# ORACLE

→ ORACLE is a database, and also known as Relational Data Base Management System (RDBMS)

→ RDBMS is a software and it is useful to store data, access data and to manage data.

→ Examples of RDBMS are:

1) SQL server
2) DB2
3) MySQL
4) Postgre SQL
5) RDS
6) Snowflake etc.

**DATA** We store data for future reference in one place which is called database.

Data is defined as the collection of Raw facts.

**INFORMATION**

Processed data is known as Information.

**DATABASE**

It is a software which stores ~~data~~ manages the collection of ~~all~~ information of all objects at one place.

**DBMS**

It is a database along with data management services like

i) Adding Data
ii) change Data
iii) Delete Data
iv) Authentication
v) security

→ Data generating from business
→ Business consists of objects.
→ Object is also known as Entity.

**RDBMS**

→ It is a collection of information of all related objects within the business.

→ Author of RDBMS is E.F. CODD.

→ In any RDBMS, data is stored in the form of tables.

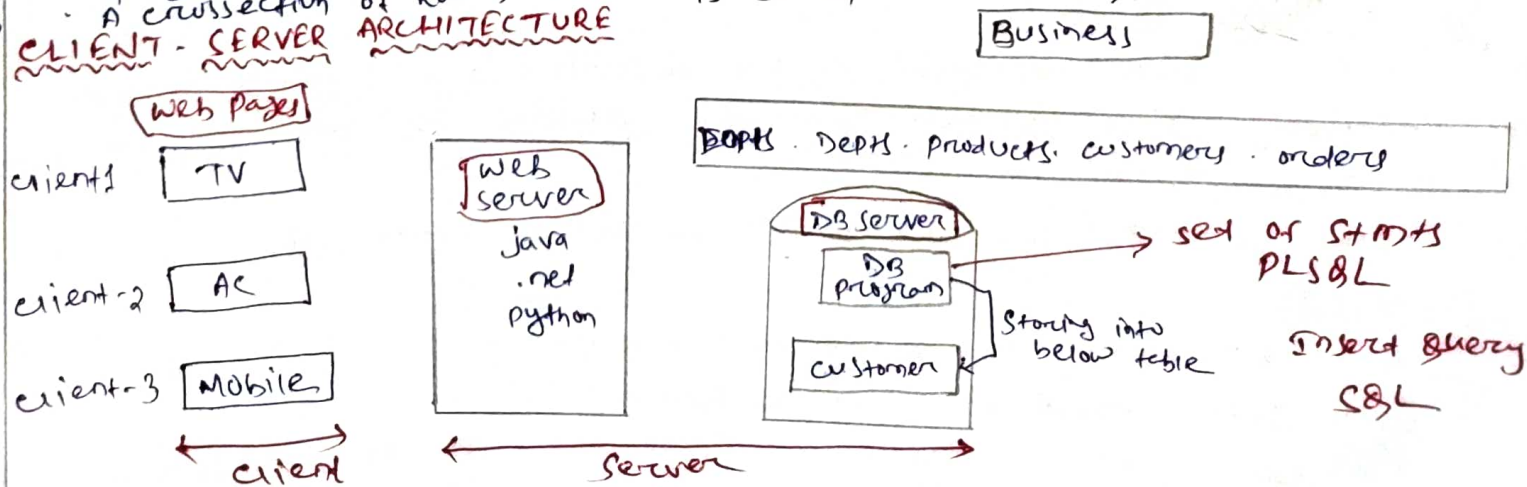
→ A table is a collection of Rows and Columns.

→ A Row is known as "Record". ROW means collection of columns.

→ A column is known as "Field".

→ A crossection of ROW & column is called TUPPLE (cell).

**CLIENT - SERVER ARCHITECTURE**



→ Information is generated from business.
→ Business is a collection of objects (Emps, Depts., Products, Customers, orders)
→ Object is also known as Entity.

## SQL

→ SQL stands for Structured Query Language
→ It is a collection of commands, keywords and syntax.
→ SQL is useful to communicate with database by writing statements (Queries).

## SQL COMMANDS

31.10.23

SQL commands are divided into 5 categories

(1) DDL commands
(2) DML commands
(3) DRL commands
(4) DCL commands
(5) TCL commands.

## DDL commands

→ DDL stands for Data Definition Language commands.
→ CREATE, ALTER, DROP, TRUNCATE, RENAME are DDL commands.

## DML commands

→ It stands for Data Manipulation Language commands.
→ INSERT, UPDATE, DELETE are DML commands.

## DRL commands

→ It stands for Data Retrieval Language commands.
→ SELECT is DRL command.

## DCL commands

→ It stands for Data Control Language commands.
→ GRANT, REVOKE are DCL commands.

## TCL commands

→ It stands for Transaction control Language commands.
→ COMMIT, ROLLBACK, SAVEPOINT are TCL commands.

## SQL * PLUS:

→ It is a client interface between developer and database server.
→ Through this we can connect to database by submitting username and password.
→ Once connected, we can write queries and programs.

## Properties

→ At a time we can execute only one query.
→ Each query must end with a semicolon (;).
→ We can write queries in any case i.e. uppercase or lowercase

SQL queries are ANSI (American National Standards Institute) standard.

## DATA MODEL

Data model is a diagram.

Based on data model, we can create table and define relations.

Data model contains 3 components

(1) Entity
(2) Property
(3) Relation

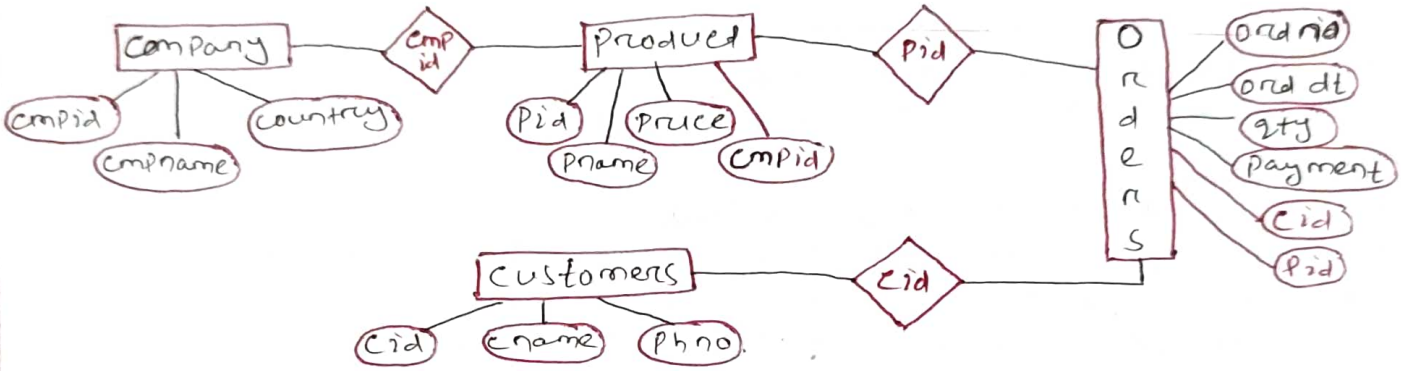
> Based on data model, we can create tables, insert columns, manage relations.

Entity is represented with rectangle (▭).

Property is represented with ellipse (⬭).

Relation is represented with Rhombus (◇).

Ex: An e-commerce (flipkart) website which sells products to the customers. The products are manufactured by some companies. Customers orders products based on their requirements.




From the above diagram, we can consider entity names as table names and property names as column names.

## INSTALLATION OF ORACLE - 11G

Extract the zip file.

Open the folder and check for setup inside disk1.

Right click on setup and run as administrator.

Click on next, next.... It will ask to set a password. Enter the password (manager) and click next.

Finally it will complete installation & click on finish.

Go to start → search for Oracle 11g folder in all apps → click on Sql command line. A command prompt like screen will open with name as Sql * plus.

Type connect & press enter. It will ask for username (system) and password (manager)

Then create an user by typing

```
create user debashis identified by welcome;
```

Then grant permission by typing

```
grant resource, connect, create session to debashis;
```

→ Then again connect by using the new username and password as username: debashis, password: welcome.

## DATATYPES

→ Datatype is a predefined name given by ORACLE company.
→ Each datatype name represents any one type of data.
→ Based on datatype, the physical memory is allocated for each value in a table.
→ Data is divided into 4 categories.
(1) Character Data
(2) Number Data
(3) Date Data
(4) Binary Data

## Character Data

→ It contains alphabets, digits and symbols.
→ character data is of 2 types:
(i) Fixed size
(ii) Variable size
→ For fixed size we use CHAR(SIZE)
→ For variable size we use VARCHAR2(SIZE)

## Number Data

→ It is used to store integer values.
→ It can store whole number as well as decimal values.
→ To store whole number, we use NUMBER(P)
→ To store decimal values, we use NUMBER(P,S).
P = Precision, S = Scale (use to store decimal values after decimal point as of S).
NUMBER(5) → Supports upto 5 digits of integer data.
Ex: -99999 to 99999
NUMBER(7,2) → Supports upto 5 digits of integer data with 2 digits of decimal data.
Ex: -99999.99 to 99999.99

## Date Data

→ It is used to store data values, month value, year value.
→ To store date data we use DATE
→ Oracle has predefined date format as follows:
DD-MON-YY
DD - Digits of date
MON - First 3 characters of month name
YY - Last 2 digits of year.

## Binary Data

→ These data types are useful to store binary data like images, thumb impressions, logos, video, and audio files.

→ We use RAW(), LONGRAW(), BLOB for storing binary data.

RAW → 2KB

LONGRAW → 2 GB

BLOB → 4 GB

## CREATE

→ This syntax is used to create a table.

→ Syntax:

```
create table tablename
(
 colname  datatype (size),
 colname  datatype (size),
          :
 colname  datatype (size)
);
```

## Naming Rules

→ Table name / column name must begin with alphabet.

→ Within the name we can use digits and symbols like @, $, #, and _.

→ Names are not case sensitive.

→ Don't use duplicate names.

→ Don't use space within the name.

→ Don't use keywords as names.

① Write a query to create company table with columns cmpid, cmpname, country.

→
```
create table company
(
 cmpid varchar2(10),
 cmpname varchar2(14),
 country varchar2(14)
);
```

② Write a query to create product table with columns pid, pname, price, mfgdt, warranty.

```
create table product
(
 Pid varchar2(10),
 Pname varchar2(14),
 Price number(7,2),
 mfgdt date
 warranty
);
```

## INSERT

→ It is used to insert a new record into a table.

→ Syntax: Insert into tablename values (val1, val2, .....);

Rules: Insert into tablename (col1, col2, col3) values (val, val2, val3);

→ Number of columns in the table and number of values in the query should be same.

→ Value datatypes & column datatypes should be same.

→ Character & data values should be in single quotes (' ').

① Insert the following data into the company table.

| cmpid | cmpname | company |
|---|---|---|
| cmp1 | Wipro | India |
| cmp2 | Sony | Japan |
| cmp3 | Samsung | Korea |
| cmp4 | Tata | India |

NULL value is a missed value. It is not equal to zero or space or other null values.

→ Insert company values ('cmp1', 'Wipro', 'India');
Insert company values ('cmp2', 'Sony', 'Japan');
Insert company values ('cmp3', 'Samsung', 'Korea');
Insert company values ('cmp4', 'Tata', 'India');

## SELECT

→ It is used to display data from table or view.

→ Syntax: Select * from tablename;
Select colname, colname, ..... from tablename;

① Display company names from company table.
Select cmpname from company;

② Display company names and their countries from company table.
Select cmpname, country from company;

③ Display all details from company table.
Select * from company;

NOTE:

④ How to display table structure?

→ describe tablename;
or
desc tablename;

⑤ How to display list of table names for an user?

→ Select * from tab;

→ Tab means table space, which contains list of tables created under current schema or user.

⑥ How to insert same record as previous again?

→ / → for same previous record insert.

⑦ How to clear the screen?

→ cl scr;

(7) How to check the current username?

→ show user;

(8) How to edit a query?

→ ed → edit
After editing → File → save.
After saving → file → exit.
For executing → / → Enter.

9.11.23

## DISTINCT CLAUSE

→ It is used to display unique values from a column or unique records from a table.

→ Syntax: select distinct * from tablename;
(or)
select distinct col1, col2, ... from tablename;

→ Ex:

① Display department numbers without duplicates. (colname → deptno, tablename → emp)
→ select distinct deptno from emp;

② Display employee designations without duplicates. (colname → job, tablename → emp)
→ select distinct job from emp;

③ Display customer names.
→ select cname from customer;

④ Display customer names without duplicates.
→ select distinct cname from customer;

⑤ Display city names.
→ select city from customer;

⑥ From which cities you have customers?
→ select distinct city from customer;

⑦ Display customer details without duplicate records.
→ select distinct * from customer.
↑ It will show distinct combination of values.

## ORDER BY CLAUSE

→ By default table has random order data. We can display ordered values from a column by using ORDER BY clause.

→ By using order by clause we can display the column data on table table data in ascending or descending order.

→ Syntax: select colname, colname .... from tablename order by colname, colname, ..... [desc];

(or)

select * from tablename order by colname, colname .... [desc];

Ex:

① Display salaries in ascending order.
→ select sal from emp order by sal;

② Display salaries in descending order.
→ select sal from emp order by sal desc;

(3) Display customer details order by city from customer table.
-) select * from customer order by city;

(4) Display customer details based on gender from each city.
-) select * from customer order by city, gender.

(5) Get employee details based on designation order.
-) select * from employee order by desg;

(6) Display least salaried employee to the highest salaried employee from each designation.
-) select * from employee order by desg, salary.

(7) Display highest salaried employee to the lowest salaried employee from each designation.
-) select * from employee order by desg, salary desc;

## OPERATORS

→ An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulation.

→ Different types of operators are:

(1) Arithmetic operators
(2) Relational operators
(3) Range operators
(4) Logical operators.

### Arithmetic operators

→ These operators are used to calculate arithmetic expressions or arithmetic operations on values & columns.

→ Arithmetic operators are: +, -, *, /

### DUAL

→ It is a system defined table.
→ It is useful to calculate constant expressions.
→ Ex:

```
select 10+20 from dual;      => 30
select 100-30 from dual;     => 70
select 200*5 from dual;      => 1000
select 300/3 from dual;      => 100
select mod(300,7) from dual; => 6
```

(8) Display 70000 as price, 10% of price as discount & also display the payment after discount.

-)
```
select
70000 as "MRP",
(0.1 * 70000) as "Discount",
(70000 - (0.1 * 70000)) as "payment"
from dual;
```

'as' is used to give columnname to a column at view time.

(8) Display empname, salary, 5% of salary as TA, 10% of salary as DA, 50% of salary as HRA & also display total salary.

-> select
ename as 'Name',
Sal as 'Salary',
(0.05 * sal) as 'TA',
(0.1 * sal) as 'DA',
(0.5 * Sal) as 'HRA',
(Sal + (0.05 * sal) + (0.1 * Sal) + (0.5 * sal)) as 'Total Salary'
from emp;

Whenever we use columnname, then we must have to use a table name. We can't use dual table

## RELATIONAL OPERATORS

13.11.23

-> These operators are used to compare values.
-> We can write conditions on the columns. The result of a condition is a BOOLEAN VALUE i.e. either TRUE or FALSE.
-> Relational operators are : <, >, =, <=, >= , <> (Not equal to).

## WHERE clause

-> WHERE clause is used to write conditions. It will display data from conditions satisfied records only.
-> Syntax: select * from tablename where condition.
(or)
select colname1, colname2, ..... from tablename where condition.

Ex:
(1) Findout developer names from emps table.
-> select ename from emps where desg = 'developer';
(2) Findout salary of Isha:
-> select salary from emps where ename = 'isha';
(3) Display developer details.
-> select * from emps where desg = 'developer';
(4) Get employee details with salary below 4000.
-> select * from emps where sal < 4000;
(5) Get employee details except developers.
-> select * from emps where desg <> 'developer';

## RANGE OPERATORS

-> These operators are used to select data according to particular range.
-> Different types of range operators are

(i) Between — not between
(ii) In — not in
(iii) Is null — is not null
(iv) Like — not like.

## Between

→ It is used to search range of values in a column.

Syntax: select .... from tablename where colname between val1 and val2;

Ex:

① Display employee details with minimum salary 20000

→ select * from emps where salary >= 20000;

② Display employee details with minimum salary 10000 and maximum salary 50000.

→ select * from emps where salary between 10000 and 50000;

③ Display product details if it is manufactured in 2023.

→ select * from product
Where mfg between '01-jan-2023' and '31-dec-2023';

④ Display employee details with salary below 10000 & above 50000.

→ select * from emps where salary not between 10000 and 50000.

## IN

→ It is used to search multiple values in a column.

Syntax: select .... from tablename where colname in (val1, val2, ....);

Ex:

① Display manager details

→ select * from emps where desg = 'manager';

② Display manager and clerk details.

→ select * from emps where desg in ('manager', 'clerk');

③ Display customer from the city hyderabad, bangalore, delhi, and order the customer names from each gender in each city.

→ Select * from customer
where city in ('hyderabad', 'bangalore', 'delhi')
order by city, gender, cname;

④ Display employee details except managers, analyst and president

→ select * from where desg not in ('manager', 'analyst', 'president');

## IS NULL

→ It is used to search for null values in a column.

→ select .... from tablename where colname is null.

Ex:

① Display customers who did not submitted PAN number.

→ select * from customer where PANNO is null;

② Get employee details who is on the bench.

→ select * from emps where projcode is null;

③ Display product details if the product don't have warranty.

→ select * from product where warranty is null;

④ Display employee details if the employee has designation designa...
-) select * from emps where desg is not null.

LIKE

-) It is used to search for patterns.
-) Pattern means a group of characters / digits / symbols.
-) Like operator is using 2 special symbols. The symbols are _, %

_ → represent any one character.

% → represent any number of character.

-) Syntax:

Select .... from tablename where colname like 'pattern';

Ex:

① Display four character length employee names.
-) select ename from emps where ename like '____';
② Display employee names ending with 'ini'.
-) select ename from emps where ename like '%ini';
③ Get employee names if the name don't have character 'y'.
-) select ename from emps where ename like '%y%';

Tue. 11.7.23

LOGICAL OPERATORS

-) These operators are useful to search for multiple values in multiple columns i.e. we can write multiple conditions on multiple columns.
-) Syntax: select .... from tablename where cond1 AND/OR cond2 .....

AND

It returns true if all conditions are satisfied in a record.

OR

It returns true if any one of the conditions is satisfied in a record.

Ex:

① Display manager details with minimum salary 2500 ?
-) select * from emp where desg='manager' AND salary >=2500;
② Display TV or mobile details with maximum price 25000 ?
-) select * from products where pname in ('TV','Mobile') AND price <=25000;
③ Display 4 character length and 6 character length employee names ?
-) select * from emp where ename like '____' OR ename like '______'
④ Display female customers from Delhi and Male customers from Chennai ?
-) select * from customer
where (city='Delhi' AND gender='Female') OR
(city='chennai' AND gender='Male');

## UPDATE

→ Used to change old values with new values.

→ Syntax:

update tablename set coiname=value, coiname=value. . . . .
where condition;

① Ex:
Increase all employee salaries with 10%

→ update emps set salary = salary + (0.1 * salary);

② Enter employee id for the employee bhavya.

→ update emps set empid = 1944 where ename = 'bhavya';

## DELETE

→ It is used to delete records from a table.

→ Syntax: delete from tablename where conditions;

Ex:

① Delete manager details

→ delete from emps where designation = 'manager';

② Delete all employee details.

→ delete from emps;

③ Delete all names from emps.

→ update emps set ename = null;

## TCL commands

→ In any database insert, update & delete commands are known as transactions.

→ We can use these transaction by using TCL commands.

→ Different types of TCL commands are

(i) Commit
(ii) Rollback
(iii) Savepoint.

## Commit

It is used to make transactions as permanent.

## Rollback

It is used to cancel transactions

## Savepoint

→ In between transaction, we can define savepoint.

→ Savepoint is useful to control the nature of rollback.

**NOTE**

We can create a table from the existing table by using

create table tablename
as
select col1, col2, .... from oldtable;

create table empinfo
as
select ename, salary
from emps;

## DCL commands

→ These commands are useful to give the permission or cancel these permission by the DBA (Database Administrator)

→ DCL commands are

(1) Grant
(2) Revoke.

### Grant

→ It is used to give permission.

→ Syntax:

```
Grant privilages
on
dbname.username.objectname
to
dbname.username;
```

Privilages:
Command / Syntax for operation
Ex: select, insert, delete etc.

### Revoke

→ It is used to cancel the permission.

→ Syntax:

```
Revoke privilages
on
dbname.username.objectname
from
dbname.username;
```

Ex:

Suppose in a database, two user named as 'Deb' and 'Devil'. If Deb wants to access the tables from Devil then he can request to DBA & DBA will grant permission as:

```
Grant select, insert
on
Devil.emp
to
Deb;
```

After this permission Deb can access select, insert operation for Devil's table

If the DBA wants to revoke the insert permission, then he can revoke permission by

```
Revoke insert
on
Devil.emp
from
Deb;
```

After this, Deb can access only select operation, he can't access insert operation for Devil's table

## DDL commands

→ DDL commands are useful for create & modify the schema of database and its objects.

→ Different types of DDL commands are:

(i) Create
(ii) Alter
(iii) Drop
(iv) Truncate
(v) Rename

## DROP

- It is used to delete a table (data + structure).
- Syntax: `drop table tablename;`

Ex: drop table emps;

## TRUNCATE

- It is used to delete all records permanently keeping the table structure.
- Syntax: `truncate table tablename;`

Ex: truncate table emps;

## RENAME

- It is used to change the tablename.
- Syntax: `rename oldname to newname;`

Ex: rename emps to employee;

## ALTER

17.11.23

- It is used to change structure of a table by following steps
  - (i) by adding a column
  - (ii) by deleting a column
  - (iii) by changing column name
  - (iv) by changing table name
  - (v) by changing datatype & size.

**By adding a column**

```
Alter table tblname
add
(colname datatype(size),
 colname datatype(size),
    :
);
```

**By deleting a column**

```
Alter table tblname
drop
column colname;
```

**By changing column name**

```
Alter table tblname
rename
column oldname to newname;
```

**By changing table name**

```
Alter table tablename
rename
to newtblname;
```

**By changing datatype & size**

```
Alter table tblname
modify
colname datatype(size);
```

**NOTE:**

- If a table column consists of data, then we cannot change the datatype of the column.
- The size of the column can increased easily but the size should be decrease upto the highest length value in the column.

Ex:

(1) Modify empinfo table to maintain gender and phone no.

→ Alter table empinfo add (gender varchar2(10), phno number(10));

(2) Enter gender value and phno for the employee Ajay.

→ update empinfo set gender='male', phno='9877098771' where ename='ajay';

(3) Delete designation column.

→ Alter table empinfo drop column designation;

(4) Change ename column as empname.

→ Alter table empinfo rename column ename to empname.

(5) Change empinfo table name as resources.

→ Alter table empinfo rename to resources.

(6) Decrease empname column size as 10. (From 14 to 10, maximum length in col = 8).

→ Alter table resources
modify empname varchar2(10);

## CONSTRAINTS

→ Constraints are known as set of rules and constraints are useful to control invalid data into the tables.

→ constraints are divided into 3 categories

(1) Key constraints
(2) Domain constraints
(3) Referential integrity constraints.

18.11.23

## Key constraints

→ Key constraints are regulations that a DBMS uses to ensure data accuracy and consistency in a database.

→ Key constraints are of 3 types

(i) unique,
(ii) Not null,
(iii) primary key

### unique

Not allows duplicate values but allows null values.

### Not null

Not allows null values but allows duplicates.

### primary key

→ Not allow duplicates & not allows null values.

→ Within a table only one primary key is valid.

→ It is used to identify the information.

Ex:

(1) Write a query to create studi table with columns rno, sname, course, fee and phno along with constraints PK, NN, NN, NN, UK respectively.

→ create table studi(
rno number(4) primary key,
sname varchar2(20) not null,
course varchar2(10) not null,
fee number(6) not null,

```
    phno number(10) unique
    );
```

**NOTE**

→ Even after key constraints still the above table is accepting invalid values such as rno as -ve number, phno as 8 characters etc

→ To avoid this we have to go for Domain constraint.

## Domain Constraint

→ The name of the domain constraint is check.

→ It is useful to define conditions on the columns.

→ Syntax: check (condition);

Ex:

Write a query to create stud2 table with columns rno, sname, course, fee, phno with constraints PK, NN, NN, NN, UQ respectively along with below rules;

(i) Min rno is 1 and max rno is 999
(ii) Valid courses are oracle, sql, java, .net.
(iv) min fee is 5000 and max fee 10000.
(iv) phno length is 10 digits
(v) phno beginning with 6/7/8/9.

→

```
create table stud2(
rno number(4) primary key,
sname varchar2(20) not null,
course varchar2(20) not null,
fee number(6) not null,
phno number(10) ~~not~~ unique,
check (rno between 1 and 999),
check(course in('oracle','sql','java','net')),
check (fee between 5000 and 10000),
check (length(phno)=10),
check(phno like '6%' or phno like '7%' or phno like '8%' or
                                    phno like '9%')
);
```

→ This table will take the valid values & shows error for invalid values.

→ When we try to check the constraint names of the ~~table~~, we get the name as numbers which is difficult to identify

that the number is which type of constraint and applied to which column. So, we can create our user defined constraint names with the keyword 'constraint'

Ex:

Write a query to create stud3 table with above columns and constraints alongwith user defined names.

```
create table stud3(
rno number(4) constraint PK_rno_stud3 primary key,
sname varchar2(20) constraint nn_sname_stud3 not null,
course varchar2(10) constraint nn_course_stud3 not null,
fee number(6) constraint nn_fee_stud3 not null,
phno number(10) constraint uq_phno_stud3 unique,
constraint ck_rno_stud3 check(rno between 1 and 999),
constraint ck_course_stud3 check(course in('oracle','sql','java','.net')),
constraint ck_fee_stud3 check(fee between 5000 and 10000),
constraint ck_phno_length_stud3 check(length(phno)=10),
constraint ck_phno_begin_stud3 check(phno like '6%' or phno like '7%'
                                     or phno like '8%' or phno like '9%')
);
```

This table will show the constraint names as user defined names.

Query to view constraint names:

```
select constraint_name from user_constraints
where table_name = 'STUD3';
```

## Referential Integrity Constraint

20.11.23

It is used to define relation between tables. Then we can fetch complete & accurate data.

By using parent table primary key column, we can define foreign key column in the child table.

Syntax: `colname datatype(size) REFERENCES parent_tbl(PK_col);`

(or)

```
colname datatype(size), constraint <fk_name> FOREIGN KEY(colname)
references parent_tbl(PK_colname);
```

Q) What is the use of 'REFERENCES' keyword?

→ It is used to define a column as a foreign key column.

Q) What is the use of 'FOREIGN KEY' keyword?

→ It is used to define foreign key constraint with user-defined name.

Q) What is foreign key column?

→ Foreign key column is defined from primary key column.

→ Foreign key column accepts values from primary key column.

Foreign key also allows duplicate & null values.

Ex:

Consider the tables company and product.

```
create table company(
cmpid varchar2(10) constraint PK_cmpid_company primary key,
cmpname varchar2(20) constraint nn_cmpname_company not null,
country varchar2(20) constraint nn_country_company not null
);
insert into company values ('cmp1', 'sony', 'japan');
insert into company values ('cmp2', 'tata', 'india');
insert into company values ('cmp3', 'wipro', 'india');
insert into company values ('cmp4', 'samsung', 'south korea');

Create table product(
Pid varchar2(10) constraint PK_Pid_Product primary key,
Pname varchar2(20) constraint nn_Pname_Product not null,
price number(10,2) constraint nn_price_product not null,
mfgdt date constraint nn_mfgdt_product not null,
Warranty varchar2(10),
cmpid varchar2(10),
constraint fk_cmpid_product foreign key (cmpid) references
                                        company(cmpid)
);

insert into product values ('P1', 'tv', 50000, '12-oct-2022', '3 years',
                            'cmp1');
insert into product values ('P25', 'oven', 15000, '25-feb-2023', '4 years', 'cmp1');
```

## Normalization & Denormalization

According to key and domain constraints we can maintain valid data in the tables.

Whenever we are accessing related information then data is not retrieved from tables. This is known as communication gap between database server and client user.

To eliminate communication gap, we can use either Denormalization or Normalization methods.

## Denormalization

Maintaining all necessary information in one big table is known as Denormalization method.

### Advantages

Communication gap is eliminated.

## Drawbacks

- Data duplicacy
- Occupy more disk space
- Data retrieval time is very long.

To decrease above drawbacks, we can apply normalization method.

## Normalization

The process of dividing big table into sub tables, until maximu data duplicity is reduced is known as Normalization.

### 1st NF

Dividing big table into sub tables based on repeated groups of dat

### 2nd NF

Eliminate duplicate records and defining primary keys in the above tables.

### 3rd NF

- We can define relation between the tables.
- Consider primary key of parent table and define foreign key column in child table.
- We can define foreign key column, by using Referential Integrity Constant.

## Advantages

- No communication gap
- Duplicity is reduced
- Occupy less disk space.
- Data fetch time is reduced.

21.11.23

## JOINS

Joins are useful to display data from multiple tables (Parent & child tables).

Joins are of 4 types

(1) Cross Join
(2) Equi Join
(3) Self Join
(4) Outer Join

## Cross Join

It will display all possible combinations i.e. each record from first table is mapping with all records in the second table.

Syntax: `Select .... from t1, t2, t3, .... ;`

Ex: Display product details alongwith company details.?

→ Select * from product, company ;

(or)

Select p.*, c.* from product p, comany c;

Here p, c known as alias name for the tables product and company.

NOTE:

→ In the cross join output we have invalid combinations also.

→ We can stop invalid combinations by using Equi join.

Equi Join

→ It will display only matched data from multiple tables, based on join condition.

→ Writing a condition on common column is known as join condition.

→ Syntax:

```
Select .... from t1, t2, t3, .....
Where t1.colname = t2.colname
and t2.colname = t3.colname
and .....
```

Ex:

① Display product details alongwith company details.

→ Select p.*, c.* from product p, company c where p.cmpid = c.cmpid ;

② Display tv and laptop details alongwith company details.

→ Select p.*, c.* from product p, company c
Where p.pname in ('tv', 'laptop')
and
p.cmpid = c.cmpid.

③ Display product details alongwith company name, if the product manufactured in India.

→ Select p.*, c.cmpname from product p, company c
Where c.country = 'India'
and p.cmpid = c.cmpid ;

④ Display employee details and their department details.

→ Select e.*, d.* from emp e, dept d
where e.deptno = d.deptno ;

⑤ Display manage details alongwith department details.

→ Select e.*, d.* from emp e, dept d
Where e.job = 'MANAGER'
and e.deptno = d.deptno ;

## Inner Join

→ It will display only matched data from multiple tables like equi join.

→ Equi join is not working with outer joins.

→ Inner join is also working with outer joins.

→ Syntax:

```
select ... from t1 inner join t2
on t1.colname = t2.colname
inner join t3
on t1/t2.colname = t3.colname ....
```

Ex:

① Display oled tv details along with company details.

→
```
select P.*, c.* from product P inner join company c
on P.pname = 'oled tv'
and P.cmpid = c.cmpid;
```

② Display customer name, phno, order details, product name, price and company name, country.

→
```
select cu.cname, cu.phno, o.*, P.pname, P.price, c.cmpname, c.country
from customer cu inner join orders o on cu.cid = o.cid
inner join product P on o.Pid = P.Pid
inner join company c on P.cmpid = c.cmpid;
```

(or)

```
select cu.cname, cu.phno, o.*, P.pname, P.price, c.cmpname, c.count
from customers cu, orders o, product P, company c
where cu.cid = o.cid
and o.Pid = P.Pid
and P.cmpid = c.cmpid;
```

## Self Join

22.11.23

→ A table which is joined itself is known as self join i.e writing a join query based on single table.

→ Syntax:

```
select b.* from tbl a, tbl b
where a.col = 'value'
and a.col = b.col;
```

Ex:

① Display employee details ?

→
```
select * from emp;
```

② Display employee scott details ?

→
```
select * from emp where ename = 'scott';
```

③ Display employee details who is working like scott ?

→
```
Select b.* from emp a, emp b
Where a.ename = 'scott'  } → filter condition
and
a.job = b.job;           } → join condition.
```

④ Display employee details from an department where employee king is working ?

→
```
Select b.* from emp a, emp b
where a.ename = 'king'
and a.deptno = b.deptno;
```

⑤ Display customer details from a city where vinod is living ?

→
```
Select b.* from customers a, customers b
where a.cname = 'vinod'
and a.city = b.city;
```

⑥ Display student details who watched a movie which is watched by Isha ?

→
```
Select b.* from students a, students b
where a.sname = 'Isha'
and a.movie = b.movie;
```

## Outer Join

→ It will display all records from one table and only matched record from other table.

→ Outer joins are 3 types
- (1) Left outer join
- (2) Right outer join
- (3) Full outer join.

→ Syntax:

```
Select .... from t1 outer_join_name t2
on t1.col = t2.col
where filter_conditions;
```

outer_join_name means either left or right or full outer join

## Left outer join

It will display all records from left table and only matched records from right table.

## Right outer join

It will display all records from right table and only matched records from left table.

## Full outer join

→ It will display matched record from both tables.
→ Display unmatched records from left table.
→ Display unmatched records from right table.

Ex:

① Display all employee details and also if any employee working in any department then display his department name?

→
```
select e.*, d.name
from emp e left outer join dept d
on e.deptno = d.deptno;
```

② Display all customer details and also if any customer makes an order then display his order details.

→
```
select c.*, o.*
from customer c left outer join orders o
on c.cid = o.cid;
```

(or)

```
select c.*, o.*
from orders o right outer join customer c
on o.cid = c.cid;
```

25.11.23

## SUBQUERY

Subquery is a select query in the where clause of other select query.

We can write subqueries to display data from one table based on input value from other table.

Syntax:

```
select ... from T1
where colname operator (select colname from T2);
```

colname → common colname.

T1 = Output table

T2 = Input table

→ Subqueries are 2 types:

(1) Single-row subquery

(2) Multi-row subquery.

Single-row subquery

It will select data from single record.

Multirow subquery

It will select data from multiple records.

Ex:

① Display product details from the company 'Samsung'?

→
```
select * from product
where cmpid = (select cmpid from company where cmpname = 'samsung');
```

② Display order details of customer 'Uday'?

→
```
select * from orders
where cid = (select cid from customers where cname = 'Uday');
```

(3) Get product details ordered by 'uday'?

→ Select * from product
where pid = (select pid from orders
where cid = (select cid from customers where cname = 'uday'));

In this case = operator will check only for one value. If customer uday has more than 1 order then it will unable to view the orders. So we have to use 'in' operator.

Select * from product
where pid in (select pid from orders
where cid = (select cid from customers where cname = 'uday'));

(4) Find out the department name of employee 'KING'?

→ select dname from dept
where deptno = (select deptno from emp where ename = 'KING');

(5) Display employee details from the department 'sales'?

→ select * from emps
where dno = (select dno from dept where dname = 'sales');

## Default

→ It is useful to define a column with fixed value. We don't insert any value into default column.

→ Syntax:

```
Colname datatype(size) DEFAULT 'value';
```

## Sequence

→ It is a database object to generate sequential integers.

→ Syntax:

```
create sequence seqname
start with 'value'
increment by 'value';
```

Ex:

(1) How to access values from a sequence?

→ select seqname.nextval from dual;
select seqname.currval from dual;

(2) How to insert sequence values into a column?

→ insert into tblname values
(seqname.nextval, value, value);

Ex:

insert into custo values
(s1.nextval, 'A', 'B');

Ex:
① create a table custinfo with column cno, cname, PANno, brancheode, ifsccode.
(i) cno is automatically generated starting with 101001.
(ii) branch code is fixed as 05152.
(iii) ifsccode is fixed as SBIN0005152.

→ create sequence custseq start with 101000;

```
create table custinfo(
cno number(6),
cname varchar2(10),
PANno char(10),
Brcode char(5) default '05152',
Ifsccode char(11) default 'SBIN0005152'
);
Insert into custinfo(cno, cname, PANno)
Values(custseq.nextval, 'SWATHI', 'URED8765NN');
Insert into custinfo(cno, cname, PANno)
Values(custseq.nextval, 'Hasritha', 'ABCD1234NN');
```

## PSEUDO COLUMNS

26.11.23

→ In ORACLE, each table is associated with two virtual columns. They are
(1) ROWNUM
(2) ROWID

## ROWNUM

→ It contains serial number for each record in the output. It is useful to display any number of records from the beginning of the number.

→ Syntax:
```
Select .... from tblname
where rownum <=n;
```
n → Any number.

Ex:
① Display first 5 records from employee?
→ Select * from emp where rownum <=5;
② Display first 2 employee names?
→ Select ename from emps where rownum <=2;

## ROWID

→ It contains physical address of each records.
→ It is useful to identify any record in the table.
→ Syntax:
```
Select ..... from tblname
where rowid = 'rowid_value';
```

Ex:
① Display employee names with rowid?
→ Select ename, rowid from emps;
② Find out employee record available for a rowid/a particular rowid?
→ Select * from emps where rowid = 'rowid_value';
③ Find out employee details for rowid 'AAAE/FAAB AAALG/LAAI'?
→ Select * from emps where rowid = 'AAAE/FAABAAALG/LAAI';

(A) How to identify a record without primary key value?

→ In the absence of primary key, we can identify a record by using rowid.

## VIEW

→ View is like a table. View is created based on frequently using data.

→ View has logical data and dynamic data.

### Advantages

→ Better performance

→ Better security

→ Dynamic data / updated information.

→ Let a ~~tv~~ view created for tv. When a customer used to search tv details then tv details search among the table which take more time. If we search tv in tv-view table, then it takes less time & performance increased.

→ Let two agents A1 & A2 have some data. All their datas inserted to a table. When DBA gives permission to the agents on the table, then A1 can see his data along with A2 data & similarly for A2. But if we create view for A1 and A2 separately, then A1 can see only his own data and A2 can see only his own data. So security will increase by using view.

→ When any data added to A1 or A2 into the main table then the data will automatically updated to the view table.

### Simple view

→ It is created on single table data.

→ Syntax: `Create view viewname as select query;`

### Composite view

→ It is created based on multiple table data.

→ Syntax: `create view viewname as select with join query;`

Inline view: select query in the from keyword.
Force view: It is created without table
Materialized view: It has physical data and dynamic data.

(B) How many types of view?

→ I know only 2 types. One is simple view and another one is composite view.

### NOTE

→ As a java developer we don't need to know about inline view, force view, materialized view.

→ Developer don't have the privilage to create view. So we have to take permission from DBA to create view.

→ DBA will give permission to developer / user as

Ex:

① Create view based on 'tv' details ?

→ create view tv
as select * from product
where pname like '%.tv%';

② ② Display tv details ?

→ select * from tv; → Here tv = view name.

③ Create a view based on tv details along with company name and make in country.

→ create view tv_details
as
select p.pname, p.price, p.mfgdt, p.warranty, c.cmpname,
c.country as make_in_country from product p, company c
where pname like '%.tv%'
and p.cmpid = c.cmpid ;

④ ② Display tv details ?

→ select * from tv_details ;

NOTE

⑦ How to display created view names ?

→ select object_name from user_objects
where object_name = 'VIEW';

⑧ How to extract day, month, year from a date ?

→ select pname, mfgdt, extract (day from mfgdt) as day,
extract (month from mfgdt) as month,
extract (year from mfgdt) as year
from product;

INDEXES

→ Index is a database like a table.
→ Index is useful to search the data as much as fast.
→ Index has 2 parts i.e.
(i) Data Part
(ii) Address Part.
→ Data part contain ordered values.
→ Address part contains rowid values.
→ Table contains values in random order, but when we create index on a column, the index contains values in ascending order.
→ If the table has at leaft 15000 records and search based on some condition, then only index creation is efficien. It will save some time
→ If the above condition not satisfies and we create index, then search time is nearly same as original table, but it will take some spaces in memory.
→ That's why creation of index should be done only when it is necessary.

## Simple Index

→ It is created on single column.

→ Syntax:
```
create index indexname
on tblname(colname);
```

## Composite Index

→ It is created based on multiple columns.

→ Syntax:
```
create index indexname
on tblname (colname, colname, ....);
```

Ex:

① Create an index on salary column ?

→
```
create index idxsal
on emps (salary);
```

② Create an index based on product name & price ?

→
```
create index idxprod
on Product (Pname, price);
```

③ How to delete an index ?

→
```
Drop index indexname;
```

## SQL FUNCTIONS

→ ORACLE is providing a set of predefined functions. Each function is useful to perform a task.

→ Functions are divided into two categories:

(i) Group or Aggregate Functions

(ii) Scalar or Single Row Functions.

Group or Aggregate functions: Execute on set of values & display single output value.

Scalar or Single row functions: Execute on set of values and display set of output values.

## Aggregate Functions

Some examples of aggregate functions are as follows:

(1) Sum(colname)

Display sum of values from given column.

(2) AVG(colname)

Display average value from given column.

(3) MIN(colname)

Display least value from given column.

(4) MAX (colname)

Display highest values from given column.

(5) COUNT (colname)

Display number of not null values from given column.

(6) COUNT (*)

Display number of records from given table.

Ex:

① Find out total salary ?

→ select sum(sal) from emp;

② Find out total salary and number of managers ?

→ select sum(sal) as totalsal, count(*) as ecount
from emp where job = 'MANAGER';

③ Find out average salary for the salesman ?

→ select avg(sal) as avgsal from emp
where job = 'SALESMAN';

④ Find out minimum salary of a clerk ?

→ select min(sal) from emp
where job = 'CLERK';

⑤ Find out highest salary of all employees ?

→ select max(sal) from emp;

⑥ Find out total number of employees and number of employees with commission and number of employees without commission ?

→ select count(*) as totalemp,
count(comm) as with_comm,
count(*) - count(comm) as no_comm from emp;

⑦ Find out number of products and total investment for the company 'samsung' ?

→ select count(P.Pname) as prdcnt,
sum(P.cost) as investment from product_dtls P
where comp_code = (select comp_code from comp_dtls
where comp_name = 'SAMSUNG');