--t-SQL queries to listout all the objects in sample database..

--1. SYSOBJECTS - TELLS DIFFERENT TYPES OF OBJECTS IN DATABASE

--2. SYS.TABLES

--3. INFORMATIONS\_SCHEMA.TABLES

SELECT \* FROM SYSOBJECTS WHERE xtype='P' --LINK: msdn.microsoft.com/en-us/library/ms177596.aspx (for all values LIKE U)

SELECT DISTINCT xtype FROM SYSOBJECTS -- RETURNS ALL TYPES OF SYSTEM OBJECTS IN THE SAMPLE DATABASE

SELECT \* FROM SYS.TABLES --SHOWS TABLE

SELECT \* FROM SYS.views -- SHOWS ALL VIEWS

SELECT \* FROM SYS.procedures --SHOWS PROCEDURES

SELECT \* FROM INFORMATION\_SCHEMA.TABLES --SHOWS TABLES AND VIEWS AT A TIME

SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_TYPE='BASE TABLE' --ONLY SHOWS THE TABLES

SELECT \* FROM INFORMATION\_SCHEMA.VIEWS --SHOWS ONLY VIEW NOT A TABLE

SELECT \* FROM INFORMATION\_SCHEMA.ROUTINES --RETURNS STORED PROCEDURES AND FUNCTIONS

SELECT \* FROM INFORMATION\_SCHEMA.DOMAINS --RETURNS DOMAIN NAMES(ATTRIBUTES ONLY

SELECT \* FROM INFORMATION\_SCHEMA.SEQUENCES

--RE-RUNNABLE SQL SERVER SCRIPTS------------------------------------->

--IT SHOWS THAT A QUERY WILL BE USED A MULTIPLE TIPE BUT NO ERROR WILL BE GIVEN.

--CREATING A GENERAL TABLE

CREATE TABLE RERUNABLITY

(

ID INT,

ROLL INT,

SEC NVARCHAR(50),

)

--CREATING A TABLE USING RERUNABLE SCRIPT. IF WE RUN THE CODE ABOVE TWICE WE GONNA GET AN ERROR

--Msg 2714, Level 16, State 6, Line 1

--There is already an object named 'RERUNABLITY' in the database.

IF NOT EXISTS(SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'REUSABILITY\_2')

BEGIN

CREATE TABLE REUSABILITY\_2

(

ID INT IDENTITY PRIMARY KEY,

ROLL INT,

SEC NVARCHAR(50),

)

PRINT 'TABLE HAS SUCCESSFULLY BEEN CREATED'

END

ELSE

BEGIN

PRINT 'ALREADY EXISTS'

END

--NEVER GIVES AN ERROR. SHOWS already exists.

--alternating process

if OBJECT\_ID('REUSABILITY\_2') is null

begin

CREATE TABLE REUSABILITY\_2

(

ID INT IDENTITY PRIMARY KEY,

ROLL INT,

SEC NVARCHAR(50),

)

print ' created exists'

end

else

begin

print'exist'

end

--ADDING A COLUMN

if not EXISTS(SELECT \* FROM INFORMATION\_SCHEMA.COLUMNS WHERE COLUMN\_NAME='GOD3' AND TABLE\_NAME='REUSABILITY\_2')

BEGIN

ALTER TABLE REUSABILITY\_2

ADD GOD3 INT

PRINT 'FUCKIN DONE'

END

ELSE

BEGIN

PRINT'EXISTS BABYYYYY'

END

--ANOTHER METHOD

IF COL\_LENGTH('REUSABILITY\_2', 'GOD4') IS NOT NULL

BEGIN

--ALTER TABLE REUSABILITY\_2

--ADD GOD4 INT --WILL NEVER WORK

PRINT 'FUCKIN EXISTS'

END

ELSE

BEGIN

PRINT'IDIOT FUCKIN COCKSUCKER ASSHOLE'

END

--ALTERING DATABASE COLUMN'S DATATYPE BY USING SCRIPT

ALTER TABLE REUSABILITY\_2

ALTER COLUMN GOD NVARCHAR(50)

--OPTIONAL PARAMETER IN STORED PROCEDURE

--GENERAL PROCEDURE where all parameters are mandatory

CREATE PROCEDURE OPTIONAL\_PARAMETER\_TRY

@ID INT,

@ROLL INT,

@SEC NVARCHAR(50),

@NAME CHAR(10)

AS

BEGIN

SELECT \* FROM REUSABILITY\_2

WHERE ID=@ID AND ROLL=@ROLL AND SEC=@SEC AND NAME=@NAME

END

-- executing :

select \* from [dbo].[REUSABILITY\_2]

OPTIONAL\_PARAMETER\_TRY --throws error

OPTIONAL\_PARAMETER\_TRY 2 --again asks for roll, sec, name

--CREATING OPTIONAL PARAMETER WITH DEFAULT VALUE

alter PROCEDURE OPTIONAL\_PARAMETER\_TRY

@ID INT=null,

@ROLL INT=null,

@SEC NVARCHAR(50)=null,

@NAME CHAR(10)=null --default value

AS

BEGIN

SELECT \* FROM REUSABILITY\_2

WHERE (ID=@ID or @ID is null) AND (ROLL=@ROLL or @roll is null) AND (SEC=@SEC or @SEC is null) AND (NAME=@NAME or @NAME is null)

END

OPTIONAL\_PARAMETER\_TRY --RETURNS ENTIRE TABLE

OPTIONAL\_PARAMETER\_TRY 2 --RETURNS WHOSE ID IS 2

OPTIONAL\_PARAMETER\_TRY 'BEGUN' --GIVES ERROR, IT SERCHES ONLY BY pk, WHEN VARIABLE NAME IS NOT MENTIONED)

OPTIONAL\_PARAMETER\_TRY @NAME='BEGUN' --NOW GIVES A PERFECT O/P

----------------------------------------VIDEO 68::::::::::::::::::::::::::\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--MERGE IN SQL SERVER: ALLOWS US TO PERFORM insert, update & delete in one line

--require two tables:

1. Source table (CONTAINS THE CHANGES THAT NEEDS TO BE APPLIED ON TARGET),

2. target table (THE TABLE THAT REQUIRS CHANGE{insert, update , delete})

USE TESTY

SELECT \* FROM [dbo].[Student\_source\_for\_MARGE]

1 MIKE

2 SARA

SELECT \* FROM [dbo].[Student\_TARGET\_for\_MARGE]

1 MIKE MM

2 JOHN

MERGE Student\_source\_for\_MARGE A

USING Student\_TARGET\_for\_MARGE B

ON A.ID=B.ID

WHEN MATCHED THEN

UPDATE SET A.NAME=B.NAME

WHEN NOT MATCHED BY TARGET THEN

INSERT (ID, NAME) VALUES(B.ID, B.NAME); ---ENDS WITH SEMI COLON

SELECT \* FROM [dbo].[Student\_source\_for\_MARGE]

1 MIKE MM

2 SARA

3 JOHN

-- **CONCURRENCY IN SQL SERVER----------------------------------------------------🡪**

--SQL SERVER CONCURRENCY PROBLEMS: if we might face error when multiple transactions are running concurrently…

--CONCURRENCY MAKES THE QUERY SLOWER.

--**CONCURRENCY TYPE**----1. dirty read, 2. lost updates, 3. nonrepetable reads, 4. phantom reads

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Isolation level** | dirty read | lost updates | nonrepeatable reads | phantom reads |
| Read uncommitted | Y | Y | Y | Y |
| Read committed | N | Y | Y | Y |
| Repeatable raed | N | N | N | Y |
| Snapshot | N | N | N | N |
| Serializable | N | N | N | N |

--DIRTY READ CONCURRENCY PROBLEM

--DIRTY DATA: A DIRTY READ HAPPENS WHEN a one transection is permitted to data that has been modified by another transection, BUT THEY HAVE STILL NOT BEEN COMMITED.

--EX: IF 1ST TRANSECTION ROLLS BACK, AFTER THE SECOND READS DATA... THEN THE SECOND SITUATION IS CALLED DIRTY DATA.

**USING READ COMMITTED TO GET DIRTY DATA:**

|  |  |
| --- | --- |
| select \* from TEST\_FOR\_DIRTY\_DATA  --making a transection  begin tran  update TEST\_FOR\_DIRTY\_DATA set NUMBER = 9 WHERE ID=1  --BILL PRINTING  WAITFOR DELAY '00:00:15' --SIMULATING THE TIME TO CHECK IF THE CARD IS READY TO FRANSECT  ROLLBACK TRANSACTION  --O/P:  1 iphone 10 | set transaction isolation level read uncommitted  select \* from TEST\_FOR\_DIRTY\_DATA  o/p  1 iphone 9 |

**USING NOLOCK TO GET DIRTY DATA:**

|  |  |
| --- | --- |
| select \* from TEST\_FOR\_DIRTY\_DATA  begin tran  update TEST\_FOR\_DIRTY\_DATA set NUMBER = 9 WHERE ID=1  WAITFOR DELAY '00:00:15' --SIMULATING THE TIME TO CHECK IF THE CARD IS READY TO FRANSECT  ROLLBACK TRANSACTION  --O/P: 1 iphone 10 | --BY DEFAULT  set transaction isolation level read committed  select \* from TEST\_FOR\_DIRTY\_DATA(nolock)  o/p  1 iphone 9 |

**READ COMMITTED: (NEVER ALLOWS THE**

|  |  |
| --- | --- |
| select \* from TEST\_FOR\_DIRTY\_DATA  --making a transection  begin tran  update TEST\_FOR\_DIRTY\_DATA set NUMBER = 9 WHERE ID=1  --BILL PRINTING  WAITFOR DELAY '00:00:15' --SIMULATING THE TIME TO CHECK IF THE CARD IS READY TO FRANSECT  ROLLBACK TRANSACTION  --O/P:  1 iphone 10 | --BY DEFAULT  select \* from TEST\_FOR\_DIRTY\_DATA  O/P:  1 iphone 10 |

**LOST UPDATE PROBLEM (**updating T1 and T2 simultaneously might eliminate the previous COMMIT and that will be replaced with LAST COMMIT, SO LOOSING THE VALUE OF PREVIOUS UPDATE IS CALLED lost update problem.**)**

|  |  |
| --- | --- |
| begin transaction  declare @take int  select @take=NUMBER from TEST\_FOR\_DIRTY\_DATA where ID=1  waitfor delay '00:00:10'  set @take=@take-1  update TEST\_FOR\_DIRTY\_DATA set NUMBER=@take where ID=1  print @take  commit transaction  --o/p::::::(1 row(s) affected) 9 | begin transaction  declare @take int  select @take=NUMBER from TEST\_FOR\_DIRTY\_DATA where ID=1  waitfor delay '00:00:01'  set @take=@take-2  update TEST\_FOR\_DIRTY\_DATA set NUMBER=@take where ID=1  print @take  commit transaction  --o/p:::::(1 row(s) affected) 8 |

Omits the change that had been done at T2, rather it puts only the updation that is done at T1. After updating T1 item no should be 9 then at T2, we sold 2 items, So, item should remain at 7. But T1 is showing (10-1)=9 & T is showing (10-2)=8.

**OVERTAKING**

|  |  |
| --- | --- |
| set transaction isolation level Repeatable Read  begin transaction  declare @take int  select @take=NUMBER from TEST\_FOR\_DIRTY\_DATA where ID=1  waitfor delay '00:00:01'  set @take=@take-2  update TEST\_FOR\_DIRTY\_DATA set NUMBER=@take where ID=1  print @take  commit transaction | set transaction isolation level Repeatable Read  begin transaction  declare @take int  select @take=NUMBER from TEST\_FOR\_DIRTY\_DATA where ID=1  waitfor delay '00:00:01'  set @take=@take-2  update TEST\_FOR\_DIRTY\_DATA set NUMBER=@take where ID=1  print @take  commit transaction  **got an error:**  Transaction (Process ID 54) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.  **Now execute T2, when execution of T1 is done:**  o/p:::::(1 row(s) affected) 7 (correct answer) |

**NON-REPEATABLE READ EXAMPLE (**NON-repeatable read happens when T1 reads same data twice, & T2 updates the data in between first & second read of T1**)**

|  |  |
| --- | --- |
| begin transaction  select NUMBER from TEST\_FOR\_DIRTY\_DATA  waitfor delay '00:00:10'  select NUMBER from TEST\_FOR\_DIRTY\_DATA  commit transaction | update TEST\_FOR\_DIRTY\_DATA set NUMBER=5 where ID=1 |
| \*\*\*IT DOESN’T HAPPEN IN SQL SERVER 2012 OR HIGHER\*\*\* | |

**PHANTHOM READ: (**WHEN A SAME QUERY IS GETTING EXECUTED TWICE BUT IT GETS DIFFERENT NUMBER OF ROWS**)**

|  |  |
| --- | --- |
| select \* from phantom\_read\_2  begin transaction  select \* from phantom\_read\_2 where id between 1 and 3  waitfor delay '00:00:10'  select \* from phantom\_read\_2 where id between 1 and 3  commit transaction  1st o/p:  1 nick  2 mark  2nd o/p:  1 nick  2 mark  3 BROCK | insert into phantom\_read\_2 values (3, 'BROCK') |

**OVERTAKING PHANTOM READ PROBLEM:**

|  |  |
| --- | --- |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  begin transaction  select \* from phantom\_read\_2 where id between 1 and 4  waitfor delay '00:00:10'  select \* from phantom\_read\_2 where id between 1 and 4  commit transaction | insert into phantom\_read\_2 values (4, 'BROCK') |
| \*\*\*this will never give an error\*\*\*  In this case a RANGE LOCK takes place, that means WHEN T1 IS GETTING EXECUTED ALL TRANSECTIONS BETWEEN THE RANGE OF (1 AND 4) HAS BEEN BLOCKED, AND WHEN T1 IS DONE THEN T2 BEGINS. | |

**Difference between REPEATABLE READ VS SERIALIZABLE READ:**

|  |  |
| --- | --- |
| REPEATABLE READ | SERIALIZABLE READ |
| * Prevents updating and deleting * BUT DOESN’T PREVENT \*NEW ROWS BEING ADDED\* * HAVE PHANTOM READ PROBLEMS | * Prevents updating and deleting of a table tread by T1 * IT PREVENTS NEW ROWS BEING ADDED/PHANTOM READ |

**SNAPSHOT ISOLATION LEVEL:**

|  |  |  |  |
| --- | --- | --- | --- |
| **SNAPSHOT** | | **SERIALIZABLE** | |
| * Works by LOCKING method * REDUCES NUMBER of concurrency | | * Works with VERSIONING in TEMPDB * INCREASES number of concurrent transactions | |
| \*\*\*PROVIDES SAME LEVEL OF DATA CONSISTENCY\*\*\* | | | |
| select \* from track  set transaction isolation level serializable  begin transaction  update track set item=6 where id=1  waitfor delay '00:00:06'  --still i've not executed \*commit\*, so its still going on.  --O/P: 1 iphone 6(TEMPORARY UPDATE)(DONE AT A TIME)  commit transaction | alter database concurrency  set allow\_snapshot\_isolation on  select \* from track  set transaction isolation level snapshot  begin transaction  select \* from track where id=1  --still i've not executed \*commit\*, but it showed the result.  --o/p: 1 iphone 10 (it was the initial val. UPDATION DIDNT MAKE ANY CHANGE)  --(DONE AT A TIME)  commit transaction | select \* from track  set transaction isolation level serializable  begin transaction  update track set item=16 where id=1  waitfor delay '00:00:06'  commit transaction | select \* from track  set transaction isolation level serializable  begin transaction  select \* from track where id=1  commit transaction |
| As SNAPSHOT ever locks that is why when T1 is in progress, rather T2 gives o/p. here it is different from serialization | | \*\*\*T2 WORKS ONLY WHEN T1 IS COMMITTED\*\*\*  \*\*\*using CONCURRENCY database\*\*\* | |
| \*\*\*the transaction will be blocked in T2 (when using SNAPSHOT) and try to update the different thing, at a time, on same table.\*\*\*  \*\*\*the system will be UNBLOCKED when T1 will be committed\*\*\* | | | |
| \*\*\*\*\*\*SNAPSHOT UPDATION CASE\*\*\*\*\*\*\* | | | |
| select \* from track  set transaction isolation level serializable  begin transaction  update track set item=7 where id=1 --(INITIAL: 1 iphone 10)  waitfor delay '00:00:10'  commit transaction --commit it, youll get T1 successfull and T2 failed with a message.  --O/P: 1 iphone 7 | | alter database concurrency  set allow\_snapshot\_isolation on  select \* from track  set transaction isolation level snapshot  begin transaction  update track set item=12 where id=1  --blocked as T1 is not committed  commit transaction  --error msg: Snapshot isolation transaction aborted due to update conflict. You cannot use snapshot isolation to access table 'dbo.track' directly or indirectly in database 'concurrency' to update, delete, or insert the row that has been modified or deleted by another transaction. Retry the transaction or change the isolation level for the update/delete statement.  --it happens to resist the OVERWRITTING.  --RETRY IT TABLE WILL RUN PERFECTLY: O/P: (1 iphone 12) | |

**READ COMMITTED SNAPSHOT ISOLATION**

|  |  |  |  |
| --- | --- | --- | --- |
| **Read committed** | | **Read committed SNAPSHOT** | |
| SET TRANSACTION ISOLATION LEVEL READ COMMITTED  begin transaction  update track set item=10 where id=1  commit transaction | set transaction isolation level read committed  begin transaction  SELECT item FROM tracK where id=1  commit transaction | SET TRANSACTION ISOLATION LEVEL READ COMMITTED  begin transaction  update track set item=10 where id=1  commit transaction | ALTER DATABASE [READ\_COMMITTED\_SNAPSHOT] SET READ\_COMMITTED\_SNAPSHOT ON  set transaction isolation level read committed  begin transaction  SELECT item FROM Table\_1 where id=1  commit transaction |
| general case where we execute t1 as \*UNCOMMITTED\* and as a result we get t2 blocked.  as soon as we \*commit\* t1, we get the output of t2.  \*\*USING TABLE “CONCURRENCY”\*\* | | \*\*\*in this case you gonna get an error as another connection is on that is why t2 READ\_COMMITTED\_SNAPSHOT will not work\*\*\*  \*\*\*in this case we can easily manage to see the T2 when T1 has not been committed\*\*\*  \*\*\*SNAPSHOT isolation level also does same thing\*\*\* | |

**difference between SNAPSHOT ISOLATION vs READ COMMITTED SNAPSHOT**

|  |  |  |  |
| --- | --- | --- | --- |
| **SNAPSHOT ISOLATION** | | **READ COMMITTED SNAPSHOT** | |
| * RESTRICTS the updation of T2 when T1 is not committed. \*\*\*\*THROWS AN ERROR\*\*\*\*\* as soon as we commit T1. | | * DOESN’T RESTRICT the updation of T2 when T1 is not committed. \*\*\*\* NEVER THROWS AN ERROR\*\*\*\*\* as soon as we commit T1. | |
| ALTER DATABASE SAMPLE  SET ALLOW\_SNAPSHOT\_ISOLATION ON  SET TRANSACTION ISOLATION LEVEL SNAPSHOT  begin transaction  update Table\_1 set item=11 where id=1  --commit transaction  select \* from [dbo].[Table\_1] | set transaction isolation level snapshot  begin transaction  update Table\_1 set item=15 where id=1  --commit transaction  \*\*\*\*THROWS ERROR\*\*\*\*\*\*  AT T1 | ALTER DATABASE [SAMPLE]  SET ALLOW\_SNAPSHOT\_ISOLATION OFF  ALTER DATABASE [SAMPLE]  SET READ\_COMMITTED\_SNAPSHOT ON \*  SET TRANSACTION ISOLATION LEVEL READ COMMITTED  begin transaction  update Table\_1 set item=11 where id=1  commit transaction | set transaction isolation level READ COMMITTED  begin transaction  update Table\_1 set item=15 where id=1  commit transaction  \*NEVER GIVES AN ERROR\*  \*UPDATES T1 FIRST and then T2, that is why when SELECT query is executed we get the UPDATED VAL FROM T2.\* |
| * Doesn’t works with existing application. We gotta CHANGE “set transection isolation level snapshot” a number of times. | | * Works with EXISTING APPLICATION SMOOTHLY, * WE JUST NEED TO CHANGE :   READ\_COMMITTED\_SNAPSHOT ON;\* | |
| * can not be used in distributive transection | | * can be used in distributive transection | |
| * provides TRANSECTION LEVEL consistency. | | * provides STATEMENT LEVEL consistency | |
| ALTER DATABASE [SAMPLE]  SET ALLOW\_SNAPSHOT\_ISOLATION OFF  SET TRANSACTION ISOLATION LEVEL READ COMMITTED  begin transaction  update Table\_1 set item=8(\*) where id=1  commit transaction | set transaction isolation level SNAPSHOT  begin transaction  select items from Table\_1 where id=1 –-10(last committed)  (\*) commit;  select items from Table\_1 where id=1 –-10(last committed)  commit transaction  select items from Table\_1 where id=1 --8 | ALTER DATABASE [SAMPLE]  SET ALLOW\_SNAPSHOT\_ISOLATION OFF  ALTER DATABASE [SAMPLE]  SET READ\_COMMITTED\_SNAPSHOT ON \*  SET TRANSACTION ISOLATION LEVEL READ COMMITTED  begin transaction  update Table\_1 set item=8(\*) where id=1  commit transaction | set transaction isolation level READ COMMITTED  begin transaction  select items from Table\_1 where id=1 –-10(last committed)  (\*) commit;  select items from Table\_1 where id=1 –-8(last committed)  commit transaction |
| Gives a transection level, as the different select statement in T2 is a part of same transection, that is why we get 10 both time,  Even after committing T1, being a part of same transection T2’s select statement(2nd) gives 10. | | As it is a statement level that is why, when we update T1’s update statement, T2’s select statement(1st) produces 10  When we commit T1, we get 8 at T2’s select statement(2nd) | |

**DEADLOCKS IS SQL SERVER:** deadlocks occurs when two or more process have a resource locked. Eack process looks for a resource that has been locked. Nither transection can move forward, as each ione is waiting for another to release the lock.

|  |  |
| --- | --- |
| begin transaction  update deadlock\_1 set item=9 where id= 1  --1. has an exclusive lock on T1  commit transaction  begin transaction  update deadlock\_2 set item=12 where id=1  --2. T1 becomes successed  commit transaction | begin transaction  update deadlock\_1 set item=5 where id= 1  --2. T2 is chosen as victim with an error  commit transaction  begin transaction  update deadlock\_2 set item=7 where id=1  --1. has an exclusive lock on T2  commit transaction |

**SQL SERVER DEADLOCKS DETECTIONS:**

LOCK MONITOR THREAD runs in SQL server within every 5 seconds by default, that finds if there is any DEADLOCKS, if it finds a deadlock then the interval will drop from sec to 100 milliseconds depending on deadlocks. If the LOCK MONITOR THREAD stops finding deadlocks the database engine increases the searching time from 5 sec.

**WHAT HAPPENS WHEN A DEADLOCK IS FACED:**

Database engine ends deadlocks by choosing one of the threads as VICTIM. VICTIM transection is ROLLED BACK with a 1205 error message. Other is moved forward.

**Deadlock priority:**

A deadlock priority is chosen by SQL server by default, whose priority is less/ easier to ROLLBACK..

Set DEADLOCK\_PRIORITY NORMAL/ high/ low or(-5 as low, 0 as normal, 5 as high)

**TEST TABLES:**

|  |  |
| --- | --- |
| 1 iphone 10  2 tab 10  3 nokia 10 | 1 s8 10 |

|  |  |
| --- | --- |
| **DEFAULT CASE:** | |
| begin transaction  update deadlock\_1 set item=10 where id between 1 and 3  update deadlock\_2 set item=10 where id between 1 and 3  --(1 row(s) affected)  commit transaction | begin transaction  update deadlock\_2 set item=55  update deadlock\_1 set item=55 where id between 1 and 3  --Transaction (Process ID 52) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.  commit transaction |
| **Set DEADLOCK\_PRIORITY high** | |
| begin transaction  update deadlock\_1 set item=85 where id between 1 and 3  update deadlock\_2 set item=10 where id between 1 and 3  --Transaction (Process ID 52) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.  commit transaction | SET DEADLOCK\_PRIORITY HIGH  begin transaction  update deadlock\_2 set item=55  update deadlock\_1 set item=55 where id between 1 and 3  --(3 row(s) affected)  commit transaction |

**ERRORLOG**

|  |  |
| --- | --- |
| create proc deadlock\_1\_modN  as  begin  begin transaction  update deadlock\_1 set name='apple '  waitfor delay '00:00:05'  update deadlock\_2 set name= 'samsung'  commit transaction  end  deadlock\_1\_modN  **sp\_readerrorlog** | create proc deadlock\_1\_mod\_2N  as  begin  begin transaction  update deadlock\_2 set name='apple\_1'  waitfor delay '00:00:05'  update deadlock\_1 set name= 'samsung\_1'  commit transaction  end  deadlock\_1\_mod\_2N |

**PARTS OF ERRORLIST**

|  |  |
| --- | --- |
| **SECTION** | **DESCRIPTION** |
| Deadlock Victim | Contains the id of the process that has become the victim and killed by SQL server. |
| Process List | Contains the list of the process that is participated in deadlock. |
| Resource List | Contains the list of the database objects owned by process involved in deadlock. |

**Process List Section**

|  |  |
| --- | --- |
| Login Name | The login name associated with the process |
| Isolation level | The isolation level is used |
| Procname | The stored procedure name |
| inputbuf | The code process is executing when deadlock occurs |

**How to overtake the error happened just before: (putting the two transactions in same order)**

|  |  |
| --- | --- |
| delete from deadlock\_1 where id between 2 and 3  create proc deadlock\_1\_modN  as  begin  begin transaction  update deadlock\_1 set name='apple '  waitfor delay '00:00:05'  update deadlock\_2 set name= 'samsung'  commit transaction  end  deadlock\_1\_modN | alter proc deadlock\_1\_mod\_2N  as  begin  begin transaction  update deadlock\_1 set name= 'samsung\_1'  waitfor delay '00:00:05'  update deadlock\_2 set name='apple\_1'  commit transaction  end  deadlock\_1\_mod\_2N |

**Error handling**

|  |  |
| --- | --- |
| select \* from deadlock\_1  select \* from deadlock\_2  update deadlock\_1 set item=15  create proc error\_handel  as  begin  begin transaction  begin try  update deadlock\_1 set item=16 --**1 apple 15**  waitfor delay '00:00:05'  update deadlock\_2 set item=56 --**1 samsung 55**  commit transaction  print 'go for susu'  end try  begin catch  if(ERROR\_NUMBER()=1205)  begin  print 'fuckin deadlock'  end  rollback transaction  end catch  end  error\_handel  o/p: **(1 apple 16) / (1 samsung 56)** | alter proc error\_handel\_2  as  begin  begin transaction  begin try  update deadlock\_2 set item= 20  waitfor delay '00:00:05'  update deadlock\_1 set item= 50  commit transaction  print 'go to pee'  end try  begin catch  if(ERROR\_NUMBER()=1205)  begin  print 'deadlock bitch'  end  rollback transaction  end catch  end  error\_handel\_2  o/p:  (1 row(s) affected)  (0 row(s) affected)  deadlock bitch |
| \*\*\* If you put the order same in T1 and T2 then no error will occur.\*\*\* | |

**FINDING BLOCKAGES WITHIN MULTIPLE TRANSACTION:**

|  |  |
| --- | --- |
| select \* from deadlock\_1  begin transaction  update deadlock\_1 set name= 'APPLE IOS'  COMMIT TRANSACTION | UPDATE deadlock\_1 SET NAME='APPLE DB' –-BLOCKED FOR T1  TRUNCATE TABLE deadlock\_1 –-BLOCKED FOR T1  SELECT \* FROM deadlock\_1 –-BLOCKED FOR T1  DBCC OPENTRAN –SHOWS OLDEST ACTIVE TRANSACTION  --SHOWS:  Transaction information for database 'testy'.  Oldest active transaction:  SPID (server process ID): 54  UID (user ID) : -1  Name : user\_transaction  LSN : (28610:200:1)  Start time : Sep 26 2017 9:16:11:750AM  SID : 0x010500000000000515000000af463958b3fca52dadc2463ee9030000  DBCC execution completed. If DBCC printed error messages, contact your system administrator. |
| \*\*\*When T1 is COMMITTED, then EXECUTING DBCC OPENTRAN gives nothing\*\*\*  No active open transactions. | |
| \*\*\*to get all transaction at a time\*\*\* | |
| SELECT      [s\_tst].[session\_id],      [s\_es].[login\_name] AS [Login Name],      DB\_NAME (s\_tdt.database\_id) AS [Database],      [s\_tdt].[database\_transaction\_begin\_time] AS [Begin Time],      [s\_tdt].[database\_transaction\_log\_bytes\_used] AS [Log Bytes],      [s\_tdt].[database\_transaction\_log\_bytes\_reserved] AS [Log Rsvd],      [s\_est].text AS [Last T-SQL Text],      [s\_eqp].[query\_plan] AS [Last Plan]  FROM      sys.dm\_tran\_database\_transactions [s\_tdt]  JOIN      sys.dm\_tran\_session\_transactions [s\_tst]  ON      [s\_tst].[transaction\_id] = [s\_tdt].[transaction\_id]  JOIN      sys.[dm\_exec\_sessions] [s\_es]  ON      [s\_es].[session\_id] = [s\_tst].[session\_id]  JOIN      sys.dm\_exec\_connections [s\_ec]  ON      [s\_ec].[session\_id] = [s\_tst].[session\_id]  LEFT OUTER JOIN      sys.dm\_exec\_requests [s\_er]  ON      [s\_er].[session\_id] = [s\_tst].[session\_id]  CROSS APPLY      sys.dm\_exec\_sql\_text ([s\_ec].[most\_recent\_sql\_handle]) AS [s\_est]  OUTER APPLY      sys.dm\_exec\_query\_plan ([s\_er].[plan\_handle]) AS [s\_eqp]  ORDER BY      [Begin Time] ASC;  GO  o/p:  57 DESKTOP-88GJBMO\HP testy 2017-09-26 09:37:59.950 248 9475 begin transaction  update deadlock\_2 set item=65 NULL  54 DESKTOP-88GJBMO\HP testy 2017-09-26 09:38:08.620 248 9475 begin transaction  update deadlock\_1 set name= 'APPLE IO' NULL | |
| \*\*\*\*\*\*TO KILL A TRANSACTION\*\*\*\*\*  RIGHT CLICK ON SERVER INSTANCE 🡪 ACTION CENTER🡪 PROCESSES🡪 KILL TASK  OR  KILL <PROCESS\_ID>  ALL TRANSACTIONS ARE ROLLED BACK AS SOON AS WE KILL A PROCESS | |

**WORKING WITH EXCEPT OPERATOR:**

|  |  |  |
| --- | --- | --- |
| SELECT \* FROM [dbo].[EXCEPT\_OPERATOR\_1]  SELECT \* FROM [dbo].[EXCEPTION\_OPERATOR\_2]  SELECT ID, NAME, ROLL FROM EXCEPT\_OPERATOR\_1  EXCEPT  SELECT ID, NAME, ROLL FROM EXCEPTION\_OPERATOR\_2 | 1 NICK 5  **2 JOHN 6** –O/P  3 SIRI 8  **4 NICCI 9** –O/P | 3 SIRI 8  1 NICK 5  7 JOHNNY 2  8 SAM 3 |

**EXCEPTION OPERATOR CAN ALSO FUNCTION ON A SINGLE TABLE:**

|  |  |  |
| --- | --- | --- |
| SELECT ID, NAME, SALARY FROM EXCEPT\_OPERATOR\_TESTING\_ON\_A\_SAME\_TABLE WHERE SALARY>=500000  EXCEPT  SELECT ID, NAME, SALARY FROM EXCEPT\_OPERATOR\_TESTING\_ON\_A\_SAME\_TABLE WHERE SALARY<=720000  **--ORDER BY** (always comes here) | 1 NICK 500000  2 RICK 250000  3 SARA 600000  **4 TAMINA 850000** –O/P  5 PAUL 550000  6 ROSE 650000  7 KART 705000  8 FLAIR 600000 |  |
| \*\*\*SOMEWHERE EXCEPT OPERATOR WORKS AS “END” or “NOT IN” CONDITIONS\*\*\* | | |

**DIFFERENCE & SIMILARITY BETWEEN “EXCEPT” & “NOT IN”**

|  |  |  |
| --- | --- | --- |
| SELECT ID, NAME, ROLL FROM EXCEPT\_OPERATOR\_1  EXCEPT  SELECT ID, NAME, ROLL FROM EXCEPTION\_OPERATOR\_2  **--O/P:**  **2 JOHN 6**  **4 NICCI 9**  SELECT ID, NAME, ROLL FROM EXCEPT\_OPERATOR\_1 WHERE ID NOT IN (SELECT ID FROM EXCEPTION\_OPERATOR\_2)  **--O/P:**  **2 JOHN 6**  **4 NICCI 9** | 1 NICK 5  2 JOHN 6  3 SIRI 8  4 NICCI 9 | 3 SIRI 8  1 NICK 5  7 JOHNNY 2  8 SAM 3 |
| 1 NICK 5  2 JOHN 6  3 SIRI 8  4 NICCI 9  4 NICCI 9 | 3 SIRI 8  1 NICK 5  7 JOHNNY 2  8 SAM 3 |
| SELECT ID, NAME, ROLL FROM EXCEPT\_OPERATOR\_1  EXCEPT  SELECT ID, NAME, ROLL FROM EXCEPTION\_OPERATOR\_2  **--O/P:**  **2 JOHN 6**  **4 NICCI 9**  SELECT ID, NAME, ROLL FROM EXCEPT\_OPERATOR\_1 WHERE ID NOT IN (SELECT ID FROM EXCEPTION\_OPERATOR\_2)  **--O/P:**  **2 JOHN 6**  **4 NICCI 9**  **4 NICCI 9** | * EXCEPT operator doesn’t allow REPETATIONS * NOT IN does. | |
| SELECT **ID, NAME, ROLL** FROM EXCEPT\_OPERATOR\_1  EXCEPT  SELECT ID, NAME, ROLL FROM EXCEPTION\_OPERATOR\_2  SELECT **ID, NAME, ROLL** FROM EXCEPT\_OPERATOR\_1 WHERE ID NOT IN (SELECT **ID** FROM EXCEPTION\_OPERATOR\_2) | * except needs same NUMBER OF ATTRIBUTES at both queries. * whereas not in allows DIFFERENT attributes. | |

**DIFFERENCE AND SIMILARITY BETWEEN INNER JOIN VS INTERSECT**

|  |  |  |  |
| --- | --- | --- | --- |
| SELECT ID, NAME, GENDER FROM INTERSECT\_OPERATION  INTERSECT  SELECT ID, NAME, GENDER FROM ITERSECT\_OPERATION\_2 | | SELECT E.GENDER, F.NAME  FROM INTERSECT\_OPERATION E  RIGHT JOIN ITERSECT\_OPERATION\_2 F  ON E.ID=F.TAKE2 | |
| 1 MARRY FEMALE  2 SARA FEMALE  3 KAKA MALE | 2 SARA FEMALE 2  3 KAKA MALE 1 | | **FEMALE SARA**  **FEMALE KAKA** |

**UNION VS INTERSECT VS EXCEPT**

* UNION TAKES different values from TAB1 and TAB2, whereas DUPLICATIONS are eliminated.
* UNION ALL takes all values from TAB and TAB2, whereas DUPLICATIONS are KEPT.
* INTERSECT takes only common elements between TAB1 & TAB2
* .EXCEPT operator takes only distinct values from ANY ONE OF TABLES.

**CROSS APPLY & OUTER APPLY**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **I**  **N**  **N**  **E**  **R** | select e.Name, e.Gender, e.Salary, f.DepartmentName  from Department\_for\_cross\_outer\_apply f  **inner join** Employee\_for\_cross\_outer\_apply e  on f.id=e.DepartmentId | | select e.Name, e.Gender, e.Salary, f.DepartmentName  from Department\_for\_cross\_outer\_apply f  **cross apply** rat(f.Id) e  --on f.id=e.DepartmentId | Mark Male 50000 IT  Mary Female 60000 Payroll  Steve Male 45000 HR  John Male 56000 IT  Sara Female 39000 HR |
| **L**  **E**  **F**  **T** | select f.DepartmentName, e.Name, e.Gender, e.Salary  from Department\_for\_cross\_outer\_apply f  left join Employee\_for\_cross\_outer\_apply e  on e.DepartmentId=f.Id | | select f.DepartmentName, e.Name, e.Gender, e.Salary  from Department\_for\_cross\_outer\_apply f  outer apply rat(f.Id) e  --on e.DepartmentId=f.Id | IT Mark Male 50000  IT John Male 56000  HR Steve Male 45000  HR Sara Female 39000  Payroll Mary Female 60000  Admini NULL NULL NULL  Sales NULL NULL NULL |
| \*\*\*\* A TABLE VALUED FUNCTION CAN’T BE JOINED WITH A TABLE THAT IS WHY , “CROSS APPLY / OUTER APPLY” IS NEEDED. | | create function rat(@takedeptID int)  returns table  as  return  (  select \*  from Employee\_for\_cross\_outer\_apply  where DepartmentId=@takedeptID  ) | | select \* from rat(1)  1 Mark Male 50000 1  4 John Male 56000 1 |

**TABLES:**

|  |  |
| --- | --- |
| 1 Mark Male 50000 1  2 Mary Female 60000 3  3 Steve Male 45000 2  4 John Male 56000 1  5 Sara Female 39000 2 | 1 IT  2 HR  3 Payroll  4 Administration  5 Sales |