Transactions:

1) Conflicting:

Transactions t1:

i) Reading table product:

Select * from drivers where driver_id=1;

ii) Writing table product

Update drivers Set rating=1 where driver_id=1;

iii) writing table vehicle

insert into vehicles(car_id,model) VALUES(222,”suv”)

Transactions t2;

Select * from drivers where driver_id=1;

ii) Writing table product

Update drivers Set rating=1 where driver_id=1;

iii) writing table vehicle

insert into vehicles(car_id,model) VALUES(222,”suv”)

CONFLICT SERIALIZABLE

T1	T2
Select * from drivers where driver_id=1;	
Update drivers Set rating=1 where driver_id=1;	
	Select * from drivers where driver_id=1;
	Update drivers Set rating=1 where driver_id=1;

insert into vehicles(car_id,model) VALUES(222,”suv”)	
Commit t1	
	insert into vehicles(car_id,model) VALUES(222,”suv”)
	Commit t2

NON CONFLICT SERIALIZABLE

T1	T2
Select * from drivers where driver_id=1;	
Update drivers Set rating=1 where driver_id=1;	
	Select * from drivers where driver_id=1;
	Update drivers Set rating=1 where driver_id=1;
	insert into vehicles(car_id,model) VALUES(222,”suv”)
	Commit t2
insert into vehicles(car_id,model) VALUES(222,”suv”)	

Commit t1	

Non conflicting transaction:

1) `"SELECT b.*, p.name AS passenger_name FROM bookings b INNER JOIN passengers p ON b.passenger_id = p.passenger_id;`
2) Insert into users(user_id,password,category) values(1,”dasdsa”,”passenger”)
3) select * from payments
   INNER join trip
   on Trip_Id=Trip_no Where Payment_Type in ("cash")
4) Update drivers Set rating=1 where driver_id=1;

In a database management system (DBMS), even non-conflicting transactions can result in non-serializable outcomes if they do not adhere to a serializable schedule. To address this issue, concurrency control techniques such as locking, timestamp ordering, and optimistic concurrency control can be employed.

Locking:
Locking is a technique where transactions acquire locks on the data they need to access, preventing other transactions from accessing the same data until the lock is released. By utilizing locking, transactions can be serialized, ensuring that non-conflicting transactions do not result in non-serializable outcomes.

Timestamp ordering:
Timestamp ordering is a technique where each transaction is assigned a unique timestamp. Transactions are then ordered based on their timestamps, and a transaction can only access data that was last updated by a transaction with an earlier timestamp. By employing timestamp ordering, transactions can be sequenced to ensure that non-conflicting transactions do not lead to non-serializable outcomes.

Optimistic concurrency control:
Optimistic concurrency control is a technique where transactions are allowed to execute concurrently without acquiring locks. Each transaction is assigned a version number, and when a transaction attempts to commit, it checks if any other transaction has modified the same data. If no other transaction has modified the data, the transaction is permitted to commit. However, if another transaction has modified the data, the transaction is rolled back and must retry. By utilizing optimistic concurrency control, non-conflicting transactions can be executed concurrently without causing non-serializable outcomes.

In conclusion, to resolve the issue of non-conflicting transactions resulting in non-serializable outcomes in a DBMS, concurrency control techniques such as locking, timestamp ordering, or optimistic concurrency control can be employed. These techniques ensure that transactions are executed in a serializable schedule, preventing non-serializable outcomes.