DBMS (Database Management System)

→ DBMS consists of two parts – Database and Management System.

→ collection of data = DBMS

→ A database management system is a computerized record keeping system. It is a repository or container for collecting data files.

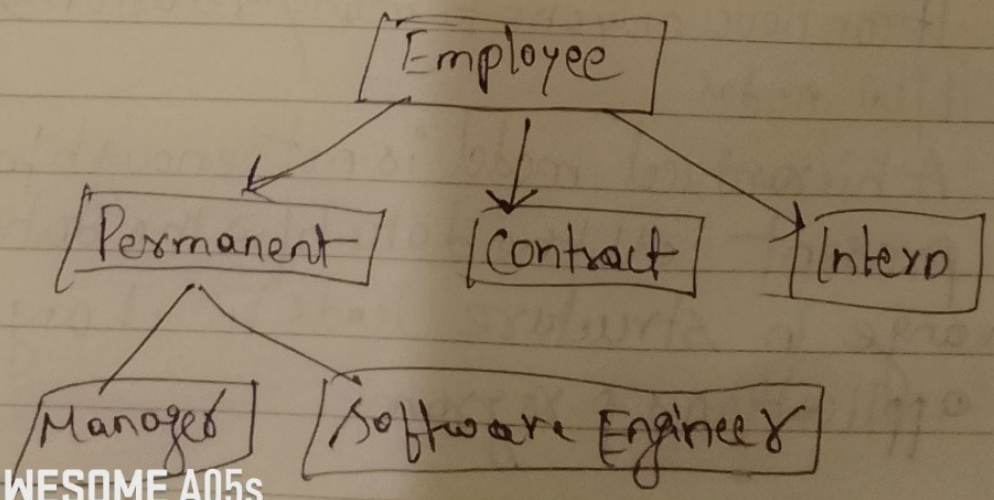
→ These are systems to store, retrieve or sometimes manipulate data.

→ Developed to handled a large amount of data.

→ DBMS helps in
- (a) controlling of Redundancy
- (b) Improved Data sharing
- (c) Storing Correct data
- (d) Maintaining the security of data
- (e) Implementing data consistency.

→ Hierarchical data Model

Data is organised into a tree like structure



Shot on AWESOME A05s

- Uses a parent child relationship
- Data is stored as records and records are connected.

- A record is a collection of fields, with each field containing a single value. One record represents all the information about one object.

- works well where the data contains nested information

Hierarchical data Model - Application

- works well with linear type data storage such as tapes

- Ex - of Hierarchical data model
  - IBM information Management System (IMS)
  - RDM Mobile

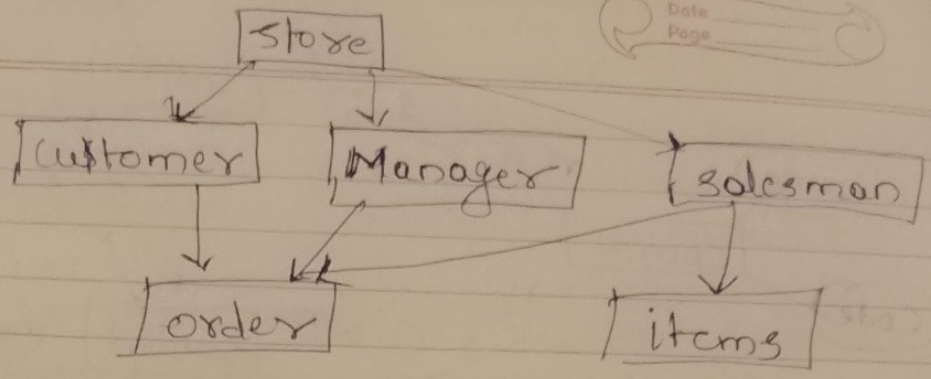
- A Hierarchical database model was widely used during the Mainframe Computer Era. Today, it is used mainly for storing file system and geographic information

Hierarchical data Model — limitations

- It can have only one to many relationships b/w nodes
- A hierarchical model is not enough to present all the data b/w the data elements
- change in structure leads to change in all application program.

Shot on AWESOME A05s

Network Data Model



- Network database model uses many to many relationship approach.
- Provide efficient way to: retrieve info and organize the data such that it can be looked at in multiple ways.

Network data model Application

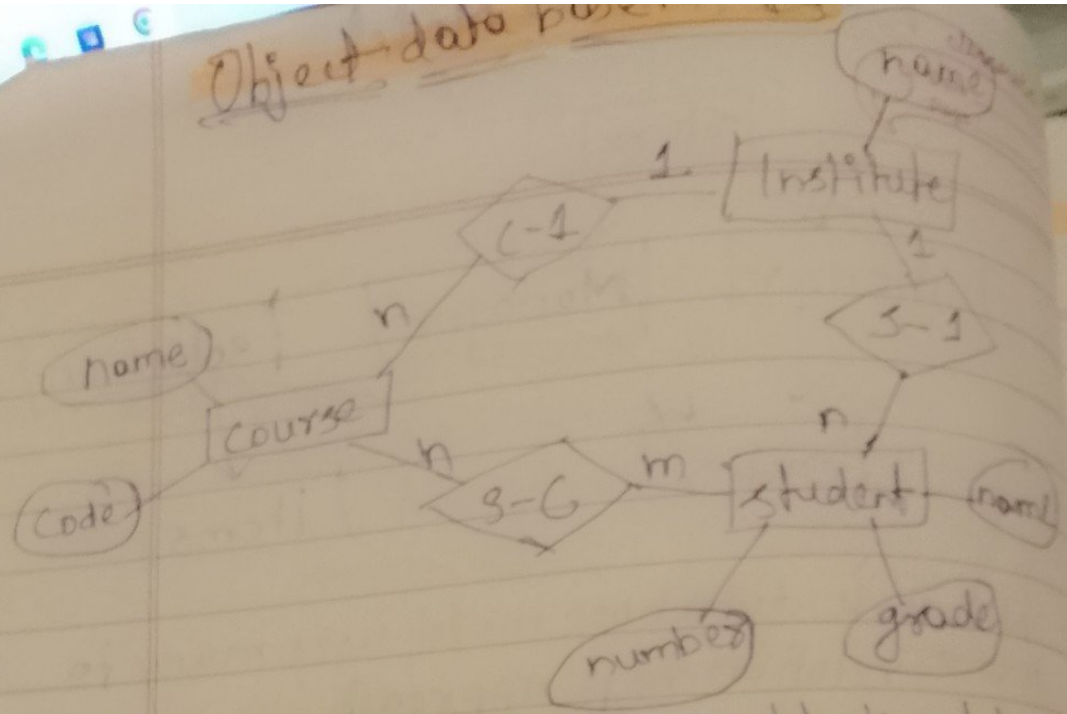
- Network data model can handle more relationship types
- Promotes database integrity
- Examples of Network data Model include
  - Integrated Data store (IDS)
  - IDMS (Integrated DBMS)
  - Raima Database Manager

Data Integrity

Database integrity refers to the correct and consistent data stored and maintained in the database.

Shot on AWESOME A05s

Object data base



- In the object based data model, the data is divided into multiple entities
- each entity has some defining attributes
- These entities are connected through relationships

Object based data model — Application:

- Object based data model provide persistent storage objects
- some objects databases can be used in multiple languages e.g., Gemstone database support C++, Smalltalk and java.
- Object based data models can store various data, for example pictures, voice and vedio including text, numbers and so on

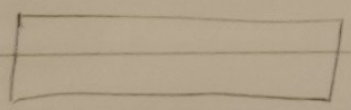
Shot on AWESOME A05s

Entity Relationship data model

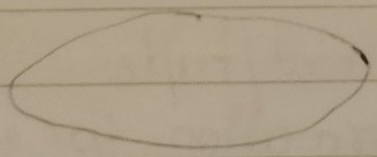
- The structure of a database is described with the help of a diagram known as the entity Relationship diagram.
- The main component of ER model are: Entity and Relationship
  - Entity: An easily recognizable living or non living real world object
    - E.g. student, customer, product
  - Relationship: An association b/w two entities.
    - E.g.: A customer buys a product

ER-Diagram

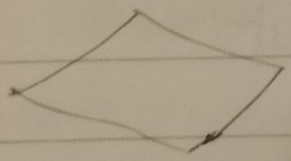
- An ER diagram displays the relationships b/w entities stored in a database
- It shows the logical structure of a database
- It is considered a best practise to design an ER diagram before implementing your database
- Notations



Entity



Attribute



Relationship.

Links 1. Attribute with entity
2. Entity(s) with relationship

Shot on AWESOME A05s

- RDMS stands for Management Systems.
- RDMS allows to store, retrieve or manipulate data, but in a more efficient
- Apart from rows and columns, the table has the following components.
  - Domain
  - Instance
  - schema
  - keys.

A database Table

- A database consists of one or more tables
- A table is the most significant component in an RDMS
- A table where all data is stored.
- A table consists of rows and columns
- Each columns represents attributes of the entity

A record in a Table

- Each row in a table is a record/tuple
- Each record is all of the information for each object, say a person or a product

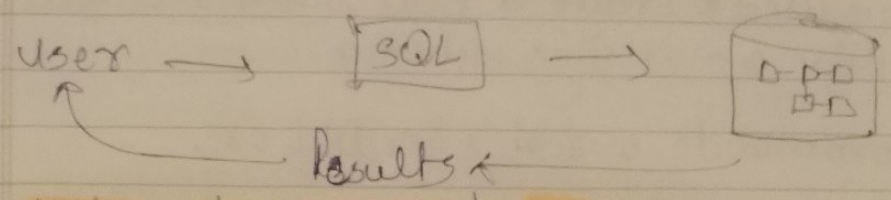
A column in a Table

- Each column in a table is an attribute
- This gives one piece of information about the attribute. For ex, the name of a customer

Shot on AWESOME A05s

DDL Commands

SQL (Structured Query language)



≡ differends commands in SQL

Data Definition Language (DDL)
Data Manipulation Language (DML)
Data Query Language (DQL)
Data control language (DCL)
Transactional control language (TCL)

SQL commands

DDL	DML	DQL	DCL	TCL
Create	Insert	Select	Grant	Commit
Alter	Update		Revoke	Roll Back
Drop	Delete			
Truncate (structure)	(content)			

DDL (Data definition Language)

- A database is a collection of many tables and a database server can hold many of these databases

Database server → Databases → Tables (defined by columns) → Rows

- Databases and tables are referred to as database objects.

Shot on AWESOME A05s

or deleting database objects is called data definition language (DDL)

- DDL is used to create a new schema as well as to modify an existing schema

Commands available in DDL are

- CREATE
- ALTER
- DROP
- TRUNCATE

Create Database

- The CREATE DATABASE statement is used to create a new SQL database.

Syntax

```
CREATE DATABASE databasename;
```

The semicolon character (;) is SQL statement terminator

- To create tables in the database you need to first select the database. use the following syntax to select the database

```
USE databasename;
```

- We will create a database called company

```
CREATE DATABASE company;
```

Shot on AWESOME A05s

e primary key example syntax
e gl_placement(

rchar(100),
cha(50),
l varchar(30),
te

(id, company));

d ⓒ Modify ⓓ Remove

olumn → Constraints
placement
cation varchar(30) not null;
nt

lacement
ginal char(1);
nt;

are used to specify
e data in a table
are to limit the
hat can go into a table
the accuracy and
e data in the table

DROP Example syntax

- alter table gl_placement
  drop column location;

- alter table gl_placement
  drop primary key; → error
  ↓
  not allowed due to fore

RENAME

Ex- rename table gl_placement to placem

TRUNCATE

Ex- truncate table gl_scores;

(29) diff b/w truncate & delete

SELECT select column list from table
= select * from gl_student
= select sname, email from gl

(3.) what is Constraints

① Check ② Default ③ NOTN

ID	Name	Age	Gender	Phon
1	Ramkumar	17	Male	4022
2	Salmankhan	19		4033
3	Meerakhan	20	Female	4022
	Saritakumari	18	Female	4088
5	Anil kapoor	19	Male	4183
NOTNULL UNIQUE		CHECK(age>=18)	NOTNULL	

create table table-name(
id INT NOT NULL UNIQUE,
name VARCHAR(50) NOT N
age INT NOT NULL CHECK
gender VARCHAR(10) NOT N
phone VARCHAR(10) NOT N
city VARCHAR(10) NOT NULL

);

DROP Example SYNTAX

- alter table gl_placement drop column location;

- alter table gl_placement drop primary key; → error → not allowed due to foreign key

RENAME

Ex: rename table gl_placement to placement;

TRUNCATE

Ex: truncate table gl_scores;

(29) diff b/w truncate & delete

SELECT select column list from table list

= select * from gl_student

= select sname, email from gl_student

(3) **what is Constraints**

① Check ② Default ③ NOT NULL ④ UNIQUE

ID	Name	Age	Gender	phone	city
1	Ramkumar	17	Male	4022155	Agra
2	salman khan	19		4033244	Agra
3	Meera khan	20	Female	4022155	
	Sarita kumari	18	Female	4066899	Agra
5	Anil kapoor	19	Male	4188733	Agra
NOT NULL, UNIQUE		CHECK(age>=18)	NOT NULL	UNIQUE	DEFAULT 'Agra'

```
create Table table-name(
id INT NOT NULL UNIQUE,
name VARCHAR(50) NOT NULL,
age INT NOT NULL CHECK(age>=18),
gender VARCHAR(10) NOT NULL,
phone VARCHAR(10) NOT NULL UNIQUE,
city VARCHAR(10) NOT NULL DEFAULT 'Agra'
);
```

remember this

```
select sname, income, grad_score
from gl_student
where grad_score > 60;
```

(31) Arithmatic Operators

(32) remember this

```
select id, name, grad_score + 5 as "New
from gl_student;
```

(33) Between Operator

~~select id, sname, grad_score + 5 as "New Score"~~
~~from gl_student;~~

remember this

```
select sname, grad_score
from gl_student
where grad_score between 70 and 80;
```

(34) Lower Limit & Upper Limit

remember this

```
select sname, grad_score
from gl_student
where grad_score between 70 and 80;
```

(35) Two Conditions Put

remember this

```
select sname, grad_score, email, income
from gl_student
where grad_score > 60 and income < 100000;
```

(36) remember this Three Conditions Put

```
select sname, grad_score, dob
from gl_student
where sname = 'Rina' or sname = 'Bina' or
sname = 'Vina';
```

Shot on AWESOME A05s

(32) IN Operator : check the data is fine / not

remember this

```
select sname, email, sid
from gl_student
where sname in('vina', 'nina', 'lina');
```

= NOT IN Operator

```
SELECT column1, column2, column3, ...
FROM table_name
WHERE column_name NOT IN (value1, value2, ...);
```

~~= SQL OPERATOR~~

(33) what is wild characters.

A wild characters is used to substitute one or more characters in a string.
wild characters are used with the LIKE operator

(34) LIKE Operator

The LIKE operator is used in a where clause to search for a specified pattern in a column.

```
select sname
from gl_student
where sname like 'na';
```

remember this

remember this

```
select sname
from gl_student
where sname like '%a%';
```

remember this

```
select sname
from gl_student
where sname like ___
```

remember this

```
select sname
from gl_student
where sname '%n_';
```

'a%' starts with "a" %a Ends with a %am% Have "am" in any position
a%m starts with a and Ends with "m" _a% a in 2 position _a%s a is 2 position

(41) Union

remember this

```
select id from gl_student where grad_score,
union
select id from gl_scores
where subject = 'sql' and grade > 25;
```

(42) Union all

remember this

```
select id from gl_student where grad_score,
union all
select id from gl_scores
where subject = 'sql' and grade > 25;
```

(43) % & _ diff b/w

(44) why we use wild characters for pattern matching

(45) Application of Truncate

— AND OPERATOR —

```
SELECT column1, column2, column3, . . .
FROM table_name
WHERE condition1 AND condition2 AND condition3. . .;
```

ex — WHERE Age >= 18 AND Age <= 21

— OR OPERATOR —

```
SELECT column1, column2, column3, . . . .
FROM table_name
WHERE condition1 OR condition2 OR condition3. . .;
```

ex — WHERE Age = 18 OR Age = 21

— Both AND, OR USE EXAMPLE —

```
SELECT * FROM personal
WHERE city = "Agra" OR city = "Agra" AND gender = "M";
```

Shot on AWESOME A05s

DAY-9

(44) DISTINCT → remove duplicacy

SYNTAX & EXAMPLE

SELECT DISTINCT column1, column2, ...
FROM table-name;

~~select * from gl_~~

[select distinct subject from gl_scores;]
→ display subjects for which exam's conducted

[select DISTINCT city FROM personal;]

(45) [select Distinct
age FROM personal
order By age;]

(45) LIMIT select column1, column2, column3, ...
FROM table-name
WHERE condition
LIMIT number;

[select * from gl_student limit 2;]
→ Display first (2) rows from student table.

[SELECT * FROM personal
LIMIT 3, 3; → MHb 4 position se 3 rows display karega
SELECT * FROM personal
LIMIT 6, 3; → MHb 7 position se 3 rows display karega]

(46) ORDER BY [ascending = ASC, decending = DESC]

[select * from gl_student
order by grad_score desc;]
or asc
→ sort the students data w.r.t graduation score.

(47) [select * from gl_scores;
select * from gl_scores order by subjects;
→ sort the students data w.r.t subjects]

[select * from gl_scores;
order by subject, grade;
→ two, / both sort]

(47) Numerical functions
- round
- ceil
- floor
- greatest
- least

Shot on AWESOME A05s

- instr

select instr("Great learning", "earning");

Output 8.

select instr("Great learning", 'x');

Output 0

- substr

select substr("Great Learning", 8);

output earning

select substr("Great Learning", 7, 11);

output learning

select substr("Great Learning", 7, 5);

Output Learn.

- concat

space dene k like

SELECT id, CONCAT(name, " ", percentage)
AS Name
FROM students;

output
id Name
1. Ramkumar45
2. sarita kumar85

- replace

select replace("Data Science", "Data", "Big Data");

select replace("Data Science", "ABC", "XYZ");

BigDataScience

- trim

select trim(" Data Science ");

output Data Science

rightside & leftside spaces remove

- replace

select replace(job_id, 'CLERK', 'CLK') from employees

output PU_clk

Shot on AWESOME A05s

- select curdate()
  select CURDATE();

- select now(); → date n time both
  SELECT NOW();

select first_name, year(hire_date)
from employees
where year(hire_date) = 1996;

Display hour min, sec.

select hour(now()), minute(now()), second(now

# add 1 day to the current date
select curdate(), adddate(curdate(), interval 1 day);

— select date_format('1979-06-06', '%d-%m-%y');

(50) NULL FUNCTIONS
- isnull - ifnull - coalease

— select isnull(null);
select isnull(1/0);
select isnull('hi');
select isnull(0);

Shot on AWESOME A05s

select ifnull(commission_pct, 0) from employees;

— coolesce

select coolesce (null, null, 1, 10);
select coolesce (1, 2, null, 10);

DAY - (4)

(51) ORDER AND LIMIT Use

Select * from employees order by salary limit 3, 2;
Select * from employees order by salary limit 1;
Select * from employees order by salary limit 2, 1;
→ third lowest
Select * from employees order by salary limit 1, 1;
→ second lowest

(52) Aggregate function

① Aggregate k sath kabhi column name nahi aega, group by mein A sakta hai
② aggregate functions mein where clause use nahi hota hai

— min

select min(salary) from employees;

— max

select max (salary) from employees

— avg.

select avg (salary) from employees

Shot on AWESOME A05s

select sum(salary) fo

— count

select count(employee_id) from employee;

select count(*) from employees;
No. of rows in the table count, if it contains null will not count

total salary given to the employee whose designation has a word clerk.

select sum(salary) from employees
where job_id like '%clerk' ;

max. salary given to a employee from the group of employees who join in the year 1987

select * from employees;
select max(salary) from employees
where year(hire_date) = 1987;

(33) GROUP BY

select avg(salary), department_id
from employees
group by department_id;

Shot on AWESOME A05s

rom employee;

s;

fit contains null

whose designation

oyees

from the group
r 1987

Find avg. salary of employee per designation those who joined in 1999

```
select avg(salary), job_id
from employees
where year(hire_date)=1999
group by job_id;
```

no. of employees where avg. salary should be more than 7000

Remember this.

```
select count(*), department_id, avg(salary)
from employees
group by department_id
having avg(salary)>7000;
```

Aggregate function use hua hai isliye where use नहीं होगा condition लगाने के लिए
having use होगा condition लगाने के लिए

Remember this.

```
select count(*), department_id, avg(salary)
from employees
group by department_id
having avg(salary)>7000;
```

Display the no. of employees per department with letter 'a' in their first name, show the output for those departments where at least 2 such employee exists.

Remember this

```
select count(*), department_id
from employees
where first_name like '%a%'
group by department_id
having count(*)>=3;
```

Remember this

```
select avg(salary), department_id
from employees
where year(hire_date)=1999
group by department_id
having avg(salary)<8000;
```

Shot on AWESOME A05s

Remember this.

```
select avg(salary), department_id
from employees
where year(hire_date) = 1999
group by department_id
having avg(salary) < 8000
order by avg(salary) desc;
```

~~JOINS~~ Sequence of clauses:

1. from table, list of tables (multiple tables)
2. where condition <, > in b/w like, AND, OR
3. group by column
4. having condition, aggregate func^n
5. ~~order by select~~ +, -, *
6. order by
7. limit

* Every aggregate functions ignores null, never print null in the output.

→ Create, use, desc
→ Insert multiple records
→ Add, modify, Remove, drop, truncate, delete
→ Constraints - check, default, Unique, NOT NULL
→ Arithmatic Operators, Between Operators, In Operator
→ Lower and upper limit, AND, OR
→ Like operator
→ Union & Union all
→ Distinct, limit
→ group by, order by, asc, desc, having by
→ Numerical func^n - ceil, floor, round, greatest, least
→ String func^n - upper, lower, concat, char length, instr, substr, concat, replace, trim
→ Date Functions curdate, Now

Shot on AWESOME A05s

DAY 5 JOINS

1. INNER JOIN
2. OUTER JOIN
3. SELF JOIN

1. Joins will be used to read the data from multiple tasks
2. Two tables are combined using cross join
3. each row of first table will be matched with each row of second table.
4. Assume table1 (m rows) cross join table2 has n rows then table1 cross join table2 generates m*n rows
5. when 2 tables having nothing in common - use cross join churn_details.

— cross join —

```
SELECT columns
FROM table1
CROSS JOIN table2
```

Basic SYNTAX

[Select * from employees cross join departments;]

[select * from personal CROSS JOIN city]

for selected columns

```
SELECT p.id, p.name, c.cityname
FROM personal p CROSS JOIN city c;
```

inner join

```
Select columns
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

[select * from employees inner join department on employee.department_id = departments.department_id;]

```
SELECT * FROM personal INNER JOIN CITY
ON personal.city = city.cid;
```

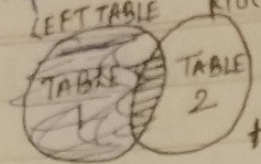
another way by using alias

```
SELECT * FROM personal p INNER JOIN city c
ON p.city = c.cid;
```

→ Aggregate function min, max, avg, sum, count

→ Case function.

[Select * from employees e left join department
on e.department_id = d.department_id

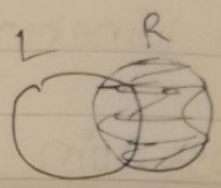


The left join returns all from the left table (table 1) the matched records from the right table

```
SELECT columns
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

common value

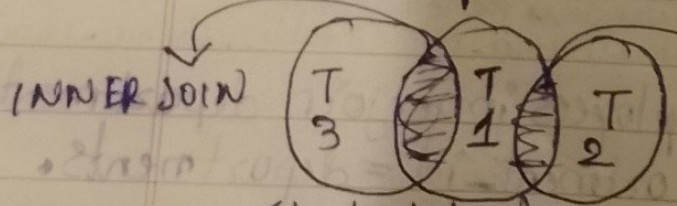
[SELECT * FROM personal LEFT JOIN city
ON personal.city = city.cid;]



— Right Join —

select * from employees e right join department
on e.department_id = d.department_id
union
select * from employees e right join department
on e.department_id = d.department_id;

JOIN Multiple tables | Yahoo Baba. | Yahoo



Student table

id	Name	Age	Course	city
1	Ramkumar	19	1	1
2	Salmankhan	18	3	2
3	Meera khan	19	1	1
4	Sarita kumari	21	2	3

join

city table

cid	city
1	Agra
2	Bhopal
3	Delhi
4	Noida

join

courses table

cid	course
1	Btech
2	BCA
3	[illegible]

Shot on AWESOME A05s

— Natural Join — // inner join

Natural join is a inner join which detects the common column between the tables, match the data and creates the join output automatically

— natural join by default equi(=) join.

```
select * from regions r join countries c
on r.region_id = c.region_id;
select * from regions natural join countries;
select * from employees natural join departments;
```

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

} — Multiple Join Syntax.

(Primary key)

```
select* FROM
personal p INNER JOIN city c
ON p.city = c.cid
INNER JOIN courses cr
ON p.courses = cr.courses_id;
```

~~Self Join~~

{ cours_id, course_name

output

id	name	m/o	age	gender	city	courses	cid	cityname

FULL JOIN

```
select* from employees e left join department d
on e.department_id = d.department_id
Union
select * from employees e right join departments d
on e.department_id = d.department_id;
```

...join returns all records from the left table (table1) ...ed records from the right ...

...2.column_name;

...JOIN city

R

...t join department d ...artment_id

...join departments d ...rtment_id;

Babar. Youtube

...IN

city table

	city
	Agra
	Bhopal
	Delhi
	noida

Shot on AWESOME A05s

→ Diff b/w windows
→ diff b/w row_number, rank & dense_rank
→ diff b/w union and union all.
→ why we use wild characters for pattern matching
→ ACID Properties. ✓

= A wild character is used to sub... one or more characters in a string. wildcard characters are used with LIKE operator. The LIKE operator is used to a where clause to search for a specified pattern in a column

= ACID (Atomicity) = The entire transaction take place at once or doesn't happen at all. There is no middle

(Consistency) This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database.

(Isolation) This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occuring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed.

(Durability) This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and persist even if sys... failure

Shot on AWESOME A05s

select count(*) as emp, (select count(*) from departments) as dept from employees;

output

emp	dept
107	27

Dual (Data Dictionary)

select 60+90 as sum from dual;

output

sum
150

dual kaise imaginary table form kar dega basis. जैसे sum 150 form किया

	outputs
select '1'+1 from dual;	2
select '1'+'1' from dual;	2
select 'namrata'+1 from dual;	1
select 'namrata123'+1 from dual;	1
select '2namrata123'+1 from dual;	3

do not depended — innerval outer depend hai

Non Correlated — **SUBQUERIES** — Correlated

जिसकी salary valli की salary से बड़ा है

select first_name, salary from employees where salary=(select first_name, salary from employees where first_name='Valli');

Output

first_name	salary
Bruce	4800
David	4800
Valli	4800

Shot on AWESOME A05s

select (select (select (select (select (salary

salary = valli , jobid='Bruce' -- first job-id

Q# select first_name, salary from employees
where salary = (select salary
from employees where first_name='valli') and
job_id = (select job-id
from employees where first_name='bruce');

Output

first_name	salary
Bruce	4800
David	4800
Valli	4800

--- department_name where nobody is working

select department_name from department
where department_id not in (select department_id
from employees);

department_name (output)

admin
Marketing
Purchusing
Human Resources
Shipping
IT
Public Relations

salary = valli but valli print nahi hona chahiye

select first_name, salary from employees
where salary = (select salary
from employees where first_name = 'valli') and
first_name != 'valli';

first_name	salary
Bruce	4800.00
David	4800.00

Shot on AWESOME A05s

select first_name, salary from employees where salary in (select salary from employees where first_name = 'John');

~~[illegible] salary John ke barabar hai~~

select first_name, salary from employees where salary > any (select salary from employees where first_name='John');

John तीन है तो 1400 2700 8200
2700 वाला हटा देगा 2700 वाले हटा रहा
Not clear
---> greater than the smallest

select first_name, salary from employees where salary < any (select salary from employees where first_name='John');

1400 छोड़ देगा 2700, 8200 print होगा.

select first_name, salary from employees where salary > (select min(salary) from employees where first_name='John');

> Any = > min
< Any = < max
> All = > Max
< All = < min

Shot on AWESOME A05s

पहले Tag एक column बनाई है, उसमें मतलब आगे manger likha, clerk के आगे CLERK likha, others के आगे others likha आएगा, उनको count कर कर के बताना है।
Tag column मे group by करके count लगाना है।

```
select tag, count(*) from (
select job_id,
case
when job_id like '%mgr%' then 'Manager'
when job_id like '%clerk%' then 'clerk'
else 'Others' end as Tag from employees) t1
group by tag;
```

non-co-related

— exists ⇒ वहाँ use होता है जहाँ पे हमको check करना होता वो भी जैसे Table मे है या नहीं, exists mainly 0-False, 1-True से deal करता है, execution of code fast

```
select * from employees
where exists (select * from departments
    where department_id = 90);
```

output
अगर department_id = 90 मे कोई data होगा तो outer वाली run करेगा मतलब अगर data होगा तो employees-table print हो जाएगी, अगर data नहीं हुआ print नहीं होगा।

Day -3 DBMS-II

Window Functions

row_number(), rank(), dense_rank()

select department_id, salary,
row_number over (partition by departmentid
order by salary desc)

salary को department wise करके arrange in descending order & then after this give rollno to them, roll no department wise ही शुरू होंगे, department में 3 है तो 1, 2, 3.

2nd highest salary

select * from(
select department_id, salary,
row_number() over (partition by -
department_id order by salary desc) r2
from employees) t1 where r2=2;

Lag / Lead

Diff of hiredate in each dept. for every emp

select *, round(datediff(hire_date, pp)/365, 2)
diff from (select department_id, hire_date,
lag(hire_date)
over (partition by departmentid over by hiredate)
pp from employees) t1) t2
group by 1 order by avg(diff);

Normalisation

Normalisation is the process to eliminate data redundancy and ensure data integrity in the table. Normalisation also helps organise the data in the database

→ 1st Normal Form (1NF)

A table is referred to as being in its First Normal Form if atomicity of the table is 1. Here atomicity states that a single cell cannot hold multiple values

It must hold only a single valued attribute

→ 2nd NF

The first condition for the table in Second Normal Form is that the table to be in First Normal Form. The table should not posses partial dependency

→ 3rd NF

The first condition for the table to be in ~~Third~~ 3 Normal form is that the table has to be in 2 Normal Form. The table should not posses partial dependency. It ensures the reduction of data duplication. It is also used to achieve data integrity

Case function

LOCKING

LOCKING

- Locks are required by user transactions in a session
- SQL Engine locks objects when the transactions begin the DML statements on each record.
  - To ensure data integrity, DML statements acquire explicit locks on data records or tables
  - Multiple users in an integrated environment can see changes applied by each other
  - After the transactions are complete locks are however released
  - Lock hierarchy
    The locks are acquired at any level for different purpose of transaction

DATABASE → TABLE → PAGE → ROW

→ Every transactions executes DML statements needs to acquire Row-level locking
① Shared lock or Read lock ✓
② Exclusive lock or Write lock ✓

Shared lock or Read lock
- Shared lock is an initial lock acquired by a SELECT statement of a transaction to lock the [illegible] applying next DML statements

- shared lock is useful to avoid deadlocks in multiple transactions if they are using SELECT DML statements

- However, the other transactions can also acquire multiple shared locks (S) on the same records locked by first transaction

Recalling Note: The SELECT statement is used to choose data from a database

Recalling Note: A deadlock occurs when one process is waiting for a resource that is being given to another process.

EXCLUSIVE (X) LOCKS

- Exclusive locks are required by DML statements: DELETE, INSERT and UPDATE
- Data cannot be modified at a time by two DML statements if they then its a deadlock or transactions keeps waiting for other transactions
- Hence this lock is mandatorily acquired by any DML statement by default when executed to modify the records
- The lock cannot be shared with other transactions as they have to keep waiting until the first DML transactions completes and commits its changes to database

Note: Once the Exclusive lock (X) is acquired, other transactions gain neither of the shared (S) nor exclusive (X) locks.

ISOLATION

when multiple D
concurrently wi
isolating one tra
the other is the
of database
It is achieved c

Assume w
internet tr
transactions
second Both
money from a
The two individ
it means the
in queue to pe
other ensu
to transfer

learning Note: The
is hid
isolat

isolation
isol
This
awe
However, th
altered
constr
integr

Shot on AWESOME A05s

ISOLATION

when multiple DML statements concurrently hit the database, isolating one transaction from the other is the responsibility of database

A Atomicity
C Consistency
I Isolation
D Durability

It is achieved only by forming the queue of transactions

Ex:- Assume withdrawal of amount from ATM & internet transfer are two different transactions records occured at the same second. Both transactions are transferring money from account

The two individual transactions are isolated it means the database keeps the transactions in queue to perform one transaction after the other ensure the balance is sufficient to transfer the money

learning note: The intermediate state of a transaction is hidden from other transactions in isolation.

isolation defines how one transaction is isolated from other transactions

This done internally and users need not aware of it

However, these ISOLATION levels can be manually altered by user and can modify such constraints and still achieve data integrity and performance

Shot on AWESOME A05s

Set Transactions perform before tasks

Set transactions encapsulate the DML statements

- Encapsulate the execution of DML statements from DML statements of other transactions
- Inter-lock the underlying affected records

Learning Note: The current transaction can be made read only or read/write by its isolation level set, or be assigned to a specific roll back segment using the SET TRANSACTION statement

≡ The data Integrity is achieved by

- Restricting other transactions performing the tasks on same records.

- Maintaining the isolation of transactions with other transactions, so that other transactions wait in Queue to perform

Recalling Note: The accuracy and consistency of the data in the table are upheld by the application of data integrity

CLASSIFICATION OF ISOLATION LEVELS

ISOLATION LEVELS are classified as

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE - READ
- SERIALIZABLE

Note: [illegible] levels including READ, WRITE [illegible] transactions can also achieve isola[illegible]

Read Uncom[illegible] the very b[illegible] uncommitt[illegible] to user 2 Known as [illegible] transac[illegible] from co[illegible]

Read commi[illegible] the dirt[illegible] Committ[illegible] by the [illegible] will acq[illegible] the wr[illegible]

Repeatable [illegible] restriction[illegible] level. Th[illegible] reading [illegible] which be[illegible] user con[illegible] transactio[illegible]

Serializa[illegible] level th[illegible] to the tra[illegible] In this le[illegible] transactio[illegible]

Shot on AWESOME A05s

Read uncommitted - Read uncommitted is the very basic isolation level. Here the uncommitted data of first user will be visible to user 2. Reading uncommited data is also known as dirty reads. In this level - transactions are not logically isolated from each other.

Read committed - This level does not allow the dirty read and make sure the committed data is available to read by the user 2. In this level transaction will acquire the read or write lock for the current row.

Repeatable Read - This level provides more restriction compared to the other 2 previous level. This level acquires read locks while reading and write lock on all rows which needs to be updated. The other user can not read or write thus the transaction avoids the non repeatable read.

Serializable - This provides the highest isolation level. It provides the complete isolation effects to the transaction from other transaction. In this level concurrent execution of transaction happen to execute serially.

Shot on AWESOME A05s

	product_category	brand	product_name	price
1.	phone	Apple	iphone12Pro Max	1200
2.	phone	Apple	iphone12Pro	1100
3.	phone	Apple	iphone12	1000
4.	phone	Apple	Galaxy Z Fold3	1800

(techTFQ) (WINDOW FUNCTION)

select * from product

--- (FIRST_VALUE)

Write a query to display the most expensive product under each category (corresponding to each record)

```
select *
first_value(product_name) over (partition by
    product_category order by price desc) as
    most_exp_product
from product;
```

OUTPUT

	Product_category	brand	Product_name	Price	most_exp_product
1	Earphone	Apple	AirPods Pro	280	AirPods Pro
2	Earphone	Sony	WF-1000XM4	250	AirPods Pro
3	Earphone	Samsung	Galaxy Buds Live	220	AirPods Pro
4	Earphone	Samsung	AirPods Max	170	AirPods Pro
5	Headphone	Apple	WH-1000XM4	550	AirPods Pro
6	Headphone	Sony	Galaxy Pro	420	AirPods Pro
7	Headphone	Microsoft	XBS17	250	AirPods Pro

Exp product wise / category wise | highest value pick करेंगे

हम नई window बना देंगे जिसमें हर row में category wise most expensive name आ जाएगा

Shot on AWESOME A05s

LAST VALUE

Write a query to display the least expensive product under each category (corresponding to each record)

```
select *
first_value (product_name) over (partition by product_category
order by price desc) as most_exp_product,
last_value (product_name) over (partition by product_category
order by price desc) as least_exp_product,
from product;
```

category	brand	product_name	Price	least_exp_prod
Earphone	Apple	AirPods Pro	280	AirPods Pro
Earphone	Sony	WF-1000XMy	250	WF-1000XMy
Earphone	Samsung	Galaxy Buds Pro	220	Galaxy Buds Pro
Earphone	Samsung	Galaxy Buds Pro	170	Galaxy Buds Live

last_value window function use kiya h and least_exp_product का column galat aya kyu ki yaha पर last_value window function k sath frame clause का use करना पड़ता and use कैसे करते है वो code से bataya H ↓

```
select *
first_value (product_name)
    over (partition by product_category order by
        price desc)
    as most_exp_product,
last_value (product_name)
    over (partition by product_category order by
        price desc)
    range between unbounded preceding and
    current row)
    as least_exp_product
from product;
```

Shot on AWESOME A05s

Previous ऊपर वाले code output देखना यह पढ़ने से

For row①

Unbounded preceedings means = very first row ①
Current Row = Here current Row means also first row

So, therefore in least_exp_product column it shows Air Pods Pro

For row②

unbounded preceding means = very first Row①
Current Row = Here current Row means also Second row

So, therefore in least_exp_product column it shows WF-1000XM4

For row③

unbounded Preceding means = very first Row①
Current Row = Here current Row means also Third row

So, therefore in least_exp_product column it shows Galaxy Buds Pro.

FRAME CLAUSE

और से range between unbounded preceding and current row default ही लगा जाता है window funcⁿ last_value use करने ही।

यह problem को solve करने के लिए current row की जगह unbounded following use करना होगा।

```
select*
first_value (product_name)
    over (partition by product_category order by price desc)
    as most_exp_product,
last_value (product_name)
    over (partition by product_category order by price desc
    range between unbounded preceding and unbounded following)
    as least_exp_product
from products;
```

output →

product_category	product_name
1 Earphone	Earphone
2 Earphone	"
3 Earphone	"
4 Earphone	"

NTH VALUE

Write query to display the second most expensive product under each category

```
select *,
first_value(product_name) over w as most_exp_product,
last_value(product_name) over w as least_exp_product,
nth_value(product_name, 2) over w as second_most_exp_product
from product
window w as (partition by product_category order by price desc
    range between unbounded preceding and unbounded following)
```

Ab new column ban jaegi nth_value = second most exp_product k liye output 3rd wali product category wise karega

NTILE

Write a query to segregate all the expensive phones, mid range phones and the cheaper phones

```
select *,
ntile(3) over (order by price desc) as buckets
from product
where product_category = 'Phone';
```

output	Product_category	brand	product_name	price	buckets
1.	Phone	Samsung	Galaxy Z Fold3	1800	1
2.	Phone	Apple	iphone 12 Pro Max	1300	1
3.	Phone	Samsung	Galaxy Note 20	1200	1
4.	Phone	Apple	iphone 12 Pro	1100	1
5.	Phone	Samsung	Galaxy Z Flip3	1000	2
6.	Phone	Apple	iphone 12	1000	2
7.	Phone	Samsung	Galaxy S21	1000	2
8.	[illegible]	Oneplus	Oneplus 9	800	3
9.	Phone	Google	Pixel 5	600	3

Shot on AWESOME A05s

6. Galaxy Z Fold3 22.22%
7. iphone 12 Pro Max 25.93%

--- PERCENT_RANK (relative rank of the current row / Percentage Ranking)

/* Value --> 1 ⇐ PERCENT_RANK > 0 */
/* Formula = current Row No - 1 / Total no of rows - 1

--- Query to identify how much percentage more expensive is "Galaxy Z Fold 3" when compared to all product

```
select *,
percent_rank() over (order by price) as percentage_rank
from product;
```

output

	product_category	brand	product_name	price	Percentage_rank
1	Earphone	Samsung		170	0.03[illegible]
2	Smartwatch	Oneplus		220	0.9[illegible]
3	Earphone	Samsung		250	0.115
4	Headphone	Microsoft		280	0.115[illegible]
5	Earphone	Sony		350	0.13[illegible]
6	Earphone	Apple		400	0.23
7	Phone	Oneplus		400	0.2[illegible]
8	Headphone	Sony		550	0.2[illegible]
9	Smartwatch	Apple		600	0.34[illegible]
10	Headphone	Apple		~~250~~	0.3[illegible]

Shot on AWESOME A05s

emp_id	emp_name	dept_name	salary	prev_emp_salary
101	Mohan	Admin	4000	[null]
108	Maryam	Admin	4000	4000
113	Gautham	Admin	2000	4000
120	Monica	Admin	5000	2000
104	Darvin	Finance	6500	[null]
106	Rajesh	Finance	5000	6500
116	Satya	Finance	6000	5000
118	Tejaswi	Finance	5500	6500

VIEW

student Table

id	Name	Age	city
1	Ramkumar	19	1
2	salmanKhan	18	2
3	Meerakhan	19	1
4	Sarita kumari	21	3

city Table

cid	cityName
1	Agra
2	Bhopal
3	Delhi
4	Noida

INNER JOIN →

```
SELECT columns
FROM student
INNER JOIN city
ON student.city = city.cid;
```

id	Name	Age	city
1	Ramkumar	19	Agra
2	Salmankhan	18	Bhopal
3	Meerakhan	19	Agra
4	Saritakumari	21	Delhi

View Syntax

```
CREATE VIEW view_name
AS
SELECT columns
FROM student
INNER JOIN city
ON student.city = city.cid;
```

```
SELECT id, name, course_name FROM
student s
INNERJOIN courses c
ON s.courses = c.course_id;
```

output

id	name	course_name
1	Ramkumar	B.tech

Shot on AWESOME A05s