# Combine Tables with PROC SQL

- You can join tables horizontally or vertically using PORC SQL.
- In Part one, we have covered combining tables horizontally.
  - Inner joins
  - Outer jions
    - Left outer join
    - Right outer join
    - Full outer join
- In this Part, we will emphasize on joining tables vertically

# Example 1 : combine tables horizontally

- You are given two tables, sasuser.staffmaster and sususer.payrollmaster, which contain all the data you need.
- Use an inner join to display the names (first initial and last name), job codes, and ages of all company employees who live in New York.
- Sort the results by job code and age.
- Include a title called “New York Employees.”
- The required form of name is not directly available from the tables; you need to produce a new column.
- The age values are also not directly available from the tables; only values of dataofbirth are listed. You need to use the following formula to calculate it: int((today() – dateofbirth)/365.25).

# The code and output

```
proc sql;
      title 'New York Employees';
      select substr(firstname,1,1) || '. ' || lastname
           as Name, jobcode,
           int((today() - dateofbirth)/365.25) as Age
      from sasuser.payrollmaster as p INNER JOIN
           sasuser.staffmaster as s
      ON p.empid = s.empid
           and state='NY'
      order by 2, 3;
quit;
```

Partial results:

New York Employees

| Name | JobCode | Age |
|---|---|---|
| R. LONG | BCK | 42 |
| T. BURNETTE | BCK | 46 |
| J. MARKS | BCK | 47 |
| N. JONES | BCK | 47 |
| R. VANDEUSEN | BCK | 53 |
| J. PEARSON | BCK | 54 |
| L. GORDON | BCK | 54 |
| C. PEARCE | FA1 | 41 |
| D. WOOD | FA1 | 42 |
| C. RICHARDS | FA1 | 44 |
| L. JONES | FA1 | 46 |
| R. MCDANIEL | FA1 | 46 |
| A. PARKER | FA1 | 49 |
| … | … | … |

# Example 2: combine tables horizontally

- You are given two tables, sasuser.**marchflights** and sususer.**flightdelays**.
- Use an outer join to display flight date, flight number, and length of delay in minutes (if any) for all March flights.
- Each flight is identified by both a flight date and a flight number.
- Sort your result by delay in minutes.
- Some of the relevant columns are listed below.

sasuser.marchflights

| Date | DepartureTime | FlightNumber | Origin | ... |
|---|---|---|---|---|
| 01MAR2000 | 8:21 | 182 | :21 | ... |
| 01MAR2000 | 7:10 | 114 | :10 | ... |
| 01MAR2000 | 10:43 | 202 | :43 | ... |
| ... | ... | ... | ... | ... |

sususer.flightdelays

| Date | FlightNumber | Origin | Destination | ... | Delay |
|---|---|---|---|---|---|
| 01MAR2000 | 182 | LGA | YYZ | ... | 0 |
| 01MAR2000 | 114 | LGA | LAX | ... | 8 |
| 01MAR2000 | 202 | LGA | ORD | ... | -5 |
| ... | ... | ... | ... | ... | ... |

# The code and output

```
proc sql;
   select m.date,
       m.Flightnumber label='Flight Number',
       delay label='Delay in Minutes'
   from sasuser.marchflights as m
          left join
          sasuser.flightdelays as f
   on m.date=f.date
        and
        m.flightnumber=f.flightnumber
   order by delay;
quit;
```

Partial results (left join):

| Date | Flight Number | Delay in Minutes |
|---|---|---|
| 14MAR2000 | 271 | . |
| 16MAR2000 | 622 | . |
| . | 132 | . |
| 22MAR2000 | 183 | . |
| 11MAR2000 | 290 | . |
| 27MAR2000 | 982 | . |
| 29MAR2000 | 829 | . |
| 11MAR2000 | 202 | . |
| 08MAR2000 | 182 | . |
| 17MAR2000 | 182 | . |
| 03MAR2000 | 416 | . |
| 25MAR2000 | 872 | . |
| 09MAR2000 | 821 | -10 |
| 25MAR2000 | 829 | -10 |
| 02MAR2000 | 387 | -10 |
| 10MAR2000 | 523 | -10 |
| 07MAR2000 | 523 | -10 |
| 18MAR2000 | 219 | -10 |
| 14MAR2000 | 829 | -10 |
| … | … | … |

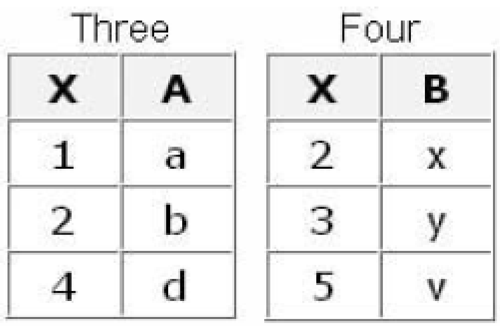
# The code and output

```
proc sql;
   select m.date,
       m.Flightnumber label=’Flight Number’,
       delay label=’Delay in Minutes’
   from sasuser.marchflights as m
          right join
          sasuser.flightdelays as f
   on m.date=f.date
        and
        m.flightnumber=f.flightnumber
   order by delay;
quit;
```

Partial results (right join):

| Date | Flight Number | Delay in Minutes |
|---|---|---|
| 09MAR2000 | 821 | -10 |
| 14MAR2000 | 829 | -10 |
| 02MAR2000 | 387 | -10 |
| 25MAR2000 | 829 | -10 |
| 18MAR2000 | 219 | -10 |
| 07MAR2000 | 523 | -10 |
| 10MAR2000 | 523 | -10 |
| 23MAR2000 | 982 | -9 |
| 02MAR2000 | 821 | -9 |
| 14MAR2000 | 387 | -9 |
| 27MAR2000 | 182 | -9 |
| 14MAR2000 | 308 | -9 |
| 19MAR2000 | 132 | -9 |
| 21MAR2000 | 183 | -9 |
| 13MAR2000 | 202 | -9 |
| 15MAR2000 | 219 | -9 |
| 28MAR2000 | 182 | -9 |
| 18MAR2000 | 182 | -9 |
| 15MAR2000 | 821 | -9 |
| … | … | … |

# Using the COALESCE function in outer join



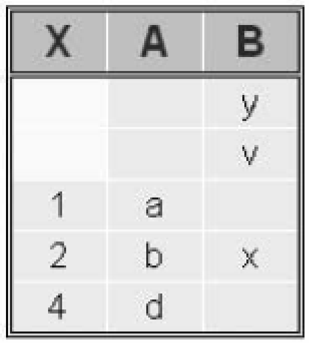
Three

| X | A |
|---|---|
| 1 | a |
| 2 | b |
| 4 | d |

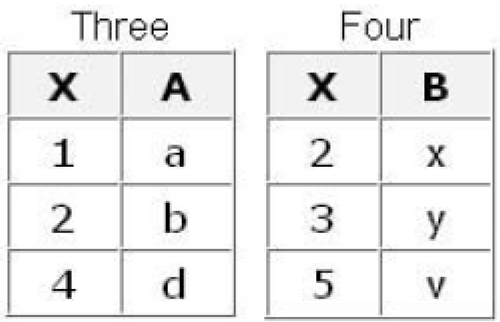
Four

| X | B |
|---|---|
| 2 | x |
| 3 | y |
| 5 | v |

```
proc sql;
  select  three.x,
    a, b
  from three
         full join
         four
  on three.x = four.x;
  order by x;
quit;
```



| X | A | B |
|---|---|---|
|  |  | y |
|  |  | v |
| 1 | a |  |
| 2 | b | x |
| 4 | d |  |



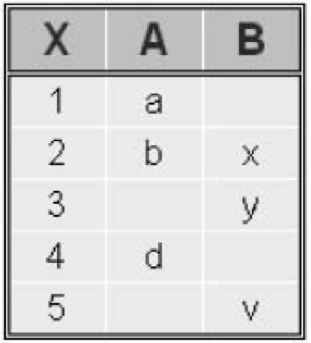
Three

| X | A |
|---|---|
| 1 | a |
| 2 | b |
| 4 | d |

Four

| X | B |
|---|---|
| 2 | x |
| 3 | y |
| 5 | v |

```
proc sql;
  select coalesce(three.x, four.x)
    as X,
    a, b
  from three
         full join
         four
  on three.x = four.x;
  order by x;
quit;
```



| X | A | B |
|---|---|---|
| 1 | a |  |
| 2 | b | x |
| 3 |  | y |
| 4 | d |  |
| 5 |  | v |

How does the COALESCE function work?

- Check the value of each column in the order in which the columns are listed
- Return the first value that is a SAS nonmissing value.
- If all returned values are missing, COALESCE returns a missing value.

# Joining Tables Vertically

- Set operations
- Using EXCEPT set operation
- Using INTERSECT set operation
- Using UNION set operation
- Using OUTER UNION set operation

# General form of set operation in PROC SQL

```
SELECT column-1<, ... column-n>
      FROM table-1 | view-1<, ... table-n | view-n>
      <optional query clauses>
set-operator <ALL> <CORR>
SELECT column-1<, ... column-n>
      FROM table-1 | view-1<, ... table-n | view-n>
      <optional query clauses>;
```

It contains only one semicolon.

where

SELECT specifies the column(s) that will appear in the result.

FROM specifies the table(s) or view(s) to be queried.

*Optional query clauses* are used to refine the query further and include the clauses WHERE, GROUP BY, HAVING, and ORDER BY.

*Set-operator* is one of the following: EXCEPT, INTERSECT, UNION, or OUTER UNION.

Optional keywords *ALL* and *CORR (CORRESPONDING)* further modify the set operation.

# A simple example

| ID | Name | RestHR | MaxHR | RecHR | TimeMin | TimeSec | Tolerance | Year |
|---|---|---|---|---|---|---|---|---|
| 2458 | Murray, W | 72 | 185 | 128 | 12 | 38 | D | 1998 |
| 2462 | Almers, C | 68 | 171 | 133 | 10 | 5 | I | 1998 |
| 2523 | Johnson, R | 69 | 162 | 114 | 9 | 42 | S | 1998 |
| … | … | … | … | … | … | … | … | … |

| ID | Name | RestHR | MaxHR | RecHR | TimeMin | TimeSec | Tolerance | Year |
|---|---|---|---|---|---|---|---|---|
| 2501 | Bonaventure, | 78 | 177 | 139 | 11 | 13 | I | 1999 |
| 2544 | Jones, M | 79 | 187 | 136 | 12 | 26 | N | 1999 |
| 2552 | Reberson, P | 69 | 158 | 139 | 15 | 41 | D | 1999 |
| … | … | … | … | … | … | … | … | … |

```
proc sql;
  select  *
  from sasuser.stress98
  union
  select  *
  from sasuser.stress99;
quit;
```

| ID | Name | RestHR | MaxHR | RecHR | TimeMin | TimeSec | Tolerance | Year |
|---|---|---|---|---|---|---|---|---|
| 2458 | Murray, W | 72 | 185 | 128 | 12 | 38 | D | 1998 |
| 2462 | Almers, C | 68 | 171 | 133 | 10 | 5 | I | 1998 |
| 2501 | Bonaventure, | 78 | 177 | 139 | 11 | 13 | I | 1999 |
| 2523 | Johnson, R | 69 | 162 | 114 | 9 | 42 | S | 1998 |
| 2539 | LaMance, K | 75 | 168 | 141 | 11 | 46 | D | 1998 |
| … | … | … | … | … | … | … | … | … |

# Precedence of set operators

- Multiple set operators can be used in a single SELECT statement (containing one semicolon).
- By default, INTERSECT is evaluated first; UNION, OUTER UNION and EXCEPT have the same level of precedence.
- Parentheses can be used to override the default precedence.

# Introducing set operators

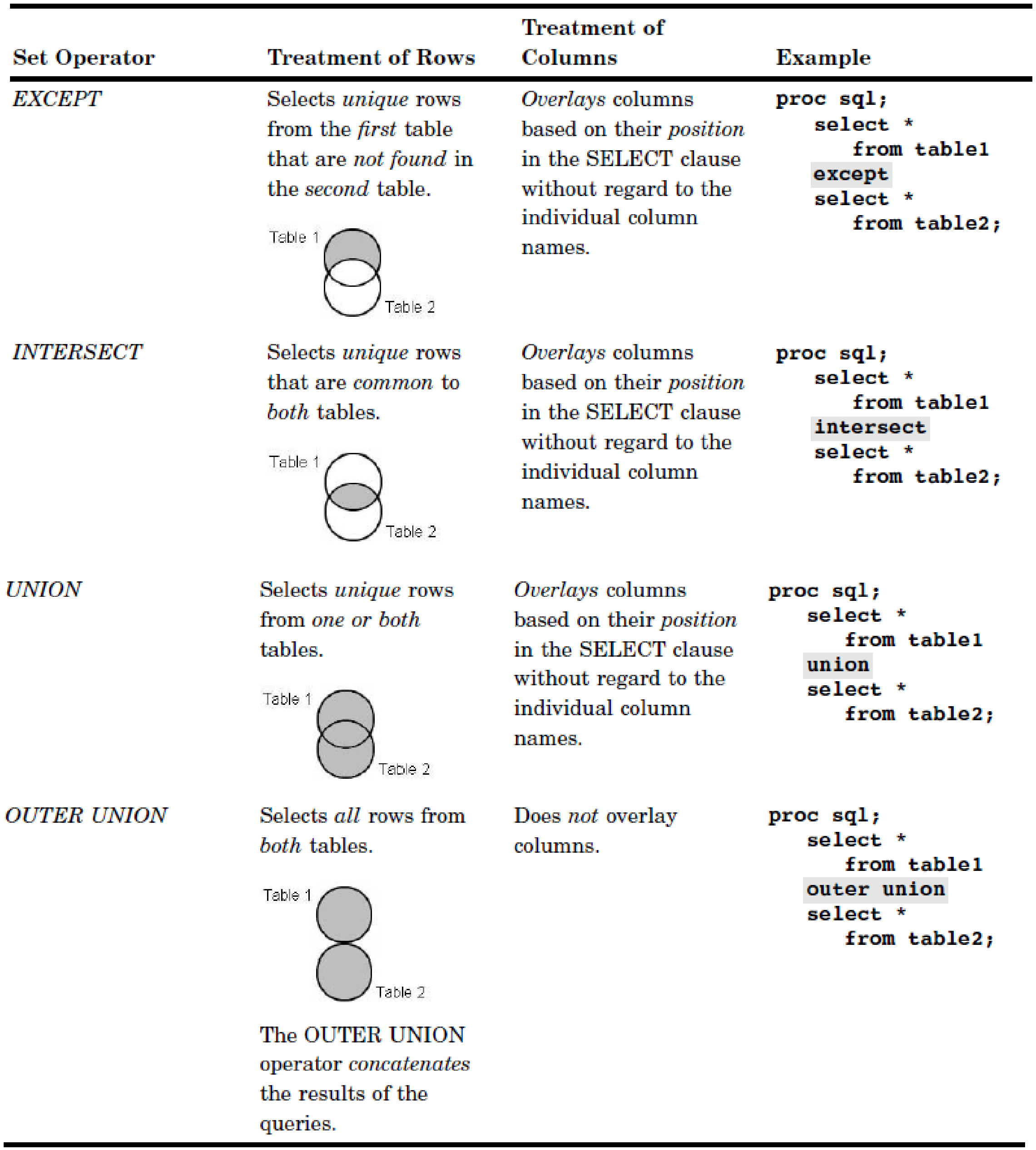


Table 1
Table 2
Table 1
Table 2
Table 1
Table 2
Table 1
Table 2

| Set Operator | Treatment of Rows | Treatment of Columns | Example |
|---|---|---|---|
| *EXCEPT* | Selects *unique* rows from the *first* table that are *not found* in the *second* table. | *Overlays* columns based on their *position* in the SELECT clause without regard to the individual column names. | `proc sql;`<br>`select *`<br>`from table1`<br>`except`<br>`select *`<br>`from table2;` |
| *INTERSECT* | Selects *unique* rows that are *common* to *both* tables. | *Overlays* columns based on their *position* in the SELECT clause without regard to the individual column names. | `proc sql;`<br>`select *`<br>`from table1`<br>`intersect`<br>`select *`<br>`from table2;` |
| *UNION* | Selects *unique* rows from *one or both* tables. | *Overlays* columns based on their *position* in the SELECT clause without regard to the individual column names. | `proc sql;`<br>`select *`<br>`from table1`<br>`union`<br>`select *`<br>`from table2;` |
| *OUTER UNION* | Selects *all* rows from *both* tables.<br>The OUTER UNION operator *concatenates* the results of the queries. | Does *not* overlay columns. | `proc sql;`<br>`select *`<br>`from table1`<br>`outer union`<br>`select *`<br>`from table2;` |

# Processing unique vs. duplicate rows

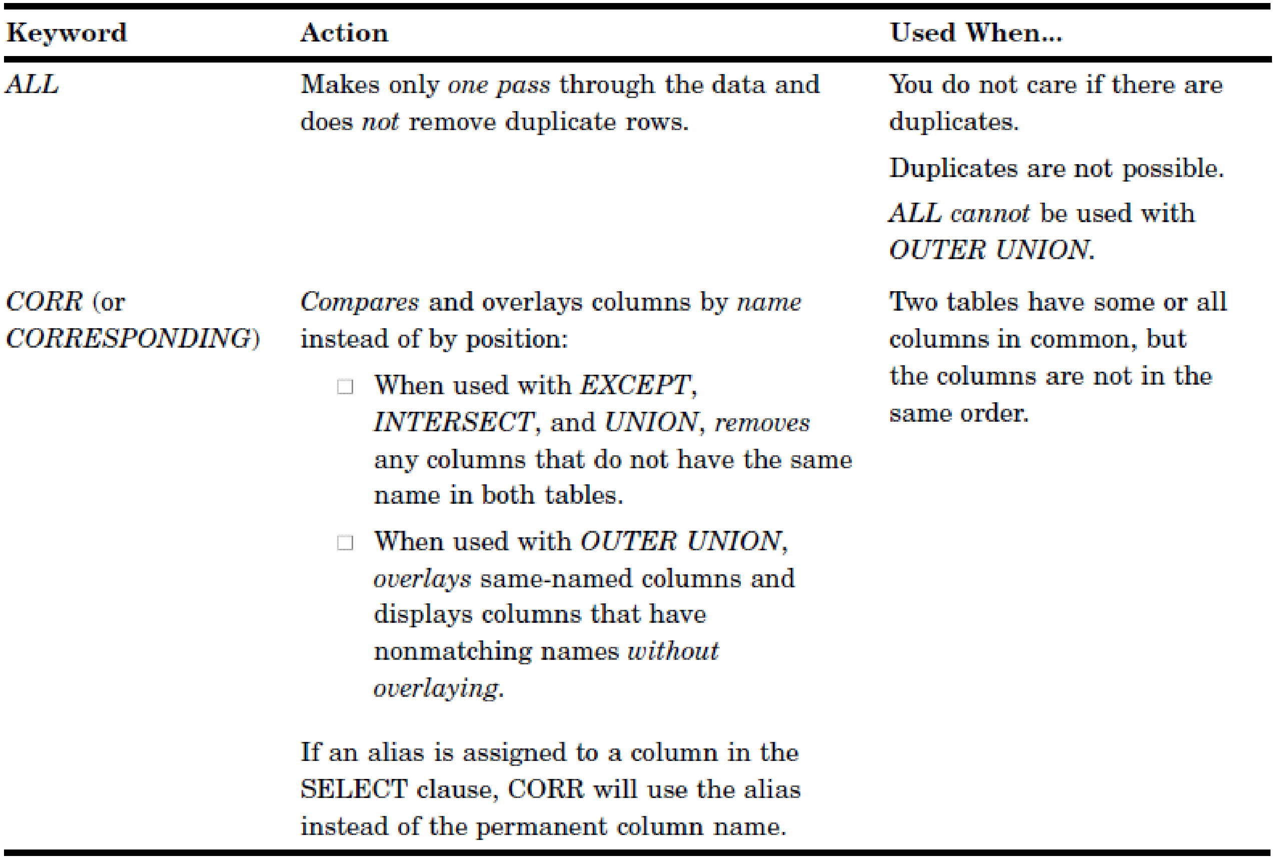
- Set operators that only display unique rows are EXCEPT, INTERSECT, and UNION. The processing makes two passes through the data:
  - Eliminates duplicate (nonunique) rows in the tables.
  - Selects the rows that meet the criteria and, where requested, overlays columns.
- The OUTER UNION displays both unique and duplicate rows, and makes only one pass through the data.

# Rules for combining and overlaying columns

- A set operation can be used to combine tables that have different numbers of columns and rows or that have columns in a different order.
- EXCEPT, INTERSECT, and UNION combine columns by overlaying them based on the relative *position* of the columns in the SELECT clause. Column names are ignored.
- You control how PROC SQL maps columns in one table to columns in another table by specifying the columns in the appropriate order in the SELECT clause. The first column specified in the first query's SELECT clause and the first column specified in the second query's SELECT clause are overlaid, and so on.
- When columns are overlaid, PROC SQL uses the column name from the first table (the table referenced in the first query).
- When the SELECT clause contains an asterisk (*****) instead of a list of column names, the set operation combines the tables (and, if applicable, overlays columns) based on the positions of the columns in the tables.
- In order to be overlaid, columns in the same relative position in the two SELECT clauses must have the *same data type*.

# Modifying results with ALL and/or CORR keywords

```
proc sql;
   select *
      from table1
   set-operator <all> <corr>
   select *
      from table2;
```



| Keyword | Action | Used When... |
|---|---|---|
| *ALL* | Makes only *one pass* through the data and does *not* remove duplicate rows. | You do not care if there are duplicates.<br>Duplicates are not possible.<br>*ALL cannot* be used with *OUTER UNION*. |
| *CORR* (or *CORRESPONDING*) | *Compares* and overlays columns by *name* instead of by position:<br>□ When used with *EXCEPT*, *INTERSECT*, and *UNION*, *removes* any columns that do not have the same name in both tables.<br>□ When used with *OUTER UNION*, *overlays* same-named columns and displays columns that have nonmatching names *without overlaying*.<br>If an alias is assigned to a column in the SELECT clause, CORR will use the alias instead of the permanent column name. | Two tables have some or all columns in common, but the columns are not in the same order. |

# Using the EXCEPT set operator

One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
   select *
     from one
   EXCEPT
   select *
     from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |

| X | A |
|---|---|
| 1 | a |
| 1 | b |
| 2 | c |
| 4 | e |
| 6 | g |

- Overlay columns: One.X and Two.X (both numeric), One.A and Two.B (both character); column names from table One are used.
- Eliminate duplicate rows.
- Select unique rows from table One that are not found in table Two.

# Using keyword ALL with the EXCEPT operator

One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
   select *
     from one
   EXCEPT ALL
   select *
     from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 4 | e |
| 6 | g |

- Overlay columns: One.X and Two.X (both numeric), One.A and Two.B (both character); column names from table One are used.
- Select all rows from table One that are not found in table Two.

# Using keyword CORR with the EXCEPT operator

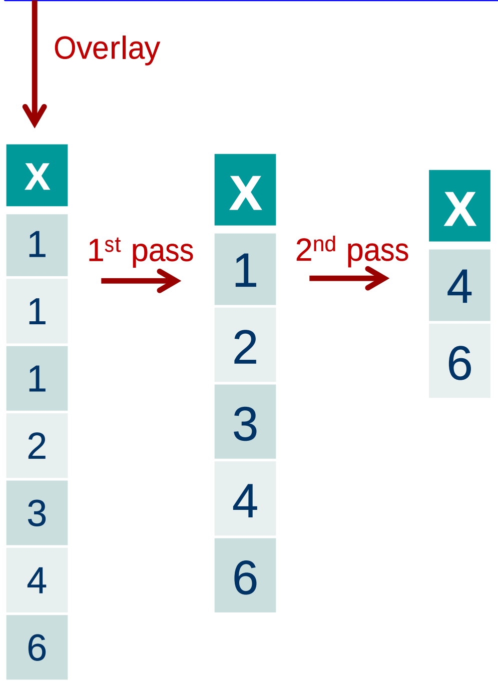
One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
    select *
      from one
    EXCEPT CORR
    select *
      from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |



| X |
|---|
| 1 |
| 1 |
| 1 |
| 2 |
| 3 |
| 4 |
| 6 |

| X |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 6 |

| X |
|---|
| 4 |
| 6 |

- Overlay columns by name: X is the only column that has the same name.
- First pass: eliminate the 2nd and 3rd rows.
- Second pass: Select all rows from table One that are not found in table Two.

# Using keywords ALL and CORR with the EXCEPT operator

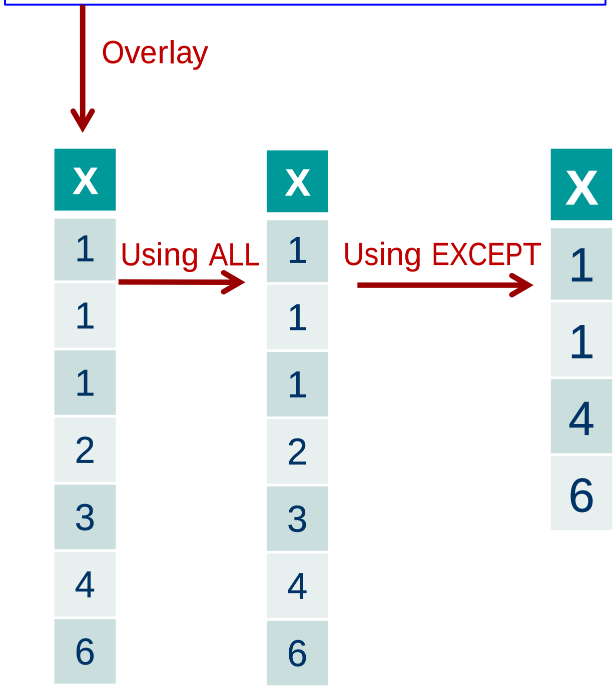
One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
   select *
     from one
   EXCEPT ALL CORR
   select *
     from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |



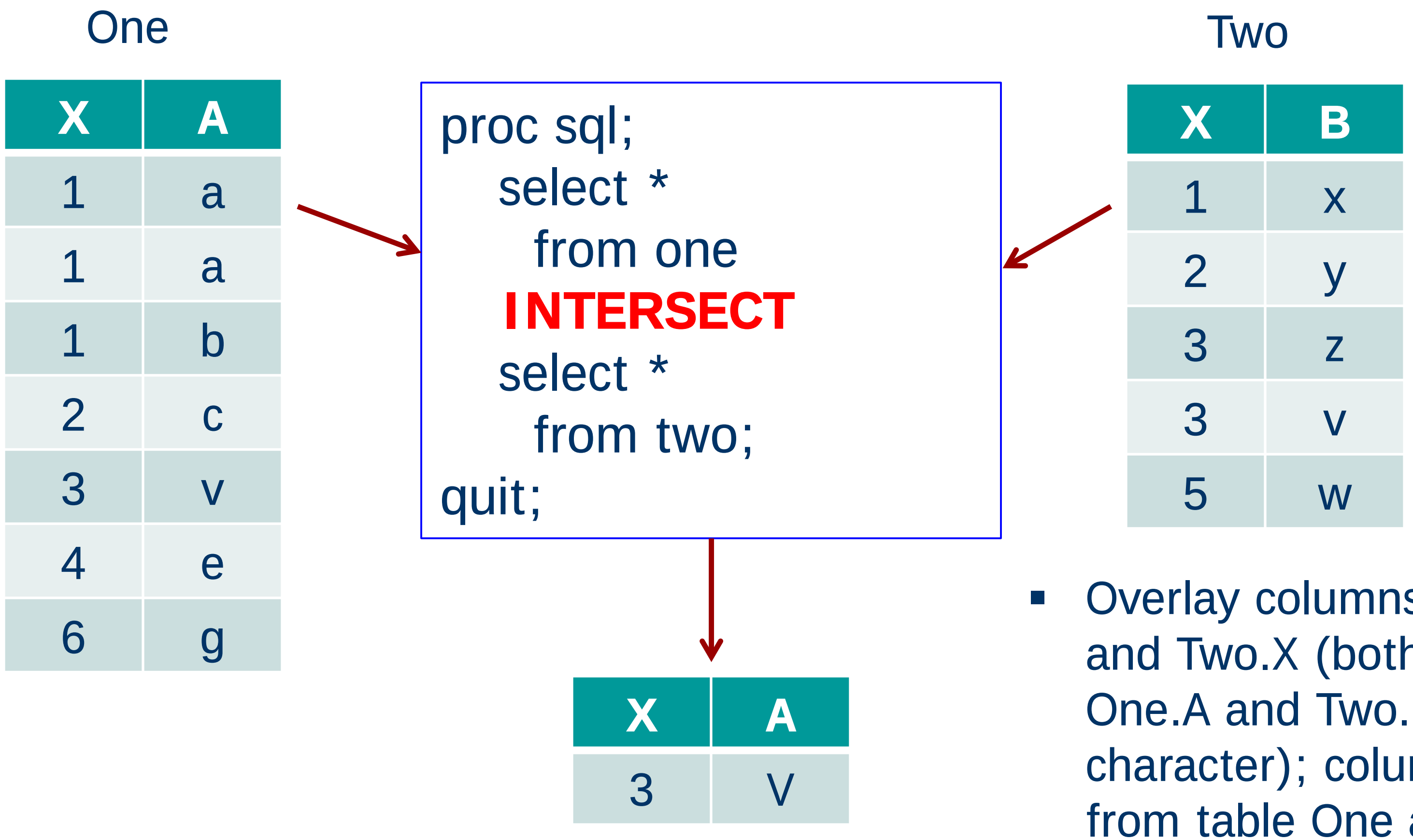
| X |
|---|
| 1 |
| 1 |
| 1 |
| 2 |
| 3 |
| 4 |
| 6 |

| X |
|---|
| 1 |
| 1 |
| 1 |
| 2 |
| 3 |
| 4 |
| 6 |

| X |
|---|
| 1 |
| 1 |
| 4 |
| 6 |

- Using CORR: Overlay columns by name; X is the only column that has the same name.
- Using ALL: It does not eliminate the 2nd and 3rd rows.
- Using EXCEPT: Eliminate rows from table One that are found in table Two.

# Using the INTERSECT operator



One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |

```
proc sql;
    select *
      from one
    INTERSECT
    select *
      from two;
quit;
```

| X | A |
|---|---|
| 3 | V |

- Overlay columns: One.X and Two.X (both numeric), One.A and Two.B (both character); column names from table One are used.
- Eliminate duplicate rows.
- Select unique rows that are common to both tables.

# Using the keyword ALL with the INTERSECT operator

One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
    select *
      from one
    INTERSECT ALL
    select *
      from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |

| X | A |
|---|---|
| 3 | V |

- Overlay columns: One.X and Two.X (both numeric), One.A and Two.B (both character); column names from table One are used.
- Skip checking and eliminating duplicates.
- Select unique rows that are common to both tables.

# Using the keyword CORR with the INTERSECT operator

One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
   select *
     from one
   INTERSECT CORR
   select *
     from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |

Overlay

| X | X |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 2 | 3 |
| 3 | 5 |
| 4 | |
| 6 | |

1st pass

| X | X |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 5 |
| 6 | |

2nd pass

| X |
|---|
| 1 |
| 2 |
| 3 |

- Overlay columns by name: X is the only column that has the same name.
- First pass: eliminate the 2nd and 3rd rows in table One and the 4th row in table Two.
- Second pass: Select all rows that are common to the two tables.

# Using the keywords ALL and CORR with the INTERSECT operator

Displays all unique and nonunique rows that are common to the two tables

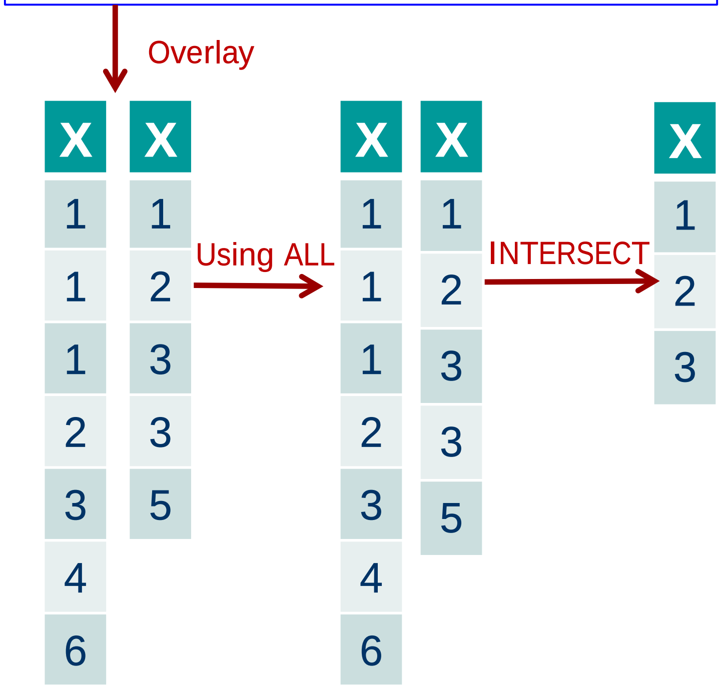
One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
    select *
      from one
    INTERSECT ALL CORR
    select *
      from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |



Overlay

| X | X |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 2 | 3 |
| 3 | 5 |
| 4 | |
| 6 | |

Using ALL

| X | X |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 2 | 3 |
| 3 | 5 |
| 4 | |
| 6 | |

INTERSECT

| X |
|---|
| 1 |
| 2 |
| 3 |

- Overlay columns by name: X is the only column that has the same name.
- Using ALL: do not eliminate duplicates.
- Using INTERSECT: Select all rows that are common to the two tables.

# Using the UNION operator

One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |

```
proc sql;
   select *
     from one
   UNION
   select *
     from two;
quit;
```

| X | A |
|---|---|
| 1 | a |
| 1 | b |
| 1 | x |
| 2 | c |
| 2 | y |
| 3 | v |
| 3 | z |
| 4 | e |
| 5 | w |
| 6 | g |

- Overlay columns: One.X and Two.X (both numeric), One.A and Two.B (both character); column names from table One are used.
- The two tables are concatenated.
- Eliminate duplicate rows in both tables: 2nd row in table One and 4th row in table Two which matches the 5th row in table One.

# Using the ALL keyword with the UNION operator

One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
    select *
      from one
    UNION ALL
    select *
      from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | V |
| 5 | w |

Note the difference in the order of the rows.

- Overlay columns: One.X and Two.X (both numeric), One.A and Two.B (both character); column names from table One are used.
- The two tables are concatenated.
- No duplicates are removed due to the use of ALL.

# Using the CORR keyword with the UNION operator

One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
   select *
     from one
   UNION CORR
   select *
     from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |

| X |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

- Overlay columns by name: X is the only column that has the same name.
- Eliminate the duplicates: the values of 1, 2, and 3 are duplicated.

# Using the ALL and CORR keywords with the UNION operator

One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
   select *
     from one
   UNION ALL CORR
   select *
     from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |

| x |
|---|
| 1 |
| 1 |
| 1 |
| 2 |
| 3 |
| 4 |
| 6 |
| 1 |
| 2 |
| 3 |
| 3 |
| 5 |

- Overlay columns by name: X is the only column that has the same name.
- Keep all the rows that are both unique and duplicate.

# Using the OUTER UNION operator

One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
    select *
      from one
    OUTER UNION
    select *
      from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |

| X | A | X | B |
|---|---|---|---|
| 1 | a | . | |
| 1 | a | . | |
| 1 | b | . | |
| 2 | c | . | |
| 3 | v | . | |
| 4 | e | . | |
| 6 | g | . | |
| . | | 1 | x |
| . | | 2 | y |
| . | | 3 | z |
| . | | 3 | v |
| . | | 5 | w |

- Select all the rows (both unique and nonunique) from both tables.
- Concatenate the results of queries but not overlay the columns. As a result, you end up with a table of two columns with the same name.

# Using the CORR keyword with the OUTER UNION operator

One

| X | A |
|---|---|
| 1 | a |
| 1 | a |
| 1 | b |
| 2 | c |
| 3 | v |
| 4 | e |
| 6 | g |

```
proc sql;
   select *
     from one
   OUTER UNION CORR
   select *
     from two;
quit;
```

Two

| X | B |
|---|---|
| 1 | x |
| 2 | y |
| 3 | z |
| 3 | v |
| 5 | w |

| X | A | B |
|---|---|---|
| 1 | a | |
| 1 | a | |
| 1 | b | |
| 2 | c | |
| 3 | v | |
| 4 | e | |
| 6 | g | |
| 1 | | x |
| 2 | | y |
| 3 | | z |
| 3 | | v |
| 5 | | w |

- With the CORR keyword, you overlay the columns with the same name X.
- Select all the rows (both unique and nonunique) from both tables.
- Concatenate the results of queries.

# Practice: Using Set Operators

- Create a report that displays the employee identification number of current Level III and Level IV sales staff hired in 2004, who made at least one sale by the end of 2005. The sasuser.**Order_fact** table contains information on all sales, and the sasuser.**Sales** table contains information about current sales employees, including job titles and hire dates.
- Think about how you can use three columns to display the employee numbers (i.e., employee IDs), job codes, and salaries of all mechanics from Levels 1 to 3 respectively, keeping the same row order as the original tables. The mechanic job has three levels and there is a separate table containing data for the mechanics at each level: *Sasuser.Mechanicslevel1*, *Sasuser.Mechanicslevel2*, and*Sasuser.Mechanicslevel3*. These tables all contain the same three columns. Write your PROC SQL code to realize it.
- Code the following business situation: You want to display vertically the following summarized data for members of a frequent-flyer program: total points earned, total points used, and total miles traveled. All three values can be calculated from columns in the table *Sasuser.Frequentflyers* by using summary functions. Also display these results horizontally.

# Solution of the first bullet

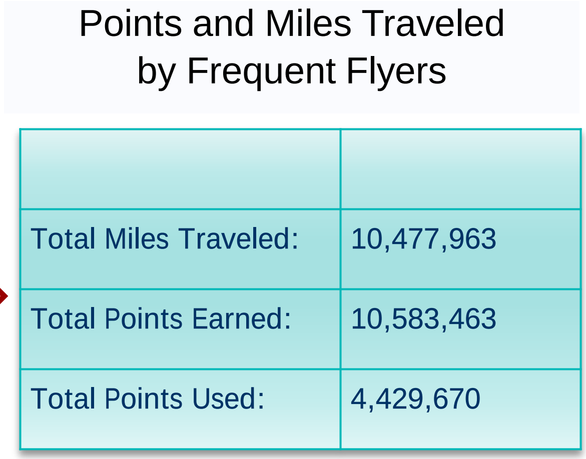
| Employee ID |
|---|
| 120179 |

# Solution of the second bullet

| EmpID | JobCode | Salary |
|---|---|---|
| 1400 | ME1 | $41,677 |
| 1403 | ME1 | $39,301 |
| 1120 | ME1 | $40,067 |
| 1121 | ME1 | $40,757 |
| 1412 | ME1 | $38,919 |
| 1200 | ME1 | $38,942 |
| 1995 | ME1 | $40,334 |
| 1418 | ME1 | $39,207 |
| 1653 | ME2 | $49,151 |
| 1782 | ME2 | $49,483 |
| 1244 | ME2 | $51,695 |
| 1065 | ME2 | $49,126 |
| 1129 | ME2 | $48,901 |
| 1406 | ME2 | $49,259 |
| 1356 | ME2 | $51,617 |
| 1292 | ME2 | $51,367 |
| 1440 | ME2 | $50,060 |
| 1900 | ME2 | $49,147 |
| 1423 | ME2 | $50,082 |
| 1432 | ME2 | $49,458 |
| 1050 | ME2 | $49,234 |
| 1105 | ME2 | $48,727 |
| 1499 | ME3 | $60,235 |
| 1409 | ME3 | $58,171 |
| 1379 | ME3 | $59,170 |
| 1521 | ME3 | $58,136 |
| 1385 | ME3 | $61,460 |
| 1420 | ME3 | $60,299 |
| 1882 | ME3 | $58,153 |

# Solution of the third bullet, part 1

Display vertically



Points and Miles Traveled
by Frequent Flyers

| | |
|---|---|
| Total Miles Traveled: | 10,477,963 |
| Total Points Earned: | 10,583,463 |
| Total Points Used: | 4,429,670 |

# Solution of the third bullet, part 2

Display horizontally

| Total Points Earned | Total Points Used | Total Miles Traveled |
|---|---|---|
| 10,583,463 | 4,429,670 | 10,477,963 |