EDB プログラマと

管理者のための手引き

平成 30 年 12 月 8 日 18 時 49 分

大家 隆弘@徳島大学
alex@db.tokushima-u.ac.jp

目 次

第I部

EDBの概要

第1章　概要

1.1　まえがき

近年，政府機関，自治体，公共団体などの情報開示がすすむなかで，大学においても様々な情報の公開が望まれている．従来，公開のための情報の取りまとめは，調査文書の配布，職員がそれに記入という方法をとっていた．そのため，たとえ同様の公開情報のための調査であってもその都度記入作業が必要であり，また調査毎に様式が変更されるため様式を変更して記入するための手間が必要であった．また，書面による調査は，最終的に書面からワードプロセッサなどへの入力作業が発生し，事務（もしくは，教官，技官）の作業負担となるほか，入力誤りの検査のために校正刷りの確認作業が必要となる．このような作業は，年々増加の傾向にあり，全体の職務のなかで次第に大きな割合を占めるようになっている．

一方，情報工学の分野ではネットワークを介して情報を共有，管理する技術が進展していることは周知の事実であり，工学部の職員がいつまでも書面による作業負担に関わっていることは社会的な水準からみて時代遅れである．

データベースソフトウェアなどを用いてネットワークを介して情報を管理，共有することの利点としては，

- 入力作業負担の分散

  入力作業を分散させることにより，特定の部署に作業が集中することがなくなる．これにより調査開始から情報の取りまとめまでの期間を短縮することができる．また，個々の情報を，その情報に一番関係深い個人が入力することにより，単純な入力ミスの回避，および入力データの確認を行なうことができる．

- 情報の再利用

  一度入力した情報は当然再利用できる．調査の際に足りない情報のみを新規に登録するだけでよい．

- 情報の表現様式の統一

  かつて，フロッピーディスクや電子メールを用いた調査が行なわれたことがあるが，ディスクに格納されたファイルやメールの添付文書の様式の不一致が情報収集者を悩ませる結果となった．

- 情報の即時性

  従来，情報の公開は１年から数年のサイクルで行なわれ，また，調査から公開までは数ヵ月の期間が必要なため，公開時には既に情報は古くなっている．

データベースから公開情報を作成する作業を自動化することによって，常に最新の情報を公開することが可能となる．

- インターネット上への情報公開促進

  前項とも関連するが WWW 等のメディアに情報を変換することによって，従来手作業で行なっていた，Web 頁の作成を自動化することができる．

などがある．

我々ワーキンググループは，本学部の情報を一つのデータベースにより管理し，必要に応じて情報を取り出すシステムを構築している．本稿は，現在作成している工学部情報管理データベースの概要を述べたものである．

1.2　要求事項

組織の情報をデータベースにより管理し，公開する際には，情報の信頼性を考慮しなくてはならない．

- **情報の履歴**

  データベースの登録情報が「何時」,「誰によって」,「どのように（登録）変更された」を履歴として保存する必要がある．特に，利用者が学部教職員のように多数にまたがる場合，情報を登録（変更）した人物を特定することは重要である．事後において情報の操作者を特定できることにより，無責任な情報登録が減少し，全体の情報の信頼性を向上させることが可能である．

- **情報の権限**

  一方，登録されている情報の変更権限を各情報に付加しておくことも必要である．大学において悪意のある情報改変は少ないと思われるが，操作の誤りによって目的外の情報を変更してしまう可能性を減らすことができる．

- **統一された様式**

  データベースから情報を取り出し目的を果たす（冊子の編纂，統計情報の作成，他）には，登録情報が統一された様式により作成されていることが必要である．

- **情報の識別性**

  データベースに登録されている情報を分類する際に，情報を正しく分類することが必要である．例えば，論文などの著者による分類を行う場合を想定すると，人名には同姓同名の問題があるため，人名そのものによる分類には信憑性がない．

- **記述の統一性**

  データベースに登録されている情報を元に業績リストを作成する際に，雑誌名や組織名の統一が取られている必要がある．

1.3　特徴

本節では，現在構築中のデータベースの目標としている特徴について述べる．

- **アクセス方法**
  - WWW による情報登録/修正
  - whois/TCP ポートを利用した情報の取り出し．
- **通知システム**

  E-mail による notification: 他者による登録情報の修正が発生したときに，情報の所有者および関係者に電子メールを用いて通知を行なう．
- **データ様式**
  - 英文，和文，和文読み
  - 変更履歴の保存
- **追記型登録**

  本データベースは登録情報の削除を行なわない．

  登録・修正された情報は全て履歴として保存し，誰が，何時，どのような登録を行なったかを全て保存する．

  削除の変わりに情報の無効化の機能を提供する．
- **参照型データ登録**

  同じ情報は複数のデータにまたがって登録されない．
- **セキュリティ**
  - SSL(https) による通信路暗号化
  - パスフレーズによるユーザ認証

  パスフレーズ認証は，主にデータベースへのアクセスの可否を認証するために利用する．

  言い替えると，データベース入口部分のログイン手順におよびアクセス不可な情報のチェックを個人認証により実現し， データの登録・修正は情報の前登録者，所有者に通知することで，登録データの確認作業を行なう．

第2章　EDBにおける情報管理

2.1　扱う情報情報について

基本原則として，現実世界においても唯一の情報を EDB においても唯一の情報として登録する．この考え方は，データベースの正規化と完全に合致するものである．

- **大原則:「現実世界において 1 つの事柄を登録情報 1 つに対応させる」**

  複数著者の著作においても登録情報は 1 つであり，複数の著者がその情報を共有する．

- **参照型記述方式**

  （他の情報を参照することによる情報記述）を採用.

  個人情報を含む全ての情報に識別子 (EID) を割り当て，著作の著者には個人情報を選択，その EID を記述する．

- **大前提:「 参照型記述方式を用いた場合のみ，分類の対象となる」**

2.2　EID と EOID

EDB では登録情報に個別に情報識別子 (EDB Identifier) を割り振っている．これはテーブルおよびタプルの区別無く割り振られている．すなわち，データベース内で EID は情報識別子としてユニークな状態を保っている．ただし，情報認証用のデータベースはこの範疇外である．

EOID (EDB Object Identifier) は情報の表現として登録されるオブジェクトの識別子である．オブジェクトは一つの情報に複数登録されることがある．その場合 EOID はデータベース内の登録順を表す値となり，大きい値の方はより未来を表す．すなわち，EID に対して複数の EOID で表現されるオブジェクトが存在する場合には，EOID が最大のオブジェクトがデータベース内での EID に対応する情報の最新の表現であり，それ以外はその情報の変更履歴と解釈される．

EOID はデータベース内のオブジェクトに関してユニーク性を保っており，オブジェクトの登録とともに値を増やす．言い換えると，データベース内に登録されているオブジェクトを EOID でソートすると，完全にこれまでのデータベース上の変更過程を再現することが可能である．

2.3　2 つの時間軸

EDB を扱う過程で我々は 2 種の時間軸に遭遇する．一つは現実世界での時間の経過であり，もう一つは EDB での時間の経過である．これは一般的なデータベースにも共通する事柄であるが，EDB では情報の登録や更新年月日を陽にユーザに提示しているため，しばしばこれら 2 つの時間軸は混同されやすい．

もし貴方が EDB の登録情報を用いて何らかのデータ処理を行おうとするならば，情報の時間軸の上で分類する目的で，オブジェクトの登録年月日 (mtime) を用いるべきではない．

2.4　情報の表現

EDB では原典情報を XML の規格に沿って記述している．記述の内容は，情報が載るテーブルの定義による．テーブルの定義もまた XML によって表現されている．

2.5　XMLDB と RDB

EDB では原典情報である XML 表現の登録内容をテーブル XMLDB に蓄積する．これとは別に，XML 表現の登録内容を関係データベース (Relational Database; RDB) に展開したテーブルを保持している．

RDB に保持されている内容はあくまで XMLDB の複写・加工物であり，各々の情報の正確な登録内容を得るためには，XMLDB から情報を取り出すべきである．

2.6　テーブル，カラムの XML 名と SQL 名，名称

EDB におけるテーブルの定義では，テーブルやカラムに XML 表現における名前とは別に，SQL における名前，および人間にわかりやすい名称を定義している．これは主にデータベースのコアに利用する DBMS の制約に依っている．

SQL における名前は，DBMS にアクセスする必要が生じたときのみに利用し，それ以外の部分では利用するべきではない．DBMS の制約の変更等により，SQL における名前は簡単に変更されてしまうからである．また，人間用に用意された名前はデータベース利用者が見るためのものであり，利用者の意見や要求によって簡単に変更されてしまうだろう．

これに対して，XML 名は EDB の上でテーブルを定義する場合に追加されることはあっても，従来利用していた名前が変更されることは (よほどの理由がない限り) ない．

EDB プログラマは，カラムの指定をハードコードする必要性やカラム指定を何らかの要求により行う場合には，必ず XML における名称を利用するべきである．XML から SQL 名への変換は面倒であるが，XML 名で指定することが作成したプログラムの長寿を保証する．

第3章　EDBにおけるデータ表現

3.1　名前空間

他の XML との混在の利用可能性を損なわないように，名前空間「edb」を用いる．

3.2　情報の記述

EDB における情報の記述は，

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE edb:table-name SYSTEM "http://web.db.tokushima-u.ac.jp/dtds/table-name.dtd">
<edb:table-name xmlns:edb="http://web.db.tokushima-u.ac.jp/dtds/">
   <edb:base base-attribute1 base-attribute2 base-attribute3 .../>
   <edb:table-name.element-name1 element-attribute1>
      element-name1 の登録内容（テキスト）
   </edb:table-name.element-name1>
   <edb:table-name.element-name2 element-attribute2>
      element-name2 の登録内容（テキスト）
   </edb:table-name.element-name2>
   <edb:table-name.element-name3 element-attribute3>
      element-name3 の登録内容（テキスト）
      <edb:table-name.element-name3.element-name3,1 element-attribute3,1>
         element-name3,1 の登録内容（テキスト）
      </edb:table-name.element-name3 element-name3,1>
   </edb:table-name.element-name3>
   ...
</edb:table-name>
```

のようになる．

3.3　ベース要素

BASE 要素は，各登録情報に必ず一つ (のみ) 記述される，属性のみの情報である．

```
<edb:base
   eid="eid" …情報識別子:  EID
   eoid="eoid" …オブジェクト識別子:  EOID
   mapto="mapto-eid" …マップ先の情報の EID
   mtime="modified-time" …登録時刻 (UNIX time)
   operator="operator-eid" …登録者の EID
   avail="availabled" …有効 (true)/無効 (false)
   date.from="aval-from-date" …有効期間 (はじまり)
   date.to="aval-to-date" …有効期間 (おわり)
   censor="censor-level" …検閲レベル
   owner="owner-eid" …所有者の EID
   read="read-permission" …読み権限レベル
   write="write-permission" …書き権限レベル
   delete="delete-permission" …削除権限レベル
/>
```

date.from, **date.to** は テーブルの属性として chronological が指定されたときのみ有効となる．

3.4　基本要素

本データベースでは，蓄積情報を全て XML により表現している．基本要素は，

```
<edb:element-name mapto="eid" date.from="avail-from-date" date.to="avail-to-date" read="read-permission" assert="assert" >
   <edb:english>english-value</edb:english>
   <edb:japanese>japanese-value</edb:japanese>
   <edb:pronounce>pronounce-value</edb:pronounce>
</edb:element-name>
```

である．ただし，「*element-name*」は XML の要素名，「*eid*」は参照識別子であり，「*avail-from-date*」，「*avail-to-date*」は有効期間，「*read-permission*」は公開範囲を示す権限レベルである．「*english-value*」，「*japanese-value*」，「*pronounce-value*」はそれぞれ英語，日本語，日本語の読みに対応する．

- **mapto** と **edb:english**, **edb:japanese**, **edb:pronounce**
  - 上記の表現において「*eid*」に有効な値が指定されたとき，「*english-value*」，「*japanese-value*」，「*pronounce-value*」は無視される．
  - 型 NAME では，「*english-value*」，「*japanese-value*」，「*pronounce-value*」の全てを利用する．
  - 型 SENTENCE, PARAGRAPH では，「*english-value*」，「*japanese-value*」，を利用する．
  - 他の型においては，「*english-value*」，のみを利用する．
  - 「*english-value*」，「*japanese-value*」，「*pronounce-value*」は改行文字を含まないテキストであり，「<」「>」「”」「&」「’」はそれぞれ「<」「>」「"」「&」「'」のようにエンコードされていなければならない．
- **date.from** と **date.to**
  - **date.from**, **date.to** はテーブルの定義におけるカラムの属性として chronological が指定されたときのみ有効となる．
  - 「*avail-from-date*」「*avail-to-date*」で，要素の記述内容の有効期間を指定する．この有効期間とは，データベースでの読み書きの有効期間を示すものではなく，登録要素記述が現実世界で有効である期間を示す．
  - 「*avail-from-date*」「*avail-to-date*」はともに省略可能で，両方とも省略された場合には，上位要素の値が代用される．上位要素を辿った結果，値が定まらない場合，それぞれの既定値は，
    - 「*avail-from-date*」：西暦 0 年 0 月 0 日
    - 「*avail-to-date*」：西暦 9999 年 99 月 99 日

    とする．
- **read**

  「*read-permission*」に指定できるものは，
  - inherit
  - public
  - university
  - user
  - personnel

- teacher
- limited
- owner
- staff

である．「*read-permission*」として”inherit”が指定された時には省略されことがある．また，「*read-permission*」が省略された時の既定値は”inherit”である．

「*read-permission*」が”inherit”の場合，上位要素，情報全体，カラム，テーブルの順番で属性を継承する．

- **assert**

  「*assert*」に指定できるものは，

  - nothing

  である．「*assert*」として”nothing”が指定された時には，登録対象なしを明示したとみなす．

3.5　要素の排列

XML では要素を任意に排列することが可能であるが，本データベースにおいては同名要素は全て連続して排列される．すなわち，

<edb:$name_1$>...</edb:$name_1$>
<edb:$name_1$>...</edb:$name_1$>
<edb:$name_1$>...</edb:$name_1$>
<edb:$name_1$>...</edb:$name_1$>
<edb:$name_2$>...</edb:$name_2$>
<edb:$name_2$>...</edb:$name_2$>
<edb:$name_2$>...</edb:$name_2$>
<edb:$name_3$>...</edb:$name_3$>
<edb:$name_3$>...</edb:$name_3$>

のように並べられる．ただし，同名要素の順序は保存される．

これは異名要素間の排列情報を用いないということであり，データベース登録インタフェースはデータ登録時に要素名のソーティングを行なう．したがって，利用者がこの排列を意識してデータを作成する必要はない．

例外として，属性 **chronological** が指定されている要素は，年代順に並べ替えることがある (属性の要素数が 1 に指定されている場合など)．

3.6　要素名の階層化

本データベースにおいて登録要素の階層化が生じる時には，要素名を下記のように階層化する．

<edb:$name_1$>
<edb:$name_1$. $name_2$>
<edb:$name_1$. $name_2$. $name_3$>
</edb:$name_1$. $name_2$. $name_3$>
</edb:$name_1$. $name_2$>
</edb:$name_1$>

階層化された要素名の区切り文字は「.」である．(各階層名には「.」は含まれない)

3.7　期間限定属性 (chronological)

3.7.1　まえがき

属性 chronological は，時間系列上で現実世界において

- 有効とするのが望ましい ⇒ 無効とするのが望ましい
- 無効とするのが望ましい ⇒ 有効とするのが望ましい

となる情報について，有効とするのが望ましい期間をデータベースシステムで統一的な手法により扱うために導入する．

これはデータベース上での情報の自動的な無効化を実現するものではない．すなわち，この期間限定属性を情報に付加したとしても，情報自体は**常に有効**な状態でデータベース中に存在する．期間限定属性はデータベースに対して発行するクエリにその条件が含まれている場合のみ利用される．

3.7.2　期間限定したい情報の例

- 組織の改組

  大学などの組織においては，不定期に学科や学部の改組が行われている．

  組織の改組にともない，改組前の【組織】情報を破棄し，改組後の【組織】情報をデータベースに登録すれば何ら問題はないように思われるかも知れないが，本データベースでは，他の情報から【組織】情報を参照 (ex. 【著作】.[組織] で著作が属する組織を登録する) するため，古い【組織】情報を削除することが出来ない．

削除されない【組織】情報は已然としてデータベースの中で有効な状態で存在するため，例えば大学の組織の構成チャートを作成するときなどに問題が発生する．

この問題を回避するために，情報自体に有効期間を定義し，情報として抽出する場合に有効であるか否かを判断できる様にしておく必要がある．

- 委員の交替

  本データベースでは，委員会委員などの定期的に個人が交替する情報を【擬人】情報として表現し，蓄積している．

  現在，この情報には同時に一人の【個人】情報しか登録できないため，新しい個人を委員として登録すると，旧委員の内容は失われてしまう．もちろん，旧委員が誰であったかという記録は，情報の変更履歴に含まれていることになるが，変更履歴は単に編集の履歴を示すものであり，後登録の修正作業をも含まれるので，信頼性がない．また，いつからいつまでという期間の情報も含まれていない．

  これに対処するためには，委員の登録内容に付加情報として期間を列記出来るようにすることが必要である．

3.7.3　基本方針

- 出来るだけ体系だてて

  期間限定が上記に掲げたもののみに該当する事柄であれば，【組織】【擬人】の情報もしくはそれらの中の項目のみを特別扱いし，例外的な処理をシステムに組み込めばよい．

  しかし，この種の期間限定の属性は，データベース中のほとんどの項目に必要性が生じる可能性がある．それらについてその都度特別なコードを記述し，例外処理を行うことは繁雑な作業をうみ，強いてはシステムの安定運用性，実現性に大きく関わってくる事柄となる．

  結局のところ，可能な限り共通のコードを利用して上記の要件を解決する方法を考えておきたい．(つまり，楽をしたいということである)

- 期間限定を行う箇所を指定する

  全ての情報および全ての項目において，期間限定を可能とするのではなく，テーブルの定義もしくはテーブル中のカラムの定義において，期間限定が可能なテーブル，カラムを指定する．指定は，テーブルやカラムの属性 (`<edb:attribute>...</edb:attribute>`) に期間限定 (**chronological**) を指定することによる．すなわち，

  – `<edb:attribute>`chronological`</edb:attribute>`

  と記述する．

- 期間限定に対する EDB 最下層 (libedb) の処理

  期間限定を行った情報について EDB の最下層 (libedb) では，

  – 期間限定された情報抽出要求が発行された場合のみ，それに合致する情報を抽出する．
  – 権限が委譲される参照登録のカラムについては，現時刻に一致する要素のみが権限委譲の対象となる．
  – 見出しに利用されるカラムについては，現時刻に一致する要素のみが見出しに利用される．

  である．

  すなわち，libedb では期間限定が行われている情報の解釈について特定の指針は与えず，期間限定を行った情報やカラムをどのように扱うかは上位層に委ねる．

  要素数 N として定義されているカラムに期間限定属性 (chronological) が指定された場合，

  – 要素数を任意数として扱う．
  – 各要素に登録されている期間が N 以上重複する場合にはエラーとする．
  – 各要素を年代順に排列する．

  を行う．

3.7.4　期間限定属性の XML での表現

- 期間限定の表現

  期間限定の情報を XML で表現するために，これらを XML の要素 (ELEMENT) ではなく，属性 (ATTRIBUTE) として記述する．属性名および指定方法は

  – date.from=”$avail-from-date$”
  – date.to=”$avail-to-date$”

  である．『$avail-from-date$』，『$avail-to-date$』は年月日 (YYYYMMDD) で指定し，0〜99999999 の整数値とする．実際に存在しない年月日を指定しても構わない．『$avail-from-date$』の既定値は 0，『$avail-to-date$』の既定値は 99999999 であり，それぞれの要素が省略された場合に用いられる．

  ある一つの情報の全体を期間限定する場合には，その情報の base 要素に

  – `<edb:base ... date.from="`$avail-from-date$`" date.to="`$avail-to-date$`" ... />`

と記述する．ただし，テーブル定義の属性に chronological が指定されていない場合には無視される．

ある一つの要素を期間限定する場合には，その要素の属性に

– `<edb:element ... date.from="`$avail-from-date$`" date.to="`$avail-to-date$`" ... >`

と記述する．ただし，カラム定義の属性に chronological が指定されていない場合には無視される．

- 期間判定

ある日時 ($date$) が指定されたとき，期間に合致するかどうかは

$$avail-from-date \leq date \quad \text{AND} \quad date \leq avail-to-date$$

で判断される．等号が含まれて**いる**ことに注意．

また，期間 ($from \sim to$) が指定されたとき，期間の重複しないかどうかは，

$$avail-to-date < from \quad \text{OR} \quad to < avail-from-date_1$$

で判断される．等号が含まれて**いない**ことに注意．

- 期間限定属性の継承

階層化されたカラムにおいて期間限定の情報は継承される．継承は，

– 情報 ⇒ 上位の要素 ⇒ 下位の要素

の方向で行われ，下位の要素に期間限定属性の指定がない場合にも有効である．

これは，期間限定できない要素においても，暗黙のうちに期間限定される可能性を示している．実際の XML の記述内容に関する処理においては，全ての要素に期間限定の可能性があると想定しておく必要がある．

3.7.5　関係データベース (RDB) への反映

- 期間限定の表現

期間限定の情報は，全て SQL レベルにおいて用意されているテーブルのベースカラム (SQL: edb_base) に反映される．反映されるカラムは，

– **date_from**: date.from 属性の値．

- **date_to**: date.to 属性の値.

である．既定値はなく，NULL になることもない．XML 上で date.from，date.to が指定されていない場合には，date_from, date_to はそれぞれ 0, 99999999 の値が記録される．

- 情報および要素の期間限定情報の記録場所

  情報自体が期間限定されている場合には，最上位のテーブル

  - *SqlTableName*.date_from
  - *SqlTableName*.date_to

  にその情報が記録される．libedb では *SqlTableName* は (EdbTableInfo)→sqlname で参照できる．要素毎の期間限定情報は

  - *SqlTableName*_*SqlColumnName*.date_from
  - *SqlTableName*_*SqlColumnName*.date_to

  に記録される．libedb では *SqlTableName*_*SqlColumnName* は (EdbColumnInfo)→reldb_sqlname で参照できる．要素が階層化されている場合には，最下層の期間限定属性は

  - $SqlTableName\_SqlColumnName_1\_SqlColumnName_2$.date_from
  - $SqlTableName\_SqlColumnName_1\_SqlColumnName_2$.date_to

  に記録される．libedb では $SqlTableName\_SqlColumnName_1\_SqlColumnName_2$ は (EdbColumnInfo)→reldb_sqlname で参照できる．

  これらは全て，期間限定属性を継承済みの期間である．

3.7.6　どのテーブル，カラムについて期間限定を指定するか?

期間限定を指定する際にどの情報やどのカラムに指定するのがよいのか，また，悪いのか?

- 短見出し (caption=1) に利用されるカラムには指定しない方がよい.

  EDB では参照登録すると，自動的に参照先の情報の短い見出しを作成し，冗長な情報としてカラムのテキストに代入する．この代入は，RDB 上で行われるとともに，補間 XML(cXML) にも反映される．期間限定が行われた場合，どの要素を見出しとするかは予想がしがたい.

  組織の改組が名称の変更のみであったとしても，【組織】.[名称] を期間限定に指定するのではなく，その場合にはマップを用いた従来の情報の同一性を利用し，別の【組織】情報として登録しておくのがよい．この場合，

  - 改組前の【組織】情報の期間指定: date.from=”xxxx” date.to=”改組日の前日”
  - 改組後の【組織】情報の期間指定: date.from=”改組日”

  とする.

- 年限が確定している様な情報，または，年次毎に内容が変更されるような情報
- 【擬人】情報の期間限定属性定義例 (情報: 徳島大学工学部長)
  - テーブルの定義
  - 登録情報の XML 表現
  - 登録情報の表示

3.8　DTD

各情報の登録様式に対応する DTD は http://web.db.tokushima-u.ac.jp/dtds/にある.

3.9　テーブルの XML の意味

本データベースでは，登録情報の様式も XML で表現している．情報の表現のための DTD は

http://web.db.tokushima-u.ac.jp/dtds/table.dtd

であり，各情報の登録様式を定義している XML は

http://web.db.tokushima-u.ac.jp/dtds/

にある.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE edb:table SYSTEM "http://web.db.tokushima-u.ac.jp/dtds/table.dtd">
<edb:table xmlns:edb="http://web.db.tokushima-u.ac.jp/dtds/">
テーブルの定義.
<edb:base base-attributes-of-table/> …ベース情報. (出現回数= 1)
<edb:xmlname>xmlname-of-table</edb:xmlname> …テーブルの XML 名. (出現回数= 1)
```

```
<edb:sqlname>SQL-name-of-table</edb:sqlname> …テーブルの SQL 名．RDBMS(PostgreSQL) 中でのテーブル名．(出現回数= 1)
<edb:name> …テーブルの名称（人間用）．(出現回数= 1)
   <edb:english>english-name-of-table</edb:english>
   <edb:japanese>japanese-name-of-table</edb:japanese>
   <edb:pronounce>pronounce-of-japanese-name-of-table</edb:pronounce>
</edb:name>
<edb:description> …テーブルの簡単な注釈（人間用）．(出現回数= 1)
   <edb:english>english-description-of-table</edb:english>
   <edb:japanese>japanese-description-of-table</edb:japanese>
</edb:description>
<edb:explain> …テーブルの少し詳しい注釈（人間用）．(出現回数≦ 1)
   <edb:english>english-explain-of-table</edb:english>
   <edb:japanese>japanese-explain-of-table</edb:japanese>
</edb:explain>
<edb:permtuple> …テーブルに登録されたタプル（情報）のデフォルトのアクセス権限．(出現回数= 1)
   <edb:permread>読出権限<edb:/permread>
   <edb:permcreate>作成権限</edb:permcreate>
   <edb:permwrite>変更権限</edb:permwrite>
   <edb:permdelete>消去権限</edb:permdelete>
</edb:permtuple>
<edb:permcolumn> …テーブルに登録されたカラム（項目）のデフォルトのアクセス権限．(出現回数= 1)
   <edb:permread>読出権限<edb:/permread>
   <edb:permcreate>作成権限</edb:permcreate>
   <edb:permwrite>変更権限</edb:permwrite>
   <edb:permdelete>消去権限</edb:permdelete>
</edb:permcolumn>
<edb:attribute>attribute-of-table</edb:attribute> …テーブルの属性 (出現回数≧ 0)．
      現在,
         auxiliary … 補助テーブル
         hierarchical … 階層化される情報（【組織】情報）
         censorship … 検閲が必要な情報
         chronological … 有効期間を指定できる情報
```

```
        がある．
<edb:module>path-name-of-module</edb:module> …テーブルのモジュール（出現回数＝ 1）．
        テーブル依存の関数をインプリメントしたモジュールのパス名を記述する．
<edb:maplookup>{xmlname-of-mapto-table}</edb:maplookup>
    情報全体の参照機能を利用する場合に参照できる先のテーブル指定．（出現回数≧ 0）
<edb:column>
    カラム（項目）の登録情報．（出現回数≧ 1）
    <edb:xmlname>xmlname-of-column</edb:xmlname> …カラムの XML 名．（出現回数＝ 1）
    <edb:sqlname>SQL-name-of-column</edb:xmlname> …カラムの SQL 名．RDBMS(PostgreSQL) 中でのカラム名．（出現回数＝ 1）
    <edb:name> …カラムの名前（人間用）（出現回数＝ 1）
        <edb:english>english-name-of-column</edb:english>
        <edb:japanese>japanese-name-of-column</edb:japanese>
        <edb:pronounce>pronounce-of-japanese-name-of-column</edb:pronounce>
    </edb:name>
    <edb:description> …カラムの簡単な注釈（人間用）（出現回数＝ 1）
        <edb:english>english-description-of-column</edb:english>
        <edb:japanese>japanese-description-of-column</edb:japanese>
    </edb:description>
    <edb:explain> …カラムの少し詳しい注釈（人間用）（出現回数≦ 1）
        <edb:english>english-explain-of-column</edb:english>
        <edb:japanese>japanese-explain-of-column</edb:japanese>
    </edb:explain>
    <edb:type>type-of-column</edb:type> …カラムの型．（出現回数＝ 1）
    <edb:minimum>minimum-value</edb:minimum> 値の最小値．（出現回数＝ 1）
    <edb:maximum>maximum-value</edb:maximum> 値の最大値．（出現回数＝ 1）
    <edb:default>default-value</edb:default> 値の既定値．（出現回数＝ 1）
    <edb:cryptography>type-of-cryptography</edb:cryptography> 値の暗号化手法．（出現回数＝ 1）．type=PASSPHRASE のときのみ．
    <edb:elements>number-of-elements</edb:elements> …要素数．「*」は任意数を表す．（出現回数＝ 1）
    <edb:elasticity>elasticity-of-elements</edb:elasticity> …入力欄の数 (初期値+増分)．（出現回数≦ 1）
    <edb:permtuple> …情報のカラム単位のデフォルトのアクセス権．（出現回数＝ 1）
        <edb:permread>読出権限<edb:/permread>
        <edb:permcreate>作成権限</edb:permcreate>
```

<edb:permwrite>変更権限</edb:permwrite>
<edb:permdelete>消去権限</edb:permdelete>
</edb:permtuple>
<edb:attribute>attribute-of-column</edb:attribute> …カラムの属性．(出現回数≧ 0)
現在,
authorize … 権限を指定された情報に委譲する
exclusive … 排他的登録項目
hierarchyparent … 階層化されるた時の上位情報 (テーブルの属性として，hierarchical 指定が必要)
personname … 人名が記述される項目
student … 学生が登録される可能性のある項目
pagenumber … 頁が記述される項目
patentnumber … 特許番号が記述される項目
nothing … 「該当なし」が指定できる項目
CLASSIFY … 大分類の対象となる．
classify … 小分類の対象となる．
alias … そのカラムに複数登録されているテキストはある事柄の別称であることを示す．
chronological … 有効期間を指定できるカラム
conductor … 情報のカラムの扱いを指定する情報の記述．このカラムに登録された EID を持つ情報がカラムの取捨選択を指定する．
multilingualcaption … 見出しに多言語の表示が望まれる．
required … 必須項目．
preferred … 推奨項目．
optional … 任意項目．
unused … 不用項目．
obsoleted … 廃止された項目．
がある．
<edb:conductor>mode</edb:conductor>
…カラムの取捨選択のモード．"required", "preferred", "optional", "unused"が有効．
<edb:maplookup>{xmlname-of-mapto-table}</edb:maplookup>
…参照機能を利用する場合に参照できる先のテーブル指定．(出現回数≧ 0)
…{xmlname-of-mapto-table} の後に「CANDIDATE={ 条件式 }」のように候補の選択肢を提示するときの補助的な情報を定義することができる．
…{xmlname-of-mapto-table} の後に「CAPTION={$column_1$ $column_2$}」のように見出しカラムを指定することが可能．各カラムは空白で区切る．
<edb:caption>caption-level</edb:caption>

```
            …見出し指定．(0... なし，1... 短見出しに利用，2... 長見出しに利用) (出現回数≦ 1)
        <edb:prefix> …前置詞．(出現回数≦ 1)
            <edb:english>english-prefix-of-column</edb:english>
            <edb:japanese>japanese-prefix-of-column</edb:japanese>
        </edb:prefix>
        <edb:postfix> …後置詞．(出現回数≦ 1)
            <edb:english>english-postfix-of-column</edb:english>
            <edb:japanese>japanese-postfix-of-column</edb:japanese>
        </edb:postfix>
        <edb:order>order-level</edb:order> …排列指定．(0... 利用しない，他... 優先順位)．(出現回数＝ 1)
        <edb:size>size-of-text</edb:size> …入力欄の大きさ（登録情報の文字の概数)．(出現回数＝ 1)
            <edb:column> …階層化された子カラムの定義．(出現回数≧ 0)
                (カラムの定義の繰り返し)
            </edb:column>
        </edb:column>
    </edb:table>
```

権限には，

- inherit … 継承．
- public … 公開．
- university … 学内．
- user … EDB にログインできる利用者．
- personnel … 職員．
- teacher … 教官．
- limited … 情報の権限継承による制限．
- owner … 情報の所有者．
- staff … 管理者．

が指定可能.

XML 名の省略記法

EDB の各所においては，長い XML 名を省略する目的で ‘@’ を利用することができる．‘@’ は現在最も近い場所にある XML 名またはその要素が持つ値を意味する．例えば，a という XML 名を利用したその内部では $@ = a$ となる．このとき，$@.b$ は $a.b$ となる．別の例では，$a.b$ の内部では $@ = a.b$ となる．このとき $@.c$ は $a.b.c$ となる．

‘@–’ は @ の親要素を表す．(‘-’ の個数だけ階層を遡る)

‘@1’ は @ の最上位の要素 (テーブルの XML 名) を表す.

最寄りの XML 名が $a.b.c.d$ のときの@記法の意味

`@`	$a.b.c.d$	現在の要素の XML 名またはその値
`@-`	$a.b.c$	現在の要素の親要素の XML 名またはその値
`@--`	$a.b$	現在の要素の親要素の親要素の XML 名またはその値
`@---`	a	現在の要素の親要素... 親要素の XML 名またはその値
`@----`	a	現在の要素の親要素... 親要素の XML 名またはその値
`@1`	a	現在の要素の属する最上位の要素の XML 名 (テーブルの XML 名) またはその値 (EID)
`@2`	$a.b$	現在の要素の階層の上にある要素 の XML 名またはその値
`@3`	$a.b.c$	現在の要素の階層の上にある要素 の XML 名またはその値
`@4`	$a.b.c.d$	現在の要素の階層の上にある要素 の XML 名またはその値
`@5`	$a.b.c.d$	現在の要素の階層の上にある要素 の XML 名またはその値

@による記法で，取り出されるのが XML 名か値かはそれが記述される場所による．例えば，条件式の左辺にて記述されれば XML 名に置き換わり，右辺で利用されれば値に置き換わる．

‘@’ の代わりに ‘@@’ を用いると外側にある XML 記述での現在最も近い XML 名を表す．外側に XML 記述がない場合にはエラーとなる．

値を取り出すときに@記法と XML 名を記述する場合とでは差異が生じることに注意して欲しい．@記法は現在の値を起点としてその上位を遡る記法であり，@記法で取り出される値は必ず 1 つである．一方，$@1.b$ のようにカラムの XML 名を記述した場合には，$@1.b$ に属する全ての値が取り出されることになる．これはそのカラムの要素数が複数の場合に違いが発生する．

ただし，以上はテーブルの定義内では現時点で利用できない.

特別なカラム名

- `REF`, `@.REF`：他の情報への参照リスト.
- `PERM`, `@.PERM`：権限を委譲している他の情報への参照リスト.
- `OWN`, `@.OWN`：所有者の EID.
- `MAP`, `@.MAP`：他の情報へのマップ.

第4章　EDBにおける日付の扱い

4.1　日付の範囲

EDB における日付の指定は，

西暦 0000 年 00 月 00 日 ～ 西暦 9999 年 99 月 99 日

の範囲とする．西暦年は必ず 4 桁で記述する．このうち，

- 西暦 0000 年 00 月 00 日…無限の過去を示す．
- 西暦 9999 年 99 月 99 日…無限の未来を示す．

は特別扱いとする．また，この範囲外の日付が指定された場合には，上記の範囲の最も近い日付に置き換えられる．

上述の規定は，日付属性の値 (DATE) に対して，ある日付 ($date$) 以前，以降という条件で検索を行う場合に

- $0 \leq$ DATE AND DATE $\leq date$
- $date \leq$ DATE AND DATE $\leq$ 99999999

のような検索式が記述される可能性を示している．

一方，期間を記述する場合に，終了日が未定の場合には，

- 西暦 YYYY 年 MM 月 DD 日 ～ 西暦 9999 年 99 月 99 日

と指定するべきであることを示唆している．

4.2　存在しない日付の扱い

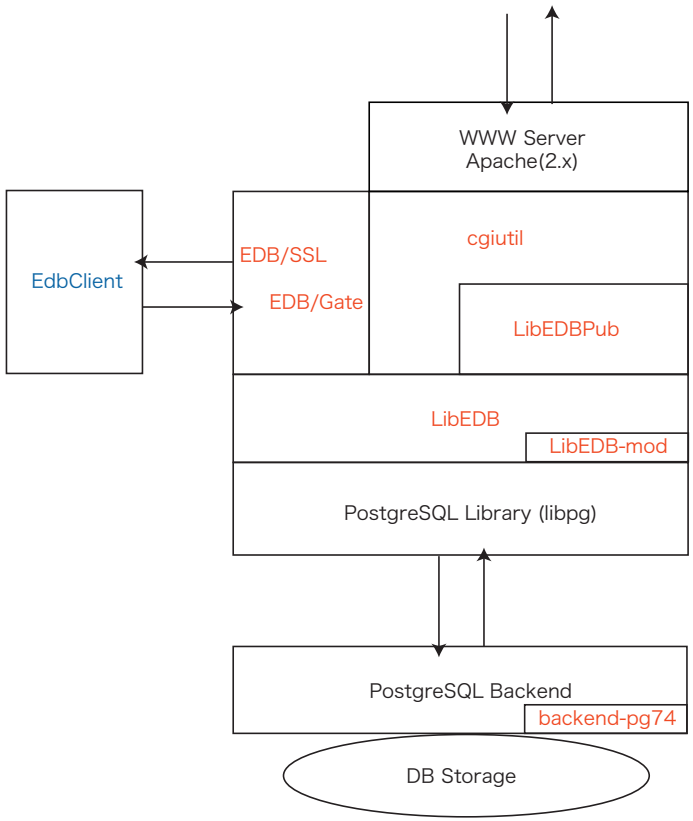
何らかの目的により，$[date_1,\ date_2]$ で指定された期間を正確な期日に変換する場合には，$[date_1,\ date_2]$ の期間内における最小の実存の日付と最大の実存の日付に変換されるべきものとして扱う．

date	mean	$date_1$	$date_2$
YYYY 年 0 月 0 日	*YYYY* 年の最初を表す.	*YYYY* 年 1 月 1 日	*YYYY*-1 年 12 月 31 日
YYYY 年 99 月 99 日	*YYYY* 年の最後を表す.	*YYYY*+1 年 1 月 1 日	*YYYY* 年 12 月 31 日
YYYY 年 *MM* 月 0 日	*YYYY* 年 *MM* 月の最初を表す.	*YYYY* 年 *MM* 月 1 日	*YYYY* 年 *MM*-1 月 28,29,30 or 31 日
YYYY 年 *MM* 月 99 日	*YYYY* 年 *MM* 月の最後を表す.	*YYYY* 年 *MM*+1 月 1 日	*YYYY* 年 *MM* 月 28,29,30 or 31 日

混乱を避けるために，99 月 99 日，99 日は $date_1$ に指定すべきではない．同様に，0 月 0 日，0 日は $date_2$ に指定すべきではない．

第5章　ソフトウェア構成

下図に EDB のソフトウェア構成を示す．赤字のものが EDB サーバにおいて利用されているソフトウェア (ライブラリ/アプリケーション) であり，青字はクライアント PC において利用されるソフトウェアである．



backend-pg74... PostgreSQL バックエンドプログラムに追加した EDB のモジュール．PostgreSQL にデータプリミティブタイプとして多言語プリミティブを追加．

LibEDB... PostgreSQL ライブラリアクセスライブラリ.

公開用ライブラリとして LibEDBR がある.

LibEDB-mod... テーブル依存のデータ処理モジュール.

LibEDBPub... EDB の登録情報出力用ライブラリ. 文書スタイル (HTML, LaTeX, ...) にあわせた出力を生成する.

公開用ライブラリとして LibEDBPubR がある.

cgiutil... WWW 経由の CGI プログラム群.

EDB/Gate... WWW に依らないアクセスを可能にするデータベースアクセスフロントエンド.

EDB/SSL... EDB/Gate とクライアントソフトウェア (e.g. EdbClient) 間の通信を SSL でトンネル化するためのモジュール.

EdbClient... EDB/Gate と通信を行なう PC 上のアプリケーション.

第II部

ライブラリ

第6章　LibEDB

6.1　情報識別子

EDB では登録情報に EID (EDB Identifier) と呼ばれる情報識別子を割り振る．また，登録されるタプル (オブジェクト) 毎に EOID (EDB Object Identifier) を割り当てる．

これらの実体は非負の整数値であり，有効な EID, EOID は 10001 以上の値である．また，0 は対応するオブジェクト無しを意味し，1∼10000 は予約された値である．

EDB ライブラリでは，次の型により EID, EOID を取り扱う．

TYPE — 情報識別子，オブジェクト識別子　　eid_t, eoid_t

SYNOPSIS :

```
typedef int32_t     eid_t ;
typedef int32_t     eoid_t ;
```

DESCRIPTION :

EID は EDB における情報の識別子であり，EOID はオブジェクト識別子である．

6.2　リンク (link)

LIBEDB および LIBEDB を利用するライブラリでは，下記の双方向リスト構造 (EDB ではこれをリンクと呼んでいる) を多用している．

STRUCTURE — リンク構造体のポインタ　　Link

SYNOPSIS :

```
Link     l ;
```

DESCRIPTION :

リンク構造体は一次元の双方向参照リスト構造を構成する．

リンク構造体の定義は下記のとおり．

```
struct LINK {
    struct LINK *p;      /* previous */
    struct LINK *n;      /* next */
    void *d;      /* object */
    void (*f)();      /* free */
};
typedef struct LINK *Link;
```

リンクの終了は NULL で示す．

FUNCTION — リンクオブジェクトの生成/開放

link_atom_new
link_atom_free
link_atom_free2

SYNOPSIS :

Link
link_atom_new　(void *obj, void (*free)())

void
link_atom_free　(Link l)

void
link_atom_free2　(void *context, Link l)

DESCRIPTION :

link_atom_new はリンク構造体 Link(3) を作成しそのポインタを返す．引数 `obj` はオブジェクトのポインタ，`free` はそのオブジェクトを開放するための関数へのポインタである．オブジェクトを開放する必要がない場合には NULL を指定する．

link_atom_free および *link_atom_free2* はともにリンクオブジェクトを開放するための関数である．あらかじめリンク構造体に登録されているオブジェクトの開放関数を呼び出す．*link_atom_free* と *link_atom_free2* の違いは，*link_atom_free* はオブジェクトの開放時にシステムコール *free(2)* と同じく *free*(obj) の形式で関数を呼び出すのに対して，*link_atom_free2* では *free*(context, obj) の形式で開放のための関数を呼び出す．

link_atom_free および *link_atom_free2* はリンクを操作しているモジュール内で呼び出される関数であり，一般にユーザがこれを利用することは稀である．

FUNCTION — リンクの開放

link_free
link_free2

SYNOPSIS :

void
link_free　(Link l)

void
link_free2　(void *context, Link l)

DESCRIPTION :

link_free および *link_free2* はともにリンクを開放する関数である．

link_free と *link_free2* の違いは，*link_free* はオブジェクトの開放時に link_atom_free(4) を呼び出し，*link_free2* は link_atom_free2(4) を呼び出す．

link_free2 を利用する必要がある場合，すべてのリンクオブジェクトに関して link_atom_free2(4) で渡すコンテキスト `context` は共通にしておかないといけない．

Return Values :
なし.

Function — リンクの先頭/最期を得る　link_head link_tail

Synopsis :

```
Link
link_head    ( Link l )

Link
link_tail    ( Link l )
```

Description :
link_head はリンクの先頭を，*link_tail* はリンクの最期を得る.

Return Values :
リンクの先頭/最期のリンク構造体のポインタ.

Function — リンクの結合　link_append

Synopsis :

```
Link
link_append    ( Link link1, Link link2 )
```

Description :
リンクを結合する.
`link1`, `link2` はともに NULL であっても良い.
ほとんどの場合，戻り値は `link1` となるが，変更される可能性もあるので *link_append* の戻り値は必ず元のリンク変数に代入すること.

Return Values :
結合後のリンクポインタ

Function — リンクオブジェクトの移動　link_atom_move_top

Synopsis :

```
Link
link_atom_move_top    ( Link link, Link l )
```

Description :
指定したオブジェクトをリンクの先頭に移動させる.

Return Values :
移動後のリンクポインタ

Function — リンクの長さ　link_atoms

Synopsis :

```
int
link_atoms    ( Link link )
```

Description :

リンクオブジェクトの数，すなわちリンクの長さを数える．

引数 `link` が NULL の場合 (すなわち，リンクが存在しない場合) には 0 を返す．

Return Values :

リンクオブジェクトの数．

Function — リンクの並べ換え

link_reverse
link_sort

Synopsis :

```
Link
link_reverse    ( Link link )

Link
link_sort    ( Link link, int (*cmp)(), void * (*aux)(), void *context, void (*free)() )
```

Description :

link_reverse は現在の並びを逆順に並べ換える．

link_sort はリンクオブジェクトを引数 `cmp` で指定した比較関数の返す値により並べ換えを行う．この関数は，内部で *qsort(3)* を呼び出している．

並べ換えにあたり，補助的なオブジェクトを一時的に作成する必要がある場合には，引数 `aux`, `context` を指定する．また，これらのオブジェクトを開放する関数を引数 *free* で指定する．いずれも，NULL 値を指定可能である．

Return Values :

並べ換え後のリンクポインタ．

Function — リンクの n 番目のオブジェクト

link_get_obj
link_get_link

Synopsis :

```
void *
link_get_obj    ( Link link, int n )

Link
link_get_link    ( Link link, int n )
```

Description :

link_get_obj はリンクから n 番目のオブジェクトのポインタを得るための関数である．

link_get_link はリンクから n 番目のオブジェクトが登録されているリンク構造体ポインタを得るための関数である．

Macro — リンクのスキャン

ForLink

SYNOPSIS :

ForLink　(Link l, Link link)

DESCRIPTION :

引数 link をスキャンするためのマクロ．実体は，for 文．引数 l はループ変数となる．

EXAMPLE — リンクのスキャン

```
Link(3) l, link;
...
ForLink(12)(l, link) {
    char *data = l->d;

    ...
}
```

MACRO — リンクオブジェクトの追加

LinkAppend
LinkAppendTop

SYNOPSIS :

Link
LinkAppend　(Link link, Link l)

Link
LinkAppendTop　(Link l , Link link)

DESCRIPTION :

通常，*link_append* の戻り値は元のリンク変数に代入される．
LinkAppend は link と l を結合し，その結果を link で指定した変数に代入する．すなわち，l で指定したリンクが link の末尾に追加される．
LinkAppendTop は l と link を結合し，その結果を link で指定した変数に代入する．すなわち，l で指定したリンクが link の先頭に追加される．

リンク構造を用いて簡易のスタック構造を模擬することが出来る．以下は，そのために用意されているマクロである．

MACRO — スタック操作

LinkPush
LinkPop

SYNOPSIS :

void
LinkPush　(Link link, void *d)

void *
LinkPop　(Link link)

DESCRIPTION :

リンクをスタック (Last-in First-out; LIFO) として利用するためのマクロ．

リンクの先頭がスタックの先頭 (次に取り出されるオブジェクト) となる.
LinkPush でスタックにプッシュし，*LinkPop* でスタックからオブジェクトをポップする.
LinkPush および *LinkPop* はオブジェクトの開放を行わない (リンク構造体の開放は行う) ので，スタックに登録するオブジェクトの管理はすべて利用者が責任をもつこと.

6.3　二分木 (binary tree)

LibEDB および LibEDB を利用するライブラリにおいて高速にオブジェクトの検索を要するような局面では，下記の二分木を利用する.

STRUCTURE — 二分木構造体　BiTree

SYNOPSIS :

BiTree　bt ;

DESCRIPTION :

二分木オブジェクトの定義は以下のとおりである.

```
    /* Binary Tree */
typedef uint32_t BiTree_hint_t;
typedef void * BiTree_obj_t;

struct BITREE {
    struct BITREE *p, *n;      /* branch */
    int bal;      /* balance */
    BiTree_obj_t obj;      /* object */
    BiTree_hint_t hint;      /* hint */
};
typedef struct BITREE *BiTree;
```

二分木にはどのようなタイプのオブジェクトのポインタも登録できるが，登録するオブジェクトは何らかの方法で再現的に比較できる必要があり，かつ比較の結果同値になることは許されない.

比較のための関数は

FUNCTION — 二分木オブジェクトの比較　BiTree_cmp_t

SYNOPSIS :

typedef int
(*BiTree_cmp_t)　(BiTree_obj_t obj1, BiTree_obj_t obj2)

DESCRIPTION :

obj1<obj2 のとき負値.
obj1=obj2 のとき 0.
obj1>obj2 のとき正値.

RETURN VALUES :
比較の結果を示す負値，0，正値．

で呼び出される．

LIBEDB で実装している二分木では，より高速に検索を実行するために，オブジェクト比較関数を呼び出す前にあらかじめ指定された *hint* を用いて比較を行う．*hint* の実体は整数値であり，BiTree$_{(15)}$ のメンバとしてオブジェクト毎に登録される．*hint* はオブジェクト比較関数を呼び出すオーバヘッドを省略するためのものであるので，オブジェクト比較関数を用いる場合，その戻り値と矛盾してはいけない．

なお，オブジェクト比較関数に依らず *hint* のみで比較が完結する場合がある．その場合には，比較関数に NULL を指定して二分木を利用することが出来る．

hint は EDB で定義されている eid_t$_{(2)}$, eoid_t$_{(2)}$ を収容できる．

二分木についての関数群を以下に説明する．

FUNCTION — 二分木に登録されているオブジェクトの数　bitree_num

SYNOPSIS :
```
int
bitree_num    ( BiTree bt )
```
DESCRIPTION :
二分木に登録されているオブジェクトの数を返す．オブジェクトをスキャンしないので時間はかからない．

RETURN VALUES :
オブジェクト数．

FUNCTION — 二分木の深さ　bitree_depth

SYNOPSIS :
```
int
bitree_depth    ( BiTree bt )
```
DESCRIPTION :
デバッグ用に用意されている．

RETURN VALUES :
二分木の深さ．

FUNCTION — 二分木にオブジェクトを追加　bitree_add

SYNOPSIS :
```
BiTree
bitree_add    ( BiTree bt, BiTree_obj_t obj, BiTree_hint_t hint, BiTree_cmp_t cmp, int (*err)(), void *err_ctxt )
```

DESCRIPTION :

引数 bt で示される二分木にオブジェクト *obj* を追加する．hint はオブジェクトの比較のための hint である．hint による比較でオブジェクトの比較が解決しなかった場合，関数 *cmp*(obj, obj2) が呼び出される．

関数 *cmp*(obj, obj2) の呼び出しによってもオブジェクトの比較が解決しない，すなわち同値のオブジェクトが存在すると判断された場合には，オブジェクトの追加を断念し，関数 *err*(obj, object_in_bitree, err_ctxt) を呼び出す．

RETURN VALUES :

オブジェクトが追加された二分木のポインタ

FUNCTION — 二分木からオブジェクトを削除

bitree_remove

SYNOPSIS :

```
BiTree
bitree_remove    ( BiTree bt, BiTree_obj_t obj, BiTree_hint_t hint, BiTree_cmp_t cmp, BiTree_obj_t *robj )
```

DESCRIPTION :

引数 bt で示される二分木からオブジェクト *obj* を削除する．hint, cmp は，bitree_add(19) と同じ意味である．

*robj には，削除されたオブジェクトのポインタがセットされる．目的のオブジェクトが存在しない場合，*robj が NULL にセットされる．

RETURN VALUES :

オブジェクトが削除された二分木のポインタ

FUNCTION — 二分木の検索

bitree_search

SYNOPSIS :

```
BiTree_obj_t
bitree_search    ( BiTree bt, BiTree_obj_t obj, BiTree_hint_t hint, BiTree_cmp_t cmp )
```

DESCRIPTION :

RETURN VALUES :

見つかったオブジェクトのポインタ．

FUNCTION — 二分木の開放

bitree_free
bitree_free2

SYNOPSIS :

```
void
bitree_free    ( BiTree bt, void (*free)() )

void
bitree_free2    ( BiTree bt, void (*free2)(), void *ctxt )
```

DESCRIPTION :

二分木を開放する．

bitree_free はオブジェクトの開放に *free*(obj) を呼び出し，*bitree_free2* はオブジェクトの開放に *free2*(ctxt, obj) を呼び出す．

Return Values :

なし.

Function — 二分木からのコールバック

bitree_callback
bitree_callback_hint

Synopsis :

```
int
bitree_callback    ( BiTree bt, int (*cb)(), void *data )

int
bitree_callback_hint    ( BiTree bt, int (*cbh)(), void *data )
```

Description :

すべてのオブジェクトについて，*cb*(obj, data) または *cbh*(obj, hint, data) を呼び出す．呼び出す順番は，二分木内での排列順である．

cb(obj, data) および *cbh*(obj, hint, data) はコールバックを継続するか否かを示す論理値を返す．コールバックを途中で終了したい場合には偽値を返せば良い．

Return Values :

コールバックがすべてのオブジェクトについて行われたことを示す論理値.

Function — 文字列オブジェクトのヒント

bitree_hint_ucs4str
bitree_hint_ucs4strcase
bitree_hint_str
bitree_hint_strcase

Synopsis :

```
BiTree_hint_t
bitree_hint_ucs4str    ( ucs4_t *u )

BiTree_hint_t
bitree_hint_ucs4strcase    ( ucs4_t *u )

BiTree_hint_t
bitree_hint_str    ( char *s )

BiTree_hint_t
bitree_hint_strcase    ( char *s )
```

Description :

文字列を二分木に登録する際に利用できる Built-in のヒント生成関数.

Return Values :

生成したヒント.

Function — 文字列オブジェクトの比較関数

bitree_compare_str
bitree_compare_strcase
bitree_compare_ucs4str
bitree_compare_ucs4strcase

SYNOPSIS :

```
int
bitree_compare_str    ( BiTree_obj_t o1, BiTree_obj_t o2 )

int
bitree_compare_strcase    ( BiTree_obj_t o1, BiTree_obj_t o2 )

int
bitree_compare_ucs4str    ( BiTree_obj_t o1, BiTree_obj_t o2 )

int
bitree_compare_ucs4strcase    ( BiTree_obj_t o1, BiTree_obj_t o2 )
```

DESCRIPTION :

文字列を二分木に登録する際に利用できる Built-in の比較関数.

MACRO — 二分木にオブジェクトを追加　BiTreeAdd

SYNOPSIS :

```
void
BiTreeAdd    ( bt, o, h, c, e, d )
```

DESCRIPTION :

二分木にオブジェクトを追加するためのマクロである.
引数は bitree_add(19) と同じ.
bitree_add(19) の戻り値を自動的に `bt` に代入しているだけ.

6.4　BASE64

FUNCTION — BASE64 エンコード　base64_encode

SYNOPSIS :

```
unsigned char *
base64_encode    ( unsigned char *s, int slen )
```

DESCRIPTION :

引数で与えられた文字列 `s`，長さ `slen` の文字列を base64 でエンコードする．エンコードされた文字列は戻り値で返される．エンコード文字列は，'\0' でターミネートされている.
得られた文字列は，呼び出し側で開放 (*free(3)*) すること.

RETURN VALUES :

base64 エンコードされた文字列.

See Also :

base64_decode(28)

Function — base64 デコード　base64_decode

Synopsis :

```
unsigned char *
base64_decode    ( unsigned char *s, int *plen )
```

Description :

引数 s で与えられた base64 エンコード文字列をデコードする．デコードされた文字列のポインタは戻り値で，長さは*plen に返される．
得られた文字列は，呼び出し側で開放 (*free(3)*) すること．

Return Values :

base64 デコードされた文字列．

See Also :

base64_encode(27)

6.5　画像

Structure — 画像オブジェクト　EdbPicture

Synopsis :

```
EdbPicture    picture ;
```

Description :

構造体の定義は下記のとおり．

```
struct EDB_PICTURE {
    int xsize;      /* 画像の横の画素数 */
    int ysize;      /* 画像の縦の画素数 */
    char *mimetype;      /* MIME タイプ */
    char *body;      /* 画像データ */
    int bytes;      /* 画像データのバイト数 */
    int dpi;      /* 画像のデフォルトの解像度 */
    double height;      /* 画像の高さ（文字の高さ=1）*/
    double depth;      /* 画像の深さ（文字の高さ=1）*/
};
typedef struct EDB_PICTURE *EdbPicture;
```

EDB における画像の取り扱いは，LaTeX 処理系における画像の取り扱いに類似している．すなわち，

- 画像を文字の一つとして扱う.
- 画像を浮遊するオブジェクトの一つとして扱う.

の両方をサポートする.

画像を文字の一つとして扱う場合には，画像構造体のメンバ `height` および `depth` を用いる．それぞれ，前後の文字の標準の高さを 1 としたときの，画像オブジェクトの高さ (ベースラインより上の部分) と深さ (ベースラインよりも下の部分) である．これらのサイズは，画像を実際に表示する部分で上書きされることがあり，画像オブジェクトに登録されるこれらの値は既定値である.

画像が浮遊して存在する場合には，メンバ `dpi` を用いる．これも表示部分において上書きされる可能性があり，既定値である.

画像オブジェクトを操作するために用意されている関数には以下がある.

Function — サポート画像

edb_picture_body_is_supported

Synopsis :

```
int
edb_picture_body_is_supported  ( EdbContext ec, unsigned char *body, int bytes )
```

Description :

画像データのポインタ `body` およびデータの長さ `bytes` から，サポートしている画像かどうかを判定する.

Return Values :

サポートを示す論理値.

Function — 画像情報の MIME タイプ

edb_picture_mimetype

Synopsis :

```
char *
edb_picture_mimetype  ( EdbContext ec, EdbTupleInfo tuple )
```

Description :

【画像】に登録されているタプル情報がもつ画像データの MIME タイプを得る.

Return Values :

画像の MIME タイプ．静的文字列であり，開放してはいけない.

Function — 画像オブジェクトの作成/開放

edb_picture_new
edb_picture_free

Synopsis :

```
EdbPicture
edb_picture_new  ( EdbContext ec, EdbTupleInfo tuple )

void
edb_picture_free  ( EdbContext ec, EdbPicture picture )
```

DESCRIPTION :

edb_picture_new は画像タプルから，画像オブジェクトを作成する．

edb_picture_new で作成した画像オブジェクトは *edb_picture_free* で開放する．

RETURN VALUES :

edb_picture_new は画像オブジェクトのポインタを返す．*edb_picture_free* の戻り値はなし．

FUNCTION — 画像オブジェクトの属性変更

edb_picture_set_attribute

SYNOPSIS :

```
int
edb_picture_set_attribute    ( EdbContext ec, EdbPicture picture, char *attribute )
```

DESCRIPTION :

画像オブジェクトの属性を変更する．引数 `attribute` には，文字列で

- "dpi=300"
- "height=2.0"
- "depth=1.0"

のように指定する．

RETURN VALUES :

成功を示す論理値．

FUNCTION — 画像オブジェクトのフォーマット変換

edb_picture_write_JPEG
edb_picture_write_PNG
edb_picture_write_EPS

SYNOPSIS :

```
int
edb_picture_write_JPEG    ( EdbContext ec, EdbPicture picture, FILE *fp )

int
edb_picture_write_PNG    ( EdbContext ec, EdbPicture picture, FILE *fp )

int
edb_picture_write_EPS    ( EdbContext ec, EdbPicture picture, FILE *fp )
```

DESCRIPTION :

画像オブジェクトのフォーマットを変換してファイルに書き出す．引数 *fp* はライトオープンされたファイル構造体のポインタ．

フォーマット変換には，ports/graphics/netpbm パッケージを利用しているため，ports/graphics/netpbm パッケージがない場合にはエラーとなる．

RETURN VALUES :

成功を示す論理値．

6.6　コンテキスト

STRUCTURE — EDB コンテキストのポインタ　EdbContext

SYNOPSIS :

```
EdbContext    ec ;
```

DESCRIPTION :

EDB 構造体の定義は下記のように宣言されている.

```
struct EDB_CONTEXT { ... };
typedef struct EDB_CONTEXT *EdbContext;
```

6.7　構造体

6.7.1　XMLDB 構造体

XMLDB のテーブル名，カラム名の定義.

```
#define EDB_XMLDB_TABLE
#define EDB_XMLDB_COLUMN_EID
#define EDB_XMLDB_COLUMN_EOID
#define EDB_XMLDB_COLUMN_TEID
#define EDB_XMLDB_COLUMN_XML
```

STRUCTURE — XMLDB 構造体　EdbXmldbInfo

SYNOPSIS :

```
EdbXmldbInfo    xi ;
```

DESCRIPTION :

```
struct EDB_XMLDB_INFO {
    eid_t eid;
    eoid_t eoid;
    eid_t teid;
    utf8_t *xml;
};
typedef struct EDB_XMLDB_INFO *EdbXmldbInfo;
```

MACRO — XMLDB 構造体メンバーマクロ　XmldbEID　XmldbEOID　XmldbTEID　XmldbXML

Synopsis :

```
eid_t
XmldbEID    ( EdbXmldbInfo xi )

eoid_t
XmldbEOID    ( EdbXmldbInfo xi )

eid_t
XmldbTEID    ( EdbXmldbInfo xi )

utf8_t *
XmldbXML    ( EdbXmldbInfo xi )
```

Description :

Function — EID, EOID の最大値

edb_xmldb_eid_max
edb_xmldb_eoid_max

Synopsis :

```
eid_t
edb_xmldb_eid_max    ( EdbContext ec )

eoid_t
edb_xmldb_eoid_max    ( EdbContext ec )
```

Description :

接続しているデータベースのテーブル EDB_XMLDB_TABLE (edb_xmldb) において，EID, EOID の最大値を得る．
この最大値を用いて新たに情報を登録する場合には，必ず edb_begin(75) でデータベースをロックしてから行うこと．

Return Values :

EID, EOID の最大値．

See Also :

edb_open(74), edb_begin(75)

Function — XMLDB 構造体の作成/開放

edb_xmldb_new
edb_xmldb_free

Synopsis :

```
EdbXmldbInfo
edb_xmldb_new    ( EdbContext ec )

void
edb_xmldb_free    ( EdbContext ec, EdbXmldbInfo xi )
```

Description :

Function — XMLDB 構造体の読み出しと書き込み

edb_xmldb_get
edb_xmldb_put

SYNOPSIS :

EdbXmldbInfo
edb_xmldb_get　(EdbContext ec, eid_t eid, eoid_t eoid)

eoid_t
edb_xmldb_put　(EdbContext ec, EdbXmldbInfo xi)

DESCRIPTION :

FUNCTION — XMLDB 構造体の読み出し　edb_xmldb_get_byTEID

SYNOPSIS :

Link
edb_xmldb_get_byTEID　(EdbContext ec, eid_t teid, int all)

DESCRIPTION :

`teid` で示されるテーブルに登録されている情報を読み出す.
`all` が真の場合すべての履歴情報を読み出し，偽の場合には最新の情報のみを読み出す.

RETURN VALUES :

EdbXmldbInfo 構造体のポインタが登録されたリンクのポインタ.

6.7.2　BASE 構造体

ベースに対応する SQL カラム名の定義.

```
#define EDB_BASE_EID
#define EDB_BASE_EOID
#define EDB_BASE_MAPTO_EID
#define EDB_BASE_PARENT_EID
#define EDB_BASE_MTIME
#define EDB_BASE_AVAILABLE
#define EDB_BASE_DATE_FROM
#define EDB_BASE_DATE_TO
#define EDB_BASE_CENSOR
#define EDB_BASE_OPERATOR
#define EDB_BASE_OWNER
#define EDB_BASE_RDL
#define EDB_BASE_WRL
#define EDB_BASE_DEL
```

Structure — BASE 構造体

EdbBaseInfo

Synopsis :

EdbBaseInfo　bi ;

Description :

ベース構造体は以下のように定義されている.

```
struct EDB_BASE_INFO {
    eid_t eid;      /* EID */
    eid_t eoid;     /* EOID */
    eid_t mapto;    /* mapto EID */
    eid_t parent;   /* parent EID */
    time_t mtime;   /* date_time */
    eid_t op;      /* operator */
    int censor;     /* censor */
    int av;      /* available */
    int date_from;      /* available date from */
    int date_to;     /* available date to */
    int owner;     /* owner */
    int rdl, wrl, del;      /* read/write/delete level */
};
typedef struct EDB_BASE_INFO *EdbBaseInfo;
```

ベース構造体へのアクセスは，以下のマクロを利用してコーディングを行う.

Macro — ベース構造体へのアクセス

BaseEID
BaseEOID
BaseMapto
BaseParent
BaseMtime
BaseOperator
BaseAvail
BaseDateFrom
BaseDateTo
BaseCensor
BaseOwner
BaseRead
BaseWrite
BaseDelete

Synopsis :

eid_t
BaseEID　(EdbBaseInfo bi)

eoid_t
BaseEOID　(EdbBaseInfo bi)

eid_t
BaseMapto　(EdbBaseInfo bi)

eid_t
BaseParent　(EdbBaseInfo bi)

time_t
BaseMtime　(EdbBaseInfo bi)

eid_t
BaseOperator　(EdbBaseInfo bi)

bool
BaseAvail　(EdbBaseInfo bi)

int
BaseDateFrom　(EdbBaseInfo bi)

int
BaseDateTo　(EdbBaseInfo bi)

int
BaseCensor　(EdbBaseInfo bi)

eid_t
BaseOwner　(EdbBaseInfo bi)

int
BaseRead　(EdbBaseInfo bi)

int
BaseWrite　(EdbBaseInfo bi)

int
BaseDelete　(EdbBaseInfo bi)

DESCRIPTION :

6.7.3　TABLE 構造体

STRUCTURE — TABLE 構造体　EdbTableInfo

SYNOPSIS :
EdbTableInfo　ti ;

DESCRIPTION :
テーブル構造体の定義は下記のように宣言されている.

```
struct EDB_TABLE_INFO { ... };
typedef struct EDB_TABLE_INFO *EdbTableInfo;
```

MACRO — テーブル構造体マクロ

SYNOPSIS :
EdbBaseInfo
TABLE_Base　(EdbTableInfo ti)

TABLE_Base
TABLE_EID
TABLE_EOID
TABLE_Mapto
TABLE_Parent
TABLE_Mtime
TABLE_Operator
TABLE_Avail
TABLE_Owner
TABLE_Read
TABLE_Write
TABLE_Delete

eid_t
TABLE_EID　(EdbTableInfo ti)

eoid_t
TABLE_EOID　(EdbTableInfo ti)

eid_t
TABLE_Mapto　(EdbTableInfo ti)

eid_t
TABLE_Parent　(EdbTableInfo ti)

time_t
TABLE_Mtime　(EdbTableInfo ti)

eid_t
TABLE_Operator　(EdbTableInfo ti)

bool
TABLE_Avail　(EdbTableInfo ti)

eid_t
TABLE_Owner　(EdbTableInfo ti)

int
TABLE_Read　(EdbTableInfo ti)

int
TABLE_Write　(EdbTableInfo ti)

int
TABLE_Delete　(EdbTableInfo ti)

int
TABLE_TupleCreate　(EdbTableInfo ti)

int
TABLE_TupleRead　(EdbTableInfo ti)

int
TABLE_TupleWrite　(EdbTableInfo ti)

int
TABLE_TupleDelete　(EdbTableInfo ti)

int
TABLE_ColumnCreate　(EdbTableInfo ti)

int
TABLE_ColumnRead　(EdbTableInfo ti)

int
TABLE_ColumnWrite　(EdbTableInfo ti)

int
TABLE_ColumnDelete　(EdbTableInfo ti)

DESCRIPTION :

MACRO — テーブル構造体マクロ

TABLE_XN
TABLE_SN

SYNOPSIS :

char *
TABLE_XN　(EdbTableInfo ti)

char *
TABLE_SN　(EdbTableInfo ti)

DESCRIPTION :

TABLE_XN はテーブルの XML 名を得る.
TABLE_SN はテーブルの SQL 名を得る.

STRUCTURE — カラム構造体

EdbColumnInfo

SYNOPSIS :

EdbColumnInfo　ci ;

DESCRIPTION :

```
struct EDB_COLUMN_INFO { ... };
typedef struct EDB_COLUMN_INFO *EdbColumnInfo;
```

MACRO — カラム構造体マクロ

ColumnXN
ColumnSN
ColumnSN_EID

SYNOPSIS :

char *
ColumnXN　(ci)

char *
ColumnSN　(ci)

char *
ColumnSN_EID　(ci)

DESCRIPTION :

ColumnXN はカラムの XML 名を得る.
ColumnSN はカラムの SQL 名を得る.
ColumnSN_EID はカラムの SQL 名 (EID) を得る.

FUNCTION — カラムの SQL における要素数

edb_table_column_reldb_elements

SYNOPSIS :

int
edb_table_column_reldb_elements　(EdbContext ec, EdbTableInfo ti, EdbColumnInfo super_ci, EdbColumnInfo ci)

DESCRIPTION :

テーブル `ti` に定義されているカラム `ci` の SQL における要素数を得る. `super_ci==NULL` のとき, テーブル全体におけるカラムの要素数を示し, `super_ci!=NULL` のとき, `super_ci` に対するカラムの要素数を示す.
SQL 構文を作成する場合, とくに条件式を作成する際には, 対象のカラムが単数か配列で定義されているかを調べる必要がある. でなければ, 条件式が構文エラーとなる.
EDB で作成されたテーブルは, *edb_table_column_reldb_elements* が返す値が 1 のときのみ単数で, それ以外は配列で定義されている.

RETURN VALUES :

SQL におけるカラムの要素数. 0 は複数 (任意数) を示す.

FUNCTION — カラムの XML における要素数

edb_table_column_xml_elements

SYNOPSIS :

int
edb_table_column_xml_elements　(EdbContext ec, EdbTableInfo ti, EdbColumnInfo super_ci, EdbColumnInfo ci)

DESCRIPTION :

テーブル `ti` に定義されているカラム `ci` の XML における要素数を得る. `super_ci==NULL` のとき, テーブル全体におけるカラムの要素数を示し, `super_ci!=NULL` のとき, `super_ci` に対するカラムの要素数を示す.

RETURN VALUES :

XML におけるカラムの要素数. 0 は複数 (任意数) を示す.

FUNCTION — カラムを探す

edb_table_seek_column
edb_table_seek_column_by_xmlname
edb_table_seek_column_by_sqlname

SYNOPSIS :

EdbColumnInfo
edb_table_seek_column　(EdbContext ec, EdbTableInfo ti, char *name)

EdbColumnInfo
edb_table_seek_column_by_xmlname　(EdbContext ec, EdbTableInfo ti, char *xn)

EdbColumnInfo
edb_table_seek_column_by_sqlname　(EdbContext ec, EdbTableInfo ti, char *sn)

DESCRIPTION :

テーブル構造体からカラム構造体のポインタを探す．

edb_table_seek_column_by_xmlname では，XML 名 xn を元にカラムを探す．

edb_table_seek_column_by_sqlname では，SQL 名 sn を元にカラムを探す．

RETURN VALUES :

カラム構造体のポインタ．見つからない場合には NULL を返す．

6.7.4　Tuple 構造体

STRUCTURE — TUPLE 構造体

EdbTupleInfo

SYNOPSIS :

```
EdbTupleInfo    tuple ;
```

DESCRIPTION :

タプル構造体の定義は下記のように宣言されている．

```
struct EDB_TUPLE_INFO { ... };
typedef struct EDB_TUPLE_INFO *EdbTupleInfo;
```

MACRO — タプル構造体マクロ

TupleBase
TupleEID
TupleEOID
TupleMapto
TupleParent
TupleMtime
TupleOperator
TupleAvail
TupleDateFrom
TupleDateTo
TupleOwner
TupleRead
TupleWrite
TupleDelete

SYNOPSIS :

```
EdbBaseInfo
TupleBase    ( EdbTupleInfo tuple )

eid_t
TupleEID    ( EdbTupleInfo tuple )

eoid_t
TupleEOID    ( EdbTupleInfo tuple )

eid_t
TupleMapto    ( EdbTupleInfo tuple )

eid_t
TupleParent    ( EdbTupleInfo tuple )

time_t
TupleMtime    ( EdbTupleInfo tuple )

eid_t
TupleOperator    ( EdbTupleInfo tuple )
```

bool
TupleAvail　(EdbTupleInfo tuple)

int
TupleDateFrom　(EdbTupleInfo tuple)

int
TupleDateTo　(EdbTupleInfo tuple)

eid_t
TupleOwner　(EdbTupleInfo tuple)

int
TupleRead　(EdbTupleInfo tuple)

int
TupleWrite　(EdbTupleInfo tuple)

int
TupleDelete　(EdbTupleInfo tuple)

DESCRIPTION :

MACRO — タプルの属しているテーブル　TupleTable

SYNOPSIS :

EdbTableInfo
TupleTable　(EdbTupleInfo tuple)

DESCRIPTION :

タプルが属しているテーブルの情報を得る.

6.7.5　TupleColumn 構造体

STRUCTURE — TUPLECOLUMN 構造体　EdbTupleColumn

SYNOPSIS :

EdbTupleColumn　tc ;

DESCRIPTION :

```
struct EDB_TUPLE_COLUMN {
    void *super;      /* カラムが属しているタプル (struct EDB_TUPLE_INFO *) またはデータ (struct EDB_DATUM *) */
    EdbColumnInfo ci;      /* カラムに対応するカラム情報 */
    EdbDatum datum;      /* データ */
```

```
    int mapped;      /* マップした結果作成された情報か (論理値) */
};
typedef struct EDB_TUPLE_COLUMN *EdbTupleColumn;
```

Macro — カラムのスキャン

ForTupleColumn

Synopsis :

ForTupleColumn　(Link l, EdbTupleInfo tuple)

Description :

タプルに登録されているカラムをスキャンする.

実体は，for 文.

Example — タプルに登録されているカラムをスキャンする

```
Link(3) l;
EdbTupleInfo(52) tuple;
...
ForTupleColumn(56)(l, tuple) {
    EdbTupleColumn(55) tc = l->d;

    ...
}
```

Function — カラムを探す

edb_tuple_seek_column
edb_tuple_seek_column_by_xmlname
edb_tuple_seek_column_by_sqlname
edb_tuple_seek_TC_by_xmlname

Synopsis :

EdbTupleColumn
edb_tuple_seek_column　(EdbContext ec, EdbTupleInfo tuple, char *name)

EdbTupleColumn
edb_tuple_seek_column_by_xmlname　(EdbContext ec, EdbTupleInfo tuple, char *xn)

EdbTupleColumn
edb_tuple_seek_column_by_sqlname　(EdbContext ec, EdbTupleInfo tuple, char *sn)

EdbTupleColumn
edb_tuple_seek_TC_by_xmlname　(EdbContext ec, EdbTupleInfo tuple, EdbTupleColumn tc, EdbDatum datum, char *xn)

Description :

タプル構造体からタプルカラム構造体のポインタを探す.

edb_tuple_seek_column_by_xmlname では，XML 名 xn を元にカラムを探す.

edb_tuple_seek_column_by_sqlname では，SQL 名 sn を元にカラムを探す．
edb_tuple_seek_TC_by_xmlname では，指定したカラム tc 以下について，XMLxn のカラムを探す．
xn には省略名 (例: "@.givenname") を利用することが出来る．
カラムの要素が登録されていない場合には NULL 値を返す可能性がある．ただし，要素が登録されていなくてもカラムのポインタが返る場合があるので，要素が登録されているかどうかは返されたカラムにおいて検査すること．

RETURN VALUES :
タプルカラム構造体のポインタ．見つからない場合には NULL を返す．

SEE ALSO :
EdbTupleColumn(55)

MACRO — TCDATUM マクロ

TCDatumFirst
TCDatumNext

SYNOPSIS :

```
EdbDatum
TCDatumFirst    ( EdbTupleColumn tc )

EdbDatum
TCDatumNext    ( EdbTupleColumn tc, EdbDatum datum )
```

DESCRIPTION :
TCDatumFirst は最初のデータ，*TCDatumNext* は次のデータを返す．
該当のデータがない場合には，NULL 値となる．

MACRO — TCDATUM マクロ

TCDatumIsFirst
TCDatumIsLast

SYNOPSIS :

```
bool
TCDatumIsFirst    ( EdbTupleColumn tc, EdbDatum datum )

bool
TCDatumIsLast    ( EdbTupleColumn tc, EdbDatum datum )
```

DESCRIPTION :
TCDatumIsFirst は先頭のデータであるかどうか，*TCDatumIsLast* は最期のデータであるかどうかの論理値を返す．

MACRO — TCDATUM のデータスキャンマクロ

ForTCDatum

SYNOPSIS :

```
ForTCDatum    ( EdbDatum datum, EdbTupleColumn tc )
```

DESCRIPTION :
引数 tc に登録されているデータをスキャンするためのマクロ．実体は，for 文．引数 datum はループ変数となり，かつ EdbDatum 構造体である．

Example — TCDatum のデータスキャンマクロ

```
EdbTupleColumn(55) tc;
EdbDatum(68) datum;
...
tc = ...  ;
ForTCDatum(60)(datum, tc) {

    ...
}
```

Function — データの開放

edb_tcdatum_free

Synopsis :

void
edb_tcdatum_free　(EdbContext ec, EdbTupleColumn tc)

Description :

登録されているデータを開放する.

Function — データの個数

edb_tcdatum_num

Synopsis :

int
edb_tcdatum_num　(EdbContext ec, EdbTupleColumn tc)

Description :

Return Values :

データの個数.

Function — n 番目のデータを得る.

edb_tcdatum_get_datum

Synopsis :

EdbDatum
edb_tcdatum_get_datum　(EdbContext ec, EdbTupleColumn tc, int nth)

Description :

登録されている nth 番目のデータを返す. データの複製は行われない. 最初のデータは nth が 0 でポイントされる.
範囲を超えた場合には NULL を返す.

Return Values :

n 番目の EdbDatum.

Function — データの逆順化

edb_tcdatum_reverse

Synopsis :

void
edb_tcdatum_reverse　(EdbContext ec, EdbTupleColumn tc)

Description :

登録データの並び順を逆順にする.

Function — データの追加

edb_tcdatum_append

Synopsis :

void
edb_tcdatum_append　(EdbContext ec, EdbTupleColumn tc, EdbDatum datum)

Description :

既に登録されているデータのリンクの最期に `datum` を追加する.

Function — タプルカラムデータの配列化

edb_tcdatum_to_array

Synopsis :

EdbDatum *
edb_tcdatum_to_array　(EdbContext ec, EdbTupleColumn tc, EdbDatum array[], int *num)

Description :

edb_tcdatum_to_array はリンクされているデータを配列に変換する．ただし，EdbDatum(68) のリンク構造は保たれており，またデータの複製も行わない.

`array` が NULL のときには，関数内で配列を用意し，その中に EdbDatum を格納する．最終的に配列は呼び出し側で開放 (*free*(array);) すること.

`array` が NULL 以外のときには，あたえられた配列に対して EdbDatum を格納する．配列の大きさのチェックは行われないので，呼び出し側で必要なサイズを edb_tcdatum_num(62) 等を用いて調べ，十分な大きさのものを用意する必要がある.

`num` には配列に格納されたデータの個数がセットされる.

Return Values :

EdbDatum 配列のポインタ.

See Also :

edb_tcdatum_from_array(67)

Function — 配列化からタプルカラムデータ

edb_tcdatum_from_array

Synopsis :

void
edb_tcdatum_from_array　(EdbContext ec, EdbTupleColumn tc, EdbDatum array[], int num)

DESCRIPTION :

edb_tcdatum_from_array は配列に保持されている EdbDatum をリンク化し，タプルカラムに格納する．それまでに EdbDatum(68) がもつリンク構造は破棄される．

SEE ALSO :

edb_tcdatum_to_array(66)

6.7.6　Datum 構造体

STRUCTURE — DATUM 構造体

EdbDatum

SYNOPSIS :

```
EdbDatum    datum ;
```

DESCRIPTION :

```
struct EDB_DATUM {
    ...
    Link l_child;
};
typedef struct EDB_DATUM *EdbDatum;
```

MACRO — DATUM 構造体マクロ (アクセス)

DatumSuper
DatumDateFrom
DatumDateTo
DatumRead
DatumEID
DatumEnglish
DatumJapanese
DatumPronounce

SYNOPSIS :

```
EdbTupleColumn
DatumSuper    ( EdbDatum datum )

int
DatumDateFrom    ( EdbDatum datum )

int
DatumDateTo    ( EdbDatum datum )

int
DatumRead    ( EdbDatum datum )

eid_t
DatumEID    ( EdbDatum datum )

utf8_t *
DatumEnglish    ( EdbDatum datum )

utf8_t *
```

DatumJapanese　(EdbDatum datum)

utf8_t *
DatumPronounce　(EdbDatum datum)

DESCRIPTION :

DatumSuper は，データが属しているカラムのポインタ (EdbTupleColumn) を得る．
DatumDateFrom，*DatumDateTo* は期間限定属性の日付を得る．
DatumRead はデータの読み権限レベルを得る．
DatumEID は EID を得る．
DatumEnglish，*DatumJapanese*，*DatumPronounce* はそれぞれ英語，日本語，読みの文字列を得るために利用する．
以上のマクロは，実際には構造体のメンバーにアクセスする．*datum* が NULL かどうかの判定は行っていない．

MACRO — DATUM 構造体マクロ (チェック)

DatumEIDisValid
DatumEIDisVALID
DatumEnglishIsValid
DatumJapaneseIsValid
DatumPronounceIsValid
DatumEnglishIsUsable
DatumJapaneseIsUsable
DatumPronounceIsUsable
DatumIsValid

SYNOPSIS :

bool
DatumEIDisValid　(EdbDatum datum)

bool
DatumEIDisVALID　(EdbDatum datum)

bool
DatumEnglishIsValid　(EdbDatum datum)

bool
DatumJapaneseIsValid　(EdbDatum datum)

bool
DatumPronounceIsValid　(EdbDatum datum)

bool
DatumEnglishIsUsable　(EdbDatum datum)

bool
DatumJapaneseIsUsable　(EdbDatum datum)

bool
DatumPronounceIsUsable　(EdbDatum datum)

bool
DatumIsValid　(EdbDatum datum)

DESCRIPTION :

DatumEIDisValid，*DatumEIDisVALID* は有効な EID の範囲が登録されているかどうかを検査する．両者の違いはなく，*DatumEIDisValid* の利用を推奨する．

DatumEnglishIsValid, *DatumJapaneseIsValid*, *DatumPronounceIsValid* はそれぞれ英語，日本語，読みの文字列が NULL 値，空の文字列 ("") のどちらかでない場合に真値を返す．

DatumEnglishIsUsable, *DatumJapaneseIsUsable*, *DatumPronounceIsUsable* はそれぞれ英語，日本語，読みの文字列が NULL 値，空の文字列 (""), ピリオドのみの文字列 (".") のいずれでもでない場合に真値を返す．

DatumIsValid は *DatumEIDisValid*, *DatumEnglishIsValid*, *DatumJapaneseIsValid*, *DatumPronounceIsValid* の論理和の真偽値を返す．

以上のマクロは，*datum* が NULL である場合には偽を返す．

Function — EdbDatum 構造体に値をセットする

edb_datum_set

Synopsis :

```
int
edb_datum_set  ( EdbContext ec, EdbDatum datum, eid_t, utf8_t *en, utf8_t *ja, utf8_t *pr )
```

Description :

EdbDatum 構造体に，EID，文字列をセットする．それまでにセットされていた文字列は内部で開放される場合がある．

EdbDatum 構造体に値をセットするために本関数以外を用いてはいけない．

Return Values :

成功した場合に真値を返す

See Also :

EdbDatum(68)

Macro — Datum 構造体マクロ (アクセス)

edb_datum_get_EID
edb_datum_get_English
edb_datum_get_Japanese
edb_datum_get_Pronounce
edb_datum_get_TEXT
edb_datum_get_INT
edb_datum_get_REAL
edb_datum_get_DATE
edb_datum_get_MONETARY
edb_datum_get_BOOL
edb_datum_get_YEAR
edb_datum_get_DATE2

Synopsis :

```
eid_t
edb_datum_get_EID  ( EdbContext ec,EdbDatum datum )

utf8_t *
edb_datum_get_English  ( EdbContext ec,EdbDatum datum )

utf8_t *
edb_datum_get_Japanese  ( EdbContext ec,EdbDatum datum )

utf8_t *
edb_datum_get_Pronounce  ( EdbContext ec,EdbDatum datum )

utf8_t *
edb_datum_get_TEXT  ( EdbContext ec,EdbDatum datum )

int
edb_datum_get_INT  ( EdbContext ec,EdbDatum datum )

double
```

edb_datum_get_REAL　(EdbContext ec,EdbDatum datum)

int32_t
edb_datum_get_DATE　(EdbContext ec,EdbDatum datum)

double
edb_datum_get_MONETARY　(EdbContext ec,EdbDatum datum)

bool
edb_datum_get_BOOL　(EdbContext ec,EdbDatum datum)

int
edb_datum_get_YEAR　(EdbContext ec,EdbDatum datum)

int
edb_datum_get_DATE2　(EdbContext ec, EdbDatum datum, int32_t *from, int32_t *to)

DESCRIPTION :
各タイプ毎に用意されているデータアクセスマクロ群.
文字列については，データがセットされているか否かに関わらず必ず非 NULL の値 ("") を返す.

6.8　データベース接続

FUNCTION — EDB の設定　edb_configure

SYNOPSIS :
EdbContext
edb_configure　(char *path)

DESCRIPTION :
EDB のコンフィガレーションを行う．path にはコンフィガレーションファイルのパス名を指定する.

RETURN VALUES :
成功した場合ポインター値 EdbContext を返す．失敗した場合には NULL を返す.

SEE ALSO :
edb_open(74), edb_close(74)

FUNCTION — EDB への接続/切断　edb_open　edb_close

SYNOPSIS :
int
edb_open　(EdbContext ec)

int
edb_close　(EdbContext ec)

DESCRIPTION :

edb_open は EDB のコアに利用されている PostgreSQL データベースバックエンドプログラム (postgres) と接続する.

edb_close は EDB のコアに利用されている PostgreSQL データベースバックエンドプログラム (postgres) との接続を切断する.

RETURN VALUES :

成功した場合 TRUE，失敗した場合 FALSE を返す.

SEE ALSO :

edb_configure(73)

FUNCTION — トランザクションの開始と終了

edb_begin
edb_end

SYNOPSIS :

```
int
edb_begin   ( EdbContext ec, int exclusive )

int
edb_end   ( EdbContext ec )
```

DESCRIPTION :

SQL トランザクションを開始と終了処理を行う.

edb_begin の引数 `exclusive` には排他的接続を行うか否かを論理値で指定する．(EDB_READER が定義されていないときのみ有効)

RETURN VALUES :

成功した場合 TRUE，失敗した場合 FALSE を返す.

SEE ALSO :

edb_open(74), edb_close(74)

EXAMPLE — EDB の接続と解除

```
#include <stdio.h>
#include "edb/edb.h"

void
main(int ac, char *av[])
{
    EdbContext(35) ec;

    ec = edb_configure(73)(NULL);
    if(!edb_open(74)(ec)) {
        fprintf(stderr, "can't open database\n");
        exit(1);
    }
    edb_begin(75)(ec, FALSE);      /* トランザクションの始まり */
```

```
    ...       /* この間では，データベースの整合性が保証される */
    edb_end(75)(ec);      /* トランザクションの終り */
    edb_close(74)(ec);
    exit(0);
}
```

FUNCTION — ブロッキングモードの設定

edb_pgsql_setnonblocking

SYNOPSIS :

int
edb_pgsql_setnonblocking (EdbContext ec, int nonblock)

DESCRIPTION :

nonblocking モードを設定する．
ノンブロッキングモードを指定すると，ちょっとはデータベースアクセスが早くなるが，現在のコードでは効率が悪い．

RETURN VALUES :

成功した場合 TRUE，失敗した場合 FALSE を返す．

SEE ALSO :

edb_begin(75)，edb_end(75)

6.9 NOTIFY

EDB では PostgreSQL データベースに接続するアプリケーション間で，イベントを通知するために SQL に実装されている NOTIFY, LISTEN コマンドを利用する．具体的には Dwarf によるデータベース間の情報の伝播に利用する．

FUNCTION — NOTIFY の送信と取得

edb_notify
edb_listen
edb_accept_notify

SYNOPSIS :

int
edb_notify (EdbContext ec, char *table)

int
edb_listen (EdbContext ec, char *table)

char *
edb_accept_notify (EdbContext ec)

DESCRIPTION :

NOTIFY コマンドの送信と取得を行う．
テーブル名は SQL 上の名前である．(XML 名でないことに注意)．
文字列テーブル名は *edb_accept_notify* 呼び出した側で開放 (free) しなくてはならない．

Return Values :

edb_notify, edb_listen は成功した場合 TRUE, 失敗した場合 FALSE を返す．edb_accpet_notify は NOTIFY イベントを受け取ったテーブル名 (XML 名) を返す．NOTIFY イベントがない場合には NULL を返す．

6.10　データベースアクセス関数

Function — PostgreSQL コマンド実行　edb_pgsql_command_exec

Synopsis :

int
edb_pgsql_command_exec　(EdbContext ec, char *command)

Description :

接続している PostgreSQL データベースにコマンドを実行させる．引数 command は SQL 文である．

Return Values :

成功した場合 TRUE, 失敗した場合 FALSE を返す．

Function — PostgreSQL に問い合わせ　edb_pgsql_query

Synopsis :

Link
edb_pgsql_query　(EdbContext ec, char *query, int *ntuples, int *nfields)

Description :

接続している PostgreSQL データベースに問い合わせを行う．

引数 query は問い合わせのための SQL 文である．引数 ntuples, nfields はそれぞれ，問い合わせの結果のタプル数およびフィールド数を戻すための整数型変数へのポインタである．

Return Values :

成功した場合には結果を格納した文字列の Link がかえる．失敗もしくは結果がない場合には NULL を返す．

Function — PostgreSQL に問い合わせ　edb_pgsql_query_one_field　edb_pgsql_query_one_tuple

Synopsis :

Link
edb_pgsql_query_one_field　(EdbContext ec, char *query, int *ntuples)

Link
edb_pgsql_query_one_tuple　(EdbContext ec, char *query, int *nfields)

Description :

FUNCTION — PostgreSQL に問い合わせを行う.

edb_pgsql_query_one_text
edb_pgsql_query_text
edb_pgsql_query_one_int

SYNOPSIS :

```
char *
edb_pgsql_query_one_text  ( EdbContext ec, char *query )

char *
edb_pgsql_query_text  ( EdbContext ec, char *query )

int
edb_pgsql_query_one_int  ( EdbContext ec, char *query )
```

DESCRIPTION :

FUNCTION — PostgreSQL に問い合わせを行う.

edb_pgsql_query_texts
edb_pgsql_query_integers

SYNOPSIS :

```
int
edb_pgsql_query_texts  ( EdbContext ec, char ***array, int *nelems, char *query )

int
edb_pgsql_query_integers  ( EdbContext ec, int **array, int *nelems, char *query )
```

DESCRIPTION :

FUNCTION — データベースを掃除する

edb_vacuum

SYNOPSIS :

```
int
edb_vacuum  ( EdbContext ec )
```

DESCRIPTION :

データベースを掃除し，解析値，統計値を計算する.
かなり時間がかかるので，インタラクティブな処理の際に呼び出すのは避けるべきである.

6.11　テーブル

FUNCTION — テーブル情報をデータベースから取得

edb_table_get
edb_table_get_by_name

Synopsis :

EdbTableInfo
edb_table_get　(EdbContext ec, eid_t eid)

EdbTableInfo
edb_table_get_by_name　(EdbContext ec, char *xn)

Description :

edb_table_get は，EID(eid) で指定したテーブル情報をデータベースから取得する．
edb_table_get_byname は，テーブルの XML 名 (xn) で指定したテーブル情報をデータベースから取得する．
取得されたテーブル構造体は LibEDB 内でキャッシングされているので，開放してはいけない．

Return Values :

テーブル構造体のポインタ．

See Also :

EdbTableInfo(44)

6.12　タプル

Function — タプルの情報をデータベースから取得　edb_tuple_get

Synopsis :

EdbTupleInfo
edb_tuple_get　(EdbContext ec, eid_t eid, ..., 0)

Description :

EID(eid) で指定したタプル情報をデータベースから取得する．
取得されたタプル構造体は LibEDB 内でキャッシングされているので，開放してはいけない．

Return Values :

タプル構造体のポインタ．

See Also :

EdbTupleInfo(52)

6.13　ドキュメント出力関数

LibEDB では，データベース登録内容を出力先の言語属性に合わせて出力するために，*doc_*()* 関数群を用意している．
現在サポートしているドキュメント作成言語 (Document Making Language; DML) には，

```
#define DocML_PLAIN     /* 平文 (plain text) */
#define DocML_XML      /* XML */
#define DocML_HTML      /* HTML */
#define DocML_TeX      /* TEX */
```

がある．

ドキュメント出力関数を利用するためには，まず，ドキュメント出力関数のコンテキストを保持する構造体

STRUCTURE — 出力関数の構造体のポインタ　Doc

SYNOPSIS :

Doc　doc ;

DESCRIPTION :

ドキュメント構造体の定義は下記のように宣言されている．

struct DOC { ... };

typedef struct DOC *Doc;

ライブラリ利用者はドキュメント構造体のメンバーの直接操作してはいけない．
ただし，文字コードセットを設定する

```
  int codeset;
#define DocCodeset_Unicode 0     /* default */
#define DocCodeset_Latin 1
```

と，画像出力関数の設定,

int (*picture_func) __P((struct DOC *, char *, void *));

void *picture_udata;

については，ユーザが直接参照，代入しても良い．

を作成する．

通常，ドキュメント出力関数の構造体を作成は doc_open(87)，開放は doc_close(88) を用いる．

FUNCTION — ドキュメントのオープン　doc_open

SYNOPSIS :

Doc
doc_open　(int dml, char *path, char *mode)

DESCRIPTION :

引数 dml には

- #define DocML_PLAIN

- #define DocML_XML
- #define DocML_HTML
- #define DocML_TeX

のいずれかを指定する．

引数 path はファイルのパス名，mode はファイルを開くときのモードを表す．引数 path，mode は内部で *fopen(3)* を呼び出すときにそのまま渡される．

Return Values :

ドキュメント構造体のポインタ．失敗した場合には NULL．

See Also :

doc_close(88)

Function — ドキュメントのクローズ　doc_close

Synopsis :

```
int
doc_close    ( Doc doc )
```

Description :

ドキュメントを閉じる．

Return Values :

通常は，*fclose(3)* に同じ．

See Also :

doc_open(87)

Function — メモリーへ出力するドキュメントのオープン　doc_sopen

Synopsis :

```
Doc
doc_sopen    ( int dml )
```

Description :

ファイルを開かずに，メモリー上のバッファ領域に出力するドキュメントを作成する．

引数 dml は doc_open(87) に同じ．

ドキュメントは doc_close(88) で閉じる．

See Also :

doc_close(88)

Function — ファイルポインタ指定ドキュメントのオープン/クローズ　doc_fpopen　doc_fpclose

Synopsis :

```
Doc
doc_fpopen    ( int dml, FILE *fp )

void
doc_fpclose    ( Doc doc )
```

Description :

引数 `dml` は doc_open(87) に同じ.
`fp` は既に *fopen(3)* でオープンされているファイルポインタ.
doc_fpopen で作成したドキュメントは，*doc_fpclose* で閉じる.

Function — テンポラリドキュメントのオープン

doc_tmpopen

Synopsis :

```
Doc
doc_tmpopen    ( int dml )
```

Description :

テンポラリファイルを用いて，ドキュメントをオープンする.
引数 `dml` は doc_open(87) に同じ.
ドキュメントは doc_close(88) で閉じる.

Function — ドキュメントのフラッシュ

doc_flush

Synopsis :

```
int
doc_flush    ( Doc doc )
```

Description :

ドキュメントのバッファ (FILE *などのストリームバッファ) の内容をファイルに強制出力する.
ドキュメントの出力先がファイルの場合には，*fflush(3)* を呼び出し，それ以外では何もしない.

Function — ドキュメント出力文字列の取得

doc_ucs4_sgets
doc_sgets
doc_sflush

Synopsis :

```
ucs4_t *
doc_ucs4_sgets    ( Doc doc )

char *
doc_sgets    ( Doc doc )

int
doc_sflush    ( Doc doc )
```

DESCRIPTION :

doc_sopen(89) で作成したドキュメントから，これまでに出力された文字列を得る．

doc_ucs4_sgets は文字列を UCS-4 エンコードで返し，*doc_sgets* は UTF-8 エンコードで返す．

doc_sflush はバッファ領域の文字列を消去する．

doc_ucs4_sgets, *doc_sgets* で，文字列を取得すると，自動的にドキュメント内部の文字列は消去される．

RETURN VALUES :

文字列のポインタを返す．得られた文字列は，呼び出し側で *free(3)* を用いて開放すること．

FUNCTION — テキスト無変換出力関数

doc_printf
doc_puts
doc_prefix_set
doc_prefix_reset
doc_putc

SYNOPSIS :

```
int
doc_printf    ( Doc doc, char *fmt, ... )

int
doc_puts    ( Doc doc, char *str )

int
doc_prefix_set    ( Doc doc, char *str )

int
doc_prefix_reset    ( Doc doc )

int
doc_putc    ( Doc doc, int c )
```

DESCRIPTION :

テキスト変換をしない出力を行う．

FUNCTION — コメントの出力関数

doc_comment_printf

SYNOPSIS :

```
int
doc_comment_printf    ( Doc doc, char *fmt, ... )
```

DESCRIPTION :

コメント文字列を出力する．

FUNCTION — テキスト変換出力関数

doc_text_puts
doc_text_putc
doc_text_puts3
doc_text_printf

SYNOPSIS :

```
int
doc_text_puts    ( Doc doc, char *str )
```

int
doc_text_putc　(Doc doc, int c)

int
doc_text_puts3　(Doc doc, char *prefix, char *str, char *postfix)

int
doc_text_printf　(Doc doc, char *fmt, ...)

DESCRIPTION :
テキスト変換を行う出力を行う.
引数 `str` および `c` は文字列変換の対象となるが, `prefix`, `postfix` は文字列変換されない.
doc_text_printf の引数 `fmt` 以降は, *printf(3)* と同じであり, 出力全体が文字列変換の対象となる.

FUNCTION — 文字属性の設定

doc_color_begin
doc_color_end
doc_underline_begin
doc_underline_end

SYNOPSIS :
void
doc_color_begin　(Doc doc, char *)

void
doc_color_end　(Doc doc)

void
doc_underline_begin　(Doc doc)

void
doc_underline_end　(Doc doc)

DESCRIPTION :
色属性, アンダーライン属性を設定する.
出力ドキュメントによっては, 何もしない.

FUNCTION — 日付の出力

doc_put_datetime

SYNOPSIS :
void
doc_put_datetime　(Doc doc, time_t t)

DESCRIPTION :
UNIX 時間 `t` を日付に変換して出力する.

MACRO — ドキュメントの DML

DocIsPLAIN
DocIsXML
DocIsHTML
DocIsTeX

Synopsis :

```
boolean
DocIsPLAIN  ( doc )

boolean
DocIsXML  ( doc )

boolean
DocIsHTML  ( doc )

boolean
DocIsTeX  ( doc )
```

Description :

ドキュメントの言語設定を検査する.

Macro — ドキュメントの言語設定　DocIsEnglish DocIsJapanese

Synopsis :

```
boolean
DocIsEnglish  ( doc )

boolean
DocIsJapanese  ( doc )
```

Description :

ドキュメントの言語設定を検査する.

Macro — ドキュメントのコードセット　DocIsLatin

Synopsis :

```
boolean
DocIsLatin  ( doc )
```

Description :

ドキュメントのコードセットを検査する.

6.14　カタログ (catalogue)

Libedb では，データベースから取得したタブルのリストを効率良く扱うために，カタログを生成する．カタログは以下の型で参照される.

Structure — カタログ構造体のポインタ　EdbCatalogue

SYNOPSIS :

EdbCatalogue　ca ;

DESCRIPTION :

カタログ構造体の定義は以下のとおり.

```
struct EDB_CATALOGUE {
    BiTree bt;      /* 二分木 */
    Link link;      /* リンク */
};
typedef struct EDB_CATALOGUE *EdbCatalogue;
```

カタログ構造体の中には，二分木 (BiTree$_{(15)}$) とリスト構造 (Link$_{(3)}$) がともに組み込まれており，必要に応じてそれらを使い分ける．二分木はタプルを EID 順で管理しタプルの登録順の情報は保持しない．リスト構造はタプルの登録順を保持する.

カタログを作成，開放は，以下の 2 つの関数によって行う.

FUNCTION — カタログの作成/開放　edb_catalogue_new　edb_catalogue_free

SYNOPSIS :

EdbCatalogue
edb_catalogue_new　( )

void
edb_catalogue_free　(EdbCatalogue ca)

DESCRIPTION :

edb_catalogue_new は，主に EDB のタプル情報を管理するためのカタログ構造体を生成する．生成直後のカタログには，オブジェクトは含まれていない.

edb_catalogue_free は，カタログ構造体を開放する．ただし，カタログに登録されているオブジェクトの実体については開放しない．オブジェクトを同時に開放したい場合には，カタログを開放する前に自ら開放する必要がある.

RETURN VALUES :

edb_catalogue_new では，作成されたカタログ構造体のポインタが返る．*edb_catalogue_free* の戻り値はなし.

空のカタログにオブジェクトを登録するには,

FUNCTION — カタログにオブジェクトを追加　edb_catalogue_append

SYNOPSIS :

int
edb_catalogue_append　(EdbCatalogue ca, eid_t eid, void *obj)

DESCRIPTION :

同じ `eid` のオブジェクトが既に含まれている場合には，何もしない.

を使用する．ただし，既に同じ *eid* がカタログ内に登録されているときには，その *eid* に対応するオブジェクトの登録は出来ない．

作成したカタログ内に目的のオブジェクトが存在しているか否かは，

FUNCTION — カタログ内のオブジェクト検索　edb_catalogue_lookup

SYNOPSIS :

```
void *
edb_catalogue_lookup  ( EdbCatalogue ca, eid_t eid )
```

DESCRIPTION :

`eid` に対応するオブジェクトをカタログから検索する．

RETURN VALUES :

`eid` に対応するオブジェクトのポインタを返す．オブジェクトがない場合には NULL を返す．

を利用することにより確認することが出来る．

FUNCTION — カタログ内のオブジェクトの個数　edb_catalogue_size

SYNOPSIS :

```
int
edb_catalogue_size  ( EdbCatalogue ca )
```

DESCRIPTION :

カタログに登録されているオブジェクトの個数を返す．

RETURN VALUES :

カタログに登録されているオブジェクトの個数．

MACRO — カタログ内のオブジェクトスキャン　ForCatalogueLink

SYNOPSIS :

```
void
ForCatalogueLink  ( Link l, EdbCatalogue ca )
```

DESCRIPTION :

実体は，for 文．

MACRO — カタログのオブジェクトのポインタを得る　CatalogueObject

SYNOPSIS :

```
void *
CatalogueObject  ( Link l )
```

DESCRIPTION :

EXAMPLE — カタログのスキャン

```
Link(3) l;
EdbCatalogue(102) ca;
...
ForCatalogueLink(107)(l, ca) {
    EdbTupleInfo(52) tuple = CatalogueObject(108)(l);

    ...
}
```

カタログはタプルの集合を手際良く操作するために作成した構造であるが，ユニークな ID (整数値) が対応づけられるオブジェクトについて操作することも可能である．

6.15　コンディション (condition)

FUNCTION — 条件式を合成する　edb_condition_make

SYNOPSIS :

char *
edb_condition_make　(EdbContext ec, EdbTableInfo ti, int logic, ...)

DESCRIPTION :

以降に示す条件式マクロの合成により，SQL 条件式を合成する．*logic* には

```
#define CONDITION_LOGIC_OR      /* 論理和 */
#define CONDITION_LOGIC_AND     /* 論理積 */
```

のいずれかを指定する．条件式のマクロの最期に，必ず ConditionTerminate(110) を記述する．

RETURN VALUES :

合成した条件式を含む文字列のポインタが返る．文字列は，呼出側で開放しなくてはならない．

MACRO — 条件式の終了　ConditionTerminate

SYNOPSIS :

ConditionTerminate

DESCRIPTION :

条件式の終了を示す．edb_condition_make(109) の引数の最期に必ず記述すること

MACRO — 空の条件式　ConditionNoOperation

SYNOPSIS :

ConditionNoOperation

DESCRIPTION :

何もしない条件式を作成する.

MACRO — 条件式のスキップ　ConditionSkipNext

SYNOPSIS :

ConditionSkipNext

DESCRIPTION :

次の条件式をスキップする.

MACRO — テーブル名の指定　ConditionPrefixTableName

SYNOPSIS :

ConditionPrefixTableName　(s)

DESCRIPTION :

edb_condition_make(109) で合成される条件式のカラム名の前につけるテーブル名を指定する.

MACRO — 続く条件式を否定する.　ConditionPrefixNot

SYNOPSIS :

ConditionPrefixNot

DESCRIPTION :

次にかかれた条件式 (1 つのみ) を否定する.

MACRO — タプルの有効性を示す条件式　ConditionAvaliable

SYNOPSIS :

ConditionAvaliable

DESCRIPTION :

登録情報の有効/無効を検査する条件式を作成する.

MACRO — 条件式の階層化　ConditionCondition

Synopsis :

ConditionCondition　(s)

Description :

既に作成した条件式を組み込むときに利用する．

Macro — EID に対する条件式

ConditionEIDMatch
ConditionEIDMatchNoExpand

Synopsis :

ConditionEIDMatch　(column, eid)

ConditionEIDMatchNoExpand　(column, eid)

Description :

対象としているカラムに指定した EID が含まれているかどうかを検査する．

ConditionEIDMatch では，検査する EID をマップの情報を元に展開する．ConditionEIDMatchNoExpand では，検査する EID をマップの情報を元に展開しない．

Macro — EID に対する条件式

ConditionCatalogueMatch
ConditionCatalogueMatchNoExpand

Synopsis :

ConditionCatalogueMatch　(column, ca)

ConditionCatalogueMatchNoExpand　(column, ca)

Description :

対象としているカラムに指定したカタログ中の EID が含まれているかどうかを検査する．ConditionEIDMatch では，検査するカタログ中の EID をマップの情報を元に展開する．ConditionEIDMatchNoExpand では，検査するカタログ中の EID をマップの情報を元に展開しない．

Macro — 整数値に対する条件式

ConditionIntegerEQ
ConditionIntegerLE
ConditionIntegerLT
ConditionIntegerGE
ConditionIntegerGT

Synopsis :

ConditionIntegerEQ　(column, v)

ConditionIntegerLE　(column, v)

ConditionIntegerLT　(column, v)

ConditionIntegerGE　(column, v)

ConditionIntegerGT　(column, v)

Description :

Macro — 日付に対する条件式

ConditionDateFrom

Synopsis :

ConditionDateFrom　(column, v)

Description :

Macro — 期間に対する条件式

ConditionDate2Overlap

Synopsis :

ConditionDate2Overlap　(column, v1, v2)

Description :

Macro — 実数値に対する条件式

ConditionRealEQ
ConditionRealLE
ConditionRealLT
ConditionRealGE
ConditionRealGT

Synopsis :

ConditionRealEQ　(column, v)

ConditionRealLE　(column, v)

ConditionRealLT　(column, v)

ConditionRealGE　(column, v)

ConditionRealGT　(column, v)

Description :

Macro — テキストに対する条件式

ConditionTextEQ

Synopsis :

ConditionTextEQ　(column, v)

Description :

Macro — テキストに対する条件式 (正規表現)

ConditionRegexTextEQ

Synopsis :

ConditionRegexTextEQ　(column, v)

Description :

6.16　見出しの生成と制御

EDB では情報の参照登録が行われた場合にその部分には情報識別子 EID を登録するが，閲覧や情報伝搬のなどのいろいろな部分で，EID の指し示す情報から見出しを作成し EID とともにテキストをリレーショナルデータベースなどに登録する．

LibEDB における見出しの作成は，以下の関数群によって行われる．

FUNCTION — 見出し構造体を作成する　edb_caption_new

SYNOPSIS :

Caption
edb_caption_new　(EdbContext ec)

DESCRIPTION :

通常，LibEDB の利用者はこの関数を利用することはない．

RETURN VALUES :

見出し構造体のポインタ．

FUNCTION — 見出し構造体を開放する　edb_caption_free

SYNOPSIS :

void
edb_caption_free　(EdbContext ec, Caption cap)

DESCRIPTION :

edb_caption_new(125)，edb_make_caption(127) などで作成した見出しを開放する．
見出し構造体を開放するのは，見出しの作成呼び出しを行った側で行う．

RETURN VALUES :

なし．

FUNCTION — 見出し構造体を開放する　edb_make_caption

SYNOPSIS :

Caption
edb_make_caption　(EdbContext ec, eid_t eid, int mode)

DESCRIPTION :

edb_make_caption で作成される構造体は，

```
struct _CAPTION {
    char *c_main;      /* 短い見出し */
    char *c_main_english;      /* 英文見出し */
    char *c_main_japanese;      /* 和文見出し */
    char *c_main_pronounce;      /* 読み見出し */
```

```
    char *c_sub;      /* 長い見出し */
};
typedef struct _CAPTION * Caption;
```

のようにメンバーが登録されている．各々のメンバーの値は NULL となり得る．
見出しを作成する際に指定できるモードとして，

```
#define CAP_long      /* 長い見出しを作成する */
#define CAP_norecurse      /* 階層化された情報の上位見出しを結合しない */
```

が指定できる．
edb_make_caption で作成した見出しは，edb_caption_free(126) で開放する．

RETURN VALUES :
見出し構造体のポインタ．

SEE ALSO :
edb_caption_free(126)

なお，【組織】情報のように階層化されている情報については，デフォルトで見出しを作成する際に上位情報の見出しを結合する．この結合は再帰的に行われる．これを阻止するためには，edb_make_caption(127) の引数 mode に CAP_long を指定すれば良いが，選択的にこれを阻止したい場合 (例えば，時組織の階層化は阻止，他組織については階層化見出しを作成する) には以下の関数群により調整を行う．

FUNCTION — 見出しを抑制する情報の追加/削除　edb_caption_omit_append　edb_caption_omit_remove

SYNOPSIS :

```
void
edb_caption_omit_append    ( EdbContext ec, eid_t eid )

void
edb_caption_omit_remove    ( EdbContext ec, eid_t eid )
```

DESCRIPTION :
edb_caption_omit_append で eid を抑制リストに追加し，*edb_caption_omit_remove* で抑制リストから削除する．
edb_caption_omit_remove において eid が既に含まれている場合には，何もしない．また，*edb_caption_omit_remove* で対応する eid がリストに含まれていない場合にも何もしない．

RETURN VALUES :
なし．

FUNCTION — 見出し抑制の確認　edb_caption_is_omitted

SYNOPSIS :

```
int
edb_caption_is_omitted    ( EdbContext ec, eid_t eid )
```

DESCRIPTION ：
　指定された eid の見出しが抑制されているかを検査する．
RETURN VALUES ：
　抑制を示す論理値．

FUNCTION — 見出し抑制リストの退避/復帰

edb_caption_omit_save
edb_caption_omit_restore

SYNOPSIS ：

```
void
edb_caption_omit_save　( EdbContext ec )

void
edb_caption_omit_restore　( EdbContext ec )
```

DESCRIPTION ：
　edb_caption_omit_save で現在の見出し抑制リストを保存し，*edb_caption_omit_restore* で復帰させる．
　edb_caption_omit_save を呼び出した後においてもそれまでの見出し抑制リストは有効である．
　これを完全に消去したい場合には，edb_caption_omit_clear(131) を呼び出すこと．
　edb_caption_omit_save で退避するリストはスタックをなす．すなわち，*edb_caption_omit_save*，*edb_caption_omit_restore* を入れ子構造で呼び出しても良い．
RETURN VALUES ：
　なし．

FUNCTION — 見出し抑制リストの消去

edb_caption_omit_clear

SYNOPSIS ：

```
void
edb_caption_omit_clear　( EdbContext ec )
```

DESCRIPTION ：
　見出し抑制リストを消去する．
RETURN VALUES ：
　なし．

6.17　プロセス制御

LibEDB では，一つのプログラムで複数の PostgreSQL バックエンドを利用して，QUERY の並列実行を行うことができる．ただし，この並列化を利用するためには，PostgreSQL バックエンドは readonly で利用すること．

LibEDB や LibEDB で利用しているライブラリはいわゆる Thread-Safe ではない．ただし，PostgreSQL の実行待ちの状態に入った時点においては Thread-Safe であることを保障し，このタイミングでプロセスを切替える．プロセスの切替えには，*edb_process_switch* を用

いる．各々のプロセスの実行は，pthread が提供する pthread_mutex_t をロックした状態で実行される．この排他制御により，複数のプロセスの内，実行状態にあるプロセスは 1 つに限定される．つまり，ここでいうプロセス制御は並列プロセスではなく，バックエンドへの QUERY 待ち状態の並列化であり，LibEDB 内部はシングルプロセスと同等に実行されていることに注意すること．

Function — 並列プロセスモードのセットアップ　edb_setup_processes

Synopsis :

```
int
edb_setup_processes    ( EdbContext ec, int num )
```

Description :

並列接続するバックエンドの個数を num で指定する．既に接続されているバックエンドの数よりも num が多い時，足りない分の接続を新たに作成する．既に接続されているバックエンドの数よりも num が少ない時には何もしない．

num に 0 を指定すると，バックエンドデータベースマシンの CPU 数から接続バックエンド数を自動的に調節する．

新しく接続されたバックエンドとの接続はノンブロッキングモードに初期化される．

この関数の戻り値は，接続バックエンドの総数を意味する．0 番目の接続はメインのスレッド専用であるため，この総数 −1 がプロセスで利用される接続の総数となる．ただし，トランザクション内において実際に利用できるプロセス数は edb_process_number_of_usable_process(??) で別途チェックすること．

Return Values :

接続されたバックエンドの総数．

Type — プロセスコールバック関数型　(*EdbProcessFunc)(void *data)

Synopsis :

```
typedef void    (*EdbProcessFunc)(void *data) ;
```

Description :

edb_process_create を用いて新規にプロセスを生成する際に，コールバックして欲しい関数の型宣言である．

Function — プロセスを作成する．　edb_process_create

Synopsis :

```
void
edb_process_create    ( EdbContext ec, EdbProcessFunc callback, void *data )
```

Description :

新規にプロセスを作成し，*callback(data)* を呼び出す．

この関数は必ずメインのスレッドから呼び出すこと．

Return Values :

なし．

Function — 実行プロセスを切替える．　edb_process_switch

Synopsis :

void

edb_process_switch　(EdbContext ec, int waiting)

Description :

プロセスの実行権を他のプロセスに渡す．プロセスの並列化はプログラム中で制御されて，同時に実行されるプロセスは 1 つに限定されるため，この関数を適宜呼び出さないと，並列化は行えない．例えば，メインループ中で入力待ちの状態になると，全てのプロセスの実行が停止する．これを避けるためには，*select (2)* 等の入力チェックを行なうシステムコールを利用し，入力がなければ *edb_process_switch* を呼び出すというようにしなければならない．

`waiting` は真理値であり，true の場合，プロセスの実行権を他のプロセスに渡した後，呼び出しプロセスの実行を数 μ 秒停止する．

Return Values :

なし．

Function — 全てのプロセスの終了を待つ．　edb_process_waitall

Synopsis :

void

edb_process_waitall　(EdbContext ec)

Description :

実行状態にある全てのプロセスの終了を待つ．

Return Values :

なし．

Function — 実行途上のプロセスの数を得る．　edb_process_number_of_active_processes

Synopsis :

int

edb_process_number_of_active_processes　(EdbContext ec)

Description :

実行状態にあるプロセスの数を得る．

Return Values :

実行待ちプロセスの数．

Function — 利用可能なプロセスの数を得る．　edb_process_number_of_usable_processes

Synopsis :

int

edb_process_number_of_usable_processes　(EdbContext ec)

DESCRIPTION :

利用可能なプロセスの数を得る．

バックエンドの並列化時に，各々のデータベースの AGE(EOID の最大値) が異なる可能性がある．複数のプロセスに対して，トランザクションを開始したときに各々のデータベースの AGE を取得し，AGE がメインのバックエンドに等しいもののみを利用する．この関数はトランザクション中において AGE がメインのバックエンドに等しいものの数を返す．

運が悪ければ利用可能なプロセス数が 0 の場合もありうる．その場合には，メインのバックエンドのみを利用する．

RETURN VALUES :

利用可能なプロセスの数．

FUNCTION — 空きプロセスの数を得る．

edb_process_number_of_idle_processes

SYNOPSIS :

```
int
edb_process_number_of_idle_processes    ( EdbContext ec )
```

DESCRIPTION :

実行状態にない利用可能なプロセスの数を得る．edb_process_number_of_usable_processes(138) で返るプロセス数から実行途上にあるプロセス数を差し引いたものになる．

RETURN VALUES :

空きプロセスの数．

第7章　LibEDBPub

7.1　出力スタイルと文書作成言語

ライブラリ LibEDBPub は

- 出力スタイル: Output Style
- 文書作成言語: Document Making Language (DML)

の 2 種の設定により，様々な出力を行えるように設計されている．

出力スタイルとしては，

スタイル	形式名
EP_PrintStyle_STANDARD	Standard
EP_PrintStyle_XML	XML
EP_PrintStyle_CSV	CSV
EP_PrintStyle_ITEMIZE	ITEMIZE
EP_PrintStyle_CAPTION	Caption

が用意されている．このスタイルと doc.c (libedb) で定義されている DML の組合せにより，目的の出力様式を作成する．

組合せは

Style/DML	XML	HTML	TEX	PLAIN	
STANDARD	×	○	plainTeX	○	「:」や「,」区切り
XML	○	○	plainTeX	○	XML の要素記述様式
CSV	×	○	plainTeX	○	TAB 区切り
ITEMIZE	×	○	LaTeX	○	要素/行で出力
CAPTION	×	○	plainTeX	○	「(」「)」や「[」「]」で区切り

のとおりである．

7.2　doc 関数系と print 関数系

doc_*() 出力関数系は，出力先の DML (Document Making Language) に依存する主に 1 文字毎，もしくは文字間のエスケープシーケンス (TEX) やタグ (HTML) の調整を行う．

これに対し，*edb_print_*()* 出力関数系はタプル全体の情報の記載方法等の調整を行う．

7.3　コンテキスト

LibEDBPub ではライブラリのコンテキストを保持するために，型 EdbPrint でポイントされる構造体 (EDB_PRINT) を利用する．

Structure — LibEDBPub 出力関数コンテキストのポインタ　EdbPrint

Synopsis :

```
EdbPrint    ep ;
```

Description :

ドキュメント構造体の定義は下記のように宣言されている．

```
struct EDB_PRINT { ... };
typedef struct EDB_CONTEXT *EdbPrint;
```

LibEDBPub に登録されている関数のほとんどは，型 EdbPrint の変数を第 1 引数にもつ．

EdbPrint の利用方法は，

Example — LibEDBPub を利用する

```
EdbPrint ep;

ep = edb_print_new(ec, doc, style);
...
edb_print_free(ep);
```

のようになる．*doc* は doc_open(87) 等で作成した，ドキュメント出力関数のコンテキスト，*style* は出力文書のスタイルを示す．

7.4　ライブラリ制御関数

Function — 出力コンテキストの作成/開放　edb_print_new　edb_print_free

Synopsis :

EdbPrint
edb_print_new　(EdbContext ec, Doc doc, int style)

void
edb_print_free　(EdbPrint ep)

Description :

ec は edb_configure(73) で作成し，edb_open(74) でオープンされた EDB コンテキストを指定する．
doc は doc_open(87) などで作成された，ドキュメント構造体を指定する．
style には

- #define EP_PrintStyle_STANDARD
- #define EP_PrintStyle_XML
- #define EP_PrintStyle_CSV
- #define EP_PrintStyle_ITEMIZE
- #define EP_PrintStyle_CAPTION

のいずれかを指定する．
edb_print_free は *edb_print_new* で作成したコンテキストを開放する．

Return Values :

edb_print_new は作成した出力用構造体のポインタが返す．*edb_print_free* の戻り値はなし．

Function — 出力コンテキストの push/pop

edb_print_push
edb_print_pop

Synopsis :

void
edb_print_push　(EdbPrint ep)

void
edb_print_pop　(EdbPrint ep)

Description :

edb_print_push で，出力コンテキスト環境の保存する．
出力コンテキストの内容を一時的に変更したい場合に *edb_print_push* で環境を保存する．保存した環境は，*edb_print_pop* で復帰させることが出来る．
出力コンテキストを push 後は，push 以前のコンテキストの内容をすべて保持している．
edb_print_pop は，*edb_print_push* で保存した出力コンテキスト環境を復帰する．
edb_print_push と *edb_print_pop* の呼び出し回数は同じになるようにしておくこと．

Return Values :

なし．

7.5　出力関数

EDB において情報を表示する関数は次の型で定義される．

FUNCTION — タプル出力関数の型　EdbPrintTupleFunc_t

SYNOPSIS :

```
int
(*EdbPrintTupleFunc_t)　( EdbPrint ep, EdbTupleInfo tuple )
```

DESCRIPTION :

edb_print_tuple(144) から呼び出される関数の型．

RETURN VALUES :

何らかの表示を行ったことを示す論理値．

SEE ALSO :

edb_print_tuple(144)

このようなタプル表示関数は，概ね EDB に登録されているテーブル単位で定義される．テーブル毎の表示関数を自動的に選別するものとして，次の edb_print_tuple(144) が用意されている．

FUNCTION — タプル情報の出力　edb_print_tuple

SYNOPSIS :

```
int
edb_print_tuple　( EdbPrint ep, EdbTupleInfo tuple )
```

DESCRIPTION :

タプル `tuple` をコンテキスト `ep` の状態に従い出力する．

RETURN VALUES :

何らかの表示を行ったことを示す論理値．

SEE ALSO :

EdbPrintTupleFunc_t(143)

EDB の登録情報を表示する場合には，この関数を入口として表示することが推奨される．

さて，タプルを表示する際に，そのタプルの情報を英語 (Latin) で表示するか日本語 (CJK) で表示するかを判別するために，以下の関数を利用する．

FUNCTION — タプル言語属性を求める関数の型　EdbPrintTupleLangFunc_t

SYNOPSIS :

```
int
(*EdbPrintTupleLangFunc_t)　( EdbPrint ep, EdbTupleInfo tuple )
```

DESCRIPTION :

edb_print_tuple_language(146) から呼び出される関数の型.

戻り値は,

```
EP_LANG_English      /* 英語 (Latin) */
EP_LANG_Japanese      /* 日本語 (CJK) */
```

のいずれか.

RETURN VALUES :

言語属性.

SEE ALSO :

edb_print_tuple_language(146)

FUNCTION — タプル情報の言語属性　edb_print_tuple_language

SYNOPSIS :

int

edb_print_tuple_language　(EdbPrint ep, EdbTupleInfo tuple)

DESCRIPTION :

表示言語の属性が EP_LANG_Auto のときにタプル tuple を英語/日本語のいずれで表示するかの情報を得る.

戻り値は,

```
EP_LANG_English      /* 英語 (Latin) */
EP_LANG_Japanese      /* 日本語 (CJK) */
```

のいずれか.

RETURN VALUES :

言語属性.

SEE ALSO :

EdbPrintTupleLangFunc_t(145)

表示の際の指定が既に EP_LANG_English や EP_LANG_Japanese に指定されている場合には，それに合わせて出力が行われるが，EP_LANG_Auto の場合には，上記の関数が呼び出される.

7.6　ユーザ定義出力関数の登録

LIBEDBPUB では，ユーザ定義の表示関数を動的にライブラリに登録することが出来る.

7.6.1　情報 (タプル) 単位

FUNCTION — タプル出力関数を登録する　edb_print_tuple_register_func

SYNOPSIS :

int
edb_print_tuple_register_func　(EdbPrint ep, char *xn, int style, EdbPrintTupleFunc_t print)

DESCRIPTION :

タプルを出力する関数を登録する.

SEE ALSO :

EdbPrintTupleFunc_t(143), UserDefinedPrintTuple(148)

FUNCTION — タプル表示 (ユーザ定義)　UserDefinedPrintTuple

SYNOPSIS :

int
UserDefinedPrintTuple　(EdbPrint ep, EdbTupleInfo tuple)

DESCRIPTION :

タプルの表示関数.

RETURN VALUES :

関数値は，何らかの表示を行ったことを示す論理値.

SEE ALSO :

edb_print_tuple_register_func(147)

FUNCTION — タプルの言語属性を調べる関数を登録する　edb_print_tuple_register_language_func

SYNOPSIS :

int
edb_print_tuple_register_language_func　(EdbPrint ep, char *xn, EdbPrintTupleLangFunc_t lang_func)

DESCRIPTION :

言語選択が自動 (デフォルト) の状態ときに，英文，和文のいずれで出力するかを決定するために，タプルの情報を調べる必要があるときに登録する.

SEE ALSO :

EdbPrintTupleLangFunc_t(145), UserDefinedLanguageDecision(150)

FUNCTION — タプル言語属性を調べる (ユーザ定義)　UserDefinedLanguageDecision

SYNOPSIS :

int
UserDefinedLanguageDecision　(EdbPrint ep, EdbTupleInfo tuple)

Description :

タプルの言語属性を調べるユーザ定義関数.

Return Values :

関数値は,

```
EP_LANG_English     /* 英語 (Latin) */
EP_LANG_Japanese     /* 日本語 (CJK) */
```

のいずれかとする.

See Also :

EdbPrintTupleLangFunc_t(145), edb_print_tuple_register_language_func(149)

Function — タプル表示プレフィックス

edb_print_tuple_prefix

Synopsis :

int
edb_print_tuple_prefix　(EdbPrint ep, EdbTupleInfo tuple)

Description :

タプルの表示前に呼び出す関数である.

See Also :

UserDefinedPrintTuple(148)

Function — タプル表示ポストフィックス

edb_print_tuple_postfix

Synopsis :

int
edb_print_tuple_postfix　(EdbPrint ep, EdbTupleInfo tuple)

Description :

タプルの表示後に呼び出す関数である.

See Also :

UserDefinedPrintTuple(148)

Function — カラム表示関数

edb_print_tuple_column

Synopsis :

int
edb_print_tuple_column　(EdbPrint ep, EdbTupleInfo tuple, char *pre, char *dpre, EdbTupleColumn tc, EdbPrintTupleColumnCallbackFunc_t func, char *dpost, char *post, char *substitute, int alert)

DESCRIPTION :

カラムを表示する関数である．
引数の意味は以下のとおり．

- ep... EdbPrint 構造体のポインタ．
- tuple... 表示対象のタプル．
- pre... 一つ目のデータの前に表示される文字列．
- dpre... 2 つ目以降のデータの前に表示される文字列．
- tc... カラム．
- func... サブカラムを表示するためのコールバック関数．
- dpost... 最期を除くデータの後ろに表示される文字列．
- post... 最期のデータの後ろに表示される文字列．
- substitute... データが無いときに表示される文字列．
- alert... データが無いときの警告レベル．
  - #define EP_WARNING /* ワーニング */
  - #define EP_ALERT /* 警告 */

表示は，

$$\texttt{pre}\ D_1\ \texttt{dpost dpre}\ D_2\ \texttt{dpost}\ \ldots\ \texttt{dpre}\ D_{n-1}\ \texttt{dpost dpre}\ D_n\ \texttt{post}$$

のようになる．スタイルパラメータ

- EpContext(ep)->tcd_mode
  - #define EP_TCD_MODE_ITALIC /* 斜自体 */
  - #define EP_TCD_MODE_BOLD /* 太字体 */
  - #define EP_TCD_MODE_TT /* テレタイプ */
- EpContext(ep)->column_mode
  - #define EP_COLUMN_MODE_AT_ANYTIME /* 期間限定属性に関わらず，全ての要素を表示する */
  - #define EP_COLUMN_MODE_LIST_AND /* 英語モードで要素が複数あるときに，最期を○○ and ○○のように表示する */

RETURN VALUES :

何らかの出力を行ったことを示す論理値．

SEE ALSO :

UserDefinedPrintTuple(148)

FUNCTION — カラム表示関数

edb_print_tuple_named_column

SYNOPSIS :

int
edb_print_tuple_named_column (EdbPrint ep, EdbTupleInfo tuple, EdbDatum datum, char *pre, char *dpre, char *tcname, EdbPrintTupleColumnCallbackFunc_t func, char *dpost, char *post, char *substitute, int alert)

DESCRIPTION :

カラムを表示する関数である.

引数の意味は以下のとおり.

- ep... EdbPrint 構造体のポインタ.
- tuple... 表示対象のタプル.
- datum... 上位のデータ.
- pre... 一つ目のデータの前に表示される文字列.
- dpre... 2 つ目以降のデータの前に表示される文字列.
- tcname... カラムの XML 名.
- func... サブカラムを表示するためのコールバック関数.
- dpost... 最期を除くデータの後ろに表示される文字列.
- post... 最期のデータの後ろに表示される文字列.
- substitute... データが無いときに表示される文字列.
- alert... データが無いときの警告レベル.
  - #define EP_WARNING /* ワーニング */
  - #define EP_ALERT /* 警告 */

表示は,

$$\texttt{pre}\ D_1\ \texttt{dpost dpre}\ D_2\ \texttt{dpost}\ \ldots\ \texttt{dpre}\ D_{n-1}\ \texttt{dpost dpre}\ D_n\ \texttt{post}$$

のようになる. スタイルパラメータ

- EpContext(ep)->tcd_mode
  - #define EP_TCD_MODE_ITALIC /* 斜自体 */
  - #define EP_TCD_MODE_BOLD /* 太字体 */
  - #define EP_TCD_MODE_TT /* テレタイプ */
- EpContext(ep)->column_mode
  - #define EP_COLUMN_MODE_AT_ANYTIME /* 期間限定属性に関わらず, 全ての要素を表示する */
  - #define EP_COLUMN_MODE_LIST_AND /* 英語モードで要素が複数あるときに, 最期を○○ and ○○のように表示する */

RETURN VALUES :

何らかの出力を行ったことを示す論理値.

SEE ALSO :

UserDefinedPrintTuple(148)

FUNCTION — カラム表示関数 (廃止予定)　edb_book_print_named_column

SYNOPSIS :

int
edb_book_print_named_column　(EdbPrint ep, EdbTupleInfo tuple, char *pre, char *dpre, char *tcname, char *dpost, char *post, char *substitute, int mode, int alert)

DESCRIPTION :

カラムを表示する関数である.
引数の意味は以下のとおり.

- ep... EdbPrint 構造体のポインタ.
- tuple... 表示対象のタプル.
- pre... 一つ目のデータの前に表示される文字列.
- dpre... 2 つ目以降のデータの前に表示される文字列.
- tcname... カラムの XML 名.
- dpost... 最期を除くデータの後ろに表示される文字列.
- post... 最期のデータの後ろに表示される文字列.
- substitute... データが無いときに表示される文字列.
- mode... カラムの表示モード.
  - #define EP_TCD_MODE_ITALIC /* 斜自体 */
  - #define EP_TCD_MODE_BOLD /* 太字体 */
  - #define EP_TCD_MODE_TT /* テレタイプ */
- alert... データが無いときの警告レベル.
  - #define EP_WARNING /* ワーニング */
  - #define EP_ALERT /* 警告 */

表示は,

$$\texttt{pre}\ D_1\ \texttt{dpost dpre}\ D_2\ \texttt{dpost}\ \ldots\ \texttt{dpre}\ D_{n-1}\ \texttt{dpost dpre}\ D_n\ \texttt{post}$$

のようになる.

RETURN VALUES :

何らかの出力を行ったことを示す論理値.

SEE ALSO :

UserDefinedPrintTuple(148)

7.6.2　項目 (カラム) 単位

FUNCTION — カラム出力関数を登録する　edb_print_tcd_register_func

SYNOPSIS :

int
edb_print_tcd_register_func　(EdbPrint ep, char *xn, int style, EdbPrintTCDFunc_t print_tcd)

DESCRIPTION :

タプルを出力関数を登録する.

SEE ALSO :

UserDefinedPrintTCD(157)

FUNCTION — カラム表示 (ユーザ定義)　UserDefinedPrintTCD

SYNOPSIS :

int
UserDefinedPrintTCD　(EdbPrint ep, EdbTupleInfo tuple, EdbTupleColumn tc, EdbDatum datum)

DESCRIPTION :

カラムの表示関数.

RETURN VALUES :

関数値は，何らかの表示を行ったことを示す論理値.

SEE ALSO :

edb_print_tcd_register_func(156)

7.6.3　型 (タイプ) 単位

FUNCTION — タイプ出力関数を登録する　edb_print_type_register_func

SYNOPSIS :

int
edb_print_type_register_func　(EdbPrint ep, int type, int style, EdbPrintTCDFunc_t print_type)

DESCRIPTION :

タプルを出力関数を登録する.

SEE ALSO :

UserDefinedPrintType(159)

FUNCTION — タイプ表示 (ユーザ定義)　UserDefinedPrintType

SYNOPSIS :

int
UserDefinedPrintType　(EdbPrint ep, EdbTupleInfo tuple, EdbTupleColumn tc, EdbDatum datum)

DESCRIPTION :

カラムの表示関数.

RETURN VALUES :

関数値は，何らかの表示を行ったことを示す論理値.

SEE ALSO :

edb_print_type_register_func(158)

7.7　システムデフォルトの出力関数

7.7.1　libedbpub/modules

ディレクトリ libedbpub/modules は，システムのデフォルトの出力関数を格納するために用意されている.
ファイルの名前は，libedbpub/のファイルと衝突しないように

mod_*xmlname*.c

を用いて，テーブル毎に出力関数のモジュールを作成する．このファイルには，

```
static struct EDB_PRINT_MODULE_INFO(160) module_info = {
        /* xmlname */    "xmlname",
        /* language */    言語選択関数のポインタ,
        /* std */    標準出力関数のポインタ,
        /* xml */    XML 出力関数のポインタ,
        /* csv */    CSV 出力関数のポインタ,
        /* item */    ITEMIZE 出力関数のポインタ,
        /* cap */    見出し出力関数のポインタ
};
DECLARE_PRINT_MODULE(161)(id, module_info)
```

を含ませておく.

STRUCTURE — 出力モジュールの情報　EDB_PRINT_MODULE_INFO

SYNOPSIS :

struct EDB_PRINT_MODULE_INFO　module_info ;

DESCRIPTION :

構造体の各メンバーは以下のとおり.

```
static struct EDB_PRINT_MODULE_INFO {
    char *xmlname;      /* XML 名 */
    EdbPrintTupleLangFunc_t lang;      /* 言語選択関数のポインタ */
    EdbPrintTupleFunc_t std;      /* 標準出力関数のポインタ */
    EdbPrintTupleFunc_t xml;      /* XML 出力関数のポインタ */
    EdbPrintTupleFunc_t csv;      /* CSV 出力関数のポインタ */
    EdbPrintTupleFunc_t item;      /* ITEMIZE 出力関数のポインタ */
    EdbPrintTupleFunc_t cap;      /* 見出し出力関数のポインタ */
};
```

SEE ALSO :

DECLARE_PRINT_MODULE(161)

MACRO — 出力モジュールの登録　DECLARE_PRINT_MODULE

SYNOPSIS :

DECLARE_PRINT_MODULE　(id, module_info)

DESCRIPTION :

引数 id は登録されるすべてのモジュールについてユニークでなくてはならない．結果として，受け持つテーブルの XML 名にすることが推奨される．

引数 module_info は，構造体 EDB_PRINT_MODULE_INFO(160) で作成した名前を指定する．

SEE ALSO :

EDB_PRINT_MODULE_INFO(160)

作成したモジュールをライブラリに組み込むには，

libedbpub/Imakefile

の

MODULE_SRCS

MODULE_OBJS

にモジュール名を追加することで行う．

7.7.2　モジュール追加上の注意

新しいモジュールを作成し，追加するときの注意を以下に記述する．

- **モジュール名**特別な理由がない限り出力関数は，テーブル毎にファイル化し，

  mod_*xmlname*.c

  のように名付ける．先頭の mod_ は libedbpub の他のファイル名と衝突しないようにするためであり，*xmlname* を用いるのは視認性を良くするためにである．

- **ソースコード**

  モジュール内にて宣言した変数，関数はすべて static (外部参照不可) のように作成する．

  これも既存の変数，関数名と衝突しないようにするためである．

  モジュールから外部への情報受渡しは，すべて DECLARE_PRINT_MODULE を介して行う．

- **DECLARE_PRINT_MODULE**

  id には，*xmlname* と同じ名前を記述する．

  マクロ DECLARE_PRINT_MODULE がないと，モジュールの追加は行われない．モジュールを外す場合には，

```
#if 0
DECLARE_PRINT_MODULE(161)(article, module_info)
#endif
```

  のようにすると，モジュールとして登録されない．ただし，オブジェクトのライブラリアーカイブへの追加は行われる．

  徳島大学では，徳島大学のテーブル定義に依存した出力関数は

```
#ifdef THE_UNIVERSITY_OF_TOKUSHIMA
DECLARE_PRINT_MODULE(161)(article, module_info)
#endif
```

  としている．

第8章　LibEDBRとLibEDBPubR

LibEDBR は LibEDB を EDB_READER 用にカスタマイズしたもの．LibEDBPubR は LibEDBPub を EDB_READER 用にカスタマイズしたもの．

コンパイル時に EDB_READER オプションをつけると，読み込み専用プログラムが作成される．

第III部

EDB運用

第9章　EDBの設定と関連ソフトウェア

9.1　Operating System (FreeBSD)

1. PostgreSQL は SysV 系の Shared Memory および Semaphore を利用している．試しに運用する場合にはデフォルトのカーネル設定で十分だが，クライアント数が増えるとこれらのリソースが足りなくなるのでほどほどに増やしておくこと．

2. 現在の EDB のソースは on Memory のデータ容量を切り詰めていないので，1 プロセスで使用可能なメモリ容量はそれ相応に大きく (少なくとも 512MB 以上) しておいた方が無難である．また，トータルの VM のサイズも大きくしておく方がベター．

9.2　File System

1. PostgreSQL はデフォルトで*/usr/local/pgsql* にインストールされるが，データベースの規模やアクセス頻度，速度などを考えた場合には別のディスクに分離しておいた方がよい．

9.3　PostgreSQL

1. MultiByte に対応したバージョンの PostgreSQL をインストールすること．

2. デフォルトで利用するコードは Unicode (UTF-8 エンコーディング) にする．

3. contrib/array/array_iterator を利用できるようにする．

4. データベース管理用の UNIX ユーザ (edb) を作成する．

   - `/etc/passwd:  edb:*:443:443:EDB Owner:/usr/local/edb:/bin/csh`
   - `/etc/group:  edb:*:443:root,edb`

ホームディレクトリ (/usr/local/edb/) には，.cshrc, .profile などの rc ファイルをおいておく (cron を利用して，定期的に何かを実行する場合に共通の設定を行っておくと便利).

5. データベース管理用のユーザ (edb) を **createuser** で作成する．このユーザにはデータベースを作成する権限を与えておくこと．このユーザが EDB のマスターになる.
6. PostgreSQL レベルで公開用のデータベースを作成する場合には guest 等のユーザを作成しておく．もちろん，このユーザには特別な権限を与えるべきではない．guest ユーザには単純なパスワードを設定しておく.
7. /usr/local/pgsql/data/postgresql.conf のチューニングは自由．ただし，tcpip_socket は true に．同時接続数は増やしておいた方がいい．Administrator's Manual をよく読んで設定する.
8. 公開用のデータベースを作成した場合には，/usr/local/pgsql/data/pg_hba.conf にアクセス権を定義しておく.
9. ログファイル (/var/log/pgsql) はすぐに大きくなるので，無効にするかもしくは定期的に rotate する仕組みを施しておいた方がいい

9.4　Apache

1. EDB では登録作業内容の秘密性を保つため，https(SSL) を利用できるように設定しておくこと．EDB で作成した CGI プログラムは SSL のセッションであるかどうかをチェックしている．なお，CA の認証を得ている必要はない.
2. EDB では各接続の認証に Cookie を利用するので，Cookie 機能を ON にしておくこと.
3. データベースにアクセスできるユーザは特定の UNIX ユーザ (edb) のみである．したがって，CGI で起動されるプログラムの実効ユーザが edb でなくてはならない．SUID で実効ユーザを変更するやり方はあまりいい方法ではないので，設定ファイル (httpd.conf) にて実効ユーザを edb に変更し，かつ suexec を利用して，CGI プログラムの実効ユーザを変更する．CGI のプログラムファイルの Owner はもちろん edb にしておく.
4. ログファイル (/var/log/access.log など) はすぐに大きくなるので，他のディレクトリに移したり rotatelog を使うなどして，ハードディスクが溢れないようにしておくのが無難.

9.5　EDB

9.5.1　データベースの構成

edb_template

EDB のデータベースを作成するためのテンプレートデータベース.

/usr/local/edb/sbin/make-edb-template.sh を利用して作成する.

edb_database

原典情報をもつデータベース.

情報の登録は必ずこのデータベースに対して行われる.

edb_database 上での処理を軽減させるために，このデータベースにおいては XML 表現のテーブル (edb_xmldb, edb_authentication) のみを利用する.

データベースを作成するためのテンプレートを共通にしているので，他のテーブルの定義も実際に行われているが使わない.

作成方法は以下の通り．ただし，それ以前の登録内容は消えてしまうので注意せよ.

1. 初期登録内容を,

   ```
   edb_database.database-table.dump
   edb_database.database-tuple.dump
   edb_database.authentication.dump
   ```

   に用意しておく.
2. EDB# `dropdb edb_database`
3. EDB# `createdb edb_database --template edb_template`
4. EDB# `edb_restore edb_database edb_database`
5. EDB# `edb_adjust edb_database`
6. EDB# `edb_check edb_database`

edb_databaser

edb_database のレプリカ.

登録情報の XML 表現だけでなく，RDB 上に展開したテーブルを保有する.

作成方法は以下の通り.

1. EDB# `createdb edb_databaser --template edb_template`
2. EDB# `edb_copy localhost:5432:edb_database localhost:5432:edb_databaser`

3. EDB# `edb_adjust localhost:5432:edb_databaser`

データベースのマネージメントには，後に述べる edb_dwarf を利用しているが，このデータベースは多数の edb_dwarf が原典情報をもつデータベースにアクセスすることを回避するために用意している．

徳島大学で運用しているシステムでは，この時点で原典情報のデータベースをもつサーバとは別のサーバになっているため，実際の名前は edb_database になっている．

edb_database$n$

edb_database のレプリカ．

登録情報の XML 表現だけでなく，RDB 上に展開したテーブルを保有する．

閲覧や検索に利用する．

n は 0 から始める．

作成方法 (edb_database0) は以下の通り．

1. EDB# `createdb edb_database0 --template edb_template`
2. EDB# `edb_copy localhost:5432:edb_databaser localhost:5432:edb_database0`
3. EDB# `edb_adjust localhost:5432:edb_database0`

edb_session

アクティブなセッションを管理するためのデータベース．

テーブル (edb_session) のみを利用する．

作成方法は以下の通り．

1. EDB# `createdb edb_session --template edb_template`

edb_user_template

EDB のユーザコンテキストを格納するデータベースを作成するためのテンプレートデータベース

/usr/local/edb/sbin/make-edb-template.sh を利用して作成する．

edb_user_eid

EDB のユーザコンテキストを格納するデータベース.

eid は，ユーザに対応する個人情報の EID.

ユーザ毎の環境設定や編集中の情報の内容などを格納する.

新しいユーザがログインしたときに自動的に作成される.

edb_latest

edb_database の登録情報中の履歴の最終情報のみをもつデータベース.

登録情報の XML 表現だけでなく，RDB 上に展開したテーブルを保有する.

edb_university, edb_public を作成するための中間データベース.

作成方法は以下の通り.

1. EDB# `createdb edb_latest --template edb_template`
2. EDB# `edb_copy -latest localhost:5432:edb_databaser localhost:5432:edb_latest`
3. EDB# `edb_adjust localhost:5432:edb_latest`

edb_university

edb_latest 中の学内公開可能な情報を蓄積するデータベース.

登録情報の XML 表現だけでなく，RDB 上に展開したテーブルを保有する.

作成方法は以下の通り.

1. EDB# `createdb edb_university --template edb_template`
2. EDB# `psql -f /usr/local/edb/etc/grant.sql edb_university`
3. EDB# `edb_copy -university localhost:5432:edb_latest localhost:5432:edb_university`
4. EDB# `edb_adjust -grant-select-to-public localhost:5432:edb_university`

edb_public

edb_latest 中の学外公開可能な情報を蓄積するデータベース.

登録情報の XML 表現だけでなく，RDB 上に展開したテーブルを保有する.

作成方法は以下の通り.

1. EDB# `createdb edb_public --template edb_template`
2. EDB# `psql -f /usr/local/edb/etc/grant.sql edb_public`
3. EDB# `edb_copy -public localhost:5432:edb_latest localhost:5432:edb_public`
4. EDB# `edb_adjust -grant-select-to-public localhost:5432:edb_public`

9.5.2　データベースアクセス設定ファイル (edb.conf)

- EDB では，`edb.conf` にデータベースのロケーション情報，アクセス情報等を記入しておき，各プログラムはこのファイルを参照する．
- `edb.conf` は*/usr/local/edb/etc/*におく．
- `edb.conf` の他に，`edb0.conf`, `university.conf`, `public.conf` を使うこともある．これらは，プログラム中で指定される．
- `edb.conf` 中の記述方法は，`conf.c` を読むことによって大部分は理解可能である．よく使うものを列記しておく．
  - '#' 以降 … 注釈
  - pgsqlOptions … PostgreSQL への接続オプション (現在は利用していない)
  - pgsqlTty … PostgreSQL への接続オプション (現在は利用していない)
  - pgsqlDBPath … EDB のデータベースのロケーション情報．「ホスト:ポート:データベース. レプリカ数」で指定する．
    (例) pgsqlDBPath localhost:5432:edb_database.4
  - WhoisServer … EDB のデータベースのロケーション情報を whois サーバから得る場合に指定する．
    (例) WhoisServer db.db.tokushima-u.ac.jp
  - WhoisQuery … EDB のデータベースのロケーション情報を whois サーバから得る場合に指定する．
    (例) WhoisQuery LOOK=DBPATH.universiy ...(学内公開データベース)
    (例) WhoisQuery LOOK=DBPATH.public ...(学外公開データベース)
    (例) WhoisQuery LOOK=DBPATH ...(接続ホストの IP アドレスによって，学内外を選択)
    pgsqlDBPath と WhoisServer，WhoisQuery が両方とも定義された場合には，whois で得られる情報の方が優先され，whois でデータベースのロケーション取得に失敗した場合 pgsqlDBPath の定義が利用される．
  - MyNetwork … 自ネットワークの IP アドレス/ネットマスク
    (例) MyNetwork 150.59.0.0/255.255.0.0

- EDBMasterUID…EDB のマスターユーザの UID
  (例) EDBMasterUID 443
- CaptionOmit…EDB のプログラム群が作成する見出しのうち，階層化情報の見出しのある部分を抑制する場合に利用する．
  (例) CaptionOmit *12345* # Government(Organization)
  階層化された上位に [政府] があった場合，それ以上の見出しは結合しない．

9.5.3　データベース管理プログラム

下記のプログラムではデータベースの指定子は全て「host:port:dbname」の形式で指定する．

make-edb-template.sh

テンプレートデータベースを作成する．

作成されるテンプレートデータベースは以下の 2 つである．

edb_template

EDB で利用する各種データベースの雛型．

`/usr/local/edb/etc/base.sql` に記述された内容に沿い，テンプレートを作成する．base.sql には，SQL に対するユーザ拡張関数の読み込みや雛型のテーブル作成等が記述されている．

edb_user_template

EDB でユーザコンテキストを記憶するデータベースの雛型．(現時点で，実質上は edb_template に同じ)

(USAGE) EDB# `make-edb-template.sh`

edb_check

データベースの登録情報，整合性を検査するプログラム．

(USAGE) EDB# `edb_check` *`db`*

edb_copy

データベース間での XMLDB のコピー．

(USAGE) EDB# `edb_copy [-all|-latest|-university|-public]` *`src-db dst-db`*

edb_adjust

XMLDB から RDB を作成する．

(USAGE) EDB# `edb_adjust [-grant-select-to_public]` *db*

edb_dump

データベースのダンプファイルを作成する．(XMLDB のみ)

(USAGE) EDB# `edb_dump` *db dumpfile*

ファイルとして

- *dumpfile*.database-table.dump
- *dumpfile*.database-tuple.dump
- *dumpfile*.authentication.dump

が作成される．

これらのファイルは，1 行 1 行が独立な XML 表現のデータになっている．

edb_restore

ダンプファイルの内容をデータベースに登録する．

(USAGE) EDB# `edb_restore` *db dumpfile*

edb_dwarf

データベース管理デーモンプログラム．

(USAGE) EDB# `edb_dwarf [-all|-latest|-university|-public]` *db ref-db*

ref-db で指定されたデータベースの更新情報をもとに，*db* の登録情報の各種調整を行う．

データベースの更新通知は，SQL コマンドの NOTIFY を利用している．

データベース一つ毎に edb_dwarf が一つ起動される．

edb_dwarf を用いたデータベースの連係例.

edb_database	←原典情報（登録は全てこのデータベースに行われる）		
↓ (edb_dwarf) ↓			
edb_databaser	CGI プログラム edb_browse, edb_edit 等からの参照		
↓ (edb_dwarf -latest) ↓			
edb_latest	→ (edb_dwarf -university) →	**edb_university**	学内公開可能情報への学内からのアクセス
	→ (edb_dwarf -public) →	**edb_public**	学外公開可能情報へのアクセス

末端のデータベースはいくつ用意してもよいが，情報が更新されたときに edb_dwarf が一斉に動くためほどほどの数にしておいた方がよい．複数台のサーバに分散させてもよい.

第10章　EDB運用のための準備

EDB を構築，運用するには，初期登録情報を用意しなくてはならない．さもなければ，一人のユーザもシステムにログインして情報を登録することができない．本章では EDB を構築する際に必要な初期情報ファイルについて述べる．

EDB では全ての情報を Unicode テキストで表現し，かつ XML 形式で保存する．したがって，初期設定ファイルも XML 形式のファイルとして最終的に作成する必要がある．作成すべきファイルは

- `edb_database.database-table.dump`
- `edb_database.database-tuple.dump`
- `edb_database.authentication.dump`

の 3 つである．

これらのファイルは，各行が XML で記述されかつ DTD によって LINT 可能なものである必要がある．すなわち，行頭は

- `<xml version="1.0" encoding="utf-8"?>`

ではじまり，行末で閉じる XML によるデータ表現である．

以下に，それぞれの内容について述べる．

`edb_database.database-table.dump`

このファイルの作成については，テーブルの定義情報.

DTD: http://web.db.tokushima-u.ac.jp/dtds/table.dtd

または，

http://web.db.tokushima-u.ac.jp/dtds/

の頁で閲覧可能なテーブルの定義を参考にして欲しい．

また，EDB での XML の表現の手法やテーブルの定義方法については

http://web.db.tokushima-u.ac.jp/edb-manual/xml.html

の頁に記述がある．

`edb_database.database-tuple.dump`

タプルの情報 (すなわち通常の登録情報)．

DTD: http://web.db.tokushima-u.ac.jp/dtds/テーブルの XML 名.dtd

`edb_database.authentication.dump`

認証パスフレーズの情報．

DTD: http://web.db.tokushima-u.ac.jp/dtds/edb-auth.dtd

1 行を展開したものは下記のように表現されている．

```
 1:    <?xml version="1.0" encoding="utf-8"?>
 2:    <!DOCTYPE edb:auth SYSTEM "http://web.db.tokushima-u.ac.jp/dtds/auth.dtd">
 3:    <edb:auth xmlns:edb="http://web.db.tokushima-u.ac.jp/dtds/">
 4:       <edb:base eid="個人情報の EID" eoid="10316"
 5:          mapto="0"
 6:          mtime="変更時刻の UNIX タイム"
 7:          operator="443" avail="true" owner="443"
 8:          read="inherit" write="inherit" delete="inherit"/>
 9:       <edb:body>個人情報の EID</edb:body>
10:       <edb:passphrase type="MD5">暗号化パスフレーズ</edb:passphrase>
11:       <edb:passphrasestaff type="MD5">暗号化パスフレーズ</edb:passphrasestaff>
12:       <edb:passphraseodin type="MD5">暗号化パスフレーズ</edb:passphraseodin>
13:    </edb:auth>
```

4 行目の eid はこの認証情報の対象となる個人情報の EID である．また，eoid は authentication 内でのオブジェクトの通し番号であり，同一 EID の情報がある場合には，eoid の値が大きいものが選択される．

6 行目の変更時刻の UNIX タイムについては，初期情報作成時は 0 でよい．

7 行目の 443 はデータベースのマスタユーザの UID である．

9 行目にはこの認証情報の対象となる個人情報の EID をもう一度記述する．

10 行目は通常の認証に利用されるパスフレーズを MD5 で暗号化した文字列記述する．作成方法については，`libedb/auth0.c` の *edb_auth_make_pp( )* を見よ．

11 行目は EDB 上でスタッフの権限を有するユーザに対するパスフレーズである．staff でないユーザの場合には記述しない．(staff は任意の情報の閲覧，書換が可能である)

12 行目は EDB 上で全ての権限を有する Odin に対するパスフレーズである．Odin でないユーザの場合には記述しない．(Odin はテーブル定義の改変が可能である．データベース全体に 1 人か 2 人いればよい)

上述した 3 つのファイルの形式は，データベースダンプコマンド (edb_dump) が出力するファイルの形式に等しい．したがって，初期情報登録はデータベースリストアコマンド (edb_restore) で行う．

必要最小限の初期登録情報.

`edb_database.database-table.dump`

全てのテーブルの定義情報.

EDB で利用しているテーブル定義は `edb_database.database-table.dump`．ただし，このファイル中には実運用において割り当てられた EID および EOID が含まれているため，EID, EOID を残したまま他のデータベースの運用に用いるのは得策ではない．(つまり，EID, EOID は振り直した方がよい)

`edb_database.database-tuple.dump`

少なくとも 1 人の個人情報.

`edb_database.authentication.dump`

上記の `edb_database.database-tuple.dump` に登録した個人の認証情報．この認証情報には，staff, Odin のパスフレーズを与えておく．

第IV部

EDB/Dwarfについて

第11章　Dwarf

11.1　Dwarf の配置

本頁では，EDB のデータベース登録情報の管理において重要な役割を担っている Dwarf (edb_dwarf) について述べる．

edb_dwarf はオリジナルデータベースを除く全てのレプリカデータベース毎に配置される．言い換えると，一つの edb_dwarf プロセスは一つのレプリカデータベースを担当する．(オリジナルデータベースに対して起動する必要はない; オリジナルデータベースは RDB をもたないため)

すなわち，

- 参照するデータベース: `<source-db>`
- 担当するデータベース: `<db>`

の場合には，edb_dwarf は以下のような接続の元に配置される．

参照するデータベース`<source-db>`→ edb_dwarf →担当するデータベース`<db>`

11.2　Dwarf のコマンドライン引数

(USAGE) EDB# `edb_dwarf [options] <dst-db> <src-db>`

- `<dst-db>`…Dwarf が担当するデータベースのパス名．
- `<src-db>`…Dwarf が参照するデータベースのパス名．

  データベースのパス名は

  　　ホスト名:ポート番号:データベース名

  の形式で指定する．Dwarf が起動するホストとデータベースサーバが別のホストでも問題はない．ただし，`<dst-db>`については，データベースの書き換え権限が必要である．

- options
  - `-fast`

    edb_dwarf のアイドリングファクタを 0 にセットする．

    アイドリングファクタ (idling factor) は，edb_dwarf の実行速度を決定するものである．

    複数個のレプリカデータベースを蓄積するサーバにおいては，データベースの個数だけ edb_dwarf が起動されているはずである．この状態でデータベースに登録が行われると，一斉に edb_dwarf の開始される．故に，通常のデータベース参照の遅延を招くことになる．アイドリングファクタは，edb_dwarf が行う 1 単位の処理に対しての何単位のアイドリング時間をもたせるか指定する．

    アイドリングファクタ: 0 が最速である．
  - `-all`
  - `-latest`
  - `-university`
  - `-public`

    Dwarf の動作モードを指定するバラメータ．省略時は -all ．

11.3　Dwarf へのコマンド送信

バックグラウンドで実行されている Dwarf の調整を行うために，コマンド受信が用意されている．
コマンドの送信の仕方は，

- TERM シグナル…Dwarf の停止．

  EDB# `kill -TERM ‘cat /usr/local/edb/var/run/<db>.pid‘`
- HUP シグナル…コマンドの送信．

  EDB# `echo "command" > /usr/local/edb/var/cmd/<db>.cmd`

  EDB# `kill -HUP ‘cat /usr/local/edb/var/run/<db>.pid‘`

HUP シグナル経由で送信できるコマンド (*command*) は現在以下のとおり．

- stop…edb_dwarf を終了させる．(TERM シグナルと同じ動作)
- terminate…edb_dwarf を終了させる．(TERM シグナルと同じ動作)

- notify…<src-db>における NOTIFY の動作をエミュレートする．Dwarf を停止して何らかの作業を行った場合に，作業期間内に発生した NOTIFY を受け取らない (すなわち，情報の更新通知を受け取らない) ことになる．Dwarf を起動後，必要ならば NOTIFY を自身で発生させる．
- fast…アイドリングファクタ (idling factor) を 1 減らす．
- slow…アイドリングファクタ (idling factor) を 1 増やす．
- rotate…ログファイルを回転させる．
- adjust…全ての登録情報において Adjust(XMLDB から RDB への情報の展開作業) を再実行する．「adjust テーブル名」のようにテーブル名を指定するとそのテーブルに属する情報のみを対象とする．

(注) edb_dwarf が行う adjust 動作と edb_adjust が行う adjust 動作は異なる．

edb_adjust が行う adjust 動作は，単に XMLDB から RDB への 1 対 1 の展開であり，また，情報の参照記述や階層構造を展開しない．(これは，edb_adjust が実行される時点では，RDB に全ての情報が準備されていないことによる．) これに対して，edb_dwarf が行う adjust 動作はそれらの展開も行う．したがって，データベースを構築し直したときには，edb_adjust を実行した後に，edb_dwarf を起動し，adjust コマンドを送信して，RDB への情報の展開を再実行するべきである．

11.4　データベース間 (src-db → dst-db) の情報転送

Dwarf は動作モードにより下記のように情報の反映の仕方を変更する．

- -all…<src-db>の全ての内容を<dst-db>へ反映させる．
- -latest…<src-db>の中で
  - 各情報の最新の内容 (履歴の最後の部分) のみ

  を<dst-db>へ反映させる．
- -university…<src-db>の中で
  - 各情報の最新の内容 (履歴の最後の部分) のみ
  - 有効な情報のみ
  - 学内に公開可能な内容のみ

を<dst-db>へ反映させる.

ただし，<dst-db>の XMLDB には補間した XML 表現の情報を蓄積する.

- -public…<src-db>の中で
  - 各情報の最新の内容 (履歴の最後の部分) のみ
  - 有効な情報のみ
  - 学外に公開可能な内容のみ

  を<dst-db>へ反映させる.

  ただし，<dst-db>の XMLDB には補間した XML 表現の情報を蓄積する.

- 補間した XML 表現

  EDB で定義されている情報の XML 表現において,

  1. エレメント属性 mapto による EID の指定.
  2. エレメント内の<edb:english>, <edb:japanese>, <edb:pronounce>の記述.

  は排他的である．補間した XML 表現では，1. が記述されている場合でも，冗長な情報として 2. を含める．また，情報全体のマップが行われている場合は，マップ先の情報の記述を冗長な表現として読み込む.

  補間された XML 表現においても mapto は保存されている.

11.5　担当データベース (dst-db) 内での XMLDB → RDB の情報の展開.

Dwarf は XMLDB に登録された XML 記述の内容を，リレーショナルデータベース (RDB) に展開する.
この展開に際して,

- ○リレーショナルデータベース上では，mapto を全て補間したテキストが登録される.

  参照形式 (mapto) で登録されている記述においてもテキストによる検索が可能とするため.

- ○【組織】情報のように階層化構造が定義されている情報に関しては，階層化の上位への参照の拡張を行う.

  例えば,

  - 徳島大学

- 工学部.(上位) = 徳島大学の EID
- 電気電子工学科.(上位) = 工学部の EID

のように情報が定義されている場合，XML 表現では直上の上位組織のみが mapto で記述されているのみであるが，これを RDB に展開する際に，

- 電気電子工学科.(上位) = 工学部の EID, 徳島大学の EID

のように拡張する．

この拡張は，RDB の種たる目的が検索であり，検索の速度を意識してのことである．

第Ⅴ部

EDB/PKI

第12章　概要

12.1　EDB/PKIの目的

- 公開鍵認証システムの構築
- EDB 利用者への公開鍵認証システムについての啓蒙と利用者の習熟

12.2　EDB/PKIの運営・特徴

- ○ **Private CA (Certificate Authority) での運営**
  - 一般にオーサライズされた CA の運用には膨大な資金が必要.
  - 利用範囲を学内 (構成員) に限定するならば公的 CA と同等のサービスを提供できる.
- ○ **教職員を対象に認証システムを提供.**
  - 学生については高度情報化基盤センターの管轄.
  - EDB に蓄積された教職員の情報を有効に活用する.
- ○ **異種認証システムとの連係**
  - EDB の登録情報をベースに LDAP, RADIUS, Kerberos などの認証システムの登録情報を作成する.
  - 認証だけでなく，LDAP 等へのディレクトリ情報，DNS/BIND 等の登録情報も提供する.
  - 個人 (クライアント) と DNS の情報 (サーバ) の認証情報を管理，矛盾のない公開鍵基盤を形成する.

12.3　EDB/PKI での制約事項

- ○ **証明書の発行**
  - 証明書発行の対象は，EDB に対応する情報が登録されていなければならない．
  - 当面，証明書発行の対象は，【個人】，【擬人】，【ホスト】情報とする．
  - EDB の登録情報 (個人，擬人，ホスト) に対して各 1 通の証明書を発行する．(複数は発行しない．)
- ○ **証明書の属性**
  - 個人 (擬人) の証明書の CN (CommonName) は，「S’ 個人 (擬人) 情報の EID’」とし，認証システムに際してはこれがアカウント名となる．(先頭の ‘S’ はソースレベルで変更可能)
  - ホストの証明書の CN は，いわゆる FQDN (Fully Qualified Domain Name) とする．FQDN は IP アドレスが直接関係づけられていない ALIAS(別称) でも構わない．
- ○ **免責事項**
  - 一般の PKI に対して，EDB/PKI の公開鍵基盤の信頼性 (証明書発行対象が正しく証明書を利用しているかの検査機構，秘密鍵の保管形態等の保障，個人やサーバ管理者に対する啓蒙) は劣っている．EDB/PKI の提供する公開鍵基盤を過信する事の無いようにお願いしたい．

第13章　実装

13.1　site.h

```
#define EDB_MANAGE_PKI     /* EDB/PKI において CA の機能を有効にする際に定義する */
#define EDB_PKI_DIR EDBDIR "/libdata/pki"     /* EDB/PKI のデータをおくディレクトリ */
#define EDB_PKI_RSA_BITs 2048     /* RSA 鍵の長さ */
#define EDB_PKI_CERTIFICATE_DAYs 3650     /* 証明書の有効日数 */
#define EDB_PKI_DN_C "JP"     /* DN の Country */
#define EDB_PKI_DN_ST "Tokushima"     /* DN の State */
#define EDB_PKI_DN_L "Tokushima City"     /* DN の City */
#define EDB_PKI_DN_O "The University of Tokushima"     /* DN の Organization */
#define EDB_PKI_DN_OU NAME_OF_DB     /* DN の OrganizationalUnit */
```

13.2　ディレクトリ構造

```
EDB_PKI_DIR/
   root.crt     /* ルート証明書 */
   CA/
      .rnd     /* 乱数ファイル */
      ca.lock     /* CA のロックファイル */
      cacert.pem     /* CA の証明書 */
      etc/
         xpextensions     /* 拡張オプション定義ファイル */
      index.txt     /* 発行済み証明書のインデックス */
      index.txt.attr
      newcerts/     /* CA が発行した証明書を保存するディレクトリ */
```

```
newkeys/     /* CA が作成した秘密鍵を一時的に格納するディレクトリ */
newreqs/     /* 証明書要求を入れるディレクトリ */
openssl.cnf     /* CA 用の設定ファイル */
private/
   cakey.pem     /* CA の秘密鍵 */
   pass     /* CA の秘密鍵の復号鍵 */
serial     /* 発行証明書のシリアル */
```

13.3　pki.h

```
#define EDB_PKI_CA_DIR EDB_PKI_DIR "/CA"     /* CA のデータを管理するディレクトリ */
#define EDB_PKI_CA_LOCKFILE EDB_PKI_CA_DIR "/ca.lock"     /* CA のロックファイル */
#define EDB_PKI_CA_CERT_INDEX EDB_PKI_DIR "/CA/index.txt"     /* 発行済み証明書のインデックスファイル */
#define EDB_PKI_OPENSSL_PATH "/usr/bin/openssl"     /* openssl コマンドの場所 */
```

ca.lock をユーザ edb がロックできる状態にしておくこと．このファイルにより排他的な CA の処理を行う．なお，ca.lock が存在しない場合，*edb_pki_IamCA* は負論理を返す．

CA が発行する証明書は以下の 3 種類がある．

```
#define EDB_PKI_CERTIFICATE_MODE_USER 0
#define EDB_PKI_CERTIFICATE_MODE_CLIENT 1
#define EDB_PKI_CERTIFICATE_MODE_SERVER 2
```

openssl を利用して証明書を発行する際の引数を以下のマクロで定義している．

```
#define EDB_PKI_CERTIFICATE_EXT_USER "-extensions user_ext -extfile " EDB_PKI_CA_DIR "/etc/xpextensions"
#define EDB_PKI_CERTIFICATE_EXT_CLIENT "-extensions user_ext -extfile " EDB_PKI_CA_DIR "/etc/xpextensions"
#define EDB_PKI_CERTIFICATE_EXT_SERVER "-extensions user_ext -extfile " EDB_PKI_CA_DIR "/etc/xpextensions"
```

EDB/PKI では，ルート証明書と CA 証明書を下記のように別々に定義しているが，EDB/PKI の CA がルートでない場合には，これらの内容を調整する．

```
#define EDB_PKI_ROOT_CRT EDB_PKI_DIR "/root.crt" #define EDB_PKI_CA_CRT EDB_PKI_DIR "/CA/cacert.pem"
```

Synopsis :

struct EDB_PKI_X509_M_INFO *x509_m ;

Description :

Structure — DN 情報の詳細

EDB_PKI_X509_DN_INFO

Synopsis :

struct EDB_PKI_X509_DN_INFO *x509_dn ;

Description :

```
char *C;
char *ST;
char *L;
char *O;
char *OU;
char *CN;
char *Email;
```

Structure — DN 情報

EDB_PKI_X509_NAME_INFO
struct EDB_PKI_X509_NAME_INFO

Synopsis :

struct EDB_PKI_X509_NAME_INFO *name ;

typedef struct EDB_PKI_X509_NAME_INFO *EdbPkiNameInfo ;

Description :

```
char *DirName;
struct EDB_PKI_X509_DN_INFO DN;
```

Structure — X.509 証明書の情報

EDB_PKI_CERTIFICATE_INFO
struct EDB_PKI_CERTIFICATE_INFO

Synopsis :

struct EDB_PKI_CERTIFICATE_INFO *x509_info ;

typedef struct EDB_PKI_CERTIFICATE_INFO *EdbPkiCertificateInfo ;

Description :

```
struct EDB_PKI_X509_M_INFO M;
struct EDB_PKI_X509_NAME_INFO I;
struct EDB_PKI_X509_NAME_INFO S;
int mode;
```

13.4　関数

13.4.1　CA 管理

FUNCTION — CA の機能保持

edb_pki_IamCA

SYNOPSIS :

int
edb_pki_IamCA　(EdbContext ec)

DESCRIPTION :

CA の機能が実装されているかを調べる．具体的には，

- EDB_MANAGE_PKI が定義された状態でコンパイルされた実行ファイルがインストールされている．
- CA のロックファイルが存在し，Read/Write 許可されている．

を調べる．

RETURN VALUES :

CA の機能を持つか否かの論理値．

FUNCTION — CA のロック/アンロック

edb_pki_CA_lock
edb_pki_CA_unlock

SYNOPSIS :

int
edb_pki_CA_lock　(EdbContext ec)

int
edb_pki_CA_unlock　(EdbContext ec)

DESCRIPTION :

証明書発行手続きを排他的に行うためのロック処理．

FUNCTION — CA の管理関数

edb_pki_CA_certificate_index
edb_pki_CA_save_certificate_request
edb_pki_CA_make_certificate_request
edb_pki_CA_make_certificate
edb_pki_CA_output_PKCS12
edb_pki_CA_unlink_keyfile
edb_pki_CA_revoke_certificate
edb_pki_CA_certificate_serial

SYNOPSIS :

Link
edb_pki_CA_certificate_index　(EdbContext ec)

int
edb_pki_CA_save_certificate_request　(EdbContext ec, char *, char *, int)

int
edb_pki_CA_make_certificate_request　(EdbContext ec, char *, char *)

char *

edb_pki_CA_make_certificate　(EdbContext ec, char *, char *, int)

void
edb_pki_CA_output_PKCS12　(EdbContext ec, char *, char *, FILE *)

void
edb_pki_CA_unlink_keyfile　(EdbContext ec, char *)

int
edb_pki_CA_revoke_certificate　(EdbContext ec, char *)

char *
edb_pki_CA_certificate_serial　(EdbContext ec)

Description :

13.4.2　その他

Function — 証明書の情報を出力する

edb_pki_print_certificate_S_DN
edb_pki_print_certificate_ex
edb_pki_print_certificate
edb_pki_print_certificate_info
edb_pki_check_certificate

Synopsis :

int
edb_pki_print_certificate_S_DN　(EdbContext, char *, Doc)

int
edb_pki_print_certificate_ex　(EdbContext, char *, Doc)

int
edb_pki_print_certificate　(EdbContext, char *, Doc)

int
edb_pki_print_certificate_info　(EdbContext, EdbPkiCertificateInfo, Doc)

int
edb_pki_check_certificate　(EdbContext, char *)

Description :

Function — タプルに証明書を発行できるかどうかを調べる

edb_pki_tuple_certifiable

Synopsis :

EdbColumnInfo
edb_pki_tuple_certifiable　(EdbContext, EdbTupleInfo)

Description :

FUNCTION — タプルの証明書を取得する　edb_pki_tuple_certificate

SYNOPSIS :

```
char *
edb_pki_tuple_certificate　( EdbContext, EdbTupleInfo )
```

DESCRIPTION :

FUNCTION — 証明書情報を開放する　edb_pki_certificate_info_free

SYNOPSIS :

```
void
edb_pki_certificate_info_free　( EdbContext, EdbPkiCertificateInfo )
```

DESCRIPTION :

FUNCTION — 証明書情報を開放する　edb_pki_certificate_info

SYNOPSIS :

```
EdbPkiCertificateInfo
edb_pki_certificate_info　( EdbContext, char * )
```

DESCRIPTION :

FUNCTION — SSL の環境変数から証明書情報を作成する　edb_pki_SSL_env_info

SYNOPSIS :

```
EdbPkiCertificateInfo
edb_pki_SSL_env_info　( EdbContext )
```

DESCRIPTION :

FUNCTION — 証明書情報を比較する　edb_pki_certificate_info_equal

SYNOPSIS :

```
int
edb_pki_certificate_info_equal　( EdbContext, EdbPkiCertificateInfo, EdbPkiCertificateInfo )
```

DESCRIPTION :

FUNCTION — CN から EID を得る　edb_pki_person_CN2EID
edb_pki_personification_CN2EID

SYNOPSIS :

```
eid_t
edb_pki_person_CN2EID    ( EdbContext, char * )

eid_t
edb_pki_personification_CN2EID    ( EdbContext, char * )
```

DESCRIPTION :

第 VI 部

EDB と外界

第14章　外部からの情報参照

14.1　WWW/CGI

14.1.1　閲覧

EDB の外部のメディアから，EDB の登録情報を閲覧する場合には，当該箇所に

https://web.db.tokushima-u.ac.jp/cgi-bin/edb_browse?EID=*eid*

のリンクを埋め込んでおく．ただし，*eid* は閲覧したい情報の情報識別子 (EID) である．
eid に指定できるのは，テーブルを含めた情報の有効な EID または 0 である．0 を指定した場合には，閲覧のトップ頁が表示される．

https://web.db.tokushima-u.ac.jp/cgi-bin/edb_browse?EID=0

EID によらず，テーブル情報を XML 名で指定したい場合には，

https://web.db.tokushima-u.ac.jp/cgi-bin/edb_browse?XN=*xmlname*

のように指定する．ただし，*xmlname* はテーブルの XML 名である．

https://web.db.tokushima-u.ac.jp/cgi-bin/edb_browse?XN=person

https://web.db.tokushima-u.ac.jp/cgi-bin/edb_browse?XN=article

これは EID を指定するのと同じであるが，XML 名の方が識別性が高いのでお薦めである．なお，この方法はタプル情報には利用できない．

14.1.2　【画像】情報の参照

EDB に登録された【画像】情報を画像として参照するためには，

https://web.db.tokushima-u.ac.jp/cgi-bin/edb_browse?ACT=PICTURE&EID=*eid*

のリンクを<img src="...">に埋め込んでおく．ただし，*eid* は参照したい【画像】情報の情報識別子 (EID) である．
なお，【画像】データを指定したフォーマットで取り出したい場合には，

https://web.db.tokushima-u.ac.jp/cgi-bin/edb_browse?ACT=PICTURE&EID=*eid*&D_ARG1=*fmt*

のように，出力フォーマットを指定する．現在，*fmt* に指定できるのは，

- JPEG ... JPEG/JFIF format
- PNG ... Portable Network Graphics format
- EPS ... Encapsulated PostScript format

である．

14.1.3　編集

EDB の外部のメディアから，EDB の編集プログラムを起動する方法．

https://web.db.tokushima-u.ac.jp/cgi-bin/edb_edit?ACT=EDITOR_OPEN&EID=*eid*

のリンクを埋め込んでおく．ただし，*eid* は編集したい情報の情報識別子 (EID) である．
eid に指定できるのは，テーブルを含めた情報の有効な EID である．
もちろん，編集を行うことが出来るのは EDB にログインできるユーザで，かつ対象の情報が編集可能なユーザに限定される．
新規に情報を作成する場合には，

https://web.db.tokushima-u.ac.jp/cgi-bin/edb_edit?ACT=TUPLE_NEW&EID=*eid*

とする．*eid* はテーブルの EID であり，XN で指定することが出来る．

https://web.db.tokushima-u.ac.jp/cgi-bin/edb_edit?ACT=TUPLE_NEW&XN=*xmlname*

ACT=TUPLE_NEW では引数 D_ARG1 を指定出来る．引数 D_ARG1 は，

- 整数 ⇒ 新規情報を作成する際に内容をコピーする情報の EID．EID は *xmlname* に属する情報でなくてはならない．
- 整数以外の文字列 ⇒
  - 「<?xml version="1.0" encoding="utf-8"?>」で始まる文字列．⇒ XML で記述された情報全体．

- 「<edb:base 」で始まる文字列．⇒「<edb:base ...」以降の XML 記述．ただし，最期の「</edb:*xmlname*>」は含まれない．
- 上記以外の文字列．⇒ base より後の XML 記述．ただし，最期の「</edb:*xmlname*>」は含まれない．

と解釈される．

初期エディタ画面をどのモードで起動するかは，以下で指定できる．

- EDITOR_MODE
  - EDITOR_MODE=ENTRY … 入力欄なし (ユーザ操作で，モードを選択する)
  - EDITOR_MODE=ALL … 全項目一括入力モード
  - EDITOR_MODE=COLUMN … 項目モード
    項目モードを選択したときには，COL=*xmlname* で対象カラムを指定する．

14.2　whois/TCP

外部からバッチ的に情報にアクセスできるように whois/TCP による接続が可能である．

14.3　EDB/Gate

14.3.1　Motivation

EDB へのアプリケーションレベルでのコネクション指向アクセスインタフェースを提供する．

14.3.2　接続方法

以下に，EDB/Gate への接続に必要な諸元を示す．EDB/Gate はクライアントサイドのアプリケーションからの接続を想定しているが，接続例のようにマニュアルでも接続は可能である．

- db.db.tokushima-u.ac.jp : 44443
  - SSL(SSLv3/TLSv1) automatic authentication mode.
  - ユーザの個人証明書を提示し，検証が成功し，かつ自動ログイン設定されている場合．
  - プライマリデータベースを参照する．

  - サービスエリア: 0.0.0.0/0
  - 接続コマンド例:

    ```
    % openssl s_client -cert your-certificate.pem -key your-certificate-key.pem -CAfile CA-certificate.pem -quiet -nbio
    -connect db.db.tokushima-u.ac.jp:44443
    ```

- db.db.tokushima-u.ac.jp : 44443
  - SSL(SSLv3/TLSv1) connection mode.
  - ユーザの個人証明書を提示しない場合.
  - ログインした場合には，プライマリデータベースを参照する．ログインするまでは，公開用 (学内，学外) のデータベースを参照する.
  - サービスエリア: 150.59.0.0/16
  - 接続コマンド例:

    ```
    % openssl s_client -CAfile CA-certificate.pem -quiet -nbio -connect db.db.tokushima-u.ac.jp:44443
    ```

- db.db.tokushima-u.ac.jp : 44080
  - without authentication.
  - Referencing the university/public readable database as backend. So you can get only information which is readable from university or world wide.
  - サービスエリア: 150.59.0.0/16
  - 接続コマンド例:

    ```
    % telnet db.db.tokushima-u.ac.jp 44080
    ```

14.3.3 Character Encoding

全ての通信データは，Unicode (UTF-8 encoding) で表現された文字列で行われる.

14.3.4 Scheme of Request/Response

EDB/Gate はクライアントサイドのアプリケーションとの接続を想定しているので，telnet のようなコマンドプロンプトは提示しない. その代わりに，クライアントからのリクエストを *request-id* によって管理し，レスポンスにそれらを付加して応答を行う.

リクエストの発行方法と，レスポンスの様式は以下に示す通り.

○ **<Request Form>**

COMMAND　arg_1　[arg_2　[arg_3 ...]]

$OPTION_1$　arg_1　[arg_2　[arg_3 ...]]

$OPTION_2$　arg_1　[arg_2　[arg_3 ...]]

...

(insert a blank line: blank line means the end-of-request)

○ **<Response Form>**

— BEGIN-OF-RESPONSE (*request-id*)

request-id(TAB)　Status: *3-digits-code*　*comment*

request-id(TAB)　$result\text{-}information_1$

request-id(TAB)　$result\text{-}information_2$

request-id(TAB)　...

request-id(TAB)　(blank)

request-id(TAB)　$content\text{-}of\text{-}response_1$

request-id(TAB)　$content\text{-}of\text{-}response_2$

request-id(TAB)　$content\text{-}of\text{-}response_3$

request-id(TAB)　$content\text{-}of\text{-}response_4$

request-id(TAB)　...

— END-OF-RESPONSE (*request-id*)

○ リクエストのブロックの終了は空行 (改行コードのみの行) で示す.
EDB/Gate は空行が現れたとき，コマンドの解析と実行を開始する.

○ リクエストは，レスポンスの終了を待たなくても，発行することができる.
ただし，ソケット接続においてブロックされないことが必要.

○ EDB/Gate からのレスポンスについて
EDB/Gate からのレスポンスは,

— BEGIN-OF-RESPONSE (*request-id*)

— END-OF-RESPONSE (*request-id*)

でブロック化されて返送される．

このブロック内に含まれる行には，

1. 行頭の '!' はその行が注釈である事を示す．
2. 行頭に *request-id*(TAB) が記述されている行

があり，前者は EDB/Gate のデバッグ用もしくは，内部処理を確認するためのものであるので，無視する．

後者は更に，

1. レスポンスヘッダ
2. コンテンツ

に分けられ，これらは，

request-id(TAB)

のみの行で区切られる．レスポンスヘッダにはリクエストの成功の可否やコンテンツの長さなどが示される．コマンドが何らかの情報を取得する場合には，コンテンツの場所において情報が返送される．

インタラクティブにコマンドとレスポンスを繰り返す場合には，*request-id* のフィールドに書かれている ID を無視することも可能であるが，コマンドを同時に発行するような場合には，ID 部分によりレスポンスを選別した後，各々のレスポンスの内容を解析する必要がある．

なお，通常 *request-id* はサーバ側で自動的に生成される．1 ずつ増える値となるが，クライアント側から *request-id* をリクエストのオプションで指定する事も可能である．

Request-ID: *request-id*

('*request-id*' は正の整数である事)

request-id を指定すると，それ以降の値は指定した *request-id* に続くものになる．

14.3.5　Command Summary

Connection Management

LOGIN — データベースにログインする．(EDB/SSL mode only)

(書式) LOGIN　*user-eid*　*passphrase*

(返答) (none)

SLOGIN — データベースにスタッフ権限でログインする．(LOGIN state only)

(書式) SLOGIN　*passphrase*

(返答) (none)

SLOGOUT — スタッフ権限からログアウトする．(STAFF state only)

(書式) SLOGOUT

(返答) (none)

ENV — コネクション状態を表示する．

(書式) ENV

(返答) environment variables and their values

VERBOSE — VERBOSE レベル (≥ 0) の設定．

(書式) VERBOSE　[*level*]

(返答) (none)

WHOAMI — 接続しているユーザの情報を得る．

(書式) WHOAMI

(返答) 形式 1

QUIT — コネクションを切断する．

(書式) QUIT

(返答) (none)

HELP — ヘルプメッセージを表示する.

(書式) HELP　[*cmd*]

(返答) summary of command usage

BEGIN — トランザクションを開始する.

(書式) BEGIN

(返答) (none)

END — トランザクションを終了する.

(書式) END

(返答) (none)

Retrive Information

TABLE — テーブルのリストを得る.

(書式) TABLE　　テーブルの全リストを得る.

(返答) 形式 2

(書式) TABLE　regular　　標準テーブルのリストを得る.

(書式) TABLE　auxiliary　　補助テーブルのリストを得る.

AGE — 全情報について最大の EOID を得る.

(書式) AGE

(返答) EOID

(書式) AGE　*eid*

eid で示される情報について，参照まで含めた EOID の最大値を得る．この値が増加している場合には，補間後の情報 (cXML; completed XML) の記述が変更された可能性がある.

INFO — 情報得る.

(書式) INFO MAX(eid)

(返答) EID

全情報について EID の最大値を得る．

(書式) INFO MAX(eoid)

(返答) EOID

全情報について EOID の最大値を得る．

GET — テーブル/タプルの XML 形式の情報を得る．

(書式) GET　*eid-of-tuple*　[*eoid*]

eid-of-tuple で指定された情報の XML 形式の登録情報を得る．*eoid* が指定された場合には，過去の変更履歴中の XML 形式の情報を得る．

(返答) 形式 3

(書式) GET　*eid-of-table*

eid-of-table で指定されたテーブルの XML 形式の定義情報を得る．

(書式) GET　*xmlname*

xmlname で指定されたテーブルの XML 形式の定義情報を得る．

FRESH — 最近に更新された情報のリストを得る．

(書式) FRESH　*seconds*

現在から遡って *seconds* 秒以内に更新された情報のリストを得る．

(返答) 形式 1

DTD — DTD の定義を得る．

(書式) DTD　*xmlname*

xmlname で指定された DTD の内容を得る．

(返答) DTD の内容

HISTORY — *eid* で指定された情報の変更履歴のリストを得る．

(書式) HISTORY　*eid* (非 LOGIN 状態では変更履歴は参照できない．最新のもののみ返される．)

(返答) 形式 1

LOOK — 条件 (*condition*) による検索を行う.

(書式) LOOK　*condition*

(返答) 形式 1

オプションとして Order を利用可.

COUNT — 条件 (*condition*) に適合する情報の数を得る.

(書式) COUNT　*condition*

(返答) 整数値

TRCOUNT — すべてのテーブルについて eid を参照している情報の数を得る.

(書式) TRCOUNT　*eid*

(返答) 形式 1+整数値

TOCOUNT — すべてのテーブルについて eid が所有している情報の数を得る.

(書式) TOCOUNT　*eid*

(返答) 形式 1+整数値

TPCOUNT — すべてのテーブルについて eid が権限をもつ情報の数を得る.

(書式) TPCOUNT　*eid*

(返答) 形式 1+整数値

TMCOUNT — すべてのテーブルについて eid をマップしている情報の数を得る.

(書式) TMCOUNT　*eid*

(返答) 形式 1+整数値

CLASSIFY — 条件 (*condition*) に適合する情報をカラム (*cxn*) で分類する.

(書式) CLASSIFY　*cxn*　*condition*

(返答) EID + 整数値 または整数値 + 整数値

オプションとして Mode: fiscal を利用可.

OWNERSHIP — テーブル (*xmlname*) の登録情報においてユーザ (*eid*) が (直接) 所有する情報のリストを得る.

(書式) OWNERSHIP　*xmlname*　*eid*

(返答) 形式 1

REFER — テーブル (*xmlname*) の登録情報において情報 (*eid*) が (直接) 参照されている情報のリストを得る.

(書式) REFER　*xmlname*　*eid*

(返答) 形式 1

CAPTION — 情報の見出しを得る.

(書式) CAPTION　*eid*

(返答) 形式 4

EXPAND — 情報を展開する.

(書式) EXPAND　*eid*

(返答) 形式 1

Editting Information

(required LOGIN-state)

EDITOR — 情報 (*eid*) を編集可能なユーザのリストを得る.

(書式) EDITOR　*eid*

(返答) 形式 1

NOTIFY — 電子メールによる変更通知の可否を設定する. (required STAFF-state)

(書式) NOTIFY　immediate

(書式) NOTIFY　delay

(書式) NOTIFY　off

(返答) (none)

CHECK — XML 記述を検査する.

(書式) CHECK *xml*

(返答) checked status

MODIFY — XML 記述を修正する.

(書式) MODIFY *xml*

(返答) modified XML

UPDATABLE — XML 記述が更新可能な内容であるかを検査する.

(書式) UPDATABLE *xml*

(返答) checked status

UPDATE — XML 形式で与えられた情報をデータベースに登録する.

(書式) UPDATE *xml*

(返答) 形式 1

CREATABLE — 指定されたテーブルにおいて新規情報作成の可否を得る.

(書式) CREATABLE *xn*

(書式) CREATABLE *eid*

(返答) boolean

WRITABLE — 指定された情報の編集の可否を得る.

(書式) WRITABLE *eid*

(返答) boolean

DELETABLE — 指定された情報の削除の可否を得る.

(書式) DELETABLE *eid*

(返答) boolean

ISUSER — 指定された情報 (person) がユーザ権限を持つかどうか判定する.

(書式) ISUSER　*eid*

(返答) boolean

ISSTAFF — 指定された情報 (person) がスタッフ権限を持つかどうか判定する.

(書式) ISSTAFF　*eid*

(返答) boolean

ISODIN — 指定された情報 (person) が ODIN 権限を持つかどうか判定する.

(書式) ISODIN　*eid*

(返答) boolean

PASSPHRASE — ログインパスフレーズを変更する.

(書式) PASSPHRASE　*passphrase*

(返答) boolean

Miscellaneous

CTEXT — テキスト (*text*) に文字変換テーブルによる変換を施す.

(書式) CTEXT　*text*

(返答) converted text.

CTABLE — 文字変換テーブルをダンプする.

(書式) CTABLE

(返答) conversion table.

14.3.6　Format of Response

Format	Fields
形式 1	*request-id* EID EOID TABLE-EID
形式 2	*request-id* EID EOID TABLE-EID xmlname
形式 3	*request-id* EID EOID TABLE-EID XML
形式 4	*request-id* EID EOID TABLE-EID 短見出し 長見出し

各フィールドは TAB 文字 (U+0009) で区切られる．(フィールド内のテキストには TAB 文字は含まれない．)

14.3.7　Condition

- 検索する情報に対する条件式を記述する．

○ 基本形は

$$XN = \{values\}$$

である．

- *XN* はテーブルもしくはカラムの XML 名である．
- 「$XN = xn_{\mathrm{tbl}}\ .\ xn_1\ .\ xn_2$」のように展開し，$xn_{\mathrm{tbl}}$ をテーブルの XML 名とする．
- 結果はカラムの値と条件の値 (*values*) が合致する情報の EID のリストとなる．
- *values* に記したどれか一つの値と適合する場合に EID のリストに追加される．(OR 結合)
- デフォルトのロジックを変更したい場合には，

$$XN = \mathrm{NOT}\{values\}$$
$$XN = \mathrm{AND}\{values\}$$
$$XN = \mathbf{OR}\{values\}$$
$$XN = \mathrm{NAND}\{values\}$$
$$XN = \mathrm{NOR}\{values\}$$
$$XN = \mathrm{XOR}\{values\}$$
$$XN = \mathrm{XNOR}\{values\}$$

のように記述する．NOT は NOR と解釈される．XOR, XNOR の場合には値は 2 つでなくてはならない．

- xn_i $(i > 0)$ では”xmlname[1]”のように要素位置を指定できる．(配列の場合のみ)

○ 複数のカラムについて条件を指定したい場合には

$XN_0.\{\ XN_1 = \{values_1\}\quad XN_2 = \{values_2\}\quad [...]\ \}$

と記述する．

- 一般に，XN_1, XN_2 は XN_0 に含まれるカラムである．
- これらの条件式は AND で結合される．
- デフォルトのロジックを変更したい場合には，

$XN_0.\mathrm{NOT}\{\ XN_1 = \{values_1\}\quad XN_2 = \{values_2\}\quad [...]\ \}$
$XN_0.\mathbf{AND}\{\ XN_1 = \{values_1\}\quad XN_2 = \{values_2\}\quad [...]\ \}$
$XN_0.\mathrm{OR}\{\ XN_1 = \{values_1\}\quad XN_2 = \{values_2\}\quad [...]\ \}$
$XN_0.\mathrm{NAND}\{\ XN_1 = \{values_1\}\quad XN_2 = \{values_2\}\quad [...]\ \}$
$XN_0.\mathrm{NOR}\{\ XN_1 = \{values_1\}\quad XN_2 = \{values_2\}\quad [...]\ \}$
$XN_0.\mathrm{XOR}\{\ XN_1 = \{values_1\}\quad XN_2 = \{values_2\}\ \}$
$XN_0.\mathrm{XNOR}\{\ XN_1 = \{values_1\}\quad XN_2 = \{values_2\}\ \}$

のように記述する．NOT は NAND と解釈される．XOR, XNOR の場合には条件は 2 つでなくてはならない．

- XN_1, XN_2 では，‘@’ を用いて，$@.xn_1$, $@.xn_2$ のように表現することができる．
- カラム $(@.xn_1)$ が配列の場合には，

$xn_{\mathrm{tbl}}.\{\ @.xn_1.xn_2 = \{values_2\}\ @.xn_1.xn_3 = \{values_3\}\ \}$

と

$xn_{\mathrm{tbl}}.xn_1.\{\ @.xn_2 = \{values_2\}\ @.xn_3 = \{values_3\}\ \}$

は異なる結果になりうる可能性があることに注意せよ．

○ サブカラムに条件をつけた配列の条件を指定したい場合には

$XN_0.\{\ XN_1 = \{values_1\}\ \}=\{values_2\}$

と記述する．すなわち，

$xn_{\mathrm{tbl}}.xn_1.\{\ @.xn_2 = \{values_2\}\ \}\ =\{values_1\}$

のような記述では,

$xn_{\mathrm{tbl}}.xn_1[\mathrm{k}] = \{values_1\}$

$xn_{\mathrm{tbl}}.xn_1[\mathrm{k}].xn_2 = \{values_2\}$

が同時に成り立つ k が存在する時，真となる．xn_1 が配列でない場合には,

$xn_{\mathrm{tbl}}.\{@.xn_1 = \{values_1\}\ @.xn_1.xn_2 = \{values_2\}\ \}$

と書いても同じ結果となる.

○ xn_x…カラム名 (XML 名)

・ カラムが配列の場合には,

@.column[x] : 配列の添字 =x

@.column[x:] : x≤ 配列の添字

@.column[:y] : 配列の添字 ≤y

@.column[x:y] : x≤ 配列の添字 AND 配列の添字 ≤y

で範囲を指定することが可能．x, y は正整数.

・ 特別なカラム名:

EID … この情報の EID

REF … この { 情報—カラム } が参照している他の情報の EID のリスト

PERM … この { 情報—カラム } が権限委譲している他の情報の EID のリスト

OWN … この情報の所有者の EID

MAP … この情報がマップしている情報の EID

DATE … カラムの期間限定属性

CAP … この情報の見出し

ALL … この情報の全文

PARENT … 情報が階層構造をなしているときの直上の親

○ オペレータ:

'=' … 一致

'~' … 正規表現による一致

を指定できる．

- 正規表現による一致は，対象となるカラムがテキスト型を含みかつ比較される値がテキストが他の場合にのみ行われる．

○ 値リスト: LOGIC{ *values* }

- LOGIC には，NOT, AND, **OR**(デフォルト), NAND, NOR, XOR, XNOR を指定できる．
- LOGIC を省略すると，OR として解釈される．
- LOGIC に NOT を指定すると，NOR と解釈される．
- LOGIC に XOR, XNOR を指定する場合には，*values* は 2 項でなくてはならない．
- LOGIC が OR であり，かつ，*values* が単項の場合には，'{' と'}' を省略することができる．

○ 値: $value_x$

- 値リストを { $value_1$ $value_2$ [...] } と記述した場合，それぞれの $value_x$ には以下の記法を用いることができる．

  !$value_x$ …単項の NOT 演算．

  ~$value_x$ …正規表現による比較 (テキストの比較のみ)．

  両方を使う場合には，!~$value_x$ の順に指定する．

- 指定する値の型は左辺から抽出可能なものでなくてはならない．

value の型	指定形式	説明	
存在	ANY	何かがあるとき．リレーショナルデータベースにおいて比較の対象となるものが存在するときに真値をとる．!ANY は使用不可．	
EID	\e{NULL}, \E{NULL}	参照なし．	
EID	\e{eid}	展開なし．	
EID	\E{eid}	展開あり．	
EID	サブクエリ	展開あり．	INTEGER, DATE
テキスト	"string"	登録内容に等しいテキスト．MTEXT のように左辺から複数のテキストを抽出可能な場合には，(英),(日),(読) のいずれかに等しいテキスト	
数値	"val"	登録内容に等しい数値．	
数値	"val_1 val_2"	もしくは登録内容を含む範囲．	
年月日	"date"	登録内容に等しい月日．	
年月日	"from-date to-date"	登録内容を含む期間．	

等を範囲で検索する場合には

column=”val_1 val_2”

と指定する．val_2 が省略された場合，完全一致とみなす．

DATE は西暦年月日 (YYYYMMDD; 年 4 桁, 月 2 桁, 日 2 桁) で指定する．

- 条件式の例
  - ○ 【著作】の [著者] で検索．

    article.author=\E{12345}

    article.author={ \E{12345} \E{12346} }

    article.author=”Tokushima Taro”

    article.author~”Tokushima Taro”

    article.author={ \E{12345} \E{12346} ”Tokushima Taro” }
  - ○ 【著作】の第 1[著者] で検索．

    article.author[1]={ \E{12345} \E{12346} }

    article.author[1]=”Tokushima Taro”
  - ○ 【著作】の第 1,2,3[著者] で検索．

    article.author[:3]={ \E{12345} \E{12346} }

    article.author[:3]=”Tokushima Taro”
  - ○ 【著作】の第 2,3[著者] で検索．

    article.author[2:3]={ \E{12345} \E{12346} }

    article.author[2:3]=”Tokushima Taro”
  - ○ 【著作】を組織 {23456} に属している [著者] で検索．

    article.author={ person.affiliation=\E{23456} }
  - ○ 上記に [発行年月日] の条件をつけて検索．

    article.{@.author={ person.affiliation=\E{23456} } @.date=”20040000 20049999”}
  - ○ 【組織】を期間限定属性で検索

    organization.DATE=”20050401”

14.3.8　Options

Order — 排列に利用するカラムを指定する．

Order: $column_1$ $column_2$ ...

先に書いたカラムが優先される．カラム名の前に'!' を記すと逆順となる．

14.3.9　Status Code

2xx	肯定的応答	処理が正常に終了したことを示す．
3xx	肯定的応答	処理の過程で何らかの修正がなされたことを示す．
4xx	否定的応答	要求にエラーがあったことを示す．
5xx	サーバのエラー	サーバ側で不可避なエラーが生じたことを示す．

14.3.10　Transaction

EDB/Gate のコマンドにはトランザクションの開始 (BEGIN) と終了 (END) が用意されているが，これは，バックエンドデータベースの SQL における"BEGIN", "END" に 1 対 1 で対応していない．以下の点で異なる．

- EDB/Gate でトランザクションに入っていなくても，EDB/Gate では各コマンド毎にバックエンドデータベースのトランザクションを開始，終了する．
- EDB/Gate でトランザクションに入っている場合には, バックエンドデータベースのトランザクションを維持するが, CHECK, MODIFY, UPDATABLE, UPDATE 等のプライマリデータベース*に影響する/の影響を受ける*コマンドにおいては，別にバックエンドデータベースへのチャネルを開いてトランザクションを開始し，処理を行うことがある．

14.3.11　Notice

サーバのリソースを節約するため，アイドル状態が約 60 分以上経過すると自動的に EDB/Gate は接続を切断する．

第15章　外部への情報参照

第16章　外部との情報交換

EDB のようなシステム間で，データのやりとりを実現することは重要な課題である．しかし，一般に各々のシステムは，独自のデータ管理様式をもっており，システム間でのデータの移動は困難である．このような問題は，個々のシステムのデータ管理機能の高度化させる程難しくなる．

本章では，EDB に限らずシステム間のデータ移動に関してのガイドラインを提案する．

16.1　まず最初に考えること

データの移動に際して，情報が変質してしまうことを避ける．

データの移動に際して情報量が減少する可能性を可能な限り回避する．

16.2　基本事項

大昔には，バイナリによるデータ表現が盛んであった．バイナリによる表現は，

- データ量 (バイト数) が節約できる．
- 誤差が少ない．

というメリットがある．半面，

- システムに依存しやすい．

というデメリットがある．しかし，今日に至ってはハードディスクやその他のメディアの容量が飛躍的に大容量化しているため，もはやデータ量を節約する必要はない．また，誤差についても桁数を十分大きくとることにより解消できる．

テキスト表現によるメリットは，

- データの可読性に富む．

- データの操作が比較的安易に行える.
- システムハードウェアに依存しない.

という点にある．したがって，システム間のデータの流通はテキスト表現で行うことを選択すべきである.

次に，テキストでデータを表現する際に選択すべきコードセットについて考える.

現在，日本語に特化した文字コードセットには,

- JIS コード (iso-2022-jp)
- シフト JIS コード (Shifted_JIS)
- EUC コード (EUC-JP)

がある．これらは，微細な部分で可換ではなく，また，他のオーバーセットにもなっていないため，これらの中から標準のコードセットを選ぶことは出来ない．また，近年の国際化に対して，これらと他の言語とのコード間の混在は，コードセットの定義としては可能なものの実際に多国語の言語を処理できるようなアプリケーションは非常に数が少ない.

近年，世界中の言語を統一して一つの文字コードセットにまとめる Unicode^{TM} コードセットを作成する試みが行われている．Unicode は，日本語を含め他の国々のコードをシフトイン/シフトアウトせずに表現し得るものであり，近年のアプリケーションプログラムの多くは，多国語のコードセットを混在して利用するよりは，Unicode コードセットによる文字コードの表現を選択している.

ただし，Unicode 文字コードセットが日本語のコードセットを完全に含み得るものかどうかについては否であるが，現時点で利用できるコードセットの中ではもっとも優れていると思われる.

システム間のデータ移動に関して，Unicode 文字セットの問題点を更に言及しておくと,

- 機種依存文字コードエリアの存在.
- 半角/全角文字の両立.
- 複合グリフの問題.

がある．残念ながら，Unicode コードセットにおいても，機種依存文字エリアが作成されている．このエリアは，オペレーティングシステムやアプリケーションが自由に文字を設定できるエリアであるが，システム間のデータの移動には障害になる．したがって，システム間のデータの移動の際には，この機種依存文字エリア

- U+0000e000∼U+0000f8ff
- U+000f0000∼U+000ffffd
- U+0x00100000∼U+0010fffd

は利用してはいけない．これらはデータ移動時に無視されることを前提とし，データを送出するシステム側で，

- これらの文字を含んでいるかどうかのチェックと代替文字への変換

を行うことが必須となる．

また，Unicode 文字セットでは初期の日本語ワードプロセッサの悪しき習慣であるところの，全角文字と半角文字の概念が残っている．すなわち，半角文字の「1」や「A」と全角文字の「１」や「Ａ」が異なる文字コードで表現される．これらの文字の意味的な部分は何ら異なるところがないにも関わらず，多くの処理系ではこれらの文字は異なる文字として処理されてしまう．データの移動，とくに数値データの移動を考えたとき，これらの文字の混在は予測しにくい効果を演出するかもしれない．また，何らかの検索処理においてこれらの文字コードの違いは決定的な検索洩れにつながる．ここで，データ移動に際して全角文字が Unicode において半角文字と同一の異なる文字グリフとして捉えられている場合には，

- 全角文字から半角文字への変換

が行われる可能性を予測して出力データを考えておくべきである．この問題は，複合文字グリフにおいても同様に存在し得る．これらの文字の変換がデータの移動に際して発生したとしても，変換によってデータ処理の手続きが異ならないように，システムはデータを設計すべきである．

エンコーディング方式は UTF-8 が望ましい．理由は，

- バイトオーダーの問題が発生しない．
- 文字列検索が可能である．

という点にある．

16.3　CSV (level-0)

現在最も巷に流通しているテキストベースのデータ表現に，CSV (Comma Separated Values) がある．

CSV はテキストによって値を表現し，

- 区切り文字としてカンマ (‘,’; U+002c) を利用する．値自体にカンマが含まれる場合には，値全体を二重引用符で囲む．
- 区切り文字として TAB (U+0009) を利用する．

等のバリエーションがある．多くのスプレッドシートアプリケーションは，テキストへの出力には，この形式を利用している．

概ねスプレッドシートアプリケーションが提供する CSV 出力で，CSV データ表現は可能であり，利用者にとってほとんど労力をかけずにデータを出力できる点は有用である．しかし，CSV にて出力されるものが何であるかを付加的な情報として明確に示しておかないと，受け取り側でそれらのデータをどのように解釈すれば良いのかがわからなくなってしまう．

CSV においてデータを EXPORT する場合には，各カラムが何を表しているかが IMPORT 側に伝わるように

```
column-definition-begin
1, 著者名,1, 説明文
2, 著者名,2, 説明文
3, 著者名,3, 説明文
4, 著者名,4, 説明文
5, 著者名,5, 説明文
6, 題名,1, 説明文
...
column-definition-end
CSV-begin
徳島太郎,,,,, 阿波踊りの歴史的意義,,,,
徳島次郎,,,,, 藍染め職人と文化交流,,,,
CSV-end
```

等のようにカラムの説明文をつけておくことが必要である.

CSV の難点は,

- スプレッドシートアプリケーションから CSV 形式にデータを出力する際に発生する情報の損失.
- 1 次元の情報表現に限定される.

等がある. 通常スプレッドシート上では, セル間でのデータの投影や計算などが行われるが, CSV への変換の際にそれらの情報はすべて欠落してしまう. また, スプレッドシート自体が各行を 1 次元のデータ並びで表現するものであるため, 出力されるデータも 1 次元での表現となってしまう.

様々なデータを考えた場合に, 1 次元でのデータ表現は困難を伴う.

16.4　行分割 (レベル-1)

行による分割

16.5　XML (level-2)

XML

第VII部

付録

第17章　FreeBSD5.3での作業記録 (Dec. 28. 2004)

これは，FreeBSD5.3 + PostgreSQL 7.4.6 をインストールする際の作業記録である．

- FreeBSD5.3 のインストール
  1. linux_base-7.1.7
- ROOT# `mkdir /usr/local/share/cvsup`
- ROOT# `cp /usr/share/examples/cvsup/* /usr/local/share/cvsup/.`
- /etc/make.conf に下記を追加
  - `CPUTYPE?=p4` ; ... Pentium IV
  - `SUP_UPDATE= yes`
  - `SUP= /usr/local/bin/cvsup`
  - `SUPFLAGS= -g -L 2`
  - `SUPHOST= cvsup.db.tokushima-u.ac.jp`
  - `SUPFILE= /usr/local/share/cvsup/standard-supfile`
  - `PORTSSUPFILE= /usr/local/share/cvsup/ports-supfile`
  - `DOCSUPFILE= /usr/local/share/cvsup/doc-supfile`
  - `MASTER_SITE_OVERRIDE?= ftp://ftp.db.tokushima-u.ac.jp/pub/FreeBSD/ports/distfiles/${DIST_SUBDIR}/`

  cvsup, ftp についてはサイト毎に適切なサーバを設定すること．
- Ports Collection のインストール
  1. net/cvsup-without-gui (cvsup-without-gui-16.1.h)

- ROOT# `cd /usr/src ; make update`
- ROOT# `cd /usr/src ; make buildworld`
- ROOT# `cd /usr/src ; make buildkernel KERNCONF=`*`kernelconf`*
- ROOT# `cd /usr/src ; make installkernel KERNCONF=`*`kernelconf`*
- (reboot)
- ROOT# `cd /usr/src ; make installworld`
- (reboot)
- Ports Collection のインストール

  1. japanese/less (ja-less+iso-358.254_2)
  2. japanese/nkf (ja-nkf-2.04)
  3. devel/gmake (gmake-3.80_2)
  4. databases/postgresql7 (postgresql-7.4.6)
  5. textproc/libxml2 (libxml2-2.6.16)
  6. graphics/jpeg (jpeg-6b_3)
  7. graphics/netpbm (netpbm-10.25_3)
  8. www/apache2

     apache-2.0.52; make options:

     WITH_SUEXEC=yes
     SUEXEC_DOCROOT=/usr/local/www
     SUEXEC_UIDMIN=443
     SUEXEC_GIDMIN=443

- edb(user, group) の追加

  1. `/etc/passwd:  edb:*:443:443::0:0:EDB Owner:/usr/local/edb:/bin/csh`
  2. `/etc/group:  edb:*:443:root,edb`

- edb directory の作成 (user: edb, group:edb)
  1. /usr/local/edb
  2. /usr/local/edb/bin, etc, include, lib, libdata, libexec, sbin, tmp, var
  3. /usr/local/edb/include/edb
  4. /usr/local/edb/libdata/dtds, font, js, ml, msg, tbl
  5. /usr/local/edb/libexec/cron, ml, pgsql
  6. /usr/local/edb/var/cmd, log, run
- DB の作成 (PGDATA=/usr/local/psql/data)
  1. PGSQL# `PGDATA=/usr/local/pgsql/data`
  2. PGSQL# `export PGDATA`
  3. PGSQL# `initdb -E UNICODE`
  4. (running postmaster)
  5. PGSQL# `createuser --adduser --createdb edb`
  6. EDB# `createdb edb`
- kernel の tune up
  1. `options MAXDSIZ="(1536*1024*1024)"`
  2. `options MAXSSIZ="(1536*1024*1024)"`
  3. `options DFLDSIZ="(1536*1024*1024)"`
  4. `options SHMMAXPGS=327680`
  5. `options SEMMNI=320`
  6. `options SEMMNS=1920`
  7. `options SEMUME=320`
  8. `options SEMMNU=960`
  9. `options PMAP_SHPGPERPROC=400`
- /etc/sysctl.conf

1. `kern.maxfiles=65536`

- EDB の作成

1. EDB# `make-edb-template.sh`

- DB ユーザの作成

1. PGSQL# `CREATE USER guest WITH PASSWORD 'guest';`

- WWW サーバの設定と WWW コンテンツの作成

1. SSL
2. Cookie
3. Log directory
4. WWW# `cd /usr/local/www/data ; mkdir edb-icons`
5. (store *.png into edb-icons/.)
6. `/etc/newsyslog.conf:  + /var/log/httpd-suexec.log 644 5 1000 * Z`

第VIII部

索引

索引