table / relation

each row should be unique

student

f_name	l_name	roll_no	c_id	email
vikash	singh	29	2	a12@gm
aman	kumar	3	4	a34@gm
aman	kumar	3	1	b1@gm
vikash	singh	15	2	c2@gm
ajay	kumar	2	3	null

keys

super key :- A super key is any set of attributes that uniquely identifies a tuple/row.

total non-empty = $2^n - 1$

n → no of attributes.

- it may contain more attributes than necessary to uniquely identifies a tuple/row

set of attributes	super key	Candidate key
{ f_name }	✗	✗
{ f_name, l_name }	✗	✗
{ email }	✓	✓
{ f_name, l_name, email }	✓	✗
{ roll_no }	✗	✗
{ roll_no, c_no }	✓	✓
{ roll_no, f_name, l_name, c_no }	✓	✗

Candidate key :- A **candidate key** is **minimal** of **super key**

- A candidate key is **minimum** **set** **of attributes** that uniquely identifies a row/tuple **and** **cannot** **reduce** **further**

{ ~~f_name~~, l_name, email } → (3)

{ ~~l_name~~, email } → (2)

{ email } → (1)

Primary key :- A primary key is candidate key that is choosen by DB designer.

- each table can have only one primary key
- it must be unique for each tuple and it cannot contain null values.

student (primary key: s_id; foreign_key: c_id)

s_id	f_name	l_name	roll_no	c_id	email
1	vikash	singh	29	2	a12@gm
2	aman	kumal	3	4	a34@gm
3	aman	kumal	3	1	~~b1@gm~~ abc3@gmail.com
4	vikash	singh	15	2	c2@gm
5	ajay	kumal	2	3	c3@gmail.

Primary key

x table

- it consumes more space in your disk

x_id	x_name	s_id
1	abc	1
2	xy2	2

foreign key: A foreign key is an attribute in a table that references the primary key of another table.

class table

primary key

C_id	C_name	C_teacher
1	abc	aman
2	xyz	Raju

composite key: A composite key is a combination of two or more attributes that uniquely identifies the table.

break

Schemas :→ it refers to the process of designing the structure and organisatio of a data base.

* schemas it is visual representation of the structure of the database.

it includes tables, columns, key and relationships.

make schema of your college.

① identify entities :- start by identify entities or objects

student courses teachers departments

② define Attributes : for each entity, we have to determine the specific attributes.

for eg:

student :- st_id, roll_no, name, email. etc.

students	
s_id	int
name	varchar
email	varchar
roll_no	int
address	varchar

courses	
course_id	int
course name	varchar
duration	int

Teachers	
T_id	int
name	varchar
email	varchar
address	varchar
d_id	(int)
fk	

stud_course

department

d_id int

name varchar

Head of Dept varchar.

s_id

c_id

3. established relationship : it defines how entities are related to each other.

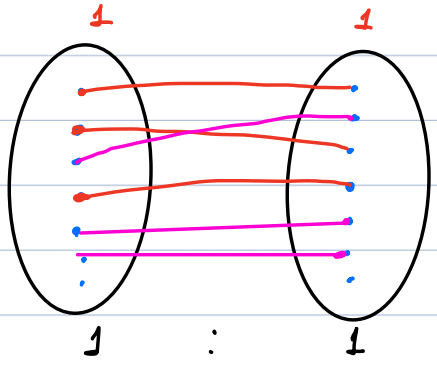
cardinality of relationships

(1) one - one relationship 1:1

Person | driving license

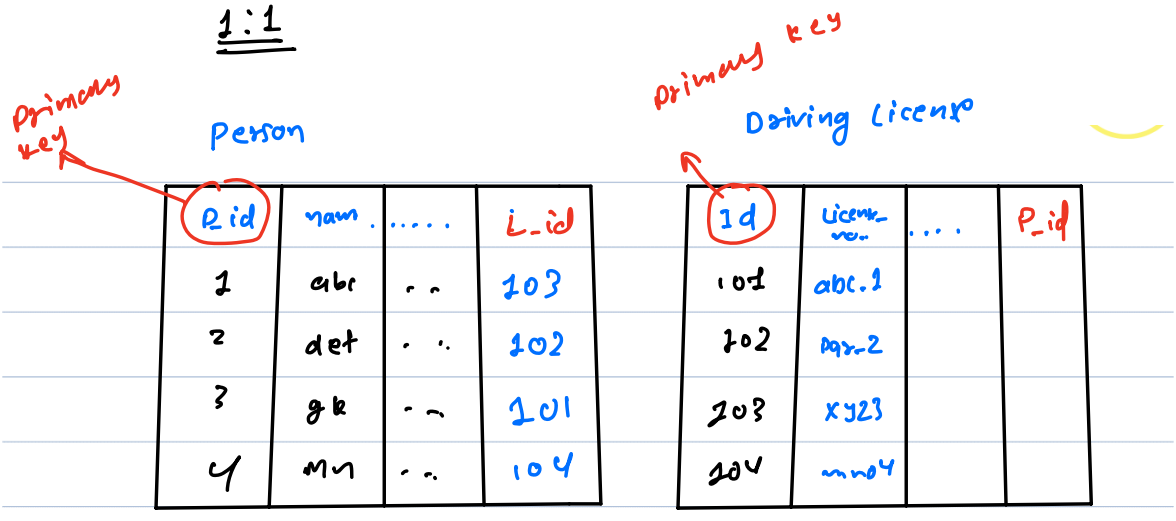
each person has only one driving license

each DL is assign to a single person.



1 : 1

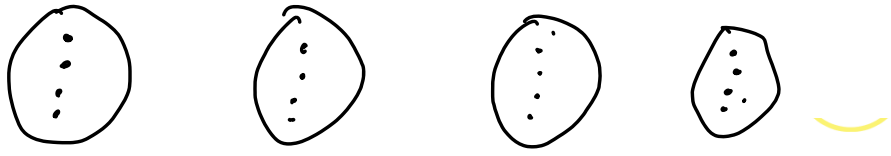
1:1



two ways

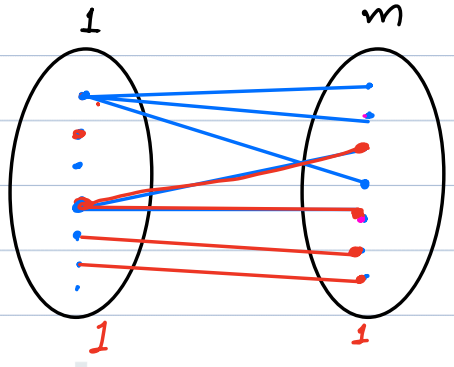
- we can create License id in person table
- we can create P_id in driving license table!

choose any of them.



② one to many relationship 1 : M

Department Teachers



1 : m = m
m : 1 = m
1 : 1 = 1
m : m = m

In each department we have many teachers.
each tecer is belongs to only one department

1 : M

Depart ment

D_id	nam.	.ad.	t_id
1	abc	..	[101,201,..]
2	def	..	
3	gk	..	
4	mn	..	

teachers foreign key

t_id	name	.add..	d_id
101	abc.1		3
102	pqr-2		4
103	xyz3		3
104	mno4		1
105	xyp		1

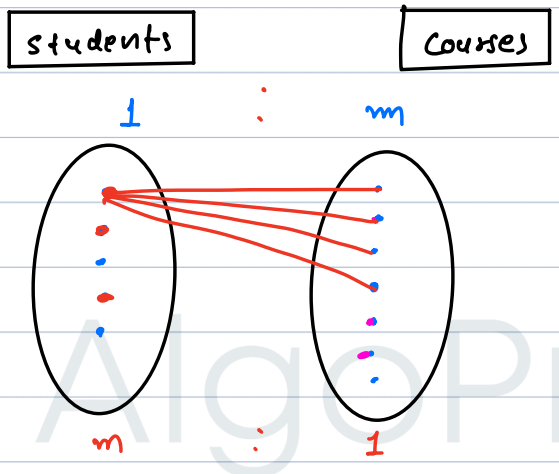
* How many teachers in a department 3

two ways

① we can put t_id in department table X
② we can put d_id in teacher table ✓

In 1:m relationship we select id from 'one' side and put this on "many" side

② many to many relationship



m : m

each student can take many courses.

In each course we have many students

s_id	name	email	cid
1	lokesh	gb@gm	[102, 107, 104]
2	aminul	kc@gm	
3	smith	det@gm	
4	abc	xyz@gm	
5	xyz	par@gm	

c_id	c_name	duration	s_id
101	eng12	30	
102	maths01	40	
107	eng12	35	[1, 3]
104	physics	40	
105	chem201	35	

How many students are enrolled in c_id = 103

Student

Courses

① choose student_id and put on courses table ✗

② choose course_id and put on studen_table ✗

we can't store multiple values in a cell.

Create a new table

junction table or join table

stud_course

s_id	c_id
1	102
1	103
1	105
2	102
2	104
2	105
3	101
3	102
3	105

we have unique records

In this scenario we dont make any PK

stud_course

id	s_id	c_id	stud_date	end_date	. . . . .
	1	102			
	1	103	July 23	Aug 30	In this
	1	105	sep 15	oct 30	scenario
	2	102			we create
	2	104			extra column
	2	105			
	3	101			
	3	102			
	3	105			

data types

- Integer
- String
- Decimal
- float / double
- boolean
- date & time
- enum
- BLOB

1. TINYINT :

storage : 1 bytes (8 bits)

range

signed (default)	unsigned
-128 to 127	0 to 255
127 − (-128) + 1 = 256	
	$2^n - 1$
-2^{n-1} to $2^{n-1} - 1$	$2^8 - 1$

n → no of bits

-2^7 to $2^7 - 1$

② SMALLINT

Storage : 2 bytes (16 bits)

Range	signed	unsigned
	-2^{15} to $2^{15}-1$	$2^{16}-1$
	-32768 to 32767	0 to 65K

③ MEDIUMINT :

Storage : 3 bytes (24 bits)

Range :	signed	unsigned
	-2^{23} to $2^{23}-1$	0 to $2^{24}-1$
	-8.3m to 8.3 million	(0 to 16.7 m)

⑤ INT

storage : 4 bytes (32 bits)

Range :	signed	unsigned
	-2^{31} to $2^{31}-1$	0 to $2^{32}-1$
	-2.1×10^9 to 2.1×10^9	

it is most common used for integer datatype

(5) BIGINT

Storage - 8 bytes (64 bits)

Range	signed	unsigned
	-2^{63} to $2^{63}-1$	0 to $2^{64}-1$

Salary (0 to 10 lakh) 10 crores

if is costly to change in your structure

- if the data types exceed the limit. it may required altering table. structure.
- if could be lead to a downtime and data inconsistency.

Decimal number

it is used to represent numbers that have both integer and fraction part

storing decimal values with fixed precision and scale.

Decimal(P,S)

Precision and scale ↳ max value is 64

35.938 → total length = 5

Precision: total number of digits including both integer and fractional part.

39827.563 → Precision = 8

Decimal(8,3) Scale = 3

Decimal(P, S) → max value is 64

(6,2)

324.987 .it is not possible

0324.99

Decimal(7,2)

39423.25 3942325

In this datatype exact value will be stored.

float / double

float

4 bytes

if approximately store 6 decimal digit of precision

398.23456

-3.4×10^{38} to 3.4×10^{38}

double

8 bytes

if approximately store 15 decimal digit of precision

234.392467385

-1.7×10^{308} to 1.7×10^{308}

String

`char(x)` :- [0, 255] 10

↳ fixed length of string

pincode, mobile no, aadhar card

`char(5)`

↳ "abcde", "parst"

↳ "ab---", "abd--"

if you insert a string which length is shorter x. it pads the remaining space with trailing spaces.

`char(7)`

↳ "abcdefg" ✓

↳ "abc----" ✓

↳ "parstabcdef" → errors / truncate string

its depending on database behaviour.

break till 10:20

varchar (x) :-

- it stores variable length string
- range :- (0 to 65,535)

x → maximum number of character allowed.

varchar (5) — storage.

↳ "ab" ✓ — 1 + 2 = 3 bytes

"abcd" ✓ — 1 + 4 = 5 bytes

"abcdefghijk...zp" — 200 ≤ 255 — 1 + 200 = 201 bytes

"abc...z" — 50000 — 2 + 50000 = 50,002 bytes

it stores actual length of the string + one or two bytes for length of the string.

			storage
varchar(6)	"ab"	✓	2+1
	"abcdef"	✓	6+1
	"abcdefgh"	error (default)	
		truncate → abcdef	
		same → "abcdefgh"	

it depends on your configuration or setting of data base.

its depends on your database as well.

Text

tiny text:	(255 bytes)	small description, commonly
text	(64KB)	articles, blog post.
medium text	(16 MB)	considering store the data in file system instead of DBMS.
long text	(4 GB)	

Boolean or bool

it can only hold two distinct values true or false.

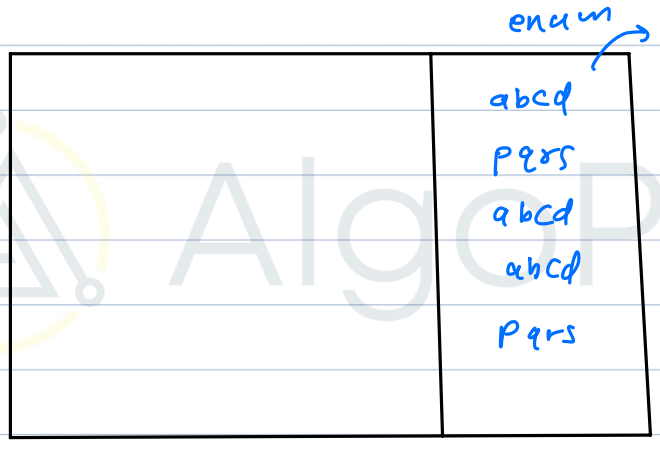
it can also represent TINYINT(1) → 0 (false), 1 (true)

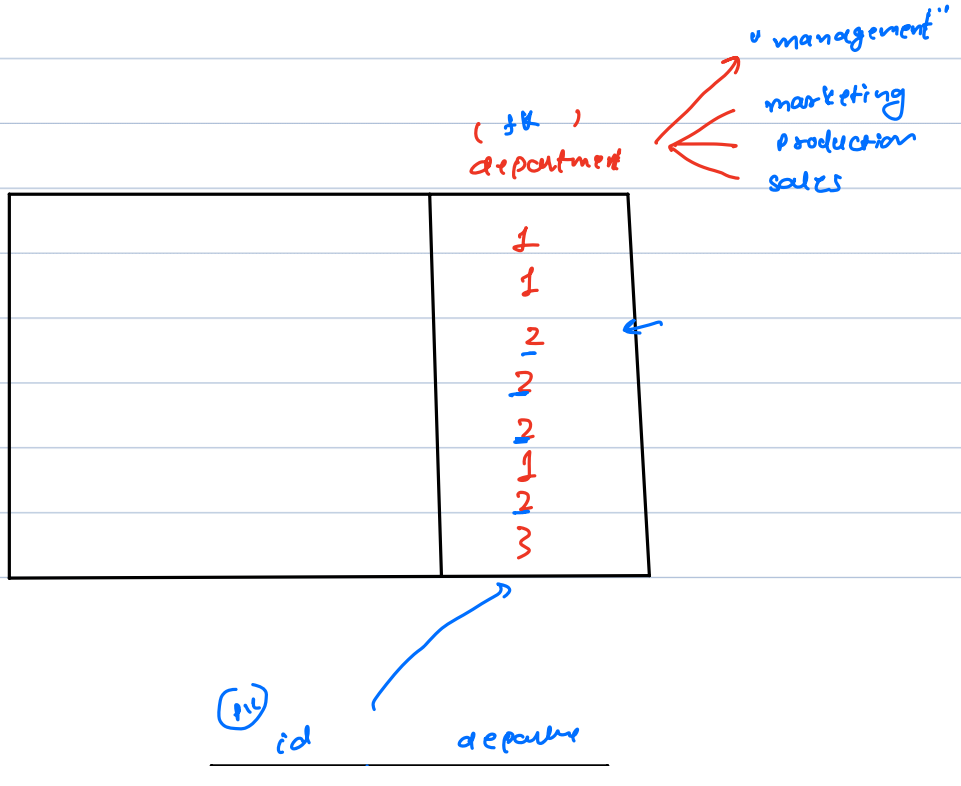
ex:- to know any product is present or not in your database.

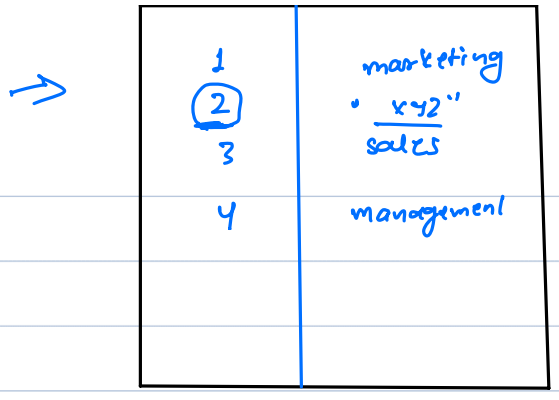
Enum :- A enum is defined with a list allowed values.

[value1, value2, value3]

["abcd", "pqrs"]







Date & time :.

Date → multiple formats of date

YYYY/MM/DD DD/MM/YYYY

1994/07/12 12/07/1994

In built function

time :→ HH:MM:SS

02:44:55

timestamp: second passed since 01/01/1910

it is valid till 2038

BLOB (Binary large objects):

TINY BLOB	(256 bytes)
BLOB	(64KB)
Medium BLOB	(16 MB)
LONG BLOB	(4 GB)

normalization

it is the process of organising the data in a database efficiently by eliminating redundancy and inconsistent dependency.

s_id	s_name	age	c_id	c_name	c_te
1	Amit	22	101	phy101	2000
1	Amit	22	103	Eng103	2500
2	Samique	24	101	phy101	2000
2.	samique	24	102	chem102	1500

there are three problems we faced

- insertion anomalies
- update anomalies
- delete anomalies

1) **Insertion anomalies**

Insertion anomalies occurs when we facing difficulties to insert new data. in a table.

2) **update anomalies**

update anomalies occur when data is redundancy.

- we need to update value multiple places.

3) **delete anomalies**

A deletion anomalies occurs when we remove any row from table, it lead to unintentional loss of data.

types of normalization

1NF

2NF

3NF

BCNF

1) First normal form (1NF)

- each cell contain only atomic values single value.
- there is no duplicate row in your table.

st. id	st-name	courses
1	Amit	Eng101, Phy102, chem103
2	LOKESH	ENG101, Phy102
3	Samique	Eng101

↓

st. id	st-name	courses
1	Amit	Eng101
1	Amit	phy102
1	Amit	chem103
2	LOKESH	ENG101
2	LOKESH	phy102
3	Samique	Eng101

2) Second normal formed. (2 NF)

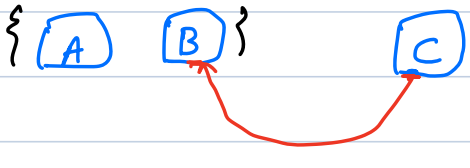
- it should be in first normal form
- all non-prime attributes are fully dependent on candidate keys
- it reduce partially dependency.

s_id	s_name	age	c_id	c_name	c_te
1	Amit	22	101	Phy101	2000
1	Amit	22	103	Eng103	2500
2	Samique	24	101	Phy101	2000
2.	samique	24	102	Chem102	1500

{s_id, c_id} ↓

s_id → s_name

it is partially dependent

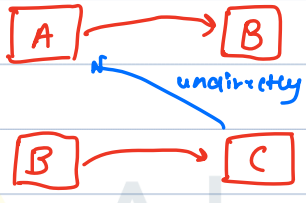


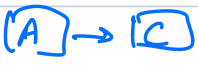
s_id	s_name	age
1	Amit	22
1	Amit	22
2	samique	24
2.	samique	24

c_id	c_name	c_te
101	Phy101	2000
103	Eng103	2500
101	Phy101	2000
102	Chem102	1500

3) third normal form (3NF)

- it should be in second normal form.
- eliminates the transitive dependency
- all non prime attributes is directly depends on prime attributes





transitivie dependency

A B C

c_id	c.name	c_te
101	Phy 101	2000
103	Eng 103	2500
104	Math 104	2000
102	Chem 102	1500

A → B B → C

A → C

4) BCNF (Boyce codd normal form)

it should be in third normal form.

$X \rightarrow Y$

it should be super key.

sid	c_id	c_name
1	101	Eng101
1	102	phy102
1	103	chem103
2	101	ENG101
2	102	phy102
3	101	Eng101

it violates the rule

c_id → c_name

X → Y

{c_id, s_id} → super key

{s_id} →

sid	c_id
1	101
1	102
1	103
2	101
2	102
3	101

c_id	c.name
101	Eng101
102	phy102
103	chem103

break till 10:10

CRUD

C → create

R → Read

U → update

D → delete

```
create table table_name(
    att_name att_datatypr.

)
```

```
Create table students(
    s_id int primary key auto_increment,
    f_name varchar(25) not null,
    l_name varchar(25) not null,
    age tinyint, not null,
    gender varchar(25),
    contact_no varchar(15),
    email_id varchar(30),
    graduation_year date
)
```

meth-1

```
insert   into   {table_name}
values (1. . . . . . . . . );
```

```
insert   into   students
values (
```

- we have give all the information.
  and also in same order.

meth-2

add data in specific columns in table.

```
insert   into   {table_name}
( col3, col1, . . . . )
values ( data3, data1 . . . . );
```

meth-3

```
insert   into   {table_name}
( col3, col1, . . . . )
values ( data3, data1 . . . . ),
       ( data3, data1 . . . . ),
       ( data3, data1 . . . . );
```

Read

Select { column_name }
from { table_name }
where { conditions }

```
select 2+5;
select 'aman', '2';
```

if we read the whole table

```
select * from students;
```

→ all columns

if we want to read data for specific columns

```
select f_name, c_name, age
from students
```

Operator

=	equal
<>	not equal
>	greater than
<	less than
>=	greater and equal to
<=	less and equal to

Good evening everyone 

class start at 8:10

Operators

=		equal to
!=	<>	not equal to
>		greater than
<		less than
>=		greater and equal to
<=		less and equal to

Students table

s_id	s_name	marks	total_marks	result
1	Lokesh	82	100	
2	Manasa	94	100	
3	Ajay	null	100	
4	Aman	74	100	
5	Vikash	58	100	
6	Rohit	null	100	
7	Manu	50	100	

- select all records where marks of the student is greater and equal to 70.

```
select *
from students
where marks ≥ 70
```

- select all records where marks of the student is greater and equal to 70 and less and equal to 80.

```
select *
from students
where marks ≥70 and marks ≤80;
```

```
select *
from students
where marks between 70 and 80.
```

inclusive

LOGICAL OPERATORS

AND , OR , NOT

Select all records where marks can be (75, 80, 85, 90, 95).

```
Select *
from students
where marks = 75 OR mark = 80 OR
marks = 85 OR mark = 90 OR marks = 95;
```

```
Select *
from students
where marks IN (75, 80, 85, 90, 95);
```


Select all records where marks is not null.

```
select *
from students
where marks <> null. ✗
```

is null , is not null

```
select *
from students
where marks is not null;
```

UPDATE

this keyword is used to update value or record in your database.

```
Update  { table_name }
SET     { column = value1 }
where   { conditions } ;
```

* if you have to update a single column for all records.

```
Update  students
SET     total_marks = 100;
```

* modify the value of marks in student table to 78 where s_id is 3.

```
Update  students
SET     marks = 78, s_name = 'vijay'
where   s_id = 3;
```

* modify each records if marks is greater and equal to 60 then set pass otherwise set fail. in result column:

```
Update  students
```

```
SET  result = if(marks ≥60, 'Pass', 'fail');
                cond^n
```

Alter

this keywood is used for modify the structure.

```
alter  table  {table_name}
```

- add an attribute total. marks in student table.

```
alter  table  students
add  total_marks  int;
```

- delete any column we use drop.

to rename any column.

```
alter  table  students
Rename  column  marks  to  score;
```

to rename table;

```
alter  table  students
rename  to  std;
```

- result is sorted in ascending order on the basis of primary key.

Order By

this keyword is used to sort the records on the basis of some specific columns.

- by default it gives result in ascending order and if you want in descending order you can use desc.

```
select *
from students
order by marks, s_name;
```

break till 09:20

AS

it is used to show data with temporary change in attributes.

```
select f_name as st_name, contact_no as m_no
from students
```

Limit

if you want some limited records you can use limit

```
select *
from students
limit 5;
```

skip top 10 values and get 5 results.

```
select *
from students
limit 5
offset 10; → skip top 10 values.
```

Like

zero character as well

% (Percentage) :→ it denotes any number of any character

_ (underscore) :→ a single any character.

- select all records where f_name is rahul.

```
select *
from students
where f_name = 'rahul';
```

```
select *
from students
where f_name like 'rahul';
```

- select all records where f_name is starts with ra ?

```
select *
from students
where f_name like 'ra';
```

→ get the result when name is exactly ra ✗

```
select *
from students
where f_name like 'ra%';
```

- select all records where f_name is starts with ra and the length of the name is five.

```
select *
from students
where f_name like 'ra___';
```

```
15
16    -- get all records between 20 to 30
17
18 •  select *
19    from students
20    where age between 20 and 30;
21
22    -- get all records between 20 to 30 in inr ord on basis of age
23
24 •  select *
25    from students
26    where age between 20 and 30
27    order by age;
28
29    -- get all records between 20 to 30 in decreasing ord on basis of age
30
31 •  select *
32    from students
33    where age between 20 and 30
34    order by age desc;
35
```

```sql

-- add a column marks in this table

alter table students
add column marks int;

update students
set marks = 80
where st_id in (20 , 22);

select f_name , marks
from students
where st_id in (20 , 22);

-- get all the records where contact_no is present

select *
from students
where contact_no is not null;


-- for temporary change we can use as keyword as means alias
select f_name as st_name , contact_no as m_no
from students;

-- if u want to see only some records in your table
select *
from students
limit 5
offset 3;

-- select all records where f_name is start from ra
select *
from students
where f_name like 'ra%';
-- select all records where f_name is start from a and length of the f_name is five?
select *
from students
where f_name like 'a____';
```

Delete

it is used to delete one or many row in your table.

- delete all records from your table.

```
delete from {table_name}
```

- delete all the records whose age is 22.

```
delete from students
where age = 22;
```

JOINS

instructor

i_id	i_name	i_age	deptid
1	Dr Anil	34	102
2	Mr umesh	38	103
3	Dr vipul	28	101
4	mr x	39	null

department

d_id	d_name	hod
101	mecm	x
102	civil	y
103	compute	z
104	physic	w

- get the details of i_name, i_id and d_name.

A table { virtual table }

i_id	i_name	i_age	deptid	d_id	d_name	hod
1	Dr Anil	34	102	102	civil	y
2	Mr umesh	38	103	103	compute	z
3	Dr vipul	28	101	101	mecm	x

```
select instructo.i_id, instructo.i_name, departments.d_name
from A;
```

JOINS

it is used when we have to combine two or more tables on a related column between them.

- get the details of g_name, g_id and d_name.

```
select instructor.g_id, instructor.g_name, department.d_name
from Instructor.
join department
ON instructor.dept_id = department.d-id;
```

1) inner join (default join)

A inner join returns only the rows from both tables that have matching values in the specific columns

it combines rows from both tables where the join condition is met.

```
select instructor.g_id, instructor.g_name, department.d_name
from Instructor.
inner join department
ON instructor.dept_id = department.d-id;
```

2) Outer joins

if there is any row in table A (left) is not connected to any row in table B (Right) and viceversa.

i) left join

all the rows from the left table and the matched rows from the right table.

if there is any row in table A (left) is not connected to any row in table B (Right)

Q) get all the detail of instructor with department_name and Hod of department.
if there is any instructor which not connected to any department in that get null values in d.name and Hod of dep

* → all information present in table.

```
select instructors.*, d.department_name, d.department_id
from instructors i
left join departments d
on i.department_id = d.id;
```

instructor

i_id	i_name	i_age	deptid
1	Dr Anil	34	102
2	Mr umesh	38	103
3	Dr vipul	28	101
4	mr x	39	null

department

d_id	d_name	HOD
101	mecm	x
102	civil	y
103	compute	z
104	physic	w

i_id	i_name	i_age	deptid	d_id	d_name	HOD
1	Dr Anil	34	102	102	civil	y
2	Mr umesh	38	103	103	compute	z
3	Dr vipul	28	101	101	mecm	x
4	mr x	39	null	null	null	null

→ left join

ii) right join

it we want all the records of right table and all the information of rows which connected to left table.

Q) get instructor_id, instructor_name, department_id dept_name, dept_hod where both are connected and get all information from right table.

break till 10:13

instructor

i_id	i_name	i_age	deptid
1	Dr Anil	34	102
2	Mr umesh	38	103
3	Dr vipul	28	101
4	mr x	39	null

department

d_id	d_name	hod
101	mecm	x
102	civil	y
103	compute	z
104	physic	w

i_id	i_name	i_age	deptid	d_id	d_name	hod
3	Dr vipul	28	101	101	mecm	x
1	Dr Anil	34	102	102	civil	y
2	Mr umesh	38	103	103	compute	z
null	null	null	null	104	physic	w

→ right join

```
select instructors.*, d.department_name, d.department_id
from instructors i
```

```
right join departments d
on i.department_id = d.id;
```

(iii) full join

we get all the records when there is match between two tables. either left or right. if there is no match we get null values for that missing columns.

```
select instructors.*, d.department_name, d.department_id
from instructors i
full join departments d
on i.department_id = d.id;
```

instructor

i_id	i_name	i_age	deptid
1	Dr Anil	34	102
2	Mr umesh	38	103
3	Dr vipul	28	101
4	mr x	39	null

department

d_id	d_name	hod
101	mecm	x
102	civil	y
103	compute	z
104	physic	w

department

i_id	i_name	i_age	deptid	d_id	d_name	hod
1	Dr Anil	34	102	102	civil	y
2	Mr umesh	38	103	103	compute	z
3	Dr vipul	28	101	101	x	mecm
4	mr x	39	null	null	null	null
null	null	null	null	104	physic	w

```
select  instructors.*, d.department_name, d.department_id
from  instructors i
left join  departments d
on  i.department_id = d.id
union
select  instructors.*, d.department_name, d.department_id
from  instructors i
right join  departments d
on  i.department_id = d.id;
```

self join

it the table is joined itself. it is used to compare rows within the same table.

Employee table

E.id	E_name	joining_date	manager_id
101	Lokesh	26-02-23	102
102	sandeep	30-06-23	null
103	Parveen	22-09-23	102
104	chhaya	28-11-23	101
105	Rakesh	09-01-24	101

get the employee name with their corresponding manager.

```
select  P.emp_name , q.emp_name
from  Instructor. P
inner join  Instructor. q
ON  P. E_id =  q. maagr_id
```

using

```
select  i.* , d.*
from  instructor
 join  department
using  dept_id
```

we can only use this keyword when the column name is same. it is replacement of on keyword.

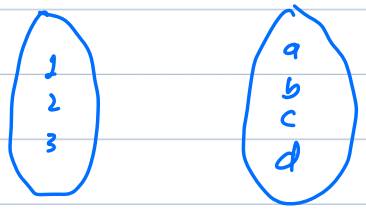
natural joins

In general we know there is only one column which name same and both table are connected through that coulumn in that case we can use natural join.

```
select *
from instructor
natural join department.
```

4 cross join

it returns cartesian product of the two table.
all possible combinations of rows from both tables.



n m = 12 combination

total combination n*m

Q) we have three tables student, course instructor.

student

s_id	s_name	c_id
1	x	102
2	y	101
3	z	102

course

c_id	c_name	i_id
101	mecm	202
102	civil	203
103	compute	201
104	physic	null

instructor

i_id	i_name	salary
201	A	100000
202	B	70000
203	C	80000

Q) get the s_name, c_name and instructor_name.
s c i

```
select  s.name , c.name, i.name
from    student s
join    course c
on      s.cid = c.c_id
join    instructor i
on      c.i_id = i.i_id
```

```sql

-- get the detail of inst_id , inst_name with dept _name

select i.instructor_id , i.instructor_name , d.department_name
from instructors i
join departments d
on i.department_id = d.department_id;

select i.instructor_id , i.instructor_name , d.department_name
from instructors i
inner join departments d
on i.department_id = d.department_id;

-- get the detail of instructor table with departmen_name , department hod
--  and if there is missing column then get null values

select i.* , d.department_name , d.hod_name
from instructors i
left join departments d
on i.department_id = d.department_id;

-- get the detail of department table with inst_name , inst qualification
--  and if there is missing column then get null values

select d.* , i.instructor_name , i.qualification
from instructors i
right join departments d
on i.department_id = d.department_id;

select i.instructor_id , d.department_id, i.instructor_name ,
 i.qualification , d.department_name , d.hod_name
from instructors i
left join departments d
on i.department_id = d.department_id
union
select i.instructor_id , d.department_id, i.instructor_name ,
 i.qualification , d.department_name , d.hod_name
from instructors i
left join departments d
on i.department_id = d.department_id;

```

* Aggregate functions

it is refers to combination or collection of values into a single entity.

(1) count

count (1, 7, 8, 10, 15) = 5

count (2, 5, null, 8, null, 7) = 4

generally aggregate functions ignore null values.

(2) sum( )

sum(2, 3, 7, 10) = 22

sum (2, null, 5, null, 3) = 10

(3) avg( )

avg(5, 10, 10, 5) $\frac{30}{4} = 7.5$

(4) max( )

max (2, 9, 7, 5) = 9

(5) min( )

min (9, 7, 3, 15) = 3

* count the number of students in students table.

```
select count(age) = 4
from students;
```

s_id	name	age	mobile_no
101	rakesh	19	1234
102	sandeep	25	2359
103	null	23	7824
110	null	null	null
115	srikant	22	7231

```
select count(name) = 3
from students;
```

```
select count(*) = 5
from students;
```

* calculate the average_salary of instructor table.

```
select avg(salary)
from instructors.
```

avg(salary)
number

```
select avg(salary) as avg_salary
from instructors.
```

avg_salary
number

instructor_table

i_id	i_name	salary	d_id
101	A	30K	202
102	B	25K	204
103	C	45K	202
104	D	40K	204
105	E	35K	204
106	F	30K	203

* calculate average salary of each department.

department_id	avg_sal
202	—
203	—
204	—

group by

```
select    d_id, avg(salary)
from      instructor
group by  d_id;
```

which used to group them

- In select we can use {the colum} or aggregate function

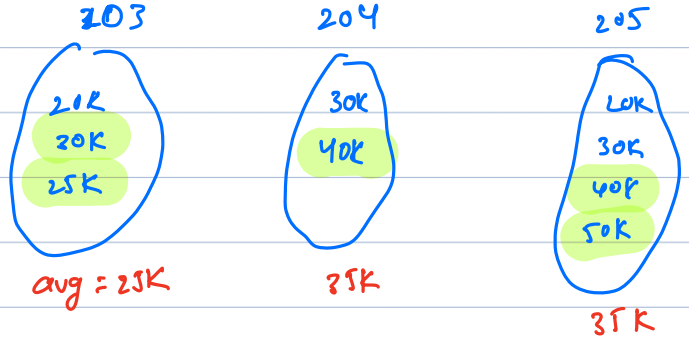
Q) show all the name of instructor whose salary is greater than average of the entire instructors.

```
select    inst_name
from      instructor
where     salary ≥ avg(salary)
```

where is applied only on single row. and to calculate average of salary we need all the rows.

```
select    inst_name
from      instructors
where     salary ≥ ( select avg(salary) from instruc);
```

Q1 display the dept_id and avg_sal of the department whose avg salary is greater and equal to the 40000.00




```
select  dept_id, avg(salary)
from    instructors
group   by  d_id
Having  avg( salary) ≥ 40000.00;
```

- where clause can't be applied after group.by.

break till 10:33

Built in functions

1) Strings

i) Length "Hello"

```
select length( string );
select length ("Hello");   5
```

ii) LOWER

LOWER(" ABccD") → abccd

iii) UPPER

UPPER(abccde) → ABCCDE

iv) SUBSTRING

substring ('Hello_world', 7) = world
(1 2 3 4 5 6 7 8 9 10 11)

In this language we used 1-based indexing

substring ("abcdefghi", 4, 5)
(1 2 3 4 5 6 7 8 9)
4 → starting index
5 → length

detgh

get 5 chacter from starting. in "abcdefghi"

substring("abcdefghi", 1, 5);

abcde

v) LEFT it gets the chasacters from starting

LEFT("abcdefghi", 5)

abcde

vi) Right it gets the characters from ending.

RIGHT("abcdefgh123456", 7)

123456

vii) trim it removes all leading and trailing spaces.

trim("....abcd.....") = "abcd"

leading trailing

(viii) Ltrim remove leading spaces

Ltrim("....abcd.....") = "abcd....."

(viii) Rtrim remove trailing spaces

Rtrim(" . . . . abcd . . . . . ") = " . . . . . abcd"

(ix) LOCATION to search this string

LOCATION(string , ori string , start_pos)

optional

LOCATION(" cde" , "abcdefg") (1 2 3 4 5 6 7)

= 3

LOCATION(" acde" , "abcdefg") = 0

if the string is not present in original string then in this case it gives 0.

2) NUMERIC

i) abs → it gives the positive value

abs(-100) = 100

ii) ceil → it represent the smallest integer greater than or equal to that number.

ceil (3.078) = 4

ceil (3.00) = 3

iii) Floor it represent the largest integer smaller than or equal to that number.

Floor (3.078) = 3

Floor (3.000) = 3

iv) Round

Round (number, numbers after decimal)

Round (359.23567, 2) 359.24

v) truncate

truncate(354.93265, 2) = 354.93

v) Rand

Rand() 0.37892

↓

[0 , 1)

v) POWER

POWER(2, 3) = 8

3) DATE

1) now() → current date and time
2) curdate() → current date
3) curtime() → current time
4) dayname("24-02-29") → thrusday

5) date_add(curdate() interval 6 month)
add this interval 2 year
7 day

6) date_sub (curdate() interval 6 month)

subtract this interval 2 year

7 day

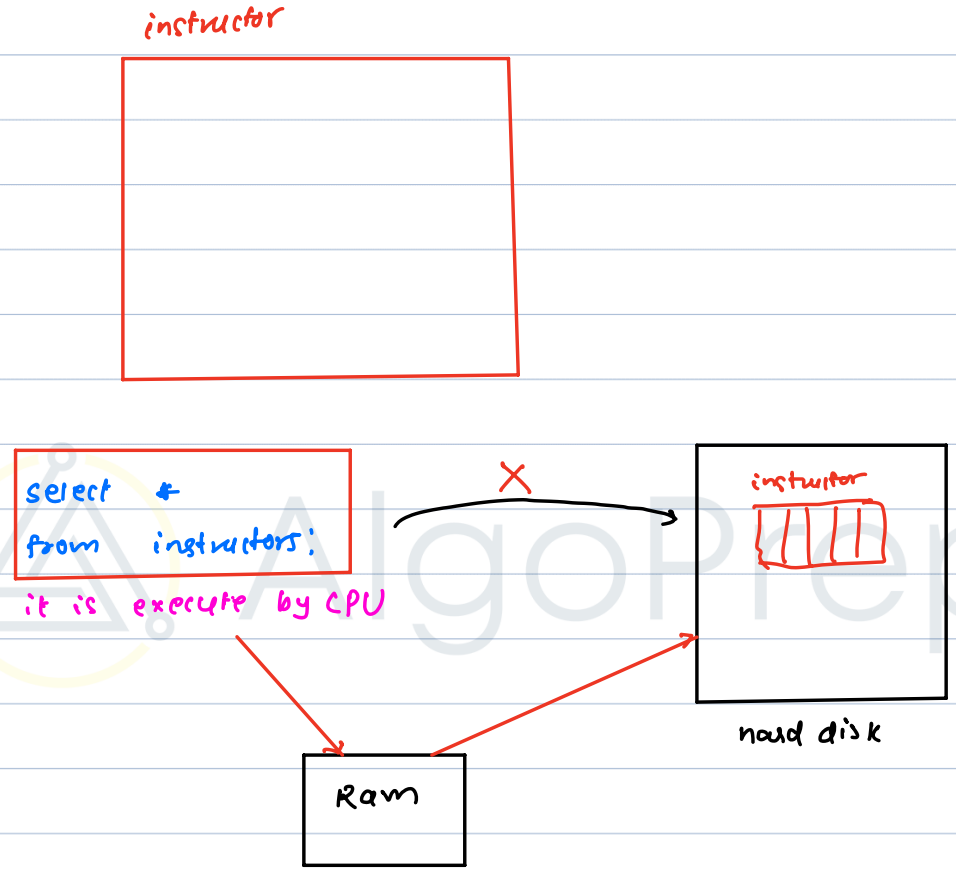
7) datediff ('24-02-29', '24-01-1')

8) date_format ("24-02-29" , "%d-%m-%y") :

yy mm DD

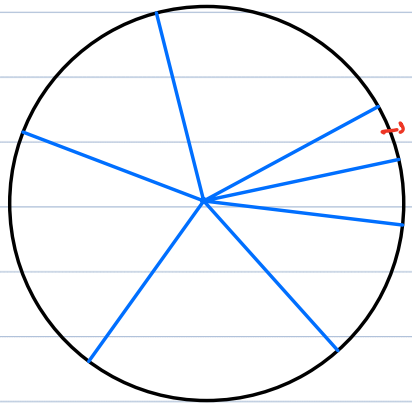
```sql
-- get the number of instructors in instructors table
select count(*) as num_of_Inst
from instructors;

select avg(salary) as avg_sal
from instructors;

-- get the count unique innst_name from instructor table
select count(distinct instructor_name)
from instructors;

-- get total salary of instructors
select sum(salary)
from instructors;

-- calculate total salary of each department
-- calculate avg salary of each department
select  avg(salary)
from instructors
group by department_id;

-- display the department_id and avg_salary of each instructor whose salary is greater than
--  and equal to 40000;
select  department_id, avg(salary)
from instructors
group by department_id
having avg(salary) >= 40000.00;

select ceil(3.067);
select floor(3.0067);

select substring("abcdefghi" , 2 , 5);
select dayname(curdate());
```

Indexing :



1) CPU can't directly interact with hard disk.

2) if we have to fetch any data we interact with CPU and then CPU fetch the data from hard disk



→ it is sectors

instructor → primary key

	i_id	i_name	i_email	d_id	
* a1	101	mr x	abc1@	202	→ we fetch the data from ram then check cond^n
* a2	102	mr y	abc7@	203	→
# a3	103	mr z	abc2@	201	→
# a4	104	mr A	abc5@	202	→
# a5	105	mr B	abc8@	203	→
# a6	106	mr C	abc6@	204	→

→ we dont search this row.

```
select  *
from    instructors
where   d_id  203
```

- if we have 10,00,000 rows in this table then how many times we have interact with disk ?.
- there is only 50 instructors in d_id 203.
  we have access 10,00,000 times.
  waitage access of disk :- 10,00,000 - 50.

- by default all the rows are sorted on the basis of primary key.

```
select *
from instructors
where i_id = 103;
```

- we search the information till we get the matching condition.

i_id	add
101	# a1
102	# a2
103	# a3
104	# a4
105	# a5
106	# a6

→ main memory

Structure → indexing

it we created this table in ram.

* How many times we access the disk.

(1)

```
select *
from instructors
where i_id = 103;
```

* it takes some extra spaces.

* Indexing

* its reduces the number of access of disk.
* its improve the perfoormance of read queries.
* if we have write queries (C/U/D), then in that case we have to update our index structure as well.

* when we have frequently queries of read type then we used indexing.

* it is sorted on specific column.
* it is used for getting fast retrieval.

map / treemap

B / B+ tree → balancing trees

- this structure is stored in your DB

- if we have student table and we have frequently get the information about marks of the student.

How to create index.

- first identify the coulumn.

create index { idx_:--.. } on
↓ name
table_name (column_name) ;

- create index idx_student_mark on students (marks);

- create index idx_instructor_d_id on instructors (d_id);

break till 10:13

- lets suppose the data-type of column is string.

```
select *                                  1,00,000
from instructors
where inst_name = "mr.y";
```

10,000 records in this table.

(a... 1
256

when we create this table it takes more spaces and also time consuming.

- if we just store single character.

m → [15, 10, 25, ... so on]
↓
5000

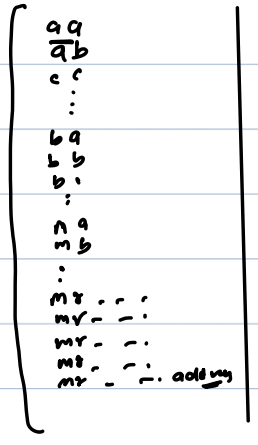
(a, b, c .... z) → only 26 combination

- we optimise the space.
- we have to access the disk 5000.

(2) we stored two character in index table.

create index idx_student_name on

student(st_name (2));



mr → [mrx . mry . . . .]
↓
500

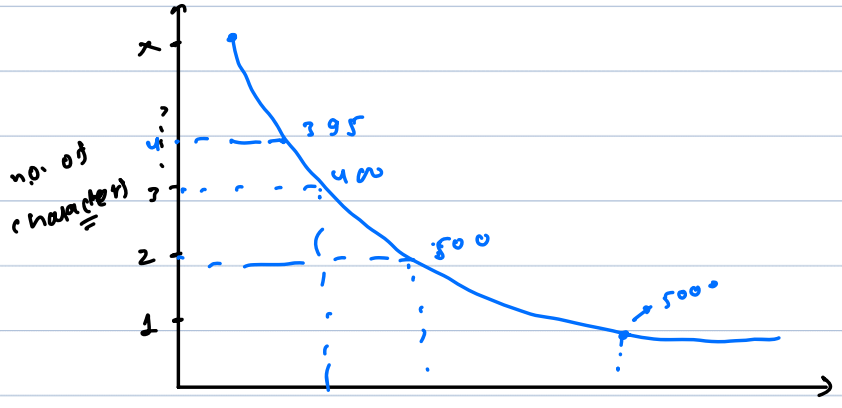
- if is less optmise in space as compare to previous
- we have to access the disk 500.

- make a three character index

mrx [a . . .]
↓
100 dat

- if is less optmise in space as compare to previous
- we have to access the disk 100.

- if we make the index table on the exact values.
  - gt is not much efficient in space optimisation.
  - we have to access the disk only one time



- if we want to make performance very fast then in that case we consider higher character index table.
- if we have maintain both time as well as space then we have to find the length to make the index table.

- we create a table on book.
  it have title, description, summary

if we have to find those books which contains treasure, adventure in decription or summary.

treasure, adventure

id	title	description	summary
201	the_ book1	"xyz..."	" "
101	the. book2	"abc pqr"	" "
104	the book3	"the treasure"	" "

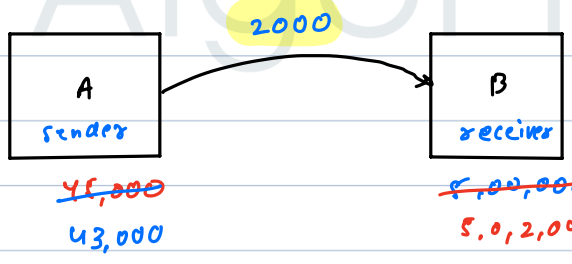
```
create  fulltext index  idx_bookdes_sum
on  books ( description, summary);

select  *
from  books
wher  match ( desc , summ )
AGAINST  ( "  treasure  adventure');
```

AND	'+'	("+treasure +adventure")
OR	no operator	("treasure adventure")
NOT	'-'	(+tresure +adventure -water")

```sql
show indexes in students;
show indexes in products;
explain select *
from students
where st_id = 50;

explain select *
from students
where marks = 50;

create index idx_students_marks
on students(marks);

drop index idx_students_marks
on students;

create fulltext index idx_product_name_desc
on products(name , description);

select * from products
where match (name , description)
against ('photography -lens' in boolean mode);

select * from products
where match (name , description)
against ('+photography +camera' in boolean mode);

```

transactions

A transaction refers to a logical unit of work that consists of one or more database operations. (update, insert, delete)

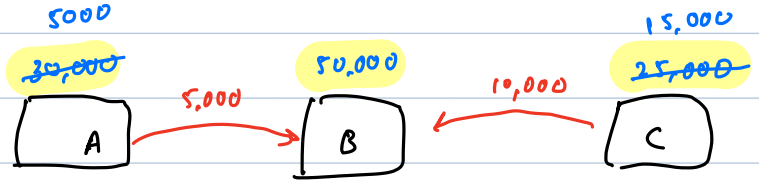


(sender account A Receives account B money)

```
(1) int x = getcurrentBal( A)
(2)   checkBal ( x, money);
(3)    updateBal ( A, x-money);   <- failure
(4) int  y = getcurrentBal (B);
(5)       updateBal ( B, y + money);
```

due to system failure partially transactions completed.

it causes data inconsistency.



transaction 1

```
int x = getCurrentBal(A)
checkBal(x, money);
updateBal(A, x-money);
int y = getCurrentBal(B);
```

50,000

```
updateBal(B, y+money);
```

transaction 2

```
int x = getCurrentBal(C)
checkBal(x, money);
updateBal(C, x-money);
int y = getCurrentBal(B);
```

50,000

```
updateBal(B, y+money);
```


if t1 execute first

50,000 ~~55,000~~ 60,000, we loose 5000

if t2 execute first

~~50,000~~ ~~60,000~~ 55,000

we loose 10,000

both are wrong

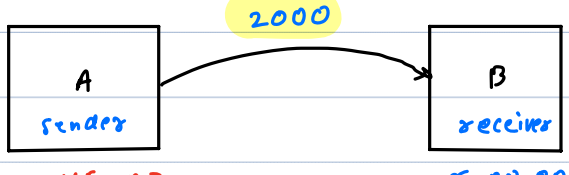
TO solve these problem we follow ACID properties

A → Atomicity
C → consistency
I → isolation
D → durability

1. Atomicity :

it ensures that a transaction is treated as single unit of work.
either all the operations within the transaction are successfully completed or none of them are performed at all.
if there is any failure it roll back to the starting of the transaction.

eg:



~~45,000~~
43,000

~~5,00,000~~
5,02,000

(sender account A receives account B money)

① int x = getcurrentBal(A)
② checkBal (x, money);
③ updateBal (A, x-money); ← failure
④ int y = getcurrentBal(B);

(5) updateBal (B, y + money);

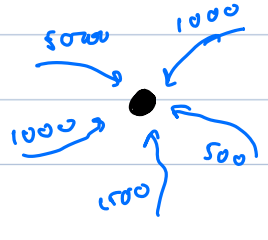
2. **consistency** :-

- its guranteed that data will be in consistent state.
- it is possible that data may be inconsistent during the transactions.
- we check database is remain in a valid state before and after the execution of transaction.

if money will be deducted from account A and not receive to the account B.
data will be inconsistent.

3 **isolation**

isolation ensure that transaction operate independently of each other. and their effects are isolated from each other.



4 **durability**

it is ensure that changes are permanently stored in a database when the transaction is complete.

if any failure happen its revert back to the last commited point.

- we just do temperasory change in the database.

How to make permanent if the transaction is successful. or How revert back if the any failure happen in transaction.

1. **commit**

it marks that the transaction is succesfully completed.

if these is any failure and we dont commited then in that case it will be automaticcely reverted back all the operation in that transaction.

2. **Roll back** :

this keyword is used to cancel all the operations changes by the transaction.

break till 10:28

Transaction isolation level

- Read uncommitted
- Read committed
- Repeatable read
- serilizable

1. **Read uncommitted**:

transaction 1	transaction 2
start	start
fetch amount from abc	fetch amount from abc
x = 1,00,000	y = 1,00,000
update (x , 1,00,000+5,000)	fetch amount from abc
→ any failure happen	105,000
	update (abc, y+10,000)
commit;	

abc
1,00,000 → commited value.

105,000 → temporary change / uncommited vale.
110,000

- this is the lowest isolation level where transactions can see changes made by other transactions befre they are commited.
- it is also known as dirty data. or dirty read.
- it offers high concurrency. it sacrifices data consistency.

Read committed

transaction 1	transaction 2
start	start
fetch amount from abc	fetch amount from abc
x = 1,00,000	y = 1,00,000
update (x , 1,00,000 + 5,000)	fetch amount from abc
	1,00,000
commit;	

abc
1,00,000 → commited value.
1,05,000 → temporary change / uncommited vale.

- In this isolation level, transactions can only see changes commited by other transaction.
- Read committed provides a balance between concurrency and data consistency.

3 Repeatable read

transaction 1	transaction 2
start	start
fetch amount from abc	fetch amount from abc
x = 1,00,000	y = 1,00,000
update (x , 1,00,000 + 5,000)	fetch amount from abc
	RUC → 1,05,000
commit	RC → 1,00,000
	RR → 1,00,000.

abc

1,00,000 → commited value.

1,05,000 → temporary change / uncommited vale.

- amount will be same throughout the transaction.
- before and after committed in other transactions its not effect the value.

serializable:

transaction 1	transaction 2
start	start
fetch amount from abc	fetch amount from abc
x = 1,00,000	y = 1,00,000
update (x , 1,00,000 + 5,000)	fetch amount from abc
	RUC → 1,05,000
commit	RC → 1,00,000
	RR → 1,00,000.

abc

1,00,000 → commited value.

1,05,000 → temporary change / uncommited vale.

- serializable is the highest isolation level.
- it is the strongest level of consistency.
- transactions are executed serially.
- it basically lock the rows uses in the current transaction and other transaction can't be interfere in this.

start t1	start t2
update p_id = 5	update p_id 7
→ update p_id : 7	update p_id = 5

- **Deadlock**

  When the two transaction is dependent on each one.

- my SQL is terminated one of the transaction.

```sql

-- without transaction
update inventory
set quantity = 180
where p_id = 4;

-- with transaction
start transaction;
update inventory
set quantity = 150
where p_id = 4;

update inventory
set quantity = 70
where p_id = 2;
-- automatically rollback
commit;

start transaction;

update inventory
set quantity = 150
where p_id = 1;

rollback;

show variables like 'transaction_isolation';

set session transaction isolation level repeatable read;
set session transaction isolation level serializable;
start transaction;

update inventory
set quantity = 70
where p_id = 1;
```

```sql
select * from inventory
where p_id = 1;

-- automatically rollback
commit;

select * from inventory;
```

subquery

it is also known as nested query and inner query.

student table

s_id	name	age	marks	
101	A	22	87	→ ✓
102	B	24	70	→ ✗
103	C	23	62	→ ✗
104	D	24	89	→ ✓
105	E	22	55	→ ✗
106	F	21	75	→ ✓

Q) find all the students name who scored greater than 70 marks ?.
✗

```
select  name
from    students
where   marks > 70;
```

its run for each row.

↳ list of students names.

Q) find all the students name who scored greater than the marks of st_id 102.

```
select  name
from    students
where   marks > x;
   x = marks of st_id = 102
      select  marks
      from    students
      where   st_id = 102;
```

with subquery O(n²)

```
select  name
from    students
where   marks > ( select  marks
                  from    students
                  where   st_id = 102);
```

with join

```
select  name
from    students  s1
join    students  s2
ON      s1.marks > s2.marks
and     s2.s_id = 102;
```

- subquery is used to get data or perform any operation based on result of other query.
- its break down complex logic in smaller or manageable chunks.
- this is easier to understand and more readable.

- find the all students who have marks greater than the average marks of students?

```
x = select avg(marks)
    from students;

select *
from students
where marks > x;
```

```
select *
from students
where marks > (select avg(marks)
               from students);
```

instructor_table.

i_id	name	salary	dept_id	
1	A	90K	105	✓
2	B	60K	110	✗
3	C	50K	108	✗
4	D	80K	110	✗
5	E	70K	110	✗
6	F	87K	104	✓

(60K, 70K, 80K) → max = 80K

Q) display instructor name who have salary greater than salary of all the instructor of dept_id = 110.

find max of a group

```
select name
from instructor a
where salary > (select max(salary)
                from instructors
                group by dept_id;
                having dept_id = 110);
```

```
select name
from instructors
where salary > ALL ( select salary
                     from instructors
                     where dept_id = 110)
```

list of salary
(60K, 70K, 80k):

8) display instructor name who have salary greater than salary of any instructor of dept_id = 110.

```
select name
from instructor
where salary > (select min (salary)
                from instructor
                group by dept_id;
                having dept_id = 110);
```

```
select name
from instructors
where salary > Any ( select salary
                     from instructors
                     where dept_id = 110)
```

list of salary
(60K, 70K, 80k):

correlated query

display the name of instructor who earn more than the average salary of their respective department.

```
select name
from instructor a
where salary > ( select avg(salary)
                 from instructor
                 group by dept_id
                 having dept_id = a.dept_id );
```

```
select name
from instructor a
where salary > ( select avg(salary)
                 from instructors
                 where dept_id = a.dept_id );
```

break till 10:23

find the number of instructor where dept_id = 110?.

```
select   count(*)
from     instructors
where    dept_id = 110;
```

8) find the number of instructor for each department ?.

```
select   dept_id, count(*)
from     instructors
group by dept_id
```
→ unique department

all department including ← null values

```
select   dept_id , ( select count(a) from instructors
                     where dept_id = a.dept_id )
from     instructors a ;
```

```sql
-- get student f_name , l_name marks where marks of that student is grater than st_id = 23;
select f_name , l_name , marks
from students
where marks > (select marks from students where st_id = 23);

-- display inst_name salary > salary of all the inst of dept_id = 104

select instructor_name
from instructors
where salary > all(select salary from instructors where department_id = 104);

-- display inst_name salary > salary of any the inst of dept_id = 110

select instructor_name
from instructors
where salary > any(select salary from instructors where department_id = 110);

-- display the name of instructor who earn more than the avg sal of their respective dept

select instructor_name , salary
from instructors a
where salary > (select avg(salary) from instructors
                group by department_id
                having department_id = a.department_id);


select instructor_name , salary
from instructors a
where salary > (select avg(salary) from instructors
                 where department_id = a.department_id);

select department_id , count(*)
from instructors
group by department_id;

select department_id , (select count(*) from instructors
                        where department_id = a.department_id)
from instructors a;
```

- there various languages or components in DBMS.

1) DDL (data definition language):

its includes

create → its used to create any object

Alter → its used to modifying objects.

drop → its used to remove objects.

2) DML (data manipulation language)

it is used for manipulating the data within tablel.

insert

update

delete

3) DQL (data query language)

select

4) TCL (transaction control language)

it is used for managing tranlations in data base.

commit

rollback

5) DCL (data control languages)

it is used to control access of data in database.

grant : (to give permission to user)

revoke : (for revoking permission)

grant privileges on table_name to 'user_name'

privileges → which type of operation we want to give like select, update, delete, drop, insert etc.

views

it is refers to virtual table.

they do not store physically . but its provide a dynamic representation of the underlying table.

(st_id, st_name, inst_id, inst_name, dept_name)

- views can simplify complex queries by encapsulating into a single
- they can be used for restrict sensitive data to others.
- we just show the relevant data that are required.

break till 10:25

windows function

find avg salary from instructor

```
select , instructor.name, avg(salary)
from instructors.
```

instructor_table

i_id	i.name	i_salary	dept_id

Salary
-
-
-
-
-

```sql
-- privileges , select , update , delete , alter , drop etc
grant select on students to 'Mayur';

grant select , update on instructors to 'Mayur';

revoke update on instructors from 'Mayur';

-- display st_id , st_name , instructor_id , instructor_name ,department_id
-- department_name from table students , instructors and departments;


select st_id , st_name , i.instructor_id , instructor_name ,
d.department_id , department_name
from students s
join instructors i
on s.instructor_id = i.instructor_id
join departments d
on i.department_id = d.department_id;


create or replace view inform_tab
as (
        select st_id , st_name as nme , i.instructor_id , instructor_name ,
        d.department_id , department_name
        from students s
        join instructors i
        on s.instructor_id = i.instructor_id
        join departments d
        on i.department_id = d.department_id
);

select * from inform_tab;

```

```sql

update inform_tab
set nme = 'bam'
where st_id = 2;

-- get the inst_id , inst_name and avg for each isnt

-- it is not possible
select instructor_name, avg(salary)
from instructors;

-- subquery
select instructor_name, (select avg(salary) from instructors)
from instructors;

-- window function
select instructor_name ,
    avg(salary) over()
from instructors;

-- fetch all the isntructor_name along with avg salary of all
-- instructot in their respective department

-- subquery
select instructor_name,
(select avg(salary) from instructors where department_id = a.department_id)
from instructors a;

-- window function
select instructor_name ,
      avg(salary) over(partition by department_id) as avg_sal
from instructors;

 -- order by
 select st_id , f_name , marks
 from students
 order by marks desc;

 -- window rank()

 select st_id , f_name , marks,
 rank() over(order by marks desc)
 from students;

 -- sparse ranking
 -- 1 , 1 , 3, 4 , 4, 4, 4, 8

 -- dense ranking
 -- 1, 1, 2, 3, 3 ,3 ,3 , 4,,,

 select st_id , f_name , marks,
 dense_rank() over(order by marks desc)
 from students;

 -- row number
 select st_id , f_name , marks,
  row_number() over(order by marks desc)
 from students;

```