**Atomicity**

Part 1 (without atomicity) explain:-

* Django Default Behaviour is to run in auto commit mode. Each query is immediately committed to the database.

Problem that occur due to this is: - User a balance reduces even if the payee does not exist.

**Customer.objects.filter(name=x).update(balance=F(‘balance’) - z**

Commit T1

**Customer.objects.filter(name=y).update(balance=F(‘balance’) + z**

Commit T2

Part 2 Implementing Transaction:-

|  |
| --- |
| **Customer.objects.filter(name=x).update(balance=F(‘balance’) - z**  Save Point  **Customer.objects.filter(name=y).update(balance=F(‘balance’) + z**  Commit T2  Fail – Roll Back Success - Saved |

Transaction ideally Have 4 Property which commonly known as **ACID:-**

A 🡪 Atomicity 🡪 It means that you guaranteed that all of the transaction succeeds or none of it done.

C 🡪 Consistency 🡪 It ensure that you guaranteed that all data will be consistent.

I 🡪 Isolated 🡪 All the transaction will occur in isolation. No tnx will be affected by any other transactions.

D 🡪 Durable 🡪 Once a transaction is committed, it will remain in system, even if there is system crash immediately following the transaction.

Note:-

Never used try catch block into the transaction block. Cause, if the error is raise than the whole block is failed to execute. Into transaction block, we write some code lines than must priority is all lines should be error free. If any case we used try catch block than only raise exception. Do not handle exception into transaction atomic block.

1) Save points :- A savepoint is a marker within a transaction that enables you to roll back part of a transaction, rather than the full transaction. Savepoints aren’t especially useful if you are using autocommit, the default behavior of Django. Savepoints provide the ability to perform a fine-grained rollback, rather than the full rollback that would be performed by **transaction.rollback()**.

2) **Select\_for\_update** method to update any queries. Also with a filter or first(). By default, *select\_for\_update*() locks all rows that are selected by the query until the transaction is done. For example, rows of related objects specified in select\_related() are locked into a transaction.

3) **Roll Back** :- Calling **transaction.rollback()** rolls back the entire transaction.

4) Save point Rollback :- To control the extent of a rollback. Before performing a database operation that could fail, you can set or update the savepoint; that way, if the operation fails, you can roll back the single offending operation, rather than the entire transaction.  
For Example :-  
a.save() *# Succeeds, and never undone by savepoint rollback*sid = transaction.savepoint()  
**try**:  
 b.save() *# Could throw exception* transaction.savepoint\_commit(sid)  
**except** IntegrityError:  
 transaction.savepoint\_rollback(sid)  
c.save() *# Succeeds, and a.save() is never undone*

*Q -> Exception Handling types?*

*@-> supervisor to run all tab automatic.*

*Q -> docker ?*

*Q -> httppy, curl,calt,azure,*

Reference :-  
https://pynative.com/python-mysql-transaction-management-using-commit-rollback/  
<https://realpython.com/transaction-management-with-django-1-6/>  
https://www.saaspegasus.com/guides/django-stripe-integrate/