Session 2 : Keys

```
create database school;

use school;

-- create batches and students table

create table batches(
 batch_id int PRIMARY KEY,
   batch_name varchar(50) not null);

create table students(
 student_id int auto_increment PRIMARY KEY,
   first_name varchar(50) not null,
   last_name varchar(50) not null,
   batch_id int,
   FOREIGN KEY (batch_id) references batches(batch_id) ON DELETE CASCADE ON
UPDATE CASCADE
   );


-- Inserting dummy data in both tables

insert into batches(batch_id, batch_name) values
(1, 'Batch A'),
(2, 'Batch B'),
(3, 'Batch C');

insert into students(first_name, last_name, batch_id) values
('John', 'Doe', 1),
('Jane', 'Doe', 1),
('Sony', 'Sonam', 2),
```

```
('Jenny', 'Smith', 3),
('Jack','Sparrow', 2);


delete from batches where batch_id = 1;


– If you want to update the structure/constraints of your table you may use
– alter command. Adding an example for that:

-- Alter table table_name
-- add foreign key(column_name)
-- references other_table(column_name_in_other_table)
-- ON delete cascade on update cascade
```

Session 3 : CRUD-1

```
-- CRUD 1

use sakila;

-- Inserting data with column names:
```

```sql
INSERT INTO film (title, description, release_year, language_id,
rental_duration, rental_rate, length, replacement_cost, rating,
special_features)
VALUES ('RRR', 'Before independene struggle', 2023, 1, 3, 4.99, 152, 19.99,
'PG-13', 'Trailers'),
      ('Bahubali', 'Son of Amrendra bahubali', 2015, 1, 3, 4.99, 165, 19.99,
'PG-13', 'Trailers'),
      ('Deadpool', 'The dead man walking', 2020, 1, 3, 4.99, 152, 19.99,
'PG-13', 'Trailers');


-- Inserting data without specifying column names:

INSERT INTO film
VALUES (default, 'The Scaler stories', 'SQL is awesome with this batch', 2024,
1, NULL, 3, 4.99, 152, 19.99, 'PG-13', 'Trailers', default);


-- Read queries using SELECT command:

-- Printing constant values

select 1;
select 'hello world';
select 'Rahul' as output;

-- Printing whole table:

select *
from film;

-- print data corresponding to film_id and tile, release_year
```

```sql
select film_id, title as movie_name, release_year
from film;

select 'Rahul'
from film;



-- Distinct
use sakila;

select release_year
from film;

select distinct release_year
from film;




-- Print distinct pair of release_year and rating

select release_year, rating
from film;

select distinct release_year, rating
from film;

-- Following query is not going to work:

-- select release_year, distinct rating
-- from film;
```

```sql
-- Get all movies along with their length in hours time

select title, round(length/60)
from film;



-- HW: Given length of each movie get total number of times a person can watch
it
-- in given rental_duration

-- Inserting data in a table using select command:

use school;

create table students_copy(
 student_id int auto_increment PRIMARY KEY,
 first_name varchar(50) not null,
   last_name varchar(50) not null);

insert into students_copy(first_name, last_name)
 select first_name, last_name
   from students;



-- Where condition:

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



-- Get all the movies released after 2006 and having rating PG-13
```

```sql
select title, rating, release_year
from film
where release_year > 2006 and rating = 'PG-13';

-- Get all the movies released in 2006 and have rating other than PG-13

select title, rating, release_year
from film
where release_year = 2006 and rating <> 'PG-13';

select title, rating, release_year
from film
where release_year = 2006 and rating != 'PG-13';

select title, rating, release_year
from film
where release_year = 2006 and not rating = 'PG-13';



-- Order by clause:


use school;

select *
from students;

use sakila;

select film_id, title, rental_duration
from film;
```

```sql
select film_id, title, rental_duration, length
from film
order by rental_duration desc, length;



-- In operator

-- Get all the movies which were released om either 2008, 2009, 2010, 2018,
2023, 2024

select title, release_year
from film
where release_year in(2008, 2009, 2010, 2018, 2023, 2024);



select *
from film
where release_year = 2010;
```

Session 4 : CRUD-2

```sql
-- CRUD 2

-- Betweeen

use sakila;
```

```sql
-- Get all the movies released between 2006 and 2016 both inclusive:

select *
from film
where release_year between 2006 and 2016;

select *
from film
where release_year >= 2006 and release_year <= 2016;



-- Get all the movies having names between A and B

select *
from film
where title between 'A' and 'B';




-- Like operator:

-- Get all the movies which are ending with word love

select *
from film
where title like '%love';

-- Get all the movies starting with word love?

select *
from film
where title like 'lOvE%';
```

```sql
-- Get all the movies with word love:

select *
from film
where title like '%love%';



-- Working with null values:

select 'Rahul' = 'Rahul';
select 'Rahul' = 'Rohit';

select null = null;
select 'rahul' = null;



-- Get full name of employees who haven't set their passwords yet?

select concat(first_name, ' ',  last_name) as full_name
from staff
where password is null;



select concat(first_name, ' ',  last_name) as 'full name'
from staff
where password is null;

-- Get full name of employees who are done with setting up their passwords

select concat(first_name, ' ',  last_name) as 'full name'
from staff
```

```sql
where password is not null;



-- Limiting the number of rows in output:

select *
from film;

-- Get top 2 rows from your query

select *
from film
limit 2;

-- Get last 10 rows of your query

select *
from film
order by film_id desc
limit 10;


-- Concept of offset

select *
from film
limit 10
offset 100;

select *
from film
limit 100, 10;
```

```sql
-- Update operation

use school;

update students_copy
set first_name = 'Rahul'
where student_id = 3;


-- Deletion operations

-- Delete

delete from students_copy
where student_id = 3;


-- Truncate

truncate students_copy;


-- Drop

drop table students_copy;
```

Session 5 : Joins-1

```sql
-- Joins 1

use school;



select *
from students
join batches
on students.batch_id = batches.batch_id;



select concat(s.first_name, ' ', s.last_name) full_name, b.batch_name
from students s
join batches b
on s.batch_id = b.batch_id;



select *
from batches
join students
on students.batch_id = batches.batch_id;




-- Quiz: What does JOIN command do in SQL?

-- Returns all rows from both tables
-- Returns only the matching rows between the tables
-- Returns all rows from the single table
```

```
-- None of the above



-- Quiz: When joining multiple tables, what is the purpose of the ON clause?
-- It specifies the conditions for joining the tables
-- It specifies the order of the tables to be joined
-- It filters the rows from the joined tables



-- Quiz: Which SQL statement combines rows from two tables where there is a
match in both tables?
-- SELECT * FROM table1 and table2 ON table1.id = table2.id;
-- SELECT * FROM table1 JOIN table2 ON table1.id = table2.id;
-- SELECT * FROM table1 JOIN table2 ON id;
-- None of the above

-- Quiz: Which one of the following is correct query to do a self join on
table based on column id?
-- Select * from table join table on table.id = table.id;
-- Select * from table t1 join table t2 on table.id = table.id;
-- Select * from table t1 join table t2 on t1.id = t2.id;
-- None of the above




-- Problem Statement: For every film get the name of actors who performed in
it.

-- film, actor, film_actor
-- film --> film_actor --> actor

use sakila;
```

```sql
select f.title, concat(a.first_name, ' ', a.last_name) full_name
from film f
join film_actor fa
on f.film_id = fa.film_id
join actor a
on a.actor_id = fa.actor_id;


-- Probelm statement:
--  Display a list of customers who rented a film in 'Horror'
--  category. Include name, last name, email, and film they rented.


-- Customer, rental, film, category, inventory, film_category

-- Customer --> rental --> inventory --> film --> film_category --> category

select concat(cu.first_name, ' ', cu.last_name) name, cu.email, f.title,
cg.name category
from customer cu
join rental r
on cu.customer_id = r.customer_id
join inventory i
on r.inventory_id = i.inventory_id
join film f
on i.film_id = f.film_id
join film_category fc
on fc.film_id = f.film_id
join category cg
on fc.category_id = cg.category_id
where cg.name = 'horror';
```

Session 6 : Joins-2

```
-- Joins 2

-- Compound joins

use sakila;

select f1.title, f1.release_year, f1.rental_rate, f2.title, f2.release_year,
f2.rental_rate
from film f1
join film f2
on (f2.release_year between f1.release_year - 2 and f1.release_year + 2)
 and (f2.rental_rate > f1.rental_rate);



-- Left Join:

use school;

select *
from students s
join batches b
on s.batch_id = b.batch_id;

select *
from students s
left join batches b
on s.batch_id = b.batch_id;
```

```sql
-- Right join:

select s.first_name, b.batch_name
from students s
right join batches b
on s.batch_id = b.batch_id;

select *
from students s
right join batches b
on s.batch_id = b.batch_id;



select *
from batches b
left join students s
on b.batch_id = s.batch_id;



-- Quiz 1: Which of the following is the correct syntax for a LEFT JOIN on
table1 with table2?
-- Ans: 3rd is correct.

-- SELECT * FROM table1 JOIN table2 ON table1.id = table2.id
-- SELECT * FROM table2 LEFT JOIN table1 ON table1.id = table2.id
-- SELECT * FROM table1 LEFT JOIN table2 ON table1.id = table2.id
-- None of the above



-- Quiz - 2: In a LEFT JOIN, if there are no matching rows in the right table:
-- Ans: Option 2 is correct
```

```sql
-- Those rows are excluded from the result.
-- Those rows are included with NULL values for columns from the right table.
-- The query returns an error.
-- Those rows are included with default values for columns from the right table.


-- Cross Join:

-- For every student get the possible batches that we can assign to them.

select s.first_name, b.batch_name
from students s
join batches b;

select s.first_name, b.batch_name
from students s
cross join batches b;


-- Implicit Join --> Cross join:

select s.first_name, b.batch_name
from students s, batches b;


select *
from students s
join batches b;



-- Using clause:
```

```sql
select *
from students s
join batches b
on s.batch_id = b.batch_id;


select *
from students s
join batches b
using(batch_id);



-- Natural join:

select *
from students s
natural join batches b;
```

Session 7 : Aggregate Functions

```sql
-- Aggregate Queries:

use school;

-- How many students have been assigned some batch?
select batch_id
from students;
```

```sql
select count(batch_id)
from students;


-- Get the count of unique batches assigned to students?

select distinct(batch_id)
from students;

select count(distinct(batch_id))
from students;

-- The below query will not result in same answer as of the above one.
select distinct(count(batch_id))
from students;


-- Get total number of students in students table?

select count(student_id)
from students;

-- Get total number of entries/rows/students in students table?

select count(*)
from students;

select 2
from students;

select 1
from students;
```

```sql
select count(1)
from students;



-- Other aggregate functions:

select max(psp), min(psp)
from students;

select min(psp)
from students;

select avg(psp)
from students;

select round(avg(psp), 2)
from students;


select sum(psp)
from students;


-- Quiz:
select avg(batch_id), sum(batch_id)/count(*)
from students;

select avg(batch_id)
from students;
-- 1.800
```

```sql
select sum(batch_id) / count(*)
from students;
-- 1.500


-- Nesting of aggregates --> aggregate(aggregate) isn't allowed:

-- Following query will not work:
-- select count(sum(psp))
-- from students;

select count(distinct(batch_id))
from students;

-- While printing an aggregate output don't print non aggregate in same select
query:

-- select max(psp), first_name
-- from students;

select max(psp), avg(psp)
from students;


-- Group by:

-- Get average psp of learners for each batch?

select avg(psp), batch_id
from students
group by batch_id;

select avg(psp), batch_id, max(psp)
```

```sql
from students
group by batch_id;



-- Get avg(psp) of learners along with their batch_id and batch_name

select avg(s.psp), b.batch_name, s.batch_id
from students s
join batches b
on s.batch_id = b.batch_id
group by s.batch_id;



-- Get batch_name which is having highest avg(psp) and the avg(psp)

select avg(s.psp), b.batch_name
from students s
join batches b
on s.batch_id = b.batch_id
group by s.batch_id
order by avg(psp) desc
limit 1;




-- Having:

-- Get all the batches with their avg(psp) where avg(psp) of the batch > 85

select avg(psp), batch_id
from students
group by batch_id;
```

```sql
-- 82, 89, 88.5, 78


-- We can't filter groups using where clause.
-- select avg(psp)
-- from students
-- group by batch_id
-- where avg(psp) > 85;


-- If we want to filter the groups, we should use Having clause:

select avg(psp), batch_id
from students
group by batch_id
having avg(psp) > 85;

-- Find out the batch_names as well?


-- Find out avg(psp) of batch_id = 1
-- Do this question using group by in HW.

select *
from students;

select *
from students
where batch_id = 1;

select avg(psp)
from students
where batch_id = 1;
```

Session 8 : Subqueries

```
-- Subqueries and Views:

-- Question 1: Get all the students having psp > psp of student with
student_id = 2

use school;

-- Step 1: Get psp of student with student_id = 2
select psp
from students
where student_id = 2;
-- x = 90

-- Step 2: Get all the students having psp > x
select *
from students
where psp > (
 select psp
 from students
 where student_id = 2);



-- Question 2: Find data of all students having psp > min(psp) of b_id 3.

-- Step 1: Find min(psp) of b_id = 3
select min(psp)
from students
where batch_id = 3;
-- x = 78
```

```sql
-- Step 2: Find all students having psp > x

select *
from students
where psp > (
 select min(psp)
 from students
 where batch_id = 3);


-- Question 3: Find all the years where avg(rental_rate) > global
avg(rental_rate)


use sakila;

-- Step 1: Global avg(rental_rate)
select avg(rental_rate)
from film;
-- 2.995


-- Step 2: Find all the years where avg(rental_rate) > x
select avg(rental_rate), release_year
from film
group by release_year
having avg(rental_rate) > (
 select avg(rental_rate)
 from film);
```

```sql
-- Subquery inside From clause
-- Question: Find data of all students where psp > (min(psp) among avg(psp) of
every batch)

use school;

-- Step 1: Find avg(psp) of every batch

select avg(psp)
from students
group by batch_id;
-- x = 82, 89, 88.5, 78


-- Step 2: Find min avg(psp) from x:
select min(psp)
from (
 select avg(psp) as psp
 from students
 group by batch_id) t1;
-- y = 78

-- Step 3:
select *
from students
where psp > (
 select min(psp)
 from (
   select avg(psp) as psp
   from students
   group by batch_id) t1);
```

```sql
-- ANY and ALL

-- Question: Find data of all learners where psp >= min(psp) of every batch

-- Step 1: Find min(psp) of every batch

select min(psp)
from students
group by batch_id;
-- x = 82, 88, 85, 78

-- Step 2: Find all the students having psp >= all values in x

select *
from students
where psp >= ALL (
 select min(psp)
 from students
 group by batch_id);



select *
from students
where psp > ANY (
 select min(psp)
 from students
 group by batch_id);



-- Co-related Subqueries:

-- Question: Find all the students where psp > avg(psp) of their batch
```

```sql
-- Step 1:
-- For John --> batch_id = 1
select avg(psp)
from students
where batch_id = 1;
-- 89

-- For student with batch_id = 2
select avg(psp)
from students
where batch_id = 2;



-- Step 2:

select *
from students s
where psp > (
 select avg(psp)
 from students
 where batch_id = s.batch_id);




-- Views:

use sakila;

select f.title, concat(a.first_name, ' ', a.last_name) actor_name
from film f
join film_actor fa
```

```sql
on f.film_id = fa.film_id
join actor a
on a.actor_id = fa.actor_id;

select f.title, concat(a.first_name, ' ', a.last_name) actor_name
from film f
join film_actor fa
on f.film_id = fa.film_id
join actor a
on a.actor_id = fa.actor_id
where f.title = 'AFRICAN EGG';

-- Create a view:

create or replace view actor_film_info as
select f.title, concat(a.first_name, ' ', a.last_name) actor_name
from film f
join film_actor fa
on f.film_id = fa.film_id
join actor a
on a.actor_id = fa.actor_id;



select *
from actor_film_info
where title = 'AFRICAN EGG';
```

Session 9 : Indexing

```
-- Indexing:

use sakila;

desc film;

select *
from film;

explain select *
from film
where description = 'It was fun';

explain select *
from film
where title = 'ALABAMA DEVIL';


explain select *
from film
where film_id = 40;


explain select *
from film
where description = 'It was fun';
-- Table scan on film  (cost=103 rows=1000) (actual time=0.127..1.17 rows=1004
loops=1)
```

```
explain analyze select *
from film
where film_id = 40;
-- Rows fetched before execution  (cost=0..0 rows=1) (actual
time=83e-6..166e-6 rows=1 loops=1)


desc customer;

explain select *
from customer
where first_name = "MARIA";

explain select *
from customer
where last_name = "Janghu";

explain select *
from customer
where last_name = "Janghu" and first_name = "LISA";


explain select *
from customer
where customer_id = 10 and last_name = "TAYLOR";


drop index idx_title on film;

-- Without index search:
explain analyze select *
from film
where title = "ALI FOREVER";
```

```
--  Table scan on film  (cost=103 rows=1000) (actual time=0.23..1.33 rows=1004
loops=1)

-- Index on full string:
create index idx_title
on film(title);

explain analyze select *
from film
where title = "ALI FOREVER";
-- Index lookup on film using idx_title (title='ALI FOREVER')  (cost=0.35
rows=1) (actual time=0.121..0.128 rows=1 loops=1)



-- Index on first 5 character of title:
create index idx_title
on film(title(5));

explain analyze select *
from film
where title = "ALI FOREVER";
-- Index lookup on film using idx_title (title='ALI FOREVER')  (cost=0.35
rows=1) (actual time=0.086..0.091 rows=1 loops=1)



-- Create index on first 1, 2, 3, 4 characters and check their cost
```

Session 10 : Transactions-1:

```
-- Transactions 1

-- Left side session:

show variables like 'autocommit';
set autocommit = 0;

show variables like 'transaction_isolation';

use sakila;

start transaction;

select *
from film
where film_id = 5;
-- AFRICAN EGG

update film
set title = 'Rohan 007'
where film_id = 5;

commit;



-- Dirty Read

start transaction;
```

```sql
select *
from film
where film_id = 5;
-- Rohan 007

update film
set title = 'Rahul 007'
where film_id = 5;

select *
from film
where film_id = 5;
-- Rahul 007

rollback;

commit;

select *
from film
where film_id = 5;
-- Rohan 007
```

```sql
-- Transactions 1

-- Right side session:
```

```sql
show variables like 'autocommit';
set autocommit = 0;

show variables like 'transaction_isolation';

SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

use sakila;

start transaction;

select *
from film
where film_id = 5;
-- AFRICAN EGG

-- After some time

select *
from film
where film_id = 5;
-- Rohan 007

commit;



start transaction;

select *
from film
where film_id = 5;
-- Rohan 007
```

```
-- Doing some work

select *
from film
where film_id = 5;
-- Rahul 007

commit;

-- We ended up with dirty read here:
select *
from film
where film_id = 5;
-- Rohan 007
```

Session 11 : Transactions-2

```
-- Transaction 2:

-- Left side transaction
-- Read Committed

use sakila;

show variables like 'autocommit';
set autocommit = 0;
```

```
start transaction;

select *
from film
where film_id = 5;
-- Rohan 007

update film
set title = 'Janghu 007'
where film_id = 5;

select *
from film
where film_id = 5;
-- Janghu 007

commit;


-- Repeatable Read:

start transaction;

select *
from film
where film_id = 5;
-- Bond 007

update film
set title = 'RRR'
where film_id = 5;

select *
```

```
from film
where film_id = 5;

commit;



-- Phantom Read

use school;

start transaction;

select *
from students
where psp < 80;
-- Jim, Rahul, Tom

-- New entry is being inserted:

insert into students(first_name, last_name, batch_id, psp)
 values('Angelina', 'Jolie', 3, 73);

commit;



-- Transactions 2:
-- Read Committed:

show variables like 'autocommit';
set autocommit = 0;
```

```sql
show variables like 'transaction_isolation';

SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;

use sakila;

start transaction;

select *
from film
where film_id = 5;
-- Rohan 007

-- After 5 mins

select *
from film
where film_id = 5;
-- Rohan 007

commit;

-- After commit in left session:

select *
from film
where film_id = 5;
-- Janghu 007


-- Repeatable Read
```

```
show variables like 'transaction_isolation';
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;



start transaction;

select *
from film
where film_id = 5;
-- Bond 007 --> snap

-- Doing some work

select *
from film
where film_id = 5;
-- Bond 007

commit;



select *
from film
where film_id = 5;
-- RRR



-- Quiz: Suppose there are two transactions T1 and T2 working concurrently on
Students table.
-- T2 reads name at id = 1 as X, meanwhile T1 updates the value of id = 1 to Y
and commited the change.
-- Now if T2 reads name at id = 1 what will be the output?
-- Isolation level of T2 is Read Committed.
```

```
-- Ans: Y

-- X
-- Y
-- Dirty Read
-- Error


-- Quiz: In an inventory management system, which isolation level ensures that
a
-- transaction reading available stock quantities does not see changes made by
-- concurrent transactions until it completes?
-- Ans:

-- Read Uncommitted
-- Read Committed
-- Repeatable Read


-- Phantom Read:

use school;

start transaction;

select *
from students
where psp < 80;
-- Jim, Rahul, Tom

-- Some work

-- Updating newly inserted data here:
```

```
update students
set first_name = 'Flower'
where student_id = 8;

select *
from students
where psp < 80;
-- Jim, Rahul, Tom, Flower
```

Revision:

```
-- Revision:

-- Get list of all films along with actor names and their categories.

-- From where do we need to get data from?
-- films, actors, film_actor, category, film_category

-- Get the order of joins.
-- film --> film_category --> category --> film_actor --> actor
use sakila;

select f.title, a.first_name, c.name
from film f
join film_category fc
on f.film_id = fc.film_id
join category c
on c.category_id = fc.category_id
join film_actor fa
on f.film_id = fa.film_id
```

```
join actor a
on a.actor_id = fa.actor_id;

-- Homework:
-- Get total movies done by each actor? --> group by, aggregate
-- Actors who belongs to action genre?
```