Create Goal:

In a system with no concurrency control, this operation may encounter issues primarily related to phantom reads. A phantom read occurs when a transaction re-issues a query and notices new records meeting certain criteria (WHERE condition), which were not present during the initial query. These new records, introduced by another transaction, are referred to as "phantoms".

For instance, if a transaction initially retrieves all workouts associated with a certain user, and another concurrent transaction adds a new goal for the same user, a re-execution of the first transaction's query would yield an additional phantom record.

Create Projection:

The creation of projections might face similar phantom read anomalies in the absence of concurrency control. Suppose a transaction fetches some records based on certain criteria, and concurrently, another transaction inserts a new record meeting the same criteria. Should the initial transaction repeat its query, it would then detect a phantom row, a record that didn't exist during the first operation.

For example, if a transaction fetches all projections associated with a particular user, and another transaction concurrently adds a new projection for that user, the re-execution of the first transaction's query will result in an extra phantom record.

Get Projection:

Although this operation mainly involves reading data, it's not immune to phantom reads if there are simultaneous transactions that modify the data it is reading.

For instance, a transaction may retrieve all projections for a specific user. Concurrently, another transaction might add a new projection for the same user. Upon re-executing its query, the first transaction would encounter a phantom record that wasn't present during the initial query.

Conclusion:

Our transactions, which don't perform update operations or re-read rows, are not susceptible to lost updates, read skew, write skew, and non-repeatable reads. Moreover, as Postgres defaults to the 'Read Committed' isolation level, dirty reads are not a concern.

To ensure adequate isolation for our transactions, we can adopt the 'Serializable' isolation level. Although this is the strictest level, it effectively prevents the aforementioned anomalies. With this level, each transaction behaves as if it's the sole operation in the system, thereby enhancing data consistency. However, it's crucial to remember that this level might impact system performance negatively due to its strictness.

Ultimately, the choice of isolation level should reflect the application's specific needs, the degree of concurrency expected, and the permissible risk level concerning the potential phenomena. In general, as isolation levels increase, the system ensures better data consistency at the potential expense of reduced concurrency and possible performance reduction.