Table Functions

- Traditionally, once we'd loaded the flat file data into a staging table via a SQL*Loader script, the next stage would be to transform the data, either through a series of DML commands or by using a PL/SQL package.
- The transformation would have one or more stages, and the stages would run sequentially, one after the other.
- Oracle 9i now gives us the opportunity to improve on this situation, by allowing us to create our transformations as functions - functions that take our external table as an input, and output the transformed data as rows as columns that can be used to update a table. These functions, known as Table Functions
- **Table functions** are defined as functions that can produce a set of rows as output. In other words, table functions return a collection type instance

Pipelined Table Functions

- Oracle9*i*, Release 1 (9.0.1), allows table functions to **pipeline** results (return results iteratively) out of the functions.
- By pipelining table functions, you can string a number of them together and have them 'pass off' rows to the next process as soon as a batch of rows are transformed, meeting our requirement to minimize our inter-process wait states.

- Simple Example - Generating Some Random Data
- To create six unique random numbers between 1 and 49 with one SQL statement
- The solution without a pipelined function is as follows:-

-
  ```
  select r
  from (select r
  from (select rownum r
  from all_objects
  where rownum < 50)
  order by dbms_random.value)
  where rownum <= 6;
  ```

- That query works by generating the numbers 1 .. 49, using the inline view.
- We wrap that innermost query as an inline view and sort it by a random value, using DBMS_RANDOM.VALUE.
- We wrap that result set in yet another inline view and just take the first six rows.
- If we run that query over and over, we'll get a different set of six rows each time.
- For example, we'd like the inclusive set of all dates between 25-FEB-2004 and 10-MAR-2004. The question becomes how to do this without a "real" table, and the answer lies in Oracle9i/10g with its **PIPELINED function** capability.

- We can write a PL/SQL function that will operate like a table.
- We need to start with a SQL collection type; this describes what the PIPELINED function will return.
- In this case, we are choosing a table of numbers; the virtual table we are creating will simply return the numbers 1, 2, 3, ... N:

```
create type array2
as table of number
/
```

```
Type created.
```

- Next, we create the actual PIPELINED function. This function will accept an input to limit the number of rows returned. If no input is provided, this function will just keep generating rows for a very long time

The PIPELINED keyword on line 4 allows this function to work as if it were a table:

```
create function
    gen_numbers(n in number default null)
    return array
    PIPELINED
    as
    begin
    for i in 1 .. nvl(n,999999999)
    loop
    pipe row(i);
    end loop;
    return;
    end;

    /
Function created.
```

Suppose we needed three rows for something. We can now do that in one of two ways:

```
select * from TABLE(gen_numbers(3));
COLUMN_VALUE
    ------------
    1
    2
    3
```

or

```
select * from TABLE(gen_numbers)
    where rownum <= 3;
COLUMN_VALUE
    ------------
    1
    2
    3
```

Now we are ready to re-answer the original question, using the following functionality:

```
select *
    from (
    select *
    from (select * from table(gen_numbers(49)))
    order by dbms_random.random
    )
    where rownum <= 6
    /
COLUMN_VALUE
    ------------
    47
    42
    40
    15
    48
    23
```

We can use this virtual table functionality for many things, such as generating that range of dates:

```
select to_date('25-feb-2004')+
    column_value-1
    from TABLE(gen_numbers(15))
    /
```

Steps to perform when using PL/SQL Table Functions

- 1. The producer function must use the PIPELINED keyword in its declaration.
- 2.The producer function must use an OUT parameter that is a record, corresponding to a row in the result set.
- 3. Once each output record is completed, it is sent to the consumer function through the use of the PIPE ROW keyword.
- 4.The producer function must end with a RETURN statement that does not specify any return value.
- 5.The consumer function or SQL statement then must use the TABLE keyword to treat the resulting rows from the PIPELINE function like a regular table.