

# Crud - II

## Agenda.



CRUD operation

→ Create

→ Read

→ WHERE

→ ~~AND, NOT, OR~~

→ ~~IN, BETWEEN~~

→ ~~LIKE~~

→ ~~IS NULL~~

→ ~~ORDER BY~~

→ ~~LIMIT~~

All the  
foundational  
topics.

→ ~~Update~~

→ ~~Delete~~

→ ~~Delete vs Truncate vs Drop~~

What is CRUD

Students			
id	name	psp	batch
1	Ujjwal	100	21
2	Alok	31	20
3	Deepak	22	40

What all type  
of operations can you  
do on data of this  
table.

- Read data (R)
  - Update data (U)
  - Delete data (D)
  - Create data (C)
- CRUD operation

- CRUD
- Create
  - Read
  - Update
  - Delete

⇒ Practical example on Sakila Database.

Sakila Database Overview (Ecommerce app)  
(library management system)

→ Digital video rental store

Why Sakila

⇒ Official MySQL DB  
(created by MySQL team)

C: Create

→ put new data in a table

Statement → INSERT.

## create in film table

```
INSERT INTO {table-name} (  
    {column-names}  
) VALUES  
    (  
        _____  
    )  
    (  
        _____  
    )  
    (  
        _____  
    );
```

[CODE]

→ proper insert with multiple values

→ insert without required field ⇒ error

③ [NOT RECOMMENDED] → insert without values i.e. in order of table creation

① (cannot skip a value)

② (it fails after alter table)

→ default keyword for default values

R : Read

Statement  $\Rightarrow$  SELECT

Select  $\circledast$   $\longrightarrow$  every coln of given table  
from {table-name}

[CODE]

$\rightarrow$  select title, film-id  
from film.

$\rightarrow$  title  $\circledast$  title-name  
 $\downarrow$   
ALIAS

$\Rightarrow$  Let's see how these queries might actually be executed internally

{ select  $\ast$  from table-name }

table\_name = [ { }, { }, { } ... ]  
ans = [ ]

for row in table\_name :  
    ans.add(row)

print(ans)

```
select name, film-id  
from film
```

```
table-name = [ { }, { }, { } - . . . ]  
ans = [ ] → intermediary table  
for row in table-name:  
    ans.add(row)
```

```
for row in ans:  
    print(row[name], row[film-id])
```

[ BREAK ]

[ CODE ]

gives distinct  
values of the  
pairs

→ get distinct values<sup>(rating)</sup>  
for a column

(distinct)

→ get release year  
& rating  
(remove duplicates)

⇐ (distinct)

→ wrong combination  
to prove distinct  
always be before  
first column name

## DISTINCT

- ① distinct should be very first thing after select
- ② distinct is applied to pair of values

```
table_name = [ { }, { } - - - - ]
```

```
ans = [ ]
```

```
for each row in table_name:  
    ans.add(row)
```

```
filtered_ans = [ ]
```

```
for each row in ans:
```

```
    filtered_ans.add(row[rating], row[year])
```

```
print(set(filtered_ans))
```

↑  
no duplicates

```
// distinct
```

↓

gets applied  
after the entire  
ans has been  
created

Till now we have done:

```
select * from film;
```

```
select title from film;
```

```
select 1;
```

print ↗

select "Hello world";

[ let see this in action ]

Q Print title of every film and alongside that print (Hello)

Jab tak hai Jaan	hello
Baazigar	hello
K3G	hello

select title, 'Hello' from film;

★ now() function to get current timestamp.

We have length in the film table in db.  
⇒ in minutes

Q → Print name, length (in hrs)

⇒ operations on cols in select query

select title, length / 60  
from films;

[ we can do normal operations in select query ]

⇒ get round off value

select round ( length / 60 ) from film.

↓  
we can use  
functions too

⇒ have operations across cols.

Print how many times a person  
can watch a movie

→ movie is rented for  $x$  hours

→ movie has a length  $y$ .

no. of times he can watch =  $\frac{x}{y}$

select title, floor (rental - duration / ( length / 60 ))  
from film.

Q given a table film

create a table ⇒ film - copy

create with all values of film table as is.



```
insert into film-copy (  
    coln_names  
) VALUES  
(  
    )
```

```
insert into film-copy (  
    coln_names  
) select coln_names  
from film
```

Same  
data  
types

// put all values of film table in  
film-copy table

Allows to  
put values of  
one table to  
another table

eg. insert into film-copy  
(title)  
select ~~date~~ description  
from film

won't work  
as diff data  
type

works as both  
are strings

[CODE]

→ random\_names table

# WHERE CLAUSE

select \* from table;  $\Rightarrow$  was fetching data from all the rows and putting in intermediary table.

select title  
from films

Q Print titles of films that are rated PG-13

table-name = [ { }, { }, { } - . . . ]

ans = [ ]  $\xrightarrow{\text{intermediary table}}$

for row in table-name:

① ans.add(row)  $\xrightarrow{\text{if rating == PG-13}}$

for row in ans:

② print(row[name], row[film-id])

$\xrightarrow{\text{removing redundant data ASAP.}}$

$\Rightarrow$  Where clause

$\hookrightarrow$  allows to filter rows based on condition.

```

select title
from film
where rating == 'PG-13';

```

film

id	name	rating
→ // // // //	→ // // // //	→ // //
→		PG-13
→ / // // //	→ // // // //	→ // // //
→		PG-13

← removed only from intermediary table

```

table = [ ]
ans = [ ]

```

```

for every row in table: // from
    if row matches condn: // where
        ans.add(row)

```

```

for every row in ans: ] // select
    print(row)

```

[ CODE ]