# Transaction

Collection of **one or more operations** on **one or more databases** which reflects a **single real world transaction**

## Transaction Properties ACID

* Atomic
* Consistent
* Isolated
* Durable

|  |  |
| --- | --- |
| **COMMIT** | **ROLLBACK** |
| * When you want to store changes made by the transaction * **Request**, meaning system might reject it with reason | * When you want to **cancel/abort transaction process** and **revert** tot eh **most recent stable state before transaction** beginning   + User’s change of mind   + Explicit program calls for **error handling**   + Integrity constraints   + System crash |

**Auto-commit**

* When commit is done after EACH SQL COMMAND

**Python DB-API** is **explicit mode** meaning you need to call **commit()** to store and complete transactions

Transaction is considered **consistent if**

* All **Static Integrity** is satisfied
* No **Dynamic constraints** have been violated
* **New State** satisfies specification of **transaction**

## Transaction Checking

|  |  |
| --- | --- |
| Automatically | Manually |
| * CHECK * ASSERTION * TRIGGER * Not always desirable since unnecessary checking might result in slower processing time | * Perform checks in application code only when it is needed * Difficult to maintain as transactions are **modified/added** |

## Deferrable Integrity constraints

* When you can withhold the Integrity constraint checking until after transaction is complete

## Incomplete Transaction, System crash recovery

* When System crash happens **during transaction** the process should clean up and call **rollback** responsively to revert transaction.
* Each item holds the value of the last committed transaction for this **rollback functionality**

## Concurrency

* To allow transactions to be **MULTI-THREAD SAFE**

### Serializability

* + Process of finding elements that do not require **changes to internal state of database** and only locking the database to those processes **that updates**
  + Performance drop due to bottleneck

## Logging

* **Append only** collection of entries showing all the changes to the data that happened in order as they happened
* they are used to **abort and rollback** Transactions as well for **crash recovery** to figure out the **last stable state to revert to**
* Requires **extra** **storage**
* Possible delay in **transaction commit** as it needs to **store information**

## Distributed commit

* When **transaction changes** are done **onto more than one tables**, they require **two batches of information** to be **relayed**
  + Checking whether **all databases have log entries safe on disk**
  + **Inform** each database of **outcome** and **tell them to commit**
* Due to the number of information transfer, it has very long delay

## Locking

* Blocking transactions if another transaction already has a lock
* Risk of **dreadlock**

**ISOLATION**

What information should Transactions read?

|  |  |  |
| --- | --- | --- |
| Transactions to read **Committed data** | Transactions to read **uncommitted data** | Transaction to have **repeatable read** |
| * Data being read is always clean and stable * It can be inconsistent because commit could be taken place **as reading is done** | * Data being read can be **dirty** as transaction is not complete | * Unless the transaction updates the record, the data is always the same |