```
DBMS_OUTPUT.PUT_LINE('Deleted ' || SQL%ROWCOUNT || ' rows:');
FOR i IN e_ids.FIRST .. e_ids.LAST
LOOP
DBMS_OUTPUT.PUT_LINE('Employee #' || e_ids(i) || ' from dept #' || d_ids(i));
END LOOP;
END;
/
```

Passing Parameters to Explicit Cursors

```
DECLARE
emp_job employees.job_id%TYPE := 'ST_CLERK';
emp_salary employees.salary%TYPE := 3000;
my_record employees%ROWTYPE;
CURSOR c1 (job VARCHAR2, max_wage NUMBER) IS
SELECT * FROM employees WHERE job_id = job and salary > max_wage;
BEGIN
-- Any of the following statements opens the cursor:
-- OPEN c1('ST_CLERK', 3000); OPEN c1('ST_CLERK', emp_salary);
-- OPEN c1(emp_job, 3000); OPEN c1(emp_job, emp_salary);
OPEN c1(emp_job, emp_salary);
LOOP
FETCH c1 INTO my_record;
EXIT WHEN c1%NOTFOUND;
-- process data record
DBMS_OUTPUT.PUT_LINE('Name = ' || my_record.last_name || ', salary = ' ||
my_record.salary || ', Job Id = ' || my_record.job_id );
END LOOP;
END;
/
```

http://www.dbanotes.com/database-development/introduction-to-oracle-11g-cursors/

To enable output from PL/SQL blocks with DBMS_OUTPUT.PUT_LINE, use the following:

```
SQL> SET SERVEROUTPUT ON
```

Fetching the Same Cursor Into Different Variables

```
DECLARE
CURSOR c1 IS SELECT last_name FROM employees ORDER BY last_name;
name1 employees.last_name%TYPE;
name2 employees.last_name%TYPE;
name3 employees.last_name%TYPE;
BEGIN
OPEN c1;
FETCH c1 INTO name1; -- this fetches first row
FETCH c1 INTO name2; -- this fetches second row
FETCH c1 INTO name3; -- this fetches third row
CLOSE c1;
END;
/
```

Cursor to delete records:

```
DECLARE
CURSOR C1 IS SELECT EMP_CODE,EMP_SALARY FROM EMP;
```

```
BEGIN
FOR I IN C1
LOOP
DELETE FROM  EMP_SAL
WHERE  EMP_CODE   = I.EMP_CODE;
END LOOP;
END;
```

```
SQL>set serveroutput on
SQL>edit explicit_cursor
DECLARE
      cursor c is select * from emp_information
      where emp_name='bhavesh';
      tmp emp_information%rowtype;
BEGIN
      OPEN c;
      Loop exit when c%NOTFOUND;
            FETCH c into tmp;
            update emp_information set tmp.emp_dept='Web Developer'
            where tmp.emp_name='Saulin';
      END Loop;
IF c%ROWCOUNT>0 THEN
      dbms_output.put_line(SQL%ROWCOUNT||' Rows Updated');
ELSE
      dbms_output.put_line('NO Rows Updated Found');
END IF;
CLOSE c;
END;
/
```

Pasted from <http://www.way2tutorial.com/plsql/plsql_explicit_cursor.php>

```
DECLARE
c_id as str_hub_transaction.id%type;
c_reg as str_hub_transaction.regulator%type;
cursor c_transaction is
select id, regulator from str_hub_transaction;
begin
open c_transaction;
fetch c_transaction into c_id, c_reg;

dbms_output.put_line(c_id || ' ' || c_reg);
close c_transaction;
END;
/
```

Function:

Function must return a value, but in **Stored Procedure** it is optional(**Procedure** can return zero or n values). **F**unction takes one input parameter it is mandatory ,but **Stored Procedure** may take 0 to n input parameters.. **Functions** can be called from **Procedure** whereas **Procedures** cannot be called from **Function**.

- Function must have a return type; for procedure it is optional.
- Function take one input is mandatory; for procedure its optional.

- Function can be called from procedure as there is return type in function; procedure cant be called from function.

```
CREATE OR REPLACE Function FindCourse
  ( name_in IN varchar2 )
  RETURN number
IS
  cnumber number;
cursor c1 is
  SELECT course_number
    FROM courses_tbl
    WHERE course_name = name_in;
BEGIN
open c1;
  fetch c1 into cnumber;
if c1%notfound then
     cnumber := 9999;
  end if;
close c1;
RETURN cnumber;
EXCEPTION
WHEN OTHERS THEN
  raise_application_error(-20001,'An error was encountered - '||SQLCODE||' -ERROR-
'||SQLERRM);
END;
```

Trigger:

http://www.folkstalk.com/2013/01/oracle-plsql-trigger-after-insert-example.html

```
CREATE OR REPLACE TRIGGER update_employees
 AFTER INSERT ON employees
 FOR EACH ROW
BEGIN
 UPDATE employees
 SET   hire_date = sysdate
 WHERE  employee_id = :new.employee_id;
END update_employees;
/
```

```
CREATE [OR REPLACE] TRIGGER trigger_name
      BEFORE | AFTER
      [INSERT, UPDATE, DELETE [COLUMN NAME..]
      ON table_nameReferencing [ OLD AS OLD | NEW AS NEW ]
      FOR EACH ROW | FOR EACH STATEMENT [ WHEN Condition ]
DECLARE
  [declaration_section
        variable declarations;
        constant declarations;
  ]
BEGIN
      [executable_section
            PL/SQL execute/subprogram body
  ]
EXCEPTION
  [exception_section
            PL/SQL Exception block
  ]
END;
```

Pasted from <http://www.way2tutorial.com/plsql/plsql_triggers.php>

Parts of a Trigger

A trigger has three basic parts:

- A triggering event or statement
- A trigger restriction - prevent database operation in core business hrs.
- A trigger action

Trigger Restriction

A trigger restriction specifies a Boolean expression that must be true for the trigger to fire. The trigger action is not run if the trigger restriction evaluates to false or unknown. In the example, the trigger restriction is:
new.parts_on_hand < new.reorder_point

Type of Triggers

1. Triggers are auto commit
   http://www.geekinterview.com/question_details/17674

2. BEFORE Trigger : BEFORE trigger execute before the triggering DML statement (INSERT, UPDATE, DELETE) execute. Triggering SQL statement is may or may not execute, depending on the BEFORE trigger conditions block.
3. AFTER Trigger : AFTER trigger execute after the triggering DML statement (INSERT, UPDATE, DELETE) executed. Triggering SQL statement is execute as soon as followed by the code of trigger before performing Database operation.
4. ROW Trigger : ROW trigger fire for each and every record which are performing INSERT, UPDATE, DELETE from the database table. If row deleting is define as trigger event, when trigger file, deletes the five rows each times from the table.
   A row trigger is fired each time the table is affected by the triggering statement. For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. If a triggering statement affects no rows, a row trigger is not run.
5. Statement Trigger : Statement trigger fire only once for each statement. If row deleting is define as trigger event, when trigger file, deletes the five rows at once from the table.
   Statement Triggers
   A statement trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects, even if no rows are affected. For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once.
   Statement triggers are useful if the code in the trigger action does not depend on the data provided by the triggering statement or the rows affected. For example, use a statement trigger to:
   - Make a complex security check on the current time or user
   - Generate a single audit record
6. Combination Trigger : Combination trigger are combination of two trigger type,
7. Before Statement Trigger : Trigger fire only once for each statement before the triggering DML statement.
8. Before Row Trigger : Trigger fire for each and every record before the triggering DML statement.
9. After Statement Trigger : Trigger fire only once for each statement after the triggering DML statement executing.
10. After Row Trigger : Trigger fire for each and every record after the triggering DML statement executing.
11. Instead of: used to update view tables.
    INSTEAD OF triggers provide a transparent way of modifying views that cannot be modified directly through DML statements (INSERT, UPDATE, and DELETE). These triggers are called INSTEAD OF triggers because, unlike other types of triggers, Oracle fires the trigger instead of executing the triggering statement.
    You can write normal INSERT, UPDATE, and DELETE statements against the view and the INSTEAD OF trigger is fired to update the underlying tables appropriately.

INSTEAD OF triggers are activated for each row of the view that gets modified.

Pasted from <http://www.way2tutorial.com/plsql/plsql_triggers.php>

Trigger is autonomous, the trigger can commit changes to the shadow table whether or not
you commit changes to the main table.

```
CREATE TABLE emp_audit ( emp_audit_id NUMBER(6), up_date DATE,
new_sal NUMBER(8,2), old_sal NUMBER(8,2) );
-- create an autonomous trigger that inserts into the audit table before
-- each update of salary in the employees table
CREATE OR REPLACE TRIGGER audit_sal
BEFORE UPDATE OF salary ON employees FOR EACH ROW
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
INSERT INTO emp_audit VALUES( :old.employee_id, SYSDATE,
:new.salary, :old.salary );
COMMIT;
END;
/
--
```

Statement :

Data definition language (DDL) statements let you to perform these tasks:

- Create, alter, and drop schema objects
- Grant and revoke privileges and roles
- Analyze information on a table, index, or cluster
- Establish auditing options
- Add comments to the data dictionary

Synonyms:
http://www.dba-oracle.com/concepts/synonyms.htm

```
SQL> CREATE SYNONYM emp FOR SCOTT.EMP;
```

```
select * from v$session;
select * from v$process;
Select * from v$bgprocess; --see the background process
select * from v$instance;
select * from v$version;
```

Merge: (update or insert)

The MERGE statement was introduced in Oracle 9i to conditionally insert or update data depending on its presence, a process also known as an "upsert". The MERGE statement reduces table scans and can perform the operation in parallel if required.

https://oracle-base.com/articles/9i/merge-statement

```
MERGE INTO employees e
   USING hr_records h
   ON (e.id = h.emp_id)
```

```
WHEN MATCHED THEN
  UPDATE SET e.address = h.address
WHEN NOT MATCHED THEN
  INSERT (id, address)
  VALUES (h.emp_id, h.address);
```

Rowid - distinct id for a row in a table
Rownum - used in select query to restrict column

Commit , rollback in trigger and procedure
Transactions cannot be started/committed/rolled back from inside stored procedure or trigger code.
Pasted from <http://forums.devshed.com/firebird-sql-development-61/commit-stored-procedure-174130.html>
https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/sqloperations.htm

Ending Transactions
You should explicitly commit or roll back every transaction. Whether you issue the commit or rollback in your PL/SQL program or from a client program depends on the application logic. If you do not commit or roll back a transaction explicitly, the client environment determines its final state.
For example, in the SQL*Plus environment, if your PL/SQL block does not include a COMMIT or ROLLBACK statement, the final state of your transaction depends on what you do after running the block. If you execute a data definition, data control, or COMMIT statement or if you issue the EXIT, DISCONNECT, or QUIT command, Oracle commits the transaction. If you execute a ROLLBACK statement or abort the SQL*Plus session, Oracle rolls back the transaction.

Trace:
https://docs.oracle.com/cd/B10500_01/server.920/a96533/sqltrace.htm

- ALTER SESSION SET SQL_TRACE = TRUE;

When the SQL Trace facility is enabled for a session, Oracle generates a trace file containing statistics for traced SQL statements for that session. When the SQL Trace facility is enabled for an instance, Oracle creates a separate trace file for each process.

To find trace file ; you can correlate session and process table.
select * from v$session where username ='<username>';
select * from v$process where addr= '00000007B48E08A0';

When the SQL **Trace** facility is enabled for an instance, **Oracle** creates a separate **trace** file for each process. Before enabling the SQL **Trace** facility: Check the settings of the TIMED_STATISTICS , MAX_DUMP_FILE_SIZE , and USER_DUMP_DEST initialization parameters

The SQL Trace facility and TKPROF are two basic performance diagnostic tools that can help you monitor and tune applications running against the Oracle Server.

TKProf is an Oracle database utility used to format SQL Trace output into human readable format. The **TKProf** executable is located in the ORACLE HOME/bin directory.

TKProf is a oracle utility to covert trace file in human readable format.

Data types in Oracle:

Scalar -

Single values with no internal components, such as a **NUMBER, DATE,** or **BOOLEAN**, varchar, varcahr2.

Large object (LOB) -
Pointers to large objects that are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms.

Clob: - stores up to 4 GB of character data

Subtype:
A subtype is a subset of another data type, which is called its base type. A subtype has the same valid operations as its base type, but only a subset of its valid values.

```
DECLARE
  SUBTYPE name IS char(20);
  SUBTYPE message IS varchar2(100);
  salutation name;
  greetings message;
BEGIN
  salutation := 'Reader ';
  greetings := 'Welcome to the World of PL/SQL';
  dbms_output.put_line('Hello ' || salutation || greetings);
END;
/
```

Pasted from <https://www.tutorialspoint.com/plsql/plsql_data_types.htm>

```
SET SERVEROUTPUT ON;
DECLARE
  message  varchar2(20):= 'Hello, World!';
BEGIN

  --dbms_output.disable;
  dbms_output.put_line(message);
END;
/
```

```
SET SERVEROUTPUT ON;
DECLARE
 -- message  varchar2(20):= 'Hello, World!';
BEGIN
  for trx in (SELECT * FROM table)
  --dbms_output.disable;
loop
  dbms_output.put_line(trx.id);
end loop;
END;
/
```

NUMBER(p,s)
where:

- *p* is the **precision**, or the total number of digits. Oracle guarantees the portability of numbers with precision ranging from 1 to 38.
- *s* is the **scale**, or the number of digits to the right of the decimal point. The scale can range from -84 to 127.

Print 1 to 10:

```
set serveroutput on;
```

```
declare
i number(2);
begin
for i in reverse 1 .. 10
loop
if (i=5) then exit;
end if;
dbms_output.put_line(i);
end loop;
end;
/
```

Pragma:

Oracle has inducted few keywords in its glossary which behave as a **clue to the compiler**. These keywords are named as Pragma(s), which are read by the compiler to be decisive over the method used for the compilation of current program unit. Note that the method of compilation of a program unit manipulates its run-time behavior.
Pragmas are processed at compile time, not at run time

Pragma EXCEPTION_INIT

To handle error conditions (typically ORA- messages) that have no predefined name, you must use the OTHERS handler or the pragma EXCEPTION_INIT. A pragma is a compiler directive that is processed at compile time, not at run time.
In PL/SQL, the pragma EXCEPTION_INIT tells the compiler to associate an exception name with an Oracle error number. That lets you refer to any internal exception by name and to write a specific handler for it. When you see an error stack, or sequence of error messages, the one on top is the one that you can trap and handle.
You code the pragma EXCEPTION_INIT in the declarative part of a PL/SQL block, subprogram, or package using the syntax

PRAGMA EXCEPTION_INIT(exception_name, -Oracle_error_number);
where exception_name is the name of a previously declared exception and the number is a negative value corresponding to an ORA- error number. The pragma must appear somewhere after the exception declaration in the same declarative section.

```
DECLARE
deadlock_detected EXCEPTION;
PRAGMA EXCEPTION_INIT(deadlock_detected, -60);
BEGIN
NULL; -- Some operation that causes an ORA-00060 error
EXCEPTION
WHEN deadlock_detected THEN
NULL; -- handle the error
END;
/
```

SERIALLY_REUSABLE Pragma - serialization *(state of a object)*

The pragma SERIALLY_REUSABLE indicates that the package state is needed only for the duration of one call to the server. An example could be an OCI call to the database or a stored procedure call through a database link. After this call, the storage for the package variables can be reused, reducing the memory overhead for long-running sessions. For more information, see Oracle Database Application Developer's Guide - Fundamentals.

http://www.exforsys.com/tutorials/oracle-11g/oracle-pragma.html

```
Create PROCEDURE xyz ( i IN integer)
AS PRAGMA AUTONOMOUS_TRANSACTION;
```

The autonomous pragma allows a separate active transaction, which is independent of the main transaction in the session. The COMMIT/ROLLBACK done within the autonomous object does not affect the main transaction and vice versa. All in all, it is an individual transaction and not the part of main transaction.

Transaction Visibility

Changes made by an autonomous transaction become visible to other transactions when the autonomous transaction commits. These changes become visible to the main transaction when it resumes, if its isolation level is set to **READ COMMITTED (the default).**

If you set the isolation level of the main transaction to SERIALIZABLE, changes made by its autonomous transactions are not visible to the main transaction when it resumes:
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Serializable isolation permits concurrent transactions to make only those database changes they could have made if the transactions had been scheduled to run one after another. Specifically, **Oracle permits a serializable transaction to modify a data row only if it can determine that prior changes to the row were made by transactions that**
had committed when the serializable transaction began.

Oracle generates an error when a serializable transaction tries to update or delete data
modified by a transaction that commits after the serializable transaction began:
ORA-08177: Cannot serialize access for this transaction

Pragma RESTRICT_REFERENCES

VARCHAR Vs VARCHAR2

varchar is reserved by Oracle to support distinction between NULL and empty string in future, as ANSI standard prescribes. **VARCHAR2** does not distinguish between a NULL and empty string, and never will. If you rely on empty string and NULL being the same thing, you should use **VARCHAR2**
Emp_name varchar(10) - if you enter value less than 10 then remaining space can not be deleted. it used total 10 spaces.
Emp_name varchar2(10) - if you enter value less than 10 then remaining space is automatically deleted

Oracle accepts NULL values on UNIQUE constraint column

It is clear that oracle implicitly creates unique index when we impose Unique constraint for a column. This is reason why it accepts NULL values too. Because Oracle index can't hold NULL values, so it accepting multiple nulls.

Pasted from <https://sowmianaraynan.wordpress.com/2012/10/17/oracle-accepts-null-values-on-unique-constraint-column/>

Array in Oracle:

```
set serveroutput on;
declare
  type num_a is varray(5) of integer;
  array num_a := num_a(98, 97, 78, 87, 92);
begin
  for i in 1 .. array.count loop
  dbms_output.put_line(array(i));
  end loop;
```

end;
/

Collection in Oracle (table);

■ Arrays in other languages become varrays in PL/SQL.
■ Sets and bags in other languages become nested tables in PL/SQL.
■ Hash tables and other kinds of unordered lookup tables in other languages become associative arrays in PL/SQL.

Nested tables can be stored in a database column, but associative arrays cannot. Nested tables can simplify SQL operations where you would normally join a single-column table with a larger table.

- Variable array
- Associative array
- Nested table

Collection Type	Number of Elements	Subscript Type	Dense or Sparse	Where Created	Can Be Object Type Attribute
Associative array **(or index-by table)**	Unbounded	String or integer	Either	Only in PL/SQL block	No
Nested **table**	Unbounded	Integer	Starts dense, can become sparse	Either in PL/SQL block or at schema level	Yes
Variablesize array (Varray)	Bounded	Integer	Always dense	Either in PL/SQL block or at schema level	Ye

Choosing Between Nested Tables and Varrays
Varrays are a good choice when:
■ The number of elements is known in advance.
Defining Collection Types and Declaring Collection Variables
■ The elements are usually all accessed in sequence
■ **A system-generated database table associated with the nested table**

Nested table
Nested table data is stored in a separate store table, **a system-generated database table**
associated with the nested table. The database joins the tables for you when you access
the nested table. This makes nested tables suitable for queries and updates that only affect some elements of the collection.
You cannot rely on the order and subscripts of a nested table remaining stable as the nested table is stored in and retrieved from the database, because the order and subscripts are not preserved in the database.

A collection method is a built-in function or procedure that operates on collections and
is called using dot notation.

Collection methods make collections easier to use, and make your applications easier
to maintain. These methods include COUNT, DELETE, EXISTS, EXTEND, FIRST, LAST,
LIMIT, NEXT, PRIOR, and TRIM.

For nested tables and associative arrays, which have no declared size, LIMIT returns NULL. For varrays, LIMIT returns the maximum number of elements that a varray can Using Collection Methods
contain. You specify this limit in the type definition, and can change it later with the TRIM and EXTEND methods.

- EXTEND appends one null element to a collection.
- EXTEND(n) appends n null elements to a collection.
- EXTEND(n,i) appends n copies of the ith element to a collection.

- TRIM removes one element from the end of a collection.
- TRIM(n) removes n elements from the end of a collection.

```
DECLARE
TYPE nested_type IS TABLE OF VARCHAR2(30);
TYPE varray_type IS VARRAY(5) OF INTEGER;
TYPE assoc_array_num_type IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
TYPE assoc_array_str_type IS TABLE OF VARCHAR2(32) INDEX BY PLS_INTEGER;
TYPE assoc_array_str_type2 IS TABLE OF VARCHAR2(32) INDEX BY VARCHAR2(64);
v1 nested_type;
v2 varray_type;
v3 assoc_array_num_type;
v4 assoc_array_str_type;
v5 assoc_array_str_type2;
```

```
DECLARE
  TYPE salary IS TABLE OF NUMBER INDEX BY VARCHAR2(20);
  salary_list salary;
  name   VARCHAR2(20);
BEGIN
  -- adding elements to the table
  salary_list('Rajnish') := 62000;
  salary_list('Minakshi') := 75000;
  salary_list('Martin') := 100000;
  salary_list('James') := 78000;

  -- printing the table
  name := salary_list.FIRST;
  WHILE name IS NOT null LOOP
    dbms_output.put_line
    ('Salary of ' || name || ' is ' || TO_CHAR(salary_list(name)));
    name := salary_list.NEXT(name);
  END LOOP;
END;
/
```

- **Nested table - not sorted (there is no INDEX by clause)**

```
DECLARE
  TYPE names_table IS TABLE OF VARCHAR2(10);
  TYPE grades IS TABLE OF INTEGER;
  names names_table;
```

```
  marks grades;
  total integer;
BEGIN
  names := names_table('Kavita', 'Pritam', 'Ayan', 'Rishav', 'Aziz');
  marks:= grades(98, 97, 78, 87, 92);
  total := names.count;
  dbms_output.put_line('Total '|| total || ' Students');
  FOR i IN 1 .. total LOOP
    dbms_output.put_line('Student:'||names(i)||', Marks:' || marks(i));
  end loop;
END;
/
```

- **Variable array**

```
set serveroutput on;
declare
 type num_a is varray(5) of integer;
 array num_a := num_a(98, 97, 78, 87, 92);
begin
 for i in 1 .. array.count loop
 dbms_output.put_line(array(i));
 end loop;
end;
/
```

```
SET serveroutput ON;
DECLARE
 CURSOR sht_1
 IS
   SELECT id
   FROM str_hub_transaction
   WHERE id IN (21764367,21764368,21764373,21764374,21764377);
type id_test
IS
 TABLE OF str_hub_transaction.id%type;
 id_list id_test := id_test();
 counter INTEGER := 0;
BEGIN
 FOR n IN sht_1
 LOOP
   counter:= counter +1;
   id_list.extend;
   id_list(counter):=n.id;
   dbms_output.put_line(id_list(counter));
 END LOOP;
END;
/
```

TYPE(table column) and ROWTYPE(table):

http://psoug.org/snippet/Oracle-PL-SQL-TYPE-vs-ROWTYPE--Whats-the-difference_700.htm

```
-- %TYPE is used to declare a field with the same type as
-- that of a specified table's column:

DECLARE
  v_EmpName  emp.ename%TYPE;
```

```
BEGIN
  SELECT ename INTO v_EmpName FROM emp WHERE ROWNUM = 1;
  DBMS_OUTPUT.PUT_LINE('Name = ' || v_EmpName);
END;
/

 -- %ROWTYPE is used to declare a record with the same types as
-- found in the specified database table, view or cursor:

DECLARE
 v_emp emp%ROWTYPE;
BEGIN
 v_emp.empno := 10;
 v_emp.ename := 'XXXXXXX';
END;
/
```

Procedure and function can be created at :

- At the schema level - use create or replace
- Inside a package
- Inside a PL/SQL block

Record(cursor is a record):

A **record** is a data structure that can hold data items of different kinds. Records consist of different fields, similar to a row of a database table.

```
DECLARE
TYPE DeptRecTyp IS RECORD (
deptid NUMBER(4) NOT NULL := 99,
dname departments.department_name%TYPE,
loc departments.location_id%TYPE,
region regions%ROWTYPE );
dept_rec DeptRecTyp;

TYPE rec4_t IS RECORD (first_name employees.first_name%TYPE,
last_name employees.last_name%TYPE,
rating NUMBER);
```

Updating a Row Using a Record

```
DECLARE
dept_info departments%ROWTYPE;
BEGIN
-- department_id, department_name, and location_id are the table columns
-- The record picks up these names from the %ROWTYPE.
dept_info.department_id := 300;
dept_info.department_name := 'Personnel';
dept_info.location_id := 1700;
-- The fields of a %ROWTYPE can completely replace the table columns
-- The row will have values for the filled-in columns, and null
-- for any other columns
UPDATE departments SET ROW = dept_info WHERE department_id = 300;
END;
/
```

- Table-based
- Cursor-based records
- User-defined records (array)
  Pasted from <https://www.tutorialspoint.com/plsql/plsql_records.htm>

Package:

Packages are schema objects that groups logically related PL/SQL types, variables, and subprograms.
A package will have two mandatory parts –

- Package specification
  The specification is the interface to the package. It just **DECLARES** the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms.
  All objects placed in the specification are called **public** objects. Any subprogram not in the package specification but coded in the package body is called a **private** object.

  Benefits of Packages
  ■ Encapsulation of related procedures and variables
  Stored packages allow you to encapsulate or group stored procedures, variables, datatypes, and so on in a single named, stored unit in the database. This provides better organization during the development process. Encapsulation of procedural constructs also makes privilege management easier. Granting the privilege to use a package makes all constructs of the package accessible to the grantee.
  ■ Declaration of public and private procedures, variables, constants, and cursors
  The methods of package definition allow you to specify which variables, cursors, and procedures are public and private. Public means that it is directly accessible to the user of a package. Private means that it is hidden from the user of a package. For example, a package can contain 10 procedures. You can define the package so that only three procedures are public and therefore available for execution by a user of the package. The remainder of the procedures are private and can only be accessed by the procedures within the package.
  ■ Better performance
  An entire package is loaded into memory when a procedure within the package is called for the first time. This load is completed in one operation, as opposed to the separate loads required for standalone procedures. Therefore, when calls to related packaged procedures occur, no disk I/O is necessary to run the compiled code already in memory.

```
CREATE PACKAGE cust_sal AS
  PROCEDURE find_sal(c_id customers.id%type);
END cust_sal;
/
```

- Package body or definition

```
CREATE OR REPLACE PACKAGE BODY cust_sal AS

  PROCEDURE find_sal(c_id customers.id%TYPE) IS
  c_sal customers.salary%TYPE;
  BEGIN
```

```
    SELECT salary INTO c_sal
    FROM customers
    WHERE id = c_id;
    dbms_output.put_line('Salary: '|| c_sal);
  END find_sal;
END cust_sal;
/
```

Sequential Control

The GOTO statement lets you branch to a label unconditionally. The label, an undeclared identifier enclosed by double angle brackets, must precede an executable statement or a PL/SQL block. When executed, the GOTO statement transfers control to the labeled statement or block, as shown in Example 1–11.

Example 1–11 Using the GOTO Statement

```
DECLARE
total NUMBER(9) := 0;
counter NUMBER(6) := 0;
BEGIN
<<calc_total>>
counter := counter + 1;
total := total + counter * counter;
-- branch to print_total label when condition is true
IF total > 25000 THEN GOTO print_total;
ELSE GOTO calc_total;
END IF;
<<print_total>>
DBMS_OUTPUT.PUT_LINE('Counter: ' || TO_CHAR(counter) || ' Total: ' ||
TO_CHAR(total));
END;
```

Active passive nodes

Active/Passive

An active/passive cluster generally contain two identical nodes. At any time, one of the nodes is active and the other is passive. **Oracle single instance database software is installed on both nodes, but the database is located on shared storage.** During normal operation, the Oracle database instance runs only on the active node. In the event of a failure of the currently active primary system, clustering software will transfer control of the disk subsystem to the secondary system, a process known as failover. As part of the failover process, the Oracle instance on the secondary nose is started, thereby recovering and resuming the service of the Oracle database. **All connections to the instance on the primary system at the time of failure are terminated and must be reestablished withe the new instance on the secondary system.** On resolution of the issue, the Oracle instance is cleanly shut down on the secondary system , control of the disk subsystem is transfered back to the primary system and oracle instance is started to resume normal operations.

a number of third party products are avilable to implement active /passive clustering on linux.

Active/Active

An active /active cluster has an identical hardware infrastruture to an active /passive one. **However, Oracle instances run concurrenlty on both servers and access the same database.** which is located on shared storage. The instances must communicated with each other to negotiate access to shared data in the database. In the event of a server failure, the remaining server can continue processing the workload and optionally failed sessions can be reconnected . the benefit of an active / active cluster over an active/passive cluster is that, **during normal processing, the workload can be shared across both servers in the cluster**

Pasted from <https://mohamedazar.com/2010/06/08/activepassive-vs-active-active/>

How is a process of PL SQL compiled?

- Syntax check
- Binding (reserve memory to hold variable)
- P-code (pre compiled code)

Compilation process includes syntax check (compilation), bind (Storage area to hold variable) and p-code(pre- compiled code) generation processes.

Syntax checking checks the PL SQL codes for compilation errors. When all errors are corrected, a storage address is assigned to the variables that hold data. It is called Binding. P-code is a list of instructions for the PL SQL engine. P-code is stored in the database for named blocks and is used the next time it is executed.

P-code is Pre-complied code stored in Public cache memory of System Global Area after the Oracle instance is started, whereas sourcecode is a simple code of sp, package, trigger, functions etc which are stored in Oracle system defined data dictionary. **Every session of oracle access the p-code which have the Execute permission on that objects.**
Source code stored in user_objects data dictinary for user defined Store proc, Trigger, Package, Function. DBA_object stores all the db objects in sp. DB. ALL_objects stores all the db objects in sp. schema.

DML Statement Processing

Stage 1: Create a Cursor A program interface call creates a cursor. The cursor is created independent of any SQL statement: it is created in expectation of any SQL statement.
In most applications, cursor creation is automatic. However, in precompiler programs, cursor creation can either occur implicitly or be explicitly declared.
Stage 2: Parse the Statement During parsing, the SQL statement is passed from the user process to Oracle, and a parsed representation of the SQL statement is loaded into a shared SQL area. Many errors can be caught during this stage of statement processing.
Parsing is the process of:
- Translating a SQL statement, verifying it to be a valid statement
- Performing data dictionary lookups to check table and column definitions
- Acquiring parse locks on required objects so that their definitions do not change during the statement's parsing (however, parse locks can be broken to allow conflicting DDL operations)
- Checking privileges to access referenced schema objects
- **Determining the optimal execution plan for the statement**
- Loading it into a shared SQL area
- Routing all or part of distributed statements to remote nodes that contain referenced data

Oracle parses a SQL statement only if a shared SQL area for an similar SQL statement does not exist in the shared

Mutation error in oracle:
A mutation table is defined as a table that is changing. But in dealing with triggers, it is a table that has the possibility of changing.

Mutating error with Triggers Occurs only for row level triggers.
not for statement level triggers

http://www.dba-oracle.com/t_avoiding_mutating_table_error.htm
http://www.databasejournal.com/features/oracle/mutating-table-errorin-oraclewhyithappensand-what-you-can-doabout-it.html
http://arunkaushikoracle.blogspot.in/2013/01/what-is-mutation-how-to-avoid-mutation.html

Reserve:

Schudule a procedure :

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name          => 'update_sales',
   job_type          => 'STORED_PROCEDURE',
   job_action        => 'OPS.SALES_PKG.UPDATE_SALES_SUMMARY',
   start_date        => '28-APR-08 07.00.00 PM Australia/Sydney',
   repeat_interval   => 'FREQ=DAILY;INTERVAL=2', /* every other day */
   end_date          => '20-NOV-08 07.00.00 PM Australia/Sydney',
   auto_drop         => FALSE,
   job_class         => 'batch_update_jobs',
   comments          => 'My new job');
END;
/
```

Pasted from
<https://docs.oracle.com/cd/E11882_01/server.112/e25494/scheduse.htm#ADMIN12384>

Pseudo colums:
Pseudo columns are database columns that are using for different purposes in oracle database like ROWNUM, ROWID, SYSDATE, UID, USER, ORA_ROWSCN, SYSTIMESTAMP,

Sequence:
Sequence is an Oracle object that is using for generating the sequence of number. IT=T is normally using for column that have primary key.
"CREATE SEQUENCE TRANS_ID_SEQ
START WITH 1
MAXVALUE 99999999
MINVALUE 1
NOCYCLE
When we use "CYCLE" option in sequence and sequence reaches its MAXVALUE, it will start again from the MINVALUE. That is not good when we are using the sequence for primary key. In "NOCYCLE" option cursor did not start from minimum value again.
NOCACHE
NOORDER"

Oracle stores the definitions of all sequences for a particular database as rows in a single data dictionary table in the SYSTEM tablespace. Therefore, all sequence definitions are always available, because the SYSTEM tablespace is always online.

External table in oracle:

- Sql Loader - load data from flat file to db.
  The default type for external tables is ORACLE_LOADER, which lets you read table data from an external table and load it into a database
- Data pump - load data from table to file.
  Oracle also provides the ORACLE_DATAPUMP type, which lets you unload data (that is, read data from a table in the
  database and insert it into an external table) and then reload it into an Oracle database
- **Index can't be created on external table.**
  External Tables let you query data in a flat file as though the file were an Oracle table. In 9i, only read operations were permitted; in 10g, you can also write out data to an external table, although you can't write to an existing table.

While external tables can be queried, they're not usable in many ways regular Oracle tables are. You cannot perform any DML operations on external tables other than table creation; **one consequence is that you can't create an index on an external table**. External tables are largely used as a convenient way of moving data into and out of the database.
Oracle uses SQL*Loader functionality, through the ORACLE_LOADER access driver to move data from the flat file into the database; it uses a Data Pump access driver to move data out of the db into a file in an Oracle-proprietary format, and back into the database from files of that format. While there are some behaviour differences and restrictions, you can think of external tables as a convenient, SQL-based way to use SQL*Loader and Data Pump functionality.
For example, suppose that you receive a daily .csv report from another department. Instead of writing a SQL*Loader script to import each day's .csv file into your database, you can simply create an external table and write an "insert ... select" SQL query to insert the data directly into your tables. Place the day's CSV file in the location specified in the external table definition, run the query, and you're done.

http://www.orafaq.com/node/848

In Oracle table there can be maximum of 255 columns.

Delete duplicate row from table:
https://sqlandplsql.com/2013/01/29/5-ways-to-delete-duplicate-records-oracle/

Delete from emp where id not in (select max(id) from emp group by id);

Using rowid:
SQL > delete from emp
where rowid not in
(select **max(rowid)** from emp group by empno);

Using selfjoin:
SQL > delete from emp e1
where rowid not in
(select max(rowid) from emp e2
where e1.empno = e2.empno);

What is the difference between UNIQUE and PRIMARY KEY?

The UNIQUE KEY column restricts entry of duplicate values but entry of NULL value is allowed. In case of PRIMARY KEY columns entry of duplicate as well as <null> value is also restricted.

COALESCE returns the first non-null expr in the expression list

The USING clause is used to specify with the column to test for equality when two tables are joined.

[sql]Select * from employee join salary using employee ID[/sql]

Employee tables join with the Salary tables with the Employee ID.

Using a WITH clause: (temporary table or inline view)

```
WITH dept_count AS (
  SELECT deptno, COUNT(*) AS dept_count
  FROM   emp
  GROUP BY deptno)
SELECT e.ename AS employee_name,
       dc.dept_count AS emp_dept_count
FROM   emp e,
```

```
    dept_count dc
WHERE  e.deptno = dc.deptno;
```

Describe <Table_Name> is used to get the field details of a specified table.

SUBSTR returns specific portion of a string and **INSTR** provides character position in which a pattern is found in a string.

Decode and case:

DECODE(expression,search1, result1, search2, result2,...., searchN, resultN, default)

```
select salary,empno,
case
when salary = 12000 then 'high'
when salary = 10000 then 'good'
else 'ok'
end salary_details
from emp e where e.empno in (10,11,12,13,14);
```

Null and empty in Oracle:

1.A NULL value represents the absence of a value for a record in a field (others softwares call it also a missing value).
2.An empty value is a "field-formatted" value with no significant data in it.
3.NULL isn't allocated any memory, the string with NUll value is just a pointer which is pointing to nowhere in memory. however, Empty IS allocated to a memory location, although the value stored in the memory is "".
4.Null has no bounds, it can be used for string, integer, date, etc. fields in a database. Empty string is just regarding a string; it's a string like 'asdfasdf' is, but is just has no length. If you have no value for a field, use null, not an empty string.
5.Null is the database's determination of an absense of a value logically, so to speak. You can query like: where FIELD_NAME is NULL

Which operator that are not coming in Indexing?
NOT IN, like operator is not coming in indexing.

What is exception handling and how we can handle the exception in Oracle?
Exception is an error situation which arises during program execution. When an error occurs exception is raised normally execution is stopped and control transfers to exception handling part. Exception handlers are routines written to handle the exception. The exceptions can be internally defined User-defined exception. In oracle we can handle exception by using these statements.
EXCEPTION WHEN
DUP_VAL_ON_INDEX
NOT_LOGGED_ON
NO_DATA_FOUND -
PL/SQL raises the predefined exception NO_DATA_FOUND if a SELECT INTO statement returns no rows **(%notfound used in cursor)**
TOO_MANY_ROWS **(%rowcount, %bulk_rowcount used in cursor)**
VALUE_ERROR

What is the difference between NO_DATA_FOUND and %NOTFOUND?

"NO_DATA_FOUND" is a system defined exception. It is generated when no record found in implicit cursor. "%NOTFOUND" is used in cursor. If cursor returns no row then %NOTFOUND returns true and if returns row then %NOTFOUND is false.

SGA(System Global Area):
The System Global Area (SGA) is a group of shared memory structures, known as SGA components, **that contain data and control information for one Oracle Database instance**. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.

Return and return statement:
http://psoug.org/definition/RETURN.htm

- **As a clause:**
```
CREATE OR REPLACE FUNCTION F_RET_VAL
RETURN NUMBER
IS
BEGIN
RETURN 1;
END;
```

- **As a statement:**
```
CREATE OR REPLACE PROCEDURE P_RET_VAL (P_EMPNO NUMBER DEFAULT NULL)
IS
BEGIN
IF P_EMPNO IS NULL THEN
 RETURN;
ELSE
 DBMS_OUTPUT.PUT_LINE(P_EMPNO);
END IF;
END;
```

what are v$ tables in oracle (Dynamic performance table)
The **Oracle** Server contains a set of underlying views that are maintained by the server and accessible to the database administrator user SYS. These views are called dynamic performance views because they are continuously updated while a database is open and in use, and their contents relate primarily to performance.

Dynamic Performance Tables V$
Dynamic performance tables are not true tables, and they should not be accessed by most users. However, database administrators can query and create views on the tables and grant access to those views to other users. These views are sometimes called fixed views because they cannot be altered or removed by the database administrator. **SYS owns the dynamic performance tables; their names all begin with V_$**. Views are created on these tables, and then public synonyms are created for the views. The synonym names begin with V$. For example, the V$DATAFILE view contains information about the database's datafiles, and the V$FIXED_TABLE view contains information about all of the dynamic performance tables and views in the database.

Dynamic SQL with PL/SQL? Execute immediate
Dynamic SQL enables you to build SQL statements dynamically at runtime. You can create more general purpose, flexible applications by using dynamic SQL because the full text of a SQL statement may be unknown at compilation.
To process most dynamic SQL statements, you use the EXECUTE IMMEDIATE statement.

The EXECUTE IMMEDIATE statement prepares (parses) and immediately executes a

dynamic SQL statement or an anonymous PL/SQL block. The main argument to EXECUTE IMMEDIATE is the string containing the SQL statement to execute. You can build up the string using concatenation, or use a predefined string.

```
CREATE OR REPLACE PROCEDURE raise_emp_salary (column_value NUMBER,
emp_column VARCHAR2, amount NUMBER) IS
v_column VARCHAR2(30);
sql_stmt VARCHAR2(200);
BEGIN
-- determine if a valid column name has been given as input
SELECT COLUMN_NAME INTO v_column FROM USER_TAB_COLS
WHERE TABLE_NAME = 'EMPLOYEES' AND COLUMN_NAME = emp_column;
sql_stmt := 'UPDATE employees SET salary = salary + :1 WHERE '
|| v_column || ' = :2';
EXECUTE IMMEDIATE sql_stmt USING amount, column_value;
IF SQL%ROWCOUNT > 0 THEN
DBMS_OUTPUT.PUT_LINE('Salaries have been updated for: ' || emp_column
|| ' = ' || column_value);
END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE ('Invalid Column: ' || emp_column);
END raise_emp_salary;
/
```

```
DECLARE
sql_stmt VARCHAR2(200);
v_column VARCHAR2(30) := 'DEPARTMENT_ID';
dept_id NUMBER(4) := 46;
dept_name VARCHAR2(30) := 'Special Projects';
mgr_id NUMBER(6) := 200;
loc_id NUMBER(4) := 1700;
BEGIN
-- note that there is no semi-colon (;) inside the quotes '...'
EXECUTE IMMEDIATE 'CREATE TABLE bonus (id NUMBER, amt NUMBER)';
sql_stmt := 'INSERT INTO departments VALUES (:1, :2, :3, :4)';
EXECUTE IMMEDIATE sql_stmt USING dept_id, dept_name, mgr_id, loc_id;
EXECUTE IMMEDIATE 'DELETE FROM departments WHERE ' || v_column || ' = :num'
USING dept_id;
EXECUTE IMMEDIATE 'ALTER SESSION SET SQL_TRACE TRUE';
EXECUTE IMMEDIATE 'DROP TABLE bonus';
END;
/
```

Sub programming - declaring function/procedure in declare section of main procedure/function.

- **A procedure/function can be defined in a exiting procedure - Sub programming.**

```
DECLARE
FUNCTION square(original NUMBER)
RETURN NUMBER AS original_squared NUMBER;
BEGIN
original_squared := original * original;
RETURN original_squared;
END;
BEGIN
DBMS_OUTPUT.PUT_LINE(square(100));
END;
```

/

Overloading Subprogram Names

PL/SQL lets you overload subprogram names and type methods. You can use the same name for several different subprograms as long as their formal parameters differ in number, order, or datatype family. F
The procedures initialize different types of collections. Because the processing in these two procedures is the same, it is logical to give them the same name.

You cannot overload subprograms whose parameters differ only in subtype. For example, you cannot overload procedures where one accepts an INTEGER parameter and the other accepts a REAL parameter, even though INTEGER and REAL are both subtypes of NUMBER and so are in the same family.
You cannot overload two functions that differ only in the datatype of the return value, even if the types are in different families. For example, you cannot overload two functions where one returns BOOLEAN and the other returns INTEGER.

How Oracle Does Implicit Rollbacks

Before executing an INSERT, UPDATE, or DELETE statement, Oracle marks an implicit savepoint (unavailable to you). If the statement fails, Oracle rolls back to the savepoint.

In PL/SQL code the RETURN keyword can appear in any subprogram as a statement. As soon as it is encountered in the subprogram body, Oracle server immediately sends the execution control back to the immediate calling code or host environment. No further statements are processed in the block following the **RETURN** statement.
n the above syntax, when the execution pointer comes to the RETURN statement, Oracle skips all subsequent statements and returns to the calling environment.

Pasted from <https://www.daniweb.com/programming/databases/threads/145832/what-is-the-main-differences-between-null-value-and-blank#post691376>

Interview question:

http://www.oracleatoz.com/oracle-sqlplsql-questions-answers1

Datafiles

Every Oracle database has one or more physical datafiles. The datafiles contain all the database data. The data of logical database structures, such as tables and indexes, is physically stored in the datafiles allocated for a database.
The characteristics of datafiles are:
- A datafile can be associated with only one database.
- Datafiles can have certain characteristics set to let them automatically extend when the database runs out of space.
- One or more datafiles form a logical unit of database storage called a tablespace.

STSTEM Tablespace

Every Oracle database contains a tablespace named SYSTEM, which Oracle creates automatically when the database is created. The SYSTEM tablespace is always online when the database is open.
The first tablespace in any database is
always the SYSTEM tablespace, so Oracle automatically allocates the first datafiles of any database for the SYSTEM tablespace during database creation.

SYSAUX Tablespace

The SYSAUX tablespace is an auxiliary tablespace to the SYSTEM tablespace. Many database components use the SYSAUX tablespace as their default location to store data. Therefore, the SYSAUX tablespace is always created during database creation or database upgrade.

The SYSAUX tablespace provides a centralized location for database metadata that does not reside in the SYSTEM tablespace. It reduces the number of tablespaces created by default, both in the seed database and in user-defined databases.

Undo Tablespaces

Undo tablespaces are special tablespaces used solely for storing undo information. You cannot create any other segment types (for example, tables or indexes) in undo tablespaces. Each database contains zero or more undo tablespaces.

CREATE
UNDO TABLESPACE

You can bring a tablespace online only in the database in which it was created because the necessary data dictionary information is maintained in the SYSTEM tablespace of that database. An offline tablespace cannot be read or edited by any utility other than Oracle. Thus, offline tablespaces cannot be transposed to other databases.

To move or copy a set of tablespaces, you must make the tablespaces read only, copy the datafiles of these tablespaces, and use export/import to move the database information (metadata) stored in the data dictionary. Both the datafiles and the metadata export file must be copied to the target database.

SPFILE

To start an instance, Oracle must read either an initialization parameter file or a server parameter file. These files contain a list of configuration parameters for that instance and database. Oracle traditionally stored initialization parameters in a text initialization parameter file. You can also choose to maintain initialization parameters in a binary server parameter file (SPFILE).

The initialization parameters tell Oracle:

- The name of the database for which to start up an instance
- How much memory to use for memory structures in the SGA
- What to do with filled redo log files
- The names and locations of the database control files
- The names of undo tablespaces in the database

Database Startup

The three steps to starting an Oracle database and making it available for systemwide use are:

1. Start an instance.
2. Mount the database.
3. Open the database.

Control Files

A control file contains information about the associated database that is required for access by an instance, both at startup and during normal operation. Control file information can be modified only by Oracle; no database administrator or user can edit a control file.

Among other things, a control file contains information such as:

- The database name
- The timestamp of database creation
- The names and locations of associated datafiles and redo log files
- Tablespace information
- Datafile offline ranges
- The log history
- Archived log information
- Backup set and backup piece information

- Backup datafile and redo log information
- Datafile copy information
- The current log sequence number

If the physical makeup of the database is altered (for example, if a new datafile or redo log file is created), then the control file is automatically modified by Oracle to reflect the change. A control file is also used in database recovery.

Redo Log Files

Every Oracle database has a set of two or more redo log files. The set of redo log files is collectively known as the redo log for the database. A redo log is made up of redo entries (also called redo records).
The primary function of the redo log is to record all changes made to data. If a failure prevents modified data from being permanently written to the datafiles, then the changes can be obtained from the redo log, so work is never lost.
The online redo log is a set of two or more online redo log files that record all changes made to the database, including uncommitted and committed changes. Redo entries are temporarily stored in redo log buffers of the system global area,

The information in a redo log file is used only to recover the database from a system or media failure that prevents database data from being written to the datafiles. For example, if an unexpected power outage terminates database operation, then data in memory cannot be written to the datafiles, and the data is lost. However, lost data can be recovered when the database is opened, after power is restored. By applying the information in the most recent redo log files to the database datafiles, Oracle restores the database to the time at which the power failure occurred.
The process of applying the redo log during a recovery operation is called rolling forward.

Undo Records - uncommitted changes

Undo records are stored in undo tablespaces. Oracle uses the undo data for a variety of purposes, including accessing before-images of blocks changed in uncommitted transactions. **During database recovery, Oracle applies all changes recorded in the redo log and then uses undo information to roll back any uncommitted transactions.**
Rolling back means undoing any changes to data that have been performed by SQL statements within an uncommitted transaction. Oracle uses undo tablespaces (or rollback segments) to store old values. The redo log contains a record of changes.

External Views

Monitor transaction and undo information with V$TRANSACTION and V$ROLLSTAT.
For automatic undo management, the information in V$ROLLSTAT reflects the behaviors of the automatic undo management undo segments.

v$transaction - to get info in running transactions.

In-doubt transactions - transaction commit was interrupted.

in-doubt distributed transactions—distributed transactions in which the commit was interrupted by any type of system or network failure.

DBA_2PC_PENDING	Lists all in-doubt distributed transactions. The view is empty until populated by an in-doubt transaction. After the transaction is resolved, the view is purged.

Table Compression

Oracle's table compression feature compresses data by eliminating duplicate values in a database block.
Deleting and inserting in compress data is fast. Updating is slower in some case.
Deleting compressed data is as fast as deleting
uncompressed data. Inserting new data is also as fast, because data is not compressed

in the case of conventional INSERT; it is compressed only doing bulk load. Updating compressed data can be slower in some cases.

Nulls Indicate Absence of Value

A null is the absence of a value in a column of a row. Nulls indicate missing, unknown, or inapplicable data.
Nulls are stored in the database if they fall between columns with data values. In these cases they require 1 byte to store the length of the column (zero).
Trailing nulls in a row require no storage because a new row header signals that the remaining columns in the previous row are null. For example, if the last three columns of a table are null, no information is stored for those columns. In tables with many columns, the columns more likely to contain nulls should be defined last to conserve disk space.

Nulls are not indexed, except when the cluster key column value is null or the index is a bitmap index.

Trace Log Files

Each server and background process can write to an associated trace file. When an internal error is detected by a process, it dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, while other information is for Oracle Support Services. Trace file information is also used to tune applications and instances.

Tablespaces

A database is divided into logical storage units called tablespaces, which group related logical structures together. For example, tablespaces commonly group together all application objects to simplify some administrative operations.
Each database is logically divided into one or more tablespaces. One or more datafiles are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace. The combined size of the datafiles in a tablespace is the total storage capacity of the tablespace.
Every Oracle database contains a SYSTEM tablespace and a SYSAUX tablespace. Oracle creates them automatically when the database is created. The system default is to create a smallfile tablespace, which is the traditional type of Oracle tablespace. The SYSTEM and SYSAUX tablespaces are created as smallfile tablespaces.

Oracle Data Blocks

At the finest level of granularity, Oracle database data is stored in data blocks. One data block corresponds to a specific number of bytes of physical database space on disk. The standard block size is specified by the DB_BLOCK_SIZE initialization

Extents

The next level of logical database space is an extent. An extent is a specific number of contiguous data blocks, obtained in a single allocation, used to store a specific type of information.

Segments

Above extents, the level of logical database storage is a segment. A segment is a set of extents allocated for a certain logical structure. The following table describes the different types of segments.

Oracle allocates space for segments in units of one extent. When the existing extents of a segment are full, Oracle allocates another extent for that segment. Because extents are allocated as needed, **the extents of a segment may or may not be contiguous on disk.**

A segment and all its extents are stored in one tablespace. Within a tablespace, a segment can include extents from more than one file; that is, the segment can span datafiles. However, each extent can contain data from only one datafile.

The high water mark is the boundary between used and unused space in a segment.

The PCTFREE Parameter (database block parameter)
The PCTFREE parameter sets the minimum percentage of a data block to be reserved as free space for possible updates to rows that already exist in that block. For example, assume that you specify the following parameter within a CREATE TABLE statement:
PCTFREE 20
This states that 20% of each data block in this table's data segment be kept free and available for possible updates to the existing rows already within each block

The PCTUSED Parameter(database block parameter)s
The PCTUSED parameter sets the minimum percentage of a block that can be used for row data plus overhead before new rows are added to the block. After a data block is filled to the limit determined by PCTFREE, Oracle considers the block unavailable for the insertion of new rows until the percentage of that block falls beneath the parameter PCTUSED.

Extents in Clustered Tables
Clustered tables store information in the data segment created for the cluster. Therefore, if you drop one table in a cluster, the data segment remains for the other tables in the cluster, and no extents are deallocated. You can also truncate clusters (except for hash clusters) to free extents.

Rollback segment
If you are operating in automatic undo management mode, then the database server manages undo space using tablespaces. Oracle recommends that you use automatic undo management.

Oracle does use a SYSTEM rollback segment for performing system transactions. There is only one SYSTEM rollback segment and it is created automatically at CREATE DATABASE time and is always brought online at instance startup. You are not required to perform any operations to manage the SYSTEM rollback segment.

Temporary Segments
When processing queries, Oracle often requires temporary workspace for intermediate stages of SQL statement parsing and execution. Oracle automatically allocates this disk space called a temporary segment. Typically, Oracle requires a temporary segment as a database area for sorting.
Operations that Require Temporary Segments
The following statements sometimes require the use of a temporary segment:
- CREATE INDEX
- SELECT ... ORDER BY
- SELECT DISTINCT ...
- SELECT ... GROUP BY
- SELECT . . . UNION
- SELECT ... INTERSECT
- SELECT ... MINUS

If no temporary tablespace is defined for the user, then the default temporary tablespace is the SYSTEM tablespace.

Manually create temp segment -

```
SQL> CREATE TABLESPACE TEMPTBS DATAFILE '/path/temp.dbf' SIZE 2048M
AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED LOGGING DEFAULT NOCOMPRESS
ONLINE EXTENT MANAGEMENT DICTIONARY;
```

Schemas and Common Schema Objects
A schema is a collection of database objects. A schema is owned by a database user

and has the same name as that user. Schema objects are the logical structures that directly refer to the database's data. Schema objects include structures like tables, views, and indexes. (There is no relationship between a tablespace and a schema. Objects in the same schema can be in different tablespaces, and a tablespace can hold objects from different schemas.)

Data Dictionary

Each Oracle database has a data dictionary. An Oracle data dictionary is a set of tables and views that are used as a read-only reference about the database. For example, a data dictionary stores information about both the logical and physical structure of the database. A data dictionary also stores the following information:

- The valid users of an Oracle database
- The definitions of all schema objects in the database (tables, views, indexes, clusters, synonyms, sequences, procedures, functions, packages, triggers, and so on)
- Information about integrity constraints defined for tables in the database **(primary key)**
- The amount of space allocated for a schema object and how much of it is in use

A data dictionary is created when a database is created. To accurately reflect the status of the database at all times, the data dictionary is automatically updated by Oracle in response to specific actions, such as when the structure of the database is altered. The database relies on the data dictionary to record, verify, and conduct ongoing work. For example, during database operation, Oracle reads the data dictionary to verify that schema objects exist and that users have proper access to them.

The data dictionary is structured in tables and views, just like other database data. All **the data dictionary tables and views for a given database are stored in that database's SYSTEM tablespace.**

```
SELECT object_name, object_type FROM USER_OBJECTS;
SELECT owner, object_name, object_type FROM ALL_OBJECTS;
SELECT owner, object_name, object_type FROM SYS.DBA_OBJECTS;
```

Cache the Data Dictionary for Fast Access

Much of the data dictionary information is kept in the SGA in the dictionary cache, because Oracle constantly accesses the data dictionary during database operation to validate user access and to verify the state of schema objects. All information is stored in memory using the least recently used (LRU) algorithm.

The Oracle Instance

An Oracle database server consists of an Oracle database and an Oracle instance. Every time a database is started, a system global area (SGA) is allocated and Oracle background processes are started.
The combination of the background processes and memory buffers is called an Oracle instance.

SGA is called as memory buffer

Oracle memory structure

System Global Area

The System Global Area (SGA) is a shared memory region that contains data and control information for one Oracle instance. Oracle allocates the SGA when an instance starts and deallocates it when the instance shuts down. Each instance has its own SGA. Users currently connected to an Oracle database share the data in the SGA. For optimal performance, the entire SGA should be as large as possible (while still fitting in real memory) to store as much data in memory as possible and to minimize disk I/O.
The information stored in the SGA is divided into several types of memory structures, including the database buffers, redo log buffer, Data dictionary cache, and the shared pool.

LRU - least recently used
MRU - most recently used

Program Global Area

The Program Global Area (PGA) is a memory buffer that contains data and control information for a server process. A PGA is created by Oracle when a server process is started. The information in a PGA depends on the Oracle configuration

Oracle Background Processes

An Oracle database uses memory structures and processes to manage and access the database. All memory structures exist in the main memory of the computers that constitute the database system. Processes are jobs that work in the memory of these computers.

select * from v$bgprocess;

PMON - process monitor
SMON - process responsible for system monitor and recovery.
DBWn - DB writter - The database writer process (DBWn) writes the contents of buffers to datafiles.
LGWR - log writer - This process responsible to write redo buffer to redo log file on disk.
QMON - queue monitor
ARCn - Archiver process

How Oracle Works

The following example describes the most basic level of operations that Oracle performs. This illustrates an Oracle configuration where the user and associated server process are on separate computers (connected through a network).

1. An instance has started on the computer running Oracle (often called the host or database server).
2. A computer running an application (a local computer or client workstation) runs the application in a user process. The client application attempts to establish a connection to the server using the proper Oracle Net Services driver.
3. The server is running the proper Oracle Net Services driver. The server detects the connection request from the application and creates a dedicated server process on behalf of the user process.
4. The user runs a SQL statement and commits the transaction. For example, the user changes a name in a row of a table.
5. The server process receives the statement and checks the shared pool for any shared SQL area that contains a similar SQL statement. If a shared SQL area is found, then the server process checks the user's access privileges to the requested data, and the previously existing shared SQL area is used to process the statement. If not, then a new shared SQL area is allocated for the statement, so it can be parsed and processed.
6. The server process retrieves any necessary data values from the actual datafile (table) or those stored in the SGA.
7. The server process modifies data in the system global area. The DBWn process writes modified blocks permanently to disk when doing so is efficient. Because the transaction is committed, the LGWR process immediately records the transaction in the redo log file.
8. If the transaction is successful, then the server process sends a message across the network to the application. If it is not successful, then an error message is transmitted.
9. Throughout this entire procedure, the other background processes run, watching for conditions that require intervention. In addition, the database server manages other users' transactions and prevents contention between transactions that request the same data.

Oracle uses several structures to provide complete recovery from an instance or disk failure: the redo log, undo records, a control file, and database backups

Sql statements

DML
DDL
Transaction control - commit, roll back
Session control - alter session
System control - alter system
Embedded sqt stmt - open , fetch, close, execute

LogMiner (redo logs) - read content of redo log through sql console.
Oracle LogMiner enables you to query redo log files through a SQL interface. All changes made to user data or to the database dictionary are recorded in the Oracle redo log files. Therefore, redo log files contain all the necessary information to perform recovery operations.

Concurrency Control

Oracle automatically provides read consistency to a query so that all the data that the query sees comes from a single point in time (statement-level read consistency).
Oracle can also provide read consistency to all of the queries in a transaction (transaction-level read consistency).
Oracle uses the information maintained in its rollback segments to provide these consistent views. The rollback segments contain the old values of data that have been changed by uncommitted or recently committed transactions.

Flashback Query

Oracle Flashback Query lets you view and repair historical data. You can perform queries on the database as of a certain wall clock time or user-specified system change number (SCN).
Flashback Query uses Oracle's multiversion read-consistency capabilities to restore data by applying undo as needed. Oracle Database 10g automatically tunes a parameter called the undo retention period. The undo retention period indicates the amount of time that must pass before old undo information—that is, undo information for committed transactions—can be overwritten.

```
CREATE TABLE old_emp AS SELECT *
FROM employees AS OF TIMESTAMP '2002-02-05 14:15:00'
```

Recovery Manager (RMAN) is an Oracle utility that can back up, restore, and recover database files. It is a feature of the Oracle database server and does not require separate installation.

Difference between 10g, 11g, 12c
http://hammadnavaidblog.blogspot.com/2017/10/difference-between-10g-11g-and-12c.html

Good link
https://www.quora.com/Whats-the-difference-between-Oracle-10g-and-11g-What-are-some-advantages-of-each

11g

Introduction of ADR (Automatic data recovery), single location of all error and trace data in the database, diagnostic_dump parameter
Flashback Database Archive (FBDA)
Read only table
Invisible index
Password security - mixed case in password security

Compared with 10g, 11g provides more simplified, improved and automated memory management and better ability to diagnose faults through inbuilt infrastructure to prevent, detect, diagnose, and help resolve critical database errors, as well as, low database performance issues. It provides invisible indexes, virtual columns, table partitioning and the

ability to redefine tables which have materialized view logs whilst online. A major difference in the two are the new security features found in 11g such as better password-based authentication with mixed case passwords, encryption on tablespace-level and enhancements for data pump encryption and compression.
11g continued the use of different editions used in 10g which are Enterprise Edition (EE), Standard Edition (SE), Standard Edition One (SE1), Express Edition (EX) and Oracle Database Lite for mobile devices.

Conclusion

All in all, 11g is a good upgrade from 10g with many positive enhancements on an evolving technology. The technical documentation which was good in 10g has become even better in 11g, a significant benefit for the DBAs, who depend on it daily. It is common for organizations not to utilize the full features of an Oracle database. Therefore, the benefits of an upgraded version must be properly utilized for the organization to reduce their cost of ownership, downtime and increase performance, which 11g can deliver.