日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2021年1月25日月曜日

クイックSQLのすすめ

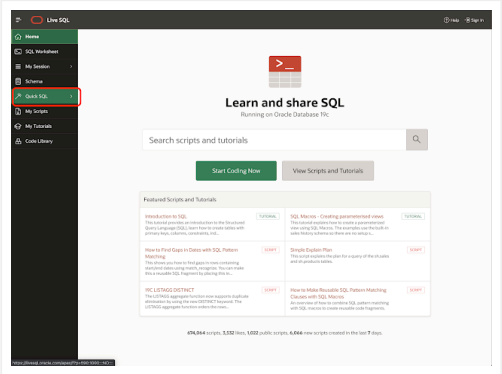
最近は新たに表を作る時は、もっぱら**クイックSQL**を使っています。こういう使い方をした方が便利だな、と感じたことをまとめてみます。

いきなり仕様を読むのもつらいので、その前段階の使い方の紹介になります。仕様そのものについては、オンライン・ヘルプを参照してください。

クイックSQLが使えるサイト

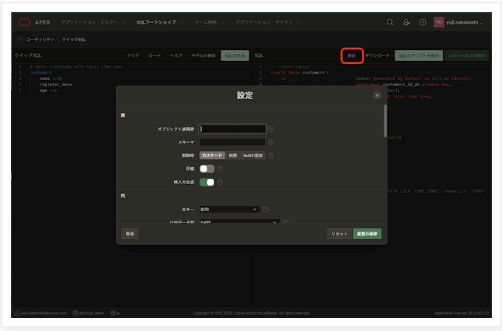
Oracle APEXがインストールされていれば、どこでも。**SQLワークショップ**に含まれる**ユーティリティ**を開いて、**クイックSQL**を実行します。

LiveSQL - https://livesql.oracle.com - でも、**クイックSQL**を実行できます。

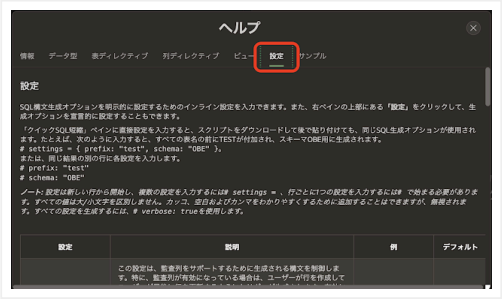


コメントによるインライン設定

クイックSQLによる記述を元に生成されるDDLは、**設定**によって変わります。



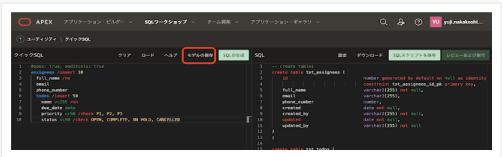
ただし、この画面上の設定は保存されず、新たにクイックSQLを開く度に、毎回設定し直す必要があります。その手間を省くために、コメントによって**インライン設定**を行うことができます。詳しい記述方法については、オンライン・ヘルプの**設定**を参照してください。



一度設定を決めると変更することはあまり無いため、インライン設定はとても便利です。

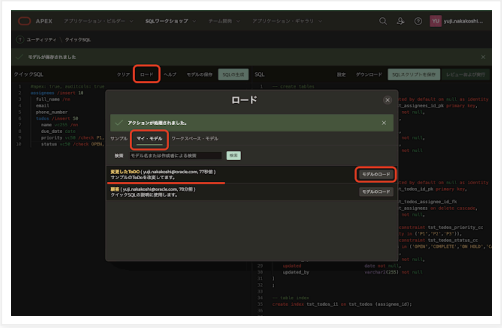
モデルの保存

クイックSQLで記載したモデルを保存することができます。**モデルの保存**をクリックします。



モデル名と**説明**を入力し、**モデルの保存**を実行します。

保存したモデルは**ロード**から、**マイ・モデル**を選んで、**モデルのロード**を実行することでリストアできます。



保存されたモデルを更新する方法は、見つけることができませんでした(LiveSQLのQuick SQLにはUpdate Modelの機能があります)。別の名前で保存するか、保存済みのモデルを事前に削除する必要があります。

コメントによるインライン設定もモデルとして保存されます。そのため、インライン設定をしていれば、モデルをロードする度に画面上で設定を繰り返す必要がありません。

表定義の基本

表の名前を記述し、次の行からインデントを付けて、その表の列を記述していきます。

```
# date: timestamp with local time zone
customers
   name vc80
   register_date
   age num
```

列名に続けて、列の型を指定します。

num (number)、**vcNN** (varchar2(NN))、**date**(型は設定による)が主に使われる型指定になるでしょう。列名の末尾にdateが付いていると型指定なしでも、date型と認識されます。

date型は、date設定によって生成されるDDLの型指定が変わります。以下のように、date設定をtimestamp with local time zoneとすると、date型として指定した列はすべてtimestamp with local time zone型としてDDLが生成されます。

```
# date: timestamp with local time zone
```

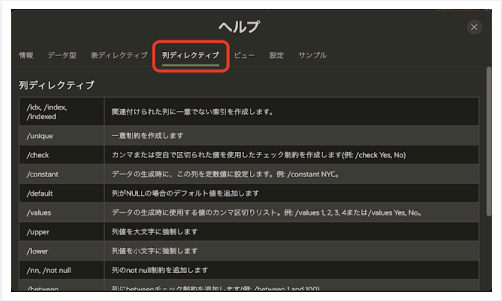
スキーマ定義でdate型とtimestamp with local time zone型を混在させることはあまりない、と思います。その場合はクイックSQLではdateとして記述し、設定で実際の型を決める方が良いでしょう。timestamp with time zone型とtimestamp with local time zone型が混在する場合など、型を特定する必要がある場合は、明示的に型を指定します。その場合でも、tstz, tsltzといった短縮した文字列で型を指定できます。

データ型の詳細は、**オンライン・ヘルプ**の**データ型**を参照してください。オンライン・ヘルプには、tstzでもtimestamp with local time zoneになると記載されていますが、実際はきちんとtimestamp with time zone型としてDDLが生成されます。

列の制約の指定

一番使うのはNOT NULL制約だと思います。列の型指定に続けて **/nn** (/not null)と記載します。そのほかでは外部キー制約を指定する **/fk** (/references, /reference)や、列のデフォルト値を指定する **/default** があります。

列の制約については、**オンライン・ヘルプ**の**列ディレクティブ**に説明があります。



クイックSQLを使うまではあまりチェック制約 **/check** を表に定義したことはなかったのですが、クイックSQLを使うようになってから、 以下のように列を定義することが増えました。

```
   is_enabled vc1 /check Y,N /default N /nn
```

以下のDDLが生成されます。

```
   is_enabled                varchar2(1) default 'N' constraint customers_is_enabled_cc
                          check (is_enabled in ('Y','N')) not null,
```

クイックSQLの癖のようなものかもしれませんが、/defaultは/checkの後に指定した方が良いです。順番によってはDDLが適切に生成されません。

```
   is_enabled vc1 /nn /default N /check Y,N
```

生成されたDDLです。**is_enabled_vc1__/default_n**がチェック制約を適用する列名になってしまっています。

```
   is_enabled                varchar2(1) default 'N' constraint customers_is_enabled_vc_cc
                          check (is_enabled_vc1__/default_n in ('Y','N')) not null,
```

マスター・ディテイル関係の表の定義

最初にマスター表を定義します。ディテイル表は**インデントを付けて定義**します。

```
# semantics: default
orders
   customer_name
   order_date
   order_items
      product_name
      price num
      quantity num
```

ordersがマスター表で、order_itemsがディテイル表です。order_itemsは続いて列が定義されているので、表と認識されます。以下のDDLが生成され、order_items表にはマスター表の列を保持するorder_id列が追加され、外部キー制約としてマスター表の主キーを参照しています。

```
-- create tables
create table orders (
   id                      number generated by default on null as identity
                          constraint orders_id_pk primary key,
   customer_name              varchar2(255),
   order_date                 date
)
;

create table order_items (
   id                      number generated by default on null as identity
                          constraint order_items_id_pk primary key,
   order_id                   number
                          constraint order_items_order_id_fk
                          references orders on delete cascade,
   product_name               varchar2(255),
   price                   number,
   quantity                   number
)
;

-- table index
create index order_items_i1 on order_items (order_id);
```

削除時の動作は、**ondelete**設定としてcascade、restrict、set nullの中から選ぶことが可能です。

主キー定義

自動主キーという設定があり、これをONにすると、主キーとなる列が自動的に追加されます。クイックSQLでは、自動主キーをONにすることが推奨されています。自動主キーがOFFのときは、主キー列は自動生成されません。そのため、主キーとなる列を定義に含め **/pk** を制約として付加します。

```
customers
   customer_id num /pk
```

```
  name vc80
  register_date
  age num
```

主キー制約 /pk が明示された列が存在する場合は、自動主キーがONでも主キー列は自動生成されません。

生成される主キーに関するDDLは、いくつかの設定から影響を受けます。 **prefixPKwithTname**、 **PK**、**DB**の３つの設定です。

```
# prefixPKwithTname: true
# PK: identity
# DB: 18c
customers
  name vc80
  register_date
  age num
```

prefixPKwithTnameがtrueの場合、主キー列は**表名_id**になります。falseの場合は単に**id**です。PKはguid, seq, identityの３種類のうちのどれかを設定します。

PKとしてguidを設定したときに生成される、主キーの定義は以下になります。

```
  customer_id               number default on null to_number(sys_guid(), 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX')
                        constraint customers_id_pk primary key,
```

seqの場合は以下です。これ以外に順序customer_seqを作成するDDLも生成されます。

```
  customer_id               number default on null customers_seq.NEXTVAL
                        constraint customers_id_pk primary key,
```

identityの場合は以下です。

```
  customer_id               number generated by default on null as identity
                        constraint customers_id_pk primary key,
```

identityはOracle Database 12c以降で利用可能な指定なので、DBが11gの場合は、guidを指定した場合と同じDDLが生成され、identityの設定は無視されます。

生成されるDDLは、Oracle APEXのバージョンによっても異なります。できるだけ新しいバージョンのOracle APEXに含まれるクイックSQLを使用することにより、より適切なDDLを生成することができます。

外部キー制約

マスター表に含める形でディテイル表を定義すると、自動的に外部キー制約がDDLに追加されます。そうではなく、外部キー制約を明示する場合は、**/fk** (または/references, /reference) を指定します。

先に利用した例ordersとorder_items表をインデントを使わずに定義すると、以下の記述になります。

```
 # semantics: default
orders
  customer_name
  order_date

order_items
  order_id /fk orders
  product_name
  price num
  quantity num
```

外部キーの宛先となる表は、すでに作成済みであるか、外部キー制約を持つ表より先に定義されている（つまり、先にDDLが実行され表として作成される）必要があります。

監査列について

設定の**追加列**に**監査列**という設定があります。インライン設定では、**auditCols**です。

この設定をONにすると、**作成された列名**、**作成者の列名**、**更新された列名**、**更新者の列名**の４つの列を含んだDDLが生成され、また、それらの列に値を設定するデータベース・トリガーも生成されます。

表については、以下のように列が追加されます。

```
   created                 date not null,
   created_by              varchar2(255) not null,
   updated                 date not null,
   updated_by              varchar2(255) not null
```

生成されるトリガーのDDLは、設定の**APEX有効**がONかOFFかで異なります。

APEX有効がONの場合は、次のDDLが生成されます。APEXが使用されていることを前提として、APEX$SESSIONよりユーザー名APEX_USERを取得し、列に保存します。作成時刻、更新時刻のデータ型はdate設定から決まります。

```
-- triggers
create or replace trigger orders_biu
   before insert or update
   on orders
   for each row
begin
   if inserting then
      :new.created := sysdate;
      :new.created_by := coalesce(sys_context('APEX$SESSION','APP_USER'),user);
   end if;
   :new.updated := sysdate;
   :new.updated_by := coalesce(sys_context('APEX$SESSION','APP_USER'),user);
end orders_biu;
/
```

OFFの場合はデータベースの接続ユーザーを取得し、列に保存します。

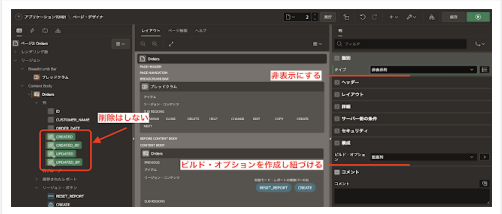
```
-- triggers
create or replace trigger orders_biu
   before insert or update
   on orders
   for each row
begin
   if inserting then
      :new.created := sysdate;
      :new.created_by := user;
   end if;
```

```
    :new.updated := sysdate;
    :new.updated_by := user;
end orders_biu;
/
```

監査列はあくまでシステム上の要件で使用し、これをビジネス・ロジックの中で使用することは推奨しません。例えば注文情報を登録した顧客を表ORDERSに保存する場合は、列CUSTOMERといった列を定義し、監査列のCREATED_BYを流用してはいけません。特にトリガーによって更新される場合、データのメンテナンスのために発行したSQLによっても監査列は更新されます。監査用途であれば、その変更が記録されるのは適切です。しかし、注文情報を登録したユーザー、つまりビジネス上の意味を持たせていると問題が発生します。

監査列をビジネス・ロジックの中で取り扱わないことにすると、Oracle APEXのアプリケーションのレポートやフォームでは、それらの列は無いものとして扱うことになります。

しかし実際に列から削除すると、表定義が変更されたときに**ページ・アイテムの同期化**を実行する度に、再度、監査列が追加されることになります。**タイプ**を**非表示**に変更する、もしくは、**ビルド・オプション**に紐づけて非表示にする、という対応を行う方が良いでしょう。



表と列の名前について

Oracle APEXでアプリケーションを作成する際に、ワークスペースに複数のアプリケーションが作られることが一般的です。ですので、単に表の名前をEMPLOYEESやPERSONSとしたような場合に、表の名前が競合する可能性があります。

設定の**prefix(オブジェクト接頭辞)**にて、名前の先頭に識別子を一律に追加することができます。以下では文字列TSTをオブジェクト接頭辞として付加しています。

```
# prefix: tst
# semantics: default
emploees
  name vc80
  sal num
  com num
```

生成されるDDLの表名はTST_EMPLOYEESとなります。

```
-- create tables
create table tst_emploees (
    id                      number generated by default on null as identity
                            constraint tst_emploees_id_pk primary key,
    name                        varchar2(80),
    sal                     number,
    com                         number
)
;
```

オブジェクトを含むスキーマで識別すべきで、表名などに接頭辞を含めなくてもよい、という考え方はあるかと思います。しかし、Oracle APEXでアプリケーションを作成する場合、解析対象スキーマの変更によるセキュティ面での影響が大きいことを考慮すると、解析対象スキーマの変更を前提とせず、ひとつのスキーマ内でもオブジェクトの名前が競合しないように、オブジェクトに接頭辞をつけるのは良い習慣です。

オブジェクト接頭辞を指定した状態で、プリファレンスの**主キーに表名を接頭辞として付けます**(prefixPKwithTname)をONにすると、オブジェクト接頭辞も含んだ主キー列名になります。

```
# prefixPKwithTname: true
# prefix: tst
# semantics: default
employees
  employee_name vc80
  sal num
```

列名はより説明的な方がSQLの可読性が高くなるという理由で、以下の表TST_EMPLOYEESの列NAMEをEMPLOYEE_NAMEにすることはあっても、TST_EMPLOYEE_NAMEとはしないでしょう。しかし、主キー列はTST_EMPLOYEE_IDになってしまいます。

```
-- create tables
create table tst_employees (
    tst_employee_id             number generated by default on null as identity
                            constraint tst_employees_id_pk primary key,
    employee_name               varchar2(80),
    sal                         number
)
;
```

次に説明する、行キー、行バージョン番号のことも考えると、プリファレンスの**主キーに表名を接頭辞として付けます**は**OFF**が良いと思います。主キー列の名前は常に**ID**になり、行キーを別に作る理由が無くなります。

行キーと行バージョン番号

追加列の指定に、**行キー**と**行バージョン番号**というものがあります。

行バージョン番号だけを設定し、生成されるDDLを確認します。

```
# prefix: tst
# semantics: default
# rowVersion: true
employees
  employee_name vc80
  sal num
```

列ROW_VERSIONが追加され、その列に値を設定するトリガーが生成されます。

```
-- create tables
create table tst_employees (
    id                          number generated by default on null as identity
                            constraint tst_employees_id_pk primary key,
    row_version                 integer not null,
    employee_name               varchar2(80),
    sal                         number
)
;


-- triggers
create or replace trigger tst_employees_biu
    before insert or update
    on tst_employees
    for each row
begin
    if inserting then
```

```
      :new.row_version := 1;
   elsif updating then
      :new.row_version := nvl(:old.row_version,0) + 1;
   end if;
end tst_emploees_biu;
/
```

行バージョン番号は、その行の更新が発生する度に 1 づつインクリメントされます。この列の使いどころの説明として、REST APIでの利用を考えてみます。

最初に、表TST_EMPLOYEESよりIDを渡して、EMPLOYEE_NAME, SALを取得します。このときROW_VERSIONも同時に取得します。

1. クライアントAがID 300のデータを取得しました。

Client A: https://xxxxxx/xxx/employee/300 (GET)

取得した値は以下です。

{ "id": 300, "employee_name": "山田太郎", "sal": 300, "row_version": 4 }

2. クライアントBも同じデータを取得しました。

Client B: https://xxxxxx/xxx/employee/300 (GET)

{ "id": 300, "employee_name": "山田太郎", "sal": 300, "row_version": 4 }

3. クライアントBがSALを400へ更新しました。

Client B: https://xxxxxx/xxx/employee/300 (PUT)

{ "id": 300, "sal": 400, "row_version": 4 }

送信されたデータに含まれるrow_version(4)と、表に保存されている行のROW_VERSIONが４で一致するので、SALは更新されます。更新された行のROW_VERSIONは+1されて５になります。

4. クライアントAがSALを200へ変更しようとしました。

Client A: https://xxxxxx/xxx/employee/300 (PUT)

{ "id": 300, "sal": 200, "row_version": 4 }

表に保存されている行のROW_VERSIONはすでに５になっており、送信されたデータとは異なるため更新は失敗します。

以上のように行バージョン番号は、異なるクライアントによって、アップデートが不用意に上書きされないことを保証するために使用されます。

HTTPのPUTを受け付けて列SALを更新する前に、ROW_VERSIONの値を参照する必要があります。そのため、以下のような問い合わせを発行します。

```
select row_version from tst_employees where id = 300 for update;
```

受信したrow_versionと比較して、一致していたら、

```
update tst_employees set sal = 300 where id = 300;
```

といった手順を踏むことになります。ここで主キー列の名称が表毎に異なっていると、row_versionを取り出す手順が煩雑になります。一意列が同じ列名ROW_KEYとして全ての表に定義されていることにより、少々扱いが容易になります。これが行キーの使い道です。

私の嗜好としては、**主キーに表名を接頭辞として付けます**が**OFF**で、主キー列がつねに**ID**という名前になっていれば、列ROW_KEYを定義する理由はないので、定義しないです。

Oracle APEXでは行バージョン番号の代わりにチェックサムを使うことで、上記の制御を実装しています。以下のような形式でチェックサムを計算し、保存されている行のチェックサムと受信したチェックサムに違いが無ければ更新処理を行います。

```
SELECT APEX_ITEM.MD5_CHECKSUM(ID, EMPLOYEE_NAME, SAL) FROM TST_EMPLOYEES WHERE ID = 300 FOR UPDATE NOWAIT;
```

行バージョン番号も使用できますが、デフォルトではチェックサムを使用します。そのため、Oracle APEXで表を扱う範囲では、列キーと行バージョン列の双方とも必要としていません。

チェックサムでは扱いが難しい、例えば表に列が大量に定義されている、Oracle APEXの外でデータが変更されることがある、といった場合に利用を検討します。

表と列のコメント

--または **[]** で表や列にコメントを付けることができます。

```
employees -- 従業員の情報
  employee_name vc80  -- 従業員名
  sal num  -- 月給
```

または

```
employees [従業員の情報]
  employee_name vc80 [従業員名]
  sal num [月給]
```

これは単にクイックSQLへの記述ではなく、COMMENT文になります。

```
-- comments
comment on table emploees is '従業員の情報';
comment on column emploees.employee_name is '従業員名';
comment on column emploees.sal is '月給';
```

結果として、DESCコマンドや、USER_TAB_COMMENTS、USER_COL_COMMENTSなどを検索することでコメントを参照することができます。オンライン・ヘルプには列ディレクティブにのみコメントの説明がありますが、表にたいしてもコメントは有効です。

テスト・データの生成

表ディレクティブとして **/insert 行数** を指定することで、テスト・データを挿入するINSERT文が自動生成されます。**設定**の**挿入の生成**が**ON**である必要があります。これに対応するインライン設定はありません。

データ言語として、英語、日本語、韓国語、ドイツ語、スペイン語のどれかを選ぶことができます。こちらはインライン設定があります。

```
# language: "JA"
```

場合によっては、それらしいデータも挿入されますが（表EMPLOYEESとして定義した表にある列NAMEや、表TASKSとして定義した表にある列NAMEなど）、基本はデータ型を見て、その型にあったデータを適当に生成します。

履歴表の作成

表ディレクティブ **/history** を指定することで、変更履歴を保持する表、および変更されたデータを書き込むトリガーを生成することができます。

以下のように /history を付けるだけで、履歴保持を行うDDLが生成されます。

```
# prefix: tst
employees /history
  name vc80
  sal num
```

DcDLは非常に長いので掲載は割愛します。生成させる指定は簡単ですので、クイックSQLで確認してみてください。

デフォルトのセマンティクス

セマンティクスのデフォルトがCHARになっています。ですので、明示的に指定しないとVARCHAR2の長さなどの単位が、文字になってしまいます。データベースの設定に従う方が望ましいため、セマンティクスの設定はインライン設定に必ず含めることをお勧めします。

```
# semantics: default
```

無指定の場合は列NAMEの型は varchar2(80 char)として定義されますが、上記のセマンティックスのインライン設定を含めると varchar2(80) となります。

以上でクイックSQLの簡単な紹介は終了です。Oracle APEXでのアプリケーション開発の一助になれば幸いです。

完

Yuji N. 時刻: 23:16

共有

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示