client

ata inside

rML)

structure
can add,

nto memory

t, it will
g event

RDBMS

Data : Unstructured / unprocessed / unmeaningful / ~~unstructured~~ unformatted / raw Data which is not organised in proper manner

Information : the data which is organised in proper manner

Meaningful Data : Processed Data / structured Data is called Information

→ Data is generated by using business activities, e-commerce website, social media platforms

Database :

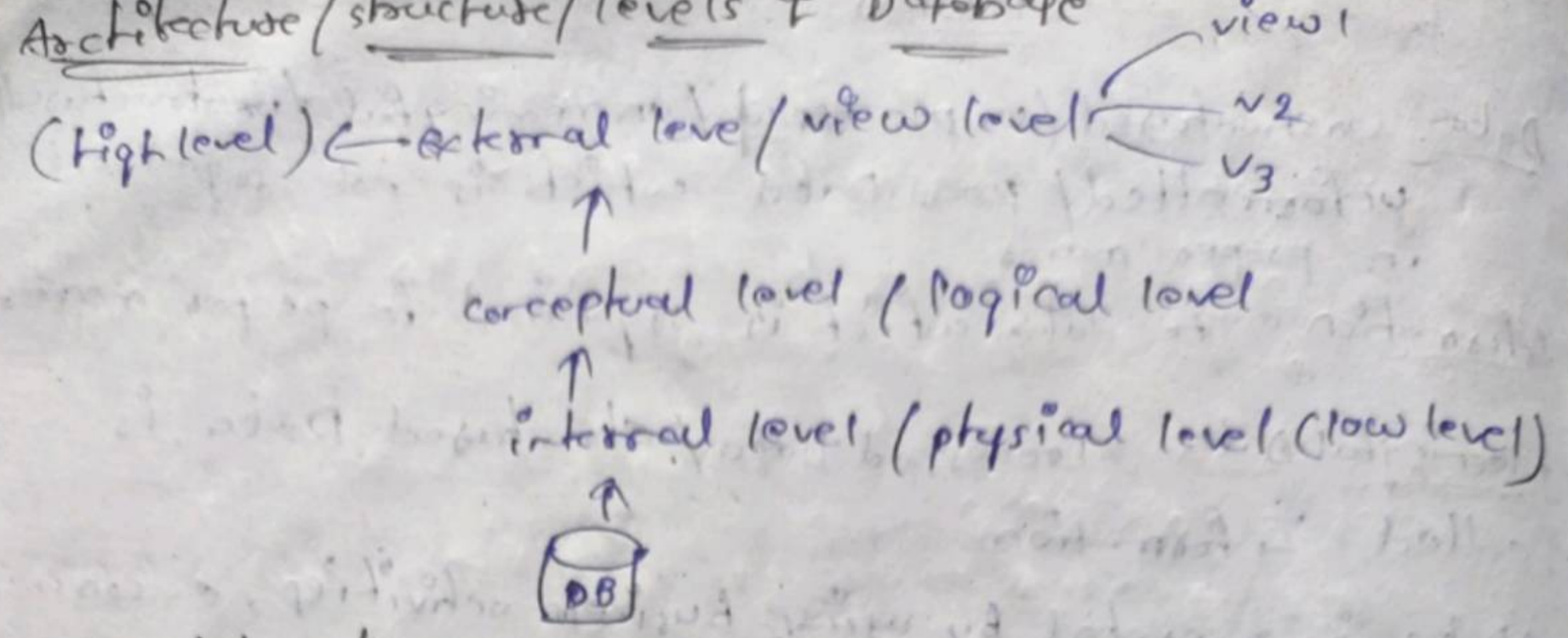
→ Database is a inter-related data.
→ It is used to store, access, process, manipulate the data
→ Advantages

1) It can store huge amount of data
2) security → only authorised users or access
3) Data Redundacy → Duplicates values are not allowed
4) Data consistency → only valid data will be entered by using constraints
5) Data Integrity → we can combine the data from the multiple resources & form a single data set
6) Backup & Restore
7) Data sharing
8) Data Independence → the changes made in upper level will not effect in lower level.

Disadvatages

→ cost
→ Maintenance

Architecture / structure / levels of Database



Data Abstraction:

→ Hiding the irrelevant details from the user, how the complex data has been stored

→ In Data Abstraction we have 3 levels:

1) physical / Internal level
2) logical / conceptual level
3) External / view level.

Physical level

→ It is the lowest level in Data Abstraction

~~PDBMS~~ ~~syllabus~~

→ this level defines how the data is stored in the memory.

→ It defines the access methods like sequential, random access

→ It defines file organization methods like B+ tree, hashing, hierarchy.

→ It defines the memory utilization

Conceptual / logical level

→ In this level, we define the information that is actually stored inside the database in the form of 2 Dimensional tables.

→ In this level, we define the attributes & their relationships

→ If we establish relationship b/w multiple tables atleast one field should be common

External/ view level
→ It is the highest level of Data Abstraction
→ In this level, we view only some part of the data in the database
→ by using data abstraction we can achieve data independence

Data Independency

view level
↑ ⟶ logical level data independence
logical level
↑ ⟶ physical level data independence
physical level

→ It is the ability to modify schema at one level without altering schema at next higher level

physical level Data Independence
→ It will separate physical level with logical level
→ If we do changes in the storage size of the DB it is not going to reflect conceptual level
→ physical level Datat independence can occur at logical interface level.

logical level Data independence
→ It will seperate logical level & view level
→ It occurs at user interface level.

Data Model
→ How the data is represented at the conceptual level is called Data Model

Types

1) Hieoretical Data Model	8) web enabled DM
2) Network DM	
3) Relational DM	
4) Entity Relational DM	
5) object oriented DM	
6) object relational DM	
7) Cwarehouse DM	

→ microsoft → MS sql server
→ oracle → oracle
→ IBM → Dbe
→ sqlite DB is used for mobile device } sql Databases
→ postg DB
→ sybase DB
→ Mongo DB (Textual DB)

Hierarchical Data model

→ It is the oldest & traditional Data model
→ It is introduced by IBM in 1950's
→ It is also called as parent-child data model
→ It organizes the data in tree-like structure
→ Each record consist of one parent record & many childs
→ this data model works on "one-to many relationships"
→ this data model can't have duplicate child records
→ In 1960 IBM introduced IMS (Information Management System) product which works on procedural language

Network Data Model

→ In 1970's CODASYL (Conference of Data systems language) community has introduced network model
→ It is the extension of Hierarchical DM
→ by using this DM we can access the data fast
→ It is implemented by using "many to many Relationships"
→ IBM has introduced IDMS (Info. Data management system) product which works on procedural language

Relational DM

→ Edgar F codd has introduced Relational DM
→ It is the latest DM implemented in every DB
→ this DM: defins the data in the form of rows & columns
→ E F codd invented sql to work with this model
→ It consists of 3 components:
1) objects like tables, views, triggers, sequence, indexes, cursors, synonyms

2) set of operators
3) Integrity Rules-

→ MYSQL 8.1 version → ORDBMS
→ MYSQL 5.5 version → RDBMS

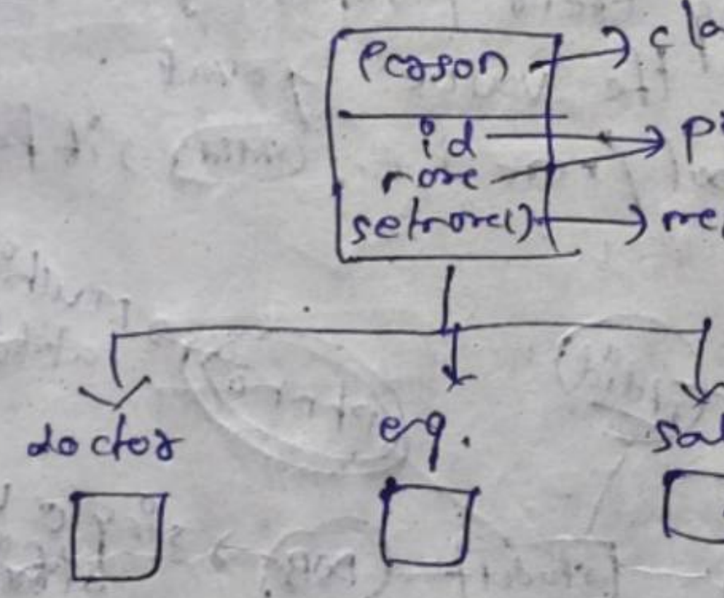
object-Relational DM

→ It is the combination of object oriented pg ming language & Relational DM.

object : It is an instance of a class
→ object encapsulates the data & code into the single unit
→ these are real world entity

Attributes : It defines properties of an object like std name, std roll no, std name etc.

Method : It represents behaviour of an object



ER Data Model (Entity Relationship DM)

→ It defines different types of entities & relations b/w different types of entities
→ ER diagram is a logical structure of the DB
→ ER diagram is created based on 3 concepts :-
1) Entity
2) Attributes
3) Relationships

Entity : Entity is a person, place, object, event which is used to store the data in the DB
→ Entity is represented by using Rectangular box with name

Types

1) storg Entity
2) weak Entity.

storg Entity

→ the Entity which is identified by unique identifier (primary key) Ex: student

weak Entity

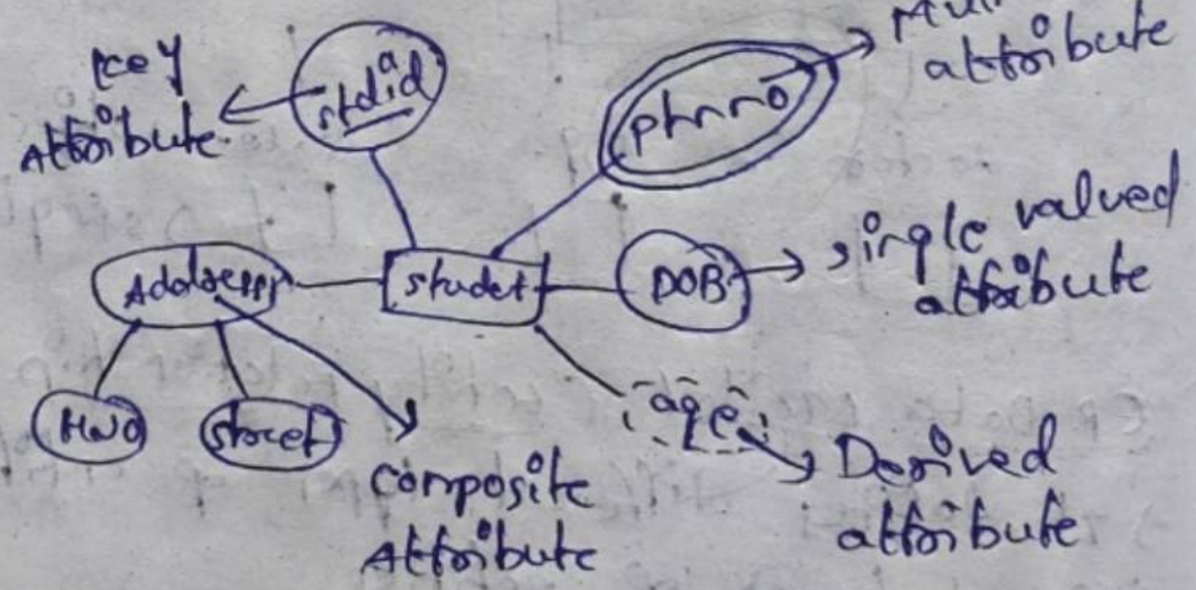
→ it depends upon the storg entity for existence i.e., it has foriegn key to another entity.
→ it is represented by double Rectangular box.

Entity set

→ it is a collection of similar type of ~~elem~~ entities
Ex: student → it has similar type of entities like stdid, stdname etc

Attributes:

→ Entity is denoted by utilizing their properties
→ Attributes are used to hold the value
→ it is denoted by the symbol / name i.e., stdid → it has some name



Types of Attributes

1) key Attribute
2) composite Att
3) single value Att
4) Multi value Att
5) Derived Att

1) key Attribute:

→ it is an attribute which uniquely identifies the entity among the entity set
Ex: stdid, rollno

~~of composite Attribute~~

Types of keys:

1) super key
2) candidate key
3) primary key

identifier

existence
ity.
box. [roa]

entities
like std id,

ity
val
id) → it has

→ multi valued attribute

→ single valued attribute

→ Derived attribute

the

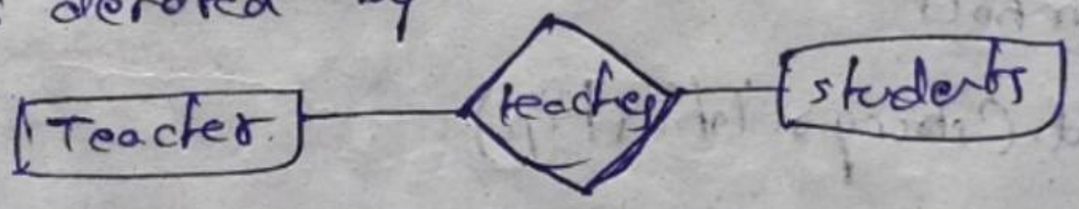
2) composite Attribute:
→ It is an attribute that is a combination of other attributes
ex: Address → it has other attributes like HNO, street, city

3) single valued Attribute:
→ An Attribute which has only one value
ex: uid, PAN no

4) Multi valued Attribute:
→ An Attribute which has multiple values
ex: phn no, email
→ It is denoted by double eclipse. (◎)

5) Derived Attribute:
→ An attribute which doesn't physically existence inside the DB & it is derived from another attribute
→ It is denoted by the symbol i.e., dotted eclipse
ex: age is derived from DOB

Relation
→ Association b/w the entities
→ It is denoted by Diamond shaped box (Rhombus

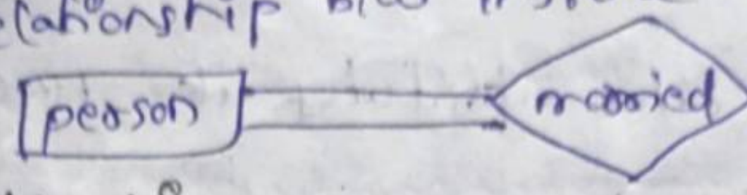


Relationship set
→ It is a collection of similar types of relationships
→ Attributes for the Relationship is called as "Descriptive Attributes".

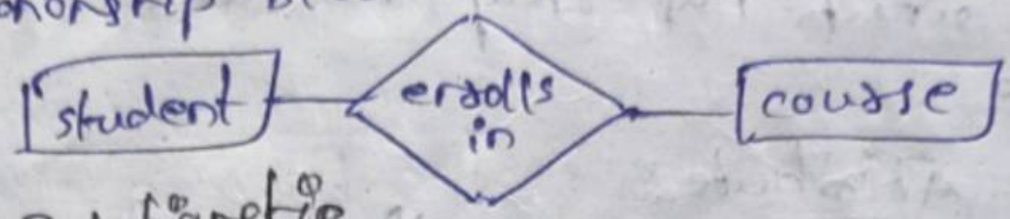
Degree of Relationship
→ No. of participating entities in a relationship is called degree of Relationship.

Types:
1) Unary (degree = 1)
2) Binary (degree = 2)
3) Ternary (degree = 3)

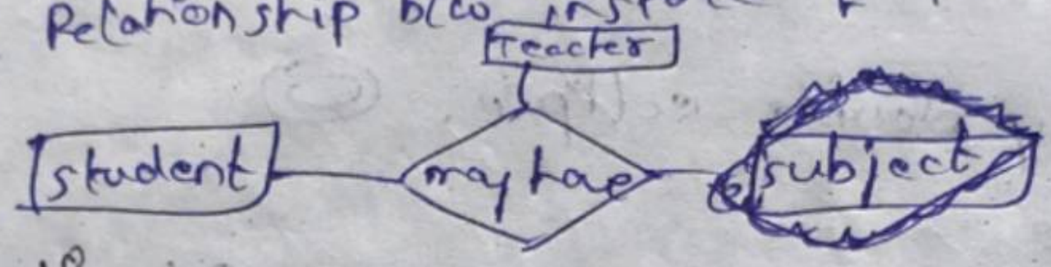
1) Unary Relationship
→ It is also called as Recursive Relationship
→ It is a relationship b/w instance of one entity



2) Binary Relationship.
→ the Relationship b/w instance of two entities.

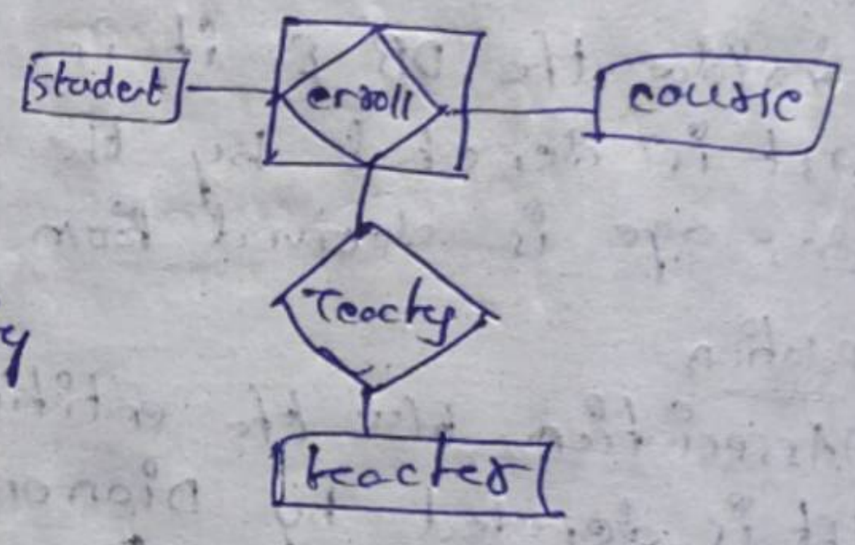


3) Ternary Relationship
→ the Relationship b/w instance of three entities



ER diagram symbols

1) Entity symbols
   1) → ▭
   2) → ▭ → weak entity
   3) → ◇ → Associative entity



2) Relationship symbols
   1) → ◇ → diamond. (strong relationship)
   2) → ◇ → weak relationship

3) Attribute symbols
   1) → ⬭ → single valued Att.
   2) → ⬭ → Multi valued Att.
   3) → ⬭ → Derived Att.
   4) → ⬭ → key Att
   5) → ⬭ → weak key Att

Normalization

It is the process of minimizing / reducing & removing is called Normalization

Normal Forms:

It is the process of removing duplicate values

we have 6 types of Normalization forms:

1) 1NF
2) 2NF
3) 3NF
4) BCF
5) 4NF
6) 5NF

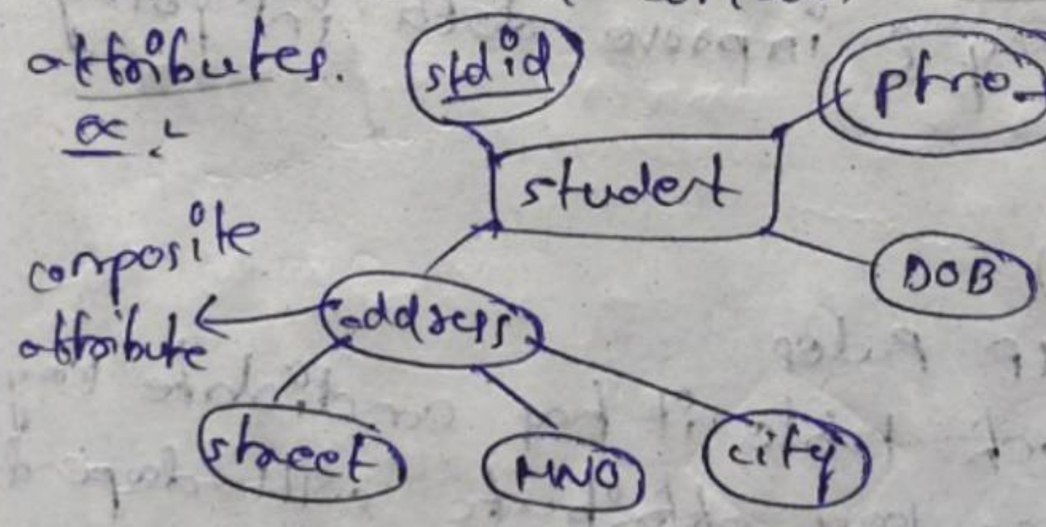
stdid	stdname	credits	dept	building code
1	aa	5	CSE	100
2	bb	10	CSE	100
3	cc	15	ECE	200
4	dd	6	ECE	200
X			MEC	300
5	ee		OEE	400

1NF:

→ Each row in a table should have a primary key & it should have single value

→ 1NF should not contain multiple values & composite attributes.

ex:-



→ It is not in 1NF

→ Each cell should contain single value

→ column should contain same type of data

→ column names should be unique

→ no ordering of rows & columns

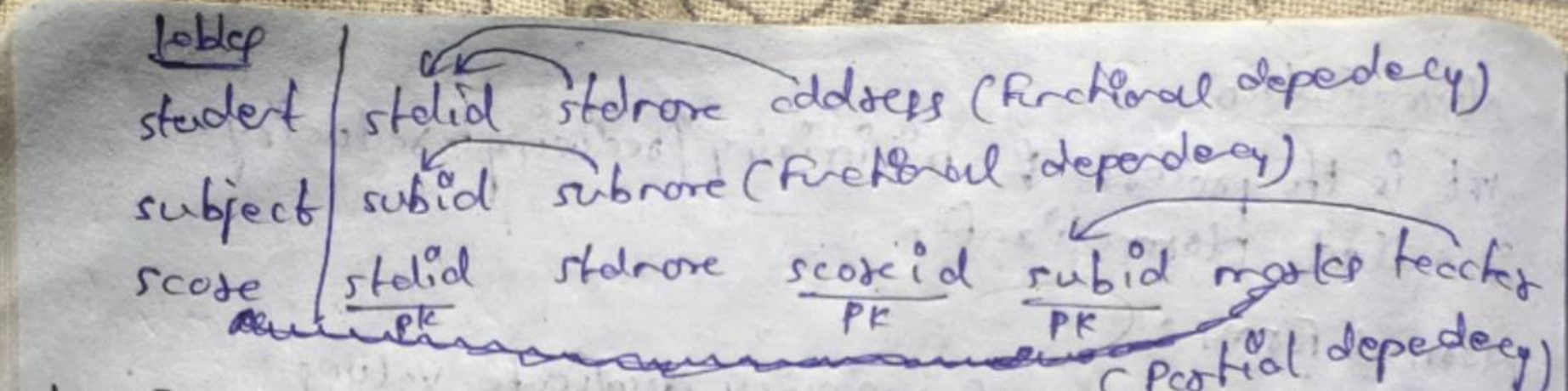
2NF

→ It should satisfies 1NF Rules

→ No partial dependency in the relation

functional dependency → all the columns will depend on one primary key

partial dependencies → when these are one/more primary keys in a table.



3) 3NF :
→ It should follows 1NF & 2NF Rules
→ No transitive dependency
→ Transitive depedency means one column will depend upon one column but it is not a primary key

4) 4NF :
→ It is the extension of BCF
→ In this form, it should not have multivalued depedencies

5) 5NF :
→ It is the highest level of Normalization
→ In this form, the table is divided into small tables to remove data redundancy & improve data integrity problem

6) BCF (Boyce Codd NF)
→ It should follows 3NF Rules
→ It determines in each table it has candidate key i.e., it ensures each non-key attribute will depend upon candidate key
→ It is named by Person F Boyce & E F Codd
→ It is the advanced version of 3NF
→ It is also called 3.5 NF
→ It should be the 3NF for any dependency A → B
A should be the super key
A ↓ super key

Keys :
→ By using keys we can uniquely identifies the tuple (rows) in a table
→ By using keys we can setup relations b/w columns & tables in a DB

al depedecy)
decy)

d marks teacher
partial depedecy)

ill depend upon
y.

lved depedencies

small tables
a integrity

ndidate key
will depend

→ B
er key

tuple (rows)

w columns {

Types of keys

1) Super key
2) primary key
3) foreign key
4) Cardidate key
5) Composite key
6) Alternate key
7) Unique key

super key: It is an attribute or set of attributes that uniquely identify a table or tuple

→ A single table can have multiple super keys
→ Candidate key, Alternate key, Primary key can be a super key but vice versa is not possible.
→ It will allow null values
→ create table std(stdid number(10) PK)
Alter table std Add key Drop keyname
→ 1st table:
create table patientdetails (patientname, patientid, gender, age, medicalinsuranceno, aadhar no).

2nd table:
create table checkupdetails (patient id, annualcheckupmonth, fees) ex: {M.I.NO} {PId, MINO, ANO}
{P.Id}
{A.NO}
{MINO, ANO}
{PId, MINO}
{PId, ANO}

2) Cardidate keys :- It is a set of attributes which uniquely identify the table.
→ A sirgle can have the multiple candidate keys
→ out of all candidate keys choose one key as primary key & all should be the alternate keys
ex: {Pid, MINO, ANO} → All are candidate keys
Pid ↑ primary key; MINO, ANO ↳ alternate keys

→ with the help of candidate keys we can write unique values to the columns to identify null values can be present.

3) primary key: It is an attribute which uniquely identify the table
→ It should be unique & not null
→ A single table can have only one primary key & multiple candidate keys.
→ No null values are allowed.

4) Alternate key: In a set of candidate keys which are not selected as primary keys are known as Alternate keys.
→ It is also called as secondary key.
→ the keys which are non-primary keys are called Alternate key

Ex: { P.id, MENO, ANO } → All are candidate keys
P.id → primary key
MENO, ANO → non primary keys (Alternate keys)

5) Unique key: It is similar to the primary key except it will allow the null values.

6) foreign key: A primary key that can referred to the ~~other table~~ values of another table
→ the relationship b/w the two tables are called as referential integrity.
→ In a table we can have multiple foreign keys
→ foreign key can have null values.
→ we can have the duplicate foreign keys.
→ A table which is having foreign key is known as child table.
→ A table that is referred by the foreign key is called parent table.

7) Composite key: It is a key which consists of multiple columns which uniquely identifies the table.

write unique
lues can

iquely

y key &

ys which
as

called

eys

ys)

except

ed to the

ed as

s

un as

is called

ts of
table.

SQL (Structured Query Language)

→ By using this query language we manipulate the data inside the DB with the help of sql queries / commands

→ sql commands are classified into 5 categories

1) DDL (Data Definition Language) → create, alter, drop, truncate, rename
2) DML (Data Manipulation Language) → insert, delete, update
3) DRL (Data Retrieval Language) → select
4) DCL (Data control language) → Grant, Revoke
5) TCL (Transaction control language) → commit, Rollback, Revoke

Datatypes

1) string datatypes:

1) char
2) Nchar
3) Varchar
4) Varchar2
5) long

1) char(size) : It is used to store fixed length character strings
→ By default size is 1 character (minimum size)
→ Max. size is 2000 characters / bytes
Ex: empid

2) Nchar (Numeric characters)

3) varchar2 (size) : It is used to store variable length character strings. It holds the unicode data
→ default size : no default value
→ max-size : 4000 characters / bytes
Ex: ename

4) long : It is used to store variable length characters
→ It is similar to varchar2 difference is max. size is 2GB
→ In a single table only one long datatype column is allowed

5) Numeric Datatypes

1) Number
number (precision, scale) → It will allow decimal values
number (precision) → It will allow numbers without decimal values
precision → It represents total no. of digits
scale → max no. of digits after decimal

Ex: food_price(7,2)

Ex: 12.75 ✓
123.1 ✓
4563.22 ✓
99999.99 ✗
123 ✓

Date Datatypes

1) Date ('DD-MON-YY')
   - digits of a day
   - 1st 3 characters of month name
   - last 2 digits of year
   - this datatype occupies 8 bytes of space

Ex: 06-JAN-25
06/JAN/25 → invalid

Binary datatypes: It is used to store the binary data

1) Raw (size) → It is used to store the images, logos, thumb impressions
   → max size: 2000 bytes
2) long raw → It is similar to the raw datatype.
   → max size: 2GB
   → In a single table only one long raw datatype field is allowed

Large objects datatypes

→ There are 3 types of large object datatypes:
→ It is used to store unstructured data that is Raw Data like images, audio & video files

1) BLOB
2) CLOB
3) NCLOB

CLOB (Character Large objects)

→ It is used to store single/multiple byte character set
→ max size is upto 4GB

BLOB (Binary large objects)

→ It is used to store unstructure binary large objects they are bit streams internally they have saved in hexadecimal format

NCLOB
→ It is used to sac both cLOB data & BLOB data
→ max size is 4 GB

ROWID
→ In a table we are having multiple tables each row in the table has a address
→ the row address can be retrieved by using rowid
→ ROWID support partitioned tables & indexes

BFILE (Binary File)
→ It is used to access binary file which is stored externally outside of the DB

TIMESTAMP : It is exactly same as date datatype
→ It will store day, month, year, hours, minutes, seconds & time zone
→ default date, 1st January 4712 bc
→ default time : 12:00 AM

first oracle
scott tiger
un pass

DDL (Data Definition Language)
In oracle
→ username : system (default)
→ password can be anything
In mysql : username : root (default)
password can be anything

→ create user username identified by password;
granting privelege to user :
Eg : grant connect to username;
↳ privelege
→ grant connect, resource to username;

its of year
4 & bytes
data
ay
ages, logos,
atype.
latype
types :
that is
byte character
ory large
nally they
b

1) create : this sql command is used to create an object

ex: syntax : create table tablename (
column1 datatype (size),
column2 datatype (size),
);

ex: create table students (
stid number(10),
fname varchar2(20),
lname varchar2(20),
dob date,
doj date,
fees number(7,4),
gender char(1),
);

→ select 10+20 from dual;
30 ↑ it is the dummy table that doesn't store any data but using this we can perform operations

→ Buffer memory : the last sql command will be stored in temporary memory that is called buffer memory

2) select * from tab;
↑ all
this will list out the tables that has been created.

→ 2) ALTER : this command is used to change the structure of the existing table

→ ALTER has 3 fields :
1) ADD
2) DROP
3) MODIFY

ADD : it is used to add new field to the existing table

syntax :
alter table tablename add (columnname datatype (size));

ex: alter table studet add (stdname varchar2(20),
stdfname varchar2(30));

MODIFY:

→ It is used to change size, datatype.

ex: alter table student modify (name varchar2(30));

→ Before inserting data we can change the datatype of a field but we can't change after inserting data

→ For inserted data: we can't decrease the size but we can increase the size.

DROP:

→ It is used to drop already existing column

syntax:

alter table tablename DROP column columnname;

ex: alter table std drop column name;

alter table std drop column std; → tablename.

→ we can't drop the all columns in a table

3) DROP: By using this sql command we can delete the object from the DB

syntax:

drop table tablename;

drop view viewname;

→ before oracle 10g version objects will be deleted permanently

from oracle 10g version onwards deleted tables can be send to recycle bin

we can get deleted table from the recycle bin. by using flashback command

syntax:

Flashback table tablename to before drop;

table can be deleted permanently from the recycle bin by using purge

syntax:-

drop table tablename purge

→ recycle bin is a predefined table inside the oracle DB which holds all the deleted objects.

It is read only table.

- we can get deleted name from the recycle bin by using ORIGINAL_NAME column.

Ex: select ORIGINAL_NAME FROM recyclebin;

TRUNCATE:

→ All the rows from the table will be deleted empty structure of the table is available.

Syntax:

truncate table tablename;

→ creating duplicate copy of another table:

Ex: create table student as select * from emp;
(student: copy of original table; emp: original table)

Restrictions on long datatype

→ In a single table only one long column allowed
→ object type can't be created on the long attribute
→ It can't be used in WHERE CLAUSE or Integrity constraint
→ Indexes can't be created on the long datatype
→ long can be return through a function. It can't be return by using stored procedures
→ It can be declared PL/SQL but it can't be referred/ reference in SQL

RENAME: By using this SQL command we can rename the table & we can rename the column by using alter command. It is introduced in oracle 9I version.

Syntax:

rename oldtablename to newtablename;

Ex: rename std to emp;

changing the name of a column

Ex: alter table emp rename column stdname to empname;

2) DML commands

1) Insert:- By using this command we insert rows into relational table, view based table, partitioned table, object table, sub partition of composite partition table.

Syntax:

insert into tablename (column1, column2) values (column1 value, column2 value);

ex: create table emp (
empno number (5),
ename varchar2 (20),
dob date,
insertedby varchar2 (15)
);

ex: insert into emp values ('10', 'isa', '01-JAN-24')

o/p: error → because we have to enter the values to all the columns must.

→ inserting data to specified columns

ex: insert into emp (empno, ename) values ('10', 'isa')

→ insert into emp values ('10', 'isa', sysdate, user);

→ sysdate & user are the built in/ aggregate functions

sysdate → returns the current date of the system

user → returns the username who was log in

→ Inserting the data at the runtime by using substitution variables

→ these variables will stored the data temporarily

1) &

2) &&

3) DEFINE

4) ACCEPT

& → it will apply for each instance when sql start, is executed i.e., data is stored temporary in substitution variable.

ex: insert into emp (&empno, '&empname');

substitution variable

&& ⇒ this substitution variable will apply for all instance until sql stmt is execute i.e., data is stored permanently in substitution variables

Ex: insert into emp (&empno, '&empname');

~~DELETE~~ C

> save D:\jdalex.sql → the last task will be stored inside this path.

> sae D:\jdalex.sql replace.

DELETE : this cmd is used to delete records from the table

syntax:

→ delete from tablename; → all records will be deleted

→ to delete specific rows

syntax:

delete from tablename where condition;

Ex: delete from emp; → all records are deleted

rollback; → all records will be back

delek from emp;

commit; → permanently deleted in the DB

> rollback;

no rows

→ Inserting the null values into 2 ways:

1) Implicit Insertion

2) Explicit Insertion by using null keyword

Ex: insert into emp values (100, 'isa', 'NULL', null);

→ DEFAULT:

→ at the time of creating the table we can use default clause. If we don't enter the value default value will be considered.

Ex create table emp(

empno number(5),

empname varchar2(20) DEFAULT 'isa'

);

UPDATE: this sql cmd is used to update old values with new values.

Syntax:

update tablename set columnname = value where condition;

ex: update emp set address = 'usa', ename = 'isa' where empno = 100;

ex: update the salesman salary to 20% increment who joined before 2005

update emp set salary = salary + 0.20, job = 'seniorsalesman' where hiredate < '01-JAN-05' and job = 'salesman';

DCL commands

→ these are the DBA commands:

1) Grant
2) Revoke

Grant: by using this cmd we are granting the privileges to the user.

Revoke: by using this cmd we revoke (taking back) all the privileges from the user

from the user.

Type of Previligies

1. Insert
2. Delete
3. Update
4. View ~~View~~ View
5. Procedure
6. function
7. index
8. Squencies.
9. Alter
10. Drop
11. All

Branch

→ Grant: By using grant command we assign specific to the user to perform certain actions only on the objects. (Select, insert, etc.)

Syntax: grant privilges on object-name to username;

Ex: grant select on ob system.emp to sridhar;

→ Revoke: These command is used to revoke some or all priviliges which are granted by another user.

Syntax: revoke privilge-name on objectname from username;

Ex: revoke all on system.emp from sridhar;

(*) Save point

→ It is the mechanism that allows you to create a point with in a transaction. It can be called label by using savepoint.

Syn: Savepoint savepoint_name;

Ex: > insert into del values (100);
1 row created
> Savepoint ins;
Savepoint created
> update del set empno=200;
1 row updated.
> Savepoint upd;
Savepoint created
> delete from del;
1 row deleted
> Savepoint de;
Savepoint created
> rollback to ins;

- Rollback ops,

Data Integrity constraints

- It is used to apply set of rules or condition at the DDL level ie, at the time of creation object so that only valid data can be entered & retrieve

Types of constraints

1) key constraint
2) domain constraint
3) Referential integrity constraint
4) user-defined constraint

key constraint

→ these constraints checks the individual values on a column

→ it is divided into 3 types:

1) unique constraint
2) not null
3) primary key

unique constraint:

→ it doesn't allow duplicate values but allows null values. ex; enrollid

notnull

→ it allows duplicate values but doesn't allow null values

ex; ename

primary key

→ it doesn't allow duplicate values & null values

→ only 1 primary key is allowed for single table

→ primary key is 2 types;

1) simple primary key → defined on a single column
2) composite primary key → if primary key constraint defined more than one column

→ we can write upto 32 composite primary key columns

→ we can see all the constraints in user_constraints table

ex: select * from USER_CONSTRAINTS;

→ for checked & not null constraints → 'c' (constraint type)

→ select constraint_name, constraint_type from USER_CONSTRAINTS;

→ for unique constraints the constraint type is 'U'

→ each constraint name is numbered uniquely like sys_c0 (system defined constraint)

→ select constraint_name, constraint_type from USER_CONSTRAINTS where table_name = 'EMP';

Restrictions

1) not null:
→ not null can't be applied at view constraint
→ not null can't be specified for an object

2) unique:
→ It can't be applied for the column
→ LOB, LONG, longraw, nested tables, object, BFile, varray, timestamp with time zone → for these we can't apply

```
create table emp(
empno number(5) not null;  → column level constraints
ename varchar2(2) constraint emp-no-un unique;
phno number(10);
constraint emp-phno-pk (primary key);  → table level
constraint primarykey (phno, ename, empno);  → composite level
```

Restrictions on primary key

→ A table or view can have single primary key
→ primary key can't be implemented on the column LOB, long, longraw, nested tables, object, BFile, varray, timestamp with time zone → for these we can't apply
→ primary key can be specified only for the top level of the view
→ some column/combination of columns can't be designed as primary key / unique key

Domain constraints

→ In this constraints we define valid range / list of values on a column by using "check" keyword
→ check keyword uses 2 operators:
in → it defines list of values
between → it defines range of values

create std table with sno between 1 and 60
course core ore oracle, mysql, java
min fees 5000
max fees 10000

create table std (
stdname varchar(2) constraint std_name_un unique,
stdno number(10) not null,

→ create table student (
sno number(5) constraint pk_sno primary key,
sname varchar2(20) not null,
course varchar2(20) not null,
fee number(5) not null,
constraint ck_sno check (sno between 1 and 60),
constraint ck_course check (course in ('oracle',
'mysql', 'java')),
constraint ck_fee check (fee between 5000 and 10000)
);

Referential Integrity constraints

→ It is also called as foreign key.
→ Design a column has a foreign key & establish the relation b/w specified primary key & unique key of the different table.
→ composite foreign key is designates combination of foreign key
→ table or view containing the foreign key is called child object
→ table or view contains refereced key is called parent object
→ In a table we can have primary key & foreign key
→ By using references keyword we establish relation b/w two tables

select empno, ename, sal, sal*12 as "Annual sal" from emp;

display emp details except job as clerk

Ex: select ename, empno, sal from emp where job<>'CLERK';

→ with multiple conditions

Ex: select ename, empno from emp where job<>'CLERK'
and sal>3000;

write a query display emp details whose deptnos are 10, 20, 30, 40, 50

select * from emp where deptno in (10, 20, 30, 40, 50);

Special operators

1) in
2) not in
3) is null
4) is not null
5) like
6) not like
7) between & not between

1) in → it will read the value one by one from the list

→ Instead of in operator we can use or operator

→ Where we are having multiple values from single table go with in operator

Ex: select * from emp where deptno in (10, 20, 30, 40, 50);

display smith & king emp details

select * from emp where ename in ('SMITH', 'KING');

2) Between: this operator is used to display range of values

→ select empno, sal from emp where sal between 1000 and 5000;

→ select sal, comm from emp where comm is null;

→ select sal, comm from emp where comm is not null;

→ select sal, comm, sal+comm from emp;

* null + some value = null

Ex: → select sal, comm, sal+nvl(comm, 0) from emp;

→ NVL() → it is a predefined fn which is used to replace user defined value for null.

→ NVL (exp1, exp2)
→ NVL2(exp1, exp2, exp3)

Ex: select nvl2(null, 10, 10) from dual;
if it is null the value 30 will be replace
if it is not null the value 10 will be replace

update emp table if commision is null update 500
if comm is not null update 500.
update test set comm = nvl2(comm, comm+500, 500);

Like operator
→ this operator is used for searching the data based on the character pattern
→ along with the like operator we can use two special wild card characters (%, _)
→ Ex
Ex:) select empno, ename from emp where ename like 'S%';
↓
starts with
select empno from emp where ename like '%S%'
ends with
'%S%' → start & ends with
display emp details who joined in 1981
→) select * from emp where year
like '%81';
→) select sal ~~where~~ from emp where sal like '____';
four digits

Aggregate fn's / Built-in fn's
→ the fn's can be operated on the single row
→ multiple row
single row fn's
→ this fn will return a single for each row
multiple row fn
→ this fn will manipulate on groups of rows & return one result

→ Display emp details who joined in 1981
>select empno, hiredate from emp where hiredate like '%81';
>select empno, hiredate, sal, from emp where sal like '____';
>select empno, hiredate, sal, ename from emp where ename like '____';

Aggregate function (or) Built In functions:
→ the functions can be operated on the group single row, multiple row.
1) single row function: this fn will return a single value for each row
2) multiple row function: this fn manipulate group of rows & return one result

distinct keyword: it is used to eliminate duplicate rows in the table
syntax: select distinct (column) from tablename;
ex: select distinct (job) from emp;

order by clause: this clause is used to display the rows in sorting nature. Default sorting order is ascending order
ex: select ename, sal from emp order by sal; → for ascending order
select ename, sal from emp order by sal desc → for descending order
syntax: select colname from tablename order by colname;
• instead of writing colname we can write numberate.
ex: select ename, sal from emp order by 1 desc;

lower(col name): converted to lower case
upper(colname): converted to upper case
initcap(colname): first capital letter
concat(colname1, colname2): to add two characters string
ex: select lower(ename) from emp;
select initcap(ename) from emp;
select ename, job, concat(ename, job) from emp;

substring: substring(min) it returns specified characters from the given string from specified position

ex: select substr('oracle java', 0, 4) from dual; → orac

select substr('oracle java', -2) from dual → va

length(): it returns no. of characters in a given string

ex: select length('oracle java') from dual; → 11

instr(): it returns numeric value of the nosed character

syntax:

instr(string, character, position)

ex: instr('shiva sarala', 'a', 1) from dual; → 5

" " " " -1) from dual; → 12

lpad(): it pads the character right justify to total width n characters.

ex: select lpad('oracle', 20, '*') from dual;

o/p: **********oracle

rpad(): it pads the character left justify to total width n characters

ltrim(): unwanted spaces & characters of a string will be deleted at left side

ex: select ltrim('xyxyoracle', 'xy') from dual;

rtrim()

trim()

replace(): it returns every occurence of string replaced by given string.

ex: select replace('oracle', 'java') from dual;

translate()

ex: select translate(job, 'e', 'r') from emp where job = 'PRESIDENT'

o/p: PRESIDENT → E replaced with r

chr (int value): it returns the character of binary equalent value

Ex: select chr(65) from dual; → A
select ascii('A') from dual; → 65

round(): this fn is used to round up to the nearest integer value

Ex: select round (19.67, 1) from dual; → 19.6
select round (19.6) from dual; → 19.7
select round (19.6) from dual; → 19.66
select round (45.926, 1) from dual; → 45.9
select round (45.926, -1) from dual; → 50
select round (45.926, -2) from dual; → 0
select round (55.926, -2) from dual; → 100

Conversion fn's

1) to_char()
2) to_date()
3) to_number()
} → there are the three explicit conversion fn's

1) to_char(): it is used to convert given date & number into character datatype

decimal indicator (D):
select 1234, to_char (1234, '9999D99') from dual;

→ Default decimal indicator is dot(.)

Scientific Notation Indicator (E)

select 1234, to_char (1234, '9.9EEEE') from dual; → 1.2E3

group indicator (G):
select 1234, to_char (1234, '9G999') from dual; 1,234

Currency Indicator (L):
select 1234, to_char (1234, 'L9999') from dual;

Ex: select sal, to_char (sal, 'L99G999D99', 'NLS_CURRENCY = $') from emp; o/p: $ 10,000.00

Minus indicator (mi) : it returns negative value with '-' sign

Negative Number Indicator (PR) : it returns negative value in <>

Ex : select to_char (-1000, 'L99G999D99PR') from dual;

L → used to represent ₹ symbol
→ by default ₹

o/p : <₹1000.00>

Ex : L99G999D99PR
<₹10,000.00>

PR → displayed in "< >"

5/2/25

round() → this fn will roundup to the nearest integer value

Ex : select round (45.926, 1) from dual; → 45.9
select round (45.926, 2) from dual; → 45.93

(45.926, -2) → 0.45926 → 0 × 100 = 0
(45.926, -1) → 4.5926 → 5 × 10 = 50
(45.926, -3) → 0.45926 → 0 × 1000 = 0
(55.926, -2) → 0.55926 → 1 × 100 = 100

ceil() → it roundup to next highest integer value

select ceil (2.33) from dual; → 3

Floor() : it rounds up to lowest integer value of a given no

select Floor (2.33) from dual; → 2

sysdate() → it is used to display system date

select sysdate from dual; → 05-FEB-2025

Ex : select to_date ('28-08-2010', 'dd-mm-yyyy') from dual;

o/p : 28-AUG-10

rank() : it is used to give sequence of increasing numbers for each row in the table

syntax : select colname, rank() over (order by colname)
from tablename;

Ex : select empno, sal, rank() over (order by sal) from
emp;

alue with

egative value

Rom dual;

t # symbol

5|2|25

L; integer value

5.9

5.93

x100 = 0

x10 = 50

1000 = 0

00 = 100

alue

f a given no

e

dual;

creasing

no rows)

sal) from

Date Functions:

Ex: select sysdate+3 from dual; 08-FEB-25

add_months(): it returns no. of months added to the specific date.

Ex: select add_months(sysdate, 2) from dual; 05-APR-25

months_between(d1, d2): it returns no. of days b/w d1 & d2

next_day(d1, char): it returns date of the 1st week named character

Ex: select sysdate, next_day(sysdate, 'WED') from dual;

o/p: 12-FEB-25

last_day(d): it returns the date of last day of the month

Ex: select sysdate, last_day(sysdate) from dual;

o/p: 28-FEB-25

Numericday indicator(D)(1-7): it returns week day number.

Ex: select sysdate, to_char(sysdate, 'D') from dual;

o/p: 05-FEB-25 4

weekday spelling indicator(Day).

Ex: select sysdate, to_char(sysdate, 'Day') from dual;

o/p: 05-FEB-25 wednesday (a character)

Abbreviation indicator(DY): it returns 3 characters of the week day

Ex: select sysdate, to_char(sysdate, 'Dy') from dual;

o/p: 05-FEB-25 WED

month day indicator(DD): it returns day of the month

Ex: select sysdate, to_char(sysdate, 'DD') from dual;

o/p: 05-FEB-25 05

Year Day indicator(DDD): it returns day of the year (1-366)

Ex: select sysdate, to_char(sysdate, 'DDD') from dual;

o/p: 05-FEB-25, Jan 31 → 31+5 → 36

Aggregate fn's

1) count() : it returns no. of records in a table

Syntax: select count(colname) from tablename;

ex: select count(empno) from emp; → 256

ex: select count(comm) from emp; → 4 (it will not consider null values)

2) sum() :

ex: select sum(sal) from emp; → 29025

3) max() :

ex: select max(sal) from emp; → 5000.

4) min() :

ex: select min(sal) from emp; → 800

5) avg()

ex: select avg(sal) from emp; 1500

6) stddev()

variance()

7) greatest() : it returns greatest value from given expression

ex: select greatest('HARRY', 'HEPRTOT') from dual; → HARRY

→ we can take values, dates, string values

ex: select greatest('05-FEB-25', '07-FEB-25') from dual;

8) least() : it returns least value from the expression

a) user → it returns current user

ex: select user from dual; → SYSTEM

10) joins :

→ it is a query that combines rows from two (more tables, views, materialized views

→ join query should contain atleast one joining condition in where clause

→ joining condition compares two tables

→ to join the two tables one column should be common in the both the tables.

emp table
empno(pk)
ename
sal
deptno(FK)

dept table
deptno(PK)
dname
dlocation

dept → parent
emp → child

types of joins:
1) cross join
2) equi/simple/inner join
3) outer join
4) self join

6/2/15

dept table

	null	type
deptno	notnull	number(2)
dname		varchar2(14)
loc		varchar2(13)

1) Cross Join:

→ It will display combination of data from multiple tables
→ In this join each value in the 1st table is mapped with all the values in the 2nd table

ex: $S_1 = \{a, b\}$
$S_2 = \{c, d\}$
$S_1 \times S_2 = \{(a,c), (a,d), (b,c), (b,d)\}$

syntax:
select colnames from table1, table2 where
<cond> order by colname

display empnames, salaries & their deptnames
select ename, sal, deptno deptname from emp, dept;

→ In this we get valid & invalid data
→ To over come this drawback we have to specify joining condition

2) Equi/Inner Join:

→ It is a cross join, if we specify joining condition by using equal (=) operator. It will display only match data from all the tables
→ In joining condition we write primary key of table1 with foreign key of table2

syntax:
select col1, col2, .. from table1, table2 where table2.pk = table2.fk;

ex: select emp.empno, emp.ename, dept.deptno, dept.loc
from emp, dept where dept.deptno = emp.deptno;

-) By using table alias

Ex: select e.empno, e.ename, d.deptno, d.loc from emp e, dept d
where e.deptno = d.deptno;

display emp details & dept details for job = manager

select e.empno, e.name, e.job, d.deptno, d.loc from emp e,
dept d where e.deptno = d.deptno and job = 'MANAGER';

Inner join

syntax: select columns... from table1 inner join table2 on
joining condition

Ex: select ename, sal, dept.deptno, dname from emp inner join dept
on emp.deptno = dept.deptno;

display emp details & dept details who is working
under Accounting & sales

~~select empno, ename, ename~~,
select e.*, d.* from emp e, inner join dept d on
d.dname in ('ACCOUNTING', 'SALES') and d.deptno = e.deptno;

Self Join:

→ It is a join of a table by itself
→ the same table appear twice in the from clause

syntax:
select col1, col2 - from tab1, tab1 where joining condition;

Ex: ename city
kiran mtk
babu cngl
jagtu kzt
sri mumbai
ram delhi
john delhi

Ex: select * from emp, emp where
where john is living
1) go to 1st Alias table & find where john is living
2) from that get the city(delhi)
3) city from the first Alias table is compared with
city name in 2nd Alias table

base table

Ex: select e2.ename, e2.job from emp e1, emp e2 where
e1.ename = 'SMITH' and e1.job = e2.job;

outer join

→ It is used to display all data from one table & only matched data from another table

Types

1) left outer join : it displays all the data from left table & only matched data from right table

2) Right outer join : it displays all data from Right table & only matched data from left table

3) Full outer join : matched data from both tables, unmatched data from left & right tables

syntax:

```
select col1, col2 from tb1 [left/right/full join] tb2
on tb1.pk = tb2.fk;
```

sub queries

→ A query within the another query is called sub query

→ A sub query are preferable to display o/p from one table & having i/p value to the another table

```
select colname from tname where colname [=/in/exist/not exist]
(select colname from tname);
```

Types

1) single Row sub query : A sub query which returns single o/p value

→ In this case in b/w inner sub query & outer sub query we use equal to (=) operator

2) multiple Row sub query : A sub query which returns multiple o/p values

→ In this case in b/w inner & outer sub query we use "in" operator

display emp details whose salary is maximum

```
select * from emp where sal = (select max(sal) from
emp);
```

display dept details of emp smith

```
select * from dept where deptno = (select deptno
from emp where ename = 'SMITH');
```

in

co-related sub queries

→ this sub queries uses exist & not exist operators

exists: it returns true if subquery fetches atleast one value

not exists: it returns true if sub query fetches no value at all

display dept details which is not having atleast one emp in it.

select d.* from dept d where not exists (select 10 from emp e where e.deptno = d.deptno;

select d.* from dept d where exists (select 10 (or) * from emp e where e.deptno = d.deptno;

7/2/25

→ we can write the sub query in where clause of select statement then it is called Nested sub query

→ A subquery in from clause of select stmt is called inline view

→ we can write 255 subqueries in the where clause

→ In the multiple row subquery we can use following operators:

1) In → equal to any number in the list

~~any~~

~~any~~

any (some → compare to each value returned by sub query

5) ~~compare to~~ ~~any~~ ALL → 4

5: select ename, sal, deptno from emp where sal in (select max(sal) from emp group by deptno));

any (some operator

→ select ename, sal, deptno from emp where sal < some (1500, 1600, 1800); → it displays the sal below 1800 (max in list)

<any → max value in the list

>any → min value in the list

All : means greater than max value in the list

>All

<All →

select ename, sal, deptno from emp where sal < All (1500, 1600, 1800);

→ we can write select stmt in the from clause it will be considered a column in the select stmt

eg: select ename, sal, (select max(sal) from emp) from emp;

→ In the co-related subqueries we perform queries upon the data with joining condition

views

→ views are used for security purpose
→ we can restrict the user to access the complete data
→ it reduces redundancy & improve the performance
→ views will not hold any data
→ It can be defined as stored select stmt
→ views can be created ~~based~~ on the table & materialized view
→ views can be created as objects view & relational views
→ object view support LOB datatype, nested tables, DML, description, select

syntax: create or replace [force] view view-name [alias name] as subquery [with {check option / READONLY}] constraint constraintname;

Types

1) simple view (updatable view)
→ It is created based on single table data
→ It allows DML operations

syntax:
```
create or replace view view_name
as
select colname from table_name where <cond>;
```
eg: create or replace view v1
```
as
select * from emp where deptno = 10;
```
select * from v1;
insert into v1 (empno, ename, deptno) values (22, 'aa', 10);
select * from emp;
```
→ while performing DML operations on simple view we have following restrictions:
1) If the simple view contains group fn's (aggregate fn's), group by, distinct (we can eliminate duplicate values)
```

, set operations, sources, joins (we can't perform DML
operations on simple views on base table)
3) we must include base table not null column values
eg: ~~select~~ insert into v1(ename, sal, deptno) values ('iso', 200, 21)
can't insert into ('system', 'ORPNO')

eg: create or replace view v1
as
select deptno, max(sal) max from emp group by
deptno;
(or)
create or replace view v1(deptno, max)
as
select deptno, max(sal) from emp group by deptno;

→ In oracle if view contains fn's/expressions then
we must create alias names for the fn's
→ we can create alias by using view parameters

2) Composite view / complex / read only view:
→ we create the view based on one/more tables
→ It doesn't allow DML operations
syntax:
create or replace view view_name
as
select colnames from table1, table2 where joining condition;

write a query the view which maintains managers, ename,
sal, job, deptname, loc
create or replace view v1
as
select ename, sal, deptname, loc from emp, dept where
job = 'MANAGER' and emp.deptno = dept.deptno;

→ select * from tab; → it displays all the tables
→ we can see the text of the view in USER_VIEWS ~~data~~
data dictionary
select view_name, text from user_views;

→ To delete a view:
syntax: DROP view viewname;

perform DML
le)
column values
alues (isa, 200, 21)
)

group by

)

by deptno;
sions then
n's
erefers

ble

ring condition;
ages, erore;

o create
ro;

VIEWS ~~alter~~

Materialised view

→ In this view we store the data
→ No DML operations are allowed
→ these are used in Data ware houses

Syntax:
create materialized view view_name
as
select stmt;

Ex: create materialized view emp_dept
as
select deptno, sum(sal), count(empno) from emp
group by deptno;

Difference b/w view & materialised view

view	materialised view
→ view doesn't stored data	→ it stores the data
→ view purpose is security	→ purpose is performance
→ once dropping base table we can't access view	→ once dropping base table we can access view
→ we can perform DML operations	→ we can't perform DML operations

Forced view

→ we can create a view without having the base table such type of view is forced view

Syntax:
create or replace force view view_name
as
select stmt;

Ex: create or replace force view v2
as
select * from ita;
O/P: warning with compilation error
so,
create table ita(sno number(4));
table created
Alter view v2 compile;
desc v2;

8/2/25

Sequence

→ It is a database object which is used to generate sequence of integer values with specified interval

→ sequences are used for generating primary key values

Syntax: create sequence sq_name
increment by <val>
start with <val>
max value <val>

→ By default sequence starts with "1" & incremented by "1"

→ sequence is an independent object

Pseudo columns

sequence.CURRVAL → display current value of sequence
sequence.NEXVAL → displays next value of sequence

Ex: create sequence sqno2;
select sqno2.currval from dual; → 1
select sqno2.nextval from dual; → 2
" " " " ; → 3

Ex: create sequence custno start with 1001;
→ update cust_dtls set custno = sqno2.nextval;

→ create table customers (id int, name varchar2(20),
sno number);
create sequence id start with 6600 increment by 5;
create sequence sno;
insert into customers(id.nextval, 'aa', null);
insert into customers(id.nextval, 'bb', null);
insert into customers(id.nextval, 'cc', null);
update customers set sno = sno.nextval;

O/P:

id	name	sno
6600	aa	1
6605	bb	2
6610	cc	3

d to gerate
d interval
ey values

erated by "1"

f sequence
f sequence

l;

+2(20),

by 5;

;

synonyms

→ these are used to create permanent alias names for the table

Types

1) **private synonym:**
→ it is the default synonym created by the owner of the table

2) **public synonym:**
→ it is the synonym created by the DBA & it can be accesible by all permit user.

syntax: create [public] synonym syn-name for tbl_name;

Ex: create synonym cd for customers;
select * from cd;
delete from cd where sno=2;
select * from cd;
Drop synonym cd; → we are dropping only the synonym not the original table.

Group By

→ It is used to group related data by considering distinct values from the column
→ Each group DB engine executes the aggregate fn & display the result

syntax:
select colnames, aggrfn's from table
where condition
group by colname
having aggrfn
order by colname;

find the no. of emp's working under each dept
select deptno, count(*) from emp
group by deptno;

find out no. of emp's under each dept getting more than 5000 salary order by deptno.
select deptno, count(*) from emp
group by deptno
where sal > 5000
order by deptno;

Having clause
→ It is used to specify conditions on group by o/p
find out no.of emps working under each dept order by deptno if dept contains atleast 3 emps

```
select deptno, count(*) from emp
group by deptno
having count(*)>3
order by deptno;
```

deptno	count(*)
20	5
30	7

PL/SQL (Procedure language SQL)
→ It is the extension of SQL
→ It contains procedures, fn's, triggers, cursors, error handling, etc.
→ It is the procedure language with SQL concepts
→ parallely we can execute multiple queries
→ It will reduce no.of hits to the database
→ It increases the performance
→ Modularity features, Reusability of code

Types of pgms
1) Anonymous pgm :- the pgm which doesn't have name. → it store use
2) sub pgm's (stored procedures)
→ there are 2 types of Anonymous pgm's :
1) static 2) dynamic

structure of PL/SQL

```
Declare
<declare variables>
Begin
<Assignments>
<output starts>
<data processing>
Exception
<error handling code>
End
```

Declare block : we can declare the variables
Ex : var-name datatype(size);
sno number(2);
name varchar2(20);

group by o/p
h dept order
3 emp's

-)

sors, error

concepts
nics
ase

have none.
stmt use

sles

Begin: the pgm starts from here

Assignment stmts:

(:=) v_eno:=100;
into
by using select keyword; into v_ename, v_esal, v_eno;
select ename, esal, eno

→ we can store the values into the variables by using assignment operator (:=) & by using select stmt with "into" keyword

→ It can be value/expression.

output stmt

→ By using fn "DBMS_OUTPUT.PUT_LINE(msg);"

→ By using "/ (slash)" it will compile & execute the pgm

single line comment (--)
multi line comment (/* text */)

→ How to on server: set serveroutput on;
→ To open the editor: ed

Ex: Begin
dbms_output.put_line('hlo');
end;

→ before executing the pgm the server should be on

display addition, avg, max, min of 300 & 500

```
declare
x int;
y int:=300;
s int;
a number(7,2);
mx int;
mn int;
begin
x:=500;
s:=x+y;
a:=s/2;
select greatest(x,y) into mx from dual;
select least (x,y) into mn from dual;
dbms_output.putline(s);
      "          ('Average='||a);
      "          ('Maximum:", mx);
end;  "          (mn);
```

9/2/25

display
write a pgm to display emp info. whose emp no is 7654

```
declare
vempro:=7654;→error  so vempro int:=7654
venore.varchar2(20);
vjob varchar2(20);
vsal number(5);
begin
 select empno, enore, job, sal into vempro, venore, vjob,
vsal from emp where empno=vempro;
dbms_output.put_line(vempro);
                     (venore);
                     (vjob);
                     (vsal);
end;
```

Dynamic pgm

→ By using "address of operator (&)" we accept i/p values dynamically at the runtime from the user

```
declare                    o/p: enter the value for vempro: 7654
vempro int;
begin
 vempro:=&vempro;
```

write a pgm display dept details under which emp is working with given empid.

```
declare
 vempro int;
 vdno number(4);
 vdnore varchar2(20);
 vdloc varchar2(20);
begin
 vempro:=&vempro;
 select * into vdno, vdnore, vdloc from dept where
 deptno=(select deptno from emp where empno=vempro);
 dbms_output.put_line(vdnore);
                      "
end;
```

9/2/25

no is 7654

ore, vjob,

ot i/p values

wer

no:7654

h emp is

where

pro = vempro);

Type compatibility keywords

→we can declare the variables dynamically / implicitly as column datatype by using '%TYPE' & Record type & by using '%ROWTYPE'.

→It eliminates datatype & size compatability issues

declare variables implicitly (%TYPE)

syntax: variable_name table_name.col_name%TYPE;

ex: vdno dept.deptno%TYPE;

→In PL/SQL we have 3 types of Records :-

1) Table based Records (%ROWTYPE)

→this record type variable supports all colums from the table

2) cursor based Records (%ROWTYPE)

→this record type variable supports all colums from the cursor

3) User Defined Records (TYPE with keyword IS RECORD)

→this variable supports all columns from the table

syntax: varname table_name%rowtype;

ex: emp_rec emp%rowtype;

write a pgm to display emp info based on given emp no

```
declare
vempo emp.empno%type;
v_rec emp%rowtype;
begin
vempno:=&empno;
select * into v_rec from emp where empno=vempno;
dbms_output.put_line(v_rec.ename);
                     (v_rec.sal);
end;
/
```

sub pgms (stored procedures)

→ A named program which is created under the DB.
→ It is also called as stored procedures
→ It is saved as pre-compiled object next it will compile once & execute n no. of times
→ It provides Reusability
→ These are executed by using "execute & exec" command
→ ~~sto~~ procedures may or may not take arguments
→ procedures can be called in another procedures / fn's
→ By default procedure will not return a value
→ Every procedure has 2 parts :-
1) procedure specification :- It contains info. like procedure name & arguments
2) procedure body :- It contains code / logic for the pgm

syntax :

```
create or replace procedure proc_name (arg1 datatype
(~~size~~))
    is/AS
    declaration block
    begin
    assigning
    data processing
    Exception
    error-handling code
    end proc_name;
```

wAP a procedure display the no. of emp's under dept 30

```
create or replace procedure proco_30
is
e_count int;
begin
select count(*) into e_count from emp where deptno=30;
dbms_output.put_line(e_count);
end proc_30;
/
exec proc_30;
```

dy :- 7

dynamic procedure

```
proco_30(vdno dept.deptno
        %type)
outer deptno = vdno.
exec proc_30(30)
```

under the DB.
ures
ct rows
no. of times

te & exec

arguments

procedures / fn's
n a value

info. like

logic for the pgm

e (arg1 datatype

s order dept 30

e deptno = 30;

procedure

o (vdno dept.deptno
%type)

eptno = vdno.

roc_30(30)

PL/SQL Records / collections:

→ Generally In PL/SQL we have 2 types of variables:

1) scalar variable: It is able to store one value at time with one datatype

2) composite variable: It is able to store collection of values with different datatypes. It is called records/ collection.

User-Based Records

→ Record is a collection of columns

syntax:

```
TYPE typename IS RECORD(
col1 datatype(size),
col2 datatype(size)
);
```

Ex:

```
TYPE empsec IS RECORD(
vename empname%type
vsal emp.sal%type
);
```

Declaring variables:

syntax: varname ctypename>;

Ex: r empsec;

Create a procedure display emp details for given empno by using user-defined records.

```
create or replace procedure proc(vempno emp.empno%type)
is
TYPE empsec is record(
vename emp.ename%type,
vsal emp.sal%type
);
rec empsec;
begin
select ename, sal into vename, vsal from emp
where empno = vempno;
dbms_output.put_line(vename);
end proc_emp;
/
```

control structures

By using control structures we can control the execution of the pgm

1) conditional stmt : it is used to execute set of stmts based on condition

i) simple if

syntax: if (condition) then
stmts
endif;

2) if-then-else

3) compound if (nested if)

Ex: create or replace procedure greatestno (num1 int, num2 int)
is
~~num1 int;~~
~~num2 int;~~
begin
~~num1 := &num1;~~
~~num2 := &num2;~~
if (num1 > num2) then
dbms_output.put_line(num1 || ' is greater')
~~num1 is greater ' || num2~~
elsif (num1 < num2) then
dbms_output.put_line(num2 || ' is greater')
else
dbms_output.put_line('equal')
endif;
end ~~proc~~ greatestno;

10/2/25

2) loops:

→ loops are used if we want to execute set of stmts for multiple times

oracle support 3 types of loops;

1) simple loop
2) for loop
3) while loop

the execution

e set of

:=

int, num2 int)

num1, num2)

n2)

of stmts

Write a procedure to find which type of customers either male/female less in count from the given city

simple loop

syntax : LOOP
stmts;
EXIT WHEN condition;
stmts;
END LOOP;

→ No condition is checked before entering into the loop
→ Before exit from the loop it check the condition

For loop

syntax: for var_name in [Reverse] range of values
LOOP
stmts;
END LOOP;

ex: for x in 1..5
loop
dbms_output.put_line(x);
end loop;

→ no condition is checked before entering into the loop
→ no need to declare the variable in for loop
→ In for loop we use "in" operator, it takes values from 1 value to last value
→ In the range we use ".." operator

while loop

Syntax: while (condition)
LOOP
stmts;
END LOOP;

write a procedure display sequence of no's from 1 to 10

create or replace procedure loops
is
begin
x int:=1;
begin
loop
dbms_output.put_line(x);
x:=x+1;
exit when x>10;
end loop;

for y in 1..10
LOOP
dbms_output.put_line(y)
end loop;
end loops;
/

write a procedure to ingest records like radius, pi, area values inside the table circle for radius range from 1 to 10

```
create or replace procedure circle_proc
is
 r int;
pi constant number(5,2) := 3.14;
a number(7,2);
begin
r:=1;
for x in 1..10
loop
a:=pi*r*r;
insert into circle values(r,pi,a);
r:=r+1
end loop;
end circle_proc;
/
```

Exceptions (Runtime Errors)

→ we get exceptions if we pass wrong i/p values / error in the logic

→ If we don't handle the exception it leads to abnormal termination

→ By using "Exception" keyword / handler we handle exception in oracle

→ there are 2 types exceptions:

1) pre-defined / Named / Built-In Exceptions

→ these are handled by DB Engine

Ex: 1) TOO_MANY_ROWS : It will be raised when select stmt reading the data from multiple table

2) ZERO_DIVIDE : It is raised when value is divided by zero.

3) CURSOR_ALREADY_OPEN : It is raised when we are trying to open the ~~cursor~~ already opened cursor

4) VALUE_ERROR : It is raised if there is incompatibility b/w declared variable & receiving value of variable

adius, pi;
radius.

Q) write a procedure display emp names job for given sal

```
create or replace procedure proc_2 (vsal emp.sal%type)
is
vname emp.ename%type;
vjob emp.job%type;
begin
select ename, job into vname, vjob from emp where
 sal = vsal;
dbms_output.put_line(vname);
  "        "    " (vjob);
EXCEPTION
when TOO_MANY_ROWS then
dbms_output.put_line('sal is duplicate');
when NO_DATA_FOUND then
dbms_output.put_line('no record');
end proc_2;
/
```

2) User-Defined Exceptions

→ The Exceptions which are created by the users
→ By using "RAISE" keyword explicitly we raise the exception.
→ create the exception variable in declaration block

syntax:
 var_name exception;
 Ex: salmiss exception;

→ Raise the Exception by using "RAISE" keyword in begin block

syntax: RAISE <EXCEPTION_VARIABLE>;
 Ex: RAISE sal_miss;

→ Handle the exception in exception block

syntax: when <exphandler> then
 stmts;
 Ex: when sal_miss then
 dbms_output.put_line('data not found');

ces / leads handle when table lce is when we opened is

write a procedure display the commission based on given empno

```
create or replace procedure excepthandle (vempno emp.
empno%.type)
is
vcomm emp.comm%.type;
comm_miss exception;
begin
select comm into vcomm from emp where
empno = vempno;
if comm is null then
RAISE comm_miss;
else
dbms_output.put_line(vcomm);
endif;
EXCEPTION
when comm_miss then
dbms_output.put_line('no comm');
end exceptHandle;
```

Cursors

→ Cursor execution process:

1) Query → sql start → executor → oracle engine → oracle db → temp memory area (cursor) → cache memory → client

Rules

→ After data is extracted it is processed into temporary memory & display to the client. Again a copy of data is stored inside cache memory.

→ If we are accessing the data from the view a copy of data will be saved into cache memory after cursor memory.

Types of cursors

1) Implicit cursor:

→ these cursors are created by SQL query.

→ this type can be assigned & managed by DBA

basedon

empro emp;

here

e → oracle

ory → client

to temporary

py of data

eco a copy

after

DBA

properties

1) SQL%.ISOPEN → it returns true if file is opened for processing

2) SQL%.FOUND → it returns true when matched data is found.

3) SQL%.NOTFOUND → it returns true, when matched data is not found.

4) SQL%.ROWCOUNT → it returns integer value which represents no. of records effected.

2) Explicit cursor

→ these are declared & defined & managed by the user

→ we can use explicit cursors to display multiple records from the table by using pgm, procedure, fn

steps to create

1) Define & declare the cursor in declaration block

syntax: cursor cursor_name
is
select colnames from tbl_name where <cond>
order by colname;

ex: cursor c_mgr is select * from emp where job = 'manager';

2) open the cursor for processing in begin block

syntax: open <cursor_name>;

ex: open c_mgr;

3) fetch the data from the cursor in begin block

Syntax: FETCH <cursor_name> into <rec_name>;
↓
rowtype variable

loop
fetch c_mgr into rec
stmts;
exit when (SQL%.NOTFOUND)
end loop;

4) close the cursor

syntax: close cursor_name;

ex: close c_mgr;

properties

1) CURSORNAME%ISOPEN
2) CURSORNAME%FOUND
3) CURSORNAME%NOTFOUND
4) CURSORNAME%.ROWCOUNT

write a procedure display emp details based on given ~~emp no~~ dept no

create or replace procedure proc-2(vdeptno emp.deptno%type)
is
cursor emp_details is select * from emp where deptno =vdeptno;
c-rec emp%.rowtype;
begin
oper emp_details;
loop
fetch emp_details into c_rec;
exit when(~~emp~~ emp_details%.notfound);
dbms_output.put_line(c_rec.empno||c_rec.ename);
end loop;
close emp_details;
end proc-2;
/

Functions

→ It is a DB object which is used to perform a task & it should return a value to the calling pgm
→ Advantage is code Reusability
→ Function also called as sub pgms
→ It is stored as pre-compiled object
→ It take ~~no~~ any or may not argument
→ Function should return a value.
→ It has 2 parts specification & body.

Specification :

create or replace function <func-name>(arg1 datatype,..)
RETURN <datatype>
is
declare block
begin
assigning block

processing
Exception
<error handling codes>
return (value/var_name);
end func_name;

ex: create or replace function sum (x int, y int)
return number
is
res int;
begin
res := x+y;
return (res);
end sum;

→ we can execute the function by using select/pgm/procedure

→ by using select :-

ex select sum(10,20) from dual;

→ by using procedure

ex: create or replace procedure proc(a int, b int)
is
res int;
begin
res := sum(a,b);
dbms_output.put_line(res);
end proc;
/
exec proc(11,12);

→ by using program

ex: create or replace function func(a int, b int)
is
res int;
begin
res := sum(a,b)

Packages

→ Package is a DB object which logically groups related objects so. it reduce the search time.

→ It contains two parts : ~~specification~~

specification : It contains fn calling strats, procedure calling starts, global variable declarations

syntax;

create or replace package <package_name>
as
<variable declaration>;
<procedure calling starts>;
<function calling starts>;
end package_name;

body of package : It contains fn's & procedures related coding.

syntax: create or replace package body <package_name>
is
<procedure_body>
<function_body>
end package_name;

write a package which contain procedures, functions then calculate salary, bonus, final salary for given empid.

create or replace package

12/2/25

→ In PL/SQL, A cursor is a pointer which points to context area (it has all the information so, that we can execute sql stmt)

→ An Explicit cursor which is not point to one command

→ Implicit cursor associated with one command

→ which implicit cursor returns true if an insert, delete, update stmts effects on one or more rows

A) SQL%.FOUND

create a package specification for the customers

```
create or replace package cust_sal
as
procedure find_sal(vempro emp.empro%type)
end cust_sal;
/
```

package body

```
create or replace package body cust_sal
is
procedure find_sal(vempro emp.empro%type) is
c_sal emp.sal%type;
begin
select sal into c_sal from emp where empro=vempro;
dbms_output.put_line('salary='||c_sal);
end find_sal;
end cust_sal;
/
```

→ Accessing the package elements (variables, procedures, functions are accessible in another program

syntax: package_name.variable_name;

ex: cust_sal.c_sal(empno);

ex: declare
code emp.empno%type;
begin
cust_sal.c_sal(code);
end;
/

list of exception handlers

exception name	oracle code	sql code	description
ACCESS_INTO_NULL	06530	-6530	It is raised when null object assigned a value
CASE_NOT_FOUND	06592	-6592	It is raised when none of the case is selected
COLLECTION_IS_NULL	06531	-6531	It is raised when pgm applies collection methods other than exit

DUP_VALUE_IN_INDEX	0001	-1	It is raised when duplicate values are stored in a column which has unique index
INVALID_CURSOR	01001	-1001	It is raised when trying to make invalid operations on the cursor ex:- trying to close the cursor which is not opened
INVALID_NUMBER	01722	-1722	It is raised when character string try to convert to the number
LOGIN_DENIED	01017	-1017	It is raised when we enter invalid username & password
NO_DATA_FOUND	01012	-1012	It is raised when select into stmt doesn't return any rows/records
NOT_LOGGED_ON	01403	-1403	It is raised when DB call is issue without connecting to the DB
PROGRAM_ERROR	06501	-6501	It is raised when PL/SQL is having internal problem
ROWTYPE_MISMATCH	06504	-6504	It is raised when cursor fetches value in a variable having incompatability datatype
STORAGE_ERROR	6500		It is raised when PL/SQL is running out of memory
TOO_MANY_ROWS	01422		It is raised when select into stmt returns more than one rows

VALUE-ERROR	06502	It is raised when an arithmetic, conversi-ons & size constraints occurs.
ZERO-DIVIDE	01476	It is raised when a number is divided by zero
OTHERS		

→ In a procedure we have i/p o/p parameters / IN OUT parameters

INPUT PARAMETERS (IN)

→ when we declare a parameter as "IN"
→ procedure is receiving a value from the calling pgm
→ procedure can't modify the value of parameters.
it is used for passing values to the procedure

Syntax:
```
create or replace procedure proc-name (p_input_param
IN number)
is
begin
proc body;
end proc_name;
```

OUTPUT PARAMETER (OUT)

→ we can declare the parameter as "OUT"
→ procedure can modify a value of the parameter this modified value can be accessible to calling pgm after procedure execution

Syntax: create or replace procedure proc-name (p_input-param OUT number)
is
begin
proc body;
end proc-name;

IN-OUT PARAMETER

syntax: create or replace procedure proc_name(p_input.
param IN OUT number)
is
begin
proc body;
end proc_name;

→ we can combine both i/p o/p functionalities.
→ It allows the procedure to receive a value from the calling pgm then we can modify the value within the procedure. & return the modified value to the calling pgm
→ when we are using IN OUT parameter we need to provide a variable that holds the value for input after procedure execution the same variable will hold modified value.

Ex
declare
my_var number:=10;
begin
my-procedure(my_var);

13/2/25

→ Default mode for a parameter when its mode is not specied IN

→ Here 'is' keyword is used instead of the 'As' keyword when creating a standalone fn in pl/sql
a) True b) False

→ How many records are supported by pl/sql?
Ans 3 (Table / cursor / user based records)

→ • which attribute enables us to create the table & cursor based records %ROWTYPE

→

CASE

→ from the multiple options it returns one value
→ CASE keyword is followed by the selector
→ CASE stmt is begins with case keyword
→ selector can be a fn call, single variable or a value
→ It can have any datatype other than BLOB, BFILE, OBJECT, PL/SQL record, varray etc
→ CASE is followed by multiple WHEN clauses which are checked sequentially
→ ELSE clause is executed when all the conditions doesn't satisfy

syntax:

```
CASE selector
    WHEN expression THEN sequence_of_stmts;
    WHEN expression THEN sequence_of_stmts;
    WHEN expression THEN sequence_of_stmts;
ELSE
stmts
END CASE;
```

Ex:

```
declare
grade char;
begin
grade := &grade;
CASE grade
   when 'A' then dbms_output.put_line('excellent');
   when 'B' then dbms_output.put_line('good');
   when 'C' then dbms_output.put_line('average');
else when grade = 'c' then
  dbms_output.put_line('invalid grade');
end case;
end;
```

Trigger

→ It is a DB object which is executed automatically when we perform any DML operations

→ Trigger is divided into 3 parts:

1) Trigger Event: Any DML operation is known as trigger event

2) Trigger Restrictions: It controls trigger execution before/after DML operations

3) Trigger Action: logic/functionality of triggers

→ we can apply the triggers at Row level (for each)
It enables the trigger execution for each record inside the table
before insert after insert
before delete after delete
before update after update

→ we can apply the triggers at stmt level

→ trigger is executed only once for effected no. of records for a DML operations

NEW → It represents new values into a column (after insert, after update)

OLD → It represents old values into the column (before update, before delete)

syntax

```
create or replace trigger trigger-name
[before/after] instead of
[insert/delete/update] of column-names on table-name
FOR EACH ROW
declare
---
begin
---
end <trigger-name>;
end;
```

automatically

or as trigger event

xecution

triggers

el (for each)
each record

ted no. of

column

e column

on table_name

write a trigger to insert/update the data of emp' in upper case

create

```
create table employee (
 ename varchar2(20),
 city varchar2(10)
);
insert into employee values ('kiran', 'Delhi');
insert into employee values ('vishnu', 'gopalswamitemple');
insert into employee values ('vishal', 'SPR Hoba');

create or replace trigger trig_upper
before
insert or update on employee
for each row
begin
:new.ename:=upper(:new.ename);
:new.city:=upper(:new.city);
 end trig_upper;
/
```

write a trigger to maintain audit transactions/emp details when trigger executed.

```
create table emp_audit_new (
 ename varchar2(20),
 city varchar2(20),
 opname varchar2(20),
 opdt date;
 );
```

if inserting then
optime = '
insert into emp-audit-new values (eno, city, optime,
sysdate);
→ we can fire trigger continuously 12 times

14/2/25

a) Triggers enables to enforce data integrity constraints
A) True

a) which stmt is used to remove a trigger DROP

a) The variables in the triggers are declared by using '@'

→ we can enable & disable the triggers

a) which prefixes are available to oracle triggle
A) :new only, :old only

a) How many types of recursion occurs when Recursive triggers enable
A) 2

→ there are 2 types
1) Direct Recursion ~~triggers~~
2) InDirect Recursion ~~trigger~~

→ 2 types of triggers are present in sql server
1) cluster 2) Non-cluster

Indexes

→ Index is a table like object which maintains ordered data of the column physically
→ It reduces no. of comparisons.
→ Indexes ~~are~~ contain 2 parts: virtual column (ROWID (Pseudo columns))
1) Data part 2) Address part
2) Address part
→ It contains physical address of each record
→ we can access the data but we can't
→ ROWID is a combination of Filero, data block no, record no Ex: 555.000.000
fno bno rno

→ Initially the block no & record no will be "zero"

B-Tree Index

→ It is also called as Balanced tree
→ B-Tree Index is on ordered list of values divided into ranges

Types of Indexes

1) simple Index: It is created on a table on single column

syntax: create index index_name on table_name(colname);
ex. create index ird-sal on emp(sal);

2) composite Index: It is created on multiple columns of a table.

syntax: create index idx_name on tbl_name (col1, col2...);

ex: create index ~~ird-sal~~ prod-ord on products (pid, name);

3) function Based Index:

→ If we specify any calculations / functions on table columns

syntax:

create index index_name on
table_name (columnname+100, function col);

ex: create index sal-ind on emp ((0.20*sal), initcap (sal));

4) Unique Index: If an index is created on unique col.

ex:

syntax:

create unique index index_name on table_name (col);

ex: create unique index em-ind on emp(comm);

5) Reverse key Index: In this index the search criteria is from rightmost leaf to ~~left~~ left

Syntax:

create index index_name on table_name (col-name)
REVERSE;

6) Bit map Index: using this index if we have very less no. of different values in a column
ex; gender, emp job status

syntax:
create bitmap index index_name on table_name (colname)
PNEPRE;

→ the indexes that we are created stored in USER_INDEXES.

→ to delete the index:
syntax: DROP INDEX index_name;

→ global & local pair of indexes are created when there is a partition in a table
global → It maintains one to many relationships
local → It maintains one to one relationship

→ the set of possible values for a given attributes is set to be "Domain"

→ A many to many relationship b/w two entities usually reserved in 1 table

→ ('Helloworld', 2, 3) → cll

→ what does Atomicity property signifies
the entire transactions takes place at once or doesn't happen at all.

→ candidate key can have the null values

15/2/25

Generalization & specialization

→ It is used to combine two/more low level entities into higher level entities based upon the common attributes so that we can reduce redudancy by generalizing common attribute into parent attribute
→ It is a bottom-up approach in which 2/more low level entities can be generalized into high level entities

Ex : entities
car, truck, Bus
Vehicle_id, board, model

Specialization
→ It is a process of dividing an entity into sub entities based on its characteristics, it is a top-down approach
Ex : Employee — High level entity divided into
programmer / Developer / Tester — low level entity

Aggregation
→ In ER model it is not capable of representing relationship b/w entities
→ Aggregation is an abstraction through which we can represent relationships as high level entities
→ Relationship b/w Relationship

Generalization → Merging low level entities into high level entities

Specialization → Dividing high level entities into low level entities

Aggregate : Treating a relationship as an entity association

ACID Properties
→ Acid stands for Atomicity, Consistency, Isolation, Durability
→ We use acid properties in DBMS to ensure reliable an database transaction

Atomicity (all/nothing)
→ Either entire transaction takes place at once / nothing no transaction at all
→ there is no transaction partially

Abort :- If a transaction abort, changes made to the DB are not visible

Commit :- If a transaction commit, changes made to DB are visible

Consistency (valid state)

→ consistency means DB remains in valid state before & after transaction

→ Any transaction made in DB while moving from one consistence date to another consistence state Maintains the rules & constraints

→ A bank total balance before & after transaction must remain same i.e., no money should be lost

Isolation

→ this property ensures that multiple transaction can occur concurrently without leading to the inconsis-tency of DB state

Ex: A 2 user are booking movie tickets Isolation prevents booking 2 tickets by handling transaction equally.

Durability property	Definition	Example
Atomicity	Transaction all (or) nothing	Money transfers either fully / roll backs
consistency	DB remains consistent Before & after transaction	Bank balance Before = bank balance after
Durability	Commited transactions are permanent	Flight tickets are confirmed even system crashes
Isolation	prevents double ticket booking	

ade to the

nade to DB

ate before

ng from
state

saction
e lost

ction can
inconsis

ndling

le
ansfers
ly / roll backs

lance Before =
lance after

tickets are
ed even
coaches

→ A manipulation command that extracts some records from a file <u>select</u>

→ ER model is top-down approach

→ Data items grouped together for storage purpose <u>record</u>

→ the relationship associating the weak entity set with the identifying set is called <u>identifying relationship</u>

→ the relationship that relates the weak entity set to its owner. we cannot identify without the owner

→ The Generalization constraints that states that same entity may belong to more than one lower level entity set with single level generalization is known as

three types of Generalization (specialization
1) stimulus
2) maintanance
3) Response

→ Generalization constraints means applying rules for governing relationship b/w super types / high types & sub types / low types

→ they specify which types / entity belong to specific subset

<u>Rules</u>

1) Determine which entity can be member of low level entity set
Ex: condition define, user define

2) whether / not entity belongs to entity belongs to more than one / low level entity
Ex: Disjoint, Overlapping

3) whether / not entity in high level must belongs to atleast in one low level entities
Ex: Total Generalization & partial generalization.