SCALER

# CRUD - 1

TABLE OF CONTENTS



# Create

- Create database
- Create table
- Adding new entries

***< / > Syntax***

```
Insert into table_name ( col1, col2 )
                values ( values_1, values_2 );
```

- Column names are optional. Let's see this scenario as well :

| film_id | title | description | release_ year | language_ id | original_ language_id | rental_ duration | rental_ rate | length | replacement_ cost | rating | special_ features | last_ update |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | |

```
INSERT INTO film
VALUES (default, 'The Dark Knight', 'Batman fights the Joker', 2008, 1, NULL, 3, 4.99, 152, 19.99, 'PG-13', 'Trailers', default);
```

## Drawbacks

**1.** This is not a good practice, as it makes the query prone to errors. So always specify the column names.

**2.** This makes writing queries tedious, as while writing query you have to keep a track of what column was where. And even a small miss can lead to a big error.

**3.** If you don't specify column names, then you have to specify values for all the columns, including film_id, original_language_id and last_update, which we may want to keep NULL.

# Read

- Print ~ Select
- You may print constant data or data from other tables.
- Most used query

***< / > Syntax***

- Printing constant value :

```
SELECT constant_value;
```

- Printing data of whole table :

```
SELECT  *

FROM table;
```

**Students**

| id | first_name | last_name | psp |
|---|---|---|---|
| 1 | Virat | Kohli | 80 |
| 2 | Rahul | KL | 75 |
| 3 | Rohit | Sharma | 95 |
| 4 | Rahul | KL | 80 |

- Printing some columns from a table :

```
SELECT col1, col4

FROM table;
```

**Students**

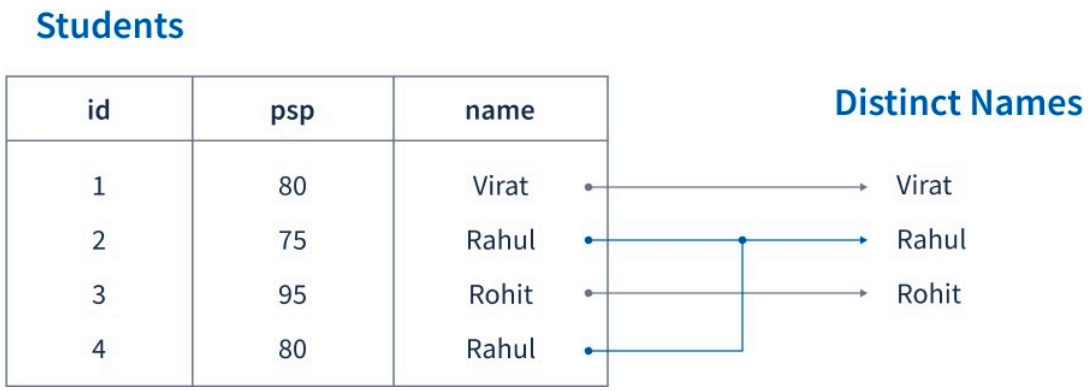
| id | first_name | last_name | psp |
|---|---|---|---|
| 1 | Virat | Kohli | 80 |
| 2 | Rahul | KL | 75 |
| 3 | Rohit | Sharma | 95 |
| 4 | Rahul | KL | 80 |

***< / > Pseudo-code***

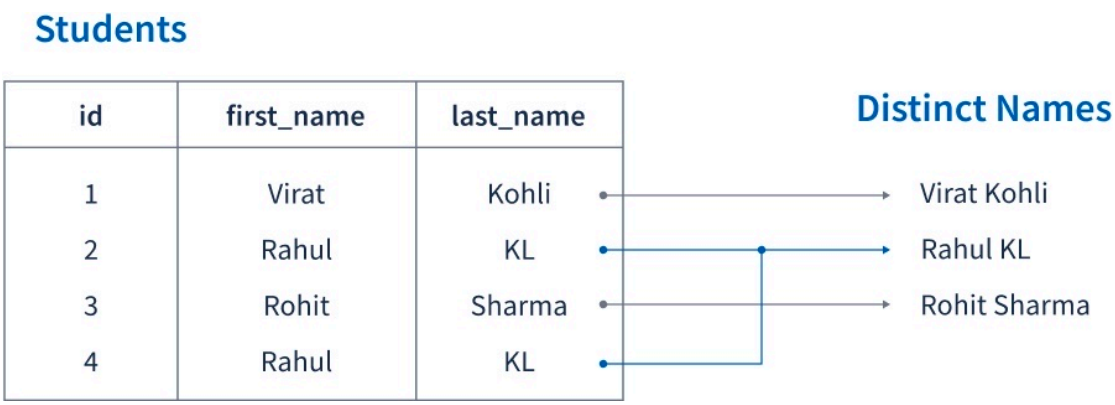
```
table_name : [ [ ] , [ ] , [ ] , [ ] ]

for row in table_name :

        ans.add( row )

for row in ans :

        print( row )
```

# Distinct ( Gives output of all unique values )

- Distinct pair of names :




- Distinct pair of first_name and last_name :



- It should be first word after SELECT.
- It can be applied on pair as well.
- Filters out duplicates.

### < / > *Syntax*

```
SELECT distinct, release_year, rating FROM film;
```

### < / > *Pseudo-code*

```
film : [  [ ] , [ ] , [ ] , [ ]  ]

for row in film :

        ans.add( row )

for row in ans :

        filtered_ans.add( row [release_year], row [rating] )

unique_ans = set (filtered_ans)

print (unique_ans)
```

# Inserting data from other table using SELECT

- We want to create a copy of students table where the data includes their id, name and last_name.

**Students**

| id | name | last_name | psp | attendance |
|---|---|---|---|---|
| 1 | Rohit | Sharma | 80 | 85 |
| 2 | Virat | Kohli | 75 | 85 |
| 3 | Shubhman | Gill | 95 | 95 |
| 4 | Rahul | KL | 92 | 85 |
| 5 | Rishabh | Pant | 80 | 88 |

**Students Copy**

| id | name | last_name |
|---|---|---|
| | | |

*' Should I add all the data one by one? '*

*' No ! We have a solution for that '*

**< / > *Syntax***

Code to insert data from existing table :

```
insert into students_copy(first_name, last_name)
SELECT first_name, last_name
FROM students;
```

*Where to insert*

*What to insert*

(CRUD - 2)

CRUD - 1

Until now → Selecting entire row or selective col.'s

# Where ( Similar to if condition )

table → rows, cols.

**Question :** Get all the movies with ' PG-13 ' ratings.

Sakila DB → film

**Note :** Assume if it is an array, how will you filter the data ? using If condition.

- We have where condition in SQL. → Similar to IF in programming

**Films**

| film_id | title | release_year | language | rating |
|---|---|---|---|---|
| 1 | KGF | 2018 | Kannada | PG |
| 2 | Kung Fu Panda | 2006 | English | G |
| 3 | Janghu 007 | 1947 | Bhojpuri | NC-17 |
| 4 | Kantara | 2022 | Kannada | PG-13 |

all row

< / > Syntax

all cols.

all rows

```
SELECT *
FROM film where rating = 'PG-13';
```

→ only those rows where rating = 'PG-13'

FROM (select)
↓
Where
↓
DISTINCT

< / > Pseudo-code

```
table_name : [ [ ], [ ], [ ], [ ] ]
ans = [ ]
for row in table_name :

        if row.matches( condition in where clause )

            ans.add( row )

for row in ans :

        print( row )
```

array of row

row

"where" clause

row['release_year']

```
SELECT DISTINCT release_year
FROM film
Where rating = 'PG-13';
```

```
filtered_answer = [ ]
for each row in answer
        filtered_answer.add(row['rating'])
unique_answer = set(filtered_answer)
return unique_answer
```

'where' clause

'Distinct' cond^n.

FROM
↓
WHERE
↓
DISTINCT
↓
ORDER BY

# AND, OR, NOT → combine more than one condition in "where" clause

- These are same as logical operators we have seen so far.
- ~~AND~~ = AND
- ~~OR~~ = OR
- NOT = < >, ! = , NOT

in programming → ! =

< > → SQL

All films with rating = 'PG-13' (1st) and release-year = 2006 (2nd)

and

```
Select * from film
where
rating = 'PG-13'  AND  release-year = 2016;
```

OR {either, or}

OR

< >
NOT . (=)

NOT rating != 'PG-13'

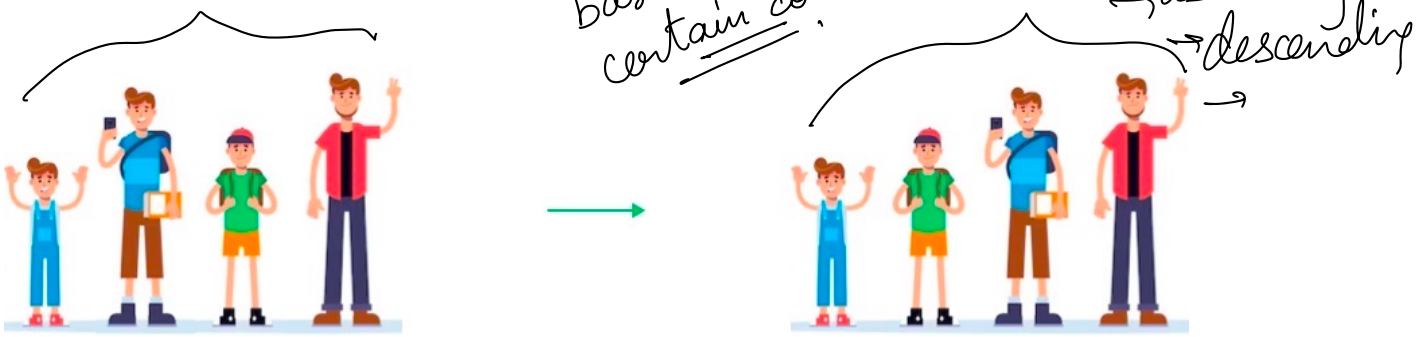
Select * from film where rating <> 'PG-13';

NOT rating = 'PG-13';

Select * from film where
(rating = 'PG-13' AND release-year = 2006) OR rental = 0.99;

NOT (rating <> 'PG-13' AND release-year <> 2006) OR rental = 0.99;

# Order By



- Order by clause allows to return value in a sorted order.
- By default the data is ordered in ascending order.

**Question :** Order the data in descending order according to rental_duration.

**< / > Syntax**

```
SELECT * FROM film ORDER BY rental_duration DESC;
```

- In case of tie, PK is always a tie-breaker.

Primary Key

| film_id | title | rental_duration |
|---|---|---|
| 1 | KGF | 1.5 hrs |
| 2 | Kung Fu Panda | 2.2 hrs |
| 3 | Janghu 007 | 3.5 hrs |
| 4 | Kantara | 2.2 hrs |

Ascending Order (rental_duration)

| film_id | title | rental_duration |
|---|---|---|
| 1 | KGF | 1.5 hrs |
| 2 | Kung Fu Panda | 2.2 hrs |
| 4 | Kantara | 2.2 hrs |
| 3 | Janghu 007 | 3.5 hrs |

- **Order By two column**

*< / > Syntax*

```
SELECT *
FROM film
ORDER BY title, release_year ;
```

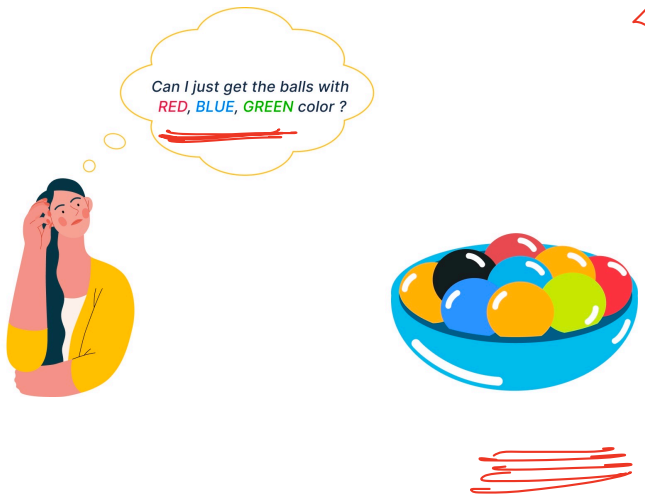
id
5 ← 2016 ✓
2 ← 2018 ✓

Tie (draw)

tie breaker

title, release-year ——— , — — — , —

# IN Operator




**Question :** Give data of all the students with batch_id 5 , 2 , 7 , 1 , 3.

***< / > Syntax***

```
SELECT  *
FROM    students
WHERE   batch_id = 5
    or    batch_id = 2
    or    batch_id = 7
    or    batch_id = 1
    or    batch_id = 3
```

- Here we can use IN operator instead of multiple OR operator.

< / > *Syntax*

```
SELECT  *
FROM    table
WHERE   value in ( list of values )
```

discrete values: rating IN (2, 2.5, 3, 4, 4.5, 5)

2, 2.5, 3, 4, 4.5, 5

range values: rating >= 2 AND rating <= 5

(*) SELECT * FROM film
WHERE
rating = 'PG-13' AND release_year = 2006 OR rental = 0.99;