GAE環境におけるSQLデータベース移行に関する技術的考察

－クラウドコンピューティングの可能性－

仙台CCTPプロジェクト
宮城大学　曽根　碧，イートス㈱金村　昌秀

0. 仙台CCTPの目的

クラウド時代到来に備え
技術ノーハウを蓄積

クラウドを活用した
ビジネスモデルを確立

0. 仙台CCTPの活動

MISAを通じて参加企業の募集

キックオフ・ミーティング

検証技術課題の提示

チーム単位での取り組み

0. 仙台CCTPの活動

16社の参加企業

2つの検証課題

・GAEによるアプリケーション開発
・RDB移行に関する検証

プロジェクトの継続

1. 検証テーマ1-3目的

従来のシステム開発と
GAEを利用したシステム開発の相違点

・データストア
・システム開発

実践的ノウハウの獲得

2. 開発環境

	既存	今回
OS	Linux	Windows
サーバー	Apache	GAE
言語	PHP	Java
データベース	MySQL	JDO

Google App Engine SDK for Java(バージョン：1.36)
Google Plug in for Eclipse (Eclipse 3.6(Helios))

3. データストア

1. Queryの使用
2. JDOQLの使用

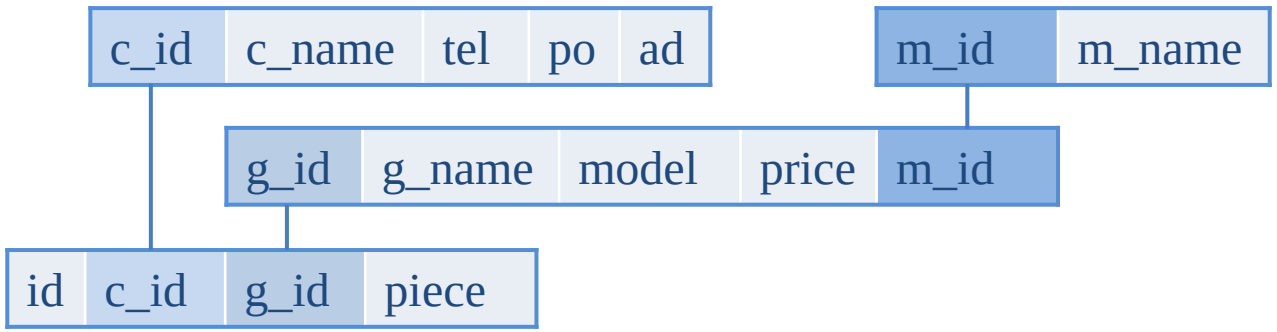
➡ORマッピング

Low level API

➡アクセス手法のみ

4. DB構造

従来のRDB



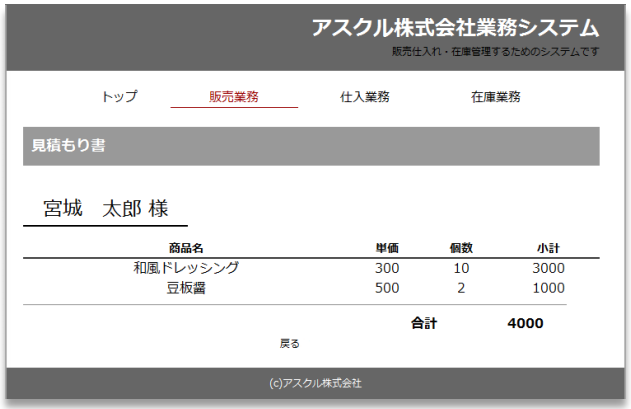
データを複数の「テーブル」に分割する。正規化が必要。

データストア

ID	ad	name	g_1	g_2	g_3	g_4	g_5	p_1	p_2	p_3	p_4	p_5	po	tell	v_1	v_2	v_3	v_4	v_5

データをそのままカインドで表現。正規化が不要。

5. アプリケーション



販売管理システム

http://sms-midori.appspot.com/

目的
開発環境
データストア
DB構造
アプリケーション
得られた知見
今後の計画

6. 得られた知見 –データストア–

例） パスワードを数字（Long）で指定していたが
英数混じり(String)に変更したい

○ **private** Long pass ⇒ **private** String password

× **private** Long pass ⇒ **private** String pass

× 合計金額 ＜ 30000 かつ 利用回数 ＜ 10

6. 得られた知見 –システム開発–

GAE上でのJDO

・JDOの機能が全て実装されている訳ではない。

非所有関係の記述方法
↓
GAE独自の記述方法

・JDOの動作が重く感じる

従来のシステムからGAEの移植について

・データストアはRDBとは異なるため、システムを設計段階から構築しなおす必要がある。

・従来のシステムをほとんど変えずにGAEに上げなおすのは不可能。

7. 今後の計画

- GAEで推奨されているJDOだが動作が重く感じる。
- JDOはリレーショナルデータベースを取り扱う事を前提に出来ている。

Low Level での開発
(Slim3の考察)

参考資料

【 すっきりわかるGoogle App Engine 】

著　中田秀基

【 Google App Engine 】

http://code.google.com/intl/ja/appengine/

2010年10月7日

イートス株式会社

SQLとJDOの比較

目的

既存システムをGoogle App Engine（以下GAE）に移行する際の課題等を明確にするために、リレーショナル型データベース（以下RDB）を利用したシステムをGAEのKeyValueStore型（以下KVS）のBigTableに移行し、Java Data Objects（以下JDO）を利用してSQL文を利用したRDBと同等のオペレーションを行えるか検証する。

実証の内容

１　テーブル定義の比較

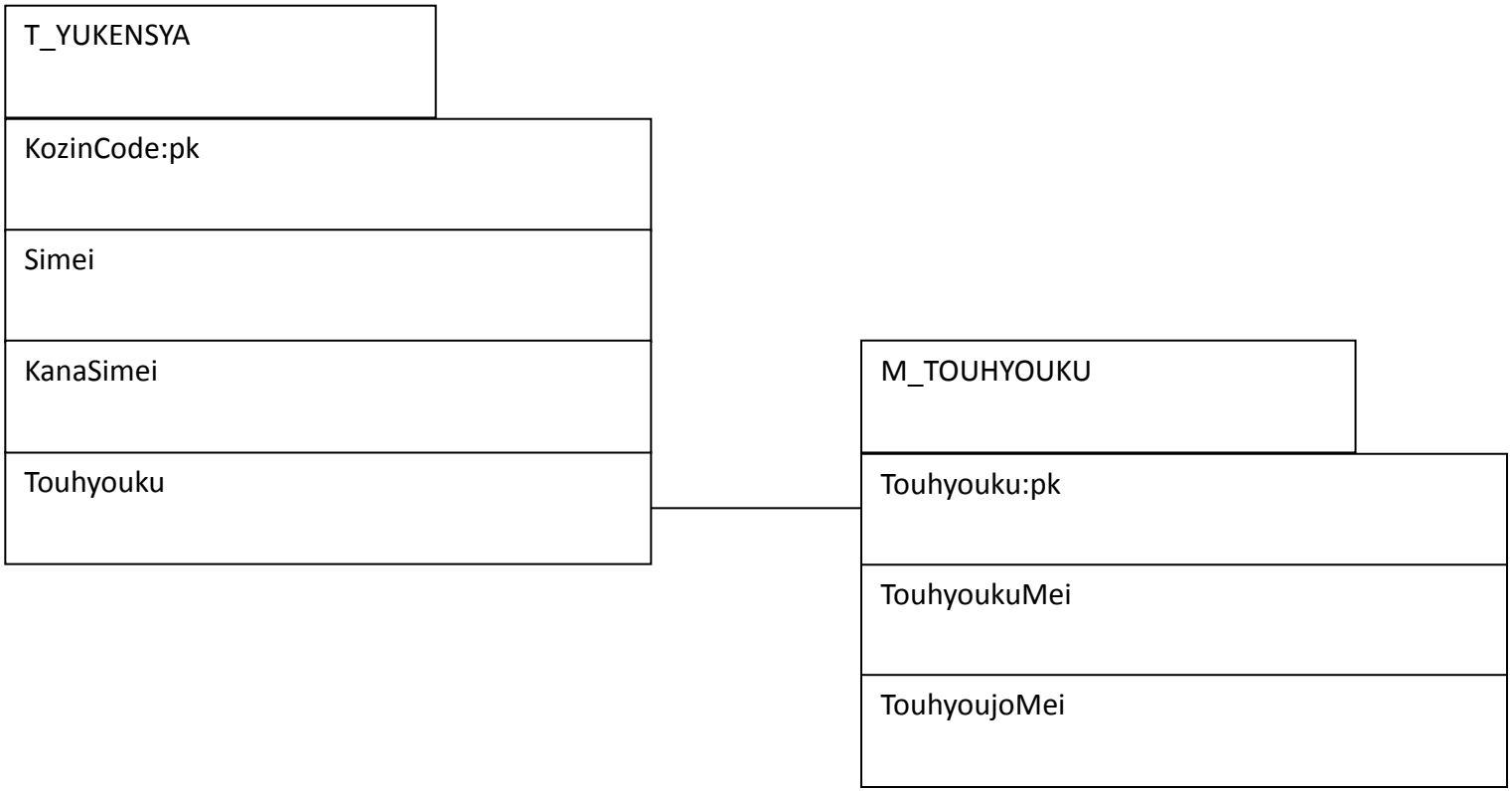
２　行追加の比較

３　照会の比較

４　行削除の比較

５　データ移行

正規化されたT_YUKENSYAテーブル



テーブル定義の比較

ＳＱＬ
テーブル定義(SQL Serverデータ型を使用)

```
1   CREATE TABLE T_YUKENSYA
2   (
3   KozinCode       BIGINT  NOT NULL  PRIMARY KEY,
4   Simei                       VARCHAR(50)     ,
5   KanaSimei       VARCHAR(25)     ,
6   SeibetuKubun    INTERGER        ,
7   Seinengappi     DATETIME        ,
8   YubinBangou     VARCHAR(8)      ,
9   Genjyusyo       VARCHAR(80)     ,
10  DenwaBangou     VARCHAR(20)     ,
11  Touhyouku       BIGINT          ,
12  Page                        BIGINT          ,
13  Bangou                      BIGINT          ,
14  TensyutsuBi     DATETIME        ,
15  TensyutsusakiGenjyusyo      VARCHAR(80)     ,
16  Senkyokubun     BIGINT          ,
17  SikkensyaFLG    BIT             ,
18  SibousyaFLG     BIT             ,
19  NijyutorokuFLG  BIT             ,
20  KosinFLG                    BIT             ,
21  SeninTourokuFLG             BIT             ,
22  IdoKousinBi     DATETIME        ,
23  TensyutsuKubun  BIGINT          ,
24  SeionKanaSimei  VARCHAR(50)     ,
25  NankyokuFLG     BIT             ,
26  TennyuBi                    DATETIME        ,
27  SeijinhikoukenninFLG        BIT             ,
28  NyujyokenHensouBi           DATETIME        ,
29  NyujyokenHassouBi           DATETIME        ,
30  Meno                        VARCHAR(80)
31  )
```

ＪＤＯ
エンティティ(クラス)を定義する。定義する内容はSQLの
CREATE TABLE文をJavaのデータ型にあわせたものとなる、
各フィールドにアクセスするためにゲッター・セッターが必要

```
1   import javax.jdo.annotations.*      //データストアでのJava クラスの格納と取得に JDO を使用できるようにする
2   @PersistenceCapable(
3   identityType = IdentityType.APPLICATION, //主キーをアプリケーションで指定する
4   detachable = "true") //PersistenceManager を閉じてからオブジェクトを修正できるようにする
5   public class T_YUKENSYA {
6    @PrimaryKey  //主キーの指定
7    @Persistent (valueStrategy = IdGeneratorStrategy.IDENTITY) //永続フィールドとして宣言する
8    private Long KozinCode; //永続化時にIDが自動採番されるように指定
9    @Persistent
10   private String Simei;
11   @Persistent
12   private String KanaSimei;
13   @Persistent
14   private boolean SeibetuKubun;
15   @Persistent
16   private Date Seinengappi;
17   @Persistent
18   private String YubinBangou;
19   @Persistent
20   private String Genjyusyo;
21   @Persistent
22   private String DenwaBangou;
23   @Persistent
24   private long Touhyouku;
25   @Persistent
26   private long Page;
27   @Persistent
28   private long Bangou;
29   @Persistent
30   private Date TensyutsuBi;
31   @Persistent
32   private String TensyutsusakiGenjyusyo;
33   @Persistent
34   private long Senkyokubun;
35   @Persistent
36   private boolean SikkensyaFLG;
37   @Persistent
38   private boolean SibousyaFLG;
39   @Persistent
40   private boolean NijyutorokuFLG;
41   @Persistent
42   private boolean KosinFLG;
43   @Persistent
44   private boolean SeninTourokuFLG;
45   @Persistent
46   private Date IdoKousinBi;
47   @Persistent
48   private long TensyutsuKubun;
49   @Persistent
50   private String SeionKanaSimei;
51   @Persistent
52   private boolean NankyokuFLG;
53   @Persistent
54   private Date TennyuBi;
55   @Persistent
56   private boolean SeijinhikoukenninFLG;
57   @Persistent
58   private Date NyujyokenHensouBi;
```

```
59        @Persistent
60        private Date NyujyokenHassouBi;
61        @Persistent
62        private String Meno;
63        //各プロパティのゲッター、セッターのアクセサを宣言する
64        public void setKozinCode(Long kozinCode) {
65         KozinCode = kozinCode;
66        }
67        public Long getKozinCode() {
68         return KozinCode;
69        }
70        public void setSimei(String simei) {
71         Simei = simei;
72        }
73        public String getSimei() {
74         return Simei;
75        }
76        public void setKanaSimei(String kanaSimei) {
77         KanaSimei = kanaSimei;
78        }
79        public String getKanaSimei() {
80         return KanaSimei;
81        }
82        public void setSeibetuKubun(boolean seibetuKubun) {
83         SeibetuKubun = seibetuKubun;
84        }
85        public boolean isSeibetuKubun() {
86         return SeibetuKubun;
87        }
88        public void setSeinengappi(Date seinengappi) {
89         Seinengappi = seinengappi;
90        }
91        public Date getSeinengappi() {
92         return Seinengappi;
93        }
```

```
94        public void setYubinBangou(String yubinBangou) {
95         YubinBangou = yubinBangou;
96        }
97        public String getYubinBangou() {
98         return YubinBangou;
99        }
100       public void setGenjyusyo(String genjyusyo) {
101        Genjyusyo = genjyusyo;
102       }
103       public String getGenjyusyo() {
104        return Genjyusyo;
105       }
106       public void setTouhyouku(long touhyouku) {
107        Touhyouku = touhyouku;
108       }
109       public long getTouhyouku() {
110        return Touhyouku;
111       }
112       public void setDenwaBangou(String denwaBangou) {
113        DenwaBangou = denwaBangou;
114       }
115       public String getDenwaBangou() {
116        return DenwaBangou;
117       }
118       public void setPage(long page) {
119        Page = page;
120       }
121       public long getPage() {
122        return Page;
123       }
124       public void setBangou(long bangou) {
125        Bangou = bangou;
126      }
127
```

```
128      public long getBangou() {
129       return Bangou;
130      }
131      public void setTensyutsuBi(Date tensyutsuBi) {
132       TensyutsuBi = tensyutsuBi;
133      }
134      public Date getTensyutsuBi() {
135       return TensyutsuBi;
136      }
137      public void setTensyutsusakiGenjyusyo(String
         tensyutsusakiGenjyusyo) {
138       TensyutsusakiGenjyusyo = tensyutsusakiGenjyusyo;
139      }
140      public String getTensyutsusakiGenjyusyo() {
141       return TensyutsusakiGenjyusyo;
142      }
143      public void setSenkyokubun(long senkyokubun) {
144       Senkyokubun = senkyokubun;
145      }
146      public long getSenkyokubun() {
147       return Senkyokubun;
148      }
149      public void setSikkensyaFLG(boolean sikkensyaFLG) {
150       SikkensyaFLG = sikkensyaFLG;
151      }
152      public boolean isSikkensyaFLG() {
153       return SikkensyaFLG;
154      }
155      public void setSibousyaFLG(boolean sibousyaFLG) {
156       SibousyaFLG = sibousyaFLG;
157      }
158      public boolean isSibousyaFLG() {
159       return SibousyaFLG;
160      }
```

```
161      public void setKosinFLG(boolean kosinFLG) {
162       KosinFLG = kosinFLG;
163      }
164      public boolean isKosinFLG() {
165       return KosinFLG;
166      }
167      public void setNijyutorokuFLG(boolean
         nijyutorokuFLG) {
168       NijyutorokuFLG = nijyutorokuFLG;
169      }
170      public boolean isNijyutorokuFLG() {
171       return NijyutorokuFLG;
172      }
173      public void setSeninTourokuFLG(boolean
         seninTourokuFLG) {
174       SeninTourokuFLG = seninTourokuFLG;
175      }
176      public boolean isSeninTourokuFLG() {
177       return SeninTourokuFLG;
178      }
179      public void setIdoKousinBi(Date idoKousinBi) {
180       IdoKousinBi = idoKousinBi;
181      }
182      public Date getIdoKousinBi() {
183       return IdoKousinBi;
184      }
185      public void setTensyutsuKubun(long tensyutsuKubun) {
186       TensyutsuKubun = tensyutsuKubun;
187      }
188      public long getTensyutsuKubun() {
189       return TensyutsuKubun;
190      }
```

```
191      public void setSeionKanaSimei(String seionKanaSimei)
         {
192       SeionKanaSimei = seionKanaSimei;
193      }
194      public String getSeionKanaSimei() {
195       return SeionKanaSimei;
196      }
197      public void setNankyokuFLG(boolean nankyokuFLG) {
198       NankyokuFLG = nankyokuFLG;
199      }
200      public boolean isNankyokuFLG() {
201       return NankyokuFLG;
202      }
203      public void setTennyuBi(Date tennyuBi) {
204       TennyuBi = tennyuBi;
205      }
206      public Date getTennyuBi() {
207       return TennyuBi;
208      }
209      public void setSeijinhikoukenninFLG(boolean
         seijinhikoukenninFLG) {
210       SeijinhikoukenninFLG = seijinhikoukenninFLG;
211      }
212      public boolean isSeijinhikoukenninFLG() {
213       return SeijinhikoukenninFLG;
214      }
215      public void setNyujyokenHensouBi(Date
         nyujyokenHensouBi) {
216       NyujyokenHensouBi = nyujyokenHensouBi;
217      }
218      public Date getNyujyokenHensouBi() {
219       return NyujyokenHensouBi;
220      }
```

```
221      public void setNyujyokenHassouBi(Date
         nyujyokenHassouBi) {
222       NyujyokenHassouBi = nyujyokenHassouBi;
223      }
224      public Date getNyujyokenHassouBi() {
225       return NyujyokenHassouBi;
226      }
227      public void setMeno(String meno) {
228       Meno = meno;
229      }
230      public String getMeno() {
231       return Meno;
232      }
233     }
```

テーブル定義の比較結果

１　RDBでテーブルを定義する場合はSQLでCREATE TABLE文を記述するがGAEのJDOではエンティティ(クラス)の定義を行わなくてはならないためソースのコーディング量がSQLでは31行に対しJDOでは232行となりコーディング量は７倍強となる。
　ただしアクセサの記述はEclipseの機能で自動で作成できるため実際のコーディング量は61行程度

２　エンティティのキーは一度格納されたら変更できないためキーの設計には注意が必要。

行追加の比較

SQL

```
1               INSERT INTO T_YUKENSYA
2               (
3                個人コード
4                ,氏名
5                ,カナ氏名
6                ,性別
7                ,生年月日
8                ,郵便番号
9                ,現住所地
10               ,電話番号
11               ,投票区
12               ,頁
13               ,番号
14               ,転出日
15               ,転出先現住所地
16               ,選挙区分
17               ,失権者FLG
18               ,死亡者FLG
19               ,二重登録者FLG
20               ,更新FLG
21               ,船員登録者FLG
22               ,異動更新日
23               ,転出区分
24               ,期日前投票所コード
25               ,指定投票区
26               ,選挙区コード
27               ,清音カナ氏名
28               ,南極FLG
29               ,転入日
30               ,成年後被後見人FLG
31               ,入場券返送日
32               ,入場券発送日
33               ,メモ
34              ) VALUES (
35               個人コードの値
36               ,氏名の値
37               ,カナ氏名の値
38               ,性別の値
39               ,生年月日の値
40               ,郵便番号の値
41               ,現住所地の値
42               ,電話番号の値
43               ,投票区の値
44               ,頁の値
45               ,番号の値
46               ,転出日の値
47               ,転出先現住所地の値
48               ,選挙区分の値
49               ,失権者FLGの値
50               ,死亡者FLGの値
51               ,二重登録者FLGの値
52               ,更新FLGの値
53               ,船員登録者FLGの値
54               ,異動更新日の値
55               ,転出区分の値
56               ,期日前投票所コードの値
57               ,指定投票区の値
```

```
58          ,選挙区コードの値
59          ,清音カナ氏名の値
60          ,南極FLGの値
61          ,転入日の値
62          ,成年後被後見人FLGの値
63          ,入場券返送日の値
64          ,入場券発送日の値
65          ,メモの値
66        )
```

ＪＤＯ

エンティティのインスタンスにプロパティ値をセットし
makePersistentで追加する

```
1       PersistenceManager pm =
        F.get().getPersistenceManager();
2       T_YUKENSYA my = new T_YUKENSYA();  //クラス(エンティ
        ティ)をインスタンス化
3       String[] si = ls.split(",", -1 );
4       //クラスの各フィールドに値をセットしていく
5       //個人コード
6       my.setKozinCode(Long.parseLong(si[0]));
7       //氏名
8       my.setSimei(si[1]);
9       //カナ氏名
10      my.setKanaSimei(si[2]);
11      //性別区分
12      if(si[3].length() != 0)
13       my.setSeibetuKubun(Integer.parseInt(si[3]));
14      //生年月日
15      if(si[4].length() != 0)
16      my.setSeinengappi(sdf.parse(si[4]));
17      //郵便番号
18      my.setYubinBangou(si[5]);
19      //現住所
20      my.setGenjyusyo(si[6]);
21      //電話番号
22      my.setDenwaBangou(si[7]);
23      //投票区
24      if(si[8].length() != 0)
25       my.setTouhyouku(Integer.parseInt(si[8]));
26      //ページ
27      if(si[9].length() != 0)
28       my.setPage(Integer.parseInt(si[9]));
```

```
29      //番号
30      if(si[10].length() != 0)
31       my.setBangou(Integer.parseInt(si[10]));
32      //転出日
33      if(si[11].length() != 0)
34       my.setTensyutsuBi(sdf.parse(si[11]));
35      //転出先現住所
36      my.setTensyutsusakiGenjyusyo(si[12]);
37      //選挙区分
38      if(si[13].length() != 0)
39       my.setSenkyokubun(Integer.parseInt(si[13]));
40      //執権者FLG
41      if(si[14].length() != 0)
42      {
43       if(!si[14].equals("0"))
44        my.setSikkensyaFLG(true);
45       else
46        my.setSikkensyaFLG(false);
47      }
48      //死亡者FLG
49      if(si[15].length() != 0)
50      {
51       if(!si[15].equals("0"))
52        my.setSibousyaFLG(true);
53       else
54        my.setSibousyaFLG(false);
55      }
56      //二重登録者FLG
57      if(si[16].length() != 0)
58      {
59       if(!si[16].equals("0"))
60        my.setNijyutorokuFLG(true);
61       else
62        my.setNijyutorokuFLG(false);
63      }
```

```
64      //更新FLG
65      if(si[17].length() != 0)
66      {
67       if(!si[17].equals("0"))
68        my.setKosinFLG(true);
69       else
70        my.setKosinFLG(false);
71      }
72      //船員登録者FLG
73      if(si[18].length() != 0)
74      {
75       if(!si[18].equals("0"))
76        my.setSeninTourokuFLG(true);
77       else
78        my.setSeninTourokuFLG(false);
79      }
80      //異動更新日
81      if(si[19].length() != 0)
82       my.setIdoKousinBi(sdf.parse(si[18]));
83      //転出区分
84      if(si[20].length() != 0)
85       my.setTensyutsuKubun(Integer.parseInt(si[20]));
86      //清音カナ氏名
87      my.setSeionKanaSimei(si[21]);
88      //南極FLG
89      if(si[22].length() != 0)
90      {
91       if(!si[22].equals("0"))
92        my.setNankyokuFLG(true);
93       else
94        my.setNankyokuFLG(false);
95      }
```

```
96      //転入日
97      if(si[23].length() != 0)
98       my.setTennyuBi(sdf.parse(si[23]));
99      //成年後被後見人FLG
100     if(si[24].length() != 0)
101     {
102      if(!si[24].equals("0"))
103       my.setSeijinhikoukenninFLG(true);
104      else
105       my.setSeijinhikoukenninFLG(false);
106     }
107     //入場券返送日
108     if(si[25].length() != 0)
109      my.setNyujyokenHensouBi(sdf.parse(si[25]));
110     //入場券発送日
111     if(si[26].length() != 0)
112      my.setNyujyokenHassouBi(sdf.parse(si[26]));
113     //メモ
114     my.setMeno(si[27]);
115     //追加
116     pm.makePersistent(my);  //データストアにデータ追加
```

行追加の比較の結果

１　RDBではINSERT文を実行するがGAEではエンティティのインスタンスを作成しエンティティのプロパティに値をセットしたものをmakePrsistentメソッドで追加する。

行削除の比較

ＳＱＬ

```
1        DELETE FROM T_YUKENSYA
```

ＪＤＯ

```
1        PersistenceManager pm =
         PMF.get().getPersistenceManager();
2        Extent<T_YUKENSYA> ext =
         pm.getExtent(T_YUKENSYA.class , false);
3        List<Long> kozincode = new ArrayList<Long>();
4
5        //すべてのデータを削除
6        //ExtentではdeletePersistentAll使えない
7        //pm.deletePersistentAll(ext);
8
9        //主キーをすべて取得
10       for(T_YUKENSYA ty : ext)
11       {
12        kozincode.add(ty.getKozinCode());
13       }
14       ext.closeAll();
15
16       //すべてのデータを主キーで取得し削除
17       for(Long l : kozincode)
18       {
19        T_YUKENSYA ty = pm.getObjectById(T_YUKENSYA.class,
          l );
20        pm.deletePersistent(ty);
21       }
22       pm.close();
```

行削除の比較の結果

１　RDBではDELETE文を実行するだけですむが、JDOで全件削除するにはExtentで全てのエンティティを取得し主キー一文ループを回して削除する必要がある。

２　deletePersistentAllで削除する場合は1000件単位でデータを削除しデータがなくなるまでループを回す必要がある。

照会の比較

全項目・全件取得

ＳＱＬ

```
1       SELECT
2        *
3       FROM
4        T_YUKENSYA
```

ＪＤＯＱＬ

SELECTでは検索結果1000上限まで
Extentは上限なし

SELECTで全件取得する場合

```
1       PersistenceManager pm =
        PMF.get().getPersistenceManager();
2       String q = "SELECT FROM " +
        T_YUKENSYA.class.getName();
3       List<T_YUKENSYA> rows = (List<T_YUKENSYA>)
        pm.newQuery(q).execute();
```

Extentで全権取得する場合

```
1       PersistenceManager pm =
        PMF.get().getPersistenceManager();
2       Extent<T_YUKENSYA> ext =
        pm.getExtent(T_YUKENSYA.class , false);
```

全項目・条件指定

ＳＱＬ

```
1       SELECT
2        *
3       FROM
4        T_YUKENSYA
5       WHERE
6        SeibetuKubun = 1
```

ＪＤＯＱＬ

クエリ文字列を作成しexecuteで実行する

条件を引数として渡す場合

```
1       PersistenceManager pm =
        PMF.get().getPersistenceManager();
2       String q = "SELECT FROM " +
        T_YUKENSYA.class.getName()
3        + " where SeibetuKubun == kbn “  //プロパティとパラメータを条件とする
4        + "parameters String kbn";  //条件式をパラメータとして指定
5       List<T_YUKENSYA> rows = (List<T_YUKENSYA>)
        pm.newQuery(q).execute(1);  //executeの引数に条件値をセット
```

条件をそのままクエリとする場合

```
1       PersistenceManager pm =
        PMF.get().getPersistenceManager();
2       String q = "SELECT FROM " +
        T_YUKENSYA.class.getName()
3        + " where SeibetuKubun == 1 “  //条件を直接指定する
4       List<T_YUKENSYA> rows = (List<T_YUKENSYA>)
        pm.newQuery(q).execute();
```

選択項目・条件指定

ＳＱＬ

```
1       SELECT
2        KozinCode
3        ,Simei
4        ,Genjyusyo
5       FROM
6        T_YUKENSYA
7       WHERE
8        SeibetuKubun = 1
```

ＪＤＯＱＬ

JDOQLではすべてのフィールドを取得するため項目の選択はできないため使用するプロパティをアプリケーションで選別する。

```
1       PersistenceManager pm =
        PMF.get().getPersistenceManager();
    //クエリ文字列を作成
2       String q = "SELECT FROM " + T_YUKENSYA.class.getName

3        + " where SeibetuKubun == 1 "
4       Query qs = pm.newQuery(q);  //クエリオブジェクトをインスタンス化
5       qs.setResult( “KozinCode,Simei,Genjyusyo” );  //取得するフィールドリストをセット
6       Collection results = (Collection)qs.execute();  //クエリ実行
7       Iterator iter = results.iterator();  //クエリ閉じる
8       while (iter.hasNext())  //次のデータがあるうちループ
9       {
10       Object[] row = (Object[])iter.next();  //イテレータからデータ取得
11       out.print("<TD>"+row[0]+"</TD>");
12       out.print("<TD>"+row[1]+"</TD>");
13       out.print("<TD>"+row[2]+"</TD>");
14       out.print("</TR>");
15      }
```

並び替え

SQL

```
1       SELECT
2        KozinCode
3        ,Simei
4        ,Genjyusyo
5        ,NyujyokenHassouBi
6       FROM
7        T_YUKENSYA
8       WHERE
9        SeibetuKubun = 1
10      ORDER BY
11       NyujyokenHassouBi
```

JDOQL

```
1       PersistenceManager pm =
        PMF.get().getPersistenceManager();
2       String q = "SELECT FROM " +
        T_YUKENSYA.class.getName()
3        + " where SeibetuKubun == 1 "
4        + " order by NyujyokenHassouBi ";  //クエリ文字列作
        成
5       Query qs = pm.newQuery(q);  //クエリオブジェクトイン
        スタンス化
6       qs.setResult("KozinCode,Simei,Genjyusyo,
        NyujyokenHassouBi");  //結果リストをセット
7       Collection results = (Collection)qs.execute();  //ク
        エリ実行
8       Iterator iter = results.iterator();
9
10      while (iter.hasNext())  //次のデータがあるうちループ
11      {
12       Object[] row = (Object[])iter.next();  //イテレータ
        から
         データ取得
13       out.print("<TR>");
14       out.print("<TD>"+row[0]+"</TD>");
15       out.print("<TD>"+row[1]+"</TD>");
16       out.print("<TD>"+row[2]+"</TD>");
17       out.print("<TD>"+row[3]+"</TD>");
18       out.print("</TR>");
19      }
```

グループ化カウント

SQL

```
1       SELECT
2        COUNT(KozinCode) AS CNT
3       FROM
4        T_YUKENSYA
5       GROUP BY
6        Touhyouku
```

JDOQL

```
1       PersistenceManager pm =
        PMF.get().getPersistenceManager();
2       String q = "SELECT FROM " +
        T_YUKENSYA.class.getName();  //クエリ文字列を作成
3       Query qs = pm.newQuery(q);  //クエリオブジェクトをイ
        ンスタンス化
4       qs.setResult("count(this.kozincode)");  //取得する
        フィールドリストをセット
5       out.print(qs.execute());  //クエリ実行
6       qs.closeAll();  //クエリ閉じる
```

結合（ＬＥＦＴ　ＪＯＩＮ）

ＳＱＬ

```
1       SELECT
2        TY.KozinCode
3        ,TY.Simei
4        ,TY.KanaSimei
5        ,TY.Touhyouku
6        ,MT.TouhyoukuMei
7        ,MT.TouhyoujoMei
8       FROM
9        T_YUKENSYA AS TY
10      LEFT JOIN
11       M_TOUHYOKU AS MT
12      ON
13       TY.Touhyouku = MT.Touhyouku
```

ＪＤＯＱＬ

（T_YUKENSYAテーブルとM_TOUHYOKUテーブルをプログラムでJOINした場合）

```
1       //JDOでデータストアを操作するため、PersistenceManager
        のインスタンスを生成
2       PersistenceManager pm =
        PMF.get().getPersistenceManager();
3       String q = "SELECT FROM " +
        T_YUKENSYA.class.getName();  //有権者データを取得する
4       List<T_YUKENSYA> rows_ty = (List<T_YUKENSYA>)
        pm.newQuery(q).execute();
5       if(rows_ty.isEmpty())
6       }
7       else
8       {
9        for(T_YUKENSYA ty : rows_ty)
10       {
11        //有権者データと投票区データをJOINした値を格納する
          D_YUKENSYAクラスのインスタンスを生成
12        D_YUKENSYA dy = new D_YUKENSYA();
13        //D_YUKENSYAクラスのインスタンスに有権者テーブルの
          値を格納
14        dy.setKozinCode(ty.getKozinCode());
15        dy.setSimei(ty.getSimei());
16        dy.setTouhyouku(ty.getTouhyouku());
17        try
18        {
19         //投票区テーブルに、対象の投票区の情報があるか検索
20         M_TOUHYOKU mt = pm.getObjectById(M_TOUHYOKU.class,
          ty.getTouhyouku());
```

```
22          //見つかった場合、投票区名、投票所名をセット
23          dy.setTouhyoukuMei(mt.getTouhyoukuMei());
24          dy.setTouhyoujoMei(mt.getTouhyoujoMei());
25         }
26          catch (JDOObjectNotFoundException e)
27         {
28          //見つからなかった場合、空文字をセット
29          dy.setTouhyoukuMei("");
30          dy.setTouhyoujoMei("");
31         }
32          out.print(dy.toStringHTML());
33          dy = null;
34       }
35      }
```

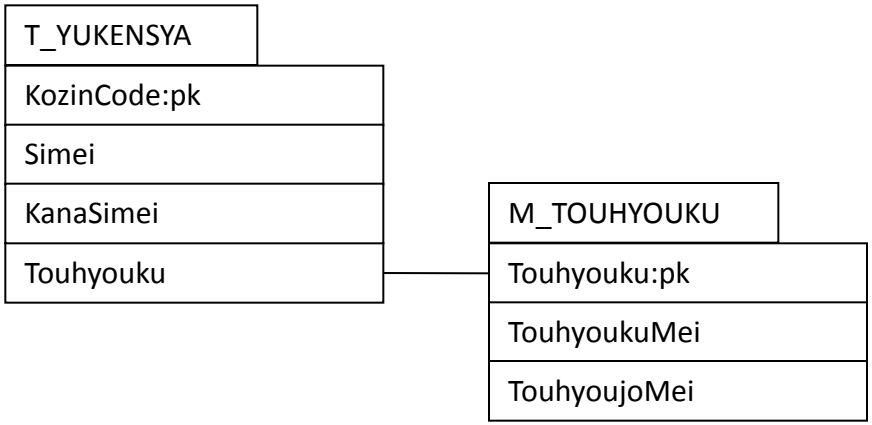
＠ＩＴ総合トップの記事より

Google App Engine担当者に聞いた
クラウド環境ではデータベースは「非正規化」して使う？
2009/06/09

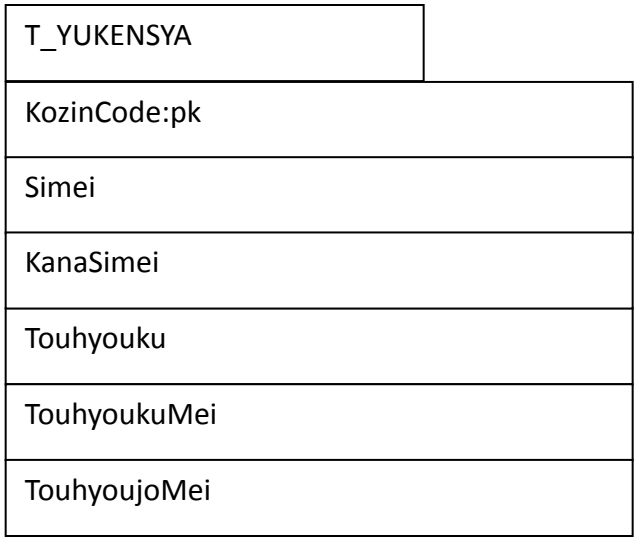
上記、記事の中で、Google App Engine担当者のソーサー氏 は、
『これまでRDBでさんざんたたき込まれてきた「正規化」とは
逆に、クラウド環境では「データを非正規化する」ように
考え方を改める必要があります。 』と話しています。

記事のＵＲＬ
http://www.atmarkit.co.jp/news/200906/09/gae.html

正規化されたT_YUKENSYAテーブル



非正規化されたT_YUKENSYAテーブル



非正規化されているテーブルの場合

ＳＱＬ

正規化されたテーブルの場合（JOINで照会）

```
1       SELECT
2        TY.KozinCode
3        ,TY.Simei
4        ,TY.KanaSimei
5        ,TY.Touhyouku
6        ,MT.TouhyoukuMei
7        ,MT.TouhyoujoMei
8       FROM
9        T_YUKENSYA AS TY
10      LEFT JOIN
11       M_TOUHYOKU AS MT
12      ON
13       TY.Touhyouku = MT.Touhyouku
```

↓

非正規化されたテーブルの場合

```
1       SELECT
2        TY.KozinCode
3        ,TY.Simei
4        ,TY.KanaSimei
5        ,TY.Touhyouku
6        ,TY.TouhyoukuMei
7        ,TY.TouhyoujoMei
8       FROM
9        T_YUKENSYA AS TY
```

ＪＤＯＱＬ

正規化されたテーブルの場合（プログラムでJOINして照会）

```
1       //JDOでデータストアを操作するため、PersistenceManager
        のインスタンスを生成
2       PersistenceManager pm =
        PMF.get().getPersistenceManager();
3       String q = "SELECT FROM " +
        T_YUKENSYA.class.getName();     //有権者データを取得す
        る
4       List<T_YUKENSYA> rows_ty = (List<T_YUKENSYA>)
        pm.newQuery(q).execute();
                …　（前述したソースを参照）
32          out.print(dy.toStringHTML());
33          dy = null;
34       }
35      }
```

↓

非正規化されたテーブルの場合

```
1       PersistenceManager pm =
        PMF.get().getPersistenceManager();
2       String q = "SELECT FROM " +
        D_YUKENSYA.class.getName();
3       List<D_YUKENSYA> rows = (List<D_YUKENSYA>)
        pm.newQuery(q).execute();
```

照会の比較の結果

１　結果セットは1000件が上限であるため、工夫が必要となる。

２　集計関数は1000件分のみ集計されるため、工夫が必要となる。

３　GAEは非正規化したデータ構造を想定しているため、データの照会を踏まえたうえで最適な設計に変更する必要がある。

データ移行
CSVファイルをアップし、データストアに展開

JDO

```
1       //KeyValueStoreにアクセスするために必要
2       PersistenceManager pm =
        PMF.get().getPersistenceManager();
3       try {
4        //データ書込み開始時間取得
5        long starttime = System.currentTimeMillis() / 1000;
6        //処理件数カウンタ
7        long l = 1;
8        //日付型に変換するための準備
9        SimpleDateFormat sdf = new SimpleDateFormat();
10       sdf.setLenient( true );
11       sdf.applyPattern( "yyyy/MM/dd" );
12       //ストリーム読み込み準備
13       InputStreamReader ir = new InputStreamReader(stream,
        "UTF-8");
14       BufferedReader br = new BufferedReader(ir);
15       String line;
16
17       //レコードの追加
18       //バッファから一行読み込む
19       while ((line = br.readLine()) != null)
20       {
21        //エンティティのインスタンス化
22        T_YUKENSYA my = new T_YUKENSYA();
23        //読み込んだ行をカンマでスプリットする
24        String[] si = line.split(",", -1 );
```

```
25        try
26        {
27         //個人コード
28         my.setKozinCode(Long.parseLong(si[0]));
29         //氏名
30         my.setSimei(si[1]);
31         //カナ氏名
32         my.setKanaSimei(si[2]);
                                  …
          (CSVファイルの各値をT_YUKENSYAクラスに格納)
                                  …
132        //入場券発送日
133        if(si[26].length() != 0)
134         my.setNyujyokenHassouBi(sdf.parse(si[26]));
135        //メモ
136        my.setMeno(si[27]);
137       }
138       catch(Exception e)
139       {
140        log.warning(String.valueOf(l));
141        e.printStackTrace();
142        continue;
143       }
144       //データ書込み
145       pm.makePersistent(my);
146       //GAEのレスポンス30秒縛りを回避するため25過ぎたら
          レスポンスを返す
```

```
147         if((System.currentTimeMillis() / 1000) -
            starttime >= 25)
148         {
149          PrintWriter out = res.getWriter();
150          out.println("<HTML>");
151          out.println("<head>");
152          out.println("<meta http-equiv=¥"content-type¥"
            content=¥"text/html; charset=UTF-8¥">");
153          out.println("</head>");
154          out.println("<BODY>");
155          out.println("処理中です" + String.valueOf(l) +
            "件処理終了");
156          out.println("</BODY>");
157          out.println("</HTML>");
158          out.flush();  //(4)データ返信の終了
159          starttime = (System.currentTimeMillis() / 1000);
160         }
161         l++;
162       }
163
164       br.close();
165       ir.close();
166      } catch (Exception e) {
167       e.printStackTrace();
168      }
169      finally
170      {
171       pm.close();
172      }
```

データ移行の結果

１　30秒のリクエストがない場合、セッションが切れるため
　大量のデータを処理する場合は工夫が必要となる。

２　大量データを処理する場合、トラフィックも問題になる。

課題のまとめ

１　リクエストが30秒縛りへの対応。

２　処理件数上限1000件への対応。

３　スピンアップの必要時間によるストレスの対応。

４　データ構造の再設計による作業冗長。

５　上記課題解決のためフレームワークの導入の検討。
（Slim3など）