LIST OF CONTENTS

RUN A SCRIPT FILE

NOTE: NO spaces in file or folder names and in the run command

```
@drive:\folder_name\file_name.extension
START drive:\folder_name\file_name.extension
```

CALL A SCRIPT FILE

```
EDIT drive:\folder_name\file_name.extension
```

COMMENT CODE

```
/* Name, version, date */  or  -- enter run path
```

CHANGE A PASSWORD

```
ALTER USER your_user_id IDENTIFIED BY your_new_password;
```

TABLES

CREATE TABLES

```
CREATE TABLE table_name (
    column_name1  DATATYPE CONSTRAINT DEFAULT,
    column_name2  DATATYPE,
    column_name3 object_type);
```

DROP TABLES

```
DROP TABLE table_name ;
DROP TABLE table_name  PURGE;
```

EMPTY RECYCLEBIN

```
PURGE RECYCLEBIN;
```

RESTORE DROPPED TABLES

```
FLASHBACK TABLE "recyclebin_table_name" TO BEFORE DROP;
```

RENAME TABLES

```
RENAME old_table_name TO new_table_name;
```

ALTER TABLES; ADD, DROP, RENAME, MODIFY COLUMNS

```
ALTER TABLE table_name MODIFY(
    column_name1  DATATYPE,
    column_name2  DATATYPE);

ALTER TABLE table_name ADD(
    column_name1  DATATYPE,
    column_name2  DATATYPE);
```

```
ALTER TABLE table_name
    DROP COLUMN column_name;

ALTER TABLE table_name RENAME COLUMN
    old_column_name  TO new_column_name;
```

ADD A DEFAULT

```
    ALTER TABLE table_name
    MODIFY  (column_name1 DEFAULT 'textvalue');
```

CONSTRAINTS

ADD A PRIMARY KEY

```
    ALTER TABLE table_name
    ADD CONSTRAINT pk_constraint_name
    PRIMARY KEY (column_name1, column_name2);
```

ADD A FOREIGN KEY

```
    ALTER TABLE table_name
    ADD CONSTRAINT fk_constraint_name
    FOREIGN KEY (column_name1, column_name2)
    REFERENCES parent_table_name(column_name1, column_name2);
```

ADD A UNIQUE CONSTRAINT

```
    ALTER TABLE table_name
    ADD CONSTRAINT uk_constraint_name
    UNIQUE (column_name1, column_name2);
```

ADD A CHECK CONSTRAINT

```
    ALTER TABLE table_name
    ADD CONSTRAINT ck_constraint_name
    CHECK (column_name1= UPPER(column_name1));

    ALTER TABLE table_name
    ADD CONSTRAINT ck_constraint_name
    CHECK (column_name1 IN ('option_1','option_2'));

    ALTER TABLE table_name
    ADD CONSTRAINT ck_constraint_name_nn
    CHECK (column_name1 IS NOT NULL);
```

DROP CONSTRAINTS

```
    ALTER TABLE table_name
    DROP CONSTRAINT constraint_name;
```

SEQUENCES

CREATE A SEQUENCE

CREATE SEQUENCE *sequence_name*
INCREMENT BY *interval*
START WITH *numbervalue*;

INCLUDE CLAUSES IN A SEQUENCE

CREATE SEQUENCE *sequence_name*
INCREMENT BY *interval*
START WITH *numbervalue*
MINVALUE *min_value* | NOMINVALUE
MAXVALUE *max_value* | NOMAXVALUE
CYCLE | NOCYCLE
CACHE *numbervalue*;

ALTER A SEQUENCE

ALTER SEQUENCE *sequence_name*
INCREMENT BY *interval*
MAXVALUE *numbervalue*;

DROP A SEQUENCE

DROP SEQUENCE *sequence_name;*

INSERT UPDATE AND DELETE DATA

INSERT DATA INTO ALL COLUMNS

INSERT INTO *table_name*
VALUES (*numbervalue,…., 'text value'***);**

INSERT DATA INTO SPECIFIC COLUMNS

INSERT INTO *table_name (column1,….,column2)*
VALUES (*numbervalue*,….,'*textvalue'***);**

INSERT INTO A TABLE USING A SEQUENCE

INSERT INTO *table_name*
VALUES (*seq_name*.NEXTVAL,….,'*text value*'**);**

INSERT INTO A TABLE USING A TEXT PREFIX + SEQUENCE

INSERT INTO *table_name*
VALUES ('*text_prefix*' **||***seq_name*.NEXTVAL,….,'*text value*'**);**

UPDATE DATA IN A COLUMN

UPDATE *table_name*
SET *column_name = expression*

```
WHERE condition;
```

DELETE A ROW FROM A TABLE

```
DELETE
FROM table_name
WHERE condition;
```

DELETE ALL ROWS FROM A TABLE (CAN NOT BE ROLLED BACK)

```
TRUNCATE table_name;
```

QUERYING THE DATABASE

QUERY THE DATABASE

```
SELECT column_name
FROM table_name
WHERE condition;
```

QUERY DATA USING & (PARAMETER)

```
SELECT column_name1
FROM table_name
WHERE column_name1 =’&variable_name’;
```

POSSIBLE CONDITIONS FOR NUMBER DATATYPES

```
= 40,  <>, 40,  < 40,  > 40,  IN (40,50,60),  != 40
NOT= 40,  BETWEEN 10 and 30,  NOT BETWEEN 10 and 30
```

POSSIBLE CONDITIONS FOR NUMBER DATATYPES

```
BETWEEN 'E%' AND 'T%',  <'F%',  LIKE '%K%'
LIKE '_OS%',  LIKE '_O%E%',  NOT LIKE '_O%'
```

EXAMPLE OF CONVERTING TO CHAR

```
SELECT Student_fname, student_lname, to_char(DOB, 'DAY') birthday
FROM students;
```

USE FUNCTIONS WHEN QUERYING

```
SELECT FUNCTION_NAME column_name1, FUNCTION_NAME
FROM table_name
GROUP BY column_name
HAVING FUNCTION_NAME condition
```

VIEWS

CREATE A VIEW

```
CREATE OR REPLACE VIEW view_name
 (column_name1, column_name3, column_name3)
AS SELECT
alias.column_name1,  alias.column_name2,  alias.column_name3
FROM table_name  alias
WHERE column_name = condition
WITH CHECK OPTION CONSTRAINT view_constraint_name
/
```

INSERT INTO A VIEW FROM AN OBJECT TABLE

```
INSERT INTO view_name
SELECT  (number_value1 , 'text_value2'), REF(object_table_name_alias)
FROM object_table_name  object_table_name_alias
```

OBJECTS

CREATE AN OBJECT TYPE

```
CREATE OR REPLACE TYPE Object_type_name AS OBJECT
(column_name1 DATATYPE,
column_name2 DATATYPE);
/
```

CREATE AN OBJECT TABLE

```
CREATE TABLE object_table_name OF object_type_name
(column_name DEFAULT 'textvalue');
```

INSERT INTO AN OBJECT TABLE

```
INSERT INTO object_table_name
VALUES  ('value1','value2');
```

APPLY A REFERENCE TO AN OBJECT TYPE IN A TABLE

```
REF object_type_name SCOPE IS object_table_name;
```

INSERT INTO A TABLE WITH AN OBJECT REFERENCE

```
INSERT INTO table_name (column_name1, column_name2, object_column_name)
SELECT number_value1, 'text_value2' , REF(object_table_alias)
FROM object_table_name   object_table_name_alias
WHERE condition;
```

VIEW A TABLE WITH ITS OBJECTS

```
SELECT DEREF (object_column_name2), column_name1
FROM table_name  alias
WHERE alias.object_column_name.object_column_attribute = ‘value’;

SELECT column_id, alias.object_name.column_name
FROM table_name  alias;
```

INCLUDE AN OBJECT COLUMN IN A STANDARD TABLE

```
CREATE TABLE table_name(
Column_name1 DATATYPE,
Column_name2 DATATYPE,
Column_name2 object_type);
```

INSERT INTO AN OBJECT COLUMN

```
INSERT INTO table_name (column_name1, column_name2, object_column_name)
VALUES (numeric_value1, ‘text_value2’, object_type_name('value1','value2'));
```

CREATE AN OBJECT TYPE FOR A VARRAY

```
CREATE OR REPLACE TYPE Object_type_name AS OBJECT
(column_name1 DATATYPE,
column_name2 DATATYPE);
/
```

CREATE THE VARRAY BASED ON THE OBJECT TYPE

```
CREATE TYPE varray_type_name AS VARRAY(50) OF Object_type_name;
```

INCLUDE A COLUMN IN A TABLE TO STORE THE VARRAY

```
CREATE TABLE table_name(
column_name1 DATATYPE,
column_name2 DATATYPE,
object_column_name varray_type);
```

INSERT INTO A TABLE WITH A VARRAY

```
INSERT INTO table_name (column_name, object_column_name)
VALUES (numbervalue, varray_type_name (‘text_value1', ‘text_value1’));
```

UPDATE A TABLE WITH A VARRAY

```
UPDATE table_name SET object_column_name =  varray_type_name (
varray_type_name (numbervalue, text_value),
varray_type_name (numbervalue, text_value),
varray_type_name (numbervalue, text_value))
WHERE condition;
```

QUERY A TABLE WITH A COLUMN VARRAY

SELECT *object_column_name*
FROM *table_name*
WHERE *condition*;

QUERY A TABLE WITH A COLUMN VARRAY

SELECT *table_name_alias. column_name1*, *varray_alias.varray_column_name1, varray_alias.varray_column_name2*
FROM *table_name table_name_alias*,
TABLE(*table_name_alias.object_column_name*) *object_column_name _alias*
WHERE *condition*;

CREATE THE OBJECT TYPE FOR A NESTED TABLE

CREATE OR REPLACE TYPE *object_type_name* AS OBJECT
(*column_name1* DATATYPE,
column_name2 DATATYPE);
/

CREATE A NESTED TABLE OBJECT TYPE

CREATE TYPE *table_type_name* AS TABLE OF *object_type_name*;

CREATE A TABLE BASED ON THE NESTED TABLE

```
CREATE TABLE table_name(
column_name1                DATATYPE,
column_name2                DATATYPE,
object_column_name          table_type_name)
NESTED TABLE object_column_name STORE AS nested_table_name;
```

INSERT INTO NESTED TABLES

INSERT INTO *table_name*
(*column_name1*, *column_name2*, *object_column_name*)
VALUES (*numbervalue*, '*textvalue*',
table_type_name(
object_type_name('*textvalue* ',' *textvalue* ', *numbervalue*),
object_type_name ('*textvalue* ',' *textvalue* ', *numbervalue*)));

QUERY NESTED TABLES

```
SELECT object_column_name
FROM table_name
WHERE condition;

SELECT table_name_alias. object_column_name, object_column_name _alias.
nested_table_column_name
FROM table_name table_name_alias,
TABLE(table_name_alias. .object_column_name) object_column_name _alias
WHERE object_column_name_alias.nested_table_column_name = condition ;
```

UPDATE A TABLE USING DATA FROM AN OBJECT TABLE

```
UPDATE table_name SET column_name1 =
(SELECT REF(x) FROM object_table_name x
WHERE  x.column_name2 = 'value1')
WHERE column_name1 – 'value2';
```

DROP AN OBJECT TYPE

```
DROP TYPE object_type_name;
```

DROP AN OBJECT TABLE

```
DROP TABLE object_table_name;
```

CREATE AN OBJECT WITH A MEMBER FUNCTION

```
CREATE OR REPLACE TYPE type_name AS OBJECT
(column_name1 DATATYPE,
column_name2 DATATYPE,
MEMBER FUNCTION member_function_name RETURN DATATYPE);
/
```

CREATE THE MEMBER FUNCTION

```
CREATE OR REPLACE TYPE BODY object_type_name IS
MEMBER FUNCTION  member_function_name RETURN datatype IS
        BEGIN
        RETURN column_name_1 , column_name_2;
        END;
END;
/
```

SELECTING FROM MEMBER FUNCTIONS

```
SELECT alias.member_function_name()
FROM object_table_name alias;
```

ADD A COLUMN TO AN OBJECT TABLE,

```
ALTER TABLE object_table_name ADD
(new_coulumn_name REF object_type_name
SCOPE IS object_table_name);
```

DISPLAY ERRORS IN OBJECTS AND PL/SQL

SHOW ERRORS

RETRIEVE DATA FROM THE OBJECT TABLE

```
UPDATE table_name1  table_name_alias1
SET table_name_alias1.attribute =
        (SELECT REF (table_name_alias2)
        FROM  table_name2 table_name_alias2
        WHERE table_name_alias2.column_name = condition)

SELECT DISTINCT object_table alias.column_name
FROM object_table object_table_alias;
```

USE THE UNION FUNCTIONS IN A SELECT WITH OBJECTS

```
SELECT alias.column_name.attribute
FROM table_name  alias
WHERE alias.oject_table.attribute = ‘value’
UNION
SELECT alias.colimn_name.attribute
FROM table_name  alias
WHERE alias.oject_table.attribute = ‘value’
/
```

USEFUL SELECTS

```
SELECT * FROM table_name;

SELECT * FROM tab;

SELECT * FROM user_tables;

SELECT type_name, table_name FROM user_types;

SELECT * FROM user_objects;

SELECT constraint_name, table_name
FROM user_constraints WHERE constraint_name LIKE ‘P%’;

SELECT constraint _name, table_name
FROM user_constraints WHERE constraint_name NOT LIKE ‘SYS%’;
```

FORMATTING

```
SELECT column_name ||’, ‘|| column_name AS new_column_name
FROM table_name;

SELECT  column_name1 ||' ,'|| column_name2, column_name3, column_name4

COLUMN column_name1 HEADING 'Title1 |Title2'
COLUMN column_name2 HEADING 'Title1 |Title2'
COLUMN column_name3 HEADING 'Title1 / Title2'
COLUMN column_name1 FORMAT A3;
```

USE FUNCTIONS

```
SELECT FUNCTION column_name1, FUNCTION(*)
FROM table_name
GROUP BY column_name
HAVING FUNCTION condition;
```

USE OPERATORS

```
SELECT alias.object_column_name.attribute
FROM table_name  alias
WHERE alias.oject_table.attribute = ‘value’
UNION
SELECT alias.object_colimn_name.attribute
FROM table_name  alias
WHERE alias.oject_table.attribute = ‘value’
/
```

NOTE: In relational algebra the select statements must contain the same number of columns

```
SELECT alias.column_name.attribute
FROM table_name  alias
WHERE alias.oject_table.attribute = ‘value’
INTERSECT
SELECT alias.colimn_name.attribute
FROM table_name  alias
WHERE alias.oject_table.attribute = ‘value’
/
```

NOTE: INTERSECT returns values where both parts of the statement are true

```
SELECT  column_name1, column_name1
FROM object_table
MINUS
SELECT table_name_alias.column.attribute, table_name_alias.column.attribute
FROM table_name  alias;
```

```
SELECT column_name
FROM table_name
WHERE column_name  IN | EXISTS
(SELECT column_name
FROM table_name
WHERE condition);
```

```
SELECT table_name_alias1.column_name, table_name_alias2.column_name
FROM table_name1  alias1, table_name2  alias2
WHERE alias1.column_name  =  alias2.column_name;
```

```
SELECT alias1.column_name, alias2.column_name
FROM table_name1  alias1, table_name2  alias2
WHERE alias1.column_name  =  alias2.column_name (+);
```