Q: How to remove duplicate rows from table in Oracle?

Ans:

DELETE FROM EMP\_TEMP\_DUPS

WHERE rowid not in

(SELECT MIN (rowid)

FROM EMP\_TEMP\_DUPS

GROUP BY id,firstname,lastname,age,salary);

Q: How to identify whether the linked list has loop?

private static boolean isCircularLinkedList(LinkedList<Integer> linkedList) {

boolean isCircular = Boolean.FALSE;

Set<Integer> circular = new HashSet<>();

for (Integer integer : linkedList) {

if (circular.contains(integer)) {

System.out.println("Circular Linked List: " + integer);

isCircular = Boolean.TRUE;

break;

} else {

circular.add(integer);

}

}

return isCircular;

}

Q: when should I use JTA transaction manager and when JPA, and what benefit and disadvantages does they have?

If you want to delegate managed transactions to your Application Server and handle complex transactions across multiple resources you need to use the JtaTransactionManager,

The Spring Framework gives you the choice of when to scale your application to a fully loaded application server. Gone are the days when the only alternative to using EJB CMT or JTA was to write code with local transactions such as those on JDBC connections, and face a hefty rework if you need that code to run within global, container-managed transactions. With the Spring Framework, only some of the bean definitions in your configuration file, rather than your code, need to change

**Spring Framework transaction management**

### **Global transactions**

Global transactions enable you to work with multiple transactional resources, typically relational databases and message queues. The application server manages global transactions through the JTA, which is a cumbersome API to use (partly due to its exception model). Furthermore, a JTA UserTransaction normally needs to be sourced from JNDI, meaning that you *also* need to use JNDI in order to use JTA. Obviously the use of global transactions would limit any potential reuse of application code, as JTA is normally only available in an application server environment.

### Local transactions

Local transactions are resource-specific, such as a transaction associated with a JDBC connection. Local transactions may be easier to use, but have significant disadvantages: they cannot work across multiple transactional resources. For example, code that manages transactions using a JDBC connection cannot run within a global JTA transaction. Because the application server is not involved in transaction management, it cannot help ensure correctness across multiple resources. (It is worth noting that most applications use a single transaction resource.) Another downside is that local transactions are invasive to the programming model.

The TransactionDefinition interface specifies:

* *Isolation*: The degree to which this transaction is isolated from the work of other transactions. For example, can this transaction see uncommitted writes from other transactions?
* *Propagation*: Typically, all code executed within a transaction scope will run in that transaction. However, you have the option of specifying the behavior in the event that a transactional method is executed when a transaction context already exists. For example, code can continue running in the existing transaction (the common case); or the existing transaction can be suspended and a new transaction created. *Spring offers all of the transaction propagation options familiar from EJB CMT*. To read about the semantics of transaction propagation in Spring, see [Section 11.5.7, “Transaction propagation”](https://docs.spring.io/spring-framework/docs/4.0.x/spring-framework-reference/html/transaction.html#tx-propagation).
* *Timeout*: How long this transaction runs before timing out and being rolled back automatically by the underlying transaction infrastructure.
* *Read-only status*: A read-only transaction can be used when your code reads but does not modify data. Read-only transactions can be a useful optimization in some cases, such as when you are using Hibernate.

Q: How can you display top two employee salaries group by department in oracle?

select

d.Salary

,d.Department

from

(

select

r.Salary

,r.Department

,row\_number() over(

partition by r.Department

order by r.Salary desc) as RowNumber

from HumanResources as r

) as d

where d.RowNumber < 3

# [**Finding duplicate values in a SQL table**](https://stackoverflow.com/questions/2594829/finding-duplicate-values-in-a-sql-table)

ID NAME EMAIL

1 John asd@asd.com

2 Sam asd@asd.com

3 Tom asd@asd.com

4 Bob bob@asd.com

5 Tom asd@asd.com

Ans:

SELECT

name, email, COUNT(\*)

FROM

users

GROUP BY

name, email

HAVING

COUNT(\*) > 1

Q: How to find top three highest salary in emp table in oracle?

SELECT \*FROM

(

SELECT \*FROM emp

ORDER BY Salary desc

)

WHERE rownum <= 3

ORDER BY Salary ;

SELECT \* FROM

(

SELECT EMPLOYEE, LAST\_NAME, SALARY,

RANK() OVER (ORDER BY SALARY DESC) EMPRANK

FROM emp

)

WHERE emprank <= 3;

# Q: [Highest Salary in each department](https://stackoverflow.com/questions/8477040/highest-salary-in-each-department)

Table Name EmpDetails

DeptID EmpName Salary

Engg Sam 1000

Engg Smith 2000

HR Denis 1500

HR Danny 3000

IT David 2000

IT John 3000

Ans:

SELECT DeptID, MAX(Salary) FROM EmpDetails GROUP BY Dept

# Q:[Differences between session vs session factory - Hibernate?](https://stackoverflow.com/questions/22470968/differences-between-session-vs-session-factory-hibernate)

SessionFactory is Hibernate’s concept of a single datastore and is threadsafe so that many threads can access it concurrently and request for sessions and immutable cache of compiled mappings for a single database.

No, Session is not Thread Safe. A Session is a light weight and a non-threadsafe object (No, you cannot share it between threads) that represents a single unit-of-work with the database. Sessions are opened by a SessionFactory and then are closed when all work is complete. Session is the primary interface for the persistence service. A session obtains a database connection lazily (i.e. only when required)

Q: Is Hibernate's session thread safe?

t is not intended that implementors be threadsafe. Instead each thread/transaction should obtain its own instance from a SessionFactory.

Even with this in mind, your behaviour might still not be what you expect, because transactions come into play. You will have to set a proper [transaction isolation level](http://en.wikipedia.org/wiki/Isolation_(database_systems)). See the [configuration guide](http://docs.jboss.org/hibernate/core/3.3/reference/en/html/session-configuration.html), hibernate.connection.isolation property.

It depends on how you are creating a session.

Session can be created in two ways in hibernate.

1. getCurrentSession()

Yes. It offers thread safety as it'll ensure that it'll create a session for each thread if session not exist. transaction and automatic session closing is attached to this.

1. openSession()

It's not thread safe. developer manually needs to manage transactions and session flush and close operations.