PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

INTRODUCTION

This document is geared toward a one-way movement of SQL from Oracle's PL/SQL syntax to Microsoft SQL Server's Transact-SQL (T-SQL). Therefore, focus is placed on converting certain PL/SQL constructs to their T-SQL equivalent.

(**Note**: Each of the following sections have PL/SQL and T-SQL source code files that accompany them. The names of the files are "<SectionTitle>.sql". For example, a section entitled SYNTAX would have two files accompanying it. Syntax.sql would appear in both a PL-SQL directory and a T-SQL directory. Also, if text is divided into two columns, the left column represents Oracle syntax and the right column represents SQL Server syntax.)

DELIMITERS

PL/SQL likes the use of delimiters. In fact, it can't live well without a lot of them. Examples are the IF…END IF construct and the ubiquitous use of the semi-colon delimiter. T-SQL is not so dependent upon delimiters. In T-SQL, prior statements are ended by the existence of a succeeding statement. In Oracle, the entire scope of a construct is self-delimiting. Therefore, T-SQL appears to be less verbose.

```
begin
        IF      0 > 1 THEN
                dbms_output.put_line ('0 > 1');
        ELSIF   0 > 2 THEN
                dbms_output.put_line ('0 > 2');
        ELSE
                dbms_output.put_line ('None');
        END IF;
end;
/
```

```
begin
        IF      0 > 1
                print '0 > 1'
        ELSE
                IF      0 > 2
                        print '0 > 2'
                ELSE
                        print 'None'
end
GO
```

The "THEN" and "END IF" portions of the "IF" statement have no SQL Server counterpart. The semi-colon delimiter is not used in SQL Server, but may be allowed. Oracle requires a forward slash to execute the anonymous PL/SQL block. SQL server allows an optional "GO" statement.

Special Characters

In Oracle, object attributes are assigned based on datatype or object definition. SQL Server also uses a variety of special characters to more explicitly denote object attributes. These special characters are prepended to the object name.

Ch	Description	Usage	Example
@	single at sign	local variable	`@Local_Variable1`
@@	double at sign	global variables	`@@Global_Variable2`
#	single pound sign	local temporary object	`#My_Local_Temporary_Table`
##	double pound sign	global temporary object	`##email` (for sending e-mail)

This nomenclature is mandatory and signals SQL Server to treat these objects in certain ways implicitly. Therefore, object maintenance is simplified. Also, keep in mind that the '@' symbol in Oracle is prepended to a .sql file to execute the contents of the file in SQL*Plus.

ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

ROWNUM

The Oracle pseudo-column, Rownum, has several uses. One of which is to ensure that a SELECT statement only returns one row for the purpose of using INTO implicit cursor processing, thereby simplifying cursor control. SQL Server has no direct equivalent to Rownum, but a work-around can be devised.

```
select distinct(job)
  from emp
 order by 1
;

declare
       v_job varchar2(10);
begin
       select distinct(job)
         into v_job
         from emp
        where rownum < 2
        order by 1
       ;
       dbms_output.put_line('Job = ' || v_job);

       select distinct(job)
         into v_job
         from emp
        --where rownum < 2
        order by 1
       ;
       dbms_output.put_line('Job = ' || v_job);

end;
/

JOB
---------
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN

Input truncated to 1 characters
Job = CLERK

declare
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested
number of rows
ORA-06512: at line 12
```

```
select job_desc
  from jobs
 where job_desc like 'P%'
 order by 1

declare
       @job_desc varchar(50)


select @job_desc = job_desc
  from jobs
 where job_desc like 'P%'
 order by 1

select 'Last entry = ' + @job_desc

select @job_desc = job_desc
  from jobs
 where job_desc like 'P%'
 order by 1 DESC


select 'First entry = ' + @job_desc




Productions Manager
Public Relations Manager
Publisher




Last entry = Publisher


First entry = Productions Manager
```

The work-around is to "`Order By`" the entire expected output set in "DESCending" order. Therefore, the variable is populated with the last value in the answer set, which is the first one that is desired.

Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

HINTS

Oracle uses hints to assist with the control of join table sequencing and index selection. SQL Server hints perform these functions with the added availability of locking hints. Prevalent Oracle hints are listed with their SQL Server counterpart. Although the counterpart is listed, testing should be performed to verify the hint performs correctly. (**Note**: Due to the /*…*/ syntax of Oracle hints, they are regarded by SQL Server as a comment.)

```
select /*+ FIRST_ROWS */ ename "First Rows"
  from emp
 where ename < 'B';

select /*+ INDEX (emp PK_EMP) */ ename "Index"
  from emp
 where ename < 'B';

select /*+ RULE */ ename "Rule"
  from emp
 where ename < 'B';
```

```
select fname "First Rows"
  from employee (FastFirstRow)
 where fname < 'B'

select fname "Index" -- SQL Server 7.0
  from employee (Index = employee_ind)
 where fname < 'B'

select /*+ RULE */ fname "Rule"
  from employee
 where fname < 'B'
```

There is no SQL Server equivalent to the Oracle RULE hint. However, if not removed, the code will still parse without error.

Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

EXCEPTIONS

Oracle has a "DECLARE BEGIN EXCEPTION END" syntax for procedural execution control. If certain exceptions occur during processing, then control of logic is transferred directly to the Exception block with subsequent exit from the module. Also, there are several predefined exceptions in Oracle whose keywords are used to navigate exception block logic.

```
declare
  v_char varchar2(1);
begin
  select dummy
    into v_char
    from dual
 where dummy = 'Y'
;
dbms_output.put_line('OK');

EXCEPTION
  WHEN no_data_found THEN
       dbms_output.put_line('No data found');
end;
/

declare
  v_char varchar2(20);

begin
  select ename
    into v_char
    from emp
;

dbms_output.put_line('OK');

EXCEPTION
  WHEN too_many_rows THEN
       dbms_output.put_line('Too many rows');
end;
/
No data found
Too many rows
```

```
declare
  @v_char varchar(1)
begin
  select @v_char = fname
    from employee
 where fname = 'No Body'
if @@rowcount = 0 goto EXCEPTION

select 'OK'
return
EXCEPTION:
  IF @@rowcount = 0 -- reset by goto
       select 'No data found'
end


declare
  @v_char varchar(20),
  @rowcount integer
begin
  select @v_char = fname
    from employee
set @rowcount = @@rowcount
if @rowcount > 0 goto EXCEPTION

select 'OK'
return
EXCEPTION:
  IF @rowcount > 0
       select 'Too many rows'
end

No data found
Too many rows
```

In the '@@rowcount = 0' conditional test, the GOTO function causes @@rowcount to be reset. It is then tested again after the EXCEPTION tag, allowing for error.
The second example shows a better way. The value of @@rowcount is captured by a local variable @rowcount.
Some Oracle predefined exceptions are: dup_val_on_index, no_data_found, too_many_rows, zero_divide, rowtype_mismatch, invalid_number, etc.

ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

Implicit Cursor Attributes (Percent.sql)

Each cursor created in Oracle, either implicitly or explicitly, carries with it cursor attributes, `%notfound, %found, %rowcount, %isopen`. If the cursor is declared explicitly as `Cursor1`, for example, the attribute is available via "`Cursor1%found`" syntax. "`SQL%found`" syntax is also available for the most recently processed implicit or explicit cursor.

```
declare v_name varchar2(20);
begin
  select ename
    into v_name
    from emp
   where ename = 'ADAMS'
  ;
  if SQL%found then
    dbms_output.put_line('It''s a hit');
  end if;
end;
/

It's a hit
```

```
declare @v_name varchar(20)
begin
  select @v_name = fname

    from employee
   where fname = 'Diego'
  ;
  if @@rowcount > 0
    select 'It''s a hit'

end


It's a hit
```

END Tags

When defining Packages, Procedures, Functions, or named blocks of name "`OracleObject`" in Oracle, the last line of the definition is "`END OracleObject;`". This construct has no equivalent in SQL Server. But, the existence of the tag helps with documentation. So, the tag should be commented out by a double dash.

```
create or replace
procedure OracleObject IS
begin
  dbms_output.put_line('Hello, World!');
end OracleObject;
/

exec OracleObject;

Hello, World!
```

```
if exists (
      select name
        from sysobjects
       where name = 'SQLObject'
         and type = 'P')
    drop procedure SQLObject
GO

create
procedure SQLObject AS
begin
  Select 'Hello, World!'
end --SQLObject
GO

SQLObject

Hello, World!
```

Also, notice the work-around for the "`or replace`" portion of the "`create`" statement. In addition, notice that SQL Server only supports "`AS`", not "`IS`".

ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

PARAMETERS

Oracle named PL/SQL blocks can have parameters. The datatype for these parameters is referenced without length, precision and/or scale. These must be added to SQL Server parameters.

```
create or replace
function First5 (
   p_Char varchar2)
return varchar2
is
begin
  return substr(p_Char,1,5);
end;
/
show errors;

Select First5('Ticonderoga') from dual;

Function created.

No errors.

FIRST5('TICONDEROGA')
---------------------
Ticon
```

```
create
function First5 (
   @p_Char varchar(20))
returnS varchar(5)
AS
begin
  return substring(@p_Char,1,5);
end
GO


Select pubs.dbo.First5('Ticonderoga')


Ticon
```

In the function header for SQL Server, the keyword "`return`**`s`**" is used. Elsewhere, "`return`" is used as in PL/SQL.
Also notice here that the function call has a fully qualified function name.

First5 above is a deterministic function because for any set of input, it will always return the same output. Some system functions, like `GETDATE()`, are non-deterministic, in that they do not return the same value with the same input. Built-in non-deterministic functions are not allowed in the body of user-defined functions.

Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

CONNECT

Tree traversal in Oracle is facilitated by "CONNECT BY...PRIOR" syntax. This type of syntax is not available in SQL Server. SQL Server Books Online recommends the following solution.

```
create table Region (
       ID      number not null,
       Parent  varchar2(20),
       Child   varchar2(20),
  primary key (ID)
);

insert into Region values (0,'World',            'Europe');
insert into Region values (1,'World',            'North America');
insert into Region values (2,'Europe',           'France');
insert into Region values (3,'France',           'Paris');
insert into Region values (4,'North America',    'United States');
insert into Region values (5,'North America',    'Canada');
insert into Region values (6,'United States',    'New York');
insert into Region values (7,'United States',    'Washington');
insert into Region values (8,'New York',         'New York City');
insert into Region values (9,'Washington',       'Redmond');

insert into Region values (10,NULL,              'World');

column Hierarchy format a50;

select lpad(' ',3*(level-1)) || child "Hierarchy"
  from region
 start with child = 'World'
connect by Parent = PRIOR child;
```

```
        ID PARENT               CHILD
---------- -------------------- -------------
        10                      World
         0 World                Europe
         1 World                North America
         2 Europe               France
         3 France               Paris
         4 North America        United States
         5 North America        Canada
         6 United States        New York
         7 United States        Washington
         8 New York             New York City
         9 Washington           Redmond


Hierarchy
-------------------------
World
   Europe
      France
         Paris
   North America
      United States
         New York
            New York City
         Washington
            Redmond
      Canada
```

```
create table Region (
       ID      numeric not null,
       Parent  varchar(20),
       Child   varchar(20),
  primary key (ID)
);

insert into Region values (0,'World',            'Europe');
insert into Region values (1,'World',            'North America');
insert into Region values (2,'Europe',           'France');
insert into Region values (3,'France',           'Paris');
insert into Region values (4,'North America',    'United States');
insert into Region values (5,'North America',    'Canada');
insert into Region values (6,'United States',    'New York');
insert into Region values (7,'United States',    'Washington');
insert into Region values (8,'New York',         'New York City');
insert into Region values (9,'Washington',       'Redmond');

CREATE PROCEDURE expand (@current char(20)) as
  SET NOCOUNT ON
DECLARE
  @level int,
  @line char(20)
CREATE TABLE #stack (
  item char(20),
  level int)
INSERT INTO #stack
  VALUES (@current, 1)
SELECT @level = 1
WHILE @level > 0
BEGIN
  IF EXISTS (SELECT * FROM #stack WHERE level=@level)
    BEGIN
       SELECT @current = item
         FROM #stack
        WHERE level = @level
       SELECT @line = space(3*(@level - 1))+@current
       PRINT @line
       DELETE FROM #stack
        WHERE level = @level
          AND item = @current
       INSERT #stack
          SELECT child, @level + 1
            FROM region
           WHERE parent = @current
       IF @@ROWCOUNT > 0
          SELECT @level = @level + 1
    END
 ELSE
    SELECT @level = @level - 1
END -- WHILE
GO

expand @current = 'World'
```

```
World
   North America
      Canada
      United States
         Washington
            Redmond
         New York
            New York
   Europe
      France
         Paris
```

The above solution uses a WHILE loop. Although SQL Server will support recursive functions, recursion is not ideal here for two reasons:

- Recurson is very resource-intensive (due to multiple function instantiation)
- SQL Server "only" supports nesting to 32 levels (4,294,967,296 binary entries)

ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

SQLERRM

The Oracle Communications Area populates several global variables upon execution of each SQL statement. SQLCODE contains an error number and SQLERRM a text error message. SQLCODE's equivalent is @@ERROR. The equivalent to SQLERRM is a lookup to the master.dbo.sysmessages table.

```
begin
        insert
          into emp (empno)
        values (7369); -- already there
exception
    when others then
        dbms_output.put_line('SQLCODE = '||SQLCODE);
        dbms_output.put_line('SQLERRM = '||SQLERRM);
end;
/
```

```
SQLCODE = -1
SQLERRM = ORA-00001: unique constraint(SCOTT.PK_EMP)
violated
```

```
create procedure SQLERRM
as
declare
        @description varchar(200),
        @ERROR int
insert
  into employee
select *
  from employee
 where emp_id = 'PMA42628M'

set @ERROR = @@ERROR
print '@ERROR = ' + convert(varchar,@ERROR)

select @description = description
  from master.dbo.sysmessages
 where error = @ERROR

print 'DESC = ' + @description
GO

SQLERRM
```

```
Server: Msg 2627, Level 14, State 1, Procedure
SQLERRM, Line 6
Violation of PRIMARY KEY constraint 'PK_emp_id'.
Cannot insert duplicate key in object 'employee'.
The statement has been terminated.
@@ERROR = 2627
DESC = Violation of %ls constraint '%.*ls'. Cannot
insert duplicate key in object '%.*ls'.
```

Even though there is error processing in SQL Server, the system generates an error message, then continues processing. Oracle exception handling transfers control to the exception block without generating system messages.

ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

Constants

The value of a declared Oracle constant cannot be subsequently modified.

```
declare
       pi constant number := 3.14159;
begin
       pi := 1.4142135;
end;
/
```

```
declare
       pi          number := 3.14159;
begin
       pi := 1.4142135;
       dbms_output.put_line('pi = ' || pi);
end;
/
```

```
        pi := 1.4142135;
        *
ERROR at line 4:
ORA-06550: line 4, column 2:
PLS-00363: expression 'PI' cannot be used as an
assignment target
ORA-06550: line 4, column 2:
PL/SQL: Statement ignored

pi = 1.4142135
```

```
declare
       @pi float
set    @pi = 3.14159
begin
       set    @pi = 1.4142135
       select  'Local pi ', @pi
end

drop function pi
GO

create function pi()
returns float
AS
begin
       declare @pi float
       set @pi = 3.14159
       return(@pi)
end
GO

select 'System pi', pi()
select 'My pi', pubs.dbo.pi()

Local pi       1.4142135
System pi      3.1415926535897931
My pi          3.1415899999999999
```

There is no equivalent constant function in SQL Server. However, a user-defined function can be created to return a constant value. In the case above, a local user-defined function “pi” was created, despite the existence of a system-provided function of the same name. Please note that the user-defined “pi” must be fully qualified.

Chr (tab and newline)

When concatenating strings, it is sometimes convenient to embed display control characters for the purpose of visual presentation.

```
select
'A tab'               || chr(9)  ||
'separates this'      || chr(10) ||
'line.' "Tab newline"
from dual
;



Tab newline
-------------------------
A tab   separates this
line.
```

```
 select
'A tab'               + char(9)  +
'separates this'      + char(10) +
'line.' "Tab newline"

declare
       @line varchar(200)
select @line =
'A tab'               + char(9)  +
'separates this'      + char(10) +
'line.'
print @line

Tab Newline
-------------------------
A tab separates this line.

A tab   separates this
line.
```

Notice that the Select has no visual effect until the print statement is executed.

ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

Cursor FOR LOOP with inline Select (CursorFor.sql)

Oracle allows implicit cursor management with the Cursor FOR LOOP. However, cursor management in SQL Server is explicit. The following is an example of a Cursor FOR LOOP with an in-line Select.

```
begin
FOR r IN (
  SELECT ename
    from emp
   where ename < 'C'
   order by 1) LOOP
       dbms_output.put_line(r.ename);
END LOOP;
end;
/

ADAMS
ALLEN
BLAKE
```

```
declare @lname varchar(30)

DECLARE c1 cursor FOR
  select lname
    from employee
   where lname < 'C'
   order by 1
OPEN c1
FETCH c1 into @lname
WHILE @@FETCH_STATUS = 0
begin
       print @lname
       FETCH c1 into @lname
end
CLOSE c1
DEALLOCATE c1

Accorti
Afonso
Ashworth
Bennett
Brown
```

Initializing

In Oracle, variables can be initialized in the Declare statement. However, SQL Server requires separate Declare and Set statements.

```
declare
       pie float := 3.14159;
begin
       dbms_output.put_line('pie = ' || pie);
end;
/

pie = 3.14159
```

```
declare
       @pie float
    set @pie = 3.14159
begin
       print 'pie = ' + convert(varchar,@pie)
end

pie = 3.14159
```


Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

Sequences

Oracle has a separate object used for incrementing unique values. Uniqueness within the sequence is guaranteed. However, the sequence is not associated with and is independent of any other object. Therefore, its use is at the discretion of the user. The SQL Server equivalent is more directly associated with and dependent upon other objects. The SQL Server `IDENTITY` property creates an identity column in a table. The `IDENTITY` function is used for `SELECT INTO` operations. `IDENT_SEED` and `IDENT_INCR` functions will return the initial value and the increment for each table identity. `IDENT_CURRENT` is the equivalent of `Currval`. `@@IDENTITY` global variable and `SCOPE_IDENTITY` also return values with scope functionality more robust than Oracle.

```
create sequence NewID increment by 1 start with 1776;

Prompt Nextval not set, so Currval is undefined
select NewID.currval from dual;

Prompt Nextval is incremented
select NewID.nextval from dual;

Prompt Currval is now defined
select NewID.currval from dual;

Prompt Nextval is used for Insert
insert into emp (empno) values (NewID.nextval);

select NewID.currval from dual;

declare
          v_curr integer := 0;
          v_emp  integer := 0;
begin
          select NewID.currval
            into v_curr
            from dual
          ;
          select empno
            into v_emp
            from emp
           where empno = v_curr
          ;
          rollback;
dbms_output.put_line('v_curr = ' || v_curr);
dbms_output.put_line('v_emp  = ' || v_emp);
end;
/

Sequence created.

Nextval not set, so Currval is undefined
select NewID.currval from dual
       *
ERROR at line 1:
ORA-08002: sequence NEWID.CURRVAL is not yet defined in this session

Nextval is incremented
   NEXTVAL
----------
      1776

Currval is now defined
   CURRVAL
----------
      1776

Nextval is used for Insert
1 row created.


   CURRVAL
----------
      1777

v_curr = 1777
v_emp  = 1777
```

```
Drop table new_employees

Create table new_employees (
    id int IDENTITY(1776,1),
    fname varchar(20),
    minit char(1),
    lname varchar(30)
)

insert new_employees
  (fname, minit, lname) --identity not mentioned
values
  ('George','W','Bush')

insert new_employees
  (fname, minit, lname) --identity not mentioned
values
  ('Al','','Gore')

select ident_seed('new_employees'),
       ident_incr('new_employees'),
       ident_current('new_employees'),
       scope_identity(),
       @@IDENTITY

select *
  from new_employees

delete from new_employees


Seed   Incr      Current     Scope  @@Identity
----   ----      -------     -----  ----------
1776      1         1777      1777        1777


ID        fname     minit     lname
----      -----     -----     -----
1776      George    W         Bush
1777      Al                  Gore
```

If a globally unique value is required, then the `uniqueidentifier` data type used with the `ROWGUIDCOL` property is available.

ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

USERENV

The built-in Oracle SQL function, USERENV, is used to access system-specific data for a session.

```
select userenv('LANGUAGE') "Language" from dual;

select userenv('LANG') "Lang" from dual;

select userenv('TERMINAL') "Terminal" from dual;



select userenv('SESSIONID') "SessionID" from dual;

Language
------------------------------
AMERICAN_AMERICA.WE8ISO8859P1

Lang
------------------------------
US

Terminal
----------------
ONE

 SessionID
----------
       319
```

```
select @@LANGUAGE "Language",

       @@LANGID "LangID",

       @@REMSERVER "Remote",
       @@SERVERNAME "Local",
       @@SERVICENAME "Service",

       @@SPID "SessionID"

Language
--------
us_English

LangID Remote Local    Service SessionID
------ ------ -------- ------- ---------
     0 NULL   ONE\GBDB GBDB           51
```


Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

ROWID

RowID is an Oracle-provided pseudo-column (like RowNum). It is implicit with each row in a table and is a representation of that row's physical relative byte address. As such, in single server systems, ROWID could be unique. It has no implicit SQL Server equivalent. However, if RowID-like functionality is required, then a work-around is available at table definition time using the `uniqueidentifier` data type with the `ROWGUIDCOL` property and the `NEWID` function as the default.

```
select RowID from dual;
```

```
ROWID
------------------
AAAADDAABAAAAHSAAA
```

```
create table #RowGuid (
  RowID
    uniqueidentifier
    Primary Key
    ROWGUIDCOL
    default NEWID() ,
  RowName varchar(30)
)

alter table #RowGuid
  ADD RowID2
    uniqueidentifier
    default NEWID()


insert into #RowGuid
      (RowName)
    values
      ('Test Row')

select *
  from #RowGuid
```

```
RowID
------------------------------------
904CD193-0203-4274-BB42-AB1B807E5D46

RowName
--------
Test Row

RowID2
------------------------------------
8CC7448E-2B49-4C6F-A638-2C9B925B40CD
```

The combination of `uniqueidentifier` data type with the `ROWGUIDCOL` property and the `NEWID` function as the default causes each row to behave very similarly to the Oracle RowID pseudocolumn. However, the SQL Server value derived will have no relationship to physical layout of the data. The main drawback of this work-around is that it should be done with `CREATE TABLE`. However, `ALTER TABLE` syntax also allows a new column to be added with this functionality. Only one `ROWGUIDCOL` column is allowed per table.

Also, you may wish to contrast this with `IDENTITY` columns which are sequential integers unique only to the table itself used to mimic Oracle sequences. Wherein `uniqueidentifier` columns are globally unique (also non-sequential and not necessarily increasing in value).

Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

Boolean

Oracle has a Boolean data type for handling two-state data. However, NULLs are allowed with Oracle Boolean data types, which allows for three-state behavior. A near equivalent in SQL Server is the BIT datatype.

```
declare
       Choice BOOLEAN := null;

begin
  IF Choice THEN
     dbms_output.put_line('Yes');
  ELSIF not Choice THEN
     dbms_output.put_line('No');
  ELSE
     dbms_output.put_line('NULL');
  END IF;
end;
/

NULL
```

```
declare
  @Choice BIT
  set @Choice = NULL
begin
  IF @Choice = 1
     select 'Yes' "Choice"
  ELSE
     IF @Choice <> 1
        select 'No' "Choice"
     ELSE
        select 'Null' "Choice"
end

declare
  @Choice2 BIT
  set @Choice2 = 3

select @Choice2 "Choice2"

Choice
------
Null

Choice2
-------
1
```

Since `@Choice` is NULL, then "`@Choice <> 1`" evaluates to false.
Even though `@Choice2` is set to a value of 3, its data type accepts the assignment without error but retains the value as `1`.

Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

%ROWTYPE

SQL Server does not have equivalents to %ROWTYPE and %TYPE. The following procedure will produce text that can be pasted into the middle of a source code file “Declare” statement to minimize typing. For %TYPE, only the column name of interest should be pasted.

```
declare
          r_emp  emp%ROWTYPE;
          r_hold emp%ROWTYPE;
begin
  r_emp.empno       := 9999;
  r_emp.ename       := 'WILLIAMS';
  r_emp.job         := 'COMEDIAN';
  r_emp.mgr         := 7902;
  r_emp.hiredate := trunc(sysdate) + 1/3;
  r_emp.sal         := 800;
  r_emp.comm        := 0;
  r_emp.deptno      := 20;

  r_hold := r_emp;

  dbms_output.put_line ('Hold: Name = ' || r_hold.ename);
end;
/

Hold: Name = WILLIAMS
```

```
create procedure Rowtype
    @TableName varchar(20)
AS
select
  --'Declare @' +    -- for individual "Declare" statements
  c.name +
  space(15-len(c.name)) +
  t.name +
    case t.name
      when 'binary'  then '(' + convert(varchar,c.length) + ')'
      when 'varbinary' then '(' + convert(varchar,c.length) + ')'
      when 'char'    then '(' + convert(varchar,c.length) + ')'
      when 'varchar' then '(' + convert(varchar,c.length) + ')'
      when 'float'   then '(' + convert(varchar,c.length) + ')'
      when 'decimal' then '(' + convert(varchar,c.prec) + ',' +
                                convert(varchar,c.scale) + ')'
      when 'numeric' then '(' + convert(varchar,c.prec) + ',' +
                                convert(varchar,c.scale) + ')'
      else                         ''
    end
    + ','    -- for one "Declare" statement
                                "Declare"
          --c.length,
          --c.prec,
          --c.scale,
          --allownulls
  from syscolumns c
    join systypes t on c.xusertype=t.xusertype
 where id=object_id(@tableName)
 order by colorder
GO

Rowtype @TableName = 'employee'

Declare
--------------------
emp_id          empid,
fname           varchar(20),
minit           char(1),
lname           varchar(30),
job_id          smallint,
job_lvl         tinyint,
pub_id          char(4),
hire_date       datetime,


  (alternative syntax below)
Declare @emp_id         empid
Declare @fname          varchar(20)
Declare @minit          char(1)
Declare @lname          varchar(30)
Declare @job_id         smallint
Declare @job_lvl        tinyint
Declare @pub_id         char(4)
Declare @hire_date      datetime
```

“TYPE…is RECORD”

SQL Server does not support syntax similar to Oracle’s “`TYPE…is RECORD`” feature.
The above technique may be used as a work-around.

eNTeGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

"TYPE…is TABLE" (Table.sql)

Oracle's array processing feature is useful. A lot of code may exist utilizing this feature. While SQL Server may not have a direct counterpart, SCROLL cursor processing may be substituted in an effort to preserve the legacy logic. This work-around uses a SQL Server temporary table. Please notice below the difference in data sequence of the PL/SQL Table (array) and a SQL Server temporary table.

```
declare
  TYPE name_record IS RECORD (
    fname varchar2(20),
    lname varchar2(20)
  );

  TYPE name_table IS TABLE OF name_record
    index by binary_integer;

  name_array name_table;

  ix integer := 0;

begin
  name_array(13).fname := '13 Robin';
  name_array(13).lname := 'Williams';


  name_array(-7).fname := '-7 Jane';
  name_array(-7).lname := 'Austen';


  name_array(00).fname := '00 Grace';
  name_array(0).lname  := 'Hopper';




ix := name_array.FIRST;
dbms_output.put_line(name_array(ix).fname ||' '|| name_array(ix).lname);

ix := name_array.NEXT(ix);
dbms_output.put_line(name_array(ix).fname ||' '|| name_array(ix).lname);

ix := name_array.LAST;
dbms_output.put_line(name_array(ix).fname ||' '|| name_array(ix).lname);

end;
/

-7 Jane Austen
00 Grace Hopper
13 Robin Williams
```

```
create table #name_array (
  ix numeric,
  fname varchar(20),
  lname varchar(20)
)









insert into #name_array values (13,
  'Robin',
  'Williams')

insert into #name_array values (-7,
  'Jane',
  'Austen')

insert into #name_array values (00,
  'Grace',
  'Hopper')

declare name_cursor CURSOR LOCAL SCROLL for
  select *
    from #name_array

open name_cursor
fetch FIRST from name_cursor


fetch NEXT from name_cursor


fetch LAST from name_cursor

close name_cursor
deallocate name_cursor

13        Robin     Williams
-7        Jane      Austen
0         Grace     Hopper
```

Note that PL/SQL table keys are not necessarily sequential. However, FIRST/NEXT processing arranges them that way.
SQL Server SCROLL cursors cannot duplicate the array index. Therefore, any non-sequential value must be captured in the table.

(**Caution**: Ensure that the differences in sequential processing mentioned above are properly remediated in the legacy code.)

Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

Functions

Transact-SQL functions can only call other deterministic functions or extended stored procedures.

```
create or replace FUNCTION Now
return date
is
begin
         return sysdate;
end Now;
/

select now, sysdate from dual;

NOW                 SYSDATE
------------------- -------------------
2002-02-11 09:35:27 2002-02-11 09:35:27
```

```
create function Now ()
returns datetime
as
begin
  return getdate()
end
GO

Server: Msg 443, Level 16, State 1, Procedure Now, Line 6
Invalid use of 'getdate' within a function.

-- One may be tempted to call a procedure from a function:
create procedure Now2 @Date2 datetime OUTPUT
as
set @Date2 = getdate()
GO

create function Now ()
returns datetime
as
begin
  declare
    @Now2 datetime
  execute Now2 @Now2 OUTPUT
  return @Now2
end
GO

-- This will compile OK
The command(s) completed successfully.

-- However, upon execution:
select pubs.dbo.Now()

Server: Msg 557, Level 16, State 2, Procedure Now, Line 8
Only functions and extended stored procedures can be executed from
within a function.
```

Extended stored procedures are registered via `sp_addextendedproc` procedure.

Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

TopN

To give the Top 3 answers from a result set, include "TOP 3" immediately preceding the SELECT list columns. After all other processing is done (grouping, sorts, etc.), SQL Server only shows the first n rows of the result set. Notice that the data is in sequence by OrderID as dictated by the GROUP BY clause.

```
select  top 3  --PERCENT
     OrderID, sum(Quantity) "Pieces"
  from [Order Details]
 group by OrderID

OrderID     Pieces
----------- -----------
10248       27
10249       49
10250       60
```

Let's add an ORDER BY clause:

```
select  top 3
     OrderID, sum(Quantity) "Pieces"
  from [Order Details]
 group by OrderID
 order by Pieces

OrderID     Pieces
----------- -----------
10782       1
10807       1
10422       2
```

However, this output looks more like the "bottom 3" than the "top 3". This is because ORDER BY is ASCending by default. Let's make the sort order DESCending and see if we get "top 3".

```
select  top 3
     OrderID, sum(Quantity) "Pieces"
  from [Order Details]
 group by OrderID
 order by Pieces DESC

OrderID     Pieces
----------- -----------
10895       346
11030       330
10847       288
```


ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

Sometimes, there are many equal values such that “top 3” is too restrictive. In this case, add “with ties” to show values that tie with the third entry of the top 3.

```
select  top 3
     OrderID, sum(Quantity) "Pieces"
  from [Order Details]
 group by OrderID having sum(Quantity) = 20
 order by Pieces DESC

OrderID     Pieces
----------- -----------
10292       20
10317       20
10322       20


select  top 3 with ties
               OrderID, sum(Quantity) "Pieces"
  from [Order Details]
 group by OrderID having sum(Quantity) = 20
 order by Pieces DESC

OrderID     Pieces
----------- -----------
10292       20
10317       20
10322       20
. . . [some rows removed for brevity]
11040       20
11069       20

(27 row(s) affected)
```


ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

GroupByAll

Here is a list of supervisor IDs with a count of the number of people who report to each supervisor:

```
select  ReportsTo "Supervisor#", count(1) "Supervisee Count"
  from Employees
 group by ReportsTo

Supervisor# Supervisee Count
----------- ----------------
NULL        1
2           5
5           3
```

There are a total of nine employees. Eight have supervisors while one is his own boss and has no supervisor.
Let's just look at the Supervisee Count for Supervisor #2:

```
select  ReportsTo "Supervisor#", count(1) "Supervisee Count"
  from Employees
 where ReportsTo = 2
 group by ReportsTo

Supervisor# Supervisee Count
----------- ----------------
2           5
```

We are now interested in seeing the other supervisors without their respective counts (via ALL):

```
select  ReportsTo "Supervisor#", count(1) "Supervisee Count"
  from Employees
 where ReportsTo = 2
 group by ALL ReportsTo

Supervisor# Supervisee Count
----------- ----------------
NULL        0
2           5
5           0

(3 row(s) affected)

Warning: Null value is eliminated by an aggregate or other SET operation.
```


Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

CUBE

For analytical processing, ROLLUP and CUBE operators behave much the same as in Oracle. However, the syntax is slightly different.

```
select o.OrderID, o.ProductID, sum(o.Quantity) "Pieces"
  from [Order Details] o
 where o.OrderID in (10847,10895)
 group by o.OrderID, o.ProductID
 order by o.OrderID, o.ProductID

OrderID     ProductID   Pieces
----------- ----------- -----------
10847       1           80
10847       19          12
10847       37          60
10847       45          36
10847       60          45
10847       71          55
10895       24          110
10895       39          45
10895       40          91
10895       60          100

select o.OrderID, o.ProductID, sum(o.Quantity) "Pieces"
  from [Order Details] o
 where o.OrderID in (10847,10895)
 group by o.OrderID, o.ProductID with ROLLUP
 order by o.OrderID, o.ProductID
```

[Note: Oracle syntax is "`group by ROLLUP(o.OrderID, o.ProductID)`".]

```
OrderID     ProductID   Pieces
----------- ----------- -----------
NULL        NULL        634
10847       NULL        288
10847       1           80
10847       19          12
10847       37          60
10847       45          36
10847       60          45
10847       71          55
10895       NULL        346
10895       24          110
10895       39          45
10895       40          91
10895       60          100
```

← Total for all orders

← Total for each order

Totals are at the beginning of each group

Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

The "`order by`" clause should be removed.
Ordering will be handled implicitly by the "`group by`" clause.

```
select o.OrderID, o.ProductID, sum(o.Quantity) "Pieces"
  from [Order Details] o
 where o.OrderID in (10847,10895)
 group by o.OrderID, o.ProductID with ROLLUP
 --order by o.OrderID, o.ProductID
```

```
OrderID     ProductID   Pieces
----------- ----------- -----------
10847       1           80
10847       19          12
10847       37          60
10847       45          36
10847       60          45
10847       71          55
10847       NULL        288    ←Total is at the end of each group, improving usability
10895       24          110
10895       39          45
10895       40          91
10895       60          100
10895       NULL        346    ←
NULL        NULL        634    ←
```

```
select o.OrderID, o.ProductID, sum(o.Quantity) "Pieces"
  from [Order Details] o
 where o.OrderID in (10847,10895)
 group by o.OrderID, o.ProductID with CUBE
```

```
OrderID     ProductID   Pieces
----------- ----------- -----------
10847       1           80
10847       19          12
10847       37          60
10847       45          36
10847       60          45
10847       71          55
10847       NULL        288
10895       24          110
10895       39          45
10895       40          91
10895       60          100
10895       NULL        346
NULL        NULL        634
NULL        1           80
NULL        19          12
NULL        24          110
NULL        37          60
NULL        39          45
NULL        40          91
NULL        45          36
NULL        60          145    ←CUBE also inlcudes sums on the other dimension
NULL        71          55
```


ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

GROUPING will add a 0 or 1 column to the output. A 1 represents a row added by CUBE/ROLLUP. A 0 represents an original data row.

```
select
     o.OrderID,     GROUPING(o.OrderID),
     o.ProductID,   GROUPING(o.ProductID),
     sum(o.Quantity) "Pieces"
  from [Order Details] o
 where o.OrderID in (10847,10895)
 group by o.OrderID, o.ProductID with ROLLUP
```

```
OrderID           ProductID         Pieces
----------- ---- ----------- ---- -----------
10847       0    1           0    80
10847       0    19          0    12
10847       0    37          0    60
10847       0    45          0    36
10847       0    60          0    45
10847       0    71          0    55
10847       0    NULL        1    288
10895       0    24          0    110
10895       0    39          0    45
10895       0    40          0    91
10895       0    60          0    100
10895       0    NULL        1    346
NULL        1    NULL        1    634
```

Another Oracle difference is that instead of using NULLs as shown above, Oracle will replace the column value with the following grammar: "**All** <column heading>**s**".

```
OrderID         ProductID            Pieces
-----------     -----------     -----------
10847           1                        80
10847           19                       12
10847           37                       60
10847           45                       36
10847           60                       45
10847           71                       55
10847           All ProductIDs          288
10895           24                      110
10895           39                       45
10895           40                       91
10895           60                      100
10895           All ProductIDs          346
All OrderIDs    All ProductIDs          634
```

Rather than pursue analytical functions here, (and training the GUI to use them,) it would be better to use Analysis Services.

Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

CURSOR

By default, SQL Server cursors are SENSITIVE to changes that take place to the underlying tables. This means that changes that are committed to the database by a (different) user before a subsequent fetch are accurately reflected at fetch time. There is no keyword in the cursor definition to choose this behavior. However, this can be turned off with the INSENSITIVE keyword. Insensitive cursors populate a temporary object in tempdb in order to insulate it from base object changes. Subsequent fetches from an insensitive cursor are done from tempdb, not the user database. Therefore, INSENSITIVE and FOR UPDATE OF are mutually exclusive syntax.

(Note: as in Oracle, FOR UPDATE OF cursor syntax is paired with WHERE CURRENT OF DML syntax with subsequent INSERT or DELETE.)

```
declare C1 INSENSITIVE CURSOR for
  select FirstName
    from Employees
   where LastName < 'D'
   order by FirstName
  FOR UPDATE OF FirstName

Server: Msg 1048, Level 15, State 1, Line 7
Conflicting cursor options FOR UPDATE and INSENSITIVE.
```

A cursor can be CLOSEd to release the current result set and free any locks. However, its data structure and definition is still available and can be reactivated with an OPEN statement.
If a closed cursor's data structure and definition is no longer needed, it can be removed with a DEALLOCATE command. A cursor can be deallocated before it is closed. But, a deallocated cursor cannot be opened again without an appropriate declare.

SCROLL must be specified if fetch options other than NEXT are desired, (i.e., FIRST, LAST, PRIOR, RELATIVE +/- n, ABSOLUTE +/- n).

```
declare C1 INSENSITIVE SCROLL CURSOR for
  select FirstName
    from Employees
   where LastName < 'D'
   order by FirstName
  --FOR UPDATE of FirstName
declare @First varchar(10)
open C1
fetch LAST from C1 into @First
print @First
--close C1
deallocate C1

Steven
```


Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

FOR UPDATE is mutually exclusive with READ ONLY. However, this does not affect cursor sensitivity. A READ ONLY cursor can still be sensitive, and therefore hold a share lock.
The most benign (inexpensive) cursor is the INSENSITIVE . . . READ ONLY cursor especially if the transaction isolation level is set to READ UNCOMMITTED.

```
set transaction isolation level READ UNCOMMITTED -- dirty reads allowed
begin transaction T1
     [with mark 'Update Commission percentages'] --for distributed transaction recovery
declare C1 INSENSITIVE CURSOR for . . .
```

The above cursor functionality is based on SQL-92 syntax. However, additional Transaction-SQL extensions are given below. SQL-92 syntax has keywords before the CURSOR keyword. Extended syntax occurs between the CURSOR and FOR keywords. These two forms are mutually exclusive.

Cursors can be either LOCAL or GLOBAL. Neither is the default. Default behavior is shown by the IsLocalCursorsDefault database property of DATABASEPROPERTYEX system function. The value should be set by the ALTER DATABASE…CURSOR_DEFAULT command. Sp_dboption should no longer be used.

Cursors are either FORWARD_ONLY or SCROLL. If forward_only is specified only FETCH NEXT is allowed.

Cursors can be one of STATIC, KEYSET, DYNAMIC, or FAST_FORWARD.
STATIC is the extended way specifying INSENSITIVE.
KEYSET freezes primary keys into tempdb. Negative @@fetch_status becomes possible.
DYNAMIC is the extended way of explicitly specifying sensitive behavior.
FAST_FORWARD is a FORWARD_ONLY, READ_ONLY cursor. No SCROLL or FOR UPDATE.

Cursors can be READ_ONLY, SCROLL_LOCKS, or OPTIMISTIC.
READ_ONLY is extended syntax similar to FOR READ ONLY SQL-92 syntax.
SCROLL_LOCKS locks rows as they are read into the cursor, thereby assuring update/delete.
OPTIMISTIC does not lock rows read into the cursor, it compares timestamps for updateability.

@@FETCH_STATUS system function can have the following values:
 0 = Immediately previous Fetch was successful.
-1 = Immediately previous Fetch failed.
-2 = The requested row was deleted outside of the current KEYSET cursor.
-9 = The cursor may have been declared or opened, but has not yet been fetched.
Beware that @@FETCH_STATUS is reset by intermediate cursor logic even if disguised within called stored procedures. A practice like the following mimics Oracle functionality (C1%found is not reset but SQL%found is) and adds processing flexibility:

```
fetch last from C1 into @First
set @C1@@fetch_status = @@FETCH_STATUS
```


ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

LockWaitfor

WAITFOR can be used in two sessions simultaneously to control the timing of locks.
`Delay` is used for relative time positioning. `Time` is used for absolute time positioning.

```
select current_timestamp "Begin"
WAITFOR delay '000:00:15'
select current_timestamp "Delay"
WAITFOR time '15:42'
select current_timestamp "Time"

Begin
-----------------------
2002-02-19 15:41:17.903

Delay
-----------------------
2002-02-19 15:41:32.907

Time
-----------------------
2002-02-19 15:42:00.007
```

Notice that it takes a little bit of time for each statement to be processed.

LockRead

An UPDATE performed before a SELECT causes the SELECT to wait until the UPDATE is committed (or rolled back).

```
use pubs
begin transaction
  WAITFOR time '15:47'
  select current_timestamp "Before Update"
  UPDATE employee
     set minit = 'x'
   where emp_id = 'PMA42628M'

  WAITFOR delay '000:00:15'
  select current_timestamp "After Update"
  WAITFOR delay '000:00:05'
rollback transaction
select current_timestamp "After Rollback"

Before Update
-----------------------
2002-02-19 15:47:00.007

After Update
-----------------------
2002-02-19 15:47:15.010

After Rollback
-----------------------
2002-02-19 15:47:20.017
```

```
use pubs



WAITFOR time '15:47:02'
select current_timestamp "Before Select"
select * from employee where emp_id like 'PM%'
select current_timestamp "After Select"


Before Select
-----------------------
2002-02-19 15:47:02.000




emp_id    fname                minit lname
--------- -------------------- ----- ----------------
PMA42628M Paolo                M     Accorti

After Select
-----------------------
2002-02-19 15:47:20.017
```


ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

LockIsol

READ UNCOMMITTED allows the reading of dirty data without waiting.

```
use pubs
begin transaction
  WAITFOR time '16:05'
  select current_timestamp "Before Update"
  UPDATE employee
     set minit = 'x'
   where emp_id = 'PMA42628M'
  WAITFOR delay '000:00:05'
  select current_timestamp "After Update"
rollback transaction

Before Update
-----------------------
2002-02-19 16:05:00.000

After Update
-----------------------
2002-02-19 16:05:05.007
```

```
use pubs
set transaction isolation level
       READ UNCOMMITTED
       --READ COMMITTED
WAITFOR time '16:05:02'
select current_timestamp "Before Select"
select * from employee where emp_id like 'PM%'
select current_timestamp "After Select"

Before Select
-----------------------
2002-02-19 16:05:02.003

emp_id    fname                minit lname
--------- -------------------- ----- ------------
PMA42628M Paolo                x     Accorti

After Select
-----------------------
2002-02-19 16:05:02.003
```

READ COMMITTED requires the read to wait until the update is complete to get a clean read.

```
use pubs
begin transaction
  WAITFOR time '16:05'
  select current_timestamp "Before Update"
  UPDATE employee
     set minit = 'x'
   where emp_id = 'PMA42628M'
  WAITFOR delay '000:00:05'
  select current_timestamp "After Update"
rollback transaction

Before Update
-----------------------
2002-02-19 16:13:00.010

After Update
-----------------------
2002-02-19 16:13:05.017
```

```
use pubs
set transaction isolation level
       READ UNCOMMITTED
       --READ COMMITTED
WAITFOR time '16:05:02'
select current_timestamp "Before Select"
select * from employee where emp_id like 'PM%'
select current_timestamp "After Select"

Before Select
-----------------------
2002-02-19 16:13:02.003

emp_id    fname                minit lname
--------- -------------------- ----- ------------
PMA42628M Paolo                M     Accorti

After Select
-----------------------
2002-02-19 16:13:05.017
```


ENTEGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

LockDead

SQL Server handles deadlocks automatically. One transaction is chosen as a victim even though it is the earlier transaction. The remedy is to lock resources in the same order (and unlock in reverse order).

```
use pubs
set transaction isolation level READ COMMITTED
begin transaction
  WAITFOR time '16:23:01'
  select current_timestamp "Before First"
  UPDATE employee
     set minit = 'x' where emp_id = 'PMA42628M'
  WAITFOR delay '000:00:05'
  select current_timestamp "Before Second"
  UPDATE jobs
     set job_desc = 'Sr. ' + job_desc where job_id = 14
  WAITFOR delay '000:00:05'
rollback transaction
select current_timestamp "After Rollback"

Before First
-----------------------
2002-02-19 16:23:01.003

Before Second
-----------------------
2002-02-19 16:23:06.010

After Rollback
-----------------------
2002-02-19 16:23:12.520
```

```
use pubs
set transaction isolation level READ COMMITTED
begin transaction
  WAITFOR time '16:23'
  select current_timestamp "Before First"
  UPDATE jobs
     set job_desc = 'Sr. ' + job_desc where job_id = 14
  WAITFOR delay '000:00:05'
  select current_timestamp "Before Second"
  UPDATE employee
     set minit = 'x' where emp_id = 'PMA42628M'
  WAITFOR delay '000:00:05'
rollback transaction
select current_timestamp "After Rollback"

Before First
-----------------------------------------------------
2002-02-19 16:23:00.003

Before Second
-----------------------------------------------------
2002-02-19 16:23:05.010

Server: Msg 1205, Level 13, State 50, Line 1
Transaction (Process ID 55) was deadlocked on {lock}
resources with another process and has been chosen as
the deadlock victim. Rerun the transaction.
```


eNTeGREAT
Ron_Harmon@EnteGreat.com

PL/SQL and T-SQL; Birds of a Feather
Syntax Examples

LockCursor

SCROLL_LOCKS delays the direct update:

```
use pubs
--dbcc useroptions --to show isolation level
set transaction isolation level READ COMMITTED
WAITFOR time '08:32:00'
declare C1 cursor Local Forward_Only Dynamic
       SCROLL_LOCKS  --Direct update delayed
       --OPTIMISTIC  --Positioned update disallowed
    for
  select * from employee where emp_id = 'PMA42628M'
order by 1
  FOR UPDATE
begin transaction
open C1
fetch C1
WAITFOR delay '000:00:05'
update employee set minit = 'z' where CURRENT OF C1
commit transaction
close C1
deallocate C1

emp_id    fname                minit lname
--------- -------------------- ----- ----------
PMA42628M Paolo                M     Accorti
```

```
use pubs
set transaction isolation level READ COMMITTED
begin transaction
  WAITFOR time '08:32:02'
  update employee set minit = 'x' where emp_id =
'PMA42628M'
  select current_timestamp 'After x'
commit transaction
WAITFOR delay '000:00:10'
begin tran
  update employee set minit = 'M' where emp_id =
'PMA42628M'
commit tran

After x
-----------------------------------------------------
2002-02-20 08:32:05.330
```

OPTIMISTIC does not lock rows (no delay), but it disallows the CURRENT OF update:

```
set transaction isolation level READ COMMITTED
WAITFOR time '08:44:00'
declare C1 cursor Local Forward_Only Dynamic
       --SCROLL_LOCKS  --Direct update delayed
       OPTIMISTIC  --Positioned update disallowed
    for
  select * from employee where emp_id = 'PMA42628M'
order by 1
  FOR UPDATE
begin transaction
open C1
fetch C1
WAITFOR delay '000:00:05'
update employee set minit = 'z' where CURRENT OF C1
commit transaction
close C1
deallocate C1

emp_id    fname                minit lname
--------- -------------------- ----- -----------
PMA42628M Paolo                M     Accorti

Server: Msg 16947, Level 10, State 1, Line 15
No rows were updated or deleted.
The statement has been terminated.
```

```
use pubs
set transaction isolation level READ COMMITTED
begin transaction
  WAITFOR time '08:44:02'
  update employee set minit = 'x' where emp_id =
'PMA42628M'
  select current_timestamp 'After x'
commit transaction
WAITFOR delay '000:00:10'
begin tran
  update employee set minit = 'M' where emp_id =
'PMA42628M'
commit tran


After x
------------------------
2002-02-20 08:44:02.000
```


ENTEGREAT
Ron_Harmon@EnteGreat.com