```
--Creating a custom tablespace

create tablespace mydata datafile '+data' size 1G;

--Creating 2 new users

create user myapp identified by MyApp2025#_;

grant DBA to myapp;
alter user myapp quota unlimited on mydata;
alter user myapp quota unlimited on USERS;
--run as sys
grant execute on DBMS_SHARED_POOL to myapp;
grant create any directory to myapp;
grant create any library to myapp;

--
create user myapprun identified by MyApp2025#_;

grant DBA to myapprun;
alter user myapprun quota unlimited on mydata;
alter user myapprun quota unlimited on USERS;

--

alter session set current_schema=myapp;

--Creating an External Directory
CREATE OR REPLACE DIRECTORY ext_tab_data AS '/external_data';

--Creating an External Table
CREATE TABLE myapp.countries_ext (
  country_code      VARCHAR2(5),
  country_name      VARCHAR2(50),
  country_language  VARCHAR2(50)
)
ORGANIZATION EXTERNAL (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY ext_tab_data
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    FIELDS TERMINATED BY ','
    MISSING FIELD VALUES ARE NULL
    (
      country_code      CHAR(5),
      country_name      CHAR(50),
      country_language  CHAR(50)
    )
  )
  LOCATION ('Countries1.txt','Countries2.txt')
)
PARALLEL 5
REJECT LIMIT UNLIMITED;


--Creating a table with encrypted column
CREATE TABLE myapp.employee_app ( first_name VARCHAR2(128), last_name
VARCHAR2(128), empID NUMBER, salary NUMBER(6) ENCRYPT);

--Creating a table with BasicLob file
ALTER Session SET db_securefile=NEVER;

CREATE TABLE myapp.MyInfoTB (
    id          NUMBER GENERATED BY DEFAULT AS IDENTITY,
```

```
    name        VARCHAR2(100),
    text_data   CLOB,
    binary_data BLOB,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT sample_lob_table_pk PRIMARY KEY (id)
);

ALTER Session RESET db_securefile;


--Creating a Cluster Table
CREATE CLUSTER myapp.emp_dept (deptno NUMBER(3))
   SIZE 600
   STORAGE (INITIAL 200K
      NEXT 300K
      MINEXTENTS 2
      PCTINCREASE 33) tablespace mydata;

CREATE TABLE myapp.dept (
   deptno NUMBER(3) PRIMARY KEY )
   CLUSTER myapp.emp_dept (deptno) ;

CREATE TABLE myapp.empl (
   emplno NUMBER(5) PRIMARY KEY,
   emplname VARCHAR2(15) NOT NULL,
   deptno NUMBER(3) REFERENCES myapp.dept)
   CLUSTER myapp.emp_dept (deptno)
;

CREATE INDEX myapp.emp_dept_index
   ON CLUSTER myapp.emp_dept
   STORAGE (INITIAL 50K
      NEXT 50K
      MINEXTENTS 2
      MAXEXTENTS 10
      PCTINCREASE 33) TABLESPACE mydata;

--Creating a ROWID columns
CREATE TABLE myapp.employee_r ( first_name VARCHAR2(128), last_name
VARCHAR2(128), empID NUMBER, salary NUMBER(6), colref rowid);

--Creating an Index Organized Table
CREATE TABLE myapp.locations
(id           NUMBER(10),
 description  VARCHAR2(50)  NOT NULL,
 map          BLOB,
 CONSTRAINT pk_locations PRIMARY KEY (id)
)
ORGANIZATION INDEX
PCTTHRESHOLD 20
INCLUDING description
OVERFLOW TABLESPACE mydata;

--Creating a Scheduler Job to run an external script
CREATE OR REPLACE DIRECTORY script_dir AS '/home/oracle/scripts/';

BEGIN
    DBMS_SCHEDULER.create_job (
        job_name        => 'myapp.process_data_job',               -- Job name
        job_type        => 'EXECUTABLE',                      -- Type of job
(EXTERNAL)
        job_action      => '/bin/sh',                          -- Program to run
(shell)
        number_of_arguments => 1,                              -- Number of
```

```
arguments passed to the executable
        enabled         => FALSE,                                   -- Enable the job
        repeat_interval => 'FREQ=DAILY; BYHOUR=0; BYMINUTE=0', -- Schedule to
run daily at midnight
        comments        => 'Job to execute the shell script process_data.sh'
    );

    -- Add the argument for the shell script
    DBMS_SCHEDULER.set_job_argument_value (
        job_name        => 'myapp.process_data_job',
        argument_position => 1,
        argument_value   => '/home/oracle/scripts/process_data.sh'
    );
END;
/

--Creating a Procedure taht calls DBMS_SHARED_POOL
CREATE OR REPLACE PROCEDURE myapp.manage_shared_pool(
    p_package_name IN VARCHAR2,   -- Name of the package to pin or unpin
    p_action        IN VARCHAR2    -- Action to perform: 'PIN' or 'UNPIN'
) IS
BEGIN
    -- Check the action and perform pinning or unpinning accordingly
    IF p_action = 'PIN' THEN
        -- Pin the package in the shared pool
        DBMS_SHARED_POOL.keep(p_package_name);
        DBMS_OUTPUT.put_line('Package ' || p_package_name || ' has been pinned
in the shared pool.');

    ELSIF p_action = 'UNPIN' THEN
        -- Unpin the package from the shared pool
        DBMS_SHARED_POOL.unkeep(p_package_name);
        DBMS_OUTPUT.put_line('Package ' || p_package_name || ' has been unpinned
from the shared pool.');

    ELSE
        -- Handle invalid action
        DBMS_OUTPUT.put_line('Invalid action: ' || p_action || '. Please use
''PIN'' or ''UNPIN''.');
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.put_line('Error occurred: ' || SQLERRM);
END;
/


--Creating a Library

--create file on OS: /home/oracle/libexample.so
/*
#include <stdio.h>

void hello_world() {
    printf("Hello, World from External Library!\n");
}
*/

CREATE OR REPLACE LIBRARY myapp.example_lib AS '/home/oracle/libexample.so';
/

CREATE OR REPLACE PROCEDURE myapp.hello_world as
  EXTERNAL NAME "hello_world@libexample"
```

```
  LIBRARY myapp.example_lib
  LANGUAGE C;
/


--Creating a table using XMLTYPE
CREATE TABLE myapp.xml_data_table (
    id NUMBER PRIMARY KEY,
    xml_data XMLTYPE
)
  XMLTYPE xml_data STORE AS BINARY XML;

--

CREATE TABLE myapp.xml_clob
(
   xml_id    NUMBER,
   xml_doc   SYS.XMLTYPE
)
XMLTYPE xml_doc STORE AS CLOB;


--Creating a table using Spatial
CREATE TABLE myapp.spatial_table (
    id NUMBER PRIMARY KEY,
    location SDO_GEOMETRY
);


--Creating a procedure that uses UTL_HTTP
CREATE OR REPLACE PROCEDURE myapp.get_http_response(p_url IN VARCHAR2,
p_response OUT VARCHAR2) AS
    l_http_req   UTL_HTTP.req;
    l_http_resp  UTL_HTTP.resp;
    l_buffer     VARCHAR2(4000);
BEGIN
    -- Initialize response variable
    p_response := '';

    -- Open a connection to the external URL
    l_http_req := UTL_HTTP.begin_request(p_url, 'GET', 'HTTP/1.1');

    -- Set headers if needed (optional)
    UTL_HTTP.set_header(l_http_req, 'User-Agent', 'Oracle/UTL_HTTP');

    -- Send the request and get the response
    l_http_resp := UTL_HTTP.get_response(l_http_req);

    -- Read the response data
    LOOP
        BEGIN
            UTL_HTTP.read_text(l_http_resp, l_buffer);
            p_response := p_response || l_buffer;  -- Append the buffer to the
output response
        EXCEPTION
            WHEN UTL_HTTP.end_of_body THEN
                EXIT;  -- Exit when the response body has been fully read
        END;
    END LOOP;

    -- Close the HTTP response
    UTL_HTTP.end_response(l_http_resp);

EXCEPTION
```

```
    WHEN OTHERS THEN
        -- Handle any errors
        p_response := 'Error: ' || SQLERRM;
END get_http_response;
/

--Creating a Java Object
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED myapp.helloClassSRC AS
public class HelloClass {
    public static String hello( String who )  {
        return "hello, " + who ;
    }
}

/

--Creating a XML Schema
alter session set current_schema=myapp;

begin
 dbms_xmlschema.registerSchema(
  'http://www.example.com/xwarehouses.xsd',
  '<schema xmlns="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.example.com/xwarehouses.xsd"
       xmlns:who="http://www.example.com/xwarehouses.xsd"
       version="1.0">

  <simpleType name="RentalType">
   <restriction base="string">
    <enumeration value="Rented"/>
    <enumeration value="Owned"/>
   </restriction>
  </simpleType>

  <simpleType name="ParkingType">
   <restriction base="string">
    <enumeration value="Street"/>
    <enumeration value="Lot"/>
   </restriction>
  </simpleType>

  <element name = "Warehouse">
    <complexType>
     <sequence>
      <element name = "WarehouseId"   type = "positiveInteger"/>
      <element name = "WarehouseName" type = "string"/>
      <element name = "Building"      type = "who:RentalType"/>
      <element name = "Area"          type = "positiveInteger"/>
      <element name = "Docks"         type = "positiveInteger"/>
      <element name = "DockType"      type = "string"/>
      <element name = "WaterAccess"   type = "boolean"/>
      <element name = "RailAccess"    type = "boolean"/>
      <element name = "Parking"       type = "who:ParkingType"/>
      <element name = "VClearance"    type = "positiveInteger"/>
     </sequence>
    </complexType>
  </element>
</schema>',
   TRUE, TRUE, FALSE, FALSE);
end;
/

CREATE TABLE xwarehouses OF XMLTYPE
   XMLSCHEMA "http://www.example.com/xwarehouses.xsd"
```

```
   ELEMENT "Warehouse";


--Creating a table with a FK in another schema
CREATE TABLE myapp.TB_Parent (
    ParentID NUMBER PRIMARY KEY,
    Name VARCHAR2(100) NOT NULL
) tablespace mydata;

grant references on myapp.TB_Parent to myapprun;

CREATE TABLE myapprun.TB_Child (
    ChildID NUMBER PRIMARY KEY,
    ParentID NUMBER,
    Name VARCHAR2(100) NOT NULL,
    CONSTRAINT fk_parent FOREIGN KEY (ParentID) REFERENCES
myapp.TB_Parent(ParentID) ON DELETE CASCADE
);

--Creating a profile using a custom password verify function
CREATE PROFILE myprof LIMIT
   SESSIONS_PER_USER          UNLIMITED
   CPU_PER_SESSION            UNLIMITED
   CPU_PER_CALL               3000
   CONNECT_TIME               45
   LOGICAL_READS_PER_SESSION  DEFAULT
   LOGICAL_READS_PER_CALL     1000
   PRIVATE_SGA                15K
   COMPOSITE_LIMIT            5000000;


CREATE OR REPLACE FUNCTION sys.custom_password_verify (
    username   VARCHAR2,
    password   VARCHAR2,
    old_password VARCHAR2
) RETURN BOOLEAN IS
    -- Define password complexity requirements
    min_length CONSTANT NUMBER := 8;
    uppercase_count NUMBER := 0;
    lowercase_count NUMBER := 0;
    digit_count NUMBER := 0;
    special_char_count NUMBER := 0;
BEGIN
    -- Check minimum length
    IF LENGTH(password) < min_length THEN
        RAISE_APPLICATION_ERROR(-20001, 'Password must be at least ' ||
min_length || ' characters long.');
    END IF;

    -- Check for complexity: uppercase, lowercase, digit, and special character
    FOR i IN 1 .. LENGTH(password) LOOP
        IF REGEXP_LIKE(SUBSTR(password, i, 1), '[A-Z]') THEN
            uppercase_count := uppercase_count + 1;
        ELSIF REGEXP_LIKE(SUBSTR(password, i, 1), '[a-z]') THEN
            lowercase_count := lowercase_count + 1;
        ELSIF REGEXP_LIKE(SUBSTR(password, i, 1), '[0-9]') THEN
            digit_count := digit_count + 1;
        ELSIF REGEXP_LIKE(SUBSTR(password, i, 1), '[!@#$%^&*()]') THEN
            special_char_count := special_char_count + 1;
        END IF;
    END LOOP;

    IF uppercase_count = 0 OR lowercase_count = 0 OR digit_count = 0 OR
special_char_count = 0 THEN
```

```
        RAISE_APPLICATION_ERROR(-20002, 'Password must contain at least one
uppercase letter, one lowercase letter, one digit, and one special character.');
    END IF;

    -- Prevent username in password
    IF INSTR(LOWER(password), LOWER(username)) > 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'Password cannot contain the
username.');
    END IF;

    -- Prevent reuse of the old password
    IF old_password IS NOT NULL AND password = old_password THEN
        RAISE_APPLICATION_ERROR(-20004, 'New password cannot be the same as the
old password.');
    END IF;

    RETURN TRUE;
END custom_password_verify;
/

ALTER PROFILE myprof LIMIT PASSWORD_VERIFY_FUNCTION custom_password_verify;

--Adding SQL PATCH and SQL PLAN Baselines
select o.order_id, order_status
  from co.orders o
      ,co.customers c
 where o.customer_id=c.customer_id
   and o.order_status = 'CANCELLED';

select * from v$sql where sql_text like '%CANCELLED%';

variable x varchar2(100);
exec :x:=dbms_sqldiag.create_sql_patch(sql_id=>'2utmd7t8q4z06',
hint_text=>'optimizer_features_enable(''11.2.0.4'')', name=>
'SQL_Patch_11.2.0.4');

--
select o.order_id, order_status, s.store_name
  from co.orders o
      ,co.stores s
 where o.store_id=s.store_id
   and s.store_name = 'Tokyo';


select * from v$sql where sql_text like '%Tokyo%';

6mfsb0z8bnsnk

DECLARE
    CNT NUMBER;
BEGIN
    --Manuel plan loading
    CNT := DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(SQL_ID=>'6mfsb0z8bnsnk');
END;
/


--Creating a JOB using DMBS_JOB

declare
  l_job  pls_integer;
begin
  dbms_job.submit (
    job       => l_job,
```

```
    what      => 'begin null; end;',
    next_date => trunc(sysdate)+1,
    interval  => 'trunc(sysdate)+1'
  );
end;
/
commit
/
```