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

- **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

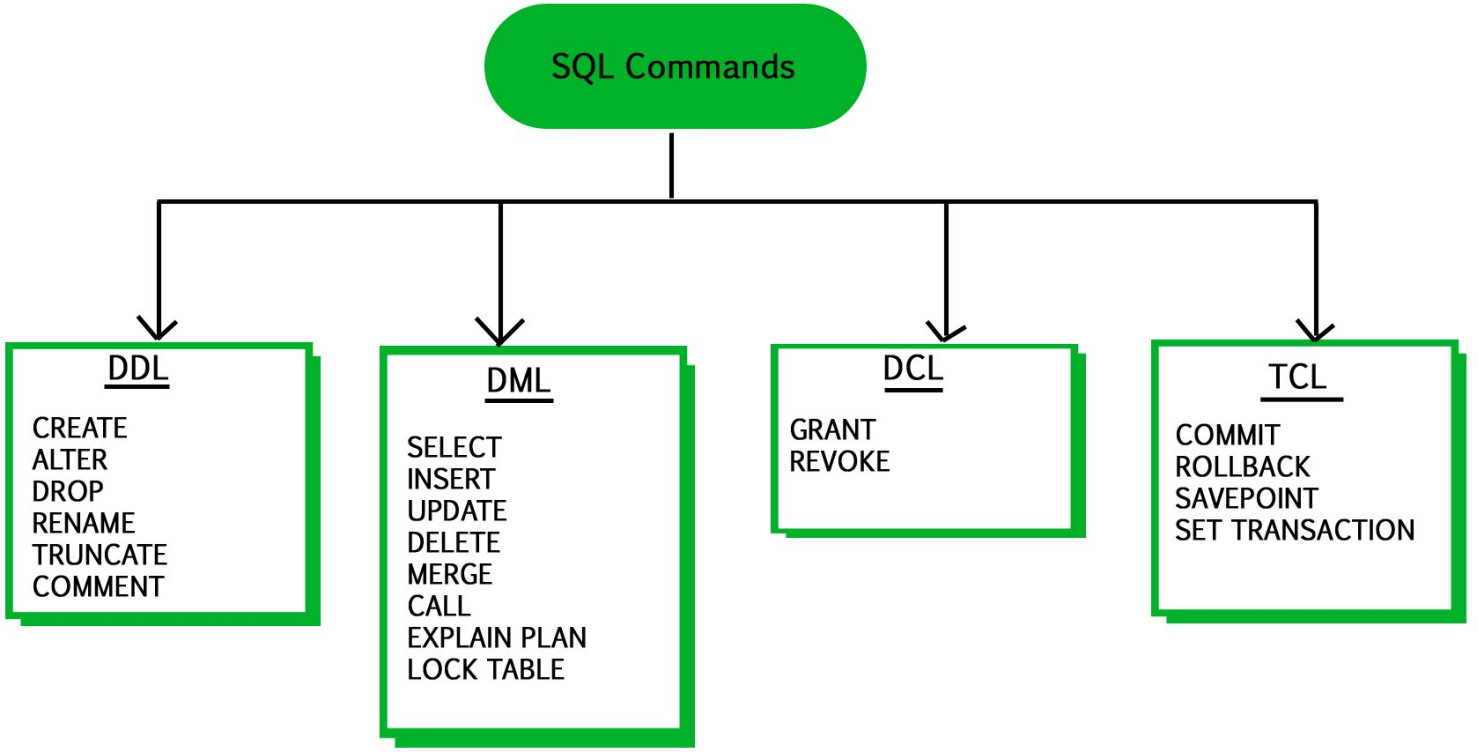
- **Course Outcomes:**

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 trea

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
- MySQL indices are stored in B-trees.

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?

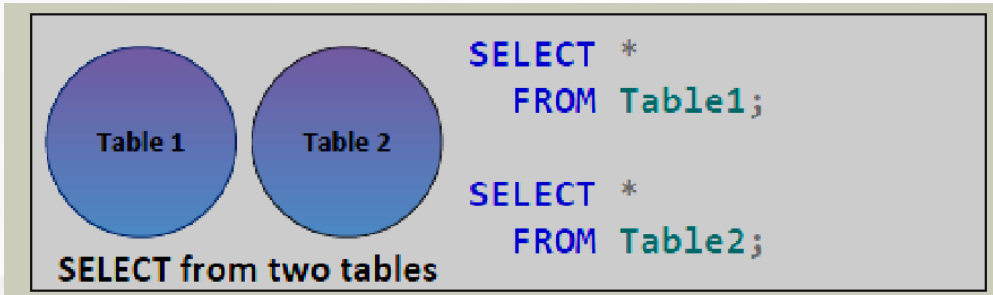
What will full outer return?

Left outer join Output (contd..)

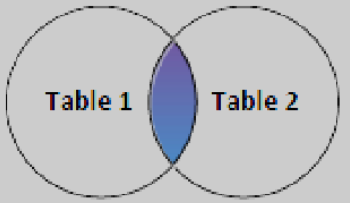
Movie_id	Title	Category	Id	First_name	Last_name	Movie_id
1	ASSASSIN'S CREED:	Animations	1	Adam	Smith	1
2	Real Steel(2012)	Animations	2	Ravi	Kumar	2
3	**Jurassic Park**	**Animation**				

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

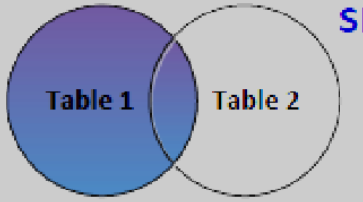
SQL Joins



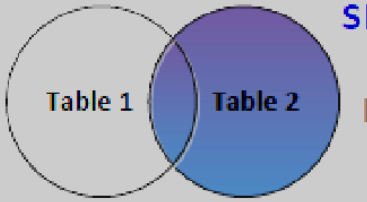

```
SELECT *
  FROM Table1;

SELECT *
  FROM Table2;
```




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

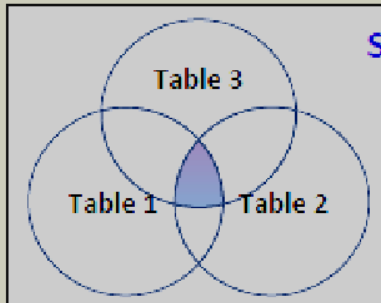



```
SELECT *
  FROM Table1 t1
LEFT OUTER JOIN Table2 t2
   ON t1.fk = t2.id;
```



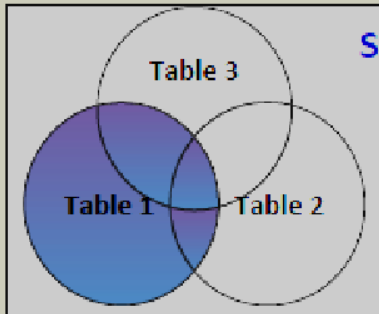

```
SELECT *
  FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
   ON t1.fk = t2.id;
```

SQL Joins



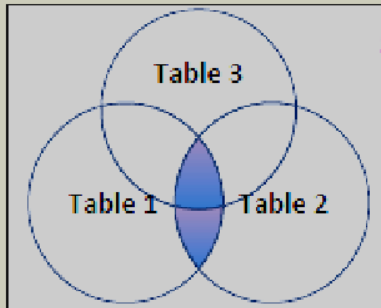
Two INNER JOINs

```
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 whe
  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**

table1

ID	NAME
1	ABHI
2	SAMEER
3	SAMEER
4	JAVED

table2

ID	NAME
3	SAMEER
4	JAVED

- Select name from table1 union select name from table2;

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

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	SAMEER

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

Exercises 01

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.**
  - Suppliers - S (S#, Name, Status, City)
  - Parts - P (P#, Pname, Colour, Weight, City)
  - Projects - J (J#, Jname, City)
  - Shipment - SPJ (S#, P#, J#, Qty)

Solve the following queries

1. Get S# for suppliers who supply project J1.
2. Get P# for parts supplied by a supplier in London.
3. Get the total quantity of part P1 supplied by S1.
4. Get project names for projects supplied by supplier S1.
5. Get colors of parts supplied by S1.
6. Get all part-color/part-city combinations.
7. Get J# for projects supplied by at least one supplier
8. Get colors of parts supplied by S1.
9. Delete all parts whose color is grren.
10. Create one view

Exercises 02

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.**
  - employee (employee name, street, city) ,employee name is primary key
  - works (employee name, company name, salary)
  - company (company name, city) ,company name is primary key
  - manages (employee name, manager name)

Solve the following queries

1. Find the names of all employees who work for First Bank Corporation.
2. Find all employees who do not work for First Bank Coorporation
3. Find the company that has most employees.
4. Find all companies located in every in which small bank corporation is located
5. Find details of employee having salary greater than 10,000.
6. Update salary of all employees who work for First Bank Corporation by 10%.
7. Find employee and their managers.
8. Find the names, street and cities of all employees who work for First Bank Corporation and earn more than 10,000.
9. Find those companies whose employees earn a higher salary,on average, than th average salary at First Bank Corporation
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 B

Exercises 01

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.**
  - Hotel (HotelNo, Name, City) HotelNo is the primary key
  - Room (RoomNo, HotelNo, Type, Price)
  - Booking (HotelNo, GuestNo, DateFrom, DateTo, RoomNo)
  - Guest (GuestNo, GuestName, GuestAddress) GuestNo **is primary key**

Solve the following queries

1. How many hotels are there?
2. List the price and type of all rooms at the Grosvenor Hotel.
3. List the number of rooms in each hotel.
4. Update the price of all rooms by 5%.
5. List full details of all hotels in London.
6. What is the average price of a room?
7. List all guests currently staying at the Grosvenor Hotel.
8. List the number of rooms in each hotel in London.
9. Create one view on above database and query it.

1.

Exercises 02

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.**

  - emp (eno,ename,Zip,hdate)
  - parts(pno,pname,qty_on_hand, price)
  - customer(cno,cname,street,Zip,phone)
  - order(ono,cno,receivedate,shippeddate)
  - odetails(ono,pno,qty)
  - zipcode(Zip,city)

Solve the following queries

1. get pno,pname values of parts that are priced less than $20.00
2. get the ono & cname values of customer whose orders have not yet been shipped
3. get the cname values of customer who have placed order with emp living in ‘Pune’ or ‘mumbai’
4. get the cities in which customer or emp are located
5. get the totalquantity of part 1060 that has been ordered
6. get the total no of customer
7. 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 C

Exercises 01

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.**
  - Project(project_id,proj_name,chief_arch) project_id is primary key
  - Employee(Emp_id,Emp_name) Emp_id is primary key
  - Assigned-To(Project_id,Emp_id)

  Solve the following queries

1. Get the details of employees working on project C353
2. Get employee number of employees working on project C353
3. Obtain details of employees working on Database project
4. Get details of employees working on both C353 and C354
5. Get employee numbers of employees who do not work on project C453
6. Get details of all employee whose name start with S.
7. Get details of all projects whose chief_arch is 'Smith'
8. Delete all project whose proj_name is 'DBMS'
9. Get project_id of Emp_id 101.
10. Create one view

Exercises 02

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.**
  - Employee(emp_no,name,skill,pay-rate) eno primary key
  - Position(posting_no,skill) posting_no primary key
  - Duty_allocation(posting_no,emp_no,day,shift)

  Solve the following queries

1. Get the duty allocation details for emp_no 123461 for the month of April 1986.
2. Find the shift details for Employee ‘xyz’
3. Get employees whose rate of pay is more than or equal to the rate of pay of employee ‘xyz’
4. Get the names and pay rates of employees with emp_no less than 123460 whose rate of pay is more than the rate of pay of at least one employee with emp_no greater than or equal to 123460.
5. Find the names of employees who are assigned to all positions that require a Chef’s skill
6. Get the employee numbers of all employees working on at least two dates.
7. Get a list of names of employees with the skill of Chef who are assigned a duty
8. Get a list of employees not assigned a duty
9. Get a count of different employees on each shift
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

Exercises 01

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.**
  - Account(Acc_no, branch_name,balance)
  - branch(branch_name,branch_city,assets)
  - customer(cust_name,cust_street,cust_city)
  - Depositor(cust_name,acc_no)
  - Loan(loan_no,branch_name,amount)
  - Borrower(cust_name,loan_no)

Solve the following queries

1. Find all loan numbers for loans made at Akurdi Branch with loan amount > 12000.
2. Find all customers who have a loan from bank. Find their names,loan_no and loan amount.
3. List all customers in alphabetical order who have loan from Akurdi branch.
4. Find all customers who have an account or loan or both at bank.
5. Find all customers who have both account and loan at bank.
6. Find all customer who have account but no loan at the bank.
7. Find average account balance at Akurdi branch.
8. Find the average account balance at each branch
9. Find no. of depositors at each branch.
10. Find the branches where average account balance > 12000.

- **Create a database which consist of the following tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.**
  - Sailors (Sid , same , rating age)
  - Boats(bid , bname, color)
  - Reserves (sid, bid, day(date))

Solve the following queries

1. find the sid of sailors who have reserved the a red boat
2. find the name of sailors who have reserved the red boat
3. find the color of boats reserved by ‘john’
4. find the sid of sailors who have traveled in the month ‘april’
5. find all sailors with less than 7 rating.
6. Create one view

MIT-WPU
।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

Thank You!