Assignment No.4

Title of Assignment: Design at least 10 SQL/NoSQL queries for suitable database application using SQL DML statements: all types of Join, SubQuery and View.

Objective:to understand all types of Joins in SQL, Sub queries, operations on views such as insert, update, delete

Relevant Theory

❖ Joins:

"JOIN" is an SQL keyword used to query data from two or more related tables.
The join operation allows the combining of two relations by merging pairs of tuples, one from each relation, into a single tuple.

In a database such as MySQL, data is divided into a series of tableswhich are then connected together in SELECT commands to generate the output required.

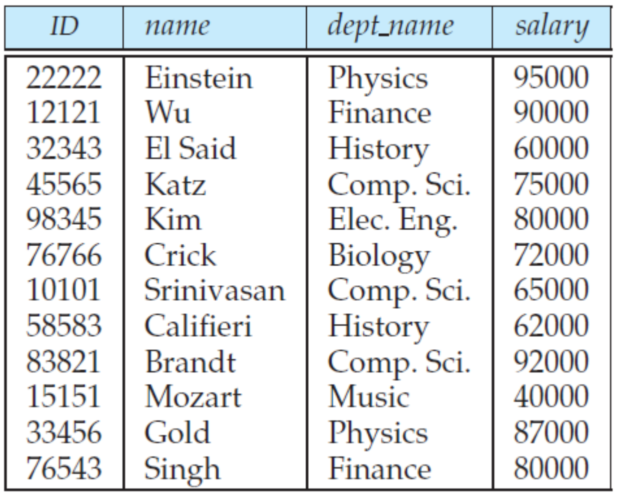
There are a number of different ways to join relations such as

- The Natural Join or Regular Join
- Inner Join
- Left outer join
- Right Outer join
- Full outer join

We will see how these join works in MySQL

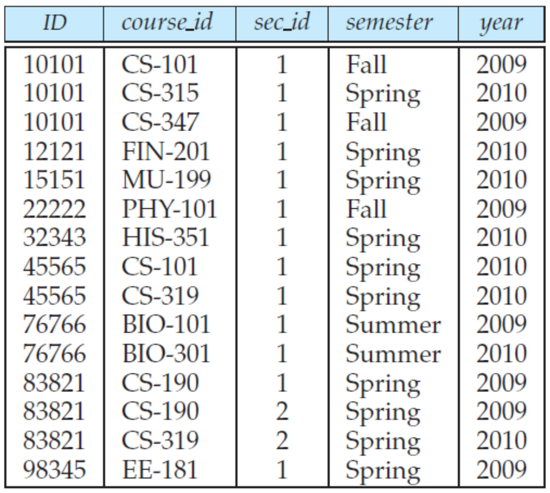
The Natural Join or Regular Join

The natural join operation operates on two relations and produces a relationas the result.
Natural join considersonly those pairs of tuples with the same value on those attributes that appear inthe schemas of both relations.
For example consider following two relations



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

Instructor Relegation



ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Teaches Relation

Computing **instructor natural join teaches** considers onlythose pairs of tuples where both the tuple from instructor and the tuple fromteaches have the same value on the common attribute, ID.
The result relation, shown in below, has only 13 tuples, the ones thatgive information about an instructor and a course that that instructor actuallyteaches. Notice that we do not repeat those attributes that appear in the schemasof both relations; rather they appear only once.
Notice also the order in which theattributes are listed: first the attributes common to the schemas of both relations,second those attributes unique to the schema of the first relation, and finally, thoseattributes unique to the schema of the second relation.

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

The natural join of the instructor relation with the teaches relation.

Consider the query "For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught", which we wrote as:

select name, course id
from instructor, teaches
where instructor.ID= teaches.ID;

This query can be written more concisely using the natural-join operation inSQL as:

select name, course id
from instructor natural join teaches;

To understand how to compute **natural join in MySQL**consider below sample data such as :
Mr. Brown, Person number 1, has a phone number 01225 708225
Miss Smith, Person number 2, has a phone number 01225 899360
Mr. Pullen, Person number 3, has a phone number 01380 724040
and also:
Person number 1 is selling property number 1 - Old House Farm
Person number 3 is selling property number 2 - The Willows

Person number 3 is (also) selling property number 3 - Tall Trees
Person number 3 is (also) selling property number 4 - The Melksham Florist
Person number 4 is selling property number 5 - Dun Roamin.

Create two tables such as **demo_people** and **demo_property** and insert above data in to it. Then after select all data from both the tables using below expressions

```
mysql> select * from demo_people;

+------------+-------------+------+

| name       | phone       | pid  |

+------------+-------------+------+

| Mr Brown   | 01225 708225 |    1 |

| Miss Smith | 01225 899360 |    2 |

| MrPullen  | 01380 724040 |    3 |

+------------+-------------+------+

3 rows in set (0.00 sec)


mysql> select * from demo_property;

+------+------+----------------------+

| pid  |spid | selling              |

+------+------+----------------------+

|    1 |    1 | Old House Farm       |

|    3 |    2 | The Willows          |

|    3 |    3 | Tall Trees           |

|    3 |    4 | The Melksham Florist |

|    4 |    5 | Dun Roamin           |

+------+------+----------------------+

5 rows in set (0.00 sec)

mysql>
```

Natural join or Regular join on above both tables will be done by following query expression

```
mysql> select name, phone, selling

from demo_peoplejoindemo_property

on demo_people.pid = demo_property.pid;

+-----------+-------------+---------------------+

| name      | phone       | selling             |

+-----------+-------------+---------------------+

| MrBrown  | 01225 708225 | Old House Farm       |

| Mr Pullen | 01380 724040 | The Willows         |

| Mr Pullen | 01380 724040 | Tall Trees          |

| Mr Pullen | 01380 724040 | The Melksham Florist |

+-----------+-------------+---------------------+

4 rows in set (0.01 sec)

mysql>
```

It will give all records that match the value of appropriate attributesin the two tables, and records in both incoming tables that do not match are discarded.

Such types of join operation that do not preserve non-matched tuples are called as **inner join operations**.

Outer Join
An outer join does not require each record in the two joined tables to have a matching record or attribute. The joined table retains each record—even if no other matching record exists.
There are in fact three forms of outer join:

- The **left outer join** preserves tuples only in the relation named before (to theleft of) the left outer join operation.
- The **right outer join** preserves tuples only in the relation named after (to theright of) the right outer join operation.
- The **full outer join** preserves tuples in both relations. (In this case left and right refer to the two sides of the JOIN keyword.)

No implicit join-notation for outer joins exists in standard SQL.

Left Outer Join
The result of a left outer join for tables A and B always contains all records of the "left" table (table A), even if the join-condition does not find any matching record in the "right" table (tableB).
This means that if the ON clause matches zero records in B for a given record in A, the join will still return a row in the result for that record—but with NULL in each column from B.

A left outer join returns all the values from an inner join plus all values in the left table that do not match to the right table, including rows with NULL (empty) values in the link field.
To understand how to compute **left outer join in MySQL** again consider **demo_people** and **demo_property**tables
After computing LEFT JOIN, result contains all records that match in the same way and IN ADDITION it will shows an extra record for each unmatched record in the left table of the join - thus ensuring that every PERSON is included in result generated by left outer join:

```
mysql> select name, phone, selling

fromdemo_peopleleft joindemo_property

on demo_people.pid = demo_property.pid;

+------------+-------------+----------------------+

| name       | phone       | selling              |

+------------+-------------+----------------------+

| Mr Brown   | 01225 708225 | Old House Farm       |

| Miss Smith | 01225 899360 | NULL                 |

| MrPullen  | 01380 724040 | The Willows          |

| MrPullen  | 01380 724040 | Tall Trees           |

| MrPullen  | 01380 724040 | The Melksham Florist |

+------------+-------------+----------------------+

5 rows in set (0.00 sec)


mysql>
```

Right Outer Join
The result of a right outer join for tables A and B always contains all records of the "right" table (table B), even if the join-condition does not find any matching record in the "Left" table (table A).
This means that if the ON clause matches zero records in A for a given record in B, the join will still return a row in the result for that record—but with NULL in each column from A.
A right outer join returns all the values from an inner join plus all values in the right table that do not match to the left table, including rows with NULL (empty) values in the link field.

To understand how to compute **right outer join in MySQL** again consider **demo_people** and **demo_property**tables

After computing RIGHT JOIN, result contains all the records that match and IN ADDITION it will show extra record for each unmatched record in the right table of the join - that means that each property gets a mention even if we don't have seller details:

```
mysql> select name, phone, selling

fromdemo_people right join demo_property

on demo_people.pid = demo_property.pid;

+-----------+-------------+---------------------+

| name      | phone       | selling             |

+-----------+-------------+---------------------+

| MrBrown  | 01225 708225 | Old House Farm       |

| Mr Pullen | 01380 724040 | The Willows         |

| Mr Pullen | 01380 724040 | Tall Trees          |

| Mr Pullen | 01380 724040 | The Melksham Florist |

| NULL      | NULL        | Dun Roamin          |

+-----------+-------------+---------------------+

5 rows in set (0.00 sec)

mysql>
```

Full Outer Join
The full outer join is a combination of the left and right outer-join types. After the operation computes the result of the inner join, it extends with nulls those tuples from the left-hand-side relation that did not match with any from the right-hand side relation, and adds them to the result.
Similarly, it extends with nulls those tuples from the right-hand-side relation that did not match with any tuples from the left-hand-side relation and adds them to the result.

A standard SQL FULL OUTER join is like a LEFT or RIGHT join, except that it includes all rows from both tables, matching them where possible and filling in with NULLs where there is no match.Here are two tables, apples and oranges:

apples	
Variety	**Price**
Fuji	5.00
Gala	6.00

Join them on price. Here is the left join:

```
select * from apples as a
left outer join oranges as o on a.price = o.price
```

variety	price	variety	price
Fuji	5	Navel	5
Gala	6	NULL	NULL

oranges	
Variety	**Price**
Valencia	4.00
Navel	5.00

And the right joins:

```
select * from apples as a
right outer join oranges as o on a.price = o.price
```

variety	price	variety	price
NULL	NULL	Valencia	4
Fuji	5	Navel	5

The FULL OUTER JOIN of these two tables, on price, should give the following result:

variety	price	variety	price
Fuji	5	Navel	5
Gala	6	NULL	NULL
NULL	NULL	Valencia	4

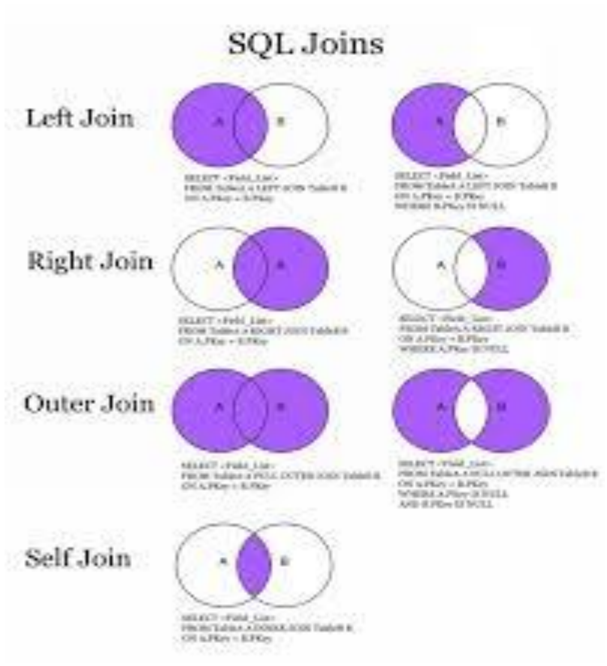
Here is a script to create and populate the example tables, so you can follow along:

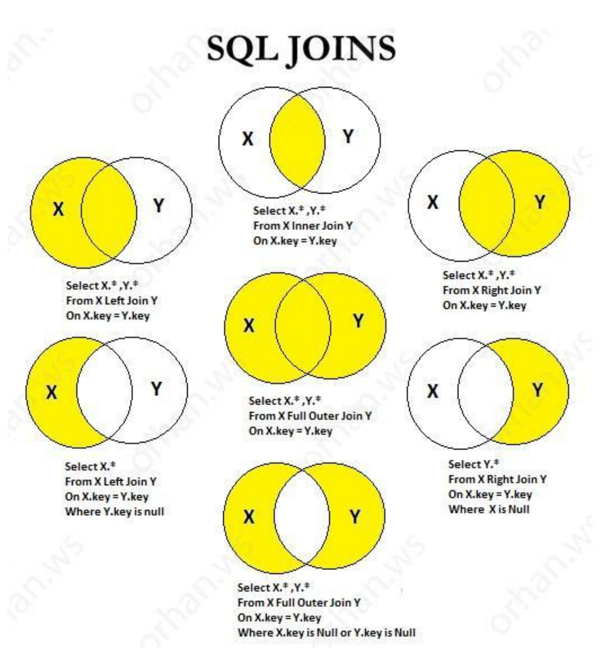
```
Mysql>create table apples (variety char(10) not null primary key, price int
not null);
Mysql>create table oranges (variety char(10) not null primary key, price int
not null);
Mysql>insert into apples(variety, price) values('Fuji',5),('Gala',6);
Mysql>insert into oranges(variety, price) values('Valencia',4),('Navel',5);
```

One method to simulate a full outer join is to take the union of two outer joins, for example,

```
Mysql>select * from apples as a
left outer join oranges as o on a.price = o.price
union
select * from apples as a
right outer join oranges as o on a.price = o.price
```

This gives the desired results in this case.

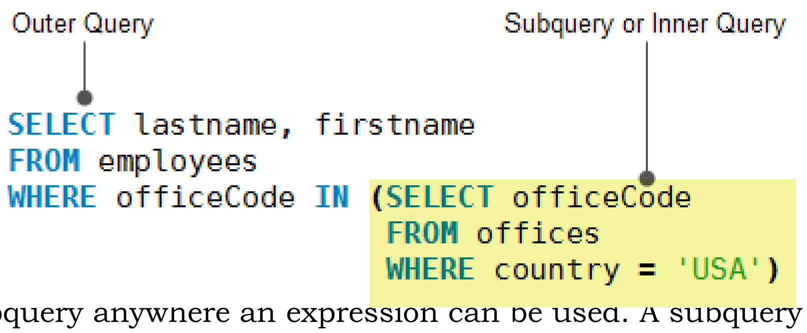



SQL JOINS
X
Y
Select X.*,Y.* From X Inner Join Y On X.key = Y.key
Select X.*,Y.* From X Left Join Y On X.key = Y.key
Select X.*,Y.* From X Right Join Y On X.key = Y.key
Select X.*,Y.* From X Full Outer Join Y On X.key = Y.key
Select X.* From X Left Join Y On X.key = Y.key Where Y.key is null
Select Y.* From X Right Join Y On X.key = Y.key Where X is Null
Select X.*,Y.* From X Full Outer Join Y On X.key = Y.key Where X.key is Null or Y.key is Null

❖ Sub Queries

A MySQL subquery is a query that is nested inside another query such as SELECT, INSERT, UPDATE or DELETE. A MySQL subquery is also can be nested inside another subquery. A MySQL subquery is also called an inner query, while the query that contains the subquery is called an outer query.
For example following subquery that returns employees who locate in the offices in the USA.

- The subquery returns all *offices codes* of the offices that locate in the USA.
- The outer query selects the last name and first name of employees whose office code is in the result set returned from the subquery.



```
SELECT lastname, firstname
FROM employees
WHERE officeCode IN (SELECT officeCode
                     FROM offices
                     WHERE country = 'USA')
```

You can use a subquery anywhere an expression can be used. A subquery also must be enclosed in parentheses.

MySQL subquery within a WHERE clause

1. MySQL subquery with comparison operators

If a subquery returns a single value, you can use comparison operators to compare it with the expression in the WHERE clause.
For example, the following query returns the customer who has the maximum payment.

```
Mysql> SELECT customerNumber, checkNumber, amount
FROM payments
WHERE amount = (
   SELECT MAX(amount)
      FROM payments
)
```

Other comparison operators can be used such as greater than (>), less than(<), etc.
For example, you can find customer whose payment is greater than the average payment.
A subquery is used to calculate the average payment by using the AVG aggregate function.
The outer query selects payments that are greater than the average payment returned from the subquery.

```
Mysql> SELECT customerNumber,
     checkNumber,
     amount
FROM payments
WHERE amount > (
   SELECT AVG(amount)
   FROM payments
)
```

2. MySQL subquery with IN and NOT IN operators

If a subquery returns more than one value, you can use other operators such as IN or NOT IN operator in the WHERE clause.
For example, you can use a subquery with NOT IN operator to find customer who has not ordered any product as follows:

```
Mysql> SELECT customername
FROM customers
WHERE customerNumber NOT IN (
   SELECT DISTINCT customernumber
   FROM orders
)
```

3. MySQL subquery with EXISTS and NOT EXISTS

When a subquery is used with EXISTS or NOT EXISTS operator, a subquery returns a Boolean value of TRUE or FALSE. The subquery acts as an existence check.
In the following example, we select a list of customers who have at least one order with total sales greater than 10K.
First, we build a query that checks if there is at least one order with total sales greater than 10K:

```
Mysql > SELECT priceEach * quantityOrdered
FROM orderdetails
WHERE priceEach * quantityOrdered> 10000
GROUP BY orderNumber
```

The query returns some records so that when we use it as a subquery, it will return TRUE; therefore the whole query will return all customers:

```
Mysql > SELECT customerName
FROM customers
WHERE EXISTS (
   SELECT priceEach * quantityOrdered
   FROM orderdetails
   WHERE priceEach * quantityOrdered> 10000
   GROUP BY orderNumber
)
```

If you replace the EXISTS by NOT EXIST in the query, it will not return any record at all.

4. MySQL subquery in FROM clause

When you use a subquery in the FROM clause, the result set returned from a subquery is used as a table.
This table is referred to as a *derived table* or *materialized subquery*.
The following subquery finds the maximum, minimum and average number of items in sale orders:

```
SELECT max(items),
     min(items),
     floor(avg(items))
FROM
(SELECT orderNumber,
   count(orderNumber) AS items
FROM orderdetails
GROUP BY orderNumber) AS lineitems
```

❖ View:

MySQL supports database views or views since version 5.X. In MySQL, almost features of views conform to the SQL: 2003 standard. MySQL process queries to the views in two ways:

- MySQL creates a temporary table based on the view definition statement and then executes the incoming query on this temporary table.
- First, MySQL combines the incoming query with the query defined the view into one query. Then, MySQL executes the combined query.

MySQL supports version system for views. Each time when the view is altered or replaced, a copy of the existing view is back up in arc (archive) folder which resides in a specific database folder.
The name of back up file is view_name.frm-00001. If you then change the view again, MySQL will create a new backup file named view_name.frm-00002.
MySQL also allows you to create a view of views. In the SELECT statement of view definition, you can refer to another view.
If the cache is enabled, the query against a view is stored in the cache. As the result, it increases the performance of the query by pulling data from the cache instead of querying data from the underlying tables.

Creating Updateable Views
In MySQL, views are not only read-only but also updateable. However in order to create an updateable view, the SELECT statement that defines the view has to follow several following rules:

- The SELECT statement must only refer to one database table.
- The SELECT statement must not use GROUP BY or HAVING clause.
- The SELECT statement must not use DISTINCT in the column list of the SELECT clause.
- The SELECT statement must not refer to read-only views.
- The SELECT statement must not contain any expression (aggregates, functions, computed columns…)

When you create updateable views, make sure that you follow the rules above.

Example of creating updateable view
Create a view named officeInfo against the offices table. The view refers to three columns of the offices table: officeCode, phone and city.

```
Mysql> CREATE VIEW officeInfo
 AS
   SELECT officeCode, phone, city
   FROM offices
```

We can query data from the officeInfo view using the SELECT statement.

```
Mysql> SELECT * FROM officeInfo
```

Then, we can change the phone number of the office with officeCode 4 through theofficeInfo view by using the UPDATE statement.

```
Mysql> UPDATE officeInfo
SET phone = '+33 14 723 5555'
WHERE officeCode = 4
```

Finally, to see the change, we can select the data from the officeInfo view by executing following query:

```
My sql> SELECT * FROM officeInfo
WHERE officeCode = 4
```

Managing Views in MySQL: Modifying views
Once a view is defined, can be modified by using the ALTER VIEW statement.
The syntax of the ALTER VIEW statement is similar to the CREATE VIEW statement except the CREATE keyword is replaced by the ALTER keyword.

```
ALTER
 [ALGORITHM =  {MERGE | TEMPTABLE | UNDEFINED}]
  VIEW [database_name].  [view_name]
   AS
 [SELECT statement]
```

The following query modifies the organization view by adding an addition email field.

```
Mysql> ALTER VIEW organization
  AS
  SELECT CONCAT(E.lastname,E.firstname) AS Employee,
         E.email AS  employeeEmail,
         CONCAT(M.lastname,M.firstname) AS Manager
  FROM employees AS E
  INNER JOIN employees AS M
    ON M.employeeNumber = E.ReportsTo
  ORDER BY Manager
```

To verify the change, you can query data from the organization view:

```
ELECT * FROM Organization
```

MySQL drop views
Once a view created, you can remove it by using the DROP VIEW statement. The following illustrates the syntax of the DROP VIEW statement:

```
DROP VIEW [IF EXISTS] [database_name].[view_name]
```

The IF EXISTS is the optional element of the statement, which allows you to check whether the view exists or not. It helps you avoid an error of removing a non-existent view.

For example, if you want to remove the organization view, you can use the DROP VIEW statement as follows:

```
DROP VIEW IF EXISTS organization
```

Each time you modify or remove a view, MySQL makes a back up of the view definition file to the/database_name/arc/ folder. In case you modify or remove a view by accident, you can get a back up from there.

Conclusion:

- We have explained the MySQL LEFT JOIN, RIGHT JOIN clause and shown you how to apply it to query data from multiple database tables.
- We have shown you how to use MySQL subquery to write more complex queries.
- We have shown you how to create an updateable view and how to update data in the underlying table through the view.
- We have learned how to manage views in MySQL including displaying, modifying and removing views.