Two files tnsnames.ora and listerner.ora

Listerner.ora exists only on server. It configures listerner, tells listerner how to configure request. Requests coming in from client. client could be running any thing a browser,

application, etc. Anything that needs to communicate with oracle db needs to go through this listerner.

tnsnames.ora is going to exist on both client machine and server.

OracleHome/Network/ADMIN/listerner.ora

OracleHome/Network/ADMIN/tnsnames.ora

tnsnames.ora => it encapsulates all the different ways that we are going to talk to db.

Transparent Network Substrate (TNS) -

TNS supports homogenous peer-to-peer connectivity on top of other networking technologies such as TCP/IP.

TNS operates mainly for connection to Oracle databases.

TNSNAMES.ORA is a SQL\*Net configuration file that defines databases addresses for establishing connections to them.

This file normally resides in the ORACLE HOME\NETWORK\ADMIN directory, but location can be defined by the TNS\_ADMIN environment variable.

Add the following entry in your TNSNAMES.ORA file and change the values shown in brackets to suit your environment:

<addressname> =

(DESCRIPTION =

(ADDRESS\_LIST =

(ADDRESS = (PROTOCOL = TCP)(Host = <hostname>)(Port = <port>))

)

(CONNECT\_DATA =

(SERVICE\_NAME = <service\_name>)

)

)

Here is a completed example:

ORA11 =

(DESCRIPTION =

(ADDRESS\_LIST =

(ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.0)(PORT = 1521))

)

(CONNECT\_DATA =

(SERVICE\_NAME = ORA12)

)

)

TNSNAMES.ORA files are located on both client and server systems.

utl\_file predefined package.

Introduced in 7.3 onwards.

Used to load and read data from os files.

Alias director. plsql doesn't directly interact with

create any directory perssion needed....admin can give

grant create any directory to username;

grant read, write on directory directoryname(xyz) to username;

create or replace directory xyz as 'c:\';

writing data into an os file - putf(), put\_line()

step 1- declare a varaible (file pointer varaible using 'file\_type')

varname utf\_file.file\_type;

step 2- open a file ('fopen()')

file\_pointer\_var := utl\_file.fopen(alias\_directory, filename, mode); (mode - w, r, a(append))

step 3- writing data into file

utf\_file.putf(file\_pointer\_var, 'content');

step 4- closing a file (fclose());

utl\_file.fclose(file\_pointer\_var);

declare

fp utl\_file.file\_type;

begin

fp:= utl\_file.fopen('xyz', 'file1.txt', 'w');

utl\_file.putf(fp,'abc0');

utl\_file.fclose(fp);

end;

select value

from v$parameter

where name = 'utl\_file\_dir';

We can also execute a java file through dbms\_job

how to monitor jobs - Below are tables through which we can monitor running jobs -

select \* from all\_scheduler\_running\_jobs where job\_name='first\_job';

select log\_id, to\_char(log\_date, 'dd/mm'yyyy') log\_date

,substr(job\_name,1, 20) job\_name

,substr(status,1,10) status

,additional\_info operation

from dba\_scheduler\_job\_log where job\_name like '%FIRST%';

select \* from dba\_scheduler\_jobs where job\_name='FIRST\_JOB';

How debug plsql blocks in plsql developer -

You need to have below privileges -

grant debug connect session to Aniket;

grant debug any procedure to Aniket;

(grant above privileges from sysuser(super user))

To debug right click on procedure and say test.

Enter input parameter and press f9.

Then we can step in step out.

Can also use breakpoints.

Make sure that when you pass the directory name to your stored procedure that it is UPPER CASE - it's often a thing that gets overlooked

Synonym – A synonym is an alternative name for object such as tables, views, sequence, stored procedure and other database object.

You can generally use synonym when you are granting access to an object from another schema and you don’t want the user to have worry about knowing which schema own the object.

Syntax – create or replace synonym synonym\_name for object\_name;

Example –

Select \* from employee;

Create synonym emp\_sym for employee;

Select \* from emp\_sym;

Drop synonym emp\_sym;

Database schema – the description of database is called the database schema.

Oracle schema – is oracle user who can be used to store the object.

Create user finance identified by fin111 //user created

Grant connect to finance; //some permissions given to user

Grant unlimited tablespace to finance; //unlimited tablespace means user can create object in any tablespaces. In real life we create a tablespace for this user and then we allocate that tablespace to this user.

Grant create table to finance; //user can create tables.

When user starts owning a object(like tables) that user becomes a schema.

Grant select on finance.tab1 to Rajini;

Difference between an oracle user and oracle schema –

Oracle user is a user which is generally a person who will use an account to login.

Now companies have peoples, so we can have oracle users representing these peoples. But people leave companies so the data will remain under the username(accessible by dba). Once the oracle user is created he can create objects in that user. (We can have generic user like finance). Once objects are created under a user we can grant access to these objects to other users.

When user starts owning a object(like tables) that user becomes a schema.

Schema is a generic user which mainly holds data. Schema starts as an oracle user and then it owns data.

**Oracle SQL - Quick overview of Oracle RDBMS Architecture -** [Sam Dhanasekaran](https://www.youtube.com/channel/UC313K63WEDXCZE3-9lzXosQ)

https://www.youtube.com/watch?v=367H3z6Ijzo&index=6&list=PLgmhWhDgxKIENmD8yF4yvQC041jsw\_KOn

Oracle software is there on C:\app\inarajp\product

Oracle Database which we create is nothing but bunch of files.. its there on C:\app\inarajp\oradata -> SYSTEM01.DBF , SYSTEM02.DBF

When we start up an oracle database two things happens –

First is, an empty oracle instance gets created in memory. Then that instance opens the datafiles related to database(C:\app\inarajp\oradata -> SYSTEM01.DBF , SYSTEM02.DBF) and then database will be available.

When an oracle database is open to access it u need to login as an user. We need to have database user account.

When users create tables they create then in a logical structure called tablespace. A tablespace can be made up of one or more data files. From os perspective we can see only datafiles but if we login into database we can see tablespaces. Objects such as table are owned by users. User1.test and User2.test both these objects of different users can be stored on same tablespace. Since User1 and User2 owns an object they can be called as a schema. User1 can be called as schema and User2 can be called as schema. User3 doesn’t owns an object thus it is just an user.

A schema user can also be called as a owner of the object. By default owners have all the privileges for the objects. They can also grant access to there objects to other users. There will be powerful users as DBA. DBAs can do anything with any users objects.

Autonomous\_Transaction –

An autonomous transaction is an independent transaction that is initiated by another transaction, and executes without interfering with the parent transaction.

An commit/rollback in the autonomous transaction would not effect the parent transaction, also a commit/rollback in the parent transaction would not effect the autonomous transaction.

In oracle PL/SQL, PRAGMA refers to a compiler directive or “hint” it is used to provide an instruction to the compiler.

Pragma directives are process at compile time where they pass necessary information to the compiler; they are not processed at runtime.

(transaction within transaction, independent of parent transaction)

Where is it used? –

Mostly it will be used in the error logging procedures, where we need to rollback the parent transaction when an error occurs and still we need to record the error in the log table.

Can be used in a trigger to commit/rollback, or when a trigger is calling a procedure which has a commit/rollback in it.

Example –

Create table error\_log(error\_date date, error\_msg varchar2(1000), error\_code varchar2(100));

--create a error logging autonomous\_transaction procedure

Create or replace procedure p\_log\_error(p\_msg varchar2, p\_code varchar2)

Is

Pragma autonomous\_transaction,

Begin

Insert into error\_log(…) values(…);

Commit;

End;

--PL/SQL block that calls the autonomous\_transaction procedure

Set serveroutput on;

Declare

P\_F\_name employee.first\_name%type;

P\_L\_name employee.last\_name%type;

Begin

Update employees set salary = salary \* 333 where first\_name like ‘Steven’;

Select first\_name, last\_name into p\_f\_name, p\_l\_name from employees where first\_name like ‘Steven’;

Dbms\_output.put\_line(‘name’ || p\_f\_name||’ ’||p\_l\_name);

Exception

When other then

p\_log\_error(substr(sqlerrm, 1, 2000), SQLCODE);

Rollback;

END;