KeyHub-Project

KH-Data

Tbl

By. lol-lshh

동기

이제는 분산 환경 (클라우드 네이티브 + 마이크로 서비스 아키텍처)

동기

이제는 분산 환경 (클라우드 네이티브 + 마이크로 서비스 아키텍처)

문제. 트랜잭션은 어떻게 할래?

동기

이제는 분산 환경 (클라우드 네이티브 + 마이크로 서비스 아키텍처)

문제. 트랜잭션은 어떻게 할래?

ACID

동기

이제는 분산 환경 (클라우드 네이티브 + 마이크로 서비스 아키텍처)

문제. 트랜잭션은 어떻게 할래?

ACID. CAP.

동기

이제는 분산 환경 (클라우드 네이티브 + 마이크로 서비스 아키텍처)

문제. 트랜잭션은 어떻게 할래?

ACID. CAP.

분산 트랜잭션을 관리해주는 OSS를 만들어 보자.

동기

이제는 분산 환경 (클라우드 네이티브 + 마이크로 서비스 아키텍처)

문제. 트랜잭션은 어떻게 할래?

ACID. CAP.

분산 트랜잭션을 관리해주는 OSS를 만들어 보자. 스프링에서.

동기

이제는 분산 환경 (클라우드 네이티브 + 마이크로 서비스 아키텍처)

문제. 트랜잭션은 어떻게 할래?

ACID. CAP.

분산 트랜잭션을 관리해주는 OSS를 만들어 보자. 스프링에서.

그러기 위해선?

동기

일단 소스단에서의 데이터 자료구조가 필요해!

LINQ

C#

```
// The Three Parts of a LINQ Query:
// 1. Data source.
int[] numbers = [ 0, 1, 2, 3, 4, 5, 6 ];

// 2. Query creation.
// numQuery is an IEnumerable<int>
var numQuery =
    from num in numbers
    where (num % 2) == 0
    select num;

// 3. Query execution.
foreach (int num in numQuery)
{
    Console.Write("{0,1} ", num);
}
```

.NET LINQ를 소개합니다.

https://learn.microsoft.com/ko-kr/dotnet/csharp/linq/get-started/introduction-to-linq-queries

LINQ

C#

```
// The Three Parts of a LINQ Query:
// 1. Data source.
int[] numbers = [ 0, 1, 2, 3, 4, 5, 6 ];

// 2. Query creation.
// numQuery is an IEnumerable<int>
var numQuery =
    from num in numbers
    where (num % 2) == 0
    select num;

// 3. Query execution.
foreach (int num in numQuery)
{
    Console.Write("{0,1} ", num);
}
```

.NET LINQ를 소개합니다.

https://learn.microsoft.com/ko-kr/dotnet/csharp/linq/get-started/introduction-to-linq-queries

소스에서 쿼리를 치네

LINQ

C#

```
// The Three Parts of a LINQ Query:
// 1. Data source.
int[] numbers = [ 0, 1, 2, 3, 4, 5, 6 ];

// 2. Query creation.
// numQuery is an IEnumerable<int>
var numQuery =
    from num in numbers
    where (num % 2) == 0
    select num;

// 3. Query execution.
foreach (int num in numQuery)
{
    Console.Write("{0,1} ", num);
}
```

.NET LINQ를 소개합니다.

https://learn.microsoft.com/ko-kr/dotnet/csharp/linq/get-started/introduction-to-linq-queries

뭔가 이상하다.

LINQ

C#

```
// The Three Parts of a LINQ Query:
// 1. Data source.
int[] numbers = [ 0, 1, 2, 3, 4, 5, 6 ];

// 2. Query creation.
// numQuery is an IEnumerable<int>
var numQuery =
    from num in numbers
    where (num % 2) == 0
    select num;

// 3. Query execution.
foreach (int num in numQuery)
{
    Console.Write("{0,1} ", num);
}
```

.NET LINQ를 소개합니다.

https://learn.microsoft.com/ko-kr/dotnet/csharp/linq/get-started/introduction-to-linq-queries

자료 구조가 그냥 숫자 배열???

LINQ

```C#
// The Three Parts of a LINQ Query:
// 1. Data source.
int[] numbers = [ 0, 1, 2, 3, 4, 5, 6 ];

// 2. Query creation.
// numQuery is an IEnumerable<int>
var numQuery =
    from num in numbers
    where (num % 2) == 0
    select num;

// 3. Query execution.
foreach (int num in numQuery)
{
    Console.Write("{0,1} ", num);
}
```

.NET LINQ를 소개합니다.

https://learn.microsoft.com/ko-kr/dotnet/csharp/linq/get-started/introduction-to-linq-queries

Java엔 StreamAPI가 있으니깐?

LINQ

```C#
// The Three Parts of a LINQ Query:
// 1. Data source.
int[] numbers = [ 0, 1, 2, 3, 4, 5, 6 ];

// 2. Query creation.
// numQuery is an IEnumerable<int>
var numQuery =
    from num in numbers
    where (num % 2) == 0
    select num;

// 3. Query execution.
foreach (int num in numQuery)
{
    Console.Write("{0,1} ", num);
}
```

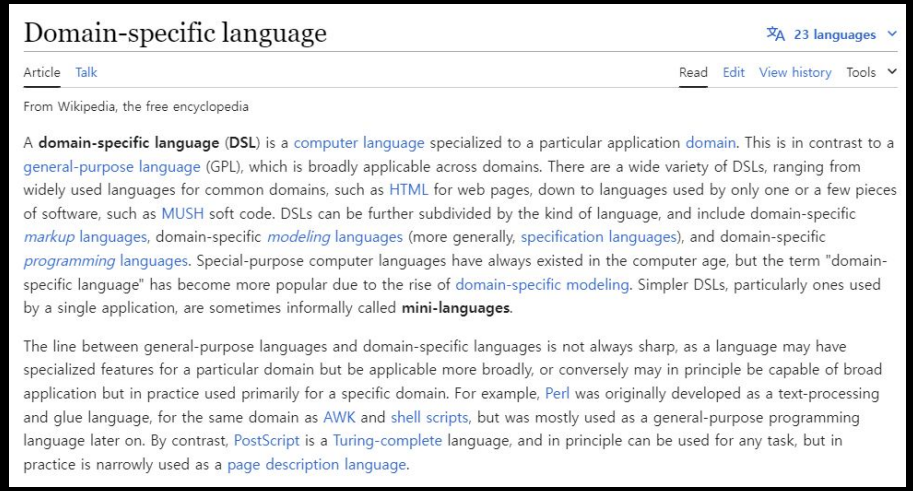
.NET LINQ를 소개합니다.

https://learn.microsoft.com/ko-kr/dotnet/csharp/linq/get-started/introduction-to-linq-queries

Java엔 StreamAPI가 있으니깐?

분산 환경에 맞는 DSL을 작성하고 싶다.

DSL?



Domain-specific language

23 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

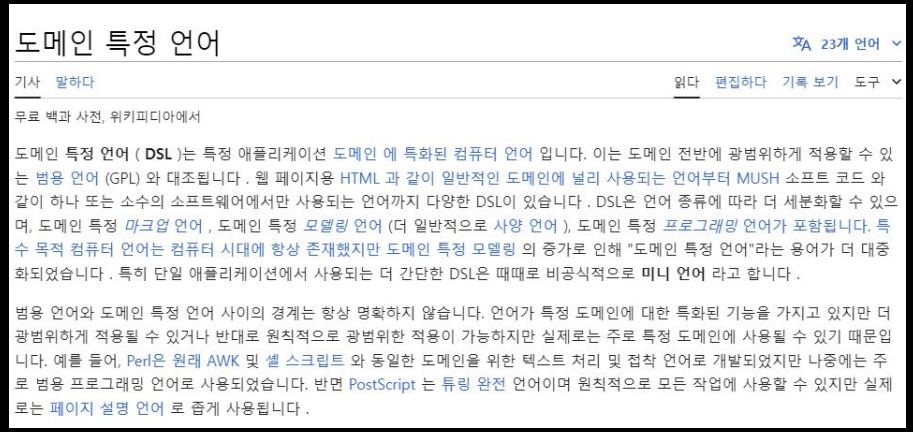
A **domain-specific language** (**DSL**) is a computer language specialized to a particular application domain. This is in contrast to a general-purpose language (GPL), which is broadly applicable across domains. There are a wide variety of DSLs, ranging from widely used languages for common domains, such as HTML for web pages, down to languages used by only one or a few pieces of software, such as MUSH soft code. DSLs can be further subdivided by the kind of language, and include domain-specific *markup* languages, domain-specific *modeling* languages (more generally, specification languages), and domain-specific *programming* languages. Special-purpose computer languages have always existed in the computer age, but the term "domain-specific language" has become more popular due to the rise of domain-specific modeling. Simpler DSLs, particularly ones used by a single application, are sometimes informally called **mini-languages**.

The line between general-purpose languages and domain-specific languages is not always sharp, as a language may have specialized features for a particular domain but be applicable more broadly, or conversely may in principle be capable of broad application but in practice used primarily for a specific domain. For example, Perl was originally developed as a text-processing and glue language, for the same domain as AWK and shell scripts, but was mostly used as a general-purpose programming language later on. By contrast, PostScript is a Turing-complete language, and in principle can be used for any task, but in practice is narrowly used as a page description language.

Domain Specific Language

https://en.wikipedia.org/wiki/Domain-specific_language

DSL?



도메인 특정 언어

23개 언어

기사 말하다 | 읽다 편집하다 기록 보기 도구

무료 백과 사전, 위키피디아에서

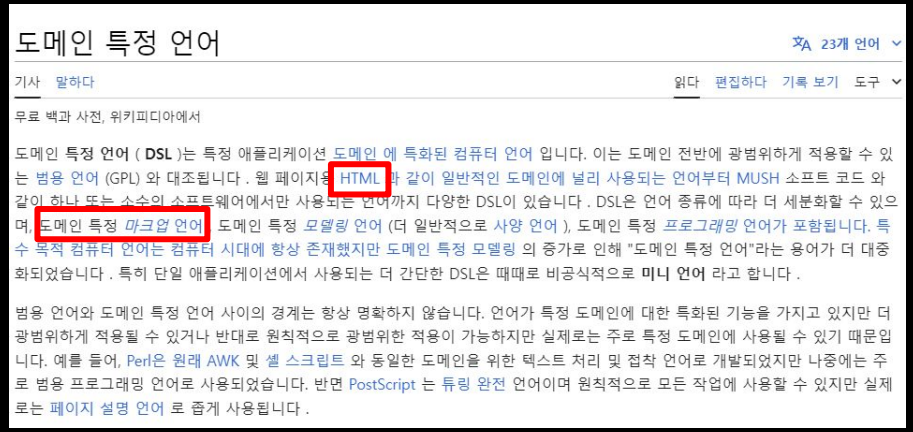
도메인 **특정 언어** (**DSL**)는 특정 애플리케이션 도메인 에 특화된 컴퓨터 언어 입니다. 이는 도메인 전반에 광범위하게 적용할 수 있는 범용 언어 (GPL) 와 대조됩니다 . 웹 페이지용 HTML 과 같이 일반적인 도메인에 널리 사용되는 언어부터 MUSH 소프트 코드 와 같이 하나 또는 소수의 소프트웨어에서만 사용되는 언어까지 다양한 DSL이 있습니다 . DSL은 언어 종류에 따라 더 세분화할 수 있으며, 도메인 특정 *마크업* 언어 , 도메인 특정 *모델링* 언어 (더 일반적으로 사양 언어), 도메인 특정 *프로그래밍* 언어가 포함됩니다. 특수 목적 컴퓨터 언어는 컴퓨터 시대에 항상 존재했지만 도메인 특정 모델링 의 증가로 인해 "도메인 특정 언어"라는 용어가 더 대중화되었습니다 . 특히 단일 애플리케이션에서 사용되는 더 간단한 DSL은 때때로 비공식적으로 **미니 언어** 라고 합니다 .

범용 언어와 도메인 특정 언어 사이의 경계는 항상 명확하지 않습니다. 언어가 특정 도메인에 대한 특화된 기능을 가지고 있지만 더 광범위하게 적용될 수 있거나 반대로 원칙적으로 광범위한 적용이 가능하지만 실제로는 주로 특정 도메인에 사용될 수 있기 때문입니다. 예를 들어, Perl은 원래 AWK 및 셸 스크립트 와 동일한 도메인을 위한 텍스트 처리 및 접착 언어로 개발되었지만 나중에는 주로 범용 프로그래밍 언어로 사용되었습니다. 반면 PostScript 는 튜링 완전 언어이며 원칙적으로 모든 작업에 사용할 수 있지만 실제로는 페이지 설명 언어 로 좁게 사용됩니다 .

Domain Specific Language

https://en.wikipedia.org/wiki/Domain-specific_language

DSL?



도메인 특정 언어

23개 언어

기사 말하다 읽다 편집하다 기록 보기 도구

무료 백과 사전, 위키피디아에서

도메인 **특정 언어** (**DSL**)는 특정 애플리케이션 도메인 에 특화된 컴퓨터 언어 입니다. 이는 도메인 전반에 광범위하게 적용할 수 있는 범용 언어 (GPL) 와 대조됩니다 . 웹 페이지용 HTML 과 같이 일반적인 도메인에 널리 사용되는 언어부터 MUSH 소프트 코드 와 같이 하나 또는 소수의 소프트웨어에서만 사용되는 언어까지 다양한 DSL이 있습니다 . DSL은 언어 종류에 따라 더 세분화할 수 있으며, 도메인 특정 *마크업* 언어 , 도메인 특정 *모델링* 언어 (더 일반적으로 사양 언어), 도메인 특정 *프로그래밍* 언어가 포함됩니다. 특수 목적 컴퓨터 언어는 컴퓨터 시대에 항상 존재했지만 도메인 특정 모델링 의 증가로 인해 "도메인 특정 언어"라는 용어가 더 대중화되었습니다 . 특히 단일 애플리케이션에서 사용되는 더 간단한 DSL은 때때로 비공식적으로 **미니 언어** 라고 합니다 .

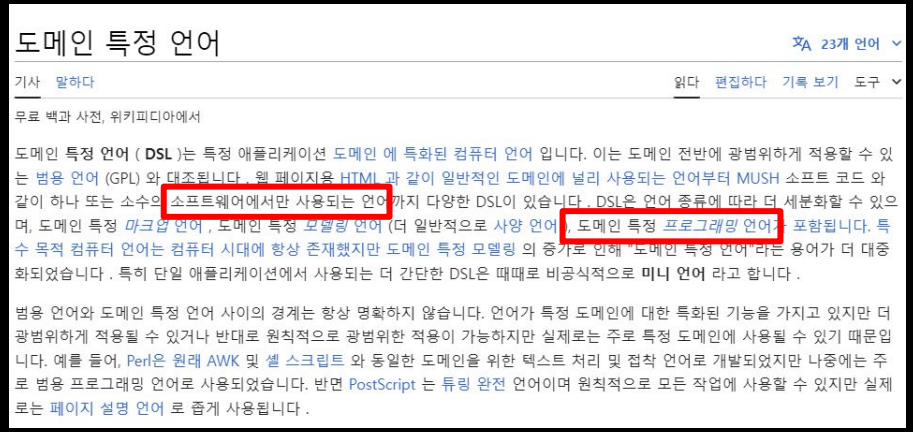
범용 언어와 도메인 특정 언어 사이의 경계는 항상 명확하지 않습니다. 언어가 특정 도메인에 대한 특화된 기능을 가지고 있지만 더 광범위하게 적용될 수 있거나 반대로 원칙적으로 광범위한 적용이 가능하지만 실제로는 주로 특정 도메인에 사용될 수 있기 때문입니다. 예를 들어, Perl은 원래 AWK 및 셸 스크립트 와 동일한 도메인을 위한 텍스트 처리 및 접착 언어로 개발되었지만 나중에는 주로 범용 프로그래밍 언어로 사용되었습니다. 반면 PostScript 는 튜링 완전 언어이며 원칙적으로 모든 작업에 사용할 수 있지만 실제로는 페이지 설명 언어 로 좁게 사용됩니다 .

Domain Specific Language

https://en.wikipedia.org/wiki/Domain-specific_language

HTML.

DSL?



도메인 특정 언어

23개 언어

기사 말하다 읽다 편집하다 기록 보기 도구

무료 백과 사전, 위키피디아에서

도메인 **특정 언어** (**DSL**)는 특정 애플리케이션 도메인 에 특화된 컴퓨터 언어 입니다. 이는 도메인 전반에 광범위하게 적용할 수 있는 범용 언어 (GPL) 와 대조됩니다 . 웹 페이지용 HTML 과 같이 일반적인 도메인에 널리 사용되는 언어부터 MUSH 소프트 코드 와 같이 하나 또는 소수의 소프트웨어에서만 사용되는 언어까지 다양한 DSL이 있습니다 . DSL은 언어 종류에 따라 더 세분화할 수 있으며, 도메인 특정 *마크업 언어* , 도메인 특정 *모델링 언어* (더 일반적으로 사양 언어), 도메인 특정 *프로그래밍 언어* 가 포함됩니다. 특수 목적 컴퓨터 언어는 컴퓨터 시대에 항상 존재했지만 도메인 특정 모델링 의 증가로 인해 "도메인 특정 언어"라는 용어가 더 대중화되었습니다 . 특히 단일 애플리케이션에서 사용되는 더 간단한 DSL은 때때로 비공식적으로 **미니 언어** 라고 합니다 .

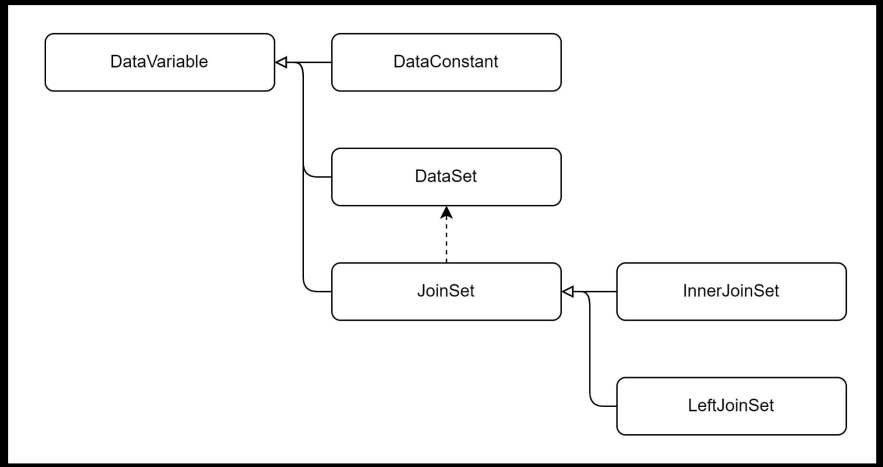
범용 언어와 도메인 특정 언어 사이의 경계는 항상 명확하지 않습니다. 언어가 특정 도메인에 대한 특화된 기능을 가지고 있지만 더 광범위하게 적용될 수 있거나 반대로 원칙적으로 광범위한 적용이 가능하지만 실제로는 주로 특정 도메인에 사용될 수 있기 때문입니다. 예를 들어, Perl은 원래 AWK 및 셸 스크립트 와 동일한 도메인을 위한 텍스트 처리 및 접착 언어로 개발되었지만 나중에는 주로 범용 프로그래밍 언어로 사용되었습니다. 반면 PostScript 는 튜링 완전 언어이며 원칙적으로 모든 작업에 사용할 수 있지만 실제로는 페이지 설명 언어 로 좁게 사용됩니다 .

Domain Specific Language

https://en.wikipedia.org/wiki/Domain-specific_language

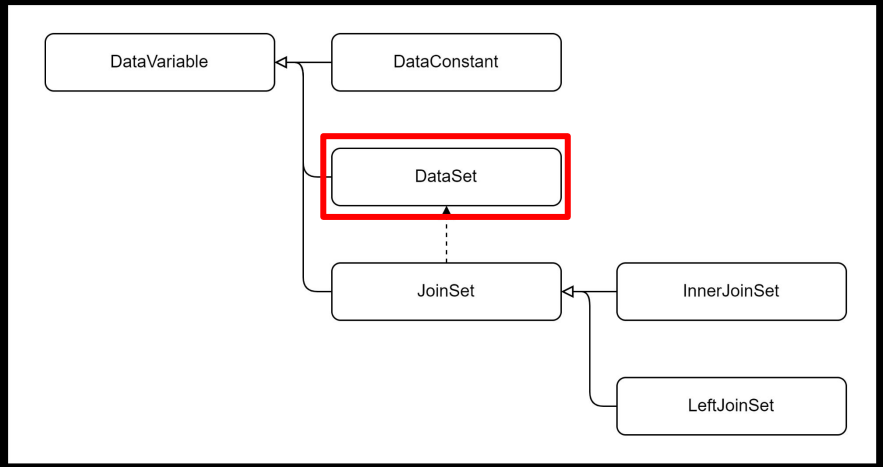
HTML. QueryDSL

초기 모델



단순한 모델, CSV

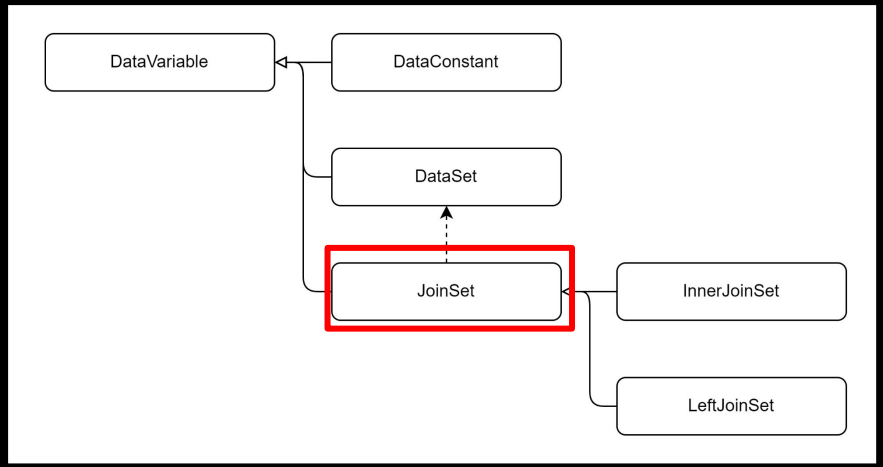
초기 모델



단순한 모델, CSV

DataSet

초기 모델

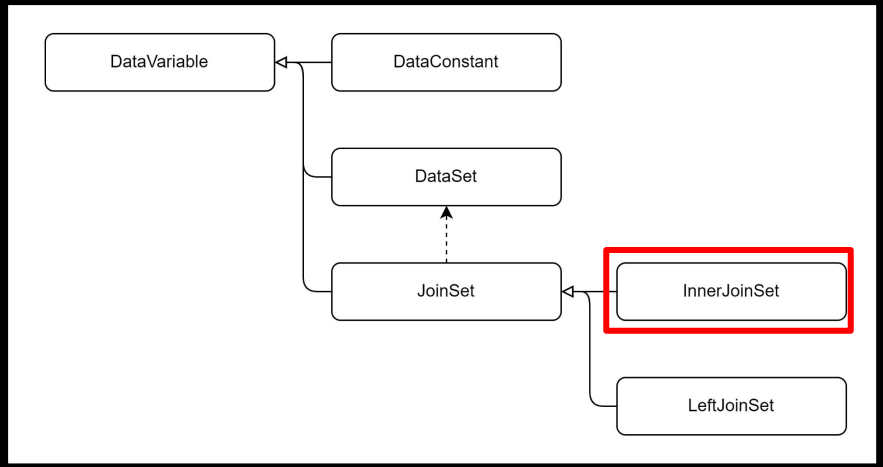


단순한 모델, CSV

DataSet

JoinSet

초기 모델



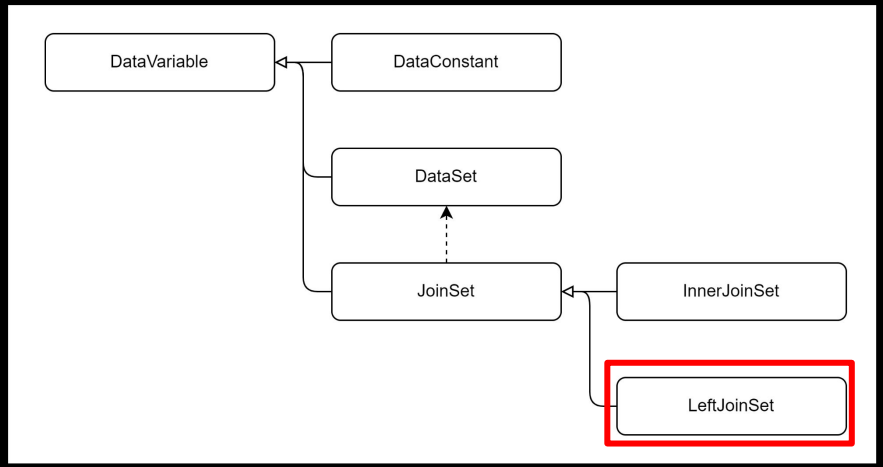
단순한 모델, CSV

DataSet

JoinSet

- InnerJoinSet

초기 모델



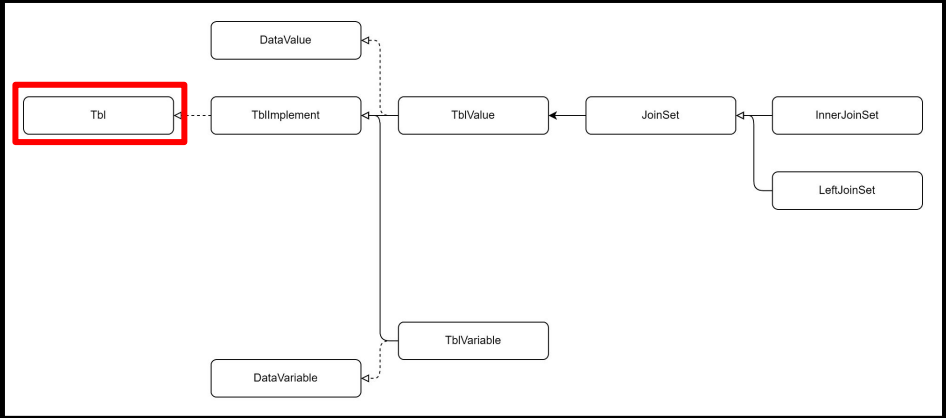
단순한 모델, CSV

DataSet

JoinSet

- LeftJoinSet

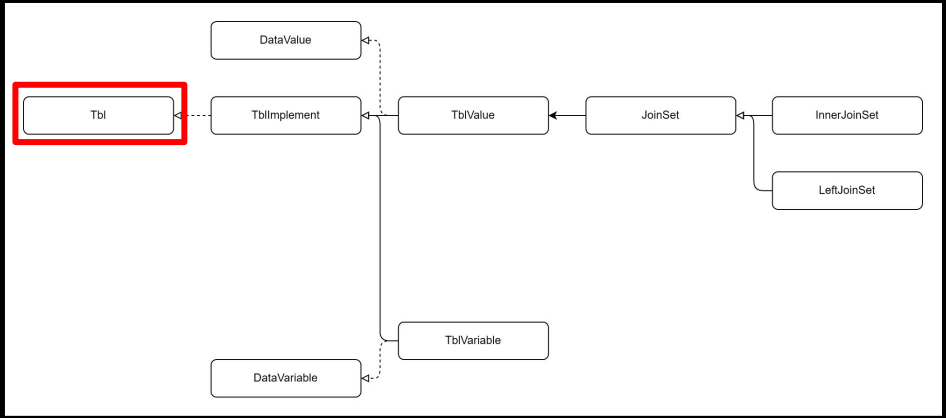
초기 모델



Tbl 등장

Table By List

초기 모델

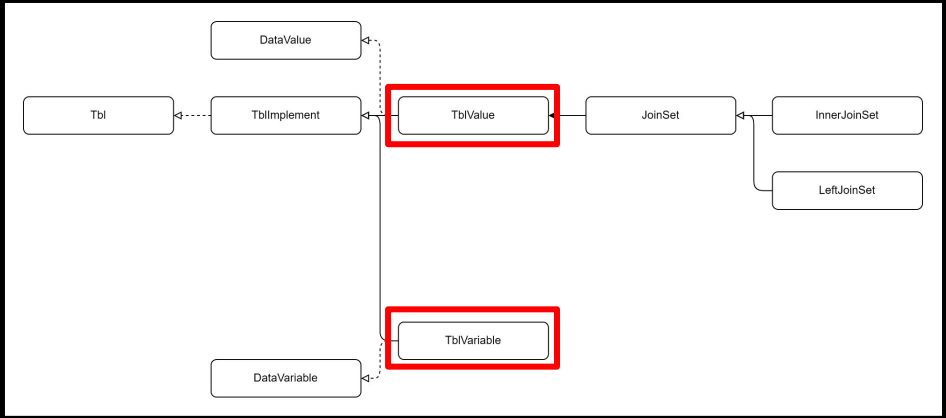


Tbl 등장

Table By List

- List<String> Column
- List<List<Object>> Data

초기 모델



Tbl 등장

Table By List

- List<String> Column
- List<List<Object>> Data

불변성 Value와 가변성 Variable

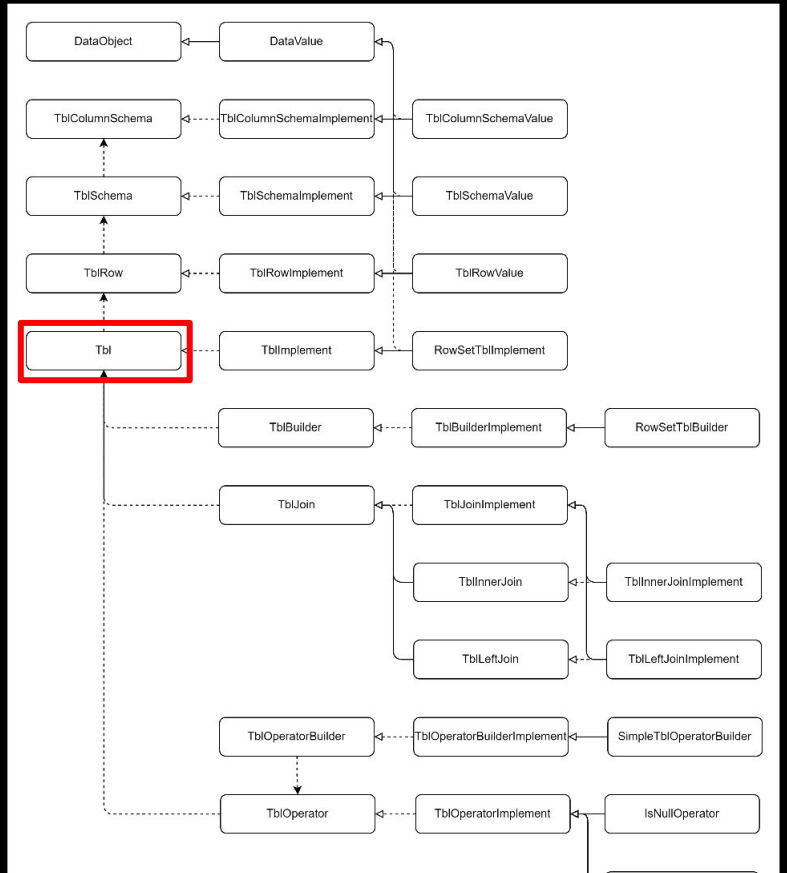
의문

가변성이 꼭 필요할까..?

의문

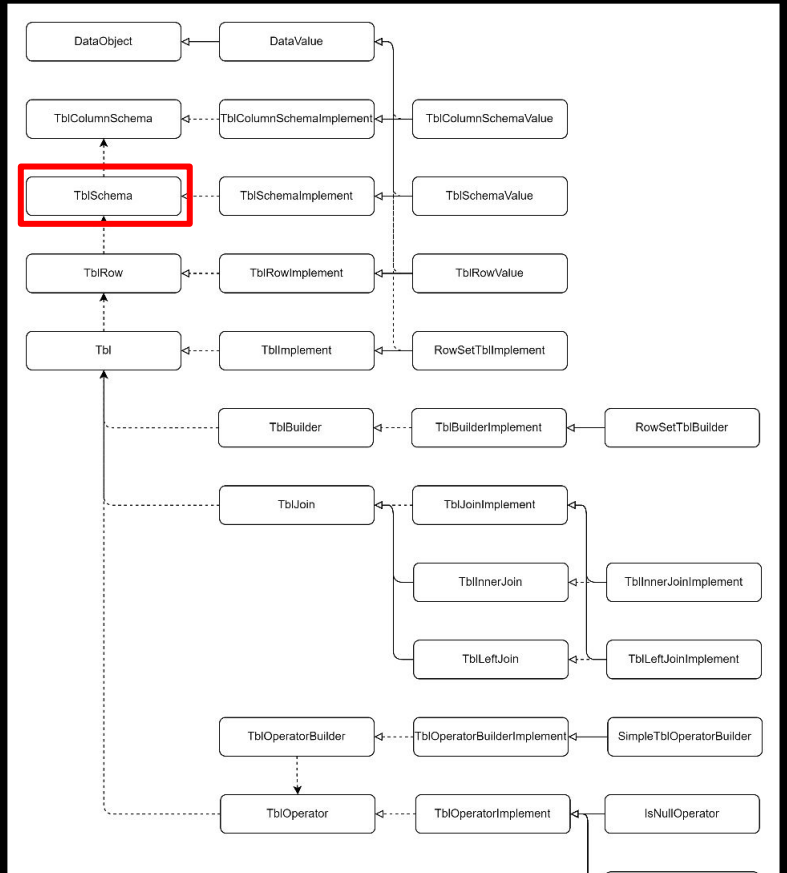
스키마가 필요하지 않는가..?

본론


DataObject
DataValue
TblColumnSchema
TblColumnSchemaImplement
TblColumnSchemaValue
TblSchema
TblSchemaImplement
TblSchemaValue
TblRow
TblRowImplement
TblRowValue
Tbl
TblImplement
RowSetTblImplement
TblBuilder
TblBuilderImplement
RowSetTblBuilder
TblJoin
TblJoinImplement
TblInnerJoin
TblInnerJoinImplement
TblLeftJoin
TblLeftJoinImplement
TblOperatorBuilder
TblOperatorBuilderImplement
SimpleTblOperatorBuilder
TblOperator
TblOperatorImplement
IsNullOperator

Tbl

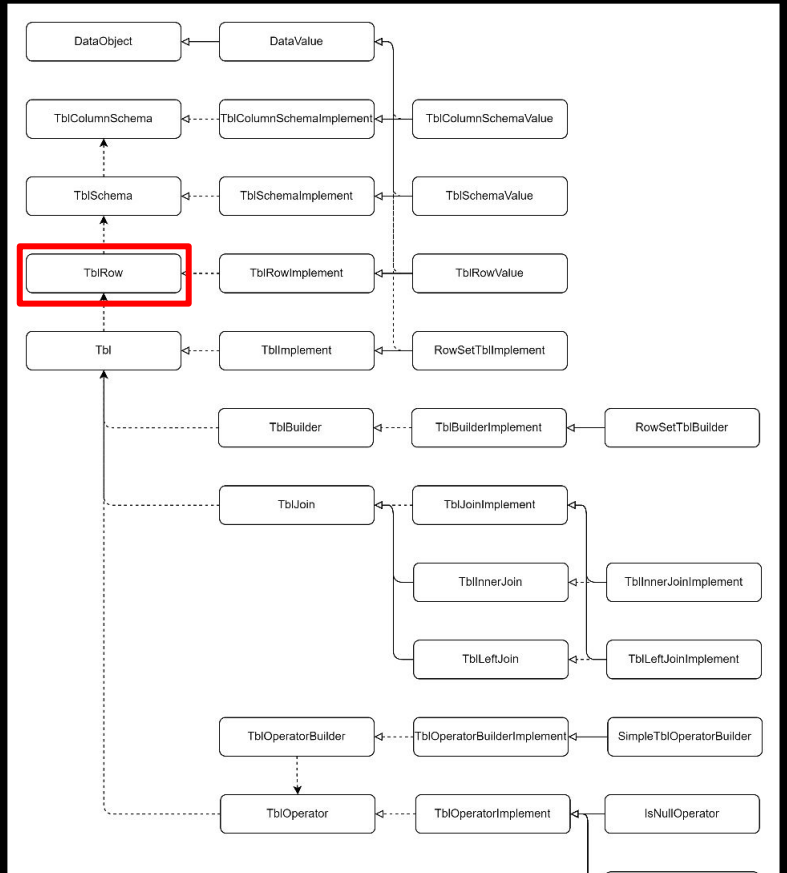
본론



Tbl

TblSchema

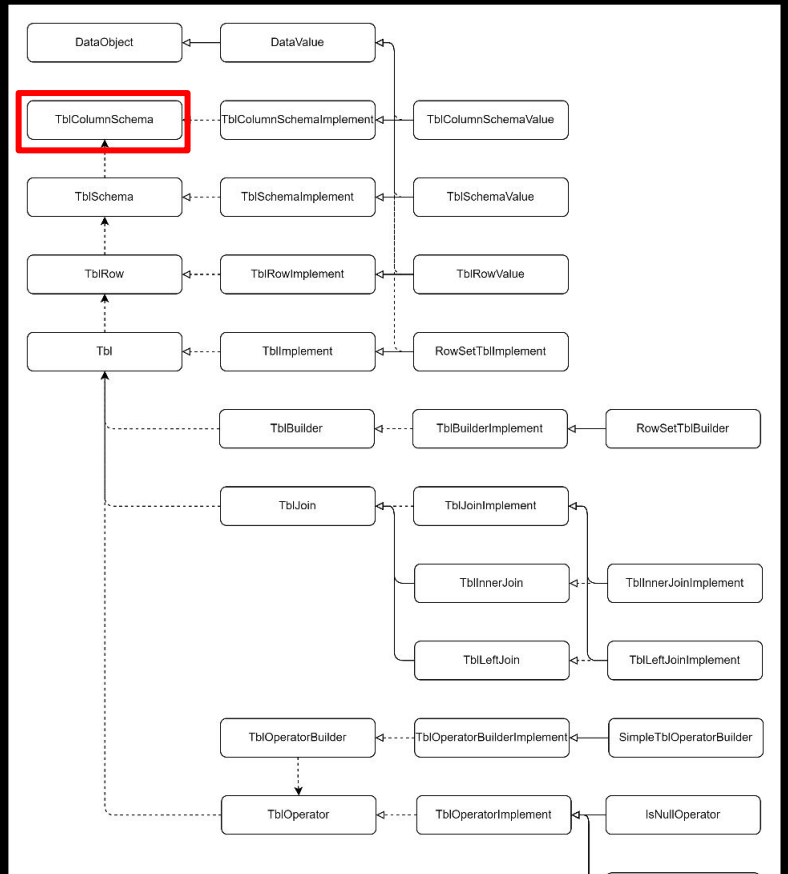
본론



Tbl
- TblRow

TblSchema

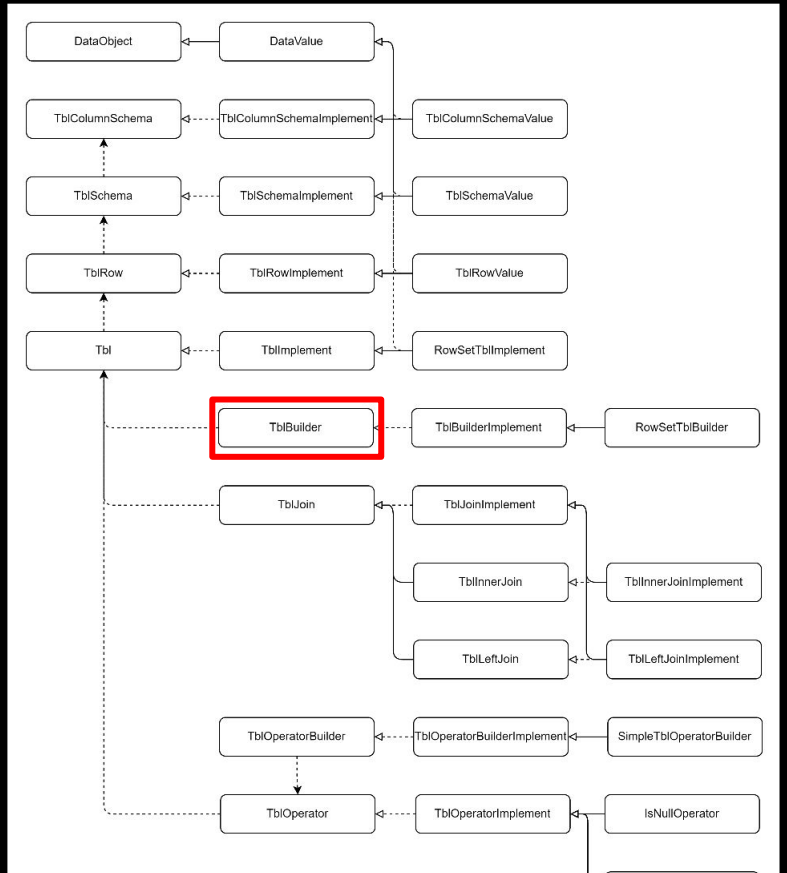
본론



Tbl
- TblRow

TblSchema
- TblColumnSchema

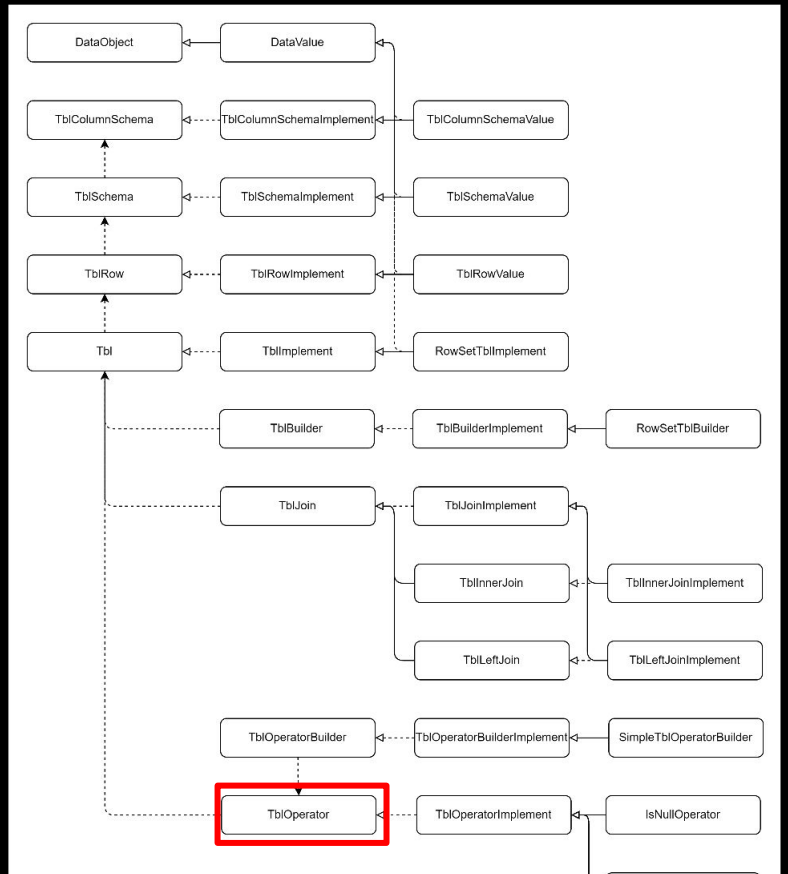
본론



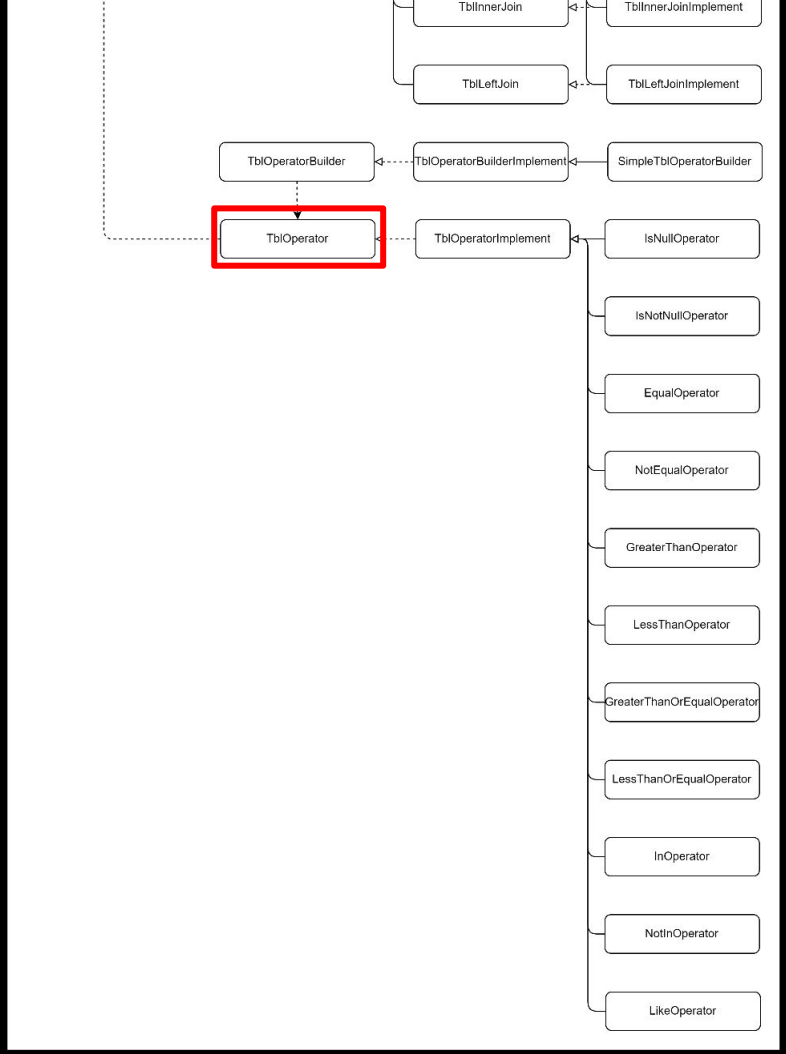
불변성을 유저가 사용하기 좋게

Builder 패턴으로

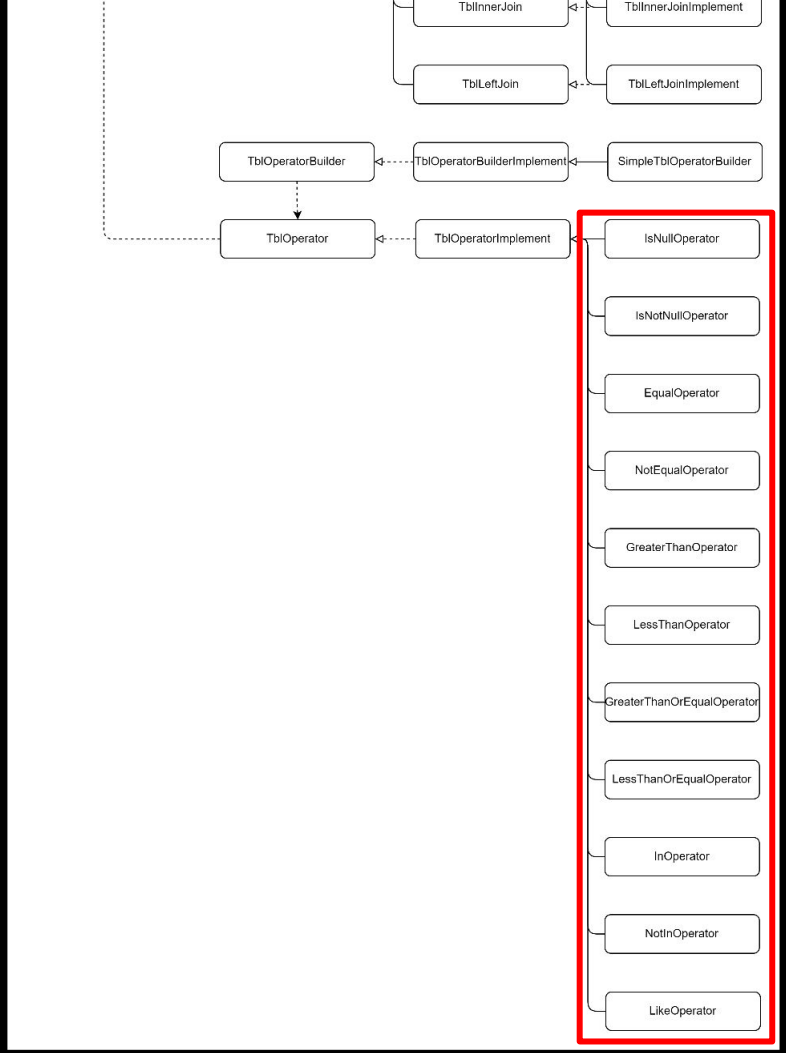
본론



Operator 테이블 연산 추가



Operator 테이블 연산 추가



Operator 테이블 연산 추가

다양한 구현체 추가

예제1 - Tbl 생성

```java
public interface Tbl extends DataObject {  3개 구현  lshh *

    static Tbl of(TblSchema schema, List<List<Object>> data) { return TblImplement.of(schema, data); }

    static Tbl of(List<Map<String, Object>> rowMapList) { return TblImplement.of(rowMapList); }

    static Tbl of(Map<String, List<Object>> columnListMap) { return TblImplement.of(columnListMap); }

    static TblBuilder builder(TblSchema schema) { return TblBuilder.forRowSet(schema); }

    int count();  5개 사용 위치  1개 구현  lshh
    TblRow getRow(int index);  6개 사용 위치  1개 구현  lshh
    List<TblRow> getRows();  1개 구현  lshh
    List<Object> getRawRow(int index);  5개 사용 위치  1개 구현  lshh
    List<List<Object>> getRawRows();  1개 구현  lshh
    TblColumnSchema<?> getColumnSchema(int index);  2개 사용 위치  1개 구현  lshh
    TblSchema getSchema();  1개 구현  lshh
    int getColumnSize();  3개 사용 위치  1개 구현  lshh
    String getColumn(int index);  0개의 사용위치  1개 구현  lshh
    List<String> getColumns();  3개 사용 위치  1개 구현  lshh
    Class<?> getColumnType(int index);  4개 사용 위치  1개 구현  lshh
    Map<String, Class<?>> getColumnTypes();  0개의 사용위치  1개 구현  lshh
    int getColumnIndex(String column);  1개 구현  lshh

    Tbl select(String... columns);  1개 사용 위치  1개 구현  lshh
    Tbl where(String column, TblOperatorType operator, Object value);  1개 구현  lshh
    Tbl where(String column, TblOperatorType operator);  1개 구현  lshh

    TblJoin leftJoin(Tbl right);  1개 사용 위치  1개 구현  lshh
    TblJoin innerJoin(Tbl right);  1개 사용 위치  1개 구현  lshh

    List<Map<String, Object>> toRowMapList();  0개의 사용위치  1개 구현  lshh
    Map<String, List<Object>> toColumnListMap();  2개 사용 위치  1개 구현  lshh

    Object findCell(String columnName, int rowIndex);  0개의 사용위치  1개 구현  lshh

    Object getCell(int columnIndex, int rowIndex);  1개 사용 위치  1개 구현  lshh

}
```

of

builder

예제1 - Tbl 생성

```java
public interface Tbl extends DataObject {  3개 구현  lshh *

    static Tbl of(TblSchema schema, List<List<Object>> data) { return TblImplement.of(schema, data); }

    static Tbl of(List<Map<String, Object>> rowMapList) { return TblImplement.of(rowMapList); }

    static Tbl of(Map<String, List<Object>> columnListMap) { return TblImplement.of(columnListMap); }

    static TblBuilder builder(TblSchema schema) { return TblBuilder.forRowSet(schema); }

    int count();  5개 사용 위치  1개 구현  lshh
    TblRow getRow(int index);  6개 사용 위치  1개 구현  lshh
    List<TblRow> getRows();  1개 구현  lshh
    List<Object> getRawRow(int index);  5개 사용 위치  1개 구현  lshh
    List<List<Object>> getRawRows();  1개 구현  lshh
    TblColumnSchema<?> getColumnSchema(int index);  2개 사용 위치  1개 구현  lshh
    TblSchema getSchema();  1개 구현  lshh
    int getColumnSize();  3개 사용 위치  1개 구현  lshh
    String getColumn(int index);  0개의 사용위치  1개 구현  lshh
    List<String> getColumns();  3개 사용 위치  1개 구현  lshh
    Class<?> getColumnType(int index);  4개 사용 위치  1개 구현  lshh
    Map<String, Class<?>> getColumnTypes();  0개의 사용위치  1개 구현  lshh
    int getColumnIndex(String column);  1개 구현  lshh

    Tbl select(String... columns);  1개 사용 위치  1개 구현  lshh
    Tbl where(String column, TblOperatorType operator, Object value);  1개 구현  lshh
    Tbl where(String column, TblOperatorType operator);  1개 구현  lshh

    TblJoin leftJoin(Tbl right);  1개 사용 위치  1개 구현  lshh
    TblJoin innerJoin(Tbl right);  1개 사용 위치  1개 구현  lshh

    List<Map<String, Object>> toRowMapList();  0개의 사용위치  1개 구현  lshh
    Map<String, List<Object>> toColumnListMap();  2개 사용 위치  1개 구현  lshh

    Object findCell(String columnName, int rowIndex);  0개의 사용위치  1개 구현  lshh

    Object getCell(int columnIndex, int rowIndex);  1개 사용 위치  1개 구현  lshh

}
```

of

builder

예제1 - Tbl 생성

```
public interface Tbl extends DataObject {  3개 구현  lshh *

    static Tbl of(TblSchema schema, List<List<Object>> data) { return TblImplement.of(schema, data); }

    static Tbl of(List<Map<String, Object>> rowMapList) { return TblImplement.of(rowMapList); }

    static Tbl of(Map<String, List<Object>> columnListMap) { return TblImplement.of(columnListMap); }

    static TblBuilder builder(TblSchema schema) { return TblBuilder.forRowSet(schema); }

    int count();  5개 사용 위치  1개 구현  lshh
    TblRow getRow(int index);  6개 사용 위치  1개 구현  lshh
    List<TblRow> getRows();  1개 구현  lshh
    List<Object> getRawRow(int index);  5개 사용 위치  1개 구현  lshh
    List<List<Object>> getRawRows();  1개 구현  lshh
    TblColumnSchema<?> getColumnSchema(int index);  2개 사용 위치  1개 구현  lshh
    TblSchema getSchema();  1개 구현  lshh
    int getColumnSize();  3개 사용 위치  1개 구현  lshh
    String getColumn(int index);  0개의 사용위치  1개 구현  lshh
    List<String> getColumns();  3개 사용 위치  1개 구현  lshh
    Class<?> getColumnType(int index);  4개 사용 위치  1개 구현  lshh
    Map<String, Class<?>> getColumnTypes();  0개의 사용위치  1개 구현  lshh
    int getColumnIndex(String column);  1개 구현  lshh

    Tbl select(String... columns);  1개 사용 위치  1개 구현  lshh
    Tbl where(String column, TblOperatorType operator, Object value);  1개 구현  lshh
    Tbl where(String column, TblOperatorType operator);  1개 구현  lshh

    TblJoin leftJoin(Tbl right);  1개 사용 위치  1개 구현  lshh
    TblJoin innerJoin(Tbl right);  1개 사용 위치  1개 구현  lshh

    List<Map<String, Object>> toRowMapList();  0개의 사용위치  1개 구현  lshh
    Map<String, List<Object>> toColumnListMap();  2개 사용 위치  1개 구현  lshh

    Object findCell(String columnName, int rowIndex);  0개의 사용위치  1개 구현  lshh

    Object getCell(int columnIndex, int rowIndex);  1개 사용 위치  1개 구현  lshh

}
```

of

builder

예제1 - Tbl 생성

```
public class TbTest {  lshh *
    @Nested  lshh *
    class ConstructorTest{...}

    @Nested  신규 *
    class ToColumnListMapTest{...}

    @Nested  신규 *
    class ToRowMapListTest{...}

    @Nested  신규 *
    class SelectTest {...}

    @Nested  신규 *
    class WhereTest{...}
}
```

TblTest

예제1 - Tbl 생성

```
public class TbTest {  lshh *
    @Nested  lshh *
    class ConstructorTest{...}

    @Nested  신규 *
    class ToColumnListMapTest{...}

    @Nested  신규 *
    class ToRowMapListTest{...}

    @Nested  신규 *
    class SelectTest {...}

    @Nested  신규 *
    class WhereTest{...}
}
```

TblTest
- ConstructorTest

예제1 - Tbl 생성

```java
public class TbTest {  lshh *
    @Nested  lshh *
    class ConstructorTest{
        @Test  lshh
        @DisplayName("rowMapList 를 이용한 Tbl 객체 생성")
        public void testOfMethodWithRowMapList() {...}

        @Test  lshh
        @DisplayName("columnListMap 을 이용한 Tbl 객체 생성")
        public void testOfMethodWithColumnListMap() {...}

        @Test  신규 *
        @DisplayName("schema 와 data 를 이용한 Tbl 객체 생성")
        public void testOfMethodWithSchemaAndData() {...}

        @Test  신규 *
        @DisplayName("TblBuilder 를 이용한 Tbl 객체 생성")
        public void testBuilderMethod() {...}
    }
```

TblTest
- ConstructorTest

예제1 - Tbl 생성

```java
public class TbTest {  lshh *
    @Nested  lshh *
    class ConstructorTest{
        @Test  lshh
        @DisplayName("rowMapList 를 이용한 Tbl 객체 생성")
        public void testOfMethodWithRowMapList() {...}

        @Test  lshh
        @DisplayName("columnListMap 을 이용한 Tbl 객체 생성")
        public void testOfMethodWithColumnListMap() {...}

        @Test  신규 *
        @DisplayName("schema 와 data 를 이용한 Tbl 객체 생성")
        public void testOfMethodWithSchemaAndData() {...}

        @Test  신규 *
        @DisplayName("TblBuilder 를 이용한 Tbl 객체 생성")
        public void testBuilderMethod() {...}
    }
```

TblTest
- ConstructorTest

 - rowMapList를 이용한 Tbl 생성

예제1 - Tbl 생성

```java
@Nested  lshh *
class ConstructorTest{
    @Test  lshh
    @DisplayName("rowMapList 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithRowMapList() {
        List<Map<String, Object>> rowMapList = new ArrayList<>();
        Map<String, Object> rowMap = new HashMap<>();
        rowMap.put("key1", "value1");
        rowMap.put("key2", 2);
        rowMapList.add(rowMap);

        Tbl result = TblImplement.of(rowMapList);

        System.out.println(result);
        assertNotNull(result);
        assertEquals( expected: 2, result.getColumnSize());
        assertEquals(String.class, result.getColumnType( index: 0));
        assertEquals(Integer.class, result.getColumnType( index: 1));
    }

    @Test  lshh
    @DisplayName("columnListMap 을 이용한 Tbl 객체 생성")
    public void testOfMethodWithColumnListMap() {...}

    @Test  신규 *
    @DisplayName("schema 와 data 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithSchemaAndData() {...}

    @Test  신규 *
    @DisplayName("TblBuilder 를 이용한 Tbl 객체 생성")
    public void testBuilderMethod() {...}
}
```

TblTest
- ConstructorTest

  - rowMapList를 이용한 Tbl 생성

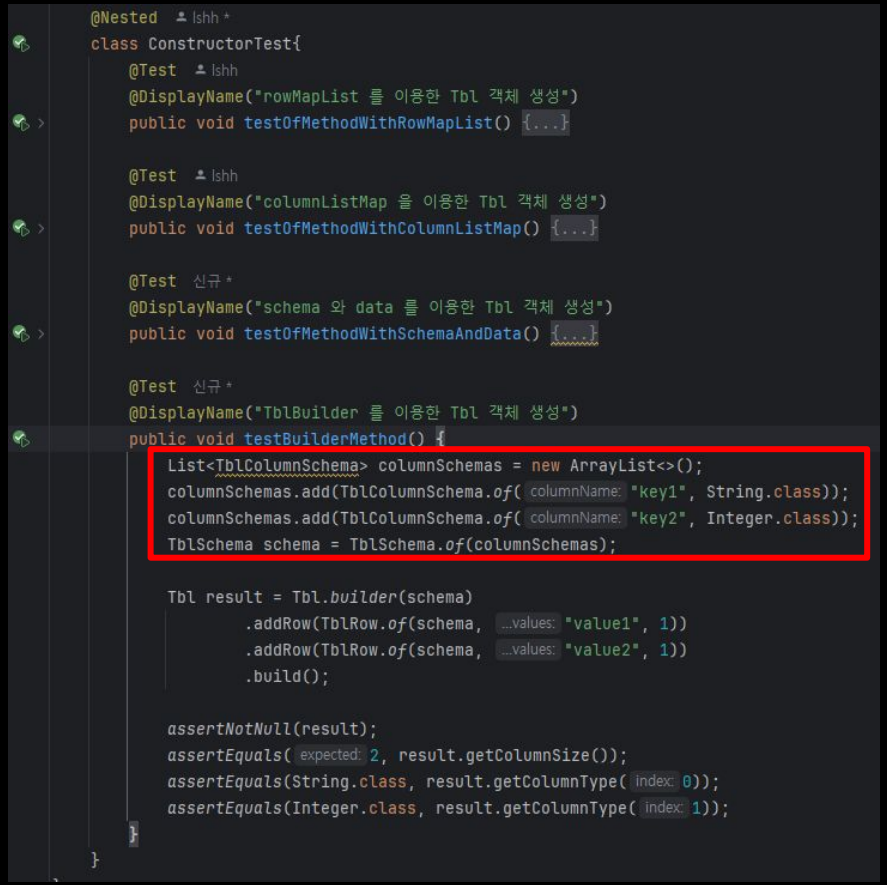
예제1 - Tbl 생성

```java
public class TbTest {  lshh *
    @Nested  lshh *
    class ConstructorTest{
        @Test  lshh
        @DisplayName("rowMapList 를 이용한 Tbl 객체 생성")
        public void testOfMethodWithRowMapList() {...}

        @Test  lshh
        @DisplayName("columnListMap 을 이용한 Tbl 객체 생성")
        public void testOfMethodWithColumnListMap() {...}

        @Test  신규 *
        @DisplayName("schema 와 data 를 이용한 Tbl 객체 생성")
        public void testOfMethodWithSchemaAndData() {...}

        @Test  신규 *
        @DisplayName("TblBuilder 를 이용한 Tbl 객체 생성")
        public void testBuilderMethod() {...}
    }
```

TblTest
- ConstructorTest

 - builder를 이용한 Tbl 생성

예제1 - Tbl 생성



```
@Nested  lshh *
class ConstructorTest{
    @Test  lshh
    @DisplayName("rowMapList 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithRowMapList() {...}

    @Test  lshh
    @DisplayName("columnListMap 을 이용한 Tbl 객체 생성")
    public void testOfMethodWithColumnListMap() {...}

    @Test  신규 *
    @DisplayName("schema 와 data 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithSchemaAndData() {...}

    @Test  신규 *
    @DisplayName("TblBuilder 를 이용한 Tbl 객체 생성")
    public void testBuilderMethod() {
        List<TblColumnSchema> columnSchemas = new ArrayList<>();
        columnSchemas.add(TblColumnSchema.of( columnName: "key1", String.class));
        columnSchemas.add(TblColumnSchema.of( columnName: "key2", Integer.class));
        TblSchema schema = TblSchema.of(columnSchemas);

        Tbl result = Tbl.builder(schema)
                .addRow(TblRow.of(schema,  ...values: "value1", 1))
                .addRow(TblRow.of(schema,  ...values: "value2", 1))
                .build();

        assertNotNull(result);
        assertEquals( expected: 2, result.getColumnSize());
        assertEquals(String.class, result.getColumnType( index: 0));
        assertEquals(Integer.class, result.getColumnType( index: 1));
    }
}
```

TblTest
- ConstructorTest

  - builder를 이용한 Tbl 생성

예제1 - Tbl 생성

```java
@Nested  lshh *
class ConstructorTest{
    @Test  lshh
    @DisplayName("rowMapList 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithRowMapList() {...}

    @Test  lshh
    @DisplayName("columnListMap 을 이용한 Tbl 객체 생성")
    public void testOfMethodWithColumnListMap() {...}

    @Test  신규 *
    @DisplayName("schema 와 data 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithSchemaAndData() {...}

    @Test  신규 *
    @DisplayName("TblBuilder 를 이용한 Tbl 객체 생성")
    public void testBuilderMethod() {
        List<TblColumnSchema> columnSchemas = new ArrayList<>();
        columnSchemas.add(TblColumnSchema.of( columnName: "key1", String.class));
        columnSchemas.add(TblColumnSchema.of( columnName: "key2", Integer.class));
        TblSchema schema = TblSchema.of(columnSchemas);

        Tbl result = Tbl.builder(schema)
                .addRow(TblRow.of(schema,  ...values: "value1", 1))
                .addRow(TblRow.of(schema,  ...values: "value2", 1))
                .build();

        assertNotNull(result);
        assertEquals( expected: 2, result.getColumnSize());
        assertEquals(String.class, result.getColumnType( index: 0));
        assertEquals(Integer.class, result.getColumnType( index: 1));
    }
}
```

TblTest
- ConstructorTest

  - builder를 이용한 Tbl 생성

예제1 - Tbl 생성

```java
@Nested  lshh *
class ConstructorTest{
    @Test  lshh
    @DisplayName("rowMapList 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithRowMapList() {...}

    @Test  lshh
    @DisplayName("columnListMap 을 이용한 Tbl 객체 생성")
    public void testOfMethodWithColumnListMap() {...}

    @Test  신규 *
    @DisplayName("schema 와 data 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithSchemaAndData() {...}

    @Test  신규 *
    @DisplayName("TblBuilder 를 이용한 Tbl 객체 생성")
    public void testBuilderMethod() {
        List<TblColumnSchema> columnSchemas = new ArrayList<>();
        columnSchemas.add(TblColumnSchema.of( columnName: "key1", String.class));
        columnSchemas.add(TblColumnSchema.of( columnName: "key2", Integer.class));
        TblSchema schema = TblSchema.of(columnSchemas);

        Tbl result = Tbl.builder(schema)
                .addRow(TblRow.of(schema,  ...values: "value1", 1))
                .addRow(TblRow.of(schema,  ...values: "value2", 1))
                .build();

        assertNotNull(result);
        assertEquals( expected: 2, result.getColumnSize());
        assertEquals(String.class, result.getColumnType( index: 0));
        assertEquals(Integer.class, result.getColumnType( index: 1));
    }
}
```

TblTest
- ConstructorTest

  - builder를 이용한 Tbl 생성

예제1 - Tbl 생성

```java
@Nested  lshh *
class ConstructorTest{
    @Test  lshh
    @DisplayName("rowMapList 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithRowMapList() {...}

    @Test  lshh
    @DisplayName("columnListMap 을 이용한 Tbl 객체 생성")
    public void testOfMethodWithColumnListMap() {...}

    @Test  신규 *
    @DisplayName("schema 와 data 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithSchemaAndData() {...}

    @Test  신규 *
    @DisplayName("TblBuilder 를 이용한 Tbl 객체 생성")
    public void testBuilderMethod() {
        List<TblColumnSchema> columnSchemas = new ArrayList<>();
        columnSchemas.add(TblColumnSchema.of( columnName: "key1", String.class));
        columnSchemas.add(TblColumnSchema.of( columnName: "key2", Integer.class));
        TblSchema schema = TblSchema.of(columnSchemas);

        Tbl result = Tbl.builder(schema)
                .addRow(TblRow.of(schema,  ...values: "value1", 1))
                .addRow(TblRow.of(schema,  ...values: "value2", 1))
                .build();

        assertNotNull(result);
        assertEquals( expected: 2, result.getColumnSize());
        assertEquals(String.class, result.getColumnType( index: 0));
        assertEquals(Integer.class, result.getColumnType( index: 1));
    }
}
```

TblTest
- ConstructorTest

  - builder를 이용한 Tbl 생성

예제1 - Tbl 생성

```java
@Nested  lshh *
class ConstructorTest{
    @Test  lshh
    @DisplayName("rowMapList 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithRowMapList() {...}

    @Test  lshh
    @DisplayName("columnListMap 을 이용한 Tbl 객체 생성")
    public void testOfMethodWithColumnListMap() {...}

    @Test  신규 *
    @DisplayName("schema 와 data 를 이용한 Tbl 객체 생성")
    public void testOfMethodWithSchemaAndData() {...}

    @Test  신규 *
    @DisplayName("TblBuilder 를 이용한 Tbl 객체 생성")
    public void testBuilderMethod() {
        List<TblColumnSchema> columnSchemas = new ArrayList<>();
        columnSchemas.add(TblColumnSchema.of( columnName: "key1", String.class));
        columnSchemas.add(TblColumnSchema.of( columnName: "key2", Integer.class));
        TblSchema schema = TblSchema.of(columnSchemas);

        Tbl result = Tbl.builder(schema)
                .addRow(TblRow.of(schema,  ...values: "value1", 1))
                .addRow(TblRow.of(schema,  ...values: "value2", 1))
                .build();

        assertNotNull(result);
        assertEquals( expected: 2, result.getColumnSize());
        assertEquals(String.class, result.getColumnType( index: 0));
        assertEquals(Integer.class, result.getColumnType( index: 1));
    }
}
```

TblTest
- ConstructorTest

  - builder를 이용한 Tbl 생성

예제2 - Select

TblTest
- SelectTest

예제2 - Select

```java
@Nested  lshh
class SelectTest {
    @Test  lshh
    @DisplayName("컬럼을 Select시, 해당 컬럼들로 이루어진 Tbl 객체를 반환한다.")
    void testSelectMethod() {
        List<TblColumnSchema> schemas = List.of(
                TblColumnSchema.of( columnName: "column1", String.class),
                TblColumnSchema.of( columnName: "column2", String.class),
                TblColumnSchema.of( columnName: "column3", String.class)
        );
        List<List<Object>> inputData = List.of(
                List.of("A", "B", "C"),
                List.of("D", "E", "F"),
                List.of("G", "H", "I")
        );
        TblSchema schema = TblSchema.of(schemas);
        Tbl tblInstance = Tbl.of(schema, inputData);

        // Select specific columns
        Tbl resultTbl = tblInstance.select( ...columns: "column1", "column2");

        // Assertions
        assertEquals( expected: 2, resultTbl.getSchema().getColumnSize());
        assertEquals( expected: 3, resultTbl.count());

        List<String> expectedColumns = Arrays.asList("column1", "column2");
        assertEquals(expectedColumns, resultTbl.getColumns());

        String expectedFirstRowFirstColumn = "A";
        assertEquals(expectedFirstRowFirstColumn, resultTbl.getRow( index: 0).findCell( columnName: "column1").orElseThrow());

        String expectedFirstRowSecondColumn = "B";
        assertEquals(expectedFirstRowSecondColumn, resultTbl.getRow( index: 0).findCell( columnName: "column2").orElseThrow());
    }
}
```

testSelectMethod

예제2 - Select

```java
@Nested  lshh
class SelectTest {
    @Test  lshh
    @DisplayName("컬럼을 Select시, 해당 컬럼들로 이루어진 Tbl 객체를 반환한다.")
    void testSelectMethod() {
        List<TblColumnSchema> schemas = List.of(
                TblColumnSchema.of( columnName: "column1", String.class),
                TblColumnSchema.of( columnName: "column2", String.class),
                TblColumnSchema.of( columnName: "column3", String.class)
        );
        List<List<Object>> inputData = List.of(
                List.of("A", "B", "C"),
                List.of("D", "E", "F"),
                List.of("G", "H", "I")
        );
        TblSchema schema = TblSchema.of(schemas);
        Tbl tblInstance = Tbl.of(schema, inputData);

        // Select specific columns
        Tbl resultTbl = tblInstance.select( ...columns: "column1", "column2");

        // Assertions
        assertEquals( expected: 2, resultTbl.getSchema().getColumnSize());
        assertEquals( expected: 3, resultTbl.count());

        List<String> expectedColumns = Arrays.asList("column1", "column2");
        assertEquals(expectedColumns, resultTbl.getColumns());

        String expectedFirstRowFirstColumn = "A";
        assertEquals(expectedFirstRowFirstColumn, resultTbl.getRow( index: 0).findCell( columnName: "column1").orElseThrow());

        String expectedFirstRowSecondColumn = "B";
        assertEquals(expectedFirstRowSecondColumn, resultTbl.getRow( index: 0).findCell( columnName: "column2").orElseThrow());
    }
}
```

테이블을 준비

예제2 - Select

```java
@Nested  lshh
class SelectTest {
    @Test  lshh
    @DisplayName("컬럼을 Select시, 해당 컬럼들로 이루어진 Tbl 객체를 반환한다.")
    void testSelectMethod() {
        List<TblColumnSchema> schemas = List.of(
                TblColumnSchema.of( columnName: "column1", String.class),
                TblColumnSchema.of( columnName: "column2", String.class),
                TblColumnSchema.of( columnName: "column3", String.class)
        );
        List<List<Object>> inputData = List.of(
                List.of("A", "B", "C"),
                List.of("D", "E", "F"),
                List.of("G", "H", "I")
        );
        TblSchema schema = TblSchema.of(schemas);
        Tbl tblInstance = Tbl.of(schema, inputData);

        // Select specific columns
        Tbl resultTbl = tblInstance.select( ...columns: "column1", "column2");

        // Assertions
        assertEquals( expected: 2, resultTbl.getSchema().getColumnSize());
        assertEquals( expected: 3, resultTbl.count());

        List<String> expectedColumns = Arrays.asList("column1", "column2");
        assertEquals(expectedColumns, resultTbl.getColumns());

        String expectedFirstRowFirstColumn = "A";
        assertEquals(expectedFirstRowFirstColumn, resultTbl.getRow( index: 0).findCell( columnName: "column1").orElseThrow());

        String expectedFirstRowSecondColumn = "B";
        assertEquals(expectedFirstRowSecondColumn, resultTbl.getRow( index: 0).findCell( columnName: "column2").orElseThrow());
    }
}
```

테이블을 준비

select

예제2 - Select

```java
@Nested  lshh
class SelectTest {
    @Test  lshh
    @DisplayName("컬럼을 Select시, 해당 컬럼들로 이루어진 Tbl 객체를 반환한다.")
    void testSelectMethod() {
        List<TblColumnSchema> schemas = List.of(
                TblColumnSchema.of( columnName: "column1", String.class),
                TblColumnSchema.of( columnName: "column2", String.class),
                TblColumnSchema.of( columnName: "column3", String.class)
        );
        List<List<Object>> inputData = List.of(
                List.of("A", "B", "C"),
                List.of("D", "E", "F"),
                List.of("G", "H", "I")
        );
        TblSchema schema = TblSchema.of(schemas);
        Tbl tblInstance = Tbl.of(schema, inputData);

        // Select specific columns
        Tbl resultTbl = tblInstance.select( ...columns: "column1", "column2");

        // Assertions
        assertEquals( expected: 2, resultTbl.getSchema().getColumnSize());
        assertEquals( expected: 3, resultTbl.count());

        List<String> expectedColumns = Arrays.asList("column1", "column2");
        assertEquals(expectedColumns, resultTbl.getColumns());

        String expectedFirstRowFirstColumn = "A";
        assertEquals(expectedFirstRowFirstColumn, resultTbl.getRow( index: 0).findCell( columnName: "column1").orElseThrow());

        String expectedFirstRowSecondColumn = "B";
        assertEquals(expectedFirstRowSecondColumn, resultTbl.getRow( index: 0).findCell( columnName: "column2").orElseThrow());
    }
}
```

테이블을 준비

select

- 불변성 => 해당 컬럼들로 재생성

예제2 - Select

```java
@Nested  lshh
class SelectTest {
    @Test  lshh
    @DisplayName("컬럼을 Select시, 해당 컬럼들로 이루어진 Tbl 객체를 반환한다.")
    void testSelectMethod() {
        List<TblColumnSchema> schemas = List.of(
                TblColumnSchema.of( columnName: "column1", String.class),
                TblColumnSchema.of( columnName: "column2", String.class),
                TblColumnSchema.of( columnName: "column3", String.class)
        );
        List<List<Object>> inputData = List.of(
                List.of("A", "B", "C"),
                List.of("D", "E", "F"),
                List.of("G", "H", "I")
        );
        TblSchema schema = TblSchema.of(schemas);
        Tbl tblInstance = Tbl.of(schema, inputData);

        // Select specific columns
        Tbl resultTbl = tblInstance.select( ...columns: "column1", "column2");

        // Assertions
        assertEquals( expected: 2, resultTbl.getSchema().getColumnSize());
        assertEquals( expected: 3, resultTbl.count());

        List<String> expectedColumns = Arrays.asList("column1", "column2");
        assertEquals(expectedColumns, resultTbl.getColumns());

        String expectedFirstRowFirstColumn = "A";
        assertEquals(expectedFirstRowFirstColumn, resultTbl.getRow( index: 0).findCell( columnName: "column1").orElseThrow());

        String expectedFirstRowSecondColumn = "B";
        assertEquals(expectedFirstRowSecondColumn, resultTbl.getRow( index: 0).findCell( columnName: "column2").orElseThrow());
    }
}
```

테이블을 준비

select

- 불변성 => 해당 컬럼들로 재생성

예제3 - Where

TblTest
- WhereTest

예제3 - Where

```java
@Nested  lshh
class WhereTest{
    @Test  lshh
    @DisplayName("컬럼 이름과 값으로 필터링")
    void testWhereMethod() {
        List<TblColumnSchema> schemas = List.of(
                TblColumnSchema.of( columnName: "column1", String.class),
                TblColumnSchema.of( columnName: "column2", String.class),
                TblColumnSchema.of( columnName: "column3", String.class)
        );
        List<List<Object>> inputData = List.of(
                List.of("A", "B", "C"),
                List.of("D", "E", "F"),
                List.of("G", "H", "I")
        );
        TblSchema schema = TblSchema.of(schemas);
        Tbl tblInstance = Tbl.of(schema, inputData);

        // Where
        Tbl resultTbl = tblInstance.where( column: "column1", TblOperatorType.EQUAL, value: "A");

        // Assertions
        assertEquals( expected: 1, resultTbl.count());
        assertEquals( expected: 3, resultTbl.getSchema().getColumnSize());
    }
}
```

testWhereMethod

예제3 - Where

```java
@Nested  lshh
class WhereTest{
    @Test  lshh
    @DisplayName("컬럼 이름과 값으로 필터링")
    void testWhereMethod() {
        List<TblColumnSchema> schemas = List.of(
                TblColumnSchema.of( columnName: "column1", String.class),
                TblColumnSchema.of( columnName: "column2", String.class),
                TblColumnSchema.of( columnName: "column3", String.class)
        );
        List<List<Object>> inputData = List.of(
                List.of("A", "B", "C"),
                List.of("D", "E", "F"),
                List.of("G", "H", "I")
        );
        TblSchema schema = TblSchema.of(schemas);
        Tbl tblInstance = Tbl.of(schema, inputData);

        // Where
        Tbl resultTbl = tblInstance.where( column: "column1", TblOperatorType.EQUAL, value: "A");

        // Assertions
        assertEquals( expected: 1, resultTbl.count());
        assertEquals( expected: 3, resultTbl.getSchema().getColumnSize());
    }
}
```

테이블을 준비

예제3 - Where

```java
@Nested  lshh
class WhereTest{
    @Test  lshh
    @DisplayName("컬럼 이름과 값으로 필터링")
    void testWhereMethod() {
        List<TblColumnSchema> schemas = List.of(
                TblColumnSchema.of( columnName: "column1", String.class),
                TblColumnSchema.of( columnName: "column2", String.class),
                TblColumnSchema.of( columnName: "column3", String.class)
        );
        List<List<Object>> inputData = List.of(
                List.of("A", "B", "C"),
                List.of("D", "E", "F"),
                List.of("G", "H", "I")
        );
        TblSchema schema = TblSchema.of(schemas);
        Tbl tblInstance = Tbl.of(schema, inputData);

        // Where
        Tbl resultTbl = tblInstance.where( column: "column1", TblOperatorType.EQUAL, value: "A");

        // Assertions
        assertEquals( expected: 1, resultTbl.count());
        assertEquals( expected: 3, resultTbl.getSchema().getColumnSize());
    }
}
```

테이블을 준비

where

예제3 - Where

```java
@Nested  lshh
class WhereTest{
    @Test  lshh
    @DisplayName("컬럼 이름과 값으로 필터링")
    void testWhereMethod() {
        List<TblColumnSchema> schemas = List.of(
                TblColumnSchema.of( columnName: "column1", String.class),
                TblColumnSchema.of( columnName: "column2", String.class),
                TblColumnSchema.of( columnName: "column3", String.class)
        );
        List<List<Object>> inputData = List.of(
                List.of("A", "B", "C"),
                List.of("D", "E", "F"),
                List.of("G", "H", "I")
        );
        TblSchema schema = TblSchema.of(schemas);
        Tbl tblInstance = Tbl.of(schema, inputData);

        // Where
        Tbl resultTbl = tblInstance.where( column: "column1", TblOperatorType.EQUAL, value: "A");

        // Assertions
        assertEquals( expected: 1, resultTbl.count());
        assertEquals( expected: 3, resultTbl.getSchema().getColumnSize());
    }
}
```

테이블을 준비

where - EQUAL

- 불변성 => 해당 컬럼들로 재생성

예제3 - Where

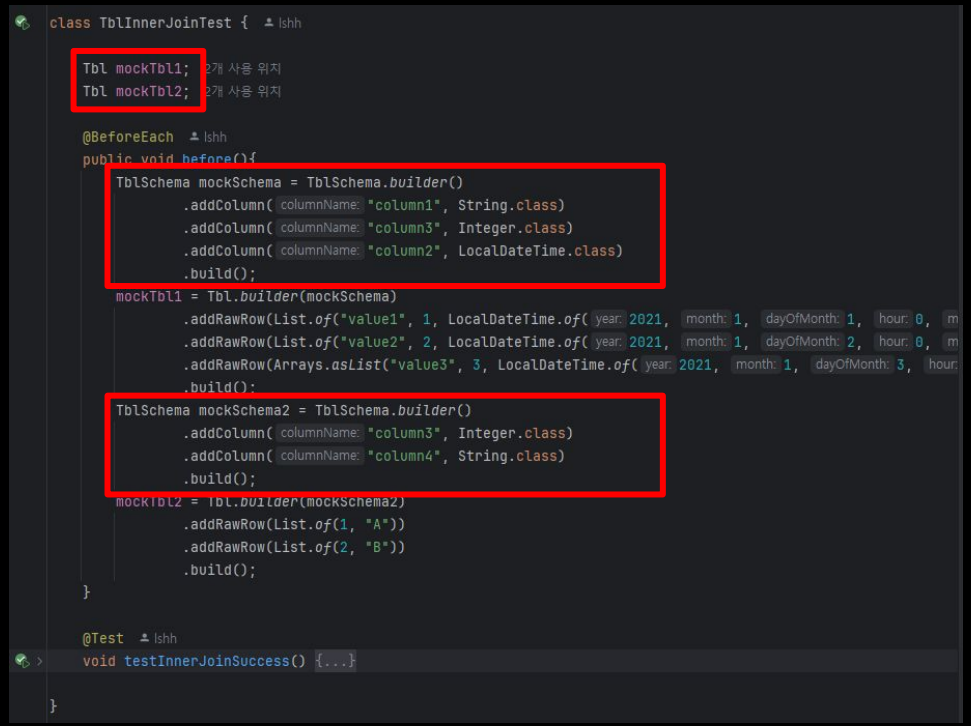
```java
@Nested  lshh
class WhereTest{
    @Test  lshh
    @DisplayName("컬럼 이름과 값으로 필터링")
    void testWhereMethod() {
        List<TblColumnSchema> schemas = List.of(
                TblColumnSchema.of( columnName: "column1", String.class),
                TblColumnSchema.of( columnName: "column2", String.class),
                TblColumnSchema.of( columnName: "column3", String.class)
        );
        List<List<Object>> inputData = List.of(
                List.of("A", "B", "C"),
                List.of("D", "E", "F"),
                List.of("G", "H", "I")
        );
        TblSchema schema = TblSchema.of(schemas);
        Tbl tblInstance = Tbl.of(schema, inputData);

        // Where
        Tbl resultTbl = tblInstance.where( column: "column1", TblOperatorType.EQUAL, value: "A");

        // Assertions
        assertEquals( expected: 1, resultTbl.count());
        assertEquals( expected: 3, resultTbl.getSchema().getColumnSize());
    }
}
```

테이블을 준비

where - EQUAL

- 불변성 => 해당 컬럼들로 재생성

예제4 - Inner Join



```
class TblInnerJoinTest {  lshh

    Tbl mockTbl1;  2개 사용 위치
    Tbl mockTbl2;  2개 사용 위치

    @BeforeEach  lshh
    public void before(){
        TblSchema mockSchema = TblSchema.builder()
                .addColumn( columnName: "column1", String.class)
                .addColumn( columnName: "column3", Integer.class)
                .addColumn( columnName: "column2", LocalDateTime.class)
                .build();
        mockTbl1 = Tbl.builder(mockSchema)
                .addRawRow(List.of("value1", 1, LocalDateTime.of( year: 2021,  month: 1,  dayOfMonth: 1,  hour: 0,  m
                .addRawRow(List.of("value2", 2, LocalDateTime.of( year: 2021,  month: 1,  dayOfMonth: 2,  hour: 0,  m
                .addRawRow(Arrays.asList("value3", 3, LocalDateTime.of( year: 2021,  month: 1,  dayOfMonth: 3,  hour:
                .build();
        TblSchema mockSchema2 = TblSchema.builder()
                .addColumn( columnName: "column3", Integer.class)
                .addColumn( columnName: "column4", String.class)
                .build();
        mockTbl2 = Tbl.builder(mockSchema2)
                .addRawRow(List.of(1, "A"))
                .addRawRow(List.of(2, "B"))
                .build();
    }

    @Test  lshh
    void testInnerJoinSuccess() {...}

}
```

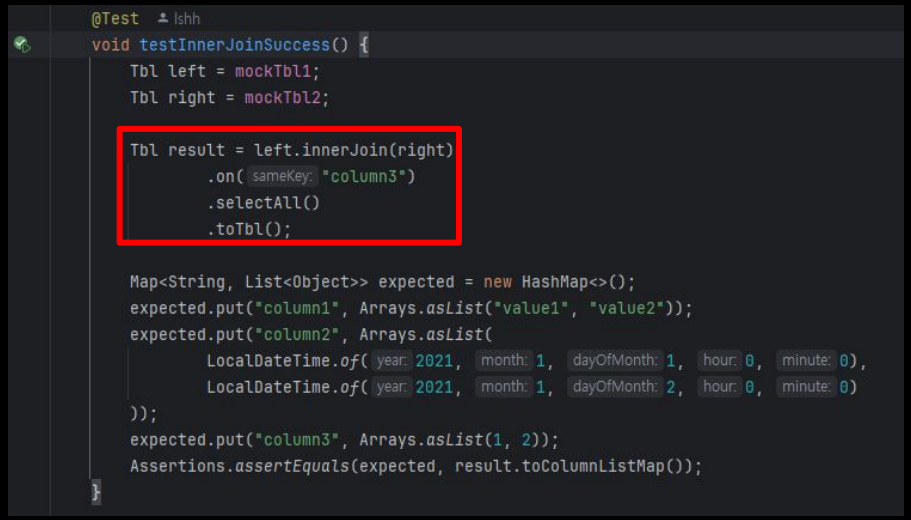
스키마가 다른 두 테이블을 준비

예제4 - Inner Join

```java
@Test  lshh
void testInnerJoinSuccess() {
    Tbl left = mockTbl1;
    Tbl right = mockTbl2;

    Tbl result = left.innerJoin(right)
            .on( sameKey: "column3")
            .selectAll()
            .toTbl();

    Map<String, List<Object>> expected = new HashMap<>();
    expected.put("column1", Arrays.asList("value1", "value2"));
    expected.put("column2", Arrays.asList(
            LocalDateTime.of( year: 2021,  month: 1,  dayOfMonth: 1,  hour: 0,  minute: 0),
            LocalDateTime.of( year: 2021,  month: 1,  dayOfMonth: 2,  hour: 0,  minute: 0)
    ));
    expected.put("column3", Arrays.asList(1, 2));
    Assertions.assertEquals(expected, result.toColumnListMap());
}
```

스키마가 다른 두 테이블을 준비

예제4 - Inner Join



```
@Test  lshh
void testInnerJoinSuccess() {
    Tbl left = mockTbl1;
    Tbl right = mockTbl2;

    Tbl result = left.innerJoin(right)
            .on( sameKey: "column3")
            .selectAll()
            .toTbl();

    Map<String, List<Object>> expected = new HashMap<>();
    expected.put("column1", Arrays.asList("value1", "value2"));
    expected.put("column2", Arrays.asList(
            LocalDateTime.of( year: 2021,  month: 1,  dayOfMonth: 1,  hour: 0,  minute: 0),
            LocalDateTime.of( year: 2021,  month: 1,  dayOfMonth: 2,  hour: 0,  minute: 0)
    ));
    expected.put("column3", Arrays.asList(1, 2));
    Assertions.assertEquals(expected, result.toColumnListMap());
}
```

스키마가 다른 두 테이블을 준비

column3로 inner join

예제4 - Inner Join

```java
@Test  lshh
void testInnerJoinSuccess() {
    Tbl left = mockTbl1;
    Tbl right = mockTbl2;

    Tbl result = left.innerJoin(right)
            .on( sameKey: "column3")
            .selectAll()
            .toTbl();

    Map<String, List<Object>> expected = new HashMap<>();
    expected.put("column1", Arrays.asList("value1", "value2"));
    expected.put("column2", Arrays.asList(
            LocalDateTime.of( year: 2021,  month: 1,  dayOfMonth: 1,  hour: 0,  minute: 0),
            LocalDateTime.of( year: 2021,  month: 1,  dayOfMonth: 2,  hour: 0,  minute: 0)
    ));
    expected.put("column3", Arrays.asList(1, 2));
    Assertions.assertEquals(expected, result.toColumnListMap());
}
```

스키마가 다른 두 테이블을 준비

column3로 inner join

예제4 - Inner Join

```java
@Test  lshh
void testInnerJoinSuccess() {
    Tbl left = mockTbl1;
    Tbl right = mockTbl2;

    Tbl result = left.innerJoin(right)
            .on( sameKey: "column3")
            .selectAll()
            .toTbl();

    Map<String, List<Object>> expected = new HashMap<>();
    expected.put("column1", Arrays.asList("value1", "value2"));
    expected.put("column2", Arrays.asList(
            LocalDateTime.of( year: 2021,  month: 1,  dayOfMonth: 1,  hour: 0,  minute: 0),
            LocalDateTime.of( year: 2021,  month: 1,  dayOfMonth: 2,  hour: 0,  minute: 0)
    ));
    expected.put("column3", Arrays.asList(1, 2));
    Assertions.assertEquals(expected, result.toColumnListMap());
}
```

스키마가 다른 두 테이블을 준비

column3로 inner join

toColumnListMap변환시, 결과가 같다.

E.O.D.

메모리적인 이슈 확인 필요

예외적인 상황 발견 필요