84   select * from perform;

```

Product Number	Nozzle Diameter	Litres/Minute	Gallons/Minute
H10098	10.45	125	28
H10099	15.85	235	52
H20048	12.50	180	40

Syntax

- Alter table name
- Add column-definition, column-definition
- Modify column-definition, column-definition
- Drop column-name, column-name

Additional Topics

Changing Options

The following Options are available on the Proc SQL statement.

(Defaults are underlined>.

Errorstop / Noerrorstop

This only has an effect when Proc SQL is running in batch or non-interactive execution modes.

It determines whether the syntax of the rest of the code is checked once an error has been encountered.

Exec / Noexec

Noexec is useful in an interactive session if you want to check the full syntax without executing the code.

Compare this with the Validate statement which only validates the syntax of a Query-Expression.

Feedback / Nofeedback

This option is used to get an expanded log:

Select * becomes a full list of column names

Views are expanded into the underlying Query

Expressions are parenthesized to indicate order of evaluation.

Inobs=n

Restricts the number of rows processed and is used for debugging errors in queries on large tables.

Loops=n

Inefficiencies are detected using the SQLoops macro variable (later this section) and this value can be used to set a value for this option.

For example, it limits internal iterations to stop the production of huge internal tables during joins.

Number / Nonumber

This option determines whether a row number is shown in the query output.

Outobs=n

This option restricts the number of rows that a query expression can pass on: for example, inserting rows as a result of a query expression.

Print / Noprint

This determines whether the default Proc Print style output is produced from the query expression.

Sortseq=sort-table-name

This determines the collating sequence to be used by the Order By clause.

Stimer / Nostimer

This gives resource usage information for each statement. The corresponding System option must also be turned on.

Note that with the system option ON and the Proc SQL option OFF resource usage is reported for the whole procedure rather than for each statement.

Options can be reset during an SQL session or job with the Reset statement:

```
Command ==> |
00001 proc sql noprint;
00002     select.....
00003     from.....
00004     where.....
00005     ;
00006     reset print;
00007     select.....
00008     .
00009     .
00010     .
00011 quit;
```

Screen Control Language Interface

Proc SQL can be called from within SCL in SAS/AF (not SAS/FSP), using a Submit...Endsubmit block with the SQL option.

The Proc SQL statement is not required.

```
Command ==> |
00001 SOURCE NEW.PROGRAM (E
00002
00003 init:
00004     SCL statements
00005 return;
00006 main:
00007     SCL statements
00008     submit continue SQL;
00009     SQL statements
00010     endsubmit;
00011     SCL statements
00012 return;
```

Macro Language Interface

Macro variables can be created in the Select statement with the Into clause. They are available for use after its execution.

```
Command ==> |
00001 proc sql;
00002     select avg(price) into :mprice
00003     from saved.products;
00004 quit;
00005
00006 title "Average Price is &mprice";
00007 proc print data=saved.products;
00008     var price;
00009 run;
```

The output is displayed below.

Average Price is &mprice

25.40722

Average Price is 25.40722

Obs	PRICE
1	23.99
2	9.99
3	13.99
4	12.99
5	10.99
6	20.99
7	14.99
8	26.99
9	32.99
10	45.99
11	54.99
12	98.99
13	17.50
14	12.99
15	7.99
16	18.99
17	16.99
18	14.99

Several automatic macro variables are set: [R/O refers to Read only].

SQLOBS	R/O	The number of rows executed by a Proc SQL statement. E.g. number of rows SELECTed or number of rows DELETED.
SQLRC	R/O	Indicates the success of a Proc SQL statement. 0 Successful completion, no errors. 4 Warnings issued and execution continued. 8 Error encountered. Execution halted at this point. 12 Internal error encountered - a bug that should be reported to SAS Institute. 16 User error. The code tries to yield an impossible result.

SQLRC (continued)		24 System error, i.e., disk may be full. 28 Internal error, i.e., a bug that should be reported to SAS Institute
SQLLOOPS		Contains the number of iterations that the inner loop of Proc SQL executes. Increases with the complexity of the query.
SQLXRC		Contains DBMS-specific return code that is returned by the Pass-Through Facility
SQLXMSG		Contains descriptive information and DBMS-specific return code for the error that is returned by the Pass-Through Facility.

Efficiency Tips

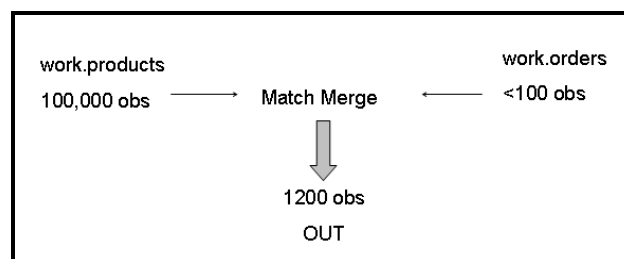
Comparing figures for conventional data/proc step coding with those for the same processing done in Proc SQL can produce surprising results.

See the following example.

```

Command ==>
00007 data out;
00008   merge work.products (in = a) work.orders (in =b);
00009   by prodno;
00010   if a and b;
00011 run;
00012
00013 proc sql noprint;
00014   create table out as
00015   select *
00016   from work.products as a,
00017        work.orders as b
00018   where a.prodno = b.prodno;
00019 quit;

```



	Proc SQL	Data Step
Indexed	1	18
Non-Indexed	40	13

The SQL query optimizer is extremely good at utilizing the benefits of an index, as shown here.

Using the data step, the index comes into play only at the time of subsetting.

The data step has to process all the observations in the master data set and uses the index at the subsetting stage only.

The ANSI Standard and Proc SQL: Enhancements

Although SQL is mainly used as an expressive, powerful query language within the main body of SAS concepts, many of the enhancements found in the ANSI standard are also available within the draft SQL2 Standard.

Reserved words

In the ANSI standard, all keywords are reserved. In Proc SQL a few keywords are reserved and then only in context.

- Case is always reserved. Give the variable an alias to get around this.
- The following cannot be used as table aliases in Proc SQL:

As, On, Full, Join, Left, From, When, Where, Order, Group, Right, Inner, Outer, Union, Except, Having and Intersect.
- The keyword User is reserved for the current Userid.

Column Modifiers

- Informat=
- Format=
- Label=

The above are all SAS Specific.

Collating Sequence

An alternate collating sequence can be used with the Sortseq option.

Order By

In Proc SQL this is supported in Create View. When the View is used the rows are always sorted in the order specified unless the Outer query has another Order By clause.

In-Line Views

This is provided in accordance with the draft SQL2 standard.

Outer Joins

This is provided in accordance with the draft SQL2 standard.

Arithmetic Operators

The following are enhancements:

- Exponentiation (**)
- Minimum (><)
- Maximum (<>)
- Sounds-like (=*)
- CONTAINS

Orthogonal Expressions

All types of Comparison, Boolean and Algebraic expressions are possible:

- If the condition is True it evaluates to a 1, if False to a 0.
- A Sub-query is allowed in any expression as required by SQL2.

Set Operators

Union, Intersect and Except are provided as per the SQL2 requirement.

Outer Union is also provided.

Statistical Functions

Many more are provided than any standard requires.

System Functions

All provided except Lag, Dif and Sound.

The ANSI Standard and Proc SQL: Omissions

Naming Conventions

The SAS limit of eight characters for table and column names is not an SQL standard restriction.

Rollback

No current provision.

Granting of User Privileges

No current provision.

System specific security is, of course, not violated and can be used as in the rest of the SAS System.

Missing Values

The SAS System uses a method of making missing i.e. Null values, the lowest possible number.

The ANSI SQL standard invokes special cases for dealing with Null values.

Any value compared to a Null value evaluates to Null in this standard.

Embedded SQL

The embedding of SQL in other languages such as Data Step or IML is not currently supported.

Updating Views

Proc SQL Views are currently read-only.

SAS/ACCESS Views can be referenced by Proc SQL to update their base tables or views.

Unique Constraint

The Unique keyword in Create Table in the standard ensures that only unique column values are created.

The same effect can be achieved by creating a unique index on the column or combination of columns.