**Difference between Dirty Read, Non Repeatable Read and Phantom Read in Database**

**Dirty Read:-**

Dirty read occurs when one transaction is changing the record, and the other transaction can read this record before the first transaction has been committed or rolled back. This is known as a dirty read scenario because there is always the possibility that the first transaction may rollback the change, resulting in the second transaction having read an invalid data.

**Dirty Read Example:-**

Transaction A begins.  
UPDATE EMPLOYEE SET SALARY = 10000 WHERE EMP\_ID= ‘123’;

Transaction B begins.  
SELECT \* FROM EMPLOYEE;  
(Transaction B sees data which is updated by transaction A. But, those updates have not yet been committed.)

**Non-Repeatable Read:-**

Non Repeatable Reads happen when in a same transaction same query yields to a different result. This occurs when one transaction repeatedly retrieves the data, while a difference transactions alters the underlying data. This causes the different or non-repeatable results to be read by the first transaction.

**Non-Repeatable Example:-**

Transaction A begins.  
SELECT \* FROM EMPLOYEE WHERE EMP\_ID= ‘123’;

Transaction B begins.  
UPDATE EMPLOYEE SET SALARY = 20000 WHERE EMP\_ID= ‘123’;  
(Transaction B updates rows viewed by the transaction A before transaction A commits.) If Transaction A issues the same SELECT statement, the results will be different.

**Phantom Read:-**

Phantom read occurs where in a transaction execute same query more than once, and the second transaction result set includes rows that were not visible in the first result set. This is caused by another transaction inserting new rows between the execution of the two queries. This is similar to a non-repeatable read, except that the number of rows is changed either by insertion or by deletion.

**Phantom Read Example:-**

Transaction A begins.  
SELECT \* FROM EMPLOYEE WHERE SALARY > 10000 ;

Transaction B begins.  
INSERT INTO EMPLOYEE (EMP\_ID, FIRST\_NAME, DEPT\_ID, SALARY) VALUES (‘111′, ‘Jamie’, 10, 35000);  
Transaction B inserts a row that would satisfy the query in Transaction A if it were issued again.

# Difference between Optimistic Locking & Pessimistic Locking in Database

Locking is a method which is used to protect records that will be accessed by multiple users so that concurrency errors do not occur (i.e when multiple users change records near simultaneously resulting in inconsistencies). Locks can be released by issuing any of the following two **SQL**commands.

i.e  
1) ROLLBACK – This will ignore the changes made by the current transaction  
2) COMMIT – This will save the changes made by the current transaction

**Optimistic Locking :-**

In this locking, multiple users can open the same record for updating, thus increase maximum concurrency. Record is only locked when updating the record. This is the most preferred way of locking practically.

Please note that, Optimistic lock is an implicit lock.

In **Hibernate**, we can achieve optimistic control with versioning. Version checking uses version numbers, or timestamps, to detect conflicting updates and to prevent lost updates. Hibernate provides three possible approaches to write application code that uses optimistic concurrency.

**Pessimistic Locking :-**

In this locking, when the user wants to update data it locks the record and till then no one can update data. Other user’s can only view the data when there is pessimistic locking. As we can see, now a days in browser based application it is very common and having pessimistic locking is not a practical solution at all. This is often done by relying on the database itself and most relational databases use this method.

Please note that, Pessimistic lock is an explicit lock that set by the client.

In **Hibernate**, we can achieve pessimistic control with the mechanism of Database. To use pessimistic locking you need either a direct connection to the database or an externally available transaction ID that can be used independently of the connection.

**For Example :-**

SQL Server 2000 locks single rows, while others may lock the entire table containing the record(s) to be changed. A drawback is that this type of locking requires that you remain connected to the database for the whole time.