ORACLE

Data :- **Data** is a collection of facts, information.

We have two types of data.

1) Unstructured data

2) Structured data

1) Unstructured data

Data which is not in readable format is called **unstructured data**.

In general, **meaningless** data is called **unstructured data**.

ex: 201 Lakemba SYD NSW AUS

2) Structured data

Data which is in **readable** format is called **structured data**.

In general, **meaningfull** data is called structured data.

ex:

Unit	Locality	City	State	Country
201	Lakemba	SYD	NSW	AUS

Oracle

It is one of the database which is used to store **structured data.**

It is a **RDBMS** (Relational Database Management System) database.

It is a product of **Oracle Corporation**.

It is classified into two types.

Oracle

|

| |

SQL **PL/SQL**

(Structured Query Language) (Procedural / Structured Query Language

Management System

It is a software which is used to manage the database.

Using management system we can perform following acitivities very easily.

1) Adding the new data

2) Updating the existing data

3) Deleting the unnecessary data

4) Selecting the required data

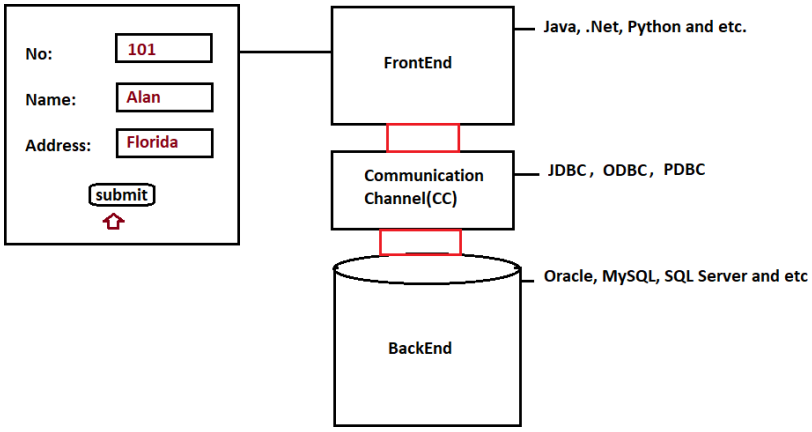
Q) What is the difference between DBMS and RDBMS ?

DBMS	RDBMS
It stands for Database Management System.	It stands for **Relational** Database Management System.
It stores the data in **files**.	It stores the data in **tables**.
It is not designed to store **large** amount of data.	It is designed to store **large amount of data**.
It provides support for **one user** at a time.	It provides support for **multiple users** at a time.
It does not support **normalization**.	It supports **normalization.** (It will remove data redundency)
There is no data **security.**	There is a **high** data security.

Client-Server Architecture

In this architecture we will see how our frontend data will store in a backend.

Diagram:oracle1.1



Frontend :-

The one which is visible to the **enduser** to perform some operations is called frontend.

ex:

Java, .Net, Python, Perl , Kotlin and etc.

Communication Channel :-

It acts like a bridge between **frontend and backend**.

ex:

JDBC - Java Database Connectivity

ODBC - Open Database Connectivity

PDBC - Python Database Connectivity

BackEnd

The one which is not visible to the enduser but it performs operations based on the instructions given by frontend is called backend.

ex:

Oracle, MySQL, SQL Server, MongoDB, NoSQL, PostegeSQL, Teradata, Sybase, DB2 and etc.

SQL

SQL stands for Structured Query Language which is pronounce as **SEQUEL.**

This language is used to interact with **oracle database software**.

It is a case **insensitive** language.

It is a **command** based language.

Every command must **starts with verb**.

ex:

insert, update, delete, merge and etc.

Every command must ends with **semicolon.**

It is developed in late **1972** by **Mr.Codd (By IBM).**

Sub languages of SQL

We have five sub languages of SQL.

1) DDL (Data Definition Language) :- **create, alter, drop , truncate and rename**

2) DML (Data Manipulation Language) :- **insert, update, delete and merge.**

3) DRL/**DQL** (Data Retrieve/Query Language) :- **select**

4) **TCL** (Transaction Control Language) :-

5) **DCL** (Data Control Language) :- Error! Bookmark not defined.

1) **DDL** (Data Definition Language)

This language is used to maintain the objects in database.

It is a collection of five commands.

ex:

create, alter, drop , truncate and rename.

2) **DML** (Data Manipulation Language)

This language is used to manipulate the data present in database.

It is a collection of four commands.

ex:

insert, update, delete and merge.

3) **DRL/DQL** (Data Retrieve/Query Language)

This language is used to retrieve the data from database.

It is a collection of one command.

ex:

select

4) **TCL** (Transaction Control Language)

This language is used to maintain the transaction of database.

It is a collection of three commands.

ex:

commit, rollback and savepoint.

5) **DCL** (Data Control Language)

This language is used to control the access of data to the user.

It is a collection of two commands.

ex:

grant and revoke

Table

A Table is an object which is used to represent the data.

A table is a collection of **Rows and columns**

Oracle is a case **insensitive**. But data which is present in a table is a case **sensitive**.

ex:

SNO	SNAME	SADD
101	Alan	Florida
102	Jose	Texas
103	Nelson	Miami

Here we **have 3 rows and 3 columns**.

Oracle

Version : 10g or 11g

Vendor : Oracle Corporation

Website : www.oracle.com/in/database

Port No : 1521

Software : Express Edition

Username : system (default)

Password : admin

Download link :

https://drive.google.com/file/d/0B9rC21sL6v0td1NDZXpkUy1oMm8/view?usp=drive_link&resourcekey=0-aKooR3NmAh_eLo_qGw_in

Establish the connection with database software

To execute any command or query in a database we need to establish the connection.

Once work with database is completed we need to close the connection.

ex:1

SQL> connect

username : system

password : admin

SQL> disconnect

ex:2

SQL> conn

username: system

password : admin

SQL> disc

ex:3

SQL> conn system/admin

SQL> disc

create command

It is used to **create** a table in a database.

syntax:

---

create table <table_name>(col1 datatype(size),col2 datatype(size), ...,colN datatype(size));

ex:

create table student(sno number(3),sname varchar2(10),sadd varchar2(12));

create table dept(deptno number(3),dname varchar2(10),dloc varchar2(10));

create table emp(eid number(3), ename varchar2(10),esal number(10,2),deptno number(3), jobvarchar2(10), comm number(8));

Describe command

It is used to display **structure of a table.**

syntax:

desc <table_name>;

ex:

desc student;

desc emp;

desc dept;

insert command

It is used to insert a **record/row** in a database table.

syntax:

insert into <table_name> values(value1,value2, ...,valueN);

ex

insert into student values(101,'raja','hyd');

insert into student values(102,'ravi',null);

Note:A null is a keyword which represent undefined or unavailable.

approach2

insert into student(sno,sname,sadd) values(103,'ramana','vizag');

insert into student(sno,sname) values(104,'ramulu');

approach3

Using '&' symbol we can read dynamic inputs in SQL.

ex:

insert into student values(&sno,'&sname','&sadd');

commit command

It is used to make the changes permanent to database.

syntax:

commit;

ex: commit;

Dept table

```
create table dept(deptno number(3),dname varchar2(10),dloc varchar2(10));
insert into dept values(10,'ECE','HYD');
insert into dept values(20,'EEE','DELHI');
insert into dept values(30,'CSE','VIZAG');
insert into dept values(40,'MEC','PUNE');
commit;
```

Emp table

```
create table emp(eid number(3), ename varchar2(10),esal number(10,2),deptno number(3), job varchar2(10), comm number(8));
insert into emp values(201,'Alan',9000,10,'Clerk',null);
insert into emp values(202,'Jose',19000,10,'Clerk',500);
insert into emp values(203,'Nancy',23000,20,'HR',500);
insert into emp values(204,'Kelvin',37000,20,'HR',600);
insert into emp values(205,'Mark',31000,30,'Manager',900);
insert into emp values(206,'Linda',42000,30,'Manager',300)
commit;
```

select command

It is used to select the records from database table

syntax:

select * from <table_name>;

Here '*' means all rows and columns.

ex:

```
select * from student;
select * from dept;
select * from emp;
```

Projection

Selecting specific **columns** from the table is called projection.

ex:

select sno,sname,sadd from student

select sno,sname from student;

select sname from student;

Column alias

A userdefined **heading** given to a column is called column alias.

Column alias are **temperory**.

ex:

select sno,sname,sadd from student;

select sno as rollno,sname as name, sadd as city from student;

Arithmetic operation

In select command we can perform arithmetic operations also.

ex:

select sno,sname,sadd from student;

select sno+100,sname,sadd from student;

select sno-100,sname,sadd from student;

select sno-100 as SNO,sname,sadd from student;

Interview Queries

Q) Write a query to display all employees information from employee table?

select * from emp;

Q) Write a query to display all tables present in database?

select * from tab;

Q) Write a query to display logical database name?

select * from global_name; // XE

Q) Write a query to display employee id, employee name and employee salary from emp table?

select eid,ename,esal from emp;

Q) Write a query to display employee id,employee name, employee salary and annual salary from emp table?

select eid,ename,esal,esal*12 from emp;

Q) Write a query to display employee id,employee name, employee salary and annual salary as ANNUAL_SAL from emp table?

select eid,ename,esal,esal*12 as ANNUAL_SAL from emp;

or

select eid,ename,esal,esal*12 ANNUAL_SAL from emp;

where clause

It is used to select **specific rows** from database table.

syntax:

---

select * from <table> where condition;

ex:

select * from student where sno=101;

select * from student where sname='ravi';

select * from student where sadd='delhi';

Interview Queries

Q) Write a query to display employee information whose employee id is 203?

select * from emp where eid=203;

Q) Write a query to display employees information those who are working in 10 department?

select * from emp where deptno=10;

Q) Write a query to display employees information those who are working as a Manager?

select * from emp where job='Manager';

Q) Write a query to display employee information whose comm is null?

select * from emp where comm is null;

Q) Write a query to display student information who is living in hyderabad?

select * from student where sadd='hyd';

Q) Write a query to display employees information whose salary is greater then 30000?

select * from emp where esal>30000;

update command

It is used to update the records in a database table.

syntax:

---

update <table_name> set <col_name>=<value> where condition;

ex:

---

update student set sname='rani' where sno=101;

update student set sname='gogo',sadd='AP' where sno=102;

update student set sadd=null where sname='raj';

commit;

Note:

If we are not using where clause then all rows will be updated.

ex:

update student set sname='raja';

Delete command

It is used to delete the records from database table.

syntax:

delete from <table_name> where condition;

ex:

delete from student where sno=105;

delete from student where sname='ravi';

delete from student where sadd='hyd';

commit;

Note:

If we won't use where clause then all rows will be deleted.

ex:

delete from student;

delete from emp;

delete from dept;

Interview Queries

Q) Write a query to increment salary by 1000 whose employee id is 201?

update emp set esal=esal+1000 where eid=201;

Q) Write a query to change the address of a student from hyd to MP?

update student set sadd='MP' where sadd='hyd';

Q) Write a query to terminate the employee whose employee id is 205?

delete from emp where eid=205;

Q) Write a query to terminate all the employees whose salary is greater then 25000?

delete from emp where esal>25000;

Q) Write a query to delete all students whose address is null?

delete from student where sadd is null;

Logical Operators

To declare multiple conditions is where clause we need to use logical operators.

We have three types of logical operators.

1) AND

2) OR

3) NOT

1) AND

It will return the records if all conditions are true.

In logical AND operator , all conditions must be from same row.

ex:

select * from emp where eid=201 AND ename='Alan'; // **record**

select * from emp where eid=201 AND ename='Jose'; **//no rows selected**

select * from emp where eid=209 AND ename='Alan';**//no rows selected**

select * from emp where eid=208 AND ename='gogo'; **//no rows selected**

2) OR

It will return the records only if one condition is true.

In logical OR operator, conditions can be from any row.

ex:

select * from emp where eid=201 OR ename='Alan'; // **1record**

select * from emp where eid=201 OR ename='Jose';//**2 records**

select * from emp where eid=209 OR ename='Alan';// **1 record**

select * from emp where eid=208 OR ename='gogo'; // **no rows selected**

3) NOT

It will return the records except the condition.

A ' < > ' symbol denoted as logical NOT

operator.ex:

select * from emp where eid=201;

select * from emp where NOT eid=201;

select * from emp where eid < > 201;

select * from emp where NOT ename = ' Nancy ';

select * from emp where ename < > ' Nancy ';

Between operator

It will return the records those who are in the range of values.

Between operator takes the support of AND operator.

In between operator we need to take lower limit then higher limit.

ex:

select * from emp where eid between 201 AND 206;

select * from emp where eid between 206 AND 201; // **no rows selected**

select * from emp where deptno between 10 AND 30;

IN operator

IN operator is a replacement of OR operator.

It will return the records those who are matching in the list of values.

ex:

```
select * from emp where eid IN (201,202,209);

select * from emp where ename IN ('Alan','Jose','Lara');
```

Interview Queries

Q) Write a query to display employee information whose employee id is 202 and employee name is Jose?

```
select * from emp where eid=202 AND ename='Jose';
```

Q) Write a query to display employee information whose employee salary is greater then 15000 AND less then 40000?

```
select * from emp where esal>15000 and esal<40000;
```

Q) Write a query to display employees information whose employee id is 201,202 and 203 ?

```
select * from emp where eid IN (201,202,203);
```

Q) Write a query to display employees information those who are not working in 10 department?

```
select * from emp where deptno < > 10;
```

Q) Write a query to delete the records whose employee name is Alan, Jose and Kelvin?

```
delete from emp where ename IN ('Alan','Jose','Kelvin');
```

Pattern Matching operators

Pattern matching operators are used to select the letters from database table.

Pattern matching operators take the support of like keyword.

We have two pattern matching operators.

1) Percentage (%)

2) Underscore (_)

1) Percentage (%)

Q) Write a query to display employees information whose name starts with 'A' letter?

```
select * from emp where ename like 'A%';
```

Q) Write a query to display employees information whose name ends with 'n' letter?

```
select * from emp where ename like '%n';
```

Q) Write query to display employees information whose name having middle letters as 'l'?

```
select * from emp where ename like '%l%';
```

2) Underscore (_)

Q) Write a query to display employees information whose second letter is 'a'?

```
select * from emp where ename like '_a%';
```

Q) Write a query to display employees information whose second last letter is 'r'?

```
select * from emp where ename like '%r_';
```

Q) Write a query to display employees information whose third letter is 'l'?

```
select * from emp where ename like '_l%';
```

DDL commands

1) create - tables
2) alter - columns
3) drop - tables
4) truncate - records/rows
5) rename - tables

2) alter command

Using alter command we can perform following activities very easily.

i) Adding the new columns
ii) Modifying the columns
iii) Dropping the columns
iv) Renaming the columns

i) Adding the new columns

Using alter command we can add new columns in a existing table.

syntax:

alter table <table_name> ADD (col datatype(size));

ex:

alter table student ADD (pincode number(8),state varchar2(10));

update student set pincode=500036 where sno=101;

ii) Modifying the columns

Using alter command we can modify the existing columns.

syntax:

alter table <table_name> modify (col datatype(size));

We can change size of a column only when existing values are fit into new size.

ex:

---

desc student;

alter table student modify (pincode number(10));

desc student;

We can change datatype of a column only if that column is empty.

ex:

--

alter table student modify (state number(10));

iii) Dropping the columns

Using alter command we can remove the existing columns.

syntax:

---

alter table <table_name> DROP (col1,col2,...,colN);

ex:

alter table student DROP (pincode,state);

iv) Renaming the columns

Using alter command we can rename the existing columns.

syntax:

---

alter table <table_name> rename column <old_name> to <new_name>;

ex:

alter table student rename column sadd to city;

alter table emp rename column esal to dailywages;

alter table emp rename column job to designation;

3) drop command

It is used to drop the table from database.

syntax:

drop table <table_name>;

ex:

drop table student;

drop table emp;

drop table dept;

4) truncate command

It is used to delete the records permanently from database.

syntax:

---

truncate table table_name;

ex:

truncate table emp;

truncate table dept;

truncate table student;

Q) What is the difference between delete and truncate command?

delete	**truncate**
----------	----------
It deletes the records temperory. We can rollback the data.	It deletes the records permanently. We can't rollback the data.
Where clause can be used.	Where clause can't be used.
It is a DML command.	It is a DDL command.

5) rename command

It is used to rename the table name.

syntax:

rename <old_name> to <new_name>;

ex:

rename emp to employee;

rename student to students;

rename dept to department;

Duplicate table or copy of a table

Using create and select command we can create a duplicate table or copy of a table.

ex:

create table employee as select * from emp;

create table employee as select eid,ename,esal from emp;

create table employee as select * from emp where deptno=10;

create table employee as select * from emp where deptno<>10;

create table employee as select * from emp where eid(201,202,203);

create table employee as select * from emp where ename like 'A%';

create table employee as select * from emp where esal between 5000 AND 20000;

create table employee as select * from emp where comm is null;

cl scr

It is used to clear the output of SQL command prompt.

ex: cl scr

Functions

Functions are used to manipulate the data items and give result.

We have two types of functions in SQL.

1) Group Functions / Multiple Row Functions

2) Scalar Functions / Single Row Functions

1) Group Functions

Group functions are applicable for multiple rows.

We have following list of group functions.

ex:

sum(), avg(), max(), min(), count(*) and count(express).

Q) Write a query to display sum of salary of each employee?

select sum(esal) from emp;

Q) Write a query to display average salary of each employee?

select avg(esal) from emp;

Q) Write a query to display highest salary from employee table?

select max(esal) from emp;

Q) Write a query to display least salary from employee table?

select min(esal) from emp;

Q) What is the difference between count(*) and count(exp) ?

count(*)

It will return number of records present in a database table.

It will include null records.

ex:

select count(*) from emp; **// 6**

select count(*) from dept; **//4**

count(exp) ?

It will return number of values present in a column.

It will not include null values.

ex:

select count(comm) from emp; // **5**

select count(eid) from emp; // **6**

Dual table

Dual tale is a **dummy table**.

It is used to perform arithmetic operations and to see the **current system date**.

It contains **one row and one column**.

ex:

select 10+20 from dual; // **30**

select 10*20 from dual; // **200**

select sysdate from dual;

select current_date from dual;

2) Scalar Functions

We have four types of scalar functions.

i) character functions

ii) number functions

iii) date functions

iv) conversion functions

i) character functions

upper()

It is used to convert in uppercase.

ex:

select upper('oracle training') from dual;

lower()

It is used to convert in lowercase.

ex:

select lower('ORACLE TRAINING') from dual;

initcap()

It is used to display the string with initial letter capital.

ex:

select initcap('oracle training') from dual;

lpad()

It is used to pad the characters to left side.

ex:

select lpad('oracle',10,'z') from dual; // **zzzzoracle**

rpad()

It is used to pad the characters to right side.

ex:

select rpad('oracle',10,'z') from dual**; // oraclezzzz**

ltrim()

It is used to trim the characters from left side.

ex:

select ltrim('zzoraclezz','z') from dual; // **oraclezz**

rtrim()

It is used to trim the characters from **right side**.

ex:

select rtrim('zzoraclezz','z') from dual; // **zzoracle**

trim()

It is used to trim the characters from both the side.

ex:

select trim('z' from 'zzoraclezz') from dual; // **oracle**

concat()

It is used to **concatinate** two strings.

ex:

select concat('mega','star') from dual; // megastar

select concat(concat('stylish','star'),'Allu') from dual;

ii) number function

abs()

It returns absoluate value.

ex:

select abs(-4) from dual; // **4**

sqrt()

It returns square root value.

ex:

select sqrt(25) from dual;

select sqrt(24) from dual;

power(A,B)

It returns power value.

ex:

select power(5,3) from dual;

ceil()

It returns **ceil value**.

ex:

select ceil(10.1) from dual; // **11**

select ceil(7.8) from dual; // **8**

floor()

It retursn floor value.

ex:

select floor(10.6) from dual; //**10**

select floor(5.2) from dual; // **5**

round()

It will return nearest value.

ex:

select round(10.5) from dual; // **11**

select round(10.4) from dual; // **10**

trunc()

It removes **decimals.**

ex:

select trunc(10.56) from dual; // **10**

select trunc(16.78) from dual; // **16**

greatest()

It returns greatest value.

ex:

select greatest(10,20,30) from dual;

least()

IT returns least value.

ex:

select least(10,20,30) from dual;

Working with Date values

Every database software date values in difference date patterns.

ex:

Oracle - dd-MMM-yy

MySQL - yyyy -MM -dd

emp1 table

drop table emp1;

create table emp1(eid number(3),ename varchar2(10),edoj date);

insert into emp1 values(201,'Alan','01-JAN-24');

insert into emp1 values(202,'Jose',sysdate);

insert into emp1 values(203,'Mark',current_date);

commit;

iii) Date functions

ADD_MONTHS()

It is used to add the months in a given date.

ex:

select ADD_MONTHS('01-JAN-24',5) from dual;

MONTHS_BETWEEN()

It will return number of months between two given dates.

ex:

```
select MONTHS_BETWEEN('01-JAN-24','01-AUG-24') from dual;
select MONTHS_BETWEEN('01-JAN-24','15-AUG-24') from dual;
select ABS(MONTHS_BETWEEN('01-JAN-24','15-AUG-24')) from dual;
```

NEXT_DAY()

It will return the date of a given day in a week.

ex:

```
select NEXT_DAY(sysdate,'sunday') from dual;
select NEXT_DAY(sysdate,'tuesday') from dual;
```

LAST_DAY()

It will return last date of a month.

ex:

```
select LAST_DAY(sysdate) from dual;
select LAST_DAY('25-SEP-24') from dual;
```

iv) Conversion functions

Conversion function is used to convert from datatype to another datatype.

ex:

TO_CHAR()

We have two pseudo for TO_CHAR() function.

a) number to_char()

It supports '9' in digits, dollar and euros symbol.

ex:

```
select eid,ename,esal from emp;
select eid,ename,TO_CHAR(esal,'9,999') from emp;
select eid,ename,TO_CHAR(esal,'99,999') from emp;
select eid,ename,TO_CHAR(esal,'$99,999') from emp;
select eid,ename,TO_CHAR(esal,'$99,999') as ESAL from emp;
```

b)date to_char()

ex:

```
select TO_CHAR(sysdate,'dd-MM-yyyy') from dual;
select TO_CHAR(sysdate,'yyyy-MM-dd') from dual;
select TO_CHAR(sysdate,'year') from dual;
select TO_CHAR(sysdate,'month') from dual;
select TO_CHAR(sysdate,'day') from dual;
select TO_CHAR(sysdate,'yyyy') from dual;
select TO_CHAR(sysdate,'MM') from dual;
select TO_CHAR(sysdate,'dd') from dual;
select TO_CHAR(sysdate,'HH:MI:SS') from dual;
select TO_CHAR(sysdate,'dd-MM-yyyy HH:MI:SS') from dual;
```

Interview queries

Q) Write a query to display employee information from employee table?

```
select eid as id,
        upper(ename) as ename,
            TO_CHAR(esal,'$99,999') as esal,
                deptno,
                  lower(job) as job,
                      comm
                          from emp;
```

userlist table

```
drop table userlist;
create table userlist(uname varchar2(10),pwd varchar2(10));
insert into userlist values('raja','rani');
insert into userlist values('king','kingdom');
commit;
```

Q) Write a query to check username and password is valid or not?

select count(*) from userlist where uname='raja' and pwd='rani';// **1**

select count(*) from userlist where uname='raja' and pwd='rani2'; // **0**

Group by clause

Group by clause is used to divided the rows into multiple groups so that we can apply group functions

A column which we used in group by clause , same column name we need too use in select clause.

ex:

Q) Write query to display sum of salary of each department?

select sum(esal),deptno from emp group by deptno;

Q) Write a query to display minimum salary of each job?

select min(esal),job from emp group by job;

Q) WRite a query to display highest salary of each department?

select max(esal),deptno from emp group by deptno;

Q) Write a query to display average salary of each job?

select avg(esal),job from emp group by job;

Having clause

Having clause is used to filter the rows in group by clause.

Having clause must place after group by clause

Q) Write a query to display sum of salary of each department whose sum of salary is greater then 30000?

select sum(esal),deptno from emp group by deptno having sum(esal)>30000;

Q) Write a query to display minimum salary of each job where minimum salary is greater then equals to 10000?

select min(esal),job from emp group by job having min(esal)>=10000;

Order by clause

Order by clause is used to arrange the rows in a table.

By default it will arrange in ascending order.

ex:

```
select * from emp order by eid;

select * from emp order by eid desc;

select * from emp order by ename;

select * from emp order by ename desc;
```

Q) Write a query to display minimum salary of each job where minimum salary is greater then equals to 10000?

select min(esal),deptno from emp group by deptno having min(esal)>=10000 order by deptno;

select min(esal),deptno from emp group by deptno having min(esal)>=10000 order by deptno desc;

Integrity Constraints

Constraints are the rules which are applied on the tables.

Using constraints we can achieve accuracy and quality of data.

We have following list of constraints in SQL.

1) NOT NULL

2) UNIQUE

3) PRIMARY KEY

4) FOREIGN KEY

5) CHECK

Constraints are created at two levels

i) Column level

ii) Table level

Assignment()

Write a java program to decode the string?

input:

AAJF

output:

1106

1) NOT NULL

NOT NULL constraint does not accept null values.

NOT NULL constraint can access duplicate values.

NOT NULL constraint can be created only at column level.

Column level

```
drop table student;
create table student(sno number(3) NOT NULL,sname varchar2(10),sadd varchar2(12));
insert into student values(101,'raja','hyd');
insert into student values(101,'ravi','delhi');
insert into student values(null,'raja','hyd'); //invalid
```

Note:

NOT NULL constraint can be created for multiple columns.

ex:

```
drop table student;
create table student(sno number(3) NOT NULL,sname varchar2(10) NOT NULL,sadd
varchar2(12) NOT NULL);
insert into student values(101,'raja','hyd');
insert into student values(null,'ravi','delhi');//invalid
insert into student values(102,null,'delhi'); //invalid
insert into student values(102,'ravi',null); // invalid
```

2) UNIQUE

UNIQUE constraint does not accept duplicate values.

UNIQUE constraint accepts null values.

UNIQUE constraint can be created at column level and table level.

Column level

drop table student;

create table student(sno number(3) UNIQUE,sname varchar2(10),sadd varchar2(12));

insert into student values(101,'raja','hyd'); **//valid**

insert into student values(101,'ravi','delhi'); **//invalid**

insert into student values(null,'ravi','delhi'); **//valid**

Table level

drop table student;

create table student(sno number(3),sname varchar2(10),sadd varchar2(12), UNIQUE(sno));

insert into student values(101,'raja','hyd'); **//valid**

insert into student values(101,'ravi','delhi'); **//invalid**

insert into student values(null,'ravi','delhi'); **//valid**

Note:

UNIQUE constraint can be created for multiple columns.

ex:

drop table student;

create table student(sno number(3) UNIQUE,sname varchar2(10) UNIQUE,sadd varchar2(12) UNIQUE);

insert into student values(101,'raja','hyd'); **//valid**

insert into student values(101,'ravi','delhi');**//invalid**

insert into student values(102,'raja','delhi');**//invalid**

insert into student values(102,'raja','hyd'); **// invalid**

3) PRIMARY KEY

Primary key is a combination of NOT NULL and UNIQUE constraint.

Primary key does not accept duplicates and null values.

Primary key can be created at column level and table level.

A table can have only one primary key.

Column level

drop table student;

create table student(sno number(3) PRIMARY KEY,sname varchar2(10),sadd varchar2(12));

insert into student values(101,'raja','hyd'); **//valid**

insert into student values(101,'ravi','delhi');**//invalid**

insert into student values(null,'ravi','delhi');**//invalid**

Table level

drop table student;

create table student(sno number(3),sname varchar2(10),sadd varchar2(12), PRIMARY KEY(sno));

insert into student values(101,'raja','hyd'); **//valid**

insert into student values(101,'ravi','delhi');**//invalid**

insert into student values(null,'ravi','delhi');**//invalid**

4) Foreign Key

It is used to establish the relationship between two tables.

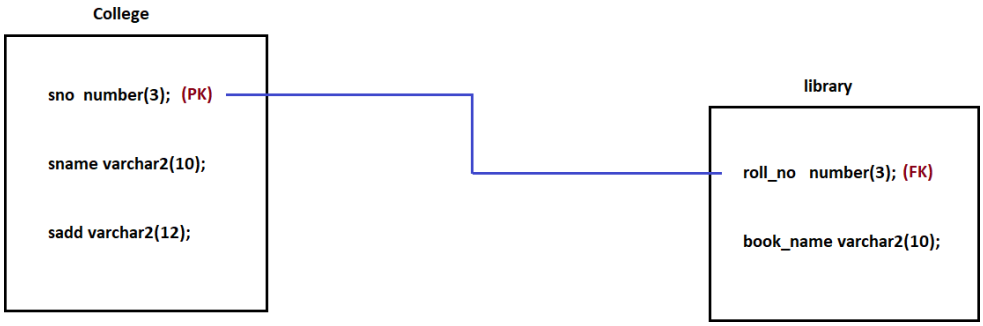
This relationship is called parent and child relationship or master and detailed relationship.

To establish the relationship between two tables. A parent table must have primary key or unique key and child table must have foreign key.

A foreign key will accept only those values which are present in primary key.

A primary key column name or foreign key column name may or not may not match but datatype must match.

Diagram: oracle6.1



College table

```
drop table college;
create table college(sno number(3) primary key, sname varchar2(10),sadd varchar2(12));
insert into college values(101,'raja','hyd');
insert into college values(102,'ravi','delhi');
insert into college values(103,'ramana','vizag');
commit;
```

Library table

```
drop table library;
create table library(roll_no number(3) REFERENCES college(sno),book_name varchar2(10));
insert into library values(101,'java');
insert into library values(102,'oracle');
insert into library values(103,'HTML');
insert into library values(103,'CSS');
insert into library values(null,'spring');
insert into library values(104,'hibernate'); //invalid
```

Note:

If we want to drop the tables then first we need to drop child table then parent table.

ex: drop table library;

drop table college

5) CHECK

CHECK constraint is used to describe domain of a column.

It describes what type of values a column must accept.

CHECK constraint can be created at column level and table level.

Column level

drop table student;

create table student(sno number(3),sname varchar2(10), smarks number(3) check(smarks<=100));

insert into student values(101,'raja',89);

insert into student values(102,'ravi',189); **//invalid**

insert into student values(103,'ramana',879); **//invalid**

commit;

Column level

drop table student;

create table student(sno number(3),sname varchar2(10), smarks number(3) check(smarks between 0 and 100));

insert into student values(101,'raja',89);

insert into student values(102,'ravi',189); **//invalid**

insert into student values(103,'ramana',879); **//invalid**

commit;

Column level

drop table student;

create table student(sno number(3),sname varchar2(10) check(sname=lower(sname)), smarks number(3));

insert into student values(101,'raja',89);

insert into student values(102,'RAVI',189); **//invalid**

insert into student values(103,'RaMaNa',879); **//invalid**

commit;

Column level

drop table student;

create table student(sno number(3),sname varchar2(10) check(sname=upper(sname)), smarks number(3));

insert into student values(101,'raja',89);**//invalid**

insert into student values(102,'RAVI',189);

insert into student values(103,'RaMaNa',879); **//invalid**

commit;

Table level

drop table student;

create table student(sno number(3),sname varchar2(10),smarks number(3), check(sname=upper(sname)));

insert into student values(101,'raja',89);**//invalid**

insert into student values(102,'RAVI',189);

insert into student values(103,'RaMaNa',879); **//invalid**

commit;

Pseudo Colums

Pseudo column means a column which is not real.

We have two pseudo columns.

1) ROWNUM

2) ROWID

1) **ROWNUM**

   It always starts with '1' and increment by '1'.

   ROWNUM values are temperory.

   Once the query is executed we will loss rownum values.

ex:

```
select eid,ename,esal from emp;
select rownum,eid,ename,esal from emp;
select rownum,deptno,dname,dloc from dept;
```

2) ROWID

It is a hexadecimal value.

ROWID is permanent.

ROWID is a memory location where our records will store in a database table.

ex:

```
select rownum,eid,ename,esal from emp;
select rowid,rownum,eid,ename,esal from emp;
```

Interview queries

Q) Write a query to display first three records from employee table?

```
select * from emp where rownum between 1 and 3;
```

or

```
select * from emp where rownum<=3;
```

Q) Write a query to display exact 4th record from employee table?

```
select * from emp where rownum<=4
minus
select * from emp where rownum<=3;
```

Q) Write a query to add the constraint to a existing table?

```
alter table emp add primary key(eid);
```

Q) Write a query to drop the constraint from existing table?

```
alter table emp DROP primary key;
```

Assignment

Q) Write a query to display all the employees information whose names ends with kumar?

employees table

```
drop table employees;
create table employees(eid number(3),ename varchar2(20), esal number(10));
insert into employees values(101,'Alan Morries',10000);
insert into employees values(101,'Raj Kumar',10000);
insert into employees values(101,'Erick Anderson',10000);
insert into employees values(101,'Prem Kumar',10000);
insert into employees values(101,'Lisa Jose',10000);
insert into employees values(101,'Sai Kumar',10000);
insert into employees values(101,'Nancy Pro',10000);
commit;
```

solution :

select * from employees where ename like '%Kumar';

DTCL/TCL Commands

We have three TCL commands.

1) commit

2) rollback

3) savepoint

1) commit

It is used to make the changes permanent to database.

ex:

```
drop table student;
create table student(sno number(3),sname varchar2(10),sadd varchar2(12));
insert into student values(101,'raja','hyd');
insert into student values(102,'ravi','delhi');
commit;
select * from student; // 2 records
```

2) rollback

It is used to undo the changes which are not permanent.

ex:

```
drop table student;
create table student(sno number(3),sname varchar2(10),sadd varchar2(12));
insert into student values(101,'raja','hyd');
insert into student values(102,'ravi','delhi');
commit;
insert into student values(103,'ramana','vizag');
insert into student values(104,'raju','pune');
select * from student; // 4 records
rollback;
select * from student; // 2 records
```

3)savepoint

It is used to mark logical transaction in a database.

Instead of complete rollback we can rollback upto savepoint.

syntax:

```
savepoint <savepoint_name>;
```

ex:

```
drop table student;
create table student(sno number(3),sname varchar2(10),sadd varchar2(12));
insert into student values(101,'raja','hyd');
insert into student values(102,'ravi','delhi');
savepoint sp1;
insert into student values(103,'ramana','vizag');
insert into student values(104,'raju','pune');
savepoint sp2;
insert into student values(105,'Alan','Texas');
insert into student values(106,'Nelson','Miami');
select * from student; // 6 records
rollback to sp2;
select * from student; // 4 record
```

DCL commands

We have two types of DCL commands.

1) grant

2) revoke

Schema

Schema is a memory location which is used to run SQL commands.

Privileges

Permission given to a user is called privileges.

In short, rights given to a user is called privileges.

We have two types of privileges.

1) System privilege

Permission given by DBA to user.

2) Object privilege

Permission given by one user to another user.

1) grant

It is used grant the permissions.

syntax:

grant <privilege1>,<privilege2> to <user_name>;

2) revoke

It is used to revoke the permissions.

syntax:

---

revoke <privilege1>,Privilege2> from <user_name>;

DBA> create user sai identified by sai;

DBA> create user ramya identified by ramya;

SAI> conn sai/sai -- logon denied

RAMYA> conn ramya/ramya --logon denied

DBA> grant connect,resource to sai,ramya;

SAI> conn sai/sai -- connected

RAMYA> conn ramya/ramya -- connected

SAI>

create table employee(eid number(3),ename varchar2(10),esal number(10));

insert into employee values(101,'Alan',1000);

insert into employee values(102,'Jose',2000);

insert into employee values(103,'Mark',3000);

commit;

select * from employee;

RAMYA> select * from sai.employee; -- table or view does not exist

SAI> grant select on employee to ramya;

RAMYA> select * from sai.employee;

RAMYA> delete from sai.employee; --insufficient privileges

SAI> grant delete on employee to ramya;

RAMYA> delete from sai.employee;

RAMYA> commit;

RAMYA> select * from sai.employee; // **no rows selected**

SAI> select * from employee;

SAI> revoke select,delete on employee from ramya;

RAMYA> disc

SAI> disc

DBA> revoke connect,resource from sai,ramya;

RAMYA> conn ramya/ramya --logon denied

SAI> conn sai/sai -- logon denied

Sequence

Sequence is an object which is used to generate the numbers.

syntax:

create sequence <sequence_name> start with value increment by value;

ex:

create sequence sq1 start with 1 increment by 1;

create sequence sq1 start with 101 increment by 1;

create sequence sq1 start with 10 increment by 10;

We have two pseudo's of a sequence.

1) NEXTVAL

It is used to generate next number in a sequence.

ex:

```
create sequence sq1 start with 101 increment by 1;
drop table student;
create table student(sno number(3),sname varchar2(10),sadd varchar2(12));
insert into student values(sq1.NEXTVAL,'raja','hyd');
insert into student values(sq1.NEXTVAL,'ravi','delhi');
insert into student values(sq1.NEXTVAL,'ramana','vizag');
commit;
```

2) CURRVAL

It returns the last number generated by the sequence.

ex:

```
select sq1.currval from dual;
```

Interview Queries

Q) Write a query to see the list of sequences present in database?

select sequence_name from user_sequences;

Q) Write a query to drop the sequence?

drop sequence sq1;

Q) Write a query to the list of users present in database?

select username from all_users;

Q) Write a query to drop the user from database?

drop user sai CASCADE;

drop user ramya cascade;

Assignment

Q) Write a query to display last three records from employee?

Indexes

Index is an object which is used to improve the performance of select command.

Index in a database is similar to index in a book.

We create index only to those columns which are widely used in where clause.

Whenever we create index , two columns will be generated. One is ROWID and second is indexed column and all the records will store in ascending order inside indexed column.

ex:

indexed table

ROWID	INDEXED column
	201
	202
	203
	204
	205
	206

Index are divided into two types.

1) Simple index

2) Complex index

1) Simple index

If index is created only for one column is called simple index.

syntax:

create index <index_name> on <table_name>(col_name);

ex:

create index idx1 on emp(eid);

Here index is used when we are using eid in where clause.

select * from emp where eid=206;

2) Complex index

If a index is created for multiple columns is called complex index.

syntax:

create index <index_name> on <table_name(col1,col2,...,colN);

ex:

create index idx2 on emp(eid,deptno);

Here index is used when we are using eid and deptno in where clause.

ex:

select * from emp where eid=201 AND deptno=10;

Interview Queries

Q) Write a query to see the list of indexes present in database?

select index_name from user_indexes;

Q) Write a query to drop the index?

drop index idx1;

drop index idx2;

Synonyms

Alternate name given to an object(table) is called synonym.

We can use synonym name instead of table name for all commands.

Using synonym length of the table will reduce.

syntax:

create synonym <synonym_name> for <object_name>;

ex:

create synonym sy1 for student;

select * from sy1;

delete from sy1;

Q) Write a query to see the list of synonyms present in database?

select synonym_name from user_synonyms;

Q) Write a query to drop the synonym?

drop synonym sy1;

Joins

select * from emp; **//6 records**

select * from dept; **//4 records**

select * from emp,dept; **// 6*4=24 records**

select eid,ename,deptno,dname,dloc from emp,dept; **//column ambiguously defined**

select emp.eid,emp.ename,emp.deptno,dept.dname,dept.dloc from emp,dept;

table alias

A user defined heading given to a table is called table alias.

Table alis is temperory.

We can use table alias in select clause as well as in where clause.

ex:

select e.eid,e.ename,e.deptno,d.dname,d.dloc from emp e,dept d;

Definition:

Joins is used to retrieve the data from one or more then one table.

We have following list of joins.

1) Equi join

2) Non-Equi join

3) Self join

4) Cartesian product

5) Inner join

6) Outer join

1) Equi join

When two tables are joined based on common column is called equi join.

ex:

```
select e.eid,e.ename,e.esal,d.dname,d.dloc from emp e,dept d
where(e.deptno=d.deptno);
```

2) Non-Equi join

If two tables are not joined by using equi join operator then we need to use non-equi join.

ex:

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc from emp e,dept d
where esal between 25000 AND 40000; // 2 * 4
```

3) Self join

If a table is joined to itself is called self join.

In self join we need to create two table alias for same table.

ex:

```
select e1.eid,e1.ename,e1.esal,e2.job,e2.comm from emp e1,emp e2
where(e1.deptno=e2.deptno); // 6 + 6 = 12 records
```

4) Cartisian Product

It returns all the possible combinations.

In catisian product we will not use where caluse.

ex:

```
select e.eid,e.ename,e.esal,d.dname,d.dloc from emp e,dept d; // 6 * 4  = 24 records
```

5) Inner join

It is similar to equi join.

It is given by ANSI people.

American National Standard Institute.

ex:

```
select e.eid,e.ename,e.esal,d.dname,d.dloc from emp e INNER JOIN dept d
ON(e.deptno=d.deptno);
or
select e.eid,e.ename,e.esal,d.dname,d.dloc from emp e JOIN dept d
ON(e.deptno=d.deptno);
```

6) Outer join

It is a extension of equi join.

A '+' symbol denoted as outer join operator.

It will return matching as well as not matching records.

We have three types of outer joins

1) Left outer join

SQL

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc from emp e,dept d
where(e.deptno=d.deptno(+));
```

ANSI

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc
from emp e LEFT OUTER JOIN dept d
ON(e.deptno=d.deptno);
```

2) Right outer join

SQL

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc from emp e,dept d
where(e.deptno(+)=d.deptno);
```

ANSI

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc
from emp e RIGHT OUTER JOIN dept d
ON(e.deptno=d.deptno);
```

3) Full outer join

ANSI

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc
from emp e FULL OUTER JOIN dept d
ON(e.deptno=d.deptno);
```

Views

It is a virtual representation of a data from one or more then one table.

A table which is used to create a view is called base table or above table.

syntax:

create view <view_name> as select stmt;

ex:

```
create view v1 as select * from emp;
```

View does not consumes the memory.

View does not have data.

View will get the data from base table when we execute select command.

We have following list of views.

1) Simple view

2) Complex view

3) With read only view

4) With check option view

5) Materialized view

1) Simple view

If a view is created by using one base table is called simple view.

DML operations are allowed in simple view.

ex:

```
create view v1 as select * from student;
select * from v1; // 3 records
delete from v1; // 3 records deleted
select * from student; //no rows selected
```

2) Complex view

If a view is created by using more then one base table is called complex view.

In complex view DML operations are not allowed.

ex:

```
create view v2 as select e.eid,e.ename,e.esal,d.dname,d.dloc from emp e,dept d
                where (e.deptno=d.deptno);
select * from v2;
delete from v2; //cannot delete from view
```

3) With read only view

If we create a view by using one base table and DML operation are not allowed then we need to use with read only view.

ex:

```
create view v3 as select * from emp with read only;
select * from v3;
delete from v3; //cannot delete from view
```

4) With check option view

If we create a view by using one base table and DML operations are allowed only when condition is true then we need to use with check option view.

ex:

```
create view v4 as select * from emp where deptno=30 with check option;
select * from v4;
```

insert into v4 values(209,'Julie',26000,70,'Salesman',500); // **view WITH CHECK OPTION**

insert into v4 values(209,'Julie',26000,30,'Salesman',500); **//inserted**

5) Materialized view

To create a materialized view a table must have primary key.

ex:

alter table emp add primary key(eid);

create materialized view v5 as select * from emp;

select * from v5**; // 9 records**

select * from emp; // **9 records**

delete from emp where eid IN (208,209); // **2 records deleted**

select * from v5; // **9 records**

select * from emp; // **7 records**

To get updated data from base table we need to refresh the view.

ex:

exec DBMS_SNAPSHOT.REFRESH('v5');

DBMS_SNAPSHOT is a package name

REFRESH is a procedure name

select * from v5; // **7 records**

select * from emp; **//7 records**

Q) Write a query to see the list of views present in database?

select view_name from user_views;

Q) Write a query to drop the view.

drop view v1;

drop view v2;

drop view v3;

drop view v4;

drop materialized view v5;

Merge command

Merge command is a combination of insert and update command.

student10 table

```
drop table student10;
create table student10(sno number(3),sname varchar2(10),sadd varchar2(12));
insert into student10 values(101,'raja','hyd');
insert into student10 values(102,'ravi','delhi');
insert into student10 values(103,'ramana','vizag');
commit;
```

student20 table

```
drop table student20;
create table student20(sno number(3),sname varchar2(10),sadd varchar2(12));
insert into student20 values(103,'Alan','Texas');
insert into student20 values(104,'Nelson','Vegas');
commit;
    merge into student10 s1
    using student20 s2
    ON(s1.sno=s2.sno)
    when matched then
    update set sname=s2.sname,sadd=s2.sadd
    when not matched then
    insert (sno,sname,sadd) values(s2.sno,s2.sname,s2.sadd);
    select * from student10; // 2 rows merged
    select * from student20;
```

Sub Queries

If we declare a query inside another query such concept is called sub query.

In sub queries , first inner query will execute then outer/parent query.

Sub queries are used to select the records based on unknown values.

we have following list of sub queries.

1) Single Row Sub query

2) Multiple Row Sub query

3) Multiple Column Sub query

1) Single Row Sub query

If a sub query returns only one row is called single row sub query.

Sub query can be nested upto 32 levels.

ex:

SQL Query

```
select * from emp where eid=201;
```

Sub Query

```
select * from emp where eid=(select eid from emp where ename='Alan');
```

ex:

SQL Query

```
select * from emp where eid=201       and ename='Alan';
```

Sub Query

```
select * from emp where
eid=(select eid from emp where esal=9000)
and
ename=(select ename from emp where eid=201);
```

Q) Write a query to display employees information whose salary is greater then Mark salary?

```
select * from emp where esal>(select esal from emp where ename='Mark');
```

Q) Write a query to display second highest salary from emp table?

```
select max(esal) from emp where esal<(select max(esal) from emp);
```

2) Multiple row sub query

If sub query returns more then one row is called multiple row sub query.

To perform multiple row sub query we need to use multiple row operators.

We have three multiple row operators.

i) ANY

ii) ALL

iii) IN

i) ANY

select * from emp where esal > ANY (select esal from emp where deptno=10);

select * from emp where esal < ANY (select esal from emp where deptno=10);

2) ALL

select * from emp where esal > ALL (select esal from emp where deptno=10);

select * from emp where esal < ALL (select esal from emp where deptno=20);

3) IN

select * from emp where esal IN (select esal from emp where deptno=10);

4) Multiple column sub query

If a sub query returns more then one column is called multiple column sub query.

In multiple column sub query we need to use IN operator.

ex:

select * from emp where(eid,ename,esal) IN (select eid,ename,esal from emp where eid=201);

select eid,ename,esal from emp where(eid,ename,esal) IN (select eid,ename,esal from emp where eid=201);

select eid,ename,esal from emp where(eid,ename,esal) IN (select eid,ename,esal from emp where deptno=10);

Q) What is the difference between RDBMS and MongoDB?

RDBMS	MongoDB
It is a relational database.	It is a non-relational or document based database
It can't store the data in key and value pair.	It can stores the data in key and value pair.
Not suitable for hierarchical data storage.	Suitable for hierarchical data storage.
It has a static schema.	It has a dynamic schema.
It contains tables.	It contains collections.
It is row based.	It is document based.
It is column based.	It is field based.
It is slower.	It is faster.
It supports SQL query language.	It supports JSON query language.

PL/SQL

PL/SQL stands for Procedural Language extension to Structured Query Language.

PL/SQL is a extension of SQL and contains following features.

1) We can achieve programming features like control statements, loops and etc.

2) It reduces network traffic.

3) We can display custom error messages by using the concept of exception handling.

4) We can perform related operations by using the concept of triggers.

5) It saves the source code permanent to database for repeated execution.

PL/SQL block

A PL/SQL program is also known as PL/SQL block.

syntax:

```
DECLARE
-
-       -- Declaration section
-
BEGIN
-
-       -- Executable section
-
EXCEPTION
-
-       -- Exception section
-
END;
/
```

Declaration section

Declaration section is used to declare variables, cursors, exceptions and etc.

It is optional section.

Executable section

Execution section contains actual code of a program.

It is mandatory section.

EXCEPTION section

It contains lines of code which is executed when exception is raised.

It is optional section.

To see the output in PL/SQL we need on server output environment.

ex:

SQL> set serveroutput on

Q) Write a PL/SQL program to display Hello World?

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Hello World');
END;
/
```

Here DBMS_OUTPUT is a package name.

Here PUT_LINE is a procedure name.

Here '/' is used to submit the PL/SQL program into database.

Q) Write a PL/SQL program to perform sum of two numbers?

```
DECLARE
A number;
B number;
C number;
BEGIN
A:=10;
```

```
B:=20;
C:=A+B;
DBMS_OUTPUT.PUT_LINE(C);
END;
/
```

Declaration and Initialization using single line

```
DECLARE
A number:=10;
B number:=20;
C number:=A+B;
BEGIN
DBMS_OUTPUT.PUT_LINE(C);
END;
/
```

To read dynamic values we need to use '&' symbol.

ex:

```
DECLARE
A number(3);
B number(3);
C number(6);
BEGIN
A:=&a;
B:=&b;
C:=A+B;
DBMS_OUTPUT.PUT_LINE('Sum of two numbers is ='||C);
END;
/
```

Using PL/SQL we can perform DML operations.

Q) Write a PL/SQL program to insert a record into student table?

```
DECLARE
L_sno number(3);
L_sname varchar2(10);
L_sadd varchar2(12);
BEGIN
L_sno:=&sno;
L_sname:='&sname';
L_sadd:='&sadd';
insert into student values(L_sno,L_sname,L_sadd);
DBMS_OUTPUT.PUT_LINE('Please check the table');
END;
/
```

Q) Write a PL/SQL program to update student name based on student number?

```
DECLARE
L_sname varchar2(10);
L_sno number(3);
BEGIN
L_sname:='&sname';
L_sno:=&sno;
update student set sname=L_sname where sno=L_sno;
DBMS_OUTPUT.PUT_LINE('Please check the table');
END;
/
```

Assignment

Q) Write a PL/SQL program to delete student record based on student number?

Indexes

=======

Index is an object which is used to improve the performance of select command.

Index in a database is similar to index in a book.

We create index only to those columns which are widely used in where clause.

Whenever we create index , two columns will be generated. One is ROWID and second is indexed column and all the records will store in ascending order inside indexed column.

ex:

indexed table

ROWID	INDEXED column
	201
	202
	203
	204
	205
	206

Index are divided into two types.

1) Simple index

2) Complex index

1) Simple index

If index is created only for one column is called simple index.

syntax:

create index <index_name> on <table_name>(col_name);

ex:

create index idx1 on emp(eid);

Here index is used when we are using eid in where clause.

select * from emp where eid=206;

2) Complex index

If a index is created for multiple columns is called complex index.

syntax:

create index <index_name> on <table_name(col1,col2,...,colN);

ex:

create index idx2 on emp(eid,deptno);

Here index is used when we are using eid and deptno in where clause.

ex:

select * from emp where eid=201 AND deptno=10;

Interview Queries

Q) Write a query to see the list of indexes present in database?

select index_name from user_indexes;

Q) Write a query to drop the index?

drop index idx1;

drop index idx2;

Synonyms

=========

Alternate name given to an object(table) is called synonym.

We can use synonym name instead of table name for all commands.

Using synonym length of the table will reduce.

syntax:

---

create synonym <synonym_name> for <object_name>;

ex:

create synonym sy1 for student;

select * from sy1;

delete from sy1;

Q) Write a query to see the list of synonyms present in database?

select synonym_name from user_synonyms;

Q) Write a query to drop the synonym?

drop synonym sy1;

Joins

======

select * from emp; // **6 records**

select * from dept; // **4 records**

select * from emp,dept; // **6*4=24 records**

select eid,ename,deptno,dname,dloc from emp,dept; //**column ambiguously defined**

select emp.eid,emp.ename,emp.deptno,dept.dname,dept.dloc from emp,dept;

Table alias

A user defined heading given to a table is called table alias.

Table alis is temperory.

We can use table alias in select clause as well as in where clause.

ex:

select e.eid,e.ename,e.deptno,d.dname,d.dloc from emp e,dept d;

Definition:

Joins is used to retrieve the data from one or more then one table.

We have following list of joins.

1) Equi join

2) Non-Equi join

3) Self join

4) Cartisian Product

5) Inner join

6) Outer join

1) Equi join

When two tables are joined based on common column is called equi join.

ex:

```
select e.eid,e.ename,e.esal,d.dname,d.dloc from emp e,dept d
where(e.deptno=d.deptno);
```

2) Non-Equi join

If two tables are not joined by using equi join operator then we need to use non-equi join.

ex:

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc from emp e,dept d
where esal between 25000 AND 40000; // 2 * 4
```

3) Self join

If a table is joined to itself is called self join.

In self join we need to create two table alias for same table.

ex:

```
select e1.eid,e1.ename,e1.esal,e2.job,e2.comm from emp e1,emp e2
where(e1.deptno=e2.deptno); // 6 + 6 = 12 records
```

4) Cartisian Product

It returns all the possible combinations.

In catisian product we will not use where caluse.

ex:

```
select e.eid,e.ename,e.esal,d.dname,d.dloc from emp e,dept d; // 6 * 4  = 24 records
```

5) Inner join

It is similar to equi join.

It is given by ANSI people.

American National Standard Institute.

ex

```
select e.eid,e.ename,e.esal,d.dname,d.dloc from emp e INNER JOIN dept d
ON(e.deptno=d.deptno);
or
select e.eid,e.ename,e.esal,d.dname,d.dloc from emp e JOIN dept d
ON(e.deptno=d.deptno);
```

6) Outer join

It is a extension of equi join.

A '+' symbol denoted as outer join operator.

It will return matching as well as not matching records.

We have three types of outer joins.

1) Left outer join

SQL

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc from emp e,dept d
where(e.deptno=d.deptno(+));
```

ANSI

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc
from emp e LEFT OUTER JOIN dept d
ON(e.deptno=d.deptno);
```

2) Right outer join

SQL

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc from emp e,dept d
where(e.deptno(+)=d.deptno);
```

ANSI

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc
from emp e RIGHT OUTER JOIN dept d
ON(e.deptno=d.deptno);
```

3) Full outer join

ANSI

```
select e.eid,e.ename,e.esal,e.deptno,d.deptno,d.dname,d.dloc
from emp e FULL OUTER JOIN dept d
ON(e.deptno=d.deptno);
```

Views

======

It is a virtual representation of a data from one or more then one table.

A table which is used to create a view is called base table or above table.

syntax:

```
create view <view_name> as select stmt;
```

ex:

```
create view v1 as select * from emp;
```

View does not consumes the memory.

View does not have data.

View will get the data from base table when we execute select command.

We have following list of views.

1) Simple view

2) Complex view

3) With read only view

4) With check option view

5) Materialized view

1) Simple view

If a view is created by using one base table is called simple view.

DML operations are allowed in simple view.

ex:

```
create view v1 as select * from student;
select  * from v1; // 3 records
delete from v1; // 3 records deleted
select * from student; // no rows selected
```

2) Complex view

If a view is created by using more then one base table is called complex view.

In complex view DML operations are not allowed.

ex:

```
create view v2 as select e.eid,e.ename,e.esal,d.dname,d.dloc from emp e,dept d
                where (e.deptno=d.deptno);
select * from v2;
delete from v2; //cannot delete from view
```

3) With read only view

If we create a view by using one base table and DML operation are not allowed then we need to use with read only view.

ex:

create view v3 as select * from emp with read only;

select * from v3;

delete from v3; **//cannot delete from view**

4) With check option view

If we create a view by using one base table and DML operations are allowed only when condition is true then we need to use with check option view.

ex:

create view v4 as select * from emp where deptno=30 with check option;

select * from v4;

insert into v4 values(209,'Julie',26000,70,'Salesman',500); **// view WITH CHECK OPTION**

insert into v4 values(209,'Julie',26000,30,'Salesman',500); **//inserted**

5) Materialized view

To create a materialized view a table must have primary key.

ex:

alter table emp add primary key(eid);

create materialized view v5 as select * from emp;

select * from v5; **// 9 records**

select * from emp; **// 9 records**

delete from emp where eid IN (208,209); **// 2 records deleted**

select * from v5; **// 9 records**

select * from emp; **// 7 records**

To get updated data from base table we need to refresh the view.

ex:

exec DBMS_SNAPSHOT.REFRESH('v5');

DBMS_SNAPSHOT is a package name

REFRESH is a procedure name

select * from v5; // 7 records

select * from emp; // 7 records

Q) Write a query to see the list of views present in database?

```
select view_name from user_views;
```

Q) Write a query to drop the view.

```
drop view v1;
drop view v2;
drop view v3;
drop view v4;
drop materialized view v5;
```

Merge command

==========

Merge command is a combination of insert and update command.

student10 table

```
drop table student10;
create table student10(sno number(3),sname varchar2(10),sadd varchar2(12));
insert into student10 values(101,'raja','hyd');
insert into student10 values(102,'ravi','delhi');
insert into student10 values(103,'ramana','vizag');
commit;
```

student20 table

```
drop table student20;
create table student20(sno number(3),sname varchar2(10),sadd varchar2(12));
insert into student20 values(103,'Alan','Texas');
insert into student20 values(104,'Nelson','Vegas');
commit;
```

```
merge into student10 s1
using student20 s2
ON(s1.sno=s2.sno)
when matched then
update set sname=s2.sname,sadd=s2.sadd
when not matched then
insert (sno,sname,sadd) values(s2.sno,s2.sname,s2.sadd);

select * from student10; // 2 rows merged
select * from student20;
```

Sub Queries

==========

If we declare a query inside another query such concept is called sub query.

In sub queries , first inner query will execute then outer/parent query.

Sub queries are used to select the records based on unknown values.

we have following list of sub queries.

1) Single Row Sub query

2) Multiple Row Sub query

3) Multiple Column Sub query

1) Single Row Sub query

If a sub query returns only one row is called single row sub query.

Sub query can be nested upto 32 levels.

ex:

SQL Query

```
select * from emp where eid=201;
```

Sub Query

```
select * from emp where eid=(select eid from emp where ename='Alan');
```

ex:

SQL Query

```
select * from emp where eid=201      and ename='Alan';
```

Sub Query

```
select * from emp where
eid=(select eid from emp where esal=9000)
and
ename=(select ename from emp where eid=201);
```

Q) Write a query to display employees information whose salary is greater then Mark salary?

select * from emp where esal>(select esal from emp where ename='Mark');

Q) Write a query to display second highest salary from emp table?

select max(esal) from emp where esal<(select max(esal) from emp);

2) Multiple row sub query

==========

If sub query returns more then one row is called multiple row sub query.

To perform multiple row sub query we need to use multiple row operators.

We have three multiple row operators.

i) ANY

ii) ALL

iii) IN

i) ANY

```
select * from emp where esal > ANY (select esal from emp where deptno=10);
select * from emp where esal < ANY (select esal from emp where deptno=10);
```

ii) ALL

```
select * from emp where esal > ALL (select esal from emp where deptno=10);
select * from emp where esal < ALL (select esal from emp where deptno=20);
```

iii) IN

```
select * from emp where esal IN (select esal from emp where deptno=10);
```

3) Multiple column sub query

==========

If a sub query returns more then one column is called multiple column sub query.

In multiple column sub query we need to use IN operator.

ex:

```
select * from emp where(eid,ename,esal) IN (select eid,ename,esal from emp
where   eid=201);
select eid,ename,esal from emp where(eid,ename,esal) IN (select eid,ename,esal
from emp  where eid=201);
select eid,ename,esal from emp where(eid,ename,esal) IN (select eid,ename,esal
 from  emp where deptno=10);
```

Q) What is the difference between **RDBMS** and **MongoDB**?

RDBMS	**MongoDB**
It is a relational database.	It is a non-relational or document based database
It can't store the data in key and value pair.	It can stores the data in key and value pair.
Not suitable for hierarchical data storage.	Suitable for hierarchical data storage.
It has a static schema.	It has a dynamic schema.
It contains tables.	It contains collections.
It is row based.	It is document based.
It is column based.	It is field based.
It is slower.	It is faster.
It supports SQL query language.	It supports JSON query language.

PL/SQL

======

PL/SQL stands for Procedural Language extension to Structured Query Language.

PL/SQL is a extension of SQL and contains following features.

1) We can achieve programming features like control statements, loops and etc.

2) It reduces network traffic.

3) We can display custom error messages by using the concept of exception handling.

4) We can perform related operations by using the concept of triggers.

5) It saves the source code permanent to database for repeated execution.

PL/SQL block

==========

A PL/SQL program is also known as PL/SQL block.

syntax:

DECLARE

-

- -- Declaration section

-

BEGIN

-

- -- Executable section

-

EXCEPTION

-

- -- Exception section

-

END;

/

Declaration section

Declaration section is used to declare variables, cursors, exceptions and etc.

It is optional section.

Executable section

Execution section contains actual code of a program.

It is mandatory section.

EXCEPTION section

It contains lines of code which is executed when exception is raised.

It is optional section.

To see the output in PL/SQL we need on server output environment.

ex:

```
SQL> set   serveroutput   on
```

Q) Write a PL/SQL program to display Hello World?

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Hello World');
END;
/
```

Here DBMS_OUTPUT is a package name.

Here PUT_LINE is a procedure name.

Here '/' is used to submit the PL/SQL program into database.

Q) Write a PL/SQL program to perform sum of two numbers?

```
DECLARE
A number;
B number;
C number;
BEGIN
A:=10;
B:=20;
C:=A+B;
DBMS_OUTPUT.PUT_LINE(C);
END;
/
```

Declaration and Initialization using single line

```
DECLARE
A number:=10;
B number:=20;
C number:=A+B;
BEGIN
DBMS_OUTPUT.PUT_LINE(C);
END;
/
```

To read dynamic values we need to use '&' symbol.

ex:

```
DECLARE
A number(3);
B number(3);
C number(6);
BEGIN
A:=&a;
B:=&b;
C:=A+B;
DBMS_OUTPUT.PUT_LINE('Sum of two numbers is ='||C);
END;
/
```

Using PL/SQL we can perform DML operations.

Q) Write a PL/SQL program to insert a record into student table?

```
DECLARE
L_sno  number(3);
L_sname varchar2(10);
L_sadd  varchar2(12);
BEGIN
```

```
L_sno:=&sno;
L_sname:='&sname';
L_sadd:='&sadd';
insert into student values(L_sno,L_sname,L_sadd);
DBMS_OUTPUT.PUT_LINE('Please check the table');
END;
/
```

Q) Write a PL/SQL program to update student name based on student number?

```
DECLARE
L_sname varchar2(10);
L_sno number(3);
BEGIN
L_sname:='&sname';
L_sno:=&sno;
update student set sname=L_sname where sno=L_sno;
DBMS_OUTPUT.PUT_LINE('Please check the table');
END;
/
```

Assignment

==========

Q) Write a PL/SQL program to delete student record based on student number?

To see the output in PL/SQL we need to execute below command.

ex:

```
SQL> set  serveroutput  on
```

In PL/SQL we can perform DRL operations.

To perform DRL operations in PL/SQL we need to use "into" clause.

Q) Write a PL/SQL program to display employe name whose employee id is 201?

```
DECLARE
L_ename  varchar2(10);
BEGIN
select ename into L_ename from emp where eid=201;
DBMS_OUTPUT.PUT_LINE(L_ename);
END;
/
```

Q) Write a PL/SQL program to display employee name, employee salary whose employee id is 202?

```
DECLARE
L_ename varchar2(10);
L_esal  number(10,2);
BEGIN
select ename,esal into L_ename,L_esal from emp where eid=202;
DBMS_OUTPUT.PUT_LINE(L_ename||' '||L_esal);
END;
/
```

Q) Write a PL/SQL program to display employee name, employee salary based on employee id?

```
DECLARE
L_ename varchar2(10);
L_esal  number(10,2);
L_eid   number(3);
BEGIN
L_eid:=&eid;
select ename,esal into L_ename,L_esal from emp where eid=L_eid;
DBMS_OUTPUT.PUT_LINE(L_ename||' '||L_esal);
END;
/
```

Percentage(%) TYPE attribute

==========

It is used to declare a local variable with respect to column type.

syntax:

<variable_name> <table_name>.<column_name>%TYPE;

ex:

L_job emp.job%TYPE;

Q) Write a PL/SQL program to display employee name, employee salary and department number from employee table based on employee id ?

```
DECLARE
L_ename  emp.ename%TYPE;
L_esal   emp.esal%TYPE;
L_deptno emp.deptno%TYPE;
L_eid    emp.eid%TYPE;
BEGIN
L_eid:=&eid;
select ename,esal,deptno into L_ename,L_esal,L_deptno from emp where eid=L_eid;
DBMS_OUTPUT.PUT_LINE(L_ename||' '||L_esal||' '||L_deptno);
END;
/
```

Percentage(%) ROWTYPE attribute

==========

It is used to declare a local variable which holds complete row of a table.

syntax:

<variable_name> <table_name>%ROWTYPE;

ROWTYPE variable we can't print directly.

Inorder to get the values from ROWTYPE we need to use variable_name.column_name.

ex:

Q) Write a PL/SQL program to display employee information whose employee id is 205?

```
DECLARE
A  emp%ROWTYPE;
BEGIN
select * into A from emp where eid=205;
DBMS_OUTPUT.PUT_LINE(A.eid||' '||A.ename||' '||A.esal||' '||A.deptno||' '||A.job||' '||A.comm);
END;
/
```

ex:

Q) Write a PL/SQL program to display employee information based on employee id?

```
DECLARE
A emp%ROWTYPE;
L_eid emp.eid%TYPE;
BEGIN
L_eid:=&eid;
select * into A from emp where eid=L_eid;
DBMS_OUTPUT.PUT_LINE(A.eid||' '||A.ename||' '||A.esal||' '||A.deptno||' '||A.job||' '||A.comm);
END;
/
```

Control Statements

==========

We have three control statements in PL/SQL.

1) IF THEN

2) IF THEN ELSE

3) IF THEN ELSIF THEN ELSE

1) IF STMT

It evaluates the code only if condition is true.

ex:

```
DECLARE
A number:=7;
BEGIN
IF A>5 THEN
DBMS_OUTPUT.PUT_LINE('Greatest');
END IF;
END;
/
```

ex:

```
DECLARE
A number:=2;
BEGIN
IF A>5 THEN
DBMS_OUTPUT.PUT_LINE('Greatest');
END IF;
END;
/
```

2) IF THEN ELSE

It evaluates the code either our condition is true or false.

ex:

```
DECLARE
A number:=7;
BEGIN
IF A>5 THEN
DBMS_OUTPUT.PUT_LINE('Greatest');
ELSE
DBMS_OUTPUT.PUT_LINE('Least');
END IF;
END;
/
```

ex:

```
DECLARE
A number:=2;
BEGIN
IF A>5 THEN
DBMS_OUTPUT.PUT_LINE('Greatest');
ELSE
DBMS_OUTPUT.PUT_LINE('Least');
END IF;
END;
/
```

3) IF THEN ELSIF THEN ELSE

It evalutes the code based on multiple conditions.

ex:

--

```
DECLARE
N  number;
BEGIN
N:=&option;
IF N=100 THEN
DBMS_OUTPUT.PUT_LINE('It is police number');
ELSIF N=103 THEN
DBMS_OUTPUT.PUT_LINE('It is enquiry number');
ELSIF N=108 THEN
DBMS_OUTPUT.PUT_LINE('It is emergency number');
ELSE
DBMS_OUTPUT.PUT_LINE('Invalid option');
END IF;
END;
/
```

LOOPS

======

We have three types of LOOPS in PL/SQL.

1) Simple LOOP

2) While LOOP

3) For LOOP

1) Simple LOOP

It evaluates the code how long our condition is true.

ex:

```
DECLARE
A number:=1;
BEGIN
DBMS_OUTPUT.PUT_LINE('Welcome');
LOOP
DBMS_OUTPUT.PUT_LINE('Hello');
EXIT WHEN A=4;
A:=A+1;
END LOOP;
DBMS_OUTPUT.PUT_LINE('Thankyou');
END;
/
```

ex:

```
DECLARE
A number:=1;
BEGIN
LOOP
DBMS_OUTPUT.PUT_LINE(A);
EXIT WHEN A=10;
A:=A+1;
END LOOP;
END;
/
```

2) While LOOP

It evaluates the code how long our condition is true.

ex:

```
DECLARE
A number:=1;
BEGIN
DBMS_OUTPUT.PUT_LINE('Welcome');
WHILE A<=4 LOOP
DBMS_OUTPUT.PUT_LINE('Hello');
A:=A+1;
END LOOP;
DBMS_OUTPUT.PUT_LINE('Thankyou');
END;
/
```

ex:

```
DECLARE
A number:=1;
N number:=5;
BEGIN
WHILE A<=10 LOOP
DBMS_OUTPUT.PUT_LINE(N||' * '||A||' = '||N*A);
A:=A+1;
END LOOP;
END;
/
```

3) For LOOP

It evalutes the code how long our condition is true.

ex:

```
DECLARE
A number;
BEGIN
DBMS_OUTPUT.PUT_LINE('Welcome');
FOR A IN 1 .. 4 LOOP
DBMS_OUTPUT.PUT_LINE('Hello');
END LOOP;
DBMS_OUTPUT.PUT_LINE('Thankyou');
END;
/
```

Interview Question

==========

Q) Write a java program to display below loop pattern?

4 4 4 4 4 4 4

4 3 3 3 3 3 4

4 3 2 2 2 3 4

4 3 2 1 2 3 4

4 3 2 2 2 3 4

4 3 3 3 3 3 4

4 4 4 4 4 4 4

ANS:-

```
class Test
{
	public static void main(String[] args)
	{
		int n=4;

		int size=7;

		for(int i=0;i<size;i++)
		{
			for(int j=0;j<size;j++)
			{
				int value= n - Math.min(Math.min(i,j),Math.min(size-i-1,size-j-
1));
				System.out.print(value+" ");
			}
			//new line
			System.out.println();
		}
	}
}
```

Q) Write a java program to reverse of a given number using recursion?

input:

	123

output:

	321

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int num=123;
		System.out.println(reverse(num));
	}
	public static int reverse(int num)
	{
		return reverseNumber(num,0);
	}
	public static int reverseNumber(int n,int reversed)
	{
		if(n==0)
		{
			return reversed;
		}
		int reminder = n % 10;

		reversed = (reversed * 10) + reminder;

		return reverseNumber(n/10,reversed);
	}
}
```

Exceptions

==========

Runtime errors are called exceptions.

We have two types of exceptions.

1) Predefined exceptions

2) Userdefined exceptions

1) Predefined exceptions

Built-in exceptions are called predefined exceptions.

We have following list of predefined exceptions.

i) NO_DATA_FOUND exception

ii) TOO_MANY_ROWS exception

iii) ZERO_DIVIDE exception

iv) VALUE_ERROR exception

v) DUP_VAL_ON_INDEX exception

vi) OTHERS

i) NO_DATA_FOUND exception

This exception will occur when select statement does not return any row.

ex:

```
DECLARE
L_ename emp.ename%TYPE;
BEGIN
select ename into L_ename from emp where eid=209;
DBMS_OUTPUT.PUT_LINE(L_ename);
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Please check employee id ');
END;
/
```

ii) TOO_MANY_ROWS exception

This exception will raise when select statement returns more then one row.

ex:

```
DECLARE
L_ename emp.ename%TYPE;
BEGIN
select ename into L_ename from emp where deptno=10;
DBMS_OUTPUT.PUT_LINE(L_ename);
EXCEPTION
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE('select stmt returns more then one row ');
END;
/
```

iii) ZERO_DIVIDE exception

This exception will raise when we divide any number with zero.

ex:

```
DECLARE
A number;
BEGIN
A:=10/0;
DBMS_OUTPUT.PUT_LINE(A);
EXCEPTION
WHEN ZERO_DIVIDE THEN
DBMS_OUTPUT.PUT_LINE('Dont divide by zero');
END;
/
```

iv) VALUE_ERROR exception

This exception will raise when there is a mismatch with datatype or size.

ex:

```
DECLARE
L_esal  emp.esal%TYPE;
BEGIN
select ename into L_esal from emp where eid=201;
DBMS_OUTPUT.PUT_LINE(L_esal);
EXCEPTION
WHEN VALUE_ERROR THEN
DBMS_OUTPUT.PUT_LINE('Please check datatype ');
END;
/
```

ex:

```
DECLARE
A number(3);
BEGIN
A:=12345;
DBMS_OUTPUT.PUT_LINE(A);
EXCEPTION
WHEN VALUE_ERROR THEN
DBMS_OUTPUT.PUT_LINE('Please check the size');
END;
/
```

v) DUP_VAL_ON_INDEX exception

This exception will raise when we are trying to insert duplicate value in a primary key.

ex:

```
alter table student add primary key(sno);
BEGIN
insert into student values(103,'ramulu','pune');
DBMS_OUTPUT.PUT_LINE('Record Inserted');
EXCEPTION
WHEN DUP_VAL_ON_INDEX then
DBMS_OUTPUT.PUT_LINE('No Duplicates Allowed');
END;
/
```

vi) OTHERS

It is a universal angular exception which can handle all types of exceptions.

ex:

```
DECLARE
L_ename emp.ename%TYPE;
BEGIN
select ename into L_ename from emp where eid=209;
DBMS_OUTPUT.PUT_LINE(L_ename);
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Please check employee id ');
END;
/
```

2) Userdefined exceptions

Exceptions which are created by the user based on the requirement are called userdefined exceptions.

steps to develop userdefined exceptions

---

step1:

Declare the exception

step2:

Raise the exception

step3:

Handle the exception

ex:

```
DECLARE
MY_EX1 EXCEPTION;
SAL number:=5000;
BEGIN

IF SAL>2000 THEN
RAISE MY_EX1;
END IF;
DBMS_OUTPUT.PUT_LINE(SAL);

EXCEPTION
WHEN MY_EX1 THEN
DBMS_OUTPUT.PUT_LINE('Salary is Too High');
END;
/
```

Cursors

=======

Cursor is a PL/SQL block which is used to run select commands.

We have two types of cursors.

1) Implicit cursor

2) Explicit cursor

1) Implicit cursor

All the activities related to cursor like opening the cursor, processing the cursor and closing the cursor which is done automatically is called implicit cursor.

We have types of implicit cursor attributes.

i) SQL%ISOPEN

It is a boolean attribute which always returns false.

ii) SQL%FOUND

It is a boolean attribute which returns true if SQL is success and returns false

is SQL command is failed.

iii) SQL%NOTFOUND

It is completely inverse of SQL%FOUND.

It is a boolean attribute which returns true if SQL is failed and returns false

is SQL command is success.

iv) SQL%ROWCOUNT

It will return number of records effected in a database table.

i) SQL%ISOPEN

```
BEGIN
IF  SQL%FOUND THEN
DBMS_OUTPUT.PUT_LINE('Cursor is opened');
ELSE
DBMS_OUTPUT.PUT_LINE('Cursor is closed');
END IF;
END;
/
```

ii) SQL%FOUND

```
BEGIN
update student set sname='raja' where sno=101;
IF SQL%FOUND THEN
DBMS_OUTPUT.PUT_LINE('Record Updated');
ELSE
DBMS_OUTPUT.PUT_LINE('Record Not Updated');
END IF;
END;
/
```

iii) SQL%NOTFOUND

```
BEGIN
update student set sname='raja' where sno=109;
IF SQL%NOTFOUND THEN
DBMS_OUTPUT.PUT_LINE('Record Updated');
ELSE
DBMS_OUTPUT.PUT_LINE('Record Not Updated');
END IF;
END;
/
```

iv) SQL%ROWCOUNT

```
BEGIN
update student set sname='raja';
DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT||' Records Updated');
END;
/
```

2) Explicit cursor

All the activities related to cursor like opening the cursor, processing the cursor and closing the cursor is done by the programmer is called explicit cursor.

We need to use explicit cursor when select statement returns more then one row.

We have following list of explicit cursor attributes.

i) %ISOPEN

It returns true if cursor is open and returns false if cursor is closed.

ii) %FOUND

It returns true if SQL command is success and returns false is SQL command is failed.

iii) %NOTFOUND

It returns false if SQL command is success and returns true is SQL command is failed.

iv) %ROWCOUNT

It returns number of records effected in a database table.

Steps to create a explicit cursor

----

step1:

Declare a cursor

step2:

Open the cursor

step3:

Fetch the data from cursor to local variables.

step4:

Close the cursor

Q) Write a PL/SQL program to display employee names from employee table?

```
DECLARE
CURSOR C1 is select ename from emp;
L_ename emp.ename%TYPE;
BEGIN
OPEN C1;
LOOP
FETCH C1 into L_ename;
EXIT WHEN C1%NOTFOUND;
DBMS_OUTPUT.PUT_LINE(L_ename);
END LOOP;
CLOSE C1;
END;
/
```

Q) Write a PL/SQL program to display employee id,employee name and employee salary from emp table?

```
DECLARE
CURSOR C2 is select eid,ename,esal from emp;
L_eid emp.eid%TYPE;
L_ename emp.ename%TYPE;
L_esal emp.esal%TYPE;
BEGIN
OPEN C2;
LOOP
FETCH C2 into L_eid,L_ename,L_esal;
EXIT WHEN C2%NOTFOUND;
DBMS_OUTPUT.PUT_LINE(L_eid||' '||L_ename||' '||L_esal);
END LOOP;
CLOSE C2;
END;
/
```

Q) Write a PL/SQL program to display employees information from emp table?

```
DECLARE
CURSOR C3 is select * from emp;
A emp%ROWTYPE;
BEGIN
OPEN C3;
LOOP
FETCH C3 into A;
EXIT WHEN C3%NOTFOUND;
DBMS_OUTPUT.PUT_LINE(A.eid||' '||A.ename||' '||A.esal||' '||A.deptno||' '||A.job);
END LOOP;
CLOSE C3;
END;
/
```

Procedure

=========

It is a named PL/SQL block which compiled and stored permanently to database for repeated execution.

syntax:

```
create or replace procedure <procedure_name>
is
begin
-
-
-
end;
/
```

Q) Write a procedure to display Hello World?

```
create or replace procedure p1
is
begin
DBMS_OUTPUT.PUT_LINE('Hello World');
END;
/
```

To execute above procedure we need to use below command.

ex:

```
exec  p1;
```

To see the output in PL/SQL we need to use below command.

ex:

```
SQL> set  serveroutput  on
```

Procedure may contains three parameters.

1) IN parameter

2) OUT parameter

3) IN OUT parameter

1) IN parameter

It is used to accept the values from the user.

Q) Write a procedure to perform sum of two numbers?

```
create or replace procedure sum(A IN number,B IN number)
is
C number;
begin
C:=A+B;
DBMS_OUTPUT.PUT_LINE(C);
END;
/
```

We can call above procedure by using below command.

ex:

```
exec   sum(10,20);
```

Using procedure we can perform DML operations.

Q) Write a procedure to delete student record based on student number?

```
create or replace procedure delete_student(L_sno IN student.sno%TYPE)
is
begin
delete from student where sno=L_sno;
DBMS_OUTPUT.PUT_LINE('Record Deleted');
END;
/
```

We can call above procedure by using below command.

ex:

```
exec  delete_student(102);
```

2) OUT parameter

It is used to return the values to the user.

Q) Write a procedure to perform sum of two numbers and return sum?

```
create or replace procedure ret_sum(A IN number,B IN number,C OUT number)
is
begin
C:=A+B;
END;
/
```

Steps to invoke the procedure having OUT parameter

---

step1:

Declare bind variable

ex:

```
variable N number;
```

step2:

Execute the procedure.

ex:

```
exec  ret_sum(10,20,:N);
```

step3:

Print the bind variable.

ex:

```
print N;
```

3) IN OUT parameter

It is used to accept and return the values from/to user.

Q) Write a procedure to display square of a given number?

```
create or replace procedure ret_square(A IN OUT number)
is
begin
A:=A*A;
END;
/
```

Steps to invoke the procedure having IN OUT parameter

---

step1:

Declare a bind variable.

ex:

```
variable N number;
```

step2:

Initialize the bind variable.

ex:

```
BEGIN
:N:=5;
END;
/
```

step3:

Execute the procedure.

ex:

```
exec ret_square(:N);
```

step4:

Print the bind variable.

ex:

print N;

Q) Write a query to see the list of procedures present in database?

select object_name from user_objects where object_type='PROCEDURE';

Q) Write a query to see the source code of a procedure?

select text from user_source where name='P1';

Q) Write a query to drop the procedure?

drop procedure p1;

drop procedure sum;

drop procedure ret_sum;

drop procedure ret_square;

Functions

=======

Function is a named PL/SQL block which must and should returns a value.

syntax:

---

create or replace function <function_name>

return datatype

is

begin

-

-

-

return datatype;

end;

/

Q) Write a function to perform sum of two numbers and return sum?

```
create or replace function f1(A number,B number)
return number
is
C number;
begin
C:=A+B;
return C;
END;
/
```

We can invoke above function by using below command.

ex:

```
select f1(10,50) from dual;
```

Q) Write a function to accept one salary and find out 10% of TDS?

```
create or replace function f2(salary number)
return number
is
tax number;
begin
tax:=salary*10/100;
return tax;
END;
/
```

We can invoke above function by using below command.

ex:

```
select f2(1000) from dual;
select eid,ename,esal,f2(esal) from emp;
select eid,ename,esal,f2(esal) AS TAX from emp;
```

Q) What is the difference between procedures and functions?

Procedures	**Functions**
----------	----------
Procedure may or may not returns a value.	Function always returns a value.
DML operations are allowed.	DML operations are not allowed.
Can't be invoked by using select command.	Can be invoked by using select command.

Q) Write a query to see the list of functions present in database?

```
select object_name from user_objects where object_type='FUNCTION';
```

Q) Write a query to see the source code of a function?

```
select text from user_source where name='F1';
```

Q) Write a query to drop the function?

```
drop function f1;
drop function f2;
```

Package

=======

A package is a collection of logical related sub programs.

PL/SQL procedures and functions are called logical related sub programs.

Package creation involved in two steps.

1) Package specification

It is a declaration of logical related sub programs.

2) Package body

It contains definition of logical related sun programs.

ex:1

---

package specification

```
create or replace package pkg1
is
procedure p1(A IN number,B IN number);
end pkg1;
/
```

package body

```
create or replace package body pkg1
is
procedure p1(A IN number,B IN number)
is
C number;
begin
C:=A+B;
DBMS_OUTPUT.PUT_LINE(C);
END;
end pkg1;
/
```

We can invoke above package by using below command.

ex:

```
exec  pkg1.p1(10,30);
```

ex:2

---

package specification

```
create or replace package pkg2
is
function f1(A number,B number)
return number;
end pkg2;
/
```

package body

```
create or replace package body pkg2
is
function f1(A number,B number)
return number
is
C number;
begin
C:=A+B;
return C;
END;
end pkg2;
/
```

We can invoke above pakage by using below command.

ex:

```
select pkg2.f1(40,60) from dual;
```

Q) Write a query to see the list of packages present in database?

```
select object_name from user_objects where object_type='PACKAGE';
```

Q) Write a query to see the source code of a package?

```
select text from user_source where name='PKG1';
```

Q) Write a query to drop the package?

```
drop package body pkg1;
drop package pkg1;
```

Triggers

=======

Trigger is a PL/SQL block which executes based on events.

Trigger events are insert, update and delete.

Triggers timings before, after and insteadof.

syntax:

```
create or replace trigger <trigger_name> <timing> <event> on <object_name>
begin
-
-
END;
/
```

ex:

```
create or replace trigger trg1 before insert on student
begin
DBMS_OUTPUT.PUT_LINE('Thank you for inserting');
END;
/
```

```
select * from student;
insert into student values(102,'ravi','delhi'); // trigger executed
```

Trigger can be created on multiple events also.

ex:

```
create or replace trigger trg2 after insert or update or delete on student
begin
IF inserting THEN
DBMS_OUTPUT.PUT_LINE('Thanks for inserting');
ELSIF updating then
DBMS_OUTPUT.PUT_LINE('Thanks for updating');
ELSE
DBMS_OUTPUT.PUT_LINE('Thanks for deleting');
END IF;
END;
/
insert into student values(104,'ramana','vizag'); // trigger executed
delete from student where sno=103; // trigger executed
```

Triggers are divided into two types.

1) Statement level trigger

If a trigger executes only for one time irrespective of number of records effected in a database table is caleld statement level trigger.

By default every trigger is a statement level trigger.

ex:

```
create or replace trigger trg3 before delete on student
begin
DBMS_OUTPUT.PUT_LINE('Thanks for delete');
END;
/
```

delete from student ; // trigger will execute one time

2) Row level trigger

In row level trigger , our trigger will execute multiple times based on number of records effected in a datase table.

To create row level trigger we need to use "FOR EACH ROW" clause.

ex:

```
create or replace trigger trg3 before delete on student FOR EACH ROW
begin
DBMS_OUTPUT.PUT_LINE('Thanks for delete');
END;
/
delete from student; // Trigger will execute three times
```

Q) Write a query to see the list of triggers present in database?

```
select object_name from user_objects where object_type='TRIGGER';
```

Q) Write a query to see the source code of a trigger?

```
select text from user_source where name='TRG1';
```

Q) Write a query to drop the trigger?

```
drop trigger trg1;
drop trigger trg2;
drop trigger trg3;
```