Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

CS312 Database Management Systems

School of Computer Engineering and Technology

CS312 Database Management Systems

Teaching Scheme
Theory: 3 Hrs / Week

Credits: 02 + 01
Practical: 2Hrs/Week

- **Course Objectives:**

1) Understand and successfully apply logical database design principles, including E-R diagrams and database normalization.
2) Learn Database Programming languages and apply in DBMS application
3) Understand transaction processing and concurrency control in DBMS
4) Learn database architectures, DBMS advancements and its usage in advance application

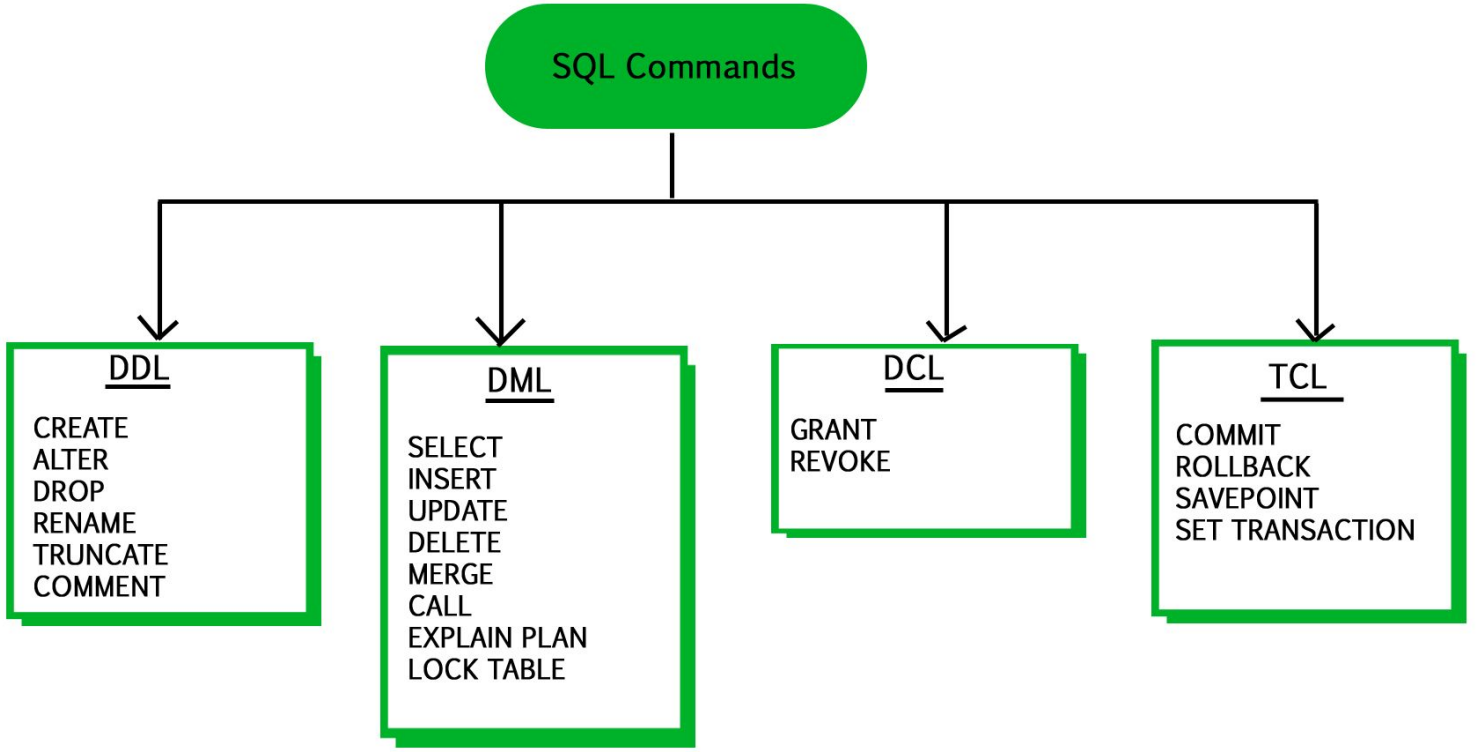
- **Course Outcomes:**

1) Design ER-models to represent simple database application scenarios and Improve the database design by normalization.
2) Design Database Relational Model and apply SQL , PLSQL concepts for database programming
3) Describe Transaction Processing and Concurrency Control techniques for databases
4) Identify appropriate database architecture for the real world database application

SQL- DDL commands(Create, Alter, Drop, Truncate, Rename, Describe) ,DCL(Grant, Revoke)

LABORATORY ASSIGNMENT NO: 04

SQL Statements Categories



SQL Joins

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the **from** clause
- **Join condition** – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

Index

- Indices are data structures used to speed up access of records with specified values for index attributes.
- Indexes are used to find rows with specific column values quickly.
- Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. (Sequential Scan)
- If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data.
- This is much faster than reading every row sequentially
- MySQL create default indexes on PRIMARY KEY, UNIQUE KEY
- User defined index can be created using CREATE INDEX COMMAND

SQL Joins : Cross Join

- Cross JOIN is a **simplest form of JOINs** which matches each row from one database table to all rows of another as a Cartesian product.
- The cross join does not establish a relationship between the joined tables.
- SELECT * FROM `Movies` CROSS JOIN `Artist` OR
- SELECT * FROM `Movies` ,`Artist`;

Movies

Movie_id	Title	Category
1	ASSASSIN'S CREED:	Animations
2	Real Steel(2012)	Animations

Artist

Id	First_name	Last_name	Movie_id
1	Adam	Smith	1
2	Ravi	Kumar	2

Cross Join of 2 tables

Movie_id	Title	Category	Id	First_name	Last_name	Movie_id
1	ASSASSIN'S CREED:	Animations	1	Adam	Smith	1
1	ASSASSIN'S CREED:	Animations	2	Ravi	Kumar	2
2	Real Steel(2012)	Animations	1	Adam	Smith	1
2	Real Steel(2012)	Animations	2	Ravi	Kumar	2

SQL Joins : Inner Join

- The inner JOIN is used to return rows from both tables that satisfy the given condition(join condition on common column).
- SELECT * FROM movies INNER JOIN `Artist` on movies.movie_id` = Artist.movie_id
- OR

  SELECT * FROM movies ,Artist WHERE movies.movie_id = Artist.movie_id

Movie_id	Title	Category	Id	First_name	Last_name	Movie_id
1	ASSASSIN'S CREED:	Animations	1	Adam	Smith	1
2	Real Steel(2012)	Animations	2	Ravi	Kumar	2

SQL Joins : Outer Join

- MySQL Outer JOINs return all records matching from both tables .It can detect records having no match in joined table. It returns **NULL** values for records of joined table if no match is found.

```
SELECT A.title , B.first_name , B.last_name
FROM movies "A" LEFT OUTER JOIN  Artist " B"
ON B.`movie_id` = A. 'movie_id'
# Some SQL Support keyword : Left join/natural left outer join
```

OR

```
SELECT A.title , B.first_name , B.last_name
FROM movies "A" LEFT OUTER JOIN  Artist " B" USING ( `movie_id` )
```

The LEFT JOIN returns all the rows from the table on the left even if no matching rows have been found in the table on the right.
Where no matches have been found in the table on the right, NULL is returned.

What will Right Outer return?

What will full outer return?

Left outer join Output (contd..)

Movie_id	Title	Category	Id	First_name	Last_name	Movie_id
1	ASSASSIN'S CREED:	Animations	1	Adam	Smith	1
2	Real Steel(2012)	Animations	2	Ravi	Kumar	2
3	**Jurassic Park**	**Animation**				

Title	First_name	Last_name
ASSASSIN'S CREED:	Adam	Smith
Real Steel(2012)	Ravi	Kumar
Jurassic Park	Null	Null

SQL Joins

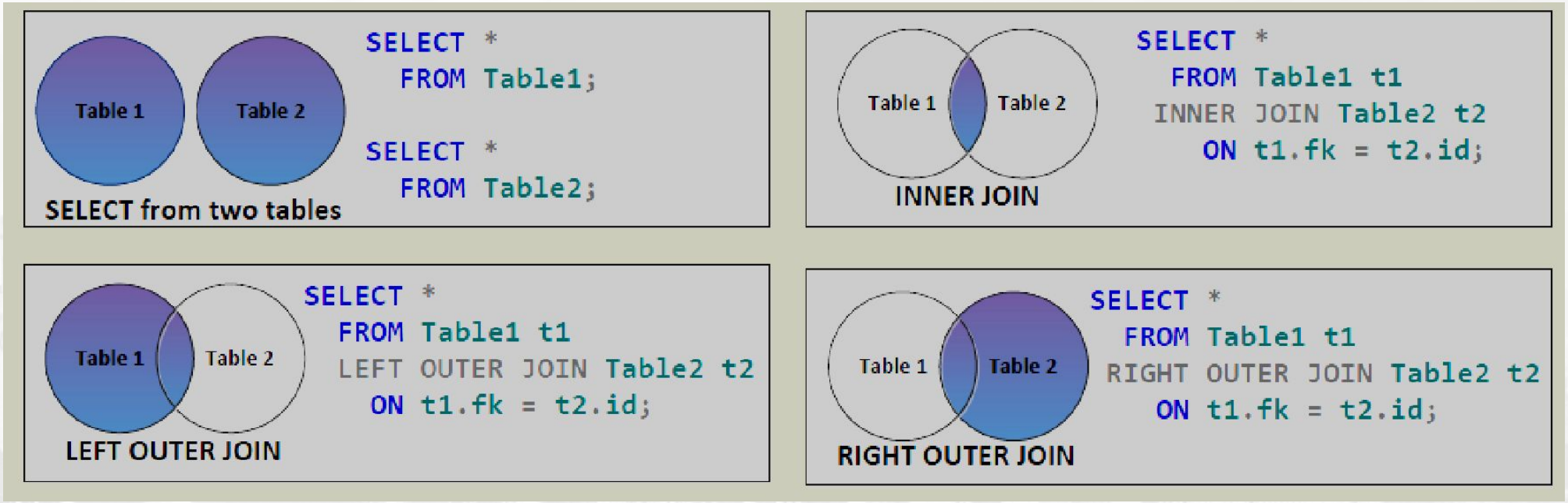
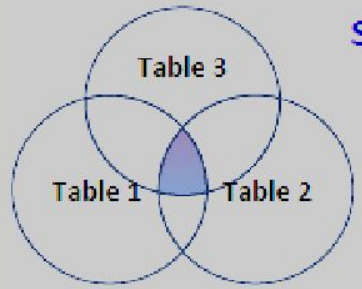


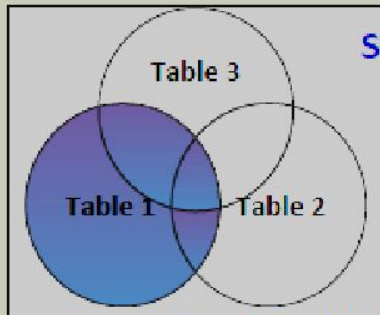
Table 1
Table 2
SELECT from two tables
SELECT *
FROM Table1;
SELECT *
FROM Table2;
Table 1
Table 2
INNER JOIN
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.fk = t2.id;
Table 1
Table 2
LEFT OUTER JOIN
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id;
Table 1
Table 2
RIGHT OUTER JOIN
SELECT *
FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
ON t1.fk = t2.id;

SQL Joins



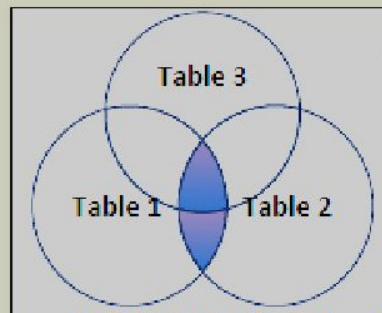
Two INNER JOINs

```
SELECT *
  FROM Table1 t1
INNER JOIN Table2 t2
   ON t1.fk = t2.id
INNER JOIN Table3 t3
   ON t1.fk_table3 = t3.id;
```



Two LEFT OUTER JOINS

```
SELECT *
  FROM Table1 t1
LEFT OUTER JOIN Table2 t2
   ON t1.fk = t2.id
LEFT OUTER JOIN Table3 t3
   ON t1.fk_table3 = t3.id;
```



INNER JOIN and a LEFT OUTER JOIN

```
SELECT *
  FROM Table1 t1
INNER JOIN Table2 t2
   ON t1.fk = t2.id
LEFT OUTER JOIN Table3 t3
   ON t1.fk_table3 = t3.id;
```

Join operations – Example

Relation

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

Relation *prereq*

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that

  prereq relation is missing for CS-315 and

  course relation is missing for CS-347

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values.

Left Outer Join And Right Outer Join

course **natural left outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

- *course* **natural right outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

course

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Full Outer Join

- *course* **natural full outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

course

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Joined Relations – Examples

- The **difference** is in **natural join** no need to specify condition but in **inner join** condition is mandatory.
- The repeated column is **avoided** in the **output** in natural join.
- *Select * from course natural join prereq*

*Select * from course* **inner join** *prereq* **on** *course.course_id = prereq.course_id*

course_id	title	dept_name	credits	prereq_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

What is the difference between the above, and a natural join?

- *Select * from course* **left outer join** *prereq* **on** *course.course_id = prereq.course_id*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Joined Relations – Examples

- *course* **full outer join** *prereq* **using** (*course_id*)

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

Aggregate Functions

Type	Use	Functions
Single –row functions	Operate on a single column of a relation of single row n the table returning single value as an output	String functions, Date Functions
Multiple –row functions	Act on a multiple row in the relation returning single value as an output	Avg, min, max, sum, count

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

Aggregate Functions Examples

Find the average salary of instructors in the Computer Science department

- **select avg** (*salary*),min(salary), max(salary),sum(salary)
  from *instructor*
  where *dept_name*= 'Comp. Sci.';

Find the number of tuples in the *course* relation

- **select count** (*) **from instructor**;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Aggregate Functions – Group By

Find the average salary of instructors in each department

- **select** *dept_name*, **avg** (*salary*) **as** *avg_salary*
  from *instructor*
  group by *dept_name*;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregation (Cont.)

Attributes in **select** clause outside of aggregate functions must appear in **group by** list

- /* erroneous query */

  select *dept_name*, *ID*, **avg** (*salary*)
  from *instructor*
  group by *dept_name*;

Discuss why query is erroneous, [Hint :refer last table]

Aggregate Functions – Having Clause

Find the names and average salaries of all departments whose average salary is greater than 42000

select *dept_name*, **avg** (*salary*) **as** *avg_salary*
from *instructor*
group by *dept_name*
having avg (*salary*) > 42000;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

Null Values and Aggregates

- To find the total all salaries

  select sum *(salary)* ***from*** *instructor*

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount

- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
  - What if collection has only null values?
    - count returns 0
    - all other aggregates return null

Subqueries (Nested Query)

- A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.
- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

```
SELECT ProductID,
       Name,
       ListPrice
FROM   production.Product
WHERE  ListPrice > (SELECT AVG(ListPrice)
                    FROM   Production.Product)
```

subquery

Examples of Subquery in DML and Select

- SQL> SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID FROM CUSTOMERS WHERE SALARY > 4500) ;
- SQL> INSERT INTO CUSTOMERS_BKP SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID FROM CUSTOMERS) ;
- SQL> UPDATE CUSTOMERS SET SALARY = SALARY * 0.25 WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP WHERE AGE >= 27);
- SQL> DELETE FROM CUSTOMERS WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP WHERE AGE >= 27);

Subqueries in the From Clause

- Find the average instructors' salaries of those departments where the average salary is greater than $42,0

```
select dept_name, avg_salary
from ( select dept_name, avg (salary) as avg_salary
        from instructor
        group by dept_name)
where avg_salary > 42000;
```

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Another way to write above query

```
select dept_name, avg_salary
from ( select dept_name, avg (salary)
        from instructor
        group by dept_name)
         as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

Set Operations

Set operations are **union**, **intersect**, and **minus**

- Each of the above operations automatically eliminates duplicates

To retain all duplicates use the keyword all

- **union all,**
- **intersect all**
- **Minus**

ID	NAME
1	ABHI
2	SAMEER
3	SAMEER
4	JAVED

- Select name from table1 union select name from table2;

*Select * from table1 union select * from table 2;*

table1

ID	NAME
1	ABHI
2	SAMEER
3	SAMEER

table2

ID	NAME
3	SAMEER
4	JAVED

Set Operations -examples

- *Select * from table1 intersect select * from table 2;*

ID	NAME
3	SAMEER

- *Select * from table1 minus select * from table 2;*

ID	NAME
1	ABHI
2	SAMEER

- *Select * from table1 union all select * from table 2;*

table1

ID	NAME
1	ABHI
2	SAMEER
3	SAMEER

table2

ID	NAME
3	SAMEER
4	JAVED

ID	NAME
1	ABHI
2	SAMEER
3	SAMEER
3	SAMEE

Set Membership

Find courses offered in Fall 2017 and in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
        course_id in (select course_id
                      from section
                      where semester = 'Spring' and year= 2018);
```

Find courses offered in Fall 2017 but not in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
        course_id  not in (select course_id
                           from section
                      where semester = 'Spring' and year= 2018);
```

Set Membership (Cont.)

- Name all instructors whose name is neither “Mozart” nor Einstein”

select distinct *name*
from *instructor*
where *name* **not in** ('Mozart', 'Einstein')

instructor

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Test for Empty Relations

- EXISTS and NOT EXISTS are used with a subquery in WHERE clause to examine if the result the subquery returns is TRUE or FALSE.
- The true or false value is then used to restrict the rows from outer query select.
- As EXISTS and NOT EXISTS only return TRUE or FALSE in the subquery, the SELECT list in the subquery does not need to contain actual column name(s).

Continued…

- SELECT * FROM customers WHERE EXISTS (SELECT * FROM order_details WHERE customers.customer_id = order_details.customer_id);
- SELECT * FROM customers WHERE NOT EXISTS (SELECT * FROM order_details WHERE customers.customer_id = order_details.customer_id);

- Insert,update,delete commands can also be used with EXISTS commands

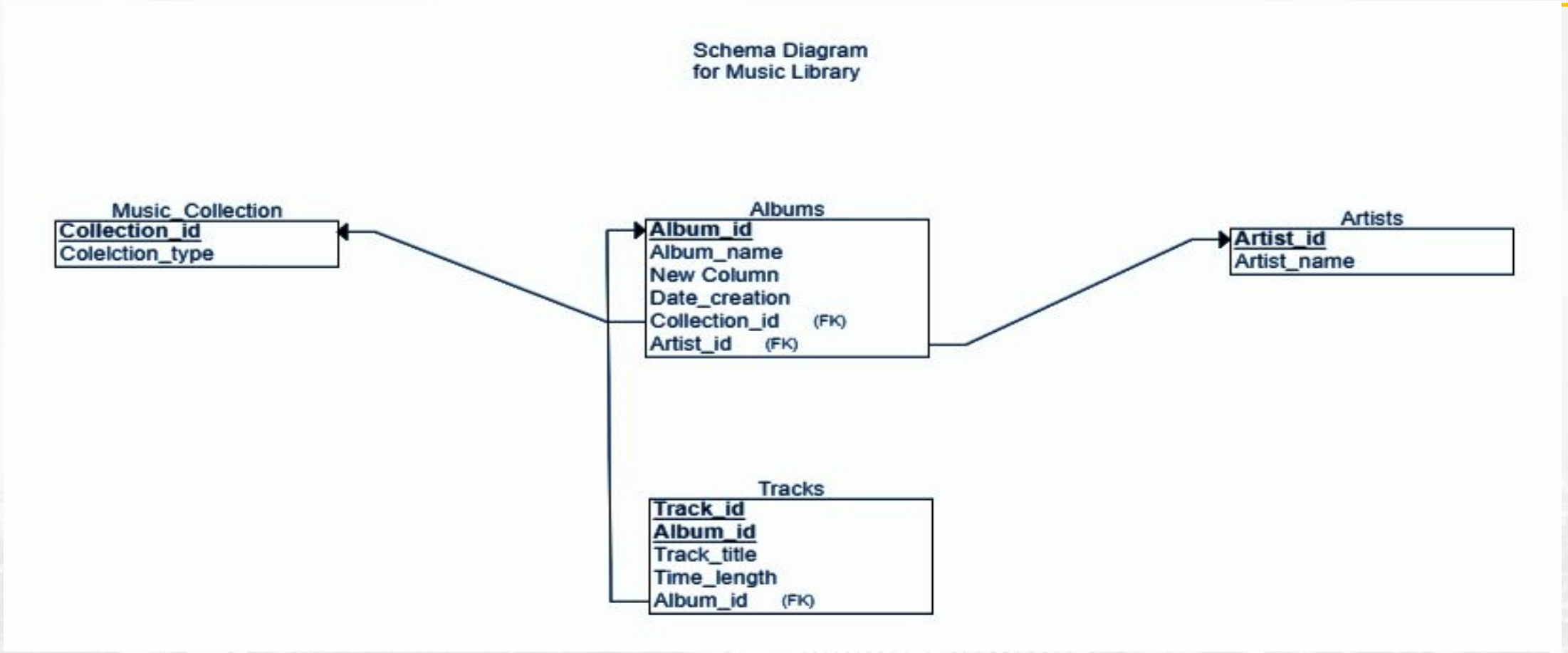
- INSERT INTO contacts (contact_id, contact_name) SELECT supplier_id, supplier_name FROM suppliers WHERE EXISTS (SELECT * FROM orders WHERE suppliers.supplier_id = orders.supplier_id);
- Delete from contacts SELECT supplier_id, supplier_name FROM suppliers WHERE EXISTS (SELECT * FROM orders WHERE suppliers.supplier_id = orders.supplier_id);

Perform Join Subqueries

SQL Queries on: Functions-Single Row, Aggregate Functions, Data Sorting, Subquery, Joins(Inner, Outer, Natural, Self), Group by-Having, Set Operations, View.TCL Commands (Rollback, Commit, Savepoint)

Exercises -Batch A

Schema Diagram for Music Library Database



Note : Music Collection Entity can be optional

Exercises

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constraints, not null etc.**
  Solve the following queries:
  Music_Collection(Collection_id,Collection_type)
  Album(Album_id,Album_name,collection_id,artist_id,year_creation)
  Artists(Artist_id,Artist_name)
  Tracks(Track_id,Track_name,album_id,price,time_length)

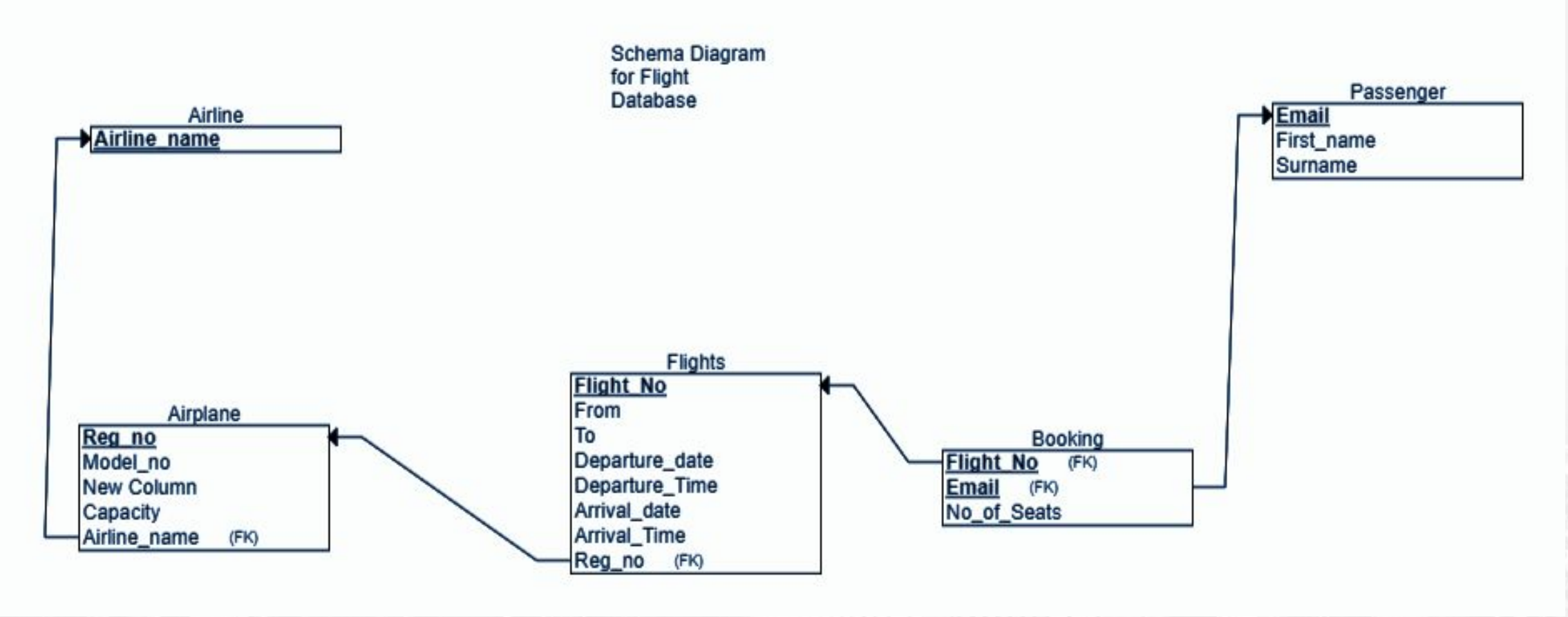
1. List the number of tracks for every album_name.
2. Display the Artist_name and Number of Albums compiled by him/her only if the number of albums compiled are more than 1
3. Display the Names of Artists only if they have compiled any albums.(Use EXISTS Operator)
4. Display the album_name,track_name if the track time_length is having the longest duration.
5. Display the Names of albums with collection_type ‘Jazz’.
6. Display the names of artist who have compiled maximum albums in a period of 2 years
7. Display all the tracks ids and names from albums composed by artists having the substring ‘a’ in their names.
8. Display the abum_id and track_name for tracks having track duration greater than the average track length in overall albums.(Use Subqueries)
9. Create a view for storing information about the Artist_name,Album_name and creation year.
10. Display the names of tracks with price greater than at least few other tracks

Perform Join and Subqueries

SQL Queries on: Functions-Single Row, Aggregate Functions, Data Sorting, Subquery, Joins(Inner, Outer, Natural, Self), Group by-Having, Set Operations, View.TCL Commands (Rollback, Commit, Savepoint)

Exercises -Batch B

Schema Diagram for Flight Reservation Database



Note: Airline Entity can be optional

Exercises

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constraints, not null etc.**
  Solve the following queries:
  Airline(Airline_name)
  Airplane(Reg_No,Model_no,Capacity,AirlineName)
  Flights(Flight_No,From,To,Departure_date,Departure_time,Arrival_date,Arrival_time,AirplaneRegNo)
  Passenger(Email,First_name,surname)
  Booking(p_email,flight_no,no_seats)

1. Display the Passenger email ,Flight_no,Source and Destination Airport Names for all flights booked
2. Display the flight and passenger details for the flights booked having Departure Date between 23-08-2021 and 25-08-2021
3. Display the top 5 airplanes that participated in Flights from 'Mumbai' to 'London' based on the airplane capacity

4.Display the passenger first names who have booked the no_of seats smaller than the average number of seats booked by all passengers for the arrival airport:"New Delhi"

Exercises

5. Display the surnames of passengers who have not booked a flight from “Pune” to “Bangalore”

6.Display the Passenger details only if they have booked flights on 21st July 2021. (Use Exists)

7. Display the Flight-wise total time duration of flights if the duration is more than 8 hours (Hint : Date function,Aggregation,Grouping)

8. Display the Airplane-wise average seating capacity for any airline

9. Display the total number of flights which are booked and travelling to “London” airport.

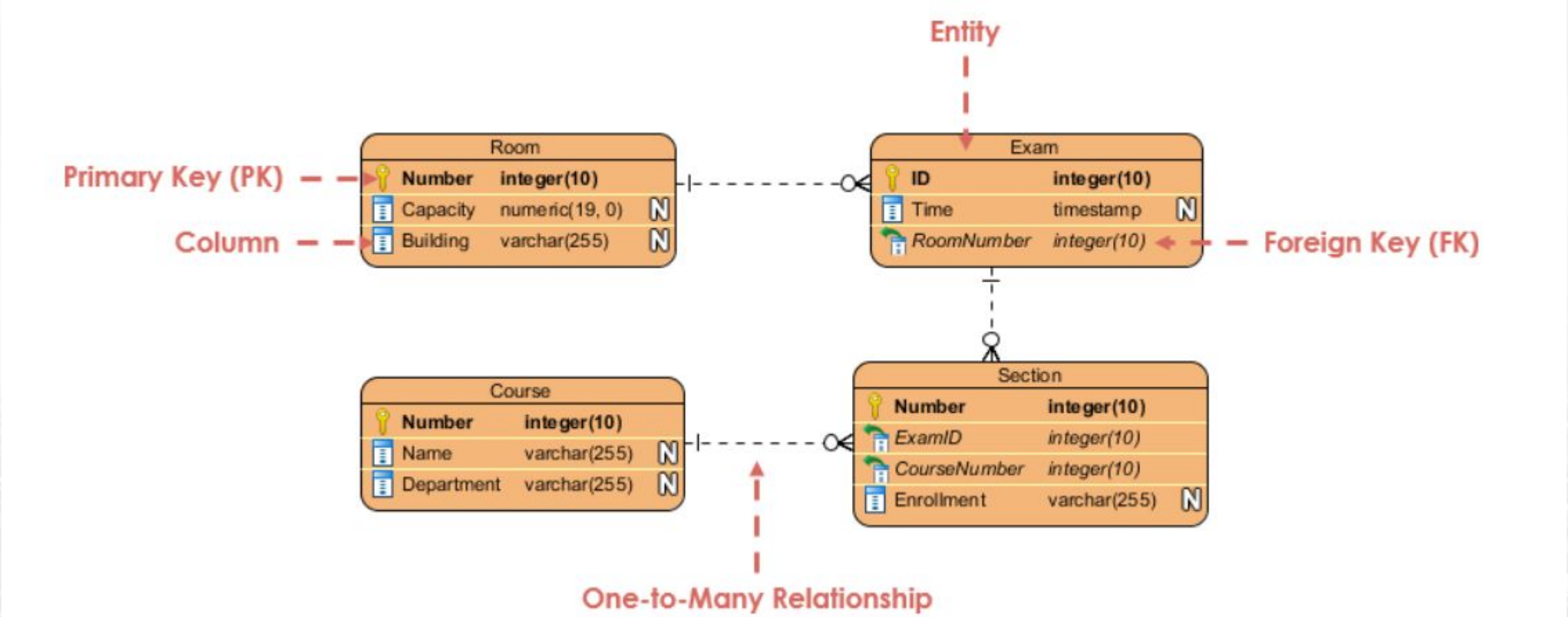
10. Create a view having information about flight_no,airplane_no,capacity.

Perform Join and Subqueries

SQL Queries on: Functions-Single Row, Aggregate Functions, Data Sorting, Subquery, Joins(Inner, Outer, Natural, Self), Group by-Having, Set Operations, View.TCL Commands (Rollback, Commit, Savepoint)

Exercises -Batch C

University Database System



Exercises 01

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.**
  - Room(r-number, capacity, building) r-number is primary key
  - Course(c-number name, department) c-number is primary key
  - Section(C-number, s-number ,enrollment) C-number, s-number is primary key
  - Exam(C-number, s-number,r-number, time)

Solve the following queries

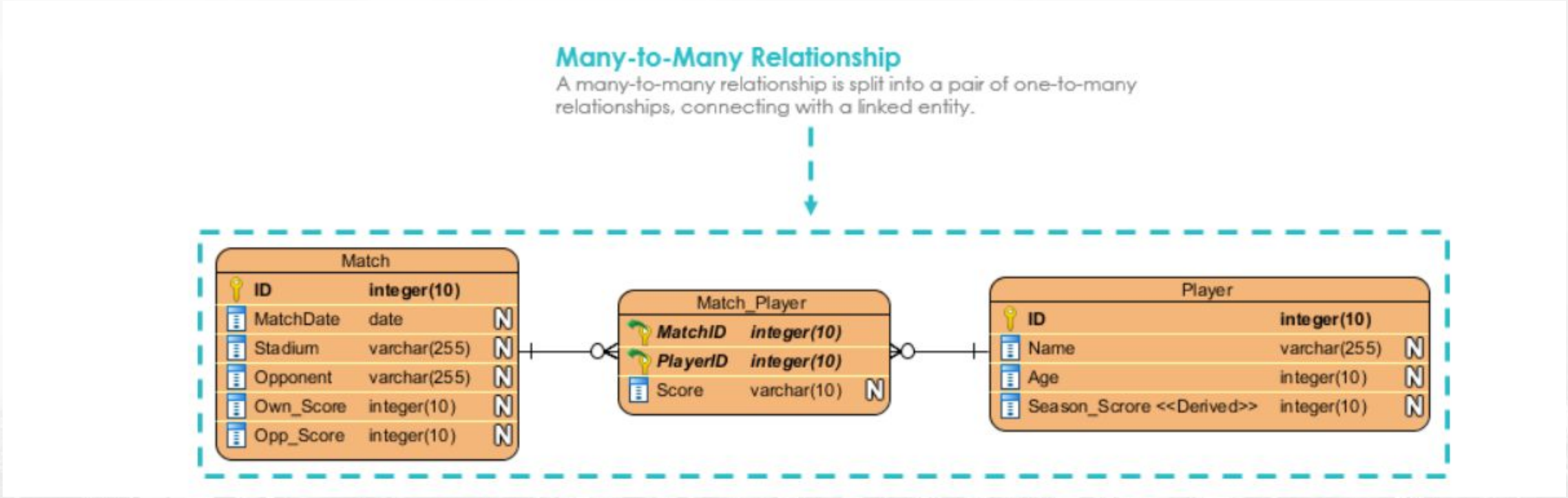
1. List the course and no. of sections in each Course.
2. List the course and no. of sections in each Course in CET department.
3. Display the course number and no of sections in each course in CET department where no of sections are more than 5;
4. Display c-number, name ,department of such courses whose exam is conducted in 'A' building.
5. Get exam details of course 'DBMS';
6. Display the exam room number, its capacity and building for course 'DBMS'
7. Display all the courses whose total enrollment is greater than total enrollment of course 'DBMS'.
8. Display all the courses whose total enrollment is greater than total enrollment of every course in CET department.
9. Display course no, name, department , section and enrollment of each course.
10. Create one view

Perform Join and Subqueries

SQL Queries on: Functions-Single Row, Aggregate Functions, Data Sorting, Subquery, Joins(Inner, Outer, Natural, Self), Group by-Having, Set Operations, View.TCL Commands (Rollback, Commit, Savepoint)

Exercises -Batch D

Favorited Team Statistics



Exercise

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.**
  - **Match(MatchID,MatchDate,Stadium,opponent,Own_Score,Opp_Score)**
  - **Player(PlayerID,Name,Age)**
  - **Match_Player(MatchID, PlayerID,Score)**

Solve the following queries

1. Get the details (Name and Age) of all players who played in match M1.
2. Get the details of all matches(match detail) in which “Sachin Tendulkar” has played.
3. Get the no of matches played by each player in stadium “Narendra Modi Stadium”.
4. Get the details of players (Name and Age) and matches they have played.
5. Get the details of all players (Name and Age) who have score greater than 250
6. List the names of all players who have played a match and scored 0 order by MatchID and PlayerID.
7. Get the details of all players whose score is larger than the score of every player in MatchID M3.
8. Get the details of all players whose score is larger than the score of at least one player in MatchID M3.
9. For each match that has more than 5 players, retrieve the MatchID and the number players whose age greater than 30 in order of their MatchID.
10. Create one View.

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Thank You!