Using Oracle Table Functions

Shiby Thomas
Oracle Corporation
Agenda

- What are table functions?
- Pipelined and parallel table functions
- Implementation approaches
  - Native PL/SQL approach
  - Interface approach
- Example
- Case study
Table Functions

- Table functions are arbitrary functions that return “virtual” tables
- They provide a set of rows as output like a normal table
  - seamless usage, in serial and parallel
  - incremental (pipelined) return of result sets
- Accepts set of rows as input
  - feed a table function directly with the result set of the previous one
  - no staging necessary
- The function can be implemented in C, Java or PL/SQL
Table Functions

Benefits

- Integrate additional functionality with the database
- Make the new functionality accessible via SQL
- More flexibility for transformation architecture
- Streaming of data from source to target
  - no interruption necessary
- Increased performance
- Efficient development
  - Reusability of transformation components
- Combine the power of procedural logic, parallelism and pipelining
Table Functions

transformation with staging

... CHANGES TO ...

pipelined, parallel transformation
Pipelined Table Functions

- Avoid the need for staging tables
- The returned collection is like a stream that can be fetched on demand
- Two mechanisms
  - Interface approach
  - Native PL/SQL mechanism
Ref Cursor Input

- Ref cursor can be used to fetch the input rows
- Can be used with interface or native PL/SQL approaches
- Example: StockPivot function
  - Converts a row (Ticker, OpenPrice, ClosePrice) to (Ticker, PriceType, Price)
  - (“ORCL”, 12, 13) would generate (“ORCL”, “O”, 12) and (“ORCL”, “C”, 13)
CREATE TYPE InputCurType IS REF CURSOR RETURN StockTable%ROWTYPE;
END;

CREATE TYPE TickerType AS OBJECT
  ( ticker VARCHAR2(4),
    PriceType VARCHAR2(1),
    price NUMBER
  );

CREATE TYPE TickerTypeSet AS TABLE OF TickerType;

CREATE FUNCTION StockPivot(p InputCurType) RETURN TickerTypeSet PIPELINED IS <body>;
create function StockPivot(p InputCurType) return TickerTypeSet pipelined is
    out_rec TickerType;
begin
    for in_rec in p loop
        -- first row
        out_rec.ticker := in_rec.Ticker;
        out_rec.PriceType := "O";
        out_rec.price := in_rec.OpenPrice;
        push out_rec;
        -- second row
        out_rec.PriceType := "C";
        out_rec.Price := in_rec.ClosePrice;
        push out_rec;
    end loop;
    return;
end;
Native PL/SQL Implementation

- The **push** statement pipelines data out
- Must have return statements that do not return any values – to transfer control back to the consumer
- No push statement => return no rows
Pipelining between PL/SQL TFs

```
select * from table(f(cursor(select * from table(g()))));
```

- Results will be pipelined from function `g` to function `f`
Interface Implementation

- Define a type that implements the ODCITable interface
- The interface consists of a set of well-defined routines
  - ODCITableStart;
  - ODCITableFetch;
  - ODCITableClose;
CREATE TYPE StockPivotImpl AS OBJECT
(
  cur InputCurType;
  static function ODCITableStart(sctx OUT StockPivotImpl, 
    p InputCurType) return number,
  member function ODCITableFetch(self IN OUT 
    StockPivotImpl,nrows IN number, outrows OUT 
    TickerTypeSet) return number,
  member function ODCITableClose(self IN StockPivotImpl) 
    return number
);
Start Routine

- First routine invoked to begin retrieving rows from a TF
- Performs the set up needed for the scan
- Creates the scan context
- Arguments to the TF are passed in as parameters to this routine

```sql
STATIC FUNCTION ODCITableStart(sctx OUT <imptype>, <args>)
    RETURN NUMBER;
```
Fetch Routine

- Invoked one or more times to retrieve all the rows from the TF
- Scan context passed in as a parameter
- nrows argument specifies the number of rows requested by the consumer
- Returns a null collection at the end

```
MEMBER FUNCTION ODCITableFetch(self IN OUT <imptype>,
nrows IN NUMBER, rws OUT <coll-type>) RETURN NUMBER;
```
Close Routine

- Invoked after the last fetch invocation
- Performs the necessary clean up

```plaintext
MEMBER FUNCTION ODCITableClose(self IN <imptype>)
    RETURN NUMBER;
```
CREATE TYPE BODY StockPivotImpl IS
-- START routine
static function ODCITableStart(sctx OUT StockPivotImpl, p InputCurType) return number is
begin
  sctx.cur := p; -- save ref cursor in scan context
  return ODCICConst.Success;
end;
-- FETCH routine
member function ODCITableFetch(nrows IN number, outrows OUT TickerTypeSet) return number is
  in_rec StockTable%ROWTYPE;
  idx integer := 1;
begin
  outrows.extend(100); -- we have chosen to return 100 rows at a time
  while (idx <= 100) loop
    fetch self.cur into in_rec;
    -- first row
    outrows(idx).ticker := in_rec.Ticker;
    outrows(idx).PriceType := "O";
    outrows(idx).price := in_rec.OpenPrice;
    idx := idx + 1;
    -- second row
    ...
  end loop;
  return ODCIConst.Success;
end;
Example (contd.)

-- CLOSE routine (no action required)
member function ODCITableClose return number is
begin
    return ODCIConst.Success;
end;
end; -- end of type body
Querying Table Functions

- Pipelined TFs can be used in the FROM clause of SELECT statements
- Usage is the same regardless of implementation approach

```
SELECT x.Ticker, x.Price
FROM TABLE(StockPivot(CURSOR(SELECT * FROM StockTable))) x
WHERE x.PriceType='C';
```
Parallel Table Functions

- Supported by integrating with Oracle’s parallel execution infrastructure
- TFs will NOT need to be coded differently for parallel execution
- Same model of data partitioning as PQ framework
Input Data Partitioning

- Specified for exactly one input parameter
- Partitioning method can be range or hash

```
create function f(p refcur_type) return rec_tab_type
    pipelined parallel_enable(p partition by [<method> (<column list>) | ANY ])
is
begin ... end;
```
Case Study: BLAST Implementation

- Sequence matching functionality for gene/protein sequences
- Implemented via interface approach
- Code written in C
- Trusted table function: code linked with the RDBMS executable
- Executes in the same address space as the RDBMS
CREATE OR REPLACE FUNCTION blastn_match (  
    query_sequence        IN CLOB,  
    seqdb_cursor             blast_cur.seqdbcur_t,  
    subsequence_from      IN INTEGER  DEFAULT NULL,  
    subsequence_to        IN INTEGER  DEFAULT NULL,  
    filter_low_complexity IN INTEGER  DEFAULT 0,  
    mask_lower_case       IN INTEGER  DEFAULT 0,  
    expect_value          IN NUMBER   DEFAULT 10,  
    open_gap_cost         IN INTEGER  DEFAULT 5,  
    extend_gap_cost       IN INTEGER  DEFAULT 2,  
    mismatch_cost         IN INTEGER  DEFAULT -3,  
    match_reward          IN INTEGER  DEFAULT 1,  
    word_size             IN INTEGER  DEFAULT 11  
)  
RETURN dmbmos AUTHID CURRENT_USER  
PIPELINED PARALLEL_ENABLE (seqdb_cursor PARTITION BY ANY)  
USING dmbmnimp;  

GRANT EXECUTE ON blastn_match TO PUBLIC;
CREATE OR REPLACE TYPE dmbmnimp AUTHID CURRENT_USER AS
  OBJECT (key RAW(4), shared_ctx RAW(4),
  STATIC FUNCTION ODCITableStart (sctx IN OUT dmbmnimp,
    rws_ptr IN RAW, query_sequence IN CLOB,
    seqdb_cursor SYS_REFCURSOR, ......) RETURN NUMBER,
  MEMBER FUNCTION ODCITableFetch (self IN OUT dmbmnimp, nrows IN NUMBER,
    outset OUT dmbmos) RETURN NUMBER,
  MEMBER FUNCTION ODCITableClose (self IN dmbmnimp) RETURN NUMBER,
  STATIC FUNCTION dmbmnStartStub (sctx IN OUT dmbmnimp,
    rws_ptr IN RAW, query_sequence IN CLOB,
    seqdb_cursor SYS_REFCURSOR, ......) RETURN NUMBER,
  MEMBER FUNCTION dmbmnFetchStub (self IN OUT dmbmnimp, nrows IN NUMBER,
    outset OUT dmbmos) RETURN NUMBER,
  MEMBER FUNCTION dmbmnCloseStub (self IN dmbmnimp) RETURN NUMBER
);

CREATE OR REPLACE TYPE BODY dmbmnimp IS
  STATIC FUNCTION ODCITableStart (......) RETURN NUMBER IS
    BEGIN
      RETURN dmbmnimp.dmbmnStartStub (......);
    END;
  STATIC FUNCTION dmbmnStartStub (......) RETURN NUMBER IS
    LANGUAGE C
    LIBRARY DMBLAST_LIB
    NAME "dmbm_n_start"
    WITH CONTEXT
    PARAMETERS (......);
    -- Similar for fetch and close
    END;

GRANT EXECUTE ON dmbmnimp TO PUBLIC WITH GRANT OPTION;
Output Type Definitions

-- DM Blast Match Output (dmbmo)
--
CREATE OR REPLACE TYPE dmbmo AS OBJECT (
    t_seq_id VARCHAR2(4000),
    score    INTEGER,
    expect   NUMBER
);
/
/

-- DM Blast Match Output Set (dmbmos)
--
CREATE OR REPLACE TYPE dmbmos AS TABLE OF dmbmo;

GRANT EXECUTE ON dmbmnimp TO PUBLIC WITH GRANT OPTION;
Coding

• Write code for the functions
  - dmbm_n_start
  - dmbm_n_fetch
  - dmbm_n_close

• Compile and link the code into the library DMBLAST_LIB

• Link the library with the RDBMS executable
Summary

- Pipelined and parallel table functions
- Implementation approaches
- Examples