**1. Class and Naming Improvements**

* **Old Code**:
  + Used ReturnDapper and ReturnDapper<T> for result handling, which lacked clear naming conventions and purpose.
  + ReturnDapper had unnecessary properties like Gridreader and IDataReader.
* **New Code**:
  + Introduced DbExecutionResult<T>, which has a clearer and more meaningful name.
  + Simplified result handling with properties:
    - ResultSet: Holds single result sets.
    - CombinedResultSets: Holds multiple result sets.
    - ReturnStatus and ErrorCode are retained for execution status.
    - Added a convenience property IsSuccess for quick status checking.

**2. Improved Methods**

* **Old Code**:
  + Many methods (Query, QueryFirst, QuerySingle, Execute, etc.) were duplicated for sync/async operations.
  + Transactional versions had repetitive try-catch blocks.
  + Methods used dynamic results without enforcing strong typing, leading to potential type issues.
* **New Code**:
  + Refactored and optimized methods for:
    - **Standard Execution**: ExecuteAsync, ExecuteScalarAsync, QueryAsync.
    - **Transactional Execution**: ExecuteAsyncTrans, QueryAsyncTrans.
    - **Multiple Result Sets**: QueryMultipleAsync.
  + Unified method patterns and removed duplication.
  + Added clear logging for success, failure, and exceptions.

**3. Enhanced Exception Handling**

* **Old Code**:
  + Logged exceptions using Console.WriteLine, which is not ideal for production code.
  + Threw raw exceptions without standardized handling.
* **New Code**:
  + Exceptions are logged using ILogger for structured and centralized logging.
  + Provides a consistent way to handle errors and return meaningful ReturnStatus and ErrorCode.

**4. Output Parameters Handling**

* **Old Code**:
  + Repeated addition of output parameters (ReturnStatus, ErrorCode) in every method.
* **New Code**:
  + Centralized the addition of output parameters in a utility method:

**5. Improved Support for Multiple Result Sets**

* **Old Code**:
  + Used Gridreader and dynamic for multiple result sets, causing casting issues and unclear handling.
* **New Code**:
  + QueryMultipleAsync now stores result sets in a list of IEnumerable<dynamic> (CombinedResultSets).
  + Allows for dynamic support of multiple result sets while being clean and maintainable.

**6. Code Quality Improvements**

* **Old Code**:
  + Many methods had similar logic repeated unnecessarily.
  + The naming was inconsistent and not aligned with modern practices.
* **New Code**:
  + Removed duplication by reusing utility methods and patterns.
  + Strongly typed methods like ExecuteScalarAsync<T> ensure better type safety.
  + Improved method names for clarity and purpose:
    - Example: ExecuteAsync, QueryAsync, ExecuteScalarAsync.

**7. Logging Enhancements**

* **Old Code**:
  + Minimal or no logging for successful operations.
  + Exceptions were logged without context.
* **New Code**:
  + Added detailed logs for:
    - Successful execution.
    - Failed execution with error details.
    - Exceptions during execution.

**8. Consistent Return Structure**

* **Old Code**:
  + Methods directly returned ReturnDapper, which had redundant properties.
* **New Code**:
  + Returns DbExecutionResult<T>, a strongly typed and well-defined result object.

**9. Nullable Property Handling**

* **Old Code**:
  + Potential issues when mapping dynamic to strongly-typed models, especially for null values.
* **New Code**:
  + Strongly typed return results ensure better type safety.
  + Nullable properties are handled safely during model mapping.

**Conclusion**

The new code refactor improves:

1. **Maintainability**: Reduced duplication, added utility methods, and clear separation of concerns.
2. **Readability**: Better naming conventions, logging, and strong typing.
3. **Performance**: Simplified handling of transactions, execution, and multiple result sets.
4. **Robustness**: Improved error handling, logging, and type safety.

The new implementation is cleaner, more scalable, and production-ready.