# DBMS UNIT V

Transactions, ACID Properties
and
Schedules

# What is a transaction?

- It is not very common to have individual INSERT, UPDATE or DELETE operations on the database.
- Often a group of statements have to be executed together.
- Transaction is a group of update (insert / modify / delete) operations on the database.
- Transaction is a group of statements modifying the database state.
- The database state changes from one (consistent) state to another (consistent) state due to transactions.

# Transaction Examples

- Consider transfer of funds from one bank account to another.
  - Two insert statements have to be issued for this (because there are two user accounts)
  - Two update statements have be be issued for this (To update the balance of the two accounts).
  - One insert statement to record a withdrawal record.
  - One UPDATE statement for decreasing the amount from one account.
  - One insert statement to record a deposit record.
  - One UPDATE statement for increasing the amount of another account.
- Either all the statements have to be executed or none of them have be executed to ensure that the database is in a consistent state (database integrity is maintained).

# Transaction Examples ...(2)

- Consider another example of a retail store:
  - When a customer buys a number of items, all the items the customer has bought have to be inserted together.
  - Insert one record into the order_header table having details about customer, date, and total amount.
  - Insert a number of records into the order_details tables having a record for each item purchased.

# Transaction Examples ...(3)

- Consider another example of a Warehouse:
  - When an item is taken out of a warehouse for delivering to customer, records for removing the items from the warehouse has to be inserted, as well as a record that keeps track of the count of items may have to be updated (by reducing the count of items remaining in the warehouse)…
  - How many tables are impacted?

# Transaction support in SQL

By enclosing the SQL statements within **BEGIN;** and **END**; statements, we are telling the database that all the statements are part of a transaction.

- example-1: Simple transfer 5000 from one account to another.

```
BEGIN;
UPDATE account_balance
      SET balance = balance - 5000
 WHERE account_number = '10000001';
UPDATE account_balance
       SET balance = balance + 5000
 WHERE account_number = '10000002';
END;
```

# Transaction support in SQL ...(2)

example-2: Transfer after checking balance:

```
BEGIN;
SELECT balance INTO :var_balance
FROM account_balance
WHERE account_number  = '10000001';

IF (var_balance - 5000 > minimum_balance) THEN
    UPDATE account_balance
          SET balance = balance - 5000
     WHERE account_number = '10000001';
    UPDATE account_balance
           ET balance = balance + 5000
     WHERE account_number = '10000002';
END;
```

# Transaction support in SQL ...(3)

**COMMIT**; and **ROLLBACK**; statements. When the DBMS encounters the END; statement it issues a COMMIT;

A commit is a confirmation of the operations performed from the BEGIN; statement.

We can also issue an explicit COMMIT; statement.

```
BEGIN;
   Statement-1;
   Statement-2;
   COMMIT; /* All the statements up to this point - 1 and 2 are "committed" (written) to the database. */
   Statement-3;
   Statement-4;
END; /* The remaining statements - 3 and 4 are written this time. */
```

# Transaction support in SQL ...(4)

**SAVEPOINT savepoint_name**; statement.

In case of long transactions, All-or-None can be very expensive. We may want to save small number of operations periodically. Or sometimes, we may have to rollback some operations (UNDO some statements).

If there was an error in executing Statement-3, or 4 then the operations of statement 3 and 4 are undone and commits only the successful operations (statement 1 and 2) else all the four statements are written to the disk.

```
BEGIN;
   Statement-1;
   Statement-2;
   SAVEPOINT savepoint_1;
   Statement-3;
   Statement-4;
   IF ERROR THEN
      ROLLBACK TO savepoint_1;
END;
```

# Transactions Properties

- ACID
  - Atomicity
    - A group of database operations (Insert, Update, Delete) executed together as a unit. Either all (insert/update/delete statements) or none.
  - Consistency
    - Each transactions receives a consistent database and leaves the database in a consistent state. User has to ensure consistent transactions.
  - Isolation
    - Users need not be aware of other transactions. Their transactions happen as if they are Isolated from others.
    - Even though many users access the database and many operations may be interleaved, the net effect is same as that of executing the transactions one after another. The user feels that he is the only user of the database.
  - Durability
    - Once user gets a confirmation of the transaction execution, the effect must persist even if a system failure occurs.

# Atomicity and Durability

- Transactions may be aborted. The database may crash during a transaction. In such situations, the incomplete (aborted) transactions have to be undone to ensure that the database is in a consistent state.
- The recovery manager will bring the database to a consistent state after each start up.
- The database ensures Atomicity by undoing the incomplete transactions.
- The DBMS writes the transactions to a log file.
- After start up, the log is read to redo the completed transactions and undo the incomplete transactions.

# Isolation and Consistency

- The user gets a consistent database. By consistent, we mean a database where all the integrity constraints are satisfied. Including constraints defined by the semantics of the application (for example, an accounting system should have debits and credits match and add up to ZERO).
- The user must ensure that the transaction he/she performs shall not leave the database in an inconsistent state. If two transaction are operating on unrelated objects(tables), then they can definitely be run in any sequence. Consistency can be lost by the interleaving of transactions.

# Isolation and Consistency ...(2)

- For example, two transactions on a single account if interleaved, can read the balance amount and increase or decrease it to leave the database in an inconsistent state:
  - txn1: read balance (say B); increase balance by X amount.
  - txn2: read balance (say B); decrease balance by Y amount.
- If txn1 is followed by txn2 in totality, then the databases is in a consistent state. However, if we interleave the transactions, we may get different scenarios:
  - 1. T1 read balance B;
  - 2. T2 read balance B;
  - 3. T2 write balance - Y;
  - 4. T1 write balance + X;
- The resulting balance (balance + X) is incorrect.

# Isolation and Consistency ...(3)

- Alternatively,
  - 1. T1 read balance B;
  - 2. T2 read balance B;
  - 3. T1 write balance + X;
  - 4. T2 write balance - Y;
- The resulting balance (balance - Y) is incorrect.
- To avoid such situation, we can run all transactions in a sequence. But, this will reduce the transaction throughput (or transactions per minute) of the database. And is definitely not worthwhile.
- The dbms software must provide a mechanism of ensuring consistency and also have mechanism to interleave transactions safely without making the database inconsistent.
- The property that the user sees a database as being available exclusively (even though there are other users) is called **Isolation.**

# Techniques

- The ACID properties are achieved by:
  - Features for grouping operations
    - BEGIN; END; COMMIT; ROLLBACK; SAVEPOINT;
  - Transaction logging (Write Ahead Log or WAL)
- DBMS supports multi-user, multi-connection, concurrent operations by:
  - Identifying opportunities for concurrent operations
  - Support for locking.

# Transactions

- A sequence of operations performed together.
  - Insert, Update, Delete, Select
- Generalized as
  - Read Operation on an Object (table)
  - Write Operation on a Object (table)
- The Reads and Writes are on a database object
  - Represented as:
    - $R_T(O)$: Read an object O(Table) (T is the transaction)
    - $W_T(O)$: Read an object O(Table) (T is the transaction)
    - The transaction is concluded by Commit or Abort(Rollback)

# Transaction Schedule

- A Schedule is a list of actions like
  - Read, Write, Commit, Abort, Shared Lock eXclusive Lock for a transaction
- The list of actions appear in the same sequence as in the transaction.

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(B) |
| | W(B) |
| R(C) | |
| W(C) | |

# Transaction Schedule

- A Transaction Schedule is a sequences of operations that specify the order in which operations of concurrent transactions are executed.
  - A schedule for a set of transactions must consist of all operations of those transactions.
  - It must preserve the order in which the operations appear in each transaction.
  - A transaction that successfully completes its execution will have a commit operations as the last operation. If not stated, it is assumed to commit.
  - A transaction that fails to successfully complete its execution will have an abort operation as the last operation.

# Transaction Schedule

- In this transaction schedule (and the previous example), the operations of the two transactions are interleaved.

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(C) |
| | W(C) |
| R(B) | |
| W(B) | |
| | R(D) |
| | W(D) |
| | Commit |
| Commit | |

# Motivation for Concurrent Execution

- A line of people waiting at
  - Retail checkout counter (one person's at a time)
- If a person wants to check out a single item.
  - Instead of waiting.
  - The retail agent, checks out that single item.
- Average time
  - For making small orders wait:
    - 6 Min for large order + 6 minutes of wait plus 30 seconds for small order
    - Average transaction time = (6 min + 6 min 30 sec)/2
  - For checking out small orders:
    - 30 seconds for small order + 6min 30 seconds for large order
    - (30 seconds + 6min 30 seconds)/2

# Motivation ...(2)

- A large order is like a big database query involving large amounts of data. Meaning multiple disk read operations.
- A small order is like a simple query.
- It makes sense to run the transactions whenever they can be executed rather than in the order in which the database receives them.

# Motivation ...(3)

- Increase **system throughput**
  - More transactions per minute.
  - While CPU is idle waiting for some I/O operation to complete, it can process other transactions.
- Improve **response time**
  - A short transaction behind a large transaction may take a long time to complete.

# Anomalies due to Interleaved Execution

- Two transactions reading the database(Read-Read) are not affected by interleaving.
- But, The following operations result in anomalies **when the same object** is involved:
  - Write-Read (write followed by a read)
  - Read-Write (read followed by a write)
  - Write-Write (write followed by a write)

# Write-Read (Dirty Read) Anomaly

- T1 Writes, T2 Reads what is written by T1
  - T1 may not be committed (is Aborted or rolled back)
- Reading uncommitted transaction is called dirty read and is generally not allowed by DBMS

| T1 (transfer 100 from A to B) | T2 (Update A & B with 6% of balance) |
|---|---|
| R(A) Check balance of A | |
| W(A) Subtract 100 from A | |
| | R(A) Read balance updated by T1 |
| | W(A) Update balance by 1.06* balance |
| | R(B) Read balance of B |
| | W(B) Update balance by 1.06* balance |
| | Commit |
| R(B) update by T2 with new balance | |
| W(B) Add 100 to B | |
| Rollback | |

# Read-Write (Unrepeatable Reads)

- T1 Reads, T2 Writes, T1 assumes old value & continues

| T1 (Sell a book) | T2 (Sell a book) |
|---|---|
| R(A) Stock available? Yes: 1 copy | |
| W(B) Sell a book. | |
| | R(A) Stock Available? Yes 1 Copy |
| | W(B) Sell a Book |
| | W(A) Update Stock 0 Copies |
| | Commit |
| W(A) Update Stock Error: Stock can't be -1” | |
| Commit | |

# Write-Write (Overwriting uncommitted data)

- T1 Writes, T2 Overwrites: Only T2 is available.
- Lost Update.

# Some Terms

- Schedule
- Dirty Read: Reading uncommitted data.
- Unrepeatable reads
- Overwriting Uncommitted Data

# Questions