コードレビュー Vol.6

CODE Review

USP 研究所技術研究員
written by 大内智明

データは、処理内容に応じた様々な形（レイアウト）を有します。同じデータであっても、人が操作するPC画面で見るデータレイアウトと、サーバーの夜間処理で使用するデータレイアウトは異なります。本稿ではサーバーが夜間に処理・作成したデータ（LV3）を、PC画面で操作しやすい形（LV4）に加工する処理について紹介します。

データの流れとレイアウト

今回紹介するコードは、LV3 から LV4 を作成する「LV4MAKE」の処理となります（図 1 で赤く示した部分）。図 1 に書かれたファイルの種類 LV1、LV2、LV3、LV4 は、ユニケージ開発手法で使用されるデータのレベルに応じた名称です（他に LV5 があります）。

● LV1

システムに入力された「生データ」（原始データ）。アプリケーションから入力されるデータや、外部システムからインターフェース（IF）されるデータです。

＊日中の更新は、アプリケーションが行います。

● LV2

LV1 をユニケージ上で使いやすいように変換したデータ

＊サーバーの夜間処理で作成します。

● LV3

LV2 を業務上使いやすい形に整理したデータ。店別、商品別などに種類分けされています。

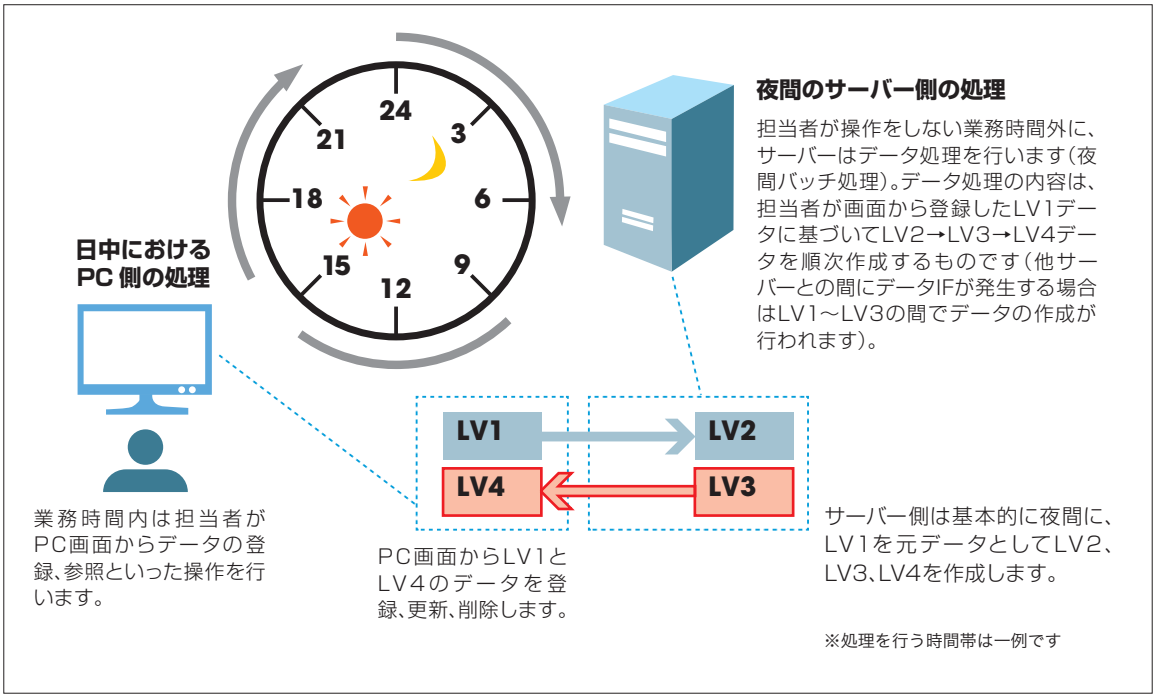
＊サーバーの夜間処理で作成します。

● LV4

業務アプリケーションが登録・参照・検索などを行うのに最も適した形式のデータ。

＊朝 1 回目は夜間処理で作成します。日中の更新は、アプリケーションが行います。

図 1　処理およびデータフロー



技術的な概要

●履歴データと有効情報の取得

サーバーの夜間処理で作成したLV3データは、必要な項目で構成されています（→補足1）。LV3には、業務アプリケーションで使用するマスタ項目が含まれていない場合もあるため、LV4を作成する際に必要な項目を付与します。使用するマスタ項目は、最新情報を使用する場合と、「○月×日」時点で有効なデータ（履歴管理しているデータから抽出した有効な情報）を使用する場合があります。ここでは、履歴管理しているデータから有効データを抽出する方法について説明します。

ユニケージには、履歴管理しているデータにレイアウト上のルールがあります。また、履歴管理されているデータから有効データを抽出するコマンドも準備されています。履歴管理されているデータと有効データを抽出するコマンドを組み合わせることで、履歴データを容易に取扱うことができます。

【補足1】 例えば、LV3データには店舗コードが含まれていても、店舗名称は含まれていません。なぜなら、サーバーの夜間処理を実行するのに、店舗コードを使用しても店舗名称は使用しないからです。店舗名称を必要とする主体は、LV4データを参照する業務アプリケーションになります。PC画面を操作する人には店舗コードは分かりにくく誤操作の一因となるため、代わりに店舗名称を用います。

画面1　dataファイルのレイアウトおよび有効データの抽出

```
$ cat data
00001 100 1 20101231 suzuki    20101231120134
00001 110 4 20130821 satou     20130821182103 ❶
00002 60  1 20100131 suzuki    20100131142109
00002 65  4 20110930 suzuki    20110930085921
00002 70  4 20120401 kobayasi 20120401104759 ❷
$ upmaster_valid --date 20120101 num=1 data  ❸
00001 100 1 20101231 suzuki 20101231120134
00002 65 4 20110930 suzuki 20110930085921
```

履歴管理しているデータ（dataファイル）のレイアウトは次の通りです。
1: 商品コード 2: 原価 3: 更新フラグ 4: 適用日 5: 入力者 6: 入力日時
※更新フラグは 1: 新規（有効）4: 修正（1でも可）7: 予約削除 8: 削除（論理削除）

❶ 商品00001の原価は100円から110円へ、2013年8月21日に変更になっています。

❷ 2012年1月1日時点で登録されていないため、無効なデータです。

❸ 有効データを抽出するコマンドupmaster_validは、--date <date>で指定した日付時点において有効なマスタレコードを出力します。このケースでは、2012年1月1日時点で有効なデータを抽出します。複数の有効データ（3、4行目）が存在する場合、最新のデータ（4行目）のみを出力します。

画面2　cjoin2コマンド

```
$ cat master
0000003 杉山_____ 26 F
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
$ cat tran
0000005 82 79 16 21 80
0000001 46 39 8  5  21
0000004 58 71 20 10 6
0000009 60 89 33 18 6
0000003 30 50 71 36 30
0000007 50 2  33 15 62
$ cjoin2 +@ key=1 master tran
0000005 崎村_____ 50 F 82 79 16 21 80
0000001 @ @ @ 46 39 8  5  21
0000004 @ @ @ 58 71 20 10 6
0000009 @ @ @ 60 89 33 18 6
0000003 杉山_____ 26 F 30 50 71 36 30
0000007 梶川_____ 42 F 50 2  33 15 62
```

- mstファイルはキーでソートされていること
- trnファイルはキーでソートされていなくてもよい
- +<string>はmstキーと一致しないtrnキーに対して<string>文字を補完する

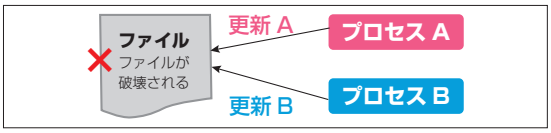
cjoin2コマンドは、mstファイルの情報をtrnファイルに結合します。
cjoin2 [+<string>] key=<n> <mst> <tran>

●データ書き込みと排他制御

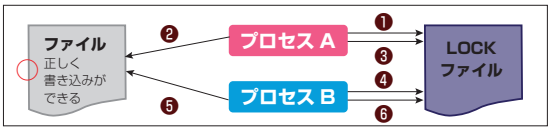
複数のプロセスが、同一ファイルにアクセスしてデータ更新処理を行う場合、同一ファイルに同時に複数の書き込みが発生してファイルが破壊されることがあります。それを防ぐため、複数プロセスが同一ファイルに対して正しく書き込みを実行するように制御（排他制御）を行います。

図2　排他制御

同一ファイルを複数のプロセスが書き込むとファイルが破壊される



排他制御を行うことにより、同一ファイルに複数のプロセスが書き込みを行っても、ファイルが破壊されることはない。



❶プロセスAがLOCKファイルを作成
❷プロセスAがファイルに書き込み処理
❸プロセスAがLOCKファイル削除
❹プロセスBがLOCKファイルを作成
❺プロセスBがファイルに書き込み処理
❻プロセスBがLOCKファイル削除

LOCKファイルを作成することで排他区間を取得します（❶と❸のLOCKファイルが存在している区間）。ファイル書き込み処理は排他区間内で行われます❷。LOCKファイルが削除されると、排他区間を終了します❸。プロセスAが排他区間を取得している間は、プロセスBは書き込み処理を実行しようとしても許されません。すなわち排他区間が終了するまで待ち状態となります。

リスト１　LV3 から LV4 を作成する処理

```
 1 #!/bin/ush -xve
 2 # システム名     :USPシステム
 3 # プログラム名   :LV4作成
 4 # 概要          :LV3からLV4作成
 5 # #
 6 # 備考(Usage)   : LV4MAKE.DAY.HAIKI_DATA [処理日付] [店舗コード]
 7 # シェル名       : LV4MAKE.DAY.HAIKI_DATA
 8 # 作成日         :2013/09/27
 9 # 会社名         :USP
10 # 作成者         :T.Ouchi
11
12 #////////////////////////////////////////////////////////////////////////////
13 # エラー時の終了処理定義（ushエラーハンドラ）
14 #////////////////////////////////////////////////////////////////////////////
15 err ERROR_EXIT(){
16   touch ${semd}/$(basename $0).${hostname}.ERROR.${gday}_${tenpocode}
17   echo "${hostname} $(basename $0)_${sday} ERROR $(date +%Y%m%d%H%M%S)" >> ${logd}/UPCNT
18
19   # ロックファイルの削除
20   rm -f ${lockfile}
21   exit 1
22 }
23
24 # 更新時の排他エラー
25 ZERO_EXIT() {
26   touch ${semd}/$(basename $0).${hostname}.END.${gday}_${tenpocode}
27   echo "${hostname} $(basename $0)_${sday} END $(date +%Y%m%d%H%M%S)" >> ${logd}/UPCNT
28   rm -f ${tmp}-* > /dev/null  2>&1
29   exit 0
30 }
    ＜中略＞
75 #////////////////////////////////////////////////////////////////////////////
76 # データ処理部
77 #////////////////////////////////////////////////////////////////////////////
78 # 更新対象のデータがないと終了
79 [ ! -s ${lv3d}/DENPYOU/${tenpocode}/DENPYO_DATA_HAIKI/TBL/HAIKI_DATA ] && ZERO_EXIT
80 #
81 #  /home/DATA/LV3/DENPYOU/${tenpocode}/DENPYO_DATA_HAIKI/TBL/HAIKI_DATA
82 #  1:事業会社コード      2:店舗コード        3:データ作成日付     4:データ作成時刻
83 #  5:SEQ                 6:廃棄日            7:商品/中分類コード  8:廃棄理由
84 #  9:数量                10:原単価           11:売単価            12:税区分
85 #  （中略）
86 #  NF-3:更新フラグ       NF-2:更新日         NF-1:ユーザID        NF:入力日時
87
88
89
90 # 廃棄理由区分
91 selr 1 HAIKI_RIYU ${lv3d}/M_SONOTA/M_KUBUN/TBL/M_KUBUN |
92 self 2 3 > ${tmp}-HAIKI_RIYU
93
94 # 税区分
95 selr 1 ZEIKUBUN ${lv3d}/M_SONOTA/M_KUBUN/TBL/M_KUBUN |
96 self 2 3 > ${tmp}-ZEIKUBUN
97
98 # 振替日ごとに履歴TBLから各必要項目を取得
99 cat ${lv3d}/DENPYOU/${tenpocode}/DENPYO_DATA_HAIKI/TBL/HAIKI_DATA |
100 lineup 6 |
101 cat      > ${tmp}-DenpBi
102
```

画面３　selr コマンド

```
$ cat data
0001 a
0002 b
0003 c
$ selr 1 "0001" < data
0001 a
```

selr コマンドは、フィールドが完全一致するレコードを抽出します。

```
103 for DenpBi in $(cat ${tmp}-DenpBi)
104 do
105   ## 各履歴TBL作成
106   #  1:中分類コード 2:中分類名
107   upmaster_valid --date ${DenpBi} num=1 ${lv3d}/M_SONOTA/M_HD_KAIKEI_BNR/TBL/CHUBNRCD_CHUBNRNM.RIREKI |
108   self 1 2 > ${tmp}-CHUBNRCD_CHUBNRNM.${DenpBi}
109
110   # 1:商品コード 2:商品名
111   upmaster_valid --date ${DenpBi} num=1 ${lv3d}/M_SHOHN/M_KIHONSHOHN/TBL/SHOHNCD_SHOHNNM.M000.RIREKI |
112   self 1 2 > $tmp-SHOHNCD_SHOHNNM.${DenpBi}
113
114   # 1:商品コード or 中分類コード 2:商品名 or 中分類名
115   up3 key=1 ${tmp}-CHUBNRCD_CHUBNRNM.${DenpBi} $tmp-SHOHNCD_SHOHNNM.${DenpBi} |
116   cat > ${tmp}-SHOHN_CHUBNR_CD_NM.${DenpBi}
117
118   # 1:店舗CD 2:事業会社コード
119   upmaster_valid --date ${DenpBi} num=1 ${lv3d}/M_TENPO/M_TENPO/TBL/TENPOCD_JGYOK.RIREKI |
120   self 1 2 > $tmp-TENPOCD_JGYOK.${DenpBi}
121
122   # 1:店舗CD 2:店舗名称
123   upmaster_valid --date ${DenpBi} num=1 ${lv3d}/M_TENPO/M_TENPO/TBL/TENPOCD_TENPOTAN.RIREKI |
124   self 1 2 > $tmp-TENPOCD_TENPOTAN.${DenpBi}
125
126   # 1:商品CD 2:商品名称
127   upmaster_valid --date ${DenpBi} num=1 ${lv3d}/M_SHOHN/M_KIHONSHOHN/TBL/SHOHNCD_SHOHNNM.M000.RIREKI |
128   self 1 2 > $tmp-SHOHNCD_SHOHNNM.${DenpBi}
129
130   # 1:事業会社CD 2:事業会社名称
131   upmaster_valid --date ${DenpBi} num=1 ${lv3d}/M_SONOTA/M_JGYK/TBL/JGYKCD_JGYKRYAKU.RIREKI |
132   self 1 2 > $tmp-JGYKCD_JGYKRYAKU.${DenpBi}
133
134
135   # 該当する伝票日付で抽出
136   zcat ${lv3d}/DENPYOU/${tenpocode}/DENPYO_DATA_HAIKI/RIREKI/${gday}.gz          |
137   upmaster_valid --date ${gday} num=5                                           |
138   selr 6 ${DenpBi}                                                             |
139   #  /home/DATA/LV3/DENPYOU/(店舗コード)/DENPYO_DATA_HAIKI/TBL/HAIKI_DATA
140   #  1:事業会社コード      2:店舗コード        3:データ作成日付    4:データ作成時刻
141   #  5:SEQ                 6:廃棄日            7:商品/中分類コード 8:廃棄理由
142   #  9:数量                10:原単価           11:売単価           12:税区分
143   #  (中略)
144   #  NF-3:更新フラグ       NF-2:更新日         NF-1:ユーザID       NF:入力日時
145   #
146   # 税区分名
147   cjoin2 +_ key=12 ${tmp}-ZEIKUBUN -                                           |
148   # 廃棄理由区分名
149   cjoin2 +_ key=8 ${tmp}-HAIKI_RIYU -                                          |
150   # 商品/中分類名
151   cjoin2 +_ key=7 ${tmp}-SHOHN_CHUBNR_CD_NM.${DenpBi} -                        |
152   # 店舗名
153   cjoin2 +_ key=2 $tmp-TENPOCD_TENPOTAN.${DenpBi} -                            |
154   # 事業会社名
155   cjoin2 +_ key=1 $tmp-JGYKCD_JGYKRYAKU.${DenpBi} -                            |
156   # 1:事業会社コード    2:事業会社名        3:店舗コード        4:店舗名            5:データ作成日付
157   # 6:データ作成時刻    7:SEQ               8:廃棄日            9:商品/中分類コード 10:商品/中分類名
158   # 11:廃棄理由         12:廃棄理由区分名   13:数量           14:原単価           15:売単価
159   # 16:税区分           17:税区分名
160   #  (中略)
161   #  NF-3:更新フラグ       NF-2:更新日         NF-1:ユーザID       NF:入力日時
```

upmaster_valid コマンドは、履歴データから有効情報を抽出する（画面 I を参照）

[1]「LV3 データ（コードのみで構成）に付与するコード+名称のテーブル」を作成する処理（91 ～ 132 行目）

cjoin2 コマンドは、mst ファイルの情報を trn ファイルに結合する（画面 2 を参照）

[2] LV3 データに [1] で作成したテーブルを結合して、名称付き LV4 データを作成する（136 ～ 164 行目）

```
162   maezero 7.5          |
163   cat > ${tmp}-lv4.${DenpBi}
164 done
165
166 # 伝票日ごとに作成した更新ファイルをまとめる
167 cat ${tmp}-lv4.*     |
168 msort key=1@3@5@6@7  |
169 cat > $tmp-lv4newdata
170
171
172 # ディレクトリ作成
173 mkdir -p ${lv4d}/${tenpocode}/DENPYO_DATA_HAIKI
174 lockfile=${lv4d}/${tenpocode}/DENPYO_DATA_HAIKI/DENPYO_DATA_HAIKI.LOCK
175 # 排他ロック
176 if ulock ${lockfile} ; then
177   #########
178   # V4更新
179   #########
180   mv $tmp-lv4newdata ${lv4d}/${tenpocode}/DENPYO_DATA_HAIKI/DENPYO_DATA_HAIKI
181
182   # ロックファイルの削除
183   rm -f ${lockfile}
184 fi
185
186 #////////////////////////////////////////////////////////////////////////////
187 # 終了
188 #////////////////////////////////////////////////////////////////////////////
189 # 終了時刻の保存
190 echo "${hostname} $(basename $0)_${sday} END $(date +%Y%m%d%H%M%S)" >> ${logd}/UPCNT
191 touch ${semd}/$(basename $0).${hostname}.END.${gday}_${tenpocode}
192 rm -f ${tmp}-* > /dev/null  2>&1
193 echo "$(basename $0) exit 0"
194 exit 0
```

msort コマンドは sort コマンドを高速化したユニケージコマンド

ulock コマンドは、排他制御を行う（図 2 を参照）

[3] [2] で作成した LV4 データに排他制御を行った上で更新処理をかける（173 ～ 184 行目）

コードの見どころ

コードリスト 1 を 3 つの観点から説明します。

[1]「LV3 データ（コードのみで構成）に付与するコード＋名称のテーブル」を作成する処理です。履歴形式の場合には、upmaster_valid コマンドで有効データの抽出を行い、テーブル作成を行います（91 ～ 132 行目）。

[2] LV3 データに **[1]** で作成したテーブルを結合して、名称付き LV4 データを作成します。結合用のコマンドに cjoin2 を使用しています（→補足 2）。リストの 136 ～ 164 行目にあたります。

[3] [2] で作成した LV4 データに排他制御を行った上で更新処理をかけます。排他処理を行っているコードは 173 ～ 184 行目です。

【補足 2】テーブルを結合するコマンドとして、ユニケージには join 系（join0、join1、join2）、cjoin 系（cjoin0、cjoin1、cjoin2）の各コマンドが存在しています。

まとめ

サーバーの夜間処理で必要とする項目（LV3 データ）と、人が PC 画面から操作して必要とする項目（LV4 データ）は、処理内容で異なります。そこで、PC 画面上から操作する際に使用する LV4 データを、あらかじめ画面で使用するレイアウトに成形しておけば、レスポンスの向上を図ることができます。このようにユニケージでは参照・検索する目的に合わせて、レイアウト（形）を作成することがあります。処理に合わせて最適なレイアウトを作成することが重要なのです。

次回は、PC 画面で参照する処理について紹介します。