DBS211 Introduction to Database Systems

WEEK- 6

Transactions and Security

Agenda

- What is a Transaction
- Transaction Scope
- Commit, Save Point and Rollback
- DDL and Transactions
- Concurrency
- User Level Security
- Grant and Revoke

Transactions and Security

- Transactions are performed millions of times every day
- Protection of data is very important
- Use error handling to determine the ultimate success or failure of the transaction
- Grant and revoke various privileges from both public and local users to a database

Transactions

- A transaction is a logical, atomic unit of work that contains one or more SQL statements.
- Committed
  - transaction is applied to the database
- Rolled back
  - transaction is undone from the database.

Real World Transaction Example

Table: TRANSACTIONS				
TransID	**Date**	**Account Number**	**Transaction Type**	**Amount**
33	May 21, 2020	5678	Debit	99.99
44	May 21, 2020	7890	Credit	1257.41
55	May 22, 2020	3456	Debit	200.00
66	May 22, 2020	1234	Credit	200.00

Note: A credit is a deposit (positive) into your account and a debit is a withdrawal (negative).

Example contd…

Table: ACCOUNTS			
Account Number	**Account Type**	**CustID**	**Balance**
1234	Chq	24	25.27
3456	Sav	24	1589.42
5678	Chq	32	4892.34
7890	Chq	56	789.46

Transaction steps

1. Check to see is user has permissions to withdrawal money from the savings account

SELECT accountno WHERE accountno = 3456 AND custID = &UsersCustomerID; -- if there is a result, permission granted

2. Check to see if there is enough money in the savings account to withdrawal $200

SELECT accountno WHERE accountno = 3456 AND balance >= 200; -- if there is a result, sufficient funds

Transaction steps contd..

3. Create a record in the transactions table for the withdrawal from savings (transactionid 55)

INSERT INTO transactions VALUES(55, sysdate, 3456, 'Debit', 200);

4. Update the accounts table record for the savings account to make sure the balance is correct (update account number 3456)

UPDATE accounts SET balance = balance - 200 WHERE accountNo = 3456;

5.

Check to make sure the user has permission to deposit money into the chequings account

SELECT accountno WHERE accountno = 1234 AND custID = &UsersCustomerID; -- if there is a result, permission granted

Transaction steps contd..

6. Create a record in the transactions table for the deposit to chequings (transactionid 66)

INSERT INTO transactions VALUES(66, sysdate, 1234, 'Credit', 200);

7. Update the accounts table record for the chequing account to make sure the balance is correct (update account number 1234)

UPDATE accounts SET balance = balance + 200 WHERE accountNo = 1234;

8. Send out a confirmation code stating everything was successful

The concept that all steps must complete successfully, or none are completed. If the deposit into chequing fails, the withdrawal from savings must be reversed and undone like it never happened.

Transaction Scope

```
BEGIN transaction;   -- or   START transaction;
or   just   BEGIN;
    .... do stuff...
COMMIT;
END transacion;   -- or   just   END;
```

This clearly states when the transaction starts and ends and the statements between are the statements of the transaction that must all complete successfully.

4 most common SQL Developer transactions

1. The user has established a new connection to the server and has a blank sheet ready to go, starts a new transaction
2. The user uses the BEGIN statement in Oracle SQL, this will start a new transaction
3. The user executes a COMMIT statement, the current transaction is commited, and a new transaction starts.
4. The user executes ANY DDL statement. This automatically triggers an auto-commit of the current transaction and starts a new transaction.

Commit, Save Points, Rollback

In Oracle, the three main statements used in transactions are COMMIT, SAVEPOINT and ROLLBACK.

Commit

any pending changes to the data are permanently made on the database server.

Example 1 with DML

1. Open a new worksheet connected to your database.
2. Create an INSERT statement on any table and execute that statement
3. Create a SELECT statement that will recall that new record and see that it is in the table.
4. SAVE YOUR open .sql file(s)
5. Now crash SQL Developer (On Windows use Ctrl-Alt-Delete and go to task manager, right click on SQL Developer and click End Task Alternatively you can just close SQL Developer and when prompted, select to abort the current session. (But it is a more effective demonstration if you crash it)
6. Now reopen SQL Developer, connect to the database and open the .sql file you were working on.
7. Run the SELECT statement again.

What happened??? The record you inserted, and saw was there, is no longer there.

Example 2:

1. Rerun the INSERT statement you ran above,
2. run the SELECT statement again to see it is there
3. create and execute a COMMIT statement
4. SAVE YOUR .sql file
5. now crash or close and abort SQL Developer again.
6. Reload SQL Developer, reconnect and open your file again
7. Execute the SELECT statement once again.

Auto-Commit

SET AUTOCOMMIT ON

SET AUTOCOMMIT OFF

By default auto-commit is off.

If autocommit is on, then DML statements are automatically commited immediately upon successful execution without the need for commit.

SET TRANSACTION Statement

Used to set a transaction as read-only, set a transaction as read/write, assign a name to a transaction, or assign a rollback segment to a transaction.

SET TRANSACTION [READ ONLY | READ WRITE]
[NAME 'transaction_name'];

SET TRANSACTION READ ONLY NAME 'RO_example';

SET TRANSACTION READ WRITE NAME 'RW_example';

Rollback

The rollback statement will undo all the executed statements in the current transaction as if nothing had happened.

```
COMMIT;    -- force starts a new transaction

INSERT INTO players (playerID, firstname, lastname, DOB, email)
    VALUES (1667, 'George', 'Dunkirk', to_date('19860428','yyyymmdd'),
'gdunkirk@email.com');

SELECT * FROM players WHERE playerID = 1667;     -- there will be one
record                                           shown
ROLLBACK;
SELECT * FROM players WHERE playerID = 1667;     -- no records shown
```

Rollback another example

```
COMMIT;    -- force starts a new transaction

INSERT INTO players (playerID, firstname, lastname, DOB, email)
    VALUES (1667, 'George', 'Dunkirk', to_date('19860428','yyyymmdd'),
'gdunkirk@email.com');

SELECT * FROM players WHERE playerID = 1667;     -- there will be one
record shown

COMMIT;

ROLLBACK;

SELECT * FROM players WHERE playerID = 1667;     -- the record is still
there
```

Savepoints

SAVEPOINT is a statement that can be used in larger transactions to place markers at specific locations throughout the transaction. This now allows ROLLBACK to only undo part of the transaction, and not the whole thing.

Example

```
COMMIT;    -- force starts a new transaction

INSERT INTO players (playerID, firstname, lastname, DOB, email)
    VALUES (1667, 'George', 'Dunkirk', to_date('19860428','yyyymmdd'), 'gdunkirk@email.com');

SAVEPOINT A;

SELECT * FROM players WHERE playerID = 1667 OR playerID = 1668     -- there will be one record
shown

INSERT INTO players (playerID, firstname, lastname, DOB, email)
    VALUES (1668, 'Henry', 'Buggles', to_date('19851203','yyyymmdd'), 'gbuggles@email.com');

SELECT * FROM players WHERE playerID = 1667 OR playerID = 1668;     -- there will be two records
shown

ROLLBACK TO A;

SELECT * FROM players WHERE playerID = 1667 OR playerID = 1668      -- only one record will be
shown as 1668 was rolled back
```

Another Example

```
COMMIT;    -- force starts a new transaction

INSERT INTO players (playerID, firstname, lastname, DOB, email)
    VALUES (1667, 'George', 'Dunkirk', to_date('19860428','yyyymmdd'), 'gdunkirk@email.com');

SAVEPOINT A;

SELECT * FROM players WHERE playerID = 1667 OR playerID = 1668     -- there will be one record
shown

INSERT INTO players (playerID, firstname, lastname, DOB, email)
    VALUES (1668, 'Henry', 'Buggles', to_date('19851203','yyyymmdd'), 'gbuggles@email.com');

SELECT * FROM players WHERE playerID = 1667 OR playerID = 1668;     -- there will be two records
shown

COMMIT;

ROLLBACK TO A;  -- error

SELECT * FROM players WHERE playerID = 1667 OR playerID = 1668      -- Both records will be shown
as they were both committed.
```

Transaction with Error Checking

```
BEGIN
    INSERT INTO players (playerID, firstname, lastname, DOB, email)
        VALUES (1667, 'George', 'Dunkirk', to_date('19860428','yyyymmdd'), 'gdunkirk@email.com');
    INSERT INTO players (playerID, firstname, lastname, DOB, email)
        VALUES (1668, 'Henry', 'Buggles', to_date('19851203','yyyymmdd'), 'gbuggles@email.com');
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Transaction Successful!');
EXCEPTION     -- when any error occurs after BEGIN and therefore COMMIT is skipped
    ROLLBACK;    -- rolls back everything in the transaction
    DBMS_OUTPUT.PUT_LINE('Transaction Failed, rolled back');
END;
```

Don’t mix DDL and DML for rollback

```
COMMIT;    -- force starts a new transaction

INSERT INTO players (playerID, firstname, lastname, DOB, email)
    VALUES (1667, 'George', 'Dunkirk', to_date('19860428','yyyymmdd'), 'gdunkirk@email.com');

INSERT INTO players (playerID, firstname, lastname, DOB, email)
    VALUES (1668, 'Henry', 'Buggles', to_date('19851203','yyyymmdd'), 'gbuggles@email.com');

SELECT * FROM players WHERE playerID = 1667 OR playerID = 1668;     -- there will be two
records shown

CREATE VIEW vsShowPlayers AS
    SELECT * FROM players WHERE playerID = 1667 OR playerID = 1668;
SELECT * FROM vwShowPlayers;

ROLLBACK;
```

Single-use vs Multiuser DBMS

- A DBMS can be
  - single-user
    - At most one user can use the database at the time
  - Multiuser
    - Multiple users can concurrently access the database at the same time.
- Multiuser Systems:
  - Banks
  - Supermarkets
  - Insurance companies
  - Stock market systems

Data Concurrency and Consistency

Data concurrency means that many users can access data at the same time.

Data consistency means that each user sees a consistent view of the data, including visible changes made by the user's own transactions and transactions of other users.

How Oracle Locks Data

Locks are mechanisms that prevent destructive interaction between transactions accessing the same resource—either user objects such as tables and rows or system objects not visible to users, such as shared data structures in memory and data dictionary rows.

Modes of Locking

- Exclusive lock mode prevents the associates resource from being shared. This lock mode is obtained to modify data. The first transaction to lock a resource exclusively is the only transaction that can alter the resource until the exclusive lock is released.

- Share lock mode allows the associated resource to be shared, depending on the operations involved. Multiple users reading data can share the data, holding share locks to prevent concurrent access by a writer (who needs an exclusive lock). Several transactions can acquire share locks on the same resource.

  - Lock Duration

- All locks acquired by statements within a transaction are held for the duration of the transaction, preventing destructive interference including dirty reads, lost updates, and destructive DDL operations from concurrent transactions.

Types of Locks

Lock	Description
DML locks (data locks)	DML locks protect data. For example, table locks lock entire tables, row locks lock selected rows.
DDL locks (dictionary locks)	DDL locks protect the structure of schema objects—for example, the definitions of tables and views.
Internal locks and latches	Internal locks and latches protect internal database structures such as datafiles. Internal locks and latches are entirely automatic.

Oracle automatically uses different types of locks to control concurrent access to data and to prevent destructive interaction between users.

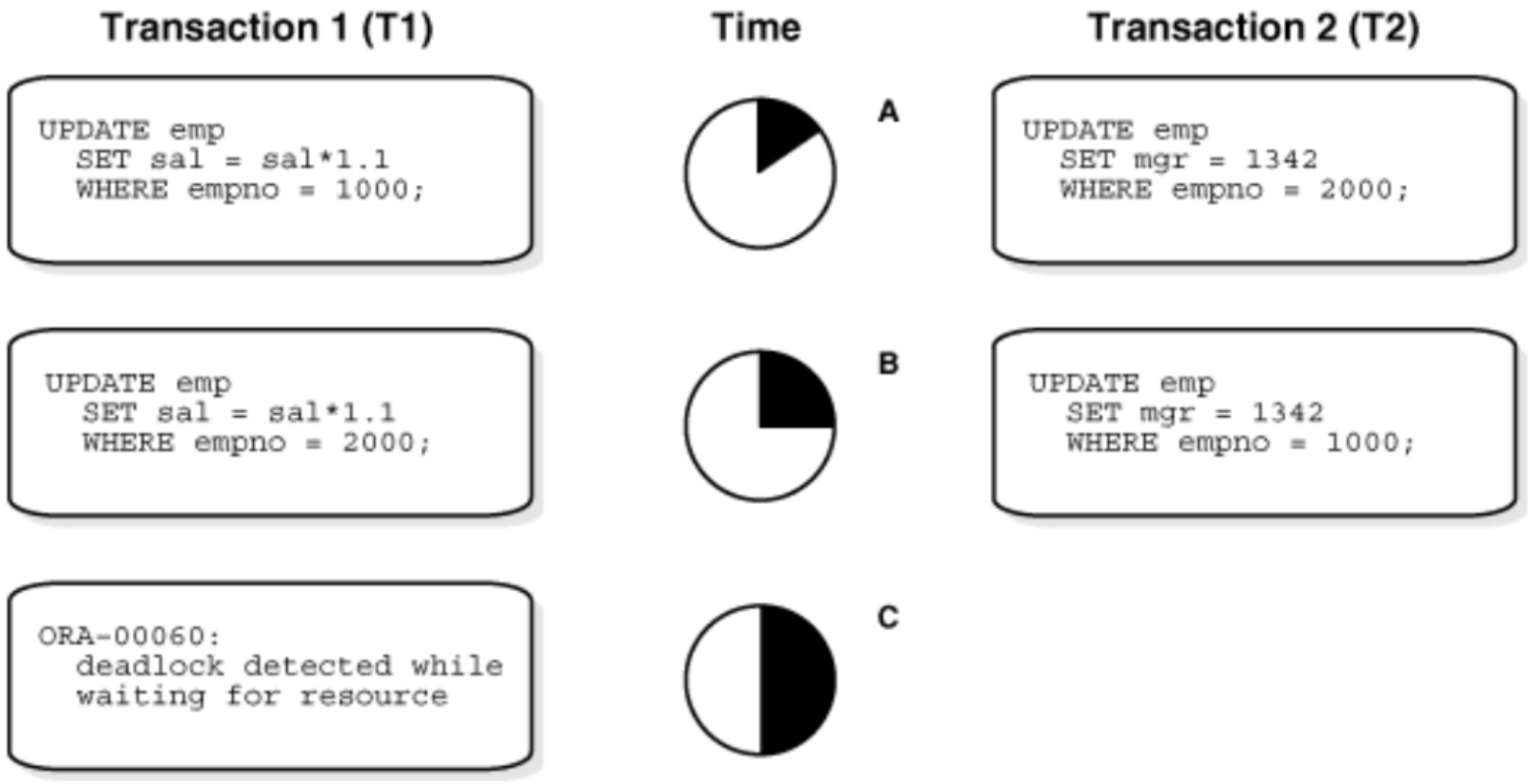
More details can be found at https://docs.oracle.com/cd/B19306_01/server.102/b14220/consist.htm

Deadlocks

A deadlock can occur when two or more users are waiting for data locked by each other.

Deadlocks prevent some transactions from continuing to work.

Two Transactions in a Deadlock



Deadlock Detection

- Oracle automatically detects deadlock situations and resolves them by rolling back one of the statements involved in the deadlock, thereby releasing one set of the conflicting row locks.

GRANT

- Use the GRANT statement to grant:
  - privileges to a specific user or role,
  - or to all users,
  - to perform actions on database objects.
  - You can also use the GRANT statement to grant a role to a user, to PUBLIC, or to another role.

Types of privileges can be granted

- Delete data from a specific table.
- Insert data into a specific table.
- Create a foreign key reference to the named table or to a subset of columns from a table.
- Select data from a table, view, or a subset of columns in a table.
- Create a trigger on a table.
- Update data in a table or in a subset of columns in a table.
- Run a specified function or procedure.
- Use a sequence generator or a user-defined type.

Grant syntax for tables

- GRANT privilege-type ON [TABLE] { table-Name | view-Name } TO grantees

privilege-types

```
ALL PRIVILEGES |
privilege-list
```

privilege-list

```
table-privilege {, table-privilege }*
```

table-privilege

```
DELETE |
INSERT |
REFERENCES [column list] |
SELECT [column list] |
TRIGGER |
UPDATE [column list]
```

column list

```
( column-identifier {, column-identifier}* )
```

Example: GRANT

- Suppose you are the owner of Sells. You may say:

  GRANT SELECT, UPDATE(price)

  ON Sells

  TO sally;

- Now Sally has the right to issue any query on Sells and can update the price component only.

More Grant Examples

- For table test and user sally, see following examples:

```
GRANT SELECT ON test TO sally;
GRANT INSERT,UPDATE, DELETE ON test TO sally;
GRANT ALL ON test TO sally;
GRANT SELECT ON test TO sally;
```

Revoke

- Use the REVOKE statement to remove privileges from a specific user or role, or from all users, to perform actions on database objects. You can also use the REVOKE statement to revoke a role from a user, from PUBLIC, or from another role.

- The following types of privileges can be revoked:
  - Delete data from a specific table.
  - Insert data into a specific table.
  - Create a foreign key reference to the named table or to a subset of columns from a table.
  - Select data from a table, view, or a subset of columns in a table.
  - Create a trigger on a table.
  - Update data in a table or in a subset of columns in a table.
  - Run a specified routine (function or procedure).
  - Use a sequence generator or a user-defined type.

Revoke contd..

- Revoke privileges from users
- Revoke privileges ON tablename FROM user;
- Example:
  - REVOKE DELETE ON `test` FROM sally;
  - REVOKE INSERT,UPDATE ON `test` FROM sally;
  - REVOKE ALL ON `test` FROM sally;

Revoke contd..

- To revoke the SELECT privilege on table test from all users, use the following syntax:
  - REVOKE SELECT ON TABLE test FROM PUBLIC
  - REVOKE UPDATE (c1,c2) ON TABLE test FROM PUBLIC