Advance Database

Terminologies

① Relational Database
. collection of data items as a set of formally described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables.
The standard user and application program interface to relational db is the SQL.
Each table ⇒ relation
row ⇒ unique instance of data
column ⇒ specification of type of data

"A database structured to recognize relations between stored items of information."

Types of DB
- Text database / file database
(~~Desktop database programs~~
~~eg. MS Excel, Access~~)
- RDBMS
- NoSQL and object-oriented databases

BCA

Date
Page

2) SQL

A domain specific language used in programming and designed for managing data held in a RDBMS.
- lets you access and manipulate database
- ANSI (American National Standards Institute) standard
- communicates to the database.

functions
- execute queries against a database.
- retrieve / insert / delete / update records.
- create databases and tables.

To be compliant with the ANSI standard, they should support major commands such as SELECT, UPDATE, DELETE, INSERT, WHERE, CREATE

3) SQL *Plus

Interactive and batch query tool that is installed with every oracle database installation.
It has • command line UI, • Windows Graphical UI, • web-based UI
It provides access to the oracle database, enables to enter & execute SQL, PLISQL, SQL*Plus commands.

functions
- format, perform calculations on, store & print from query results
- examine table & object definitions
- develop and run batch scripts
- perform database administration.

5)

SQL*Plus also generates reports interactively, also as batch processes. And outputs the result to text file, screen or HTML file

example:

- To rename a column
  COLUMN ADDRESS HEADING 'FULL-ADDRESS'
- To list column definitions of a table
  DESCRIBE TABLE-NAME

4) PL/SQL (Procedural Language)

Combination of SQL along with the procedural features of programming language. Developed by oracle corp to enhance the capabilities of SQL.

PL/SQL is one of three key programming languages embedded in oracle databases along with SQL and JAVA.

* Procedural Language :- Derived from structured programming, based upon the concept of procedure call. Procedures also k/a routines, subroutines or functions, simply contains a series of computational steps to carry out. A procedure might be called at any time during a program's execution.

5) Database Language

i) Data-Manipulation Language (DML)

access or manipulate data by app. data model.

- retrieval
- insertion
- deletion
- modification

ii) Data-Definition Language (DDL)

specify additional properties to data, making of the db & table.

- database schema
- create database and tables
- modification of tables
- deleting a column

Data Types

* Built-in domain types
(Basic types)

type	full-form	Description
char(n)	character	fixed length character string
varchar(n)	character varying	variable length " "
int	integer	machine dependent length
smallint	small integer	" " subset of integer
numeric(p,d)	numerical	total p digits, d after decimal (3,1) ⇒ 44.5 but not 444.5 or 1.22.
real, double	~~float~~ -	floating-point and double-precision floating point numbers (Midpoint)
float(n)	floating-point	

others: long, dec(p,d), decimal(p,d)

(advanced built-in types)

date : YYYY, MM, DD

time HH, MM, SS (time with timezone)

timestamp date and time (timestamp with timezone)

extracting timezone : timezone-hour
timezone-minute

(create), (update)

eg. date '2018-02-20'
time '09:00:00'
timestamp '2018-02-20 09:00:00'

SQL current date time functions
current_date(), current_time(), current_timestamp (with timezone)
local timestamp (local date and time without time zone)

Large Object (LOB) datatypes

bfile	file locator that points to a binary file on the server file system (outside the database)
blob	binary large objects
clob	single-byte / multi-byte character data
nclob	unicode data

User-defined data types

```
create type Dollars as numeric(12,2) final
create type fullname as varchar(50) final

create table account(
        acc-id int,
        emp-name fullname,
        salary Dollars)
```

Date
Page

Basic SQL Statements

① CREATE

- Database

CREATE DATABASE db-name;

- Table

CREATE TABLE table-name
(
col1 datatype [NULL | NOT NULL],
col2 datatype [NULL | NOT NULL],
....
coln datatype [NULL | NOT NULL],
);

② INSERT

INSERT INTO table-name
(col1, col2, ..., coln)
VALUES
(val1, val2, ..., valn);

③ DELETE

DELETE FROM table [WHERE conditions];

④ ALTER

Add, modify or drop/delete columns in a table

```
ALTER TABLE table-name
    ADD col-name col-definition;

ALTER TABLE table-name
    ADD ( col-1 col-definition,
          col-2 col-definition,
          .... );
```

Add new column

```
ALTER TABLE table-name
    MODIFY col-name col-type
eg.
    ....
    MODIFY cust-name varchar2(100)
    not null;

    MODIFY ( col-1 col-type,
             col-2 col-type,
             ..... );
```

Modify column da

```
ALTER TABLE table-name
    DROP COLUMN col-name;
```

delete a column

```
ALTER TABLE table-name
    RENAME COLUMN old-name to new-name;
```

rename column

```
ALTER TABLE table-name
    RENAME TO new-table-name;
```

rename a table

⑤ DROP

DROP TABLE [schema-name]. table-name
[CASCADE CONSTRAINTS] ;

CASCADE CONSTRAINTS :- all referential integrity constraints will be dropped as well

⑥ DESCRIBE

DESCRIBE [schema-name]. table-name ;

- Column names
- NULL or NOT NULL
- data types
- precision of numeric column, if any

Note :- ; → sql terminator
can be changed as
set sqlterminator #

(;)

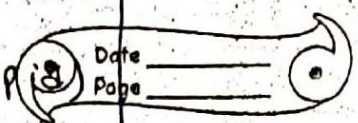
terminate the current statement and store into SQL PLUS Buffer but do not run it.

(/)

- run the statements in the buffer.
- shortcut for RUN command

eg. drop table foo ;
/

```
id int PRIMARY KEY,
name varchar,
pid int, FOREIGN KEY(pid) REFERENCES Person(P
);
```



Data Constraints

Constraints are the rules enforced on data columns on table. It limits the type of data that can go into a table. It ensures data accuracy and reliability.

① NULL

ensures whether a column can have null value or not.

② DEFAULT.

provides a default value for a column when none is specified.

③ UNIQUE

ensures that all values in a column are different.

④. PRIMARY KEY

uniquely identified each row/record in database table.

⑤ FOREIGN KEY

uniquely identified a row/record in another table.

⑥ CHECK

ensures that all values in a column satisfy certain conditions.

eg. AGE INT NOTNULL CHECK (AGE >= 18),
.....

⑦ Composite key.

PRIMARY KEY(id, code)

CREATE TABLE t(
col1 datatype not null,
....
CONSTRAINT cnst_name PRIMARY KEY(col1));

Dropping constraints

ALTER TABLE table-name
DROP CONSTRAINT column-name;

ALTER TABLE table-name
DROP CONSTRAINT PRIMARY KEY;

ALTER TABLE table-customer
ADD CONSTRAINT myconstraint UNIQUE(AGE, SALARY);
ALTER TABLE table-customer
DROP CONSTRAINT myconstraint;

Integrity Constraints
Ensures accuracy and consistency of data in Relational Database. It is handled through referential integrity.

It includes primary key, foreign key. It states that table relationships must always be consistent.
Any foreign key must agree with the primary key that is referenced by the foreign key i.e. any primary key field changes must be applied to all foreign keys.

* Create table test(
id int, sid int,
primary key(id),
foreign key(sid) references student));

Use of SELECT Statement

Restricting rows

- SELECT [PERCENT] TOP 3 * FROM customers;
- SELECT * FROM customers LIMIT 4;
- SELECT * FROM customers WHERE ROWNUM <= 3;
- SELECT DISTINCT col1, col2 ... FROM table-name;
- To select bottom 'N' rows: Select * From (SELECT * FROM table-name order by id DESC) where rownum <= 3;

Sorting data

SELECT * FROM table-name [WHERE condition] [ORDER BY col1, col2, ...] [ASC | DESC];

=> First the rows are ordered by col1 and then inside col1 ordered by col2.

Date
Page

Computations on Table Data

Arithmetic Operators

+, −, *, /, %

eg. ~~SELECT (10+20) AS ADDITION~~ SELECT (col1+col2) as Summ from tbl-name;

=> ~~ADDITION~~

~~30~~

Logical Operators

- ALL | ANY | SOME

=> SELECT col-name FROM table-name
WHERE expression1 comparison-opr {ALL | ANY | SOME} (subquery)

- AND | OR

=> WHERE col1 = expression1 AND col2 = expression2;

- BETWEEN

=> WHERE col BETWEEN ~~to~~ value1 AND value2;

- EXISTS

=> WHERE EXISTS (subquery);

- NOT

=> NOT EXISTS, NOT BETWEEN

Pattern matching

using LIKE

wild cards : % :- represents zero, one or multiple characters
_ :- a single character (? in MS Access)

```
SELECT col1, col2, ..... FROM table-name
WHERE colN LIKE pattern;
```

a% :- value starts with a.
%a :- ends with a
%a% :- a in any position
_a% :- a in second position
a%b :- starts with a and ends with b.

Range searching

BETWEEN
<, >, <=, >=

Unit-2

Grouping data

Arranges identical data into groups. The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

```
SELECT col1, col2 FROM table-name
WHERE [conditions]
GROUP BY col1, col2
[ORDER BY col1, col2] optional
```

Used with aggregate functions like avg, min, max, count, sum.

eg. SELECT NAME, SUM(SALARY) FROM CUSTOMERS GROUP BY NAME;

ID	NAME	SALARY
1	Ramesh	1000
2	Ramesh	2000
3	Suresh	3000
4	Suresh	4000

⇒

NAME	SUM(SALARY)
Ramesh	3000
Suresh	7000

Sub-queries (inner query / nested query)

It is a query within another query and embedded within the WHERE clause. A subquery returns data that will be used in the main query as a condition to further restrict the data to be retrieved.

It can be used with SELECT, INSERT, UPDATE and DELETE and along with the operators like =, <, >, IN, BETWEEN

Rules for subqueries

- Subqueries must be enclosed within parentheses. [( )]
- It can have only one column in its SELECT clause.
- ORDER BY can't be used in a subquery but the main query can. GROUP BY can be used in subquery.
- Subqueries that return multiple rows can only be used with multiple value operators such as IN operator
- Don't use BLOB, CLOB, NCLOB, ARRAY in SELECT list.
- BETWEEN operator can't be used with subquery but can be used within subquery

```
SELECT col1, col2 FROM table1, [table2]
WHERE column-name OPERATOR
( SELECT col1, [col2] FROM table1, [table2] [WHERE] );
```

ID	NAME	AGE
1	A	10
2	B	12
3	C	14
4	D	16

⇒

ID	NAME	AGE
1	A	10
2	B	12

```
SELECT * FROM table WHERE ID IN
(SELECT ID FROM table WHERE AGE < 13);
```

For INSERT

```
INSERT INTO table_name [(col1, [col2])]
    SELECT [*, col1, col2] FROM table1, [table2]
        [WHERE condition]
```

eg.

```
INSERT INTO cust SELECT * FROM student
    WHERE ID IN (SELECT ID FROM cust);
```

For UPDATE

```
UPDATE table SET column_name = new_value
    [WHERE OPERATOR [VALUE]
        (SELECT col_name FROM table_name [WHERE])
```

* For multiple columns

```
Select * from Student where (id, phone) in
(select id, phone from student);
```

Date
Page

Having clause

It enables you to specify conditions that filter which group results appear in the final results.

SELECT FROM WHERE
GROUP BY HAVING ORDER BY } sequence

SELECT col1, col2 FROM tbl1, tbl2 WHERE [conditions]
GROUP BY col1 HAVING [conditions] ORDER BY col1 ;

eg.

ID	NAME	AGE
1	A	10
2	B	12
3	C	14
4	D	14
5	E	15

⇒

ID	NAME	AGE
3	C	14

SELECT * FROM table GROUP BY AGE HAVING COUNT(age) >= 2;

The having clause places conditions on groups created by the GROUP BY clause.

suppliers

sup-id	sup-name
1	IBM
2	HP
3	MS
4	NVIDIA

order-id	sup-id	order-date
901	1	2003/10/12
902	2	2004/05/01

Date
Page

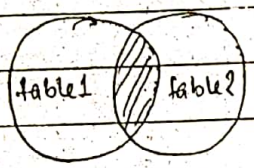
Joining multiple tables

- INNER JOIN (simple join)
- LEFT OUTER JOIN (LEFT JOIN)
- RIGHT OUTER JOIN (RIGHT JOIN)
- FULL OUTER JOIN (FULL JOIN)

1) INNER JOIN

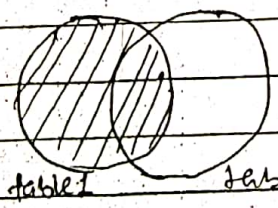
It is the most common type of join which returns all rows from multiple tables where the join condition is met.

```
SELECT columns FROM table1 INNER JOIN table2
ON table1.column = table2.column;
```



2) LEFT OUTER JOIN

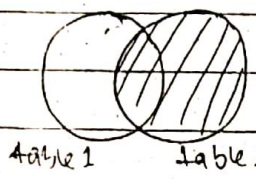
It returns all rows from the left hand table specified in the ON condition and the intersected rows from right hand table.



SELECT columns FROM table1 LEFT [OUTER] JOIN table2
ON table1.column = table2.column;

RIGHT OUTER JOIN

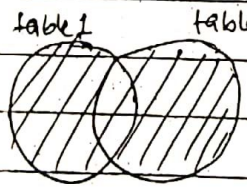
It returns all rows from the RIGHT hand table specified in the ON condition and only the intersected parts from LEFT hand table.



SELECT columns FROM table1 RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;

FULL OUTER JOIN

It returns all rows from the left hand table and right hand table.



SELECT columns FROM table1 FULL [OUTER] JOIN table2
ON table1.column = table2.column;

Joining three tables

SELECT columns FROM table1 INNER JOIN table2 ON table1.column = table2.column INNER JOIN table3 ON table2.column = table3.column;

UNION Operator

It is used to combine the result sets of two or more SELECT statements. It removes the duplicate rows between the various SELECT statements.

- Each SELECT statement within UNION must have the same number of columns.
- The columns must also have similar data types.
- The columns must also be in same order.

SELECT columns FROM table1 [WHERE conditions] UNION
SELECT columns FROM table2 [WHERE conditions];

eg. SELECT city FROM customers UNION
SELECT city FROM suppliers;

Customers

id	name	city
1	Alfred	NY
2	Jack	Berlin
3	Tom	TX

Suppliers

id	sup-name	city
1	Ben	TX
2	Schultz	NY
3	Sam	LV

⇒

city
NY
Berlin
TX
LV

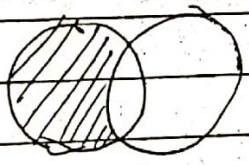
INTERSECT operater

It is used to return the results of 2 or more SELECT statements where only the common results are shown.

```
SELECT columns FROM table1 INTERSECT SELECT columns
FROM table2 [WHERE condition];
```

MINUS operater

It returns all rows in the first SELECT statement that are not returned by the second select statement.

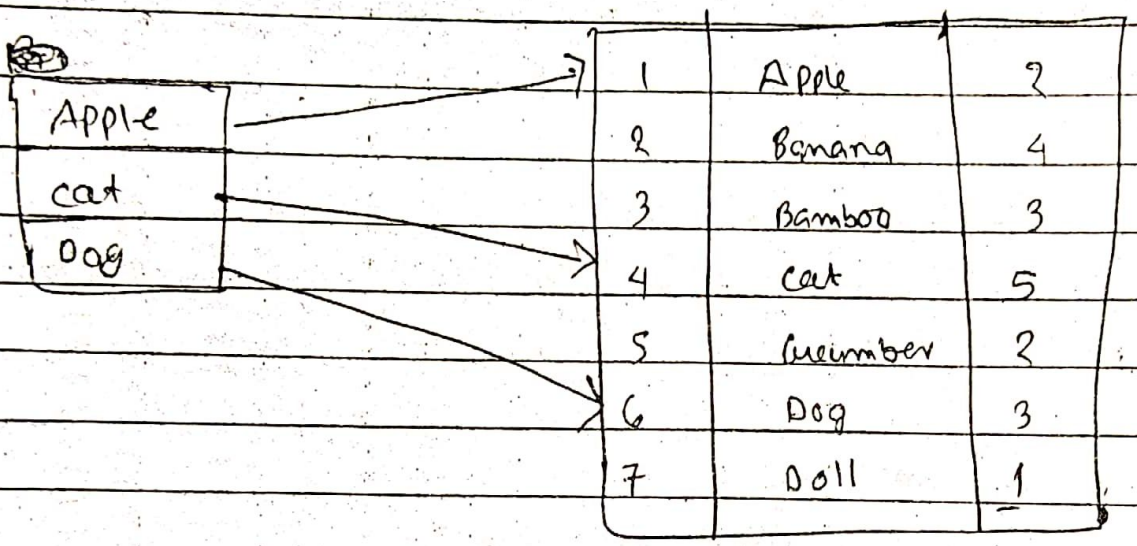


SELECT columns FROM table1 [WHERE conditions]
MINUS SELECT columns FROM table2 [WHERE conditions];

SQL Performance Tuning

Indexes

Allows faster retrieval of records. It creates search keys that is used to access records quickly.



Syntax:

CREATE [UNIQUE] INDEX index_name
ON table_name (col1, col2, ... coln);

➢ The users cannot see the indexes, they are just used to speed up searches/queries. It is implemented internally.

DROP INDEX

DROP INDEX index-name; ~~ON table-name;~~

RENAME INDEX

ALTER INDEX index-name
RENAME TO new-index-name;

Views

It is a virtual table that physically doesn't exist. Rather it is created by a query joining multiple or single tables.

CREATE VIEW view-name AS
SELECT columns FROM tables [WHERE conditions];

and then

SELECT * FROM view-name;

Update view

CREATE OR REPLACE VIEW view-name AS
SELECT columns FROM table [WHERE conditions];

Drop view

DROP VIEW view-name;

- The view can be updated ~~automatically after updating the~~ which also reflected in its appropriate tables.
- View exists even if the table is deleted but it shows error on querying.

Sequences

It is an object that is used to generate a number sequence. It can be useful when you need to create a unique number to act as a primary key.

```
CREATE SEQUENCE sequence-name
    MINVALUE value
    MAXVALUE value
    START WITH value
    INCREMENT BY value ;    (- value for decrementing)
    CACHE value ;  [NOCACHE]
```

eg.
```
CREATE SEQUENCE cust_id
    MINVALUE 1
    MAXVALUE 999999999999
    START WITH 1
    INCREMENT BY 1
    CACHE 20;
```

=> CACHE :- total no. of sequences to be generated and stored in memory for faster access.

Retrieve sequence

```
sequence_name.NEXTVAL;
```

DROP Sequence

```
DROP SEQUENCE sequence-name;
```

Users in Oracle database

Oracle database requires users to perform tasks on the database. Different users have different permissions.

Permissions and privileges includes
- create database
- execute
- DROP
- insert

SELECT * FROM DBA_SYS_PRIVS;

GRANTEE :- name, role, user that was assigned the privilege.
PRIVILEGE :- assigned privileges
. ADMIN_OPTION :- specifies if the granted privilege also includes ADMIN option.

** CONTD...

Security Management in SQL

Oracle provides extensive security features to safeguard information stored in tables from unauthorized viewing and damage.
Depending upon users privilege/permission, the oracle resources can be assigned by the DBA.

The owner of the object (data) can give permission or access to a user upon request. This is called Granting of Privileges.

Privileges once given can be taken back by the owner of the object. This is called revoking of privileges.

* objects :- tables, views, sequences ...

Granting Privileges
using GRANT

GRANT <object privileges> ON <objectname>
To <username> [WITH GRANT OPTION];

object privileges
ALTER, DELETE, INDEX, INSERT, SELECT, UPDATE

WITH GRANT OPTION :- The grantee can grant object privileges to other users.

- GRANT ALL ON Product-table To John;

- GRANT SELECT, UPDATE ON Client-table To Jackie WITH GRANT OPTION;

* Granting permission by Grantee (with grant option)
GRANT SELECT ON owner-name.table-name To username;

Revoking privileges

REVOKE <object privileges> ON <objectname>
from <username>;

eg. REVOKE DELETE ON Supplier FROM harry;
(ALL)

Oracle users (*** CONTD...)

user accounts on oracle to access database and gran
privileges. It includes:

- Authentication method
- Password
- default tablespace for permanent/ temporary data storage
- account status (locked/unlocked)
- password status (expired or not)
- Current the system privileges

Administrative user accounts

SYS and SYSTEM

To view all the users: SELECT username FROM DBA_USERS;
username: sys as sysdba || ALTER user sys identified
password: *** by "anil";

Unit-3 PL/SQL

PL/SQL is a combination of SQL along with procedural features of programming language.
Programming language with extension of SQL.

Advantages

- Portable, high performance transaction processing language.
- Supports both Static SQL (DML operation and transaction control) and dynamic SQL (embedding DDL statements in PL/SQL blocks)
- Allows sending an entire block of statements to the database at one time.
- High productivity to programmers.
- Saves times on design & debugging by exception handling, encapsulation, data hiding. (Support for Obj-oriented programming)
- High Security

Date
Page

PL/SQL features

Block structure

1) Declarations

It defines all variables, cursors, subprograms to be used in the program. It is optional section.
keyword: DECLARE

2) Executable command

Mandatory section, consists of executable PL/SQL statements. It should have at least one executable which may be just a NULL command.

keyword: BEGIN and END;

3) Exception handling

It contains exceptions that handle errors in the program. It is optional.

Eg.
```
DECLARE
    message varchar2(20) := 'Hello, world!';
BEGIN
    dbms_output.put_line(message);
END;
/
```

PL/SQL Execution Environment

- After entering the SQL plus environment, you can use PL/SQL in several ways:
  - Input, store and run a PL/SQL block
  - Create, load & run a script containing PL/SQL blocks / subprograms.

1) Open Sqlplus.
2) Enter credentials.
3) Write PL SQL program into *.sql file.
4) Enter following:

```
set serveroutput on size 30000;
start filename.sql;
```

sets buffer for dbms_output.

PL/SQL Datatypes

PL/SQL has many data types and sub types to determine storage format, constraints, range of values. A subtype is a subset of another data type which is called its base type.

A subtype can have operations as its base types but only a subset of its valid values.

Scalar datatype

Single values with no internal components.

- Numeric (PLS-INTEGER (BINARY-INTEGER) 32b, BINARY-FLOAT 32, SIMPLE-INTEGER, BINARY-DOUBLE 64, NUMBER)
- character
- boolean
- datetime : YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, time
- interval (time intervals)

Numeric : DECIMAL, NUMERIC, FLOAT, INT, INTEGER, SMALLINT, REAL, double precision,

character : char, varchar2

BOOLEAN : TRUE, FALSE, NULL

datetime : eg. num date := '12-JAN-17';
(DD-MON-YY).
time = (HH:MM:SS.MS)
log_ft TIMESTAMP(3) WITH TIME ZONE;
↳ no. of digits in the frac
parts of seconds field.

INTERVAL : log_time INTERVAL DAY(3) TO SECOND(3);

LOB : BLOB, CLOB, NCLOB
BFILE :- used to store large binary objects outs
the database.

timestamp :- current_timestamp

Date
Page

User defined Sub-types

SUBTYPE CHARACTER IS CHAR;
SUBTYPE INTEGER IS NUMBER(10,0);

eg. DECLARE
IS
SUBTYPE bdate DATE NOT NULL;
~~SUBTYPE counter I~~
BEGIN
NULL;
END;
/

Data Type Conversion

TO_DATE, TO_NUMBER :- To convert CHAR to DATE/NUMBER
TO_CHAR

eg. DECLARE
start_time CHAR(5);
finish_time CHAR(5); elapsed_time NUMBER(5);
BEGIN
-- Get system time
SELECT TO_CHAR(SYSDATE, 'SSSSS') INTO start_time
FROM sys.DUAL;
SELECT TO_CHAR(SYSDATE, 'SSSSS') INTO finish_time
FROM sys.DUAL;
elapsed_time := finished_time - start_time;
Dbms_output.put_line('Elapsed Time: ' || TO_CHAR(elapsed_time));
END; /

Date
Page

PL/SQL Variables

variable_name datatype [NOT NULL := value];

eg. num number(6);
department varchar2(10) NOT NULL := "HR Dept";

1) Local variables = declared in an inner block and can't be referenced by outside blocks.
2) Global variables = declared in outer block and can be referenced by itself & inner blocks

Literal/Constant

PI CONSTANT NUMBER := 3.1415;

- Numeric literals (6.6, 5078)
- character literals ('A', '$', '10')
- String literals ('Hello')
- Boolean literals (TRUE, FALSE, NULL)
- Date & time literals

Date
Page

Condition control in PL/SQL

(1) IF ELSE condition.

1) IF condition THEN
....
END IF;

2) IF condition THEN
....
ELSE
....
END IF;

3) Nested IF.

IF condition THEN
....
ELSE
IF condition THEN
....
ELSE
....
END IF;
END IF;

4) IF condition THEN
....
ELSEIF condition THEN
....
ELSE
....
END IF;

(2) CASE

CASE grade
WHEN 'A' THEN .... ;
WHEN 'B' THEN .... ;
....
ELSE .... ;
END CASE;

num := 10;
CASE num
when 1 then num := 0;
when 10 then num := 100;
ELSE num := 1;
END CASE;
dbms_output.put_line (num);

Iterative control in PL/SQL

- **For loop**

```
FOR counter IN val1 .. val2 LOOP
    LOOP statements;
END LOOP;
```

=> val 1 - start integer value
val 2 - end integer value

eg.
```
DECLARE
    a number(2);
BEGIN
    FOR a in 10 .. 20 LOOP
        dbms_output.put_line ('value of a: ' || a);
    END LOOP;
END;
/
```

output :- 10, 11, 12 ... 20

* **Reverse for loop**

```
DECLARE
    a number(2);
BEGIN
    FOR a IN REVERSE 10 .. 20 LOOP
        dbms_output.put_line ( a );
    END LOOP;
END;
/
```

output : 20, 19, ... 10

- While LOOP

```
WHILE condition LOOP
        statements
END LOOP;
```

eg.
```
DECLARE
    a number(2) := 10;
BEGIN
    WHILE a < 20 LOOP
        dbms_output.put_line ('value of a:' || a);
        a := a+1;
    END LOOP;
END;
/
```

Label

```
<< label_name >>
```

Exit

```
EXIT;
```

GOTO

```
GOTO label;
<< label >>
    statements;
```

COMMENT

```
-- comments
```

Continue

```
CONTINUE;
```

eg.

```
DECLARE :
   a number (2) := 10;
BEGIN
   -- while loop execution
   WHILE a < 20 LOOP
      dbms_output.put_line ('value of a :' ||a);
      a := a+1;
      IF a > 15 THEN
         EXIT;
      END IF;
      IF a = 18 THEN
         GOTO end_execution; END IF;
      END LOOP;
      dbms_output.put_line ('Final statement');
      << end_execution >>
END;
/
```

Unit-4

oracle transactions

A transaction is a logical piece of work consisting of one or more SQL statements. A transaction is started whenever data is read or written and they are ended by a COMMIT or ROLLBACK.

Transaction Properties

Atomicity :- A transaction either happens completely or doesn't happen.

Consistency :- The data of transaction is consistent.

Isolation :- The effects of a transaction is not visible to other transaction until committed.

Durability :- once the transaction is committed, it is permanent.

COMMIT

This statement commits all changes for the current transaction. Once a commit is issued, other users will be able to see your changes.

COMMIT [comment clause];

eg. COMMIT COMMENT 'first commit';

* Comments are stored along with transaction id in DBA-2PC-PEND system view, if there is a problem.

ROLLBACK

It is used to undo the work performed by the current transaction.

ROLLBACK ~~COMMIT~~ [TO [SAVEPOINT] savepoint_name];

eg. ROLLBACK ~~COMMIT~~ TO SAVEPOINT savepoint1;

SAVEPOINT

It marks the current point in the processing of a transaction. It undoes parts of a transaction instead of whole transaction.

SAVEPOINT save-point name;

```
CREATE TABLE emp-name (
    id int,
    last-name varchar(20),
    salary number(5) );

DECLARE
    emp-id    emp-name.id %type;
    emp-lastname   emp-name.last_name %type;
    emp-salary    emp-name.salary %TYPE;
BEGIN
    SELECT ~~employee~~-id, last-name, salary INTO emp-id,
    emp-lastname, emp-salary FROM emp-name WHERE
    ~~employee~~ emp-id = 12;
    UPDATE emp-name SET salary = salary * 1.1 WHERE
    id = emp-id;
    DELETE FROM emp-name WHERE emp-id = 18;
    SAVEPOINT do-insert;
    INSERT INTO emp-name VALUES (emp-id, emp-lastname,
    emp-salary);
EXCEPTION
    WHEN DUP-VAL-ON-INDEX THEN
        ROLLBACK TO do-insert;
        dbms-output.put-line ('Rolled back');
    END;
    /
```

Concurrency Control

It is concerned with simultaneous access of the same data by many users. A multi-user database management system must provide adequate concurrency controls, so that data can't be updated or changed improperly.

Concurrency Anomalies:

- Data overlay: one user is updating a data and another user is viewing the same data. The another user might overlay the changes made by first user.
- Incorrect update decisions: multiple users might be updating the data simultaneously. If a change is applied which is further changed by another user, the previous user may have to re-apply their updates.

Implicit locking

Oracle locking mechanism is fully automatic and requires no user action. Implicit locking occurs for all SQL statements so that database users never need to lock any resource explicitly.

Oracle's default locking mechanisms lock data at the lowest level of restrictiveness to guarantee data integrity while allowing the highest degree of data concurrency. For eg. an operation that reads an object will obtain a read lock, an operation that modifies an object will obtain a write lock.

Explicit locking

Some applications may need to reserve access to all required resources in advance. Reasons for this might be to secure required access rights to the necessary objects before beginning an operation or to prevent other sessions from modifying objects critical to operation.

Such applications needing to reserve access to all required objects in advance can explicitly lock objects. In such case, the lock is explicitly requested and applied.

Levels of locking

1) Statement-level

It is enforced implicitly. It guarantees that all data returned by a single query comes from a single point in time. Therefore, a query never sees dirty data or any of the changes made by transaction.

As query execution proceeds, only data committed before the query began is visible to the query.

eg. SELECT, INSERT with a subquery UPDATE, DELETE return consistent data implicitly.

2) Transaction - Level

It guarantees all data returned by a transaction comes as a single point. It means a transaction happens completely or not at all. Except queries made by a serializable transaction do see some changes made by transaction itself.

Transaction isolation levels:

- Read committed :- (Default level) each query executed by a transaction only data that was committed before the query, not the transaction began.
- Serializable :- serializable transactions see only those changes that were committed at the time the transaction began, plus changes made by transaction itself through INSERT, UPDATE and DELETE statements.
- Read-only :- sees only those changes that were committed at the time the transaction began and do not allow INSERT, UPDATE and DELETE statements.

eg. SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION READ ONLY;

Cursors in PL/SQL

Oracle creates a memory area, known as the context area, for processing an SQL statement, which contains all the information needed for processing the statement, for eg. the number of rows processed etc.

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

A cursor can be renamed so that it could be refered to in a program to fetch and process the rows returned by the SQL statement, one at a time.

Implicit Cursor

It is automatically created by oracle whenever an SQL statement is executed. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE, DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE & DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, the most recent implicit cursor is referred as the 'SQL cursor' which has attributes as follows:

%FOUND :- Returns TRUE if an INSERT, UPDATE, DELETE statement affected one or more rows or SELECT statement returned one or more rows. Otherwise FALSE.

%NOTFOUND :- The logical opposite of %FOUND.

%ISOPEN :- Always returns FALSE for implicit cursors because oracle automatically closes, after executing the SQL statement.

%ROWCOUNT :- Returns the number of rows affected or selected.

SYNTAX:- SQL%ATTRIBUTE_NAME

eg.
```
DECLARE
    rows number(2);
BEGIN
    UPDATE customers SET salary = salary + 500;
    IF sql%notfound THEN
        dbms_output.put_line('no customers selected.');
    ELSEIF sql%found THEN
        rows := sql%rowcount;
    dbms_output.put_line(rows||' customers selected');
    END IF;
END;
/
```

Explicit Cursor

It is Programmer-defined cursors for gaining more control over the context area. It should be defined in the declaration section of PL/SQL block. It is created on SELECT statement which returns more than one row.

```
CURSOR cursor-name IS select-statement;
```

Steps to work with explicit cursor

1) Declaring the cursor for initializing the memory.
2) Opening the cursor for allocating the memory.
3) Fetching the cursor for retrieving the data.
4) closing the cursor to release the allocated memory.

1) Declaring the cursor

```
CURSOR c_customers IS
    SELECT id, name, address FROM customer;
```

2) opening the cursor

```
OPEN c_customers;
```

3) Fetching the cursor

```
FETCH c_customers INTO c_id, c_name, c_add;
```

4) Closing the cursor

```
CLOSE c_customers;
```

eg.

```
DECLARE
    c_id customers.id %type;
    c_name customers.name %type;
    c_add customers.address %type;
    CURSOR c_customer IS SELECT id, name, address
                FROM customers;
BEGIN
    OPEN c_customers;
    LOOP
    FETCH c_customers into c_id, c_name, c_add;
        EXIT WHEN c_customers %notfound;
        dbms_output.put_line (c_id || c_name || c_add)
    END LOOP;
    CLOSE c_customers;
END;
/
```

Parameterized Cursors

```
DECLARE
    my_record emp %ROWTYPE;
    CURSOR curs (wage NUMBER) IS
        SELECT * FROM emp WHERE sal < wage;
BEGIN
    OPEN curs (10000);
    LOOP
      FETCH curs INTO my_record;
```

```
    EXIT WHEN curs%NOTFOUND;
    dbms_output.put_line('Name:' || my_record.name ||
                         'Salary:' || my_record.sal);
  END LOOP;
  CLOSE curs;
END;
/
```

Unit-5 Using PL/SQL Subprograms

Subprograms are the building blocks of modular, maintainable applications. It contains set of statements into reusable subprograms.

A PL/SQL subprogram is a named PL/SQL block that can be invoked with a set of parameters. A subprogram can be either a procedure or a function. Typically, a procedure is used to perform an action and a function to compute and return a value.

Advantages of PL/SQL subprograms:
- It promotes re-usability.
- It breaks a program into manageable, well-defined modules.
- It lets db programmer to extend the PL/SQL language.
- It can be grouped into PL/SQL package.

Procedures

It is a subprogram that does not return a value directly but used to perform an action.

Functions

It is a subprogram that returns a value. It is mainly used to compute and return a value.

Parts of a PL/SQL subprogram

1) Declarative Part
   optional, contains declarations of types, cursors, variables, constants, exceptions and nested subprograms. It is local to the subprogram only.

2) Executable Part
   Mandatory part, contains statements to be executed.

3) Exception handling
   optional part, contains the code that handles run-time errors.

Creating a procedure

```
CREATE [OR REPLACE] PROCEDURE procedure-name
[(parameter-name [IN | OUT | IN OUT type [, ...])]
   {IS, AS}
BEGIN
      <procedure-body>
END procedure-name;
```

IN :- value that will be passed from outside (Read only) default
OUT :- Parameter that will be used to return a value.
AS :- for creating a standalone procedure.
IS :- " " Packaged "
IN OUT :- passes an initial value and updates & returns the value.

Standalone subprogram (at schema level)

Created with 'CREATE PROCEDURE' or 'CREATE FUNCTI
statement. It is stored in the database and can be
deleted with 'DROP PROCEDURE' or 'DROP FUNCTION' Stateme

Packaged subprogram (package level)

It is stored in the database and can be deleted
only when the package is deleted with 'DROP Package'
statement.

eg. CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
dbms_output.put_line ('Hello');
END;
/

Executing a standalone procedure

- Using the EXECUTE keyword.
- Calling the name of the procedure from a PL/SQL block

Start greetings;

eg. EXECUTE greetings;

To call from another PL/SQL block

BEGIN
greetings;
. . . .
END;
/

eg. DECLARE
a number;
b number;
c number;
PROCEDURE findMin (x IN number, y IN number, z OUT number) IS
BEGIN
IF X < Y THEN
z := X;
ELSE
z := Y;
END IF;
END;
BEGIN
a := 23;
b := 45;
findMin (a, b, c);
dbms_output.put_line ('Min of 23 & 45: ' || c);
END;
/

eg.
```
DECLARE
    a number;
PROCEDURE SquareNum (X IN OUT number) IS
BEGIN
    X := X * X;
END;
BEGIN a := 20;
    SquareNum (a);
    dbms_output.put_line ('Square of 20 :' || a);
END;
/
```

Methods of passing parameters

Formal Parameter

It is the name declared in the parameter list of the header of a module.

eg.

```
PROCEDURE Square (a IN, b IN) IS
```

⇒ a & b are formal parameter.

Actual parameter

It is the value or expressions placed in the parameter list of the actual call to the module.

eg.

```
Square (10, 20);
```

(actual parameter)

```
square (x, y);
```

Passing Actual Parameter

1) Positional Notation

The actual parameter and formal parameter are substituted in its relative order or position.

eg.

```
find Min (a, b, c, d);

PROCEDURE find Min (W IN number, X IN number, Y IN number,
                    Z IN number) IS
BEGIN ..... END;
```

a ⟶ W
b ⟶ X
c ⟶ Y
d ⟶ Z

2) Named Notation

The actual parameter is associated with the formal parameter using the '=>' symbol.

eg. formal-param => actual-param (value)

```
find Min (W => a, X => b, Y => c, Z => d);
```

3) Mixed Notation

Both of the notation can be mixed but positional notation should precede the named ~~local~~ notation.

eg.

```
findMin (a, b, c => x);   ✓
findMin (c => x, b, a);   ✗
```

Q. DECLARE
a number;
b number;
c number;
PROCEDURE findMin(x IN number, y IN number,
z OUT number) IS
BEGIN
if x < y then
z := x;
else
z := y;
end if;
END;
BEGIN
a := 23;
b := 50;
findmin(z => c, y => 10, x => 20);
dbms_output.put_line('Min value:' || c);
findmin(a, b, c);
dbms_output.put_line('Min value:' || c);
findMin(a, y => 10, z => c);
dbms_output.put_line('Min value:' || c);

END;
/

Functions

A function is same as a procedure except that it returns a value.

```
CREATE OR REPLACE function func_name
[(param_name [IN | OUT | IN OUT ] type [, ...])]
RETURN return_datatype
  IS | AS
BEGIN
    ....
END
```

eg.

```
CREATE OR REPLACE function totalEmployees
  RETURN number IS
      total number := 0;      /* -- declaration.
BEGIN
      SELECT count(id) into total
      FROM employee;
      RETURN total;
END;
/
```

Calling a function

```
DECLARE
    c number;
BEGIN
    c := totalEmployees();
    dbms_output.put_line('Total employees:' || c);
END;
/
```

Program

```
DECLARE
    a number;
    b number;
    c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
    z number;
    BEGIN
      IF x > y then
        z := x;
      else
        z := y;
      END IF;   Return z;
    END;
```

```
BEGIN
    a:=20;
    b:=10;
    c:= find max(a,b);
    dbms_output.put_line('max:'||c);
END;
/
```

Recursion in PL/SQL

When a subprogram calls itself, it is referred to as a recursive call and the process is known as recursion.

eg.

```
DECLARE
    num number;
    factorial number;
    FUNCTION fact (x number)
     RETURN number
     IS
        f number;
     BEGIN
        IF x =0 then
           f:= 1
        ELSE
           f:= x * fact(x-1);
        END IF;
```

```
    RETURN f;
END;

BEGIN
    num := 6;
    factorial := fact(num);
    dbms_output.put_line('factorial : '||factorial);
END;
/
```

Overloading subprograms

Overloading is the act of creating multiple subprograms (procedures or functions) with the same name but different parameter lists.

eg. PROCEDURE greet(msg varchar);
    PROCEDURE greet(msg varchar, num number);

eg.
```
DECLARE
    PROCEDURE show(x IN number) IS
    BEGIN
        dbms_output.put_line('value: '||x);
    END;
PROCEDURE show(x IN number, y IN number) IS
BEGIN dbms_output.put_line('value: '||(x+y));
END;
```

```
BEGIN
    show(10);
    show(10,20);
END;
/
```

Unit-6 PL/SQL Packages

Packages are schema objects that groups logically related PL/SQL types, variables and subprograms.

A package has two mandatory parts
- Package specification
- Package body or definition.

Package specification

It declares the types, variables, constants, exceptions, cursors and subprograms that can be referenced from outside of package.

```
CREATE PACKAGE employee_info AS
    PROCEDURE find_salary ( c_id employee.id%type );
END employee_info;
/
```

Package Body

The package body has codes for various procedures, functions declared in the package specifications, which are hidden from the code outside the package.

```
CREATE OR REPLACE PACKAGE BODY employee_info AS
    PROCEDURE find_salary ( c_id employee.id%type ) IS
        c_sal employee.salary %type;
    BEGIN
        SELECT salary INTO c_sal FROM employee
        where id = c_id;
```

to get input : &c_id

```
    dbms_output.put_line('Salary: ' || c_sal);
END find_salary;
END employee_info;
/
```

Using package elements

The package elements (variables, procedures, functions) are accessed with :

```
package_name.element_name;
```

eg.
```
DECLARE
    code employee.id%type := 2;
BEGIN
    employee_info.find_salary(code);
END;
/
```

Example program

Package specification

```
CREATE OR REPLACE PACKAGE emp AS
    PROCEDURE addEmp(eid employee.id%.type, name employee.name
                     sal employee.salary%.type);
    PROCEDURE removeEmp (eid employee.id%.type);
    PROCEDURE listEmp;
END emp;
/
```

Package Body

```
CREATE OR REPLACE PACKAGE BODY AS
    PROCEDURE addEmp(eid employee.id%.type, name employee.name
                     , sal employee.salary%.type) IS
    BEGIN
        INSERT INTO employee values(eid, name, sal);
    END addEmp;

    PROCEDURE removeEmp (eid employee.id%.type) IS
    BEGIN
        DELETE FROM employee where id = eid;
    END removeEmp;
```

```
PROCEDURE listEmp IS ename varchar;
    CURSOR emp-info IS
        SELECT name from employee;
        BEGIN
            open emp-info;
            LOOP
            FETCH emp-info INTO ename;
            EXIT WHEN emp-info%notfound;
            dbms-output.put-line('Name:' ||ename);
            END LOOP;
            close emp-info;
        END;
END emp;
/
```

Using the package

```
DECLARE

BEGIN
    emp.addEmp(10, 'ABC', 5000);
    emp.remove Emp(1);
    emp.listEmp ;

END;
/
```

Date
Page

Advantages of Packages

- Modularity
- Easier application design
- Information hiding
- Better performance

Private and public items in PL/SQL package

Items declared in the body are restricted to use within the package. PL/SQL code outside the package cannot reference the variable, such items are called private.

Items declared in the specification of package are visible outside the package. Such items are called public.

Package STANDARD

A package STANDARD defines the PL/SQL environment. The package specification globally declares types, exceptions and subprograms, which are automatically available to PL/SQL programs.

Foreg. FUNCTION ABS (n number)
RETURN number;

The contents of package STANDARD are directly visible to applications. You need not qualify references to its contents by prefixing the package name.

Overview of Product Specific PL/SQL Packages

Various oracle tools are supplied with product-specific packages that define application programming interfaces (API) that you can invoke from PL/SQL, SQL, Java and other programming languages environments.

1) DBMS_ALERT

It lets you use database triggers to alert an application when specific database values change.

2) DBMS_OUTPUT

It enables you to display output from PL/SQL blocks.

3) DBMS_PIPE

A pipe is an area of memory used by one process to pass information to another. It is used to communicate over different sessions.

You can use PACK_MESSAGE and SEND_MESSAGE to send into a pipe and RECEIVE_MESSAGE and UNPACK_MESSAGE to receive messages.

4) HTF and HTP packages
It lets your PL/SQL program to generate HTML tags.

5) UTL-FILE package
It lets your PL/SQL program to read and write files
FOPEN : to open a file
PUT_LINE : write into a file
GET_LINE : read from a file.

6) UTL-HTTP Package
It enables to make HTTP callouts, It can retrieve data from the internet or invoke web server URLs.

7) UTL-SMTP
It enables to send emails over SMTP (Simple Mail Transfer Protocal)

Guidelines for writing Packages
- Keep the name of the package general so as to reuse in future applications.
- Design & define package specification before the package body.
- place in a spec only those things that must be visible to invoking programs.

Unit-7 Database Triggers

Like a stored procedure, a trigger is a named PL/SQL unit that is stored in the database and can be invoked repeatedly. The triggers can be enabled or disabled, but it can't be explicitly invoked.

While a trigger is enabled, the database automatically invokes it i.e. the trigger fires and while a trigger is disabled, it does not fire.

PROCEDURES	TRIGGERS
• It can be invoked explicitly.	• It cannot be invoked explicitly.
• It can be used within PL/SQL programs.	• It is used automatically when some events happen.

SYNTAX

```
CREATE [OR REPLACE] TRIGGER trigger-name
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] | UPDATE [OR] | DELETE}
  [OF col-name]
ON table-name
[REFERENCING OLD AS O NEW AS N]
[FOR EACH ROW]
WHEN (condition)
```

```
DECLARE
    Declaration-statements
BEGIN
    executable-statements
EXCEPTION
    exception-handling-statements
END;
```

Explanation

- BEFORE | AFTER | INSTEAD OF : This specifies when the trigger will be executed. INSTEAD OF clause is used for creating trigger on a view.
- INSERT OR UPDATE OR DELETE : This specifies the DML operation
- OF col-name :- specifies column name that will be updated.
- ON table-name :- name of the table associated with trigger.
- REFERENCING OLD AS O NEW AS N
  Allows to refer new and old values.
- FOR EACH ROW : row level trigger i.e. a trigger is executed for each row being affected. Otherwise the trigger will execute just once when the SQL stmt is executed, which is called table level trigger.
- WHEN (condition) :- specifies a condition for rows for which the trigger would fire. It is valid only for row level triggers.

Row level trigger

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON employee
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
    sal_diff number;
BEGIN
    sal_diff := :NEW.salary - :OLD.salary;
    dbms_output.put_line('Old salary: ' || :OLD.salary);
    dbms_output.put_line('New Salary: ' || :NEW.salary);
    dbms_output.put_line('Salary diff: ' || sal_diff);
END;
/
```

Statement Triggers

```
CREATE OR REPLACE TRIGGER trig
  BEFORE INSERT OR UPDATE of name, salary OR DEL
  ON employee
  BEGIN
    CASE
    WHEN INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Inserting');
    WHEN UPDATING THEN
      DBMS_OUTPUT.PUT_LINE('Updating');
```

```
WHEN UPDATING('salary') THEN
    DBMS_OUTPUT.PUT_LINE ('updating salary');
WHEN DELETING THEN
    DBMS_OUTPUT.PUT_LINE ('Deleting');
END CASE;
END;
/
```

Note :- A trigger (row or statement level) may modify one or many rows of a same or other tables. The main difference between row and statement level is how many times the trigger is activated.

Imagine you have 1 million rows in a table and you run:

```
UPDATE tbl SET col_name = 0;
```

A statement level trigger will be activated once and even if no rows are updated. A row level trigger will be activated a million times, once for every updated row.

Another difference is the order of execution.

eg. Before stmt level trigger executes
    Before row level trigger executes
    one row is updated
    After row level trigger executes

Before row-level trigger executes
Second row is updated
After row-level trigger executes
.....
After statement level trigger executes

Before vs After triggers

BEFORE triggers run before an action is taken, It is usually used when validation needs to take place before accepting the change. They run before any changes is made to the database.

AFTER triggers are usually used when information needs to be updated in a separate table due to a change. They run after changes have been made to database (not necessarily committed).

Date
Page

Combinations triggers

Multiple DML operations can be combined into one trigger so that all types of event triggers share a common code.

for eg.

```
CREATE OR REPLACE TRIGGER trig
BEFORE | AFTER
INSERT OR UPDATE OR DELETE ON tbl-name
.....
```

Deleting a trigger

```
DROP TRIGGER trigger-name;
```

ENABLE / DISABLE a trigger

```
ALTER TRIGGER trigger-name DISABLE;

ALTER TRIGGER trigger-name ENABLE;
```

Generation of a primary key using a database triggers

1) Create a sequence first.
2) The sequence generates primary keys but these are not inserted into the table automatically.
3) Create a trigger which fires before a row is inserted.

```
CREATE SEQUENCE sequence_pk
INCREMENT BY 1
START WITH 1;

CREATE OR REPLACE TRIGGER trigger_gen_pk
BEFORE INSERT ON employee
FOR EACH ROW
  BEGIN
    :NEW.id := sequence_pk.NEXTVAL;
  END;
```

Unit-8 Handling PL/SQL Errors

In PL/SQL, an error condition is called an exception. Exceptions can be internally defined (by system) or user defined.

When an error occurs, an exception is raised. That is, normal execution stops and control transfers to the exception handling part of PL/SQL block. Internal exceptions are raised implicitly by the run-time system. User defined exceptions must be raised explicitly by RAISE statements, which can also raise predefined exceptions.

Some examples of internal exceptions are ZERO_DIVIDE and STORAGE_ERROR (out of memory).

eg.

```
DECLARE
total_price number := 10.5;
total_quantity number := 0;
unit_price number;
BEGIN
    unit_price := total_price / total_quantity;
dbms_output.put_line('Unit Price :' || unit_price);

EXCEPTION
    WHEN ZERO_DIVIDE THEN
```

- dbms_output.put_line('Total quantity must be non zero.');
WHEN OTHERS THEN -- handles all other errors.
dbms_output.put_line('Error occured.');
END;
/

Advantages of PLSQL exceptions

- Potential errors can be handled which might cause crashing of application.
- Instead of checking for an error at every point, just add an exception handler which saves extra lines of code.
- It catches all possible error that might not be clear or obvious to programmers.
- It increases the readability of the program.

Predefined Exceptions

ACCESS_INTO_NULL : attempt to read values from uninitialized object.
CASE_NOT_FOUND : when none of the choice in WHEN clause matches.
CURSOR_ALREADY_OPEN
DUP_VAL_ON_INDEX : store duplicate values in primary key column.
INVALID_CURSOR : operation on unopened cursor.
INVALID_NUMBER : not a number
NO_DATA_FOUND : select statement returns no rows.
STORAGE_ERROR : out of memory.

User defined exceptions

```
DECLARE
    invalid_id EXCEPTION;
```

Exception and variable declarations are similar, but an exception is an error condition not a data item. Unlike variables, exceptions cannot appear in assignment statement or SQL statements.

eg. DECLARE

```
    invalid_id EXCEPTION;
    eid number;
BEGIN
    ~~select eid from employee~~
    eid := 5;
    If eid < 10 or eid > 20 Then
        raise invalid_id;
    END IF;

EXCEPTION
    WHEN invalid_id Then
        dbms_output.put_line('ID must be bet 10 & 20.

END;
/
```