Paper TT13

CFB: A Programming Pattern for Creating “Change from Baseline” Datasets
Lei Zhang, Celgene Corporation, Summit, NJ

ABSTRACT
In many clinical studies, “Change from Baseline” analysis is frequently used to evaluate the effect of medical intervention by comparing the measures of change from baseline in some clinical characteristic between treatment groups. One of common and recurring challenges in this analysis is the creation of “Change from Baseline” datasets, which is often made complicated by protocol designs, statistical methods, sponsor SOPs, and “real world” data issues that arise during the trial. In this paper, I introduce a programming pattern called CFB (shortened for Change from Baseline) that provides a general solution to creating various “Change from Baseline” datasets. This programming pattern is an extension to the Delta pattern described in [1]. I start the CFB pattern with a theoretical CFB data model and the motivation for using this pattern, and then use UML diagram to present a generic CBF programming structure, which can be conveniently implemented with SQL statements in PROC SQL. In addition, I discuss the variants and other uses of the CFB programming pattern and explain the applicability, trade-off, and consequences of the solution. Throughout this paper, the CFB programming pattern is illustrated with the examples from clinical data analysis.

INTRODUCTION AND MOTIVATION
Creating “Changes from Baseline” datasets for various efficacy and safety endpoints are common and recurring tasks in clinical data analysis. The CDISC ADaM team, therefore, proposed a “Change from Baseline” analysis data model for clinical researchers to estimate the treatment effect by comparing the measurement changes from baseline in some clinical characteristic between treatment groups [2]. In the context of a clinical study, the baseline value is typically a measure of a clinical characteristic before the medical intervention while the treatment or post baseline value is the corresponding measure at a defined time point or period after the intervention.

For a clinical characteristic, given a statistical object \( i \) in a population, the measurement change from baseline can be formulated as follows:

\[
c_{TB} = f_1(S_T) - f_2(S_B)
\]

where

- \( c_{TB} \): The measurement change in the clinical characteristic from the baseline period \( B \) to a treatment period \( T \) for the statistical object \( i \).
- \( S_B \): A set of observations related to the clinical characteristic that are collected from the statistical object \( i \) during the baseline period \( B \).
- \( S_T \): A set of observations related to the clinical characteristic that are collected from the statistical object \( i \) during the treatment period \( T \).
- \( f_1, f_2 \): The two functions that are applied to the sets \( S_B \) and \( S_T \). They can be aggregate functions, such as mean, sum, max, count, or user-defined functions like the first or last non-missing value of two, or more values. Usually those functions produce a single numeric value from sets \( S_B \) and \( S_T \), respectively.

In clinical data analysis, variables related to patient ID, and/or treatment group are used to indicate individual statistical objects, and time variables such as patient visit number, visit date, or treatment phase is used to organize or derive the changes. It is important to note that there are many issues that can arise during the creation of “Change from Baseline” analysis datasets. For example, there may be more than one baseline and treatment measurements in some studies. The definition of baseline or treatment values may vary. They can be a single value, an average, or a summary of multiple measures at a time point or period. Before we go further, let’s take a look at some CFB dataset problems in the context of a clinical study.

PROBLEM
Suppose we have a double-blind, placebo-controlled, randomized parallel study that analyzes the different effects of drug A, drug B, and placebo on the growth of pediatric patients. The study has three parallel arms and two periods. Period I is a 2-visit placebo run-in period; Period II is a 3-visit double-blind treatment period. At baseline visit 2, eligible patients are randomly allocated to one of three treatment groups: drug A (TRT=1), drug B (TRT=2), or placebo (TRT=3). Both height (in cm) and weight (in kg) of a patient will be measured and recorded at each visit, either scheduled or unscheduled. All the height and weight data are collected in a dataset called Growth, which has following structure per patient and visit:
Variables *patient* and *visit* are the composite key to this dataset. The information collected from each visit of a pediatric patient corresponds to an observation in the Growth dataset. A 5-patient sample dataset is listed at the end of this paper.

With the Growth dataset, we may have following tasks that derive the study endpoints for analysis or graphical presentation:

1. For each patient, calculate the change in mean height from the baseline period to the treatment period.

2. For each patient, calculate the change from the mean height in the baseline period to the last non-missing height in the treatment period.

3. For each patient, calculate the change between the mean BMI (Mass Body Index) in the baseline period and the BMI at each treatment visit period. Where BMI = 10000 x Weight in kg / (Height in cm) x (Height in cm).

4. For each treatment group, calculate the paired difference of mean height at each visit.

5. For each patient, calculate the time interval or during between visits in the treatment period.

6. For each patient, calculate the change from baseline, where the baseline value is the mean of last two non-missing values in the baseline period, and treatment value is the last non-missing value in the treatment period.

The traditional SAS approach to those tasks is to use DATA/PROC steps. Here are two examples:

The DATA/PROC step solution to task 1:

```sas
proc sort data=Growth;
  by patient visit;
run;

/* Compute the baseline mean height */
proc means data=Growth(where = (1 <= visit <= 2)) noprint;
  by patient;
  var height;
  output out=H1(drop=_TYPE_ _FREQ_) mean=T1Height;
run;

/* Compute the treatment mean height */
proc means data=Growth(where = (3 <= visit <= 5)) noprint;
  by patient;
  var height;
  output out=H2(drop=_TYPE_ _FREQ_) mean=T2Height;
run;

/* Compute the change from baseline */
data CFB;
  merge H1 H2;
  by patient;

  meanchg = T2Height - T1Height;
run;
```

*Listing 1. A DATA/PROC Step Solution to Task 1*

The DATA/PROC step solution to task 2:

```sas
proc sort data=Growth;
  by patient visit;
```
/* Compute the baseline mean height */
proc means data=Growth(where = (1 <= visit <=2)) noprint;
   by patient;
   var height;
   output out=H1(drop=_TYPE_ _FREQ_) mean=T1Height;
run;

/* Compute the last non-missing height in the treatment period */
Data H2;
   set Growth(where = (3 <= visit <=5));
   by patient visit;
   if last.patient;
       T2Height=height;
   keep patient T2Height;
run;

/* Compute the change from baseline */
data CFB;
   merge H1 H2;
   by patient;
   lastChg = T2Height - T1Height;
run;

Listing 2. A DATA/PROC Step Solution to Task 2

A careful examination of the code in Listing 1 and 2 reveals following shortcomings of the traditional solution:

- Different CFB tasks require the different number of DATA/PROC steps. It is hard to identify the code boundary of a solution.
- Similar CFB tasks may require very different implementations, meaning the slight modification of a CFB requirement may lead to the significant code modification.
- Besides, the code is sensitive to the change in dataset structure. For example, if the height are measured twice at each visit and recorded as two separate variables in an observation, or patients have unscheduled visit observations, more DATA steps may have to be added.

CFB PROGRAMMING PATTERN

In this section, I give a programming pattern called CFB for the creation of “Change from Baseline” datasets. This is inspired by design patterns in the object-oriented programming communities [3]. The idea is to provide a general and systematic way to reuse common programming expertise for various CFB dataset problems. The general solution structure shown in Figure 1 is described with UML (Unified Model Language [3]). In the UML diagram, both query and subquery are treated as objects similar to those in an object-oriented language. The pattern can be implemented with SQL statements in PROC SQL.
As you will see from Figure 1, the CFB programming pattern consists of three participants: a main SQL query, two separate simple subqueries for baseline and treatment data. The baseline and treatment subqueries work out the $f_1(S_B)$ and $f_2(S_T)$ for individual statistical objects, and the main SQL query combines their outputs and then gets the required change, $c_{TB}$. When needed, WHERE clause in the main query can be used to subset the result.

The essential part of this programming pattern is the subqueries. There are three key factors to be considered when constructing the subqueries:

- Conditions (Where/Having clauses) that shape datasets $S_B$ and $S_T$.
- Functions $f_1$ and $f_2$ that map $S_B$ and $S_T$ to a single numeric value respectively.
- Variables that are used to join two subqueries.

Under the CFB pattern structure, tweaking the conditions and functions $f_1$ and $f_2$ in the subqueries often gives you the right answers to various CFB dataset problems.

Below is the CFB solution to task 1:

```sql
PROC SQL;
/* Compute the change from baseline */
create table CFB as
select BL.patient, BL.height as BLHeight, T1.height as T1Height,
T1.height - BL.height as meanChg
from
/* Compute the baseline data*/
(select patient, mean(height) as height
from Growth
where visit >=1 and visit <=2
group by patient ) BL full join
```
/* Compute the treatment data */
(select patient, mean(height) as height
 from Growth
 where visit >=3 and visit <=5
 group by patient ) T1

 on BL.patient = T1.patient
;
quit;

Listing 3. A CFB Solution to Task 1

The CFB solution to task 2 is

PROC SQL;

/* Compute the change from baseline */
Create Table CFB as
select BL.patient, BL.height as BLHeight, T1.height as T1Height,
 T1.height - BL.height as lastChg
from
/* Compute the baseline data */
(select patient, mean(height) as height
 from Growth
 where visit >=1 and visit <=2
 group by patient ) BL

Full join

/* Compute the treatment data */
(select patient, height
 from Growth
 where visit >=3 and visit <=5
 group by patient
 having visit= max(visit)) as T1

 on BL.patient = T1.patient
;
quit;

Listing 4. A CFB Solution to Task 2

The solutions based on the CFB pattern can be further simplified or made more efficient under certain circumstances. For example, if you want to exclude patients who were absent from all the treatment visits, you can replace “full join” with “left join” or “right join”. Listing 5 gives a solution to task 3, in which the patients who don’t have any observations in the treatment period are excluded.

PROC SQL;

Create Table CFB as
select BL.patient, T1.visit, BL.BMI as BLBMI,
 T1.BMI as T1BMI, T1.BMI - BL.BMI as BMIChg
from
/* Compute the baseline data */
(select patient, mean(10000*weight/(height*height)) as bmi
 from Growth
 where visit >=1 and visit <=2
 group by patient ) BL

Listing 5. A CFB Solution to Task 3
Right join
/* Compute the treatment data */
(select patient, visit, 10000*weight/(height*height) as bmi
from Growth
where visit >=3 and visit <=5) T1

on BL.patient = T1.patient
;
quit;

Listing 5. A CFB Solution to Task 3

GENERALIZED CFB PROGRAMMING PATTERN
The CFB programming pattern can be extended to create more than one CFB variables in a dataset by simply adding more
subqueries.

Listing 6 is the code that creates two CFB variables in one dataset for both task 1 and task 2. All the code does is to append
another subquery to the code in Listing 3, and add the CFB variable lastChg in the main query.

PROC SQL;
Create Table CFB as
/* Compute the change from baseline from both task 1 and 2 */
select BL.patient, BL.height as BLHeight,
T1.height as T1Height,
T1.height - BL.height as meanChg,
T2.height as T2Height,
T2.height - BL.height as lastChg
From
/* Compute the baseline data */
(select patient, mean(height) as height
from Growth
where visit >=1 and visit <=2
group by patient ) BL
left join
/* Compute the treatment data f ro task 1 */
(select patient, mean(height) as height
from Growth
where visit >=3 and visit <=5
group by patient ) T1

on BL.patient = T1.patient
Left Join
/* Compute the treatment data fro task 2*/
(select patient, height
from Growth
where visit >=3 and visit <=5
group by patient
having visit= max(visit)) as T2
on BL.patient = T2.patient
;
Quit;

Listing 6. A CFB Solution to Both Task 1 and Task 2

OTHER USES
The CFB programming pattern can be used to solve other difference calculation problems. Listing 7 is the solution to the
problem 4, which calculates the paired differences in mean height between three treatments groups for each visit period. The
function \texttt{int()} in the code is used to handle the values collected at unscheduled visits.

```sql
PROC SQL;
cREATE TABLE CFB AS
    SELECT T0.visit,
           T1.height AS T1Height,
           T2.height AS T2Height,
           T3.height AS T3Height,
           (T1.height - T3.height) AS delta1,
           (T2.height - T3.height) AS delta2,
           (T1.height - T2.height) AS delta3
    FROM (SELECT DISTINCT \texttt{int}(visit) AS visit FROM Growth) T0
         LEFT JOIN /* Compute the mean height for treatment group 1 */
             (SELECT \texttt{int}(visit) AS visit, mean(height) AS height FROM Growth
                 WHERE trt = 1
                 GROUP BY calculated visit) T1
             ON T0.visit = T1.visit
         LEFT JOIN /* Compute the mean height for treatment group 2 */
             (SELECT \texttt{int}(visit) AS visit, mean(height) AS height FROM Growth
                 WHERE trt = 2
                 GROUP BY calculated visit) T2
             ON T0.visit = T2.visit
         LEFT JOIN /* Compute the mean height for treatment group 3 */
             (SELECT \texttt{int}(visit) AS visit, mean(height) AS height FROM Growth
                 WHERE trt = 3
                 GROUP BY calculated visit) T3
             ON T0.visit = T3.visit
;
QUIT;
```

Listing 7. A CFB Solution to Task 4

Another interesting example is to compute the time interval or during between two visits for each patient as requested in task 4.

```sql
PROC SQL;
CREATE TABLE CFB AS
    SELECT coalesce(T1.Patient, T2.Patient) AS Patient,
           T1.visit AS ID,
           (T1.visdate - T2.visdate) AS interval
    FROM /* Get the visit dates for visit 3, 4 and 5 */
        (SELECT Patient, visit, visdate
         FROM Growth
         WHERE visit IN (3, 4, 5)) T1
    FULL JOIN /* Get the visit dates for visit 2, 3 and 4 */
        (SELECT Patient, visit +1 AS visit, visdate
         FROM Growth
         WHERE visit IN (2, 3, 4)) T2
```

7
on T1.patient=T2.Patient and T1.visit = T2.visit
    where not missing(T1.visit)
;
quit;

Listing 8. A CFB Solution to Task 5

VARIANTS OF CFB PROGRAMMING PATTERN
The CFB programming pattern can be implemented with other styles of SQL statements. Listing 9 is the alternative solution to task 4 that uses correlated SQL subqueries to calculate the paired differences of mean height between treatment groups for each visit period. For more information about the correlated SQL query, please see [4].

PROC SQL;
    Create Table CFB as
    select T0.visit as visit,
        /* Compute mean height for treatment group 1 per visit */
        (select mean(height)
            from Growth T1
            where trt = 1 and
                T0.visit= int(T1.visit)) as T1Height,
        /* Compute mean height for treatment group 2 per visit */
        (select mean(height)
            from Growth T2
            where trt = 2 and
                T0.visit= int(T2.visit)) as T2Height,
        /* Compute mean height for treatment group 3 per visit */
        (select mean(height)
            from Growth T3
            where trt = 3 and
                T0.visit= int(T3.visit)) as T3Height,
        (calculated T1Height - calculated T3Height) as delta1,
        (calculated T2Height - calculated T3Height) as delta2,
        (calculated T1Height - calculated T2Height) as delta3
    from ( select distinct int(visit) as visit from Growth) T0
;
quit;

Listing 9. Another SQL Solution to Task 4

This solution uses three correlated sub-queries as expressions in the main SQL SELECT statement. It calculates the paired differences for each visit period one by one. The solution will run very fast if the number of observations in subquery T0 is small, but may suffer from performance issues when the number of observations in the T0 is large. For example, suppose there are $n$ observations in T0, the SQL solution will have to execute $3n$ subqueries before giving the final result.

CFB PROGRAMMING PATTERN AND SQL VIEW
In case two subqueries in the CFB pattern become too complicate to be understood when they are included in the main query, they can be defined as two separate SQL views, and then use the views in the main SQL query. Listing 10 is the solution to task 6 with SQL views. It defines two SQL views - one for baseline data and the other for treatment data - and then joins them to form the main query.

PROC SQL;
    /* Baseline SQL view: mean of last two non-missing values */
    Create View Baseline as
        Select T1.Patient, Mean(T1.height, T2.height) as height,
            sum(T1.visit, T2.visit) as SumVisit
        from Growth(where=(visit>=1 and visit <=2)) T1
            left join
                Growth(where=(visit>=1 and visit <=2)) T2

7

8
on T1.patient = T2.patient and T1.Visit < T2.Visit
group by T1.patient
having calculated SumVisit = max(calculated SumVisit)

/* Treatment SQL view: last non-missing value */
Create View Treatment as
    select patient, height
    from Growth
    where visit >= 3 and visit <= 5
    group by patient
    having visit = max(visit)

/* Compute change from baseline */
Create Table CFB as
    Select BL.Patient, BL.Height as BLHeight, T1.Height as T1Height,
    (T1.Height - BL.Height) as Change
    From Baseline as Bl
    Left Join
    Treatment as T1
    on BL.Patient = T1.Patient

Quit;

Listing 10. Another CFB SQL Solution to Task 6

CFB PROGRAMMING PATTERN AND MACRO
SAS Macro language can be used to simplify the implementation of a CFB solution. For example, we can define the following macro called %subquery as a template of the subqueries used in Listing 7.

%Macro subquery(trt=);
    select int(visit) as visit, mean(height) as height from Growth
    where trt = &trt
    group by calculated visit
%Mend subquery;

The solution to task 4 then can be rewritten as follows:

PROC SQL;
    Create table CFB as
        select T0.visit,
            T1.height as T1Height,
            T2.height as T2Height,
            T3.height as T3Height,
            (T1.height - T3.height) as delta1,
            (T2.height - T3.height) as delta2,
            (T1.height - T2.height) as delta3
        from (select distinct int(visit) as visit from Growth) T0
        left join
            (%subquery(trt=1)) T1
        on T0.visit = T1.visit
        left join
            (%subquery(trt=2)) T2
        on T0.visit = T2.visit
        left join
            (%subquery(trt=3)) T3
        on T0.visit = T3.visit

;
SAS Macro also provides support for generating the code for a CFB solution. For example, Listing 12 defines a simple macro to change for all the tasks similar to task 1, in which, the macro parameters BL_DSN and TRT_DSN are the names of Baseline and Treatment datasets respectively, PatID indicates the patient ID, VAR indicates analysis variable for the change, and OutDSN is the output dataset name. The macro takes care of the mundane aspects of the implementation of any type of task 1 so that a SAS programmer can define it once and apply it many times.

```sas
%Macro change(BL_DSN=, TRT_DSN=, PatID=, Var=, OutDSN=);
  PROC SQL;
  create Table &OutDSN as
    select coalesce(BL.&PatID, T1.&PatID) as &PatID,
      BL.&Var as BL&Var, T1.&Var as T1&Var,
      T1.&Var - BL.&Var as meanChg
    from (select &PatID, mean(&Var) as &Var
      from &BL_DSN
      group by &PatID ) BL
    full join
      (select &PatID, mean(&Var) as &Var
       from &TRT_DSN
       group by &PatID ) T1
    on T1.&PatID = BL.&PatID
  ;
  quit;
%Mend;
```

Listing 12. A Macro Implementation for Task 1

For example, if you want to calculate the change in mean weight from baseline to treatment period, you can call the macro as follows:

```sas
%Change(BL_DSN=Growth(where=(visit >=1 and Visit <=2)),
         TRT_DSN=Growth(where=(visit >=3 and Visit <=5)),
         PATID=patient,Var=Weight,
         OutDsn=CFB)
```

BENEFITS AND DRAWBACKS IN USING CFB PATTERN

There are several benefits in using CFB programming pattern:

- **Cleaner code and reduction in code size** – CFB programming pattern usually provides one-step solution to creating various “Change from Baseline” datasets. The structured code gives a powerful natural language-like implementation.

- **Ease of code change and reuse** - the standard CFB pattern has a constant solution structure that can be used over and over again for various CFB tasks. Besides, within the main query, the two subqueries can be modified or changed independently.

- **Improve code extensibility** – The code following the CFB pattern can be easily extended to create multiple CFB variables in one dataset for different CFB endpoints, or be quickly tailored for the variants of the CFB tasks.

- **Facilitate the communication between SAS programmers** – The CFB programming pattern provides a vocabulary for the design of CFB programs, which will help SAS programmers to write, communicate and document the programs better.

There are some drawbacks in using CFB programming pattern:

- It requires SAS programmers to know how to use SQL statements, and how to write subqueries and join them properly to form a main query.

- If the two subqueries are combined with main query in one SQL statement, the interim baseline and treatment datasets will not be available for data review or verification purpose. However, you may define baseline and treatment subqueries with SQL views and then create the main query from them; you therefore can have an opportunity to check the interim values from the subqueries.
KNOWN USES
The CFB Pattern has following known uses in clinical data analysis.
- Compute the changes from baseline;
- Compute the paired differences
- Compute the time intervals.
- Compute drug compliance rates
- Compute other complex percentages and ratios

RELATED PATTERN
Delta Pattern [1]

Conclusion
The CFB programming Pattern introduced in this paper describes a generic and flexible solution to the creation of various “Change from Baseline” datasets. It attempts to capture the expertise in semi-formal way, and allows clinical SAS programmers to easily reuse high-level program designs of the proven solutions while avoid having to solve common CFB dataset problems each time they occur.

DISCLAIMER: The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Celgene Corporation.

REFERENCE
2. www.cdisc.org, “Guideline for the Creation of Analysis Files and Documentation of Statistical Analyses for Submission to the FDA”.
3. E. Gamma, R. Helm, R.Johnson, and J.Vlissides (1995). Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley Publishing Company, Reading, MA.

APPENDIX
Sample SAS dataset Growth:

```sas
data Growth;
  input patient 1-3 trt 5 visit 7-9
  height 11-16 weight 18-21 visdatec $ 23-28;
  visdate=input(visdatec, ddmmyy.);
  format visdate date9.;
  drop visdatec;
  datalines;
100 1 1 123.45 40.2 010201
100 1 1.1 123.01 40.5 090401
100 1 1.2 123.49 41.4 090401
100 1 2 . . 090401
100 1 3 124.17 40.9 180801
100 1 4 123.60 41.1 211201
100 1 5 124.10 41.5 270402
200 2 1 122.45 38.7 030100
200 2 2 122.49 38.1 040400
200 2 3 123.12 38.4 020800
200 2 4 122.60 38.1 011200
200 2 5 122.15 38.2 030401
300 3 1 125.45 42.1 070201
300 3 2 125.49 42.0 050601
300 3 3 125.23 42.5 061101
300 3 3.1 125.23 43.5 061101
```

11
<table>
<thead>
<tr>
<th>Size</th>
<th>Row</th>
<th>Feedback</th>
<th>Mean</th>
<th>SD</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>3</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>030502</td>
</tr>
<tr>
<td>300</td>
<td>5</td>
<td>126.10</td>
<td>44.9</td>
<td>080902</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>1</td>
<td>125.45</td>
<td>36.1</td>
<td>020101</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>2</td>
<td>125.49</td>
<td>36.5</td>
<td>030501</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>3</td>
<td>125.00</td>
<td>36.1</td>
<td>010901</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>1</td>
<td>127.45</td>
<td>36.1</td>
<td>150100</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>2</td>
<td>127.49</td>
<td>37.2</td>
<td>150500</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>1</td>
<td>125.41</td>
<td>35.1</td>
<td>161000</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>1.1</td>
<td>124.98</td>
<td>35.5</td>
<td>211000</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>2</td>
<td>124.29</td>
<td>35.3</td>
<td>140201</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>3</td>
<td>127.17</td>
<td>36.0</td>
<td>150601</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>4</td>
<td>127.60</td>
<td>36.9</td>
<td>141001</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>5</td>
<td>128.10</td>
<td>36.3</td>
<td>160202</td>
<td></td>
</tr>
</tbody>
</table>

; run;