# Transaction Isolation Level for the after\_user\_insert Trigger

## Introduction

In high-traffic systems like the Netflix API, concurrent user activities such as simultaneous registration require robust handling to ensure consistency and performance. Database transaction isolation levels control how and when changes made by one transaction become visible to others, reducing the risk of conflicts and ensuring accurate data processing.

This document explains the use of the Read Committed isolation level for PostgreSQL’s after\_user\_insert trigger. It details why this level strikes the right balance between data accuracy and system performance, with practical examples.

## Why Read Committed Was Chosen

The Read Committed isolation level ensures that each query within a transaction sees only committed (finalized) changes made by other transactions. Uncommitted updates from concurrent transactions remain invisible. This behavior is ideal for Netflix’s user registration workflow, where speed and correctness are critical.

The after\_user\_insert trigger performs the following tasks upon a new user registration:  
1. Checks whether the new user was invited by another user (via the invitations table).  
2. Updates credits both the inviter and the new user if the invitation is valid.

Using Read Committed guarantees that only complete, valid invitation data is read and processed. For instance, if a concurrent transaction is still inserting an invitation record, it will be ignored until committed. This prevents errors due to incomplete data reading.

Additionally, Read Committed allows for better performance during high traffic such as during marketing campaigns—because it avoids unnecessary row-level locking. Transactions can proceed concurrently without blocking each other unless they modify the same rows, which is rare in this scenario.

## Detailed Examples

Example 1: Single User Registration  
A user registers with the email newuser@netflix.com. During this process, the after\_user\_insert trigger:  
- Queries the invitations table to check for a valid inviter.  
- If found, updates credits for both the inviter and the new user.

If the invitation record is still being written by another transaction, it will be skipped until it's committed. This ensures only consistent, finalized data is acted upon, preventing partial updates or incorrect credit allocation.

Example 2: Concurrent Registrations  
Suppose two users, userA@netflix.com and userB@netflix.com, register at the same time, each referred by a different subscriber. The trigger:  
- Processes both registrations independently.  
- Updates different rows in the users and invitations tables.

With Read Committed, each transaction sees only committed rows, avoiding conflict. Since the rows being modified are unrelated, these operations do not block or interfere with one another. This allows the system to scale under load.

## Why Not a Stricter Isolation Level?

More restrictive levels, like Serializable, simulate serial execution and offer the strongest consistency guarantees. However, they also impose greater overhead, locking more rows and increasing transaction wait times.

For user registration—where each transaction typically updates different records—such strictness is unnecessary. Serializable would reduce system throughput with little to no benefit, especially during peak traffic.

Read Committed is sufficient because:  
- It avoids reading uncommitted/incomplete data.  
- It enables safe concurrent inserts and updates.  
- It maintains system responsiveness and scalability.

## Implementation Note

Node.js service uses an explicit transaction block to manage operations:

A screen shot of a computer code

AI-generated content may be incorrect.

This setup is compatible with Read Committed, which is PostgreSQL's default isolation level. Unless explicitly overridden, each transaction uses this level, ensuring consistent behavior without additional configuration.

## Conclusion

The Read Committed isolation level is the optimal choice for the after\_user\_insert trigger in the Netflix user registration system. It ensures:  
- Processing only finalized, valid data  
- Efficient handling of concurrent user registrations  
- High system throughput without compromising data integrity

By choosing Read Committed, Netflix’s API maintains scalability, correctness, and responsiveness during even the most intense traffic spikes.