数据湖计算

SQL 语法

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

文档目录

- SQL 语法
  - SuperSQL 语法
    - SuperSQL 语法概览
    - 统一语法
      - 常用数据类型
      - DDL 语法
        - CREATE DATABASE
        - SHOW DATABASES
        - DESCRIBE DATABASE
        - ALTER DATABASE
          - ALTER DATABASE SET DBPROPERTIES
          - ALTER DATABASE SET LOCATION
        - DROP DATABASE
        - CREATE TABLE
        - REPLACE TABLE AS SELECT
        - SHOW TABLES
        - SHOW CREATE TABLE
        - SHOW TBLPROPERTIES
        - DESCRIBE TABLE
        - SHOW COLUMNS IN TABLE
        - ALTER TABLE
          - ALTER TABLE ADD COLUMNS
          - ALTER TABLE ADD COLUMN AFTER/FIRST
          - ALTER TABLE DROP COLUMN
          - ALTER TABLE ADD PARTATION
          - SHOW PARTITIONS
          - ALTER TABLE DROP PARTITION
          - ALTER TABLE ADD PARTITION FIELD
          - ALTER TABLE DROP PARTITION FIELD
          - ALTER TABLE ... RENAME COLUMN
          - ALTER TABLE SET TBLPROPERTIES
          - ALTER TABLE SET LOCATION
          - ALTER TABLE ... WRITE ORDERED BY
          - ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION
          - ALTER TABLE ... SET IDENTIFIER FIELDS

版权所有：腾讯云计算（北京）有限责任公司

ALTER TABLE ... DROP IDENTIFIER FIELDS
MSCK REPAIR TABLE
ANALYZE TABLES
DROP TABLE
EXPLAIN
CALL STATEMENT
CREATE VIEW AS
SHOW VIEWS
DESCRIBE VIEW
SHOW CREATE VIEW
SHOW COLUMNS IN VIEW
ALTER VIEW
ALTER VIEW RENAME TO
ALTER VIEW SET TBLPROPERTIES
DROP VIEW
CREATE FUNCTION
SHOW FUNCTION
DROP FUNCTION
DML 语法
INSERT STATEMENT
INSERT INTO
INSERT OVERWRITE
MERGE INTO
UPDATE
DELETE STATEMENT
TABLE METADATA
DQL 语法
SELECT STATEMENT
Iceberg 表语法
DDL 语法
DML 语法
DQL 语法
Procedure
Iceberg 外部表与原生表语法差异
物化视图语法
SQL 隐式转换
函数
统一函数

统一函数概览
二进制函数
位运算函数
集合函数
日期时间函数
JSON 函数
数学函数
字符串函数
聚合函数
窗口函数
其他函数
Presto 内置函数
Hive 函数对照
标准 Spark 语法概览
标准 Presto 语法概览
保留字

版权所有：腾讯云计算（北京）有限责任公司

SQL 语法
SuperSQL 语法
SuperSQL 语法概览

最近更新时间：2024-08-07 17:09:24

DLC 通过一套标准 SQL 语法几乎可以无缝在 DLC Serverless Spark 和 DLC Serverless Presto 引擎上运行。元数据和分析语法、函数，基本兼容 Hive 和 Spark 语法，支持自定义函数。
系统内置函数支持范围可参考 统一函数概览，如您需要使用 Presto 内置函数，使用方式及函数支持范围参见 Presto 内置函数 。
如您需要在数据湖计算中对外部 Iceberg 表进行数据查询分析，部分语法与原生表存在差异，具体可参见 Iceberg 外部表与原生表语法差异。
DLC 支持的语法如下表所示：

DDL 语法

数据库相关语法

用途	语法
新建数据库	CREATE DATABASE
展示在该元数据中定义的所有数据库	SHOW DATABASES
查看数据库属性	DESCRIBE DATABASE
数据库属性变更	ALTER DATABASE SET DBPROPERTIES
数据库存储位置变更	ALTER DATABASE SET LOCATION
删除数据库	DROP DATABASE

数据表相关语法

用途	语法
新建数据表	CREATE TABLE
更新表快照	REPLACE TABLE AS SELECT
查询数据表建表信息	SHOW CREATE TABLE

版权所有：腾讯云计算（北京）有限责任公司

查询表属性	SHOW TBLPROPERTIES
查询数据库中的所有表	SHOW TABLES
查看数据表列信息及元数据信息	DESCRIBE TABLE
查询数据表列信息	SHOW COLUMNS IN TABLE
向数据表添加列	ALTER TABLE ADD COLUMNS
对数据表新增列	ALTER TABLE ADD COLUMN AFTER/FIRST
变更字段名称	ALTER TABLE ... RENAME COLUMN
删除数据表某个字段	ALTER TABLE DROP COLUMN
对数据表新增分区信息	ALTER TABLE ADD PARTATION
列出表分区	SHOW PARTITIONS
删除数据表分区信息	ALTER TABLE DROP PARTITION
对 Iceberg 表添加分区字段	ALTER TABLE ADD PARTITION FIELD
删除 Iceberg 表分区字段	ALTER TABLE DROP PARTITION FIELD
数据表属性变更	ALTER TABLE SET TBLPROPERTIES
数据表存储位置变更	ALTER TABLE SET LOCATION
修改表数据的排序方式	ALTER TABLE ... WRITE ORDERED BY
修改分区表的分配策略	ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION
添加 identifier fields 属性	ALTER TABLE ... SET IDENTIFIER FIELDS
删除 identifier fields 属性	ALTER TABLE ... DROP IDENTIFIER FIELDS
更新分区信息	MSCK REPAIR TABLE
对数据表进行统计	ANALYZE TABLES
删除元数据表	DROP TABLE
展示执行 sql 的逻辑或物理计划	EXPLAIN
调用表存储过程	CALL STATEMENT

版权所有：腾讯云计算（北京）有限责任公司

视图相关语法

用途	语法
将 select 结果创建为视图	CREATE VIEW AS
查询数据库中的视图	SHOW VIEWS
查看视图的列信息	DESCRIBE VIEW
展示视图创建语句	SHOW CREATE VIEW
查看视图列信息	SHOW COLUMNS IN VIEW
修改视图名称	ALTER VIEW RENAME TO
修改视图属性	ALTER VIEW SET TBLPROPERTIES
删除视图	DROP VIEW

函数相关语法

用途	
创建函数	CREATE FUNCTION
查看创建函数语法	SHOW FUNCTION
删除函数	DROP FUNCTION

DML 语法

用途	
插入一行数据	INSERT STATEMENT
替换一行数据	INSERT OVERWRITE
行级数据更新操作，可用于替换 INSERT OVERWRITE 操作	MERGE INTO
Iceberg 表元数据查询	TABLE METADATA
将查询结果插入数据表	INSERT INTO
删除 Iceberg 表的数据	DELETE STATEMENT

版权所有：腾讯云计算（北京）有限责任公司

更新指定行	UPDATE

DQL 语法

用途	
数据查询	SELECT STATEMENT

相关查询保留字参见 保留字。

统一语法

常用数据类型

最近更新时间：2024-08-07 17:09:51

数据类型分类	数据类型	描述
数字	ByteType：BYTE, TINYINT	1字节有符号整数，数字的范围是从-128到127
	ShortType：SHORT, SMALLINT	2字节有符号整数，数字的范围是从-32768到3276
	IntegerType：INT, INTEGER	4字节有符号整数，数字的范围是从-2147483648到214748364
	LongType：LONG, BIGINT	8字节有符号整数，范围是从-9223372036854775808到922337203685477580
	FloatType：FLOAT, REAL	4字节单精度浮点数
	DoubleType：DOUBLE	8字节双精度浮点数
	DecimalType：DECIMAL, DEC, NUMERIC	任意精度的有符号十进制数
字符串	StringType：STRING	字符串值，例如：'abc'
字节	BinaryType：BINARY	字节序列值
布尔	BooleanType：BOOLEAN	布尔值，例如：true/false
时间	DateType：DATE	年、月、日字段值组成的值，不带时区 例如：'2023-10-01'
	TimestampType：TIMESTAMP	年、月、日、小时、分钟和秒字段值以及本地时区组成的值 例如：'2023-10-01 23:59:59'
复杂类型	ArrayType：ARRAY<element_type>	由 element_type 类型元素组成的序列值 例如：array<int>
	MapType：MAP<key_type, value_type>	由一组键值对组成的值，key_type 指定键的类型，value_type 指定值的类型 例如：map<string, int>

	StructType：STRUCT<field1_name: field1_type, field2_name: field2_type, ...>	由 fields 组成的结构值 例如：struct<id:int, grade:string>

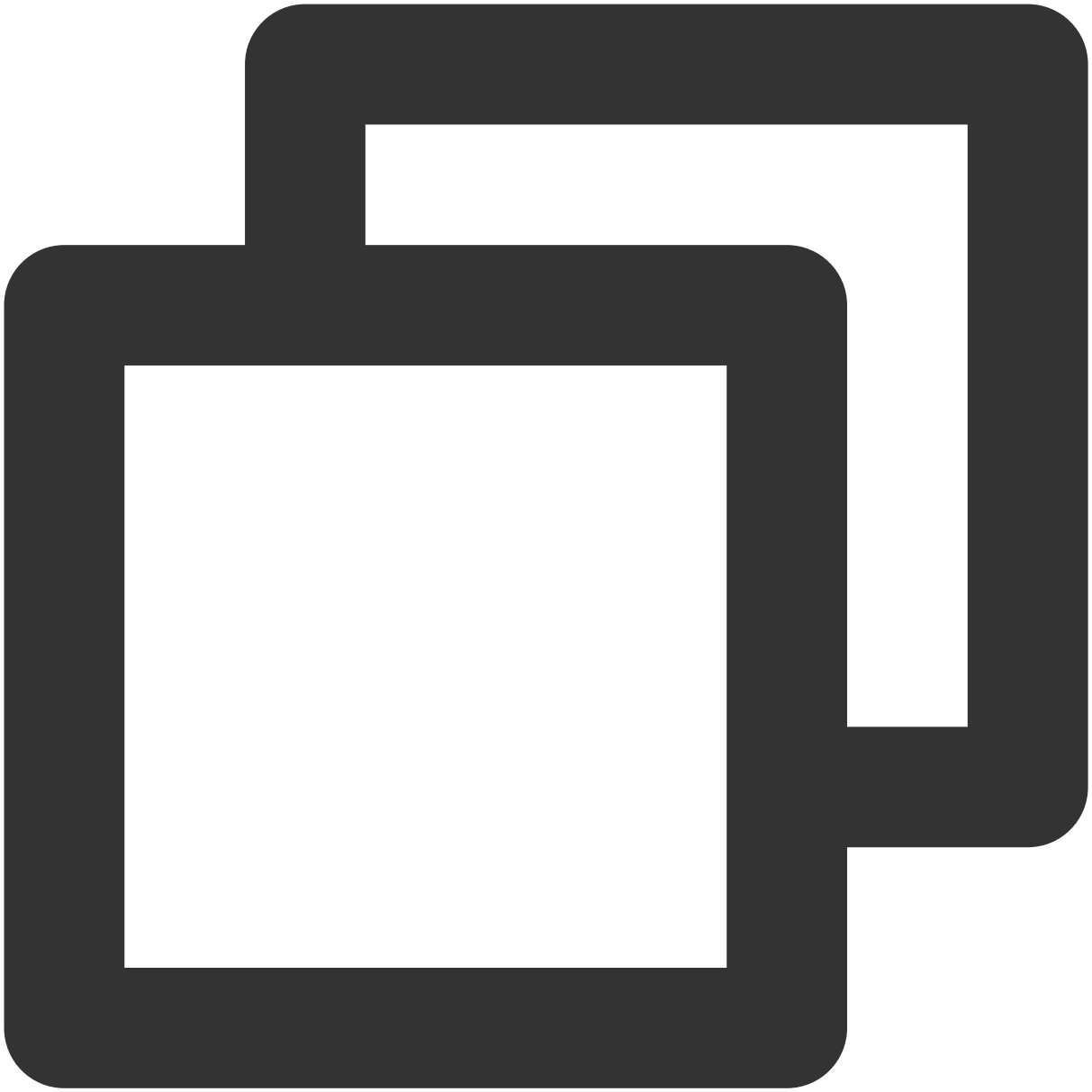
DDL 语法
CREATE DATABASE

最近更新时间：2024-08-07 17:10:14

说明

支持内核：Presto、SparkSQL。
用途：创建一个数据库，也可以通过指定和改变表或者分区表的存储位置。

语法



```
CREATE {DATABASE|SCHEMA} [IF NOT EXISTS] database_name
     [COMMENT 'database_comment']
     [WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]
```

参数

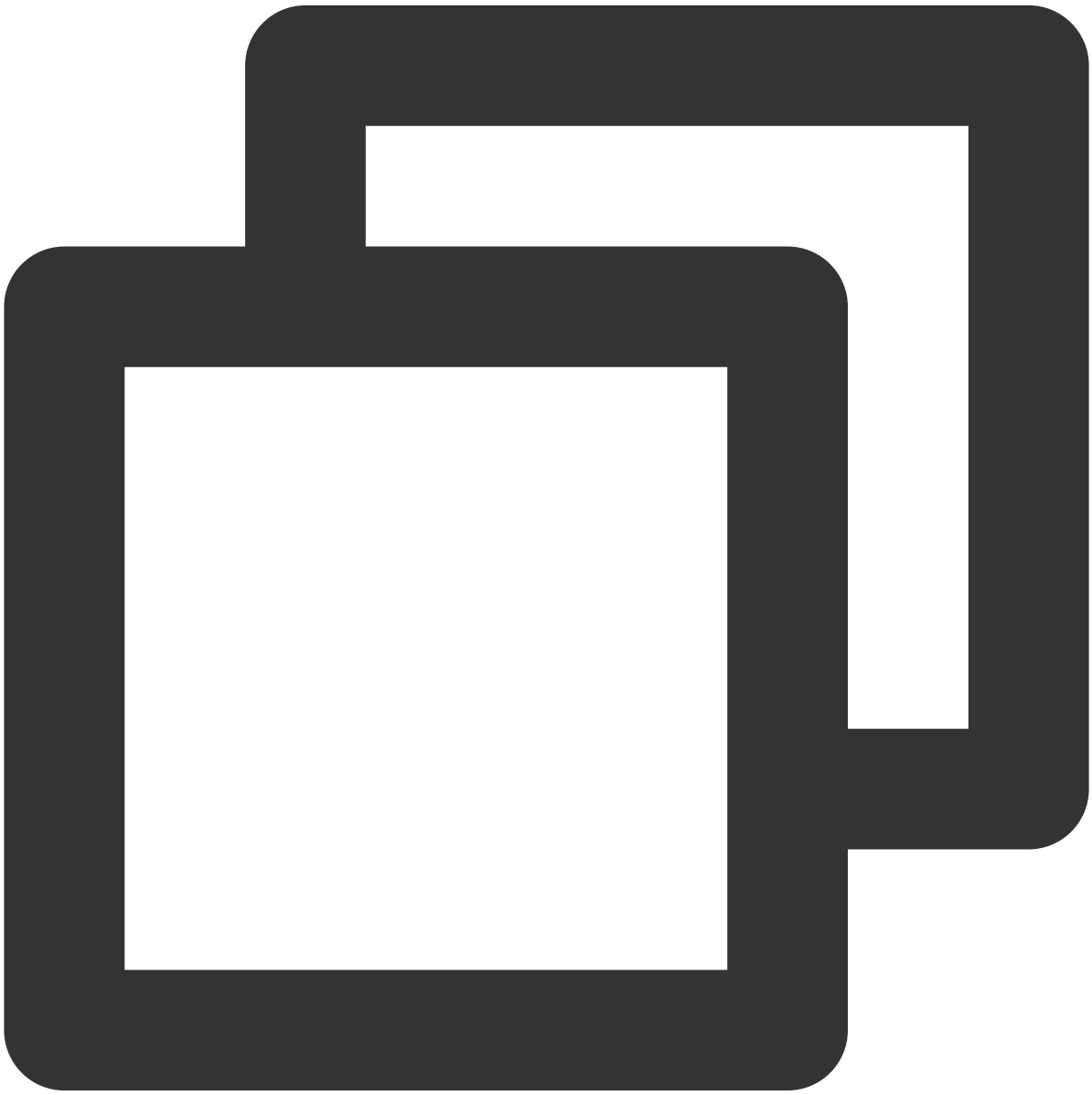
`DATABASE|SCHEMA` ：同一个意思，都可以使用。
`database_name` ：数据库名称。

`database_comment` ：数据库注释。

`[WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]` ：以 `key-value` 的形式配置数据库参数。

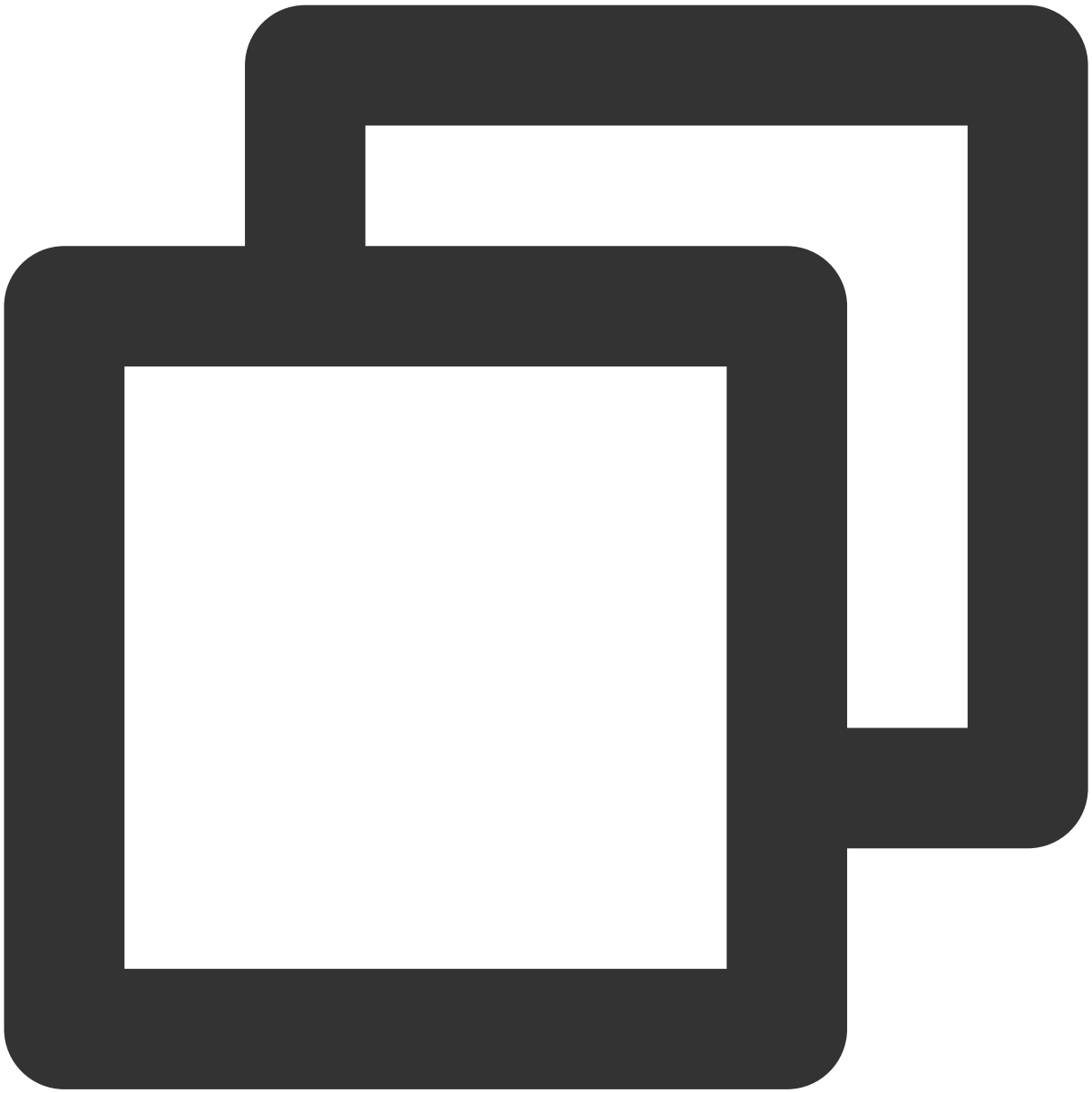
示例

Create database `db` only if database with same name doesn't exist.



```
CREATE DATABASE IF NOT EXISTS db;
```

Create database `db` only if database with same name doesn't exist with `Comment` and `Database Properties`.



```
CREATE DATABASE db
COMMENT 'db1_name'
WITH DBPROPERTIES('k1' = 'v1','k2' = 'v2');
```

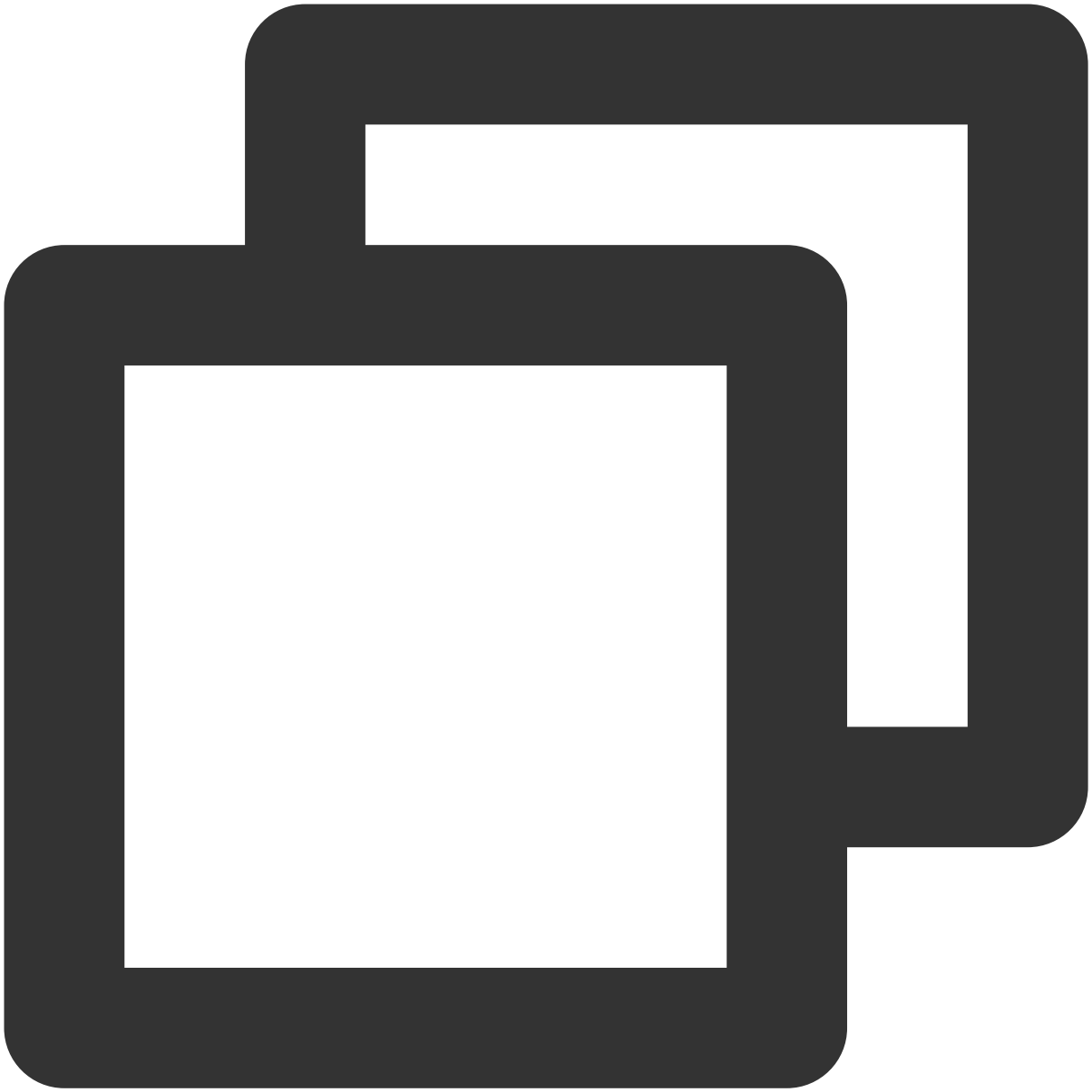
SHOW DATABASES

最近更新时间：2024-08-07 17:10:43

说明

支持内核：Presto、SparkSQL。
用途：列出所有在该元数据中定义的所有数据库，可使用 DATABASES 或者 SCHEMAS 做相同的查询。

语法



```
SHOW {DATABASES | SCHEMAS} [IN catalog_name] [LIKE 'regular_expression']
```

参数

`[IN catalog_name]` ：数据源名称。

`[LIKE 'regular_expression']` ：过滤出匹配的数据库名称。

示例

```
SHOW DATABASES;
SHOW DATABASES LIKE '.*analytics';
SHOW DATABASES IN catalog1 LIKE '.*analytics';
```

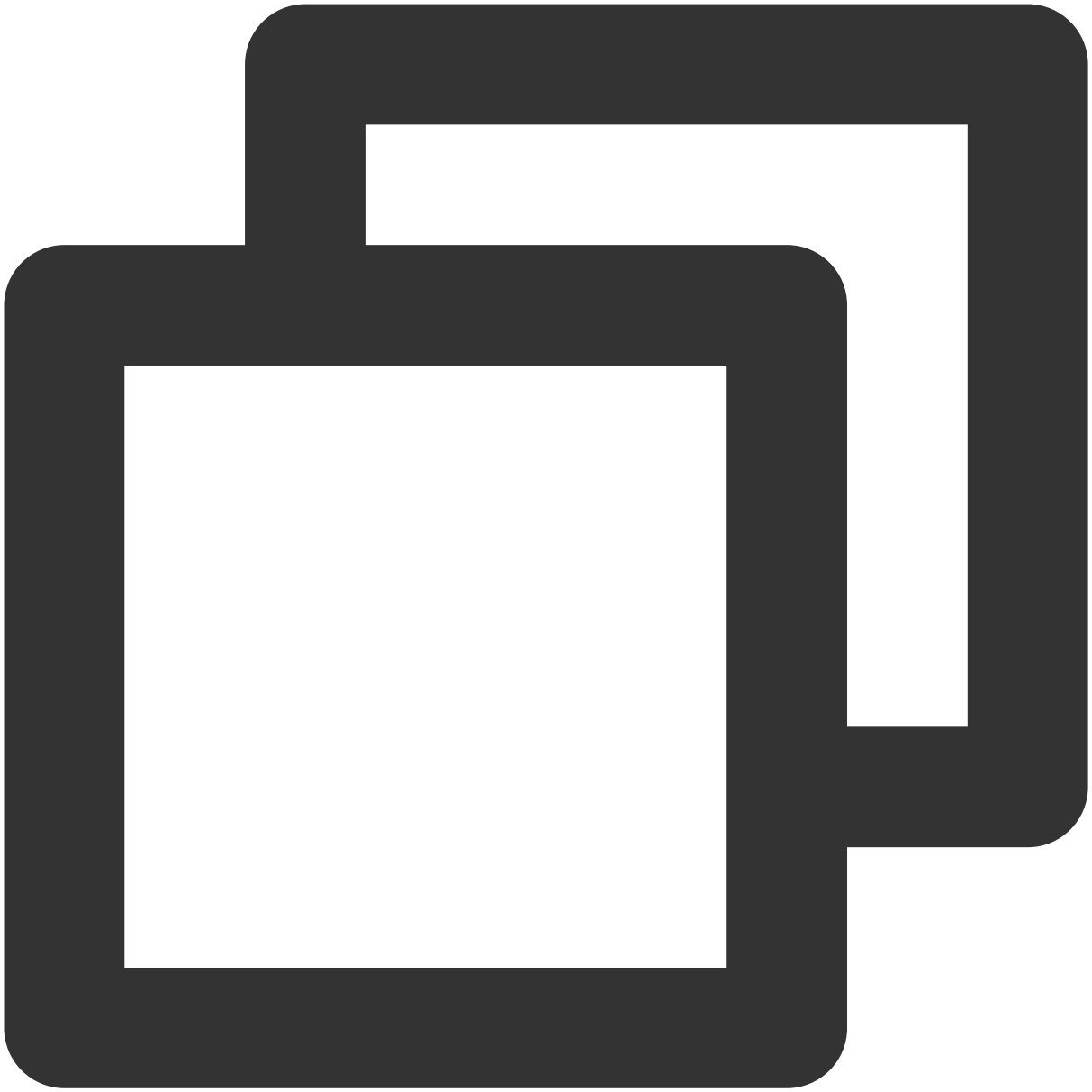
DESCRIBE DATABASE

最近更新时间：2024-08-07 17:10:58

说明

支持内核：Presto、SparkSQL。
用途：查看数据库属性。

标准语法



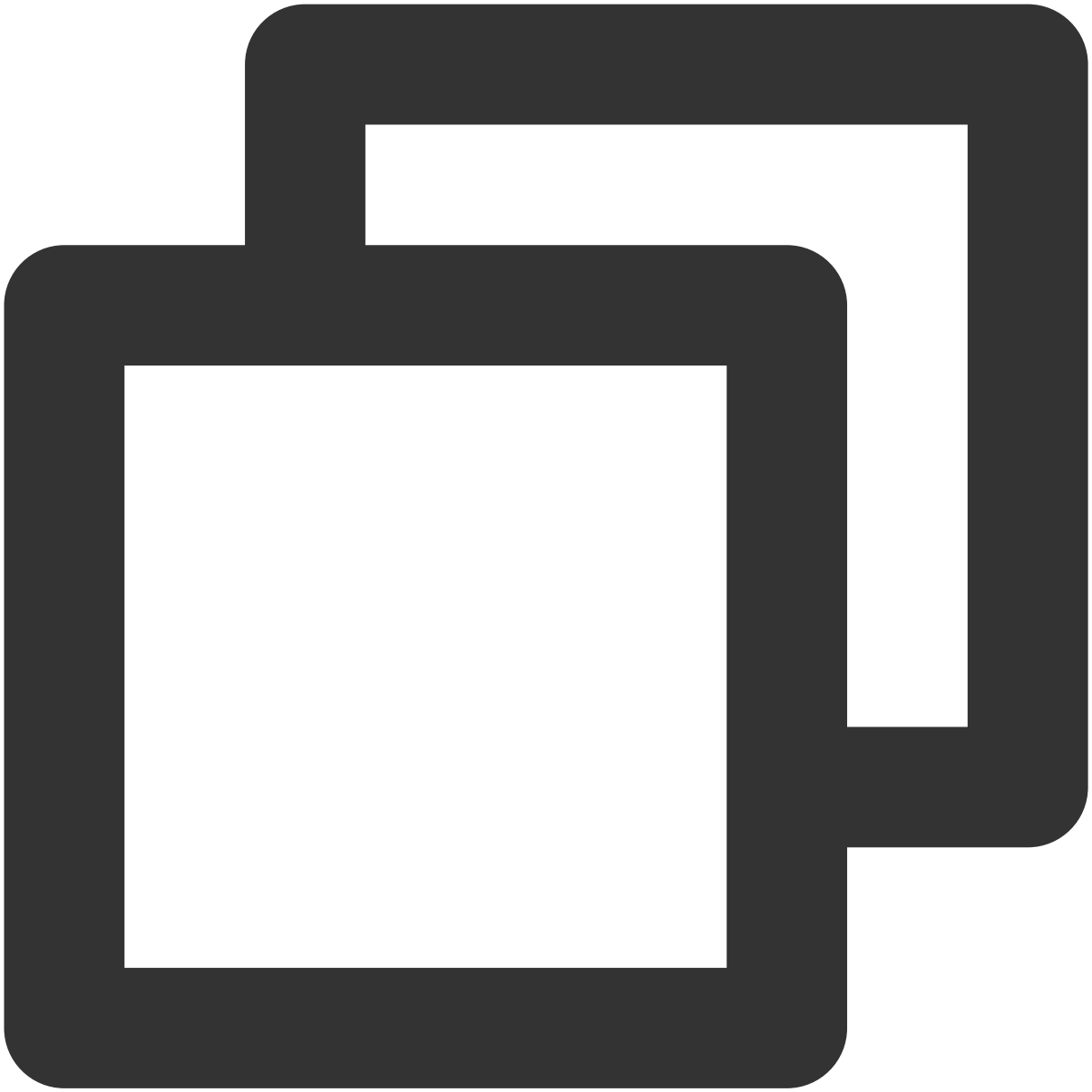
```
DESCRIBE SCHEMA|DATABASE [EXTENDED] DB_NAME
```

参数

`SCHEMA|DATABASE` ：指定库为 SCHEMA 或者 DATABASE。

`EXTENDED` ：该库是否为 EXTENDED。

示例



```
DESCRIBE DATABASE db_name;
DESCRIBE DATABASE EXTENDED db_name;
```

版权所有：腾讯云计算（北京）有限责任公司

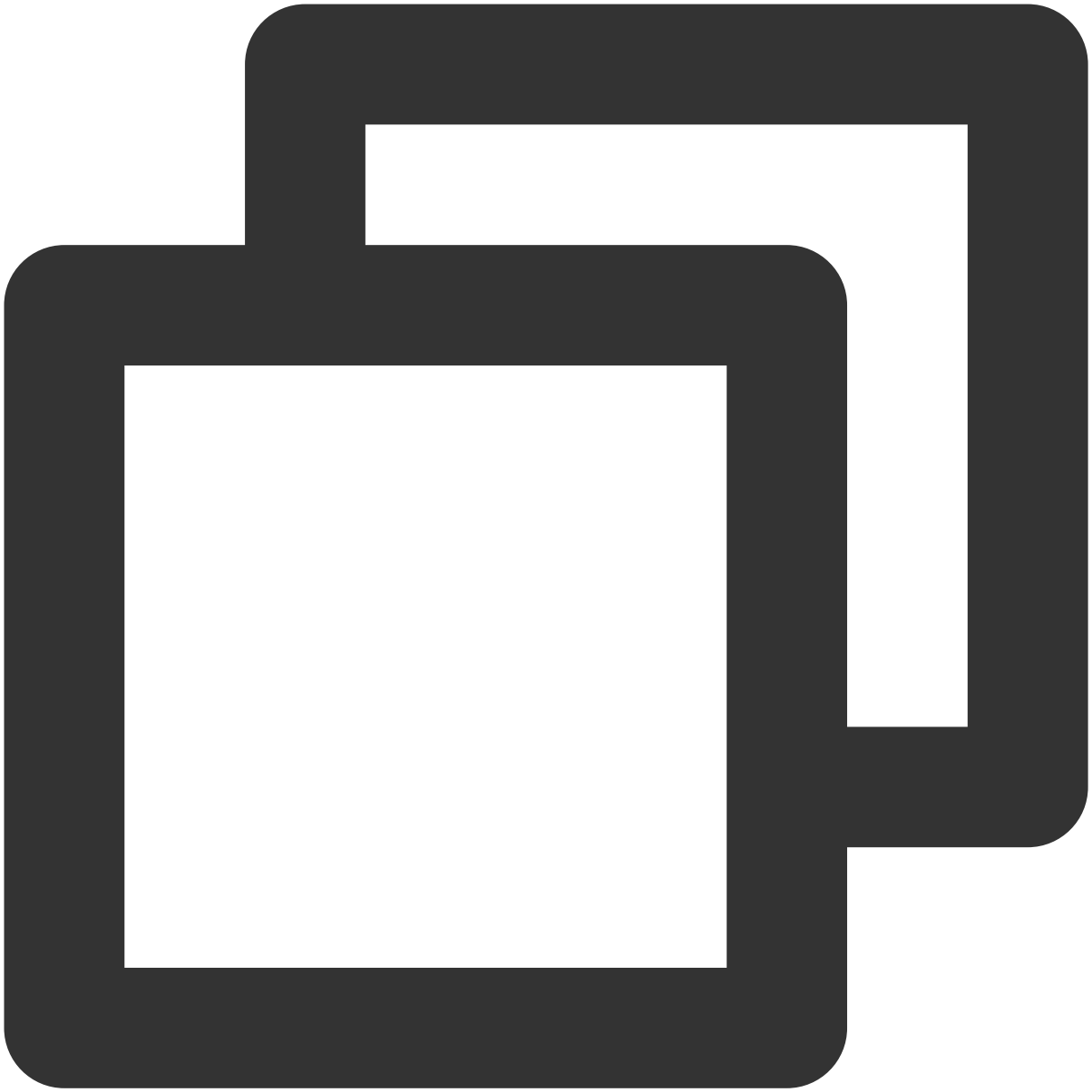
ALTER DATABASE
ALTER DATABASE SET DBPROPERTIES

最近更新时间：2024-08-07 17:11:23

说明

支持内核：Presto、SparkSQL。
用途：为数据库添加一个或多个属性，如果属性相同则会进行覆盖。

标准语法

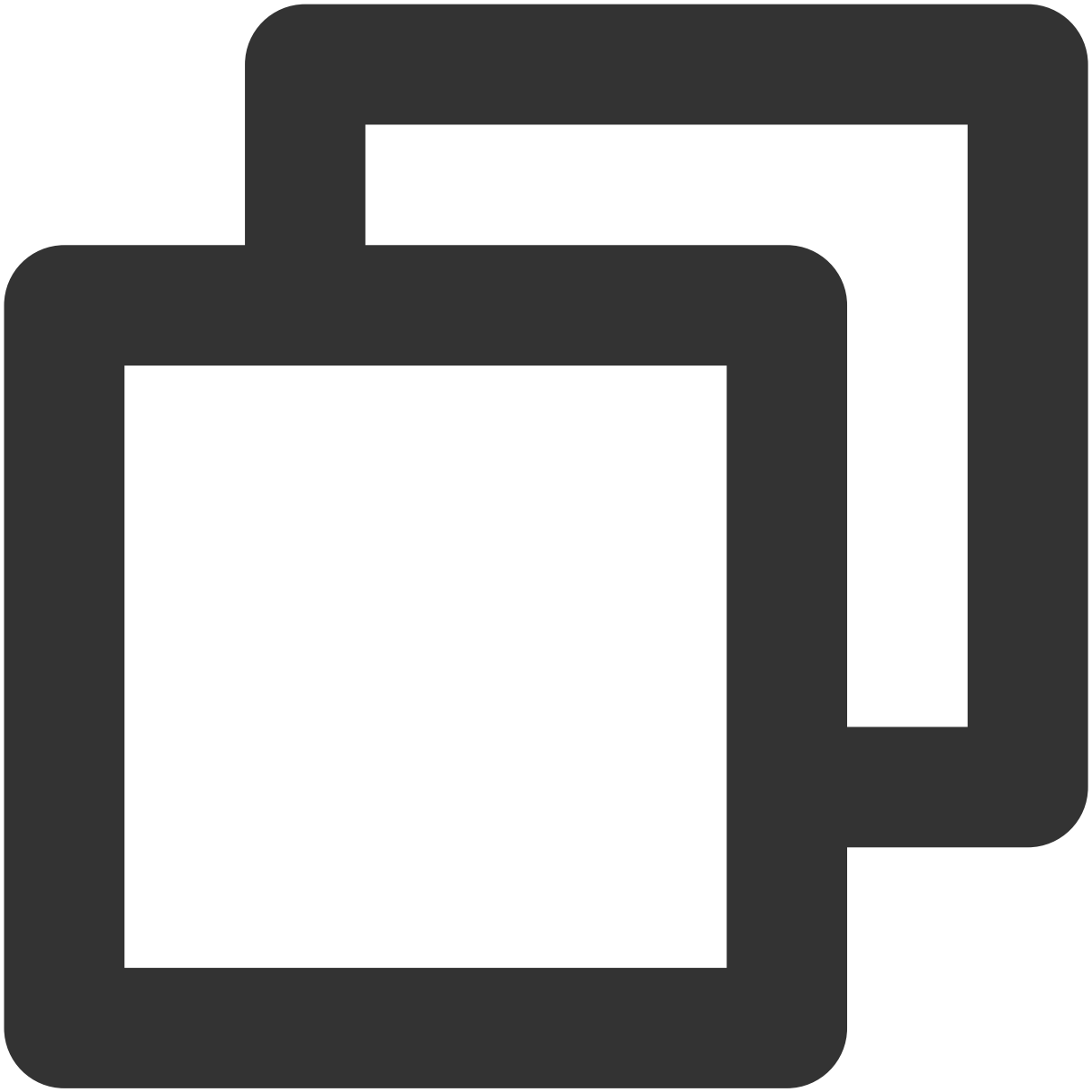


```
ALTER (DATABASE|SCHEMA) database_name SET DBPROPERTIES (property_name=property_valu
```

参数

`database_name` ：数据库名称。

示例



```
-- Alters the database to set properties `author`.
ALTER DATABASE product SET DBPROPERTIES ('author' = 'allen');
```

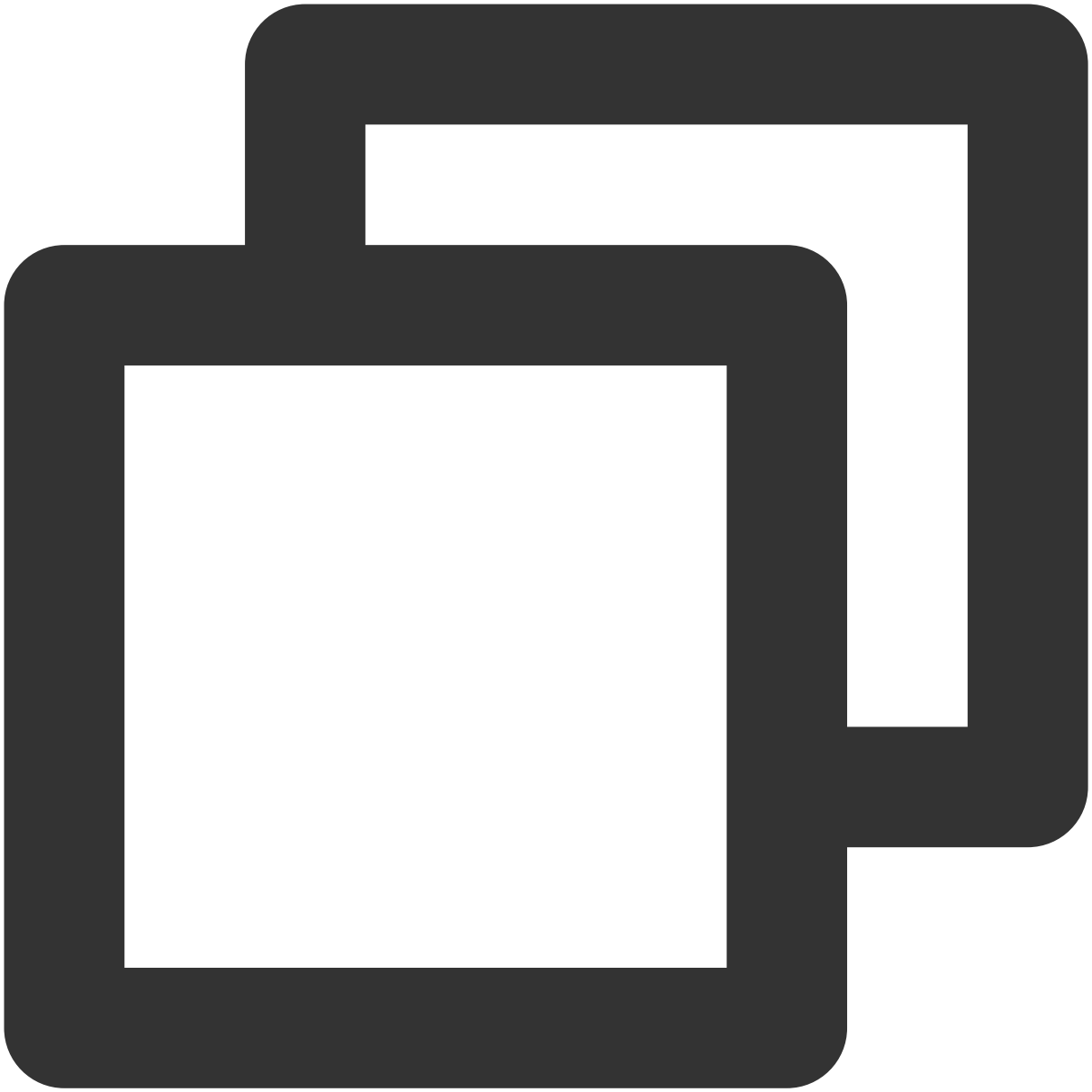
ALTER DATABASE SET LOCATION

最近更新时间：2024-08-07 17:11:37

说明

支持内核：Presto。
用途：修改数据库的存储路径。

标准语法

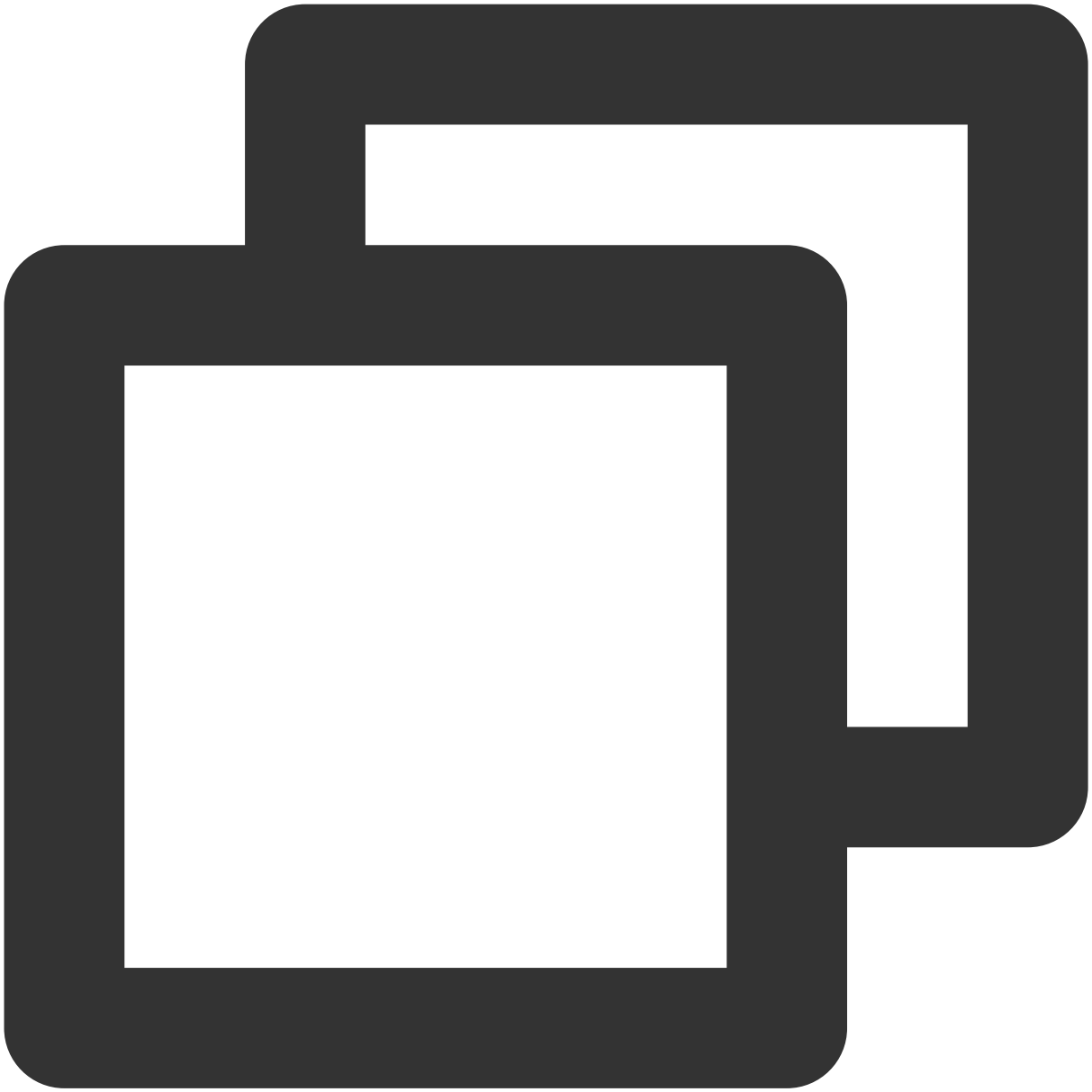


```
ALTER (DATABASE|SCHEMA) database_name SET LOCATION hdfs_path
```

参数

`[database_name]` ：数据库名称。

示例



```
ALTER DATABASE db01 SET LOCATION 'cosn:///new/path'
```

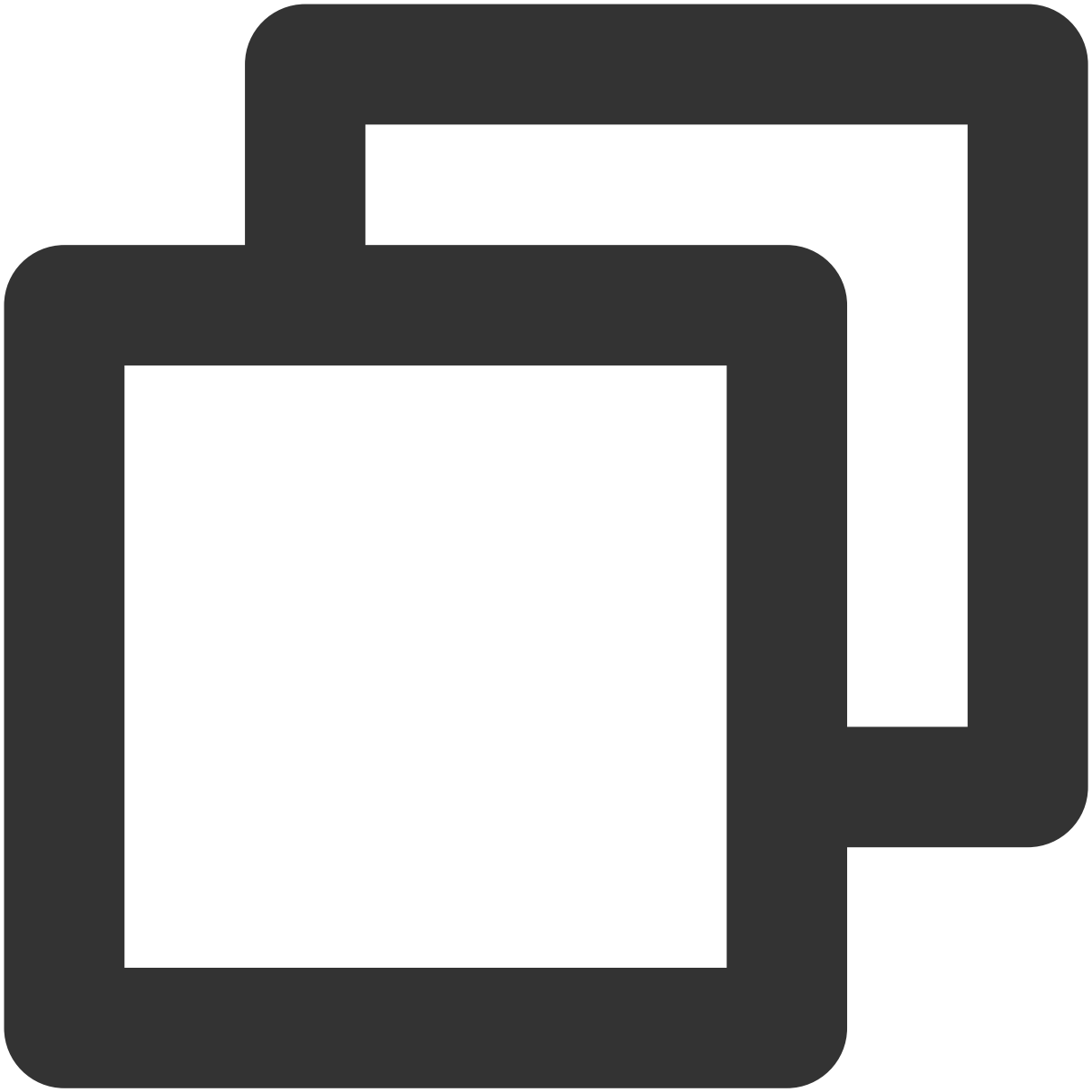
DROP DATABASE

最近更新时间：2024-08-07 17:12:01

说明

支持内核：Presto、SparkSQL。
用途：删除指定的数据库。

语法



```
DROP {DATABASE | SCHEMA} [IF EXISTS] database_name [RESTRICT | CASCADE]
```

参数

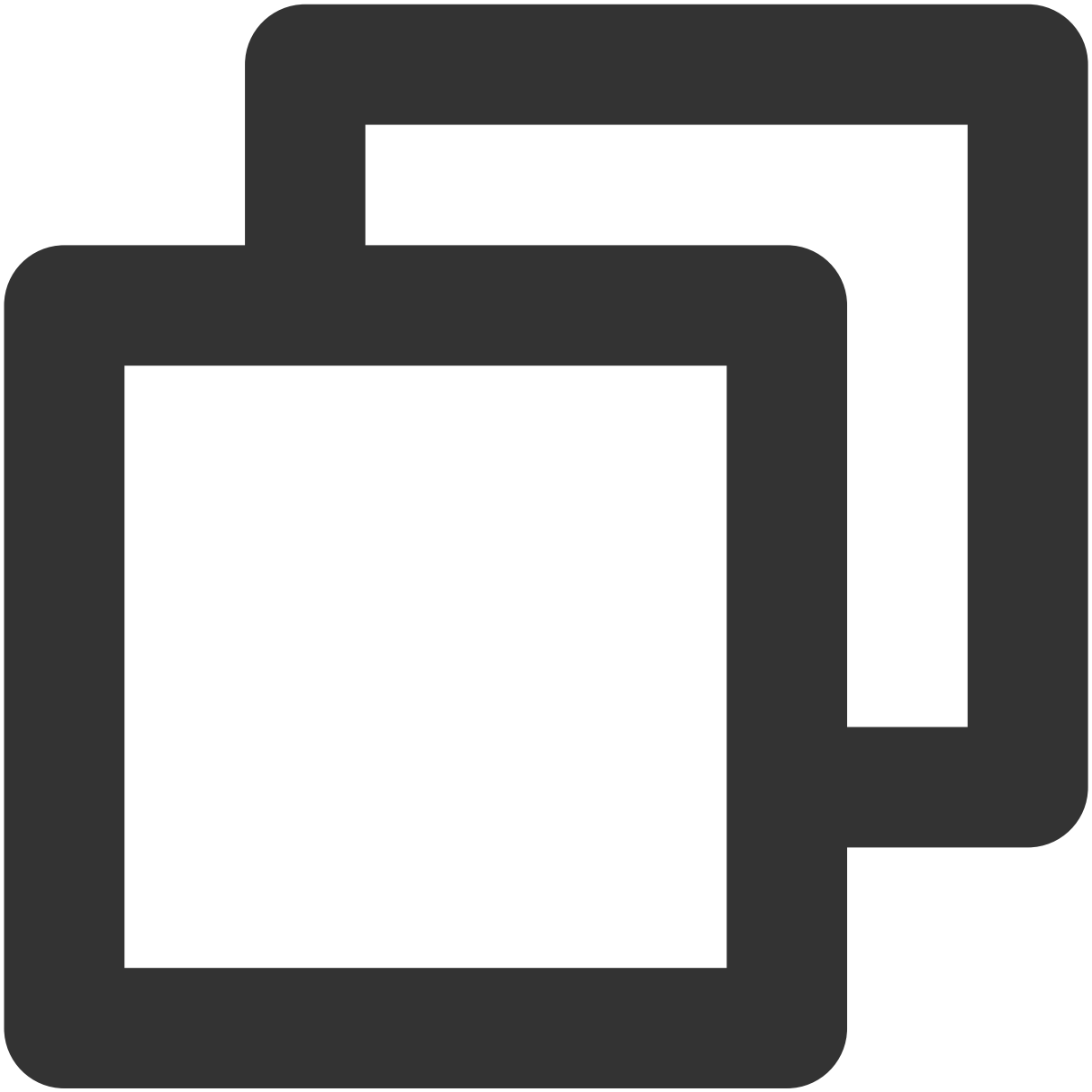
`DATABASE|SCHEMA` ：同一个意思，都可以使用。

`database_name` ：数据库名称。

`RESTRICT` ：如果数据库包含表，则不会删除该数据库，不填，默写是该模式。

`CASCADE` ：强制删除所有数据库表。

示例



```
-- Drop the database and it's tables
DROP DATABASE test CASCADE;

-- Drop the database using IF EXISTS
DROP DATABASE IF EXISTS test;
```

版权所有：腾讯云计算（北京）有限责任公司

CREATE TABLE

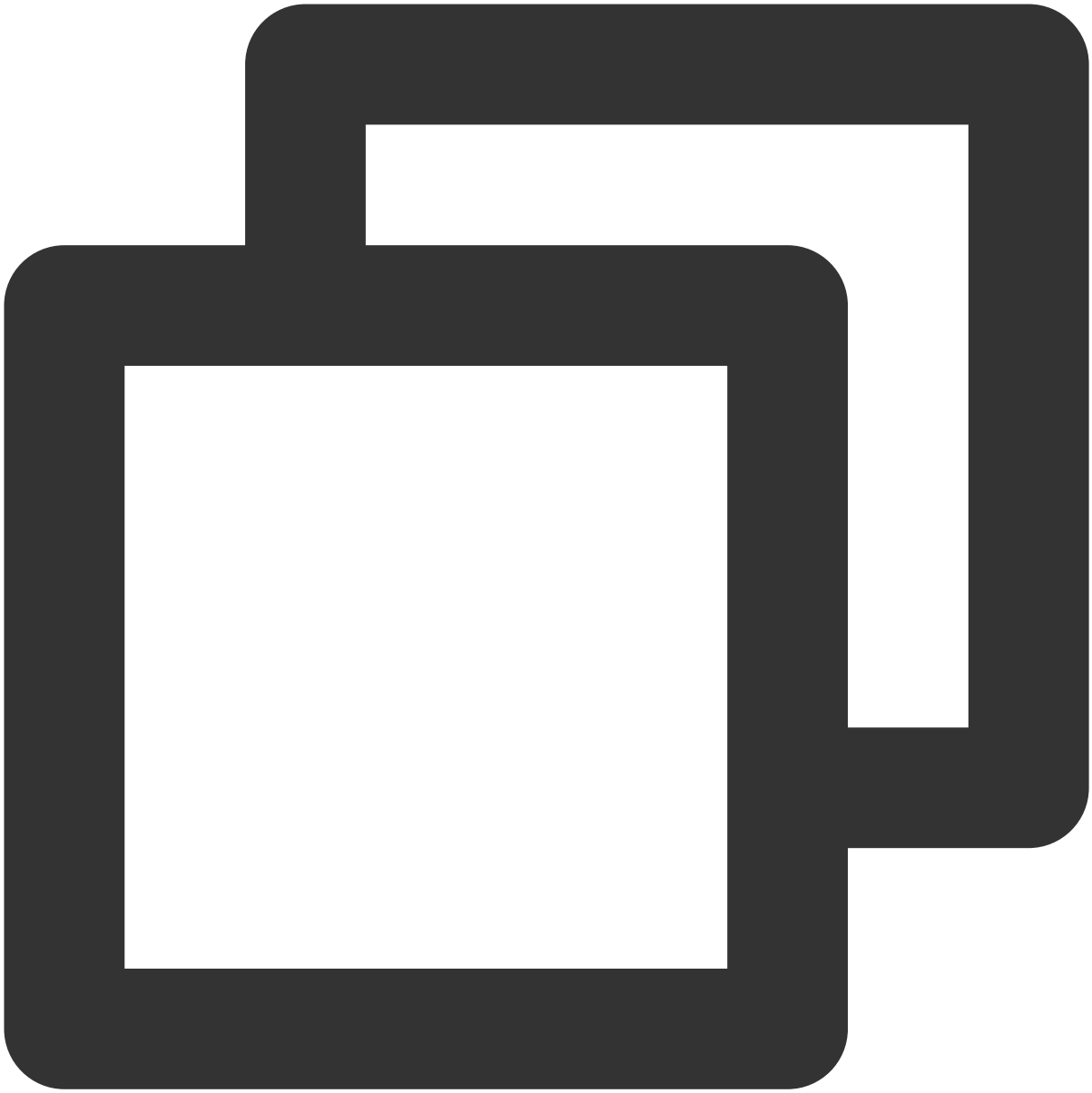
最近更新时间：2024-08-07 17:12:22

说明

支持内核：Presto、SparkSQL。
适用表范围：原生 Iceberg 表、外部表。
用途：创建一个表同时带一些属性，支持使用 CREATE TABLE AS 语法。
建表存储路径：建表存储路径支持指定至 COS 目录，不能指定到文件。

外部表语法

语法



```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
    ( col_name[:] col_type [ COMMENT col_comment ], ... )
USING data_source
    [ COMMENT table_comment ]
    [ OPTIONS ( 'key1'='value1', 'key2'='value2' )]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
    [ LOCATION path ]
    [ TBLPROPERTIES ( property_name=property_value, ... ) ]
```

参数

`USING data_source` ：建表时，数据的输入类型，目前有：CSV，ORC，PARQUET，ICEBERG 等。

`table_identifier` ：指定表名，支持三段式，例如：catalog.database.table。

`COMMENT` ：表的描述信息。

`OPTIONS` ：USING data_source支持的额外参数，用于存储时参数注入。

`PARTITIONED BY` ：基于指定的列创建分区。

`LOCATION path` ：数据表存储路径。

`TBLPROPERTIES` ：一组 k-v 值，用于指定表的参数。

USING和OPTIONS 参数详细说明

USING CSV

USING ORC

USING PARQUET

参考链接：Working with CSV

CSV 数据表的可支持配置如下。

OPTIONS 支持的 key	key 对应的 value 默认值	含义
sep或delimiter	,	csv存储时每列之间的分隔符，默认英文逗号
mode	PERMISSIVE	定义当数据转换时不符合预期时的处理模式。 PERMISSIVE：较宽松的模式，默认，尝试转换某一行数据，例如某一行多出来几列，会自动只取需要的列 DROPMALFORMED：丢弃不符合预期的数据，例如某一个行多出来列则该行会被丢弃 FAILFAST：严格要求csv格式，一旦某行不符合预期就失败，例如多出列的情况。
encoding或charset	UTF-8	字符串编码格式。 例如：UTF-8、US-ASCII、ISO-8859-1、UTF-16BE、UTF-16LE、UTF-16
quote	\\"	引号是单引号还是双引号，注意使用转义符
escape	\\\\	逃逸字符，注意使用转义符
charToEscapeQuoteEscaping	-	引号内部需要逃逸的字符
comment	\\u0000	备注信息
header	false	存在表头
inferSchema	false	推断每列类型，不推断则每一列均为字符串
ignoreLeadingWhiteSpace	读：false	忽略开头的空字符串

	写：true	
ignoreTrailingWhiteSpace	读：false 写：true	忽略结尾的空字符串
columnNameOfCorruptRecord	_corrupt_record	无法转换的列的列名，该参数受spark.sql.columnNameOfCorruptRecord影响，以表配置为主
nullValue	-	null的存储格式，默认为空字符串，此时按emptyValue的方式写
nanValue	NaN	非数值类型的值的存储格式
positiveInf	Inf	正无穷大的存储格式
negativeInf	-Inf	负无穷大的存储格式
compression或codec	-	压缩算法的类名，默认不压缩，可以使用简称，bzip2、deflate、gzip、lz4、snappy
timeZone	系统默认时区	默认时区，该参数取值受spark.sql.session.timeZone影响，例如Asia/Shanghai，以表配置为主
locale	en-US	语言类型
dateFormat	yyyy-MM-dd	默认日期的格式
timestampFormat	yyyy-MM-dd'T'HH:mm:ss.SSSXXX	默认时间的格式，非LEGACY模式下为yyyy-MM-dd'T'HH:mm:ss[.SSS][XXX]
multiLine	false	允许多行
maxColumns	20480	最大列数
maxCharsPerColumn	-1	每列最大字符数，-1表示不限制
escapeQuotes	true	逃逸引号
quoteAll	quoteAll	写时全文加引号
samplingRatio	1.0	采样比例
enforceSchema	true	强制使用指定的schema读取，会忽略表头的定义
emptyValue	读： 写：\\"\\"	空值的读写格式

lineSep	-	换行符
inputBufferSize	-	读时的buffer size，该参数可受spark.sql.csv.parser.inputBufferSize影响，以表配置为主
unescapedQuoteHandling	STOP_AT_DELIMITER	非逃逸引号被发现时的处理策略。 STOP_AT_DELIMITER：读到分隔符停止 BACK_TO_DELIMITER：回退到分隔符 STOP_AT_CLOSING_QUOTE：读到下一个引号停止 SKIP_VALUE：跳过该列数据 RAISE_ERROR：报错

ORC 数据表可支持的配置如下：

OPTIONS 支持的 key	key 对应的 value 默认值	含义
compression或orc.compress	snappy	压缩算法，支持简写snappy/zlib/lzo/lz3/zstd，该参数受spark.sql.orc.compression.codec影响，以表参数为主
mergeSchema	false	合并schema，该参数受spark.sql.orc.mergeSchema影响，以表参数为主

如果是使用 HiveRead 和 HiveWriter（配置spark.sql.hive.convertMetastoreOrc=false）来读写，OPTIONS 还可以支持 Orc 原生的配置，详情请参考 LanguageManual ORC。

PARQUET 数据表相关参数大多可以通过 Spark conf 配置，也更建议从 spark conf 配置。options 也可支持的配置如下：

OPTIONS 支持的 key	key 对应的 value 默认值	含义
compression或parquet.compression	snappy	压缩算法，默认使用snappy，受参数spark.sql.parquet.compression.codec影响，以表参数为主。
mergeSchema	false	是否合并schema，受参数spark.sql.parquet.mergeSchema影响，以表参数为主。
datetimeRebaseMode	EXCEPTION	写parquet文件时日期的转换策略。LEGACY模

		式将日期做公历转换、CORRECTED不对日期做公历转换、EXCEPTION在遇到日期属于不同格式时报错。受参数spark.sql.parquet.datetimeRebaseModeInRead影响，以表参数为主。
int96RebaseMode	EXCEPTION	读parquet文件时时间的转换策略。LEGACY模式将对时间做公历转换、CORRECTED不对时间做转换、EXCEPTION则在遇到不同格式的时间时报错。受参数spark.sql.parquet.int96RebaseModeInRead影响，以表参数为主。

如果是使用HiveRead和HiveWriter（配置spark.sql.hive.convertMetastoreParquet=false）来读写，OPTIONS还可以支持Parquet原生的配置参考：Hadoop integration。

示例

```
CREATE TABLE dempts(
    id bigint COMMENT 'id number',
    num int,
    eno float,
    dno double,
    cno decimal(9,3),
    flag  boolean,
    data string,
    ts_year timestamp,
    date_month date,
    bno binary,
```

```
    point struct<x: double, y: double>,
    points array<struct<x: double, y: double>>,
    pointmaps map<struct<x: int>, struct<a: int>>
    )
USING iceberg
COMMENT 'table documentation'
PARTITIONED BY (bucket(16,id), years(ts_year), months(date_month), identity(bno),
LOCATION '/warehouse/db_001/dempts'
TBLPROPERTIES ('write.format.default'='orc');
```

常见问题

CREATE_TABLE 时的关键字里，Spark 的 USING 和 Hive 的 STORED AS 存在差异，可能会导致创建表后文件格式、读取都不符合预期。这里做一个特殊说明：

USING DATA_SOURCE：Spark 语法，该关键字表示创建表时使用哪种数据源作为输入的格式，直接影响表的 Location 下的文件格式和读取方式。可取值例如 CSV、TXT、Iceberg、Parquet、Orc 等。

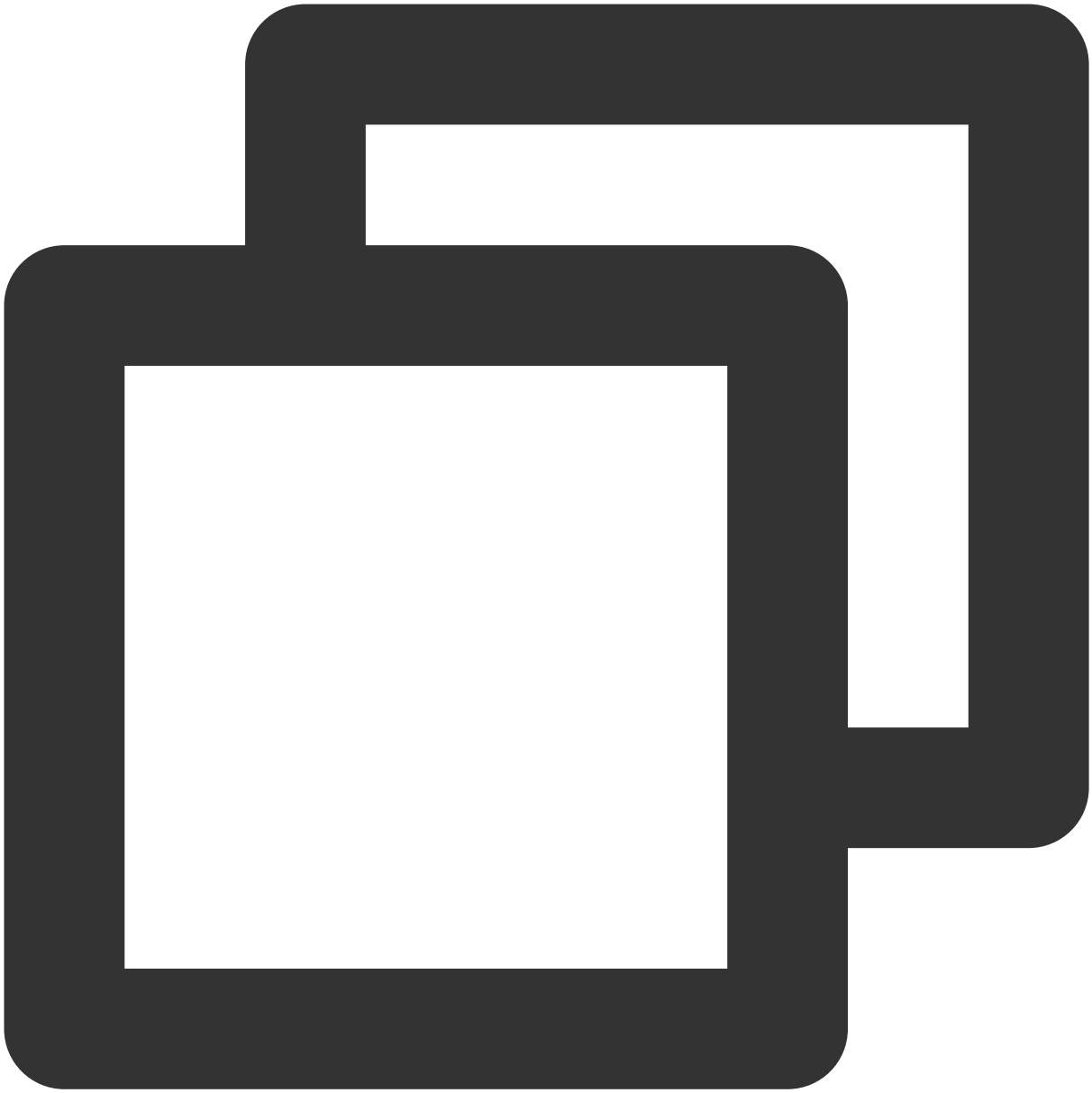
STORED AS FILE_FORMAT：Hive 语法，该关键字用于创建一个 HIVE 格式的表，表示表中存储的数据文件的格式。可取值例如 TXT、Parquet、Orc 等。不建议使用这种语法，可能会导致 Spark 原生的 reader/writer 无法支持，例如不支持 CSV。

原生表 Iceberg 语法

注意

该语法仅支持创建原生表。

语法

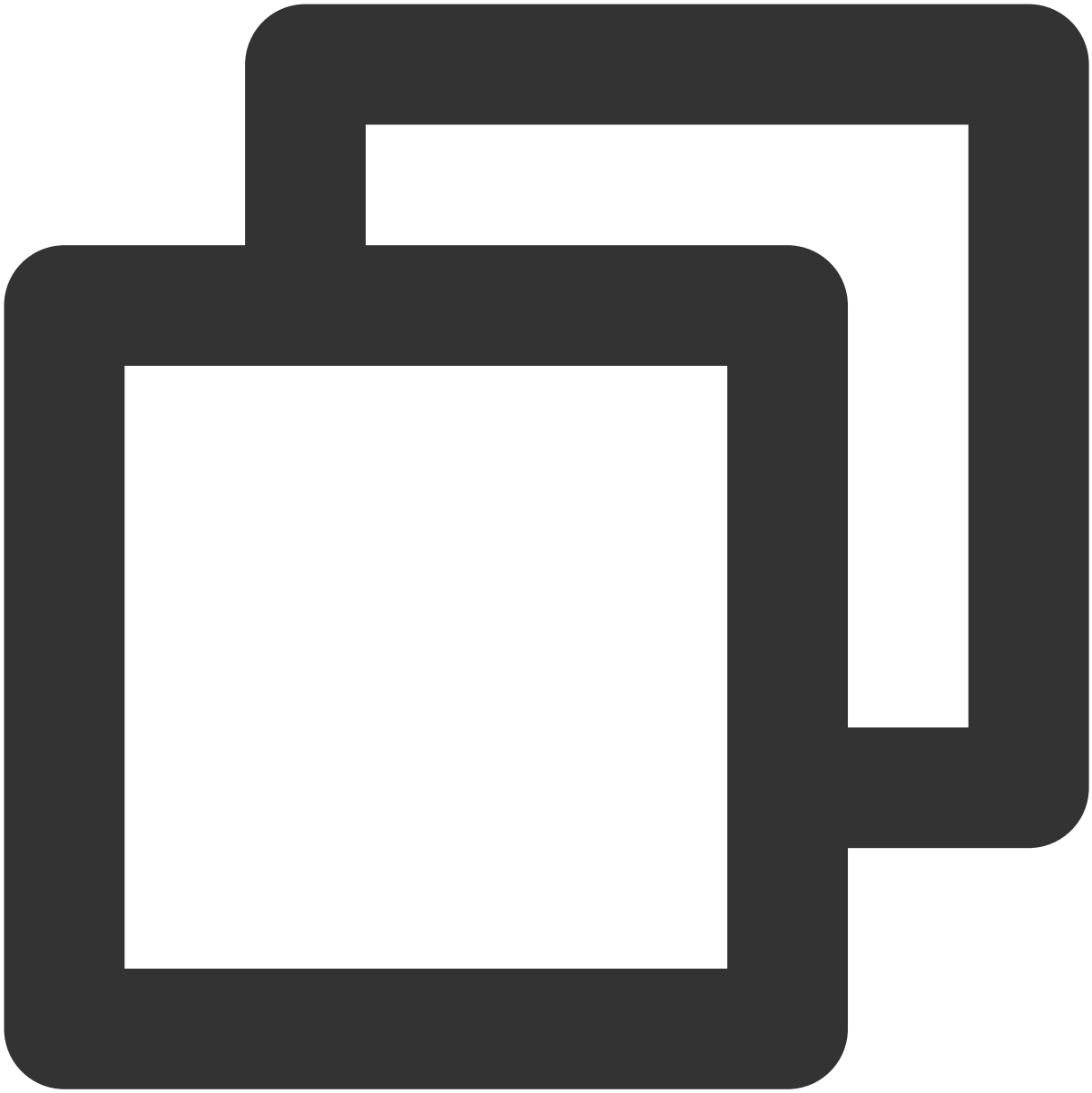


```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
    ( col_name[:] col_type [ COMMENT col_comment ], ... )
[ COMMENT table_comment ]
[ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
```

参数

`table_identifier` ：支持三段式，catalog.db.name

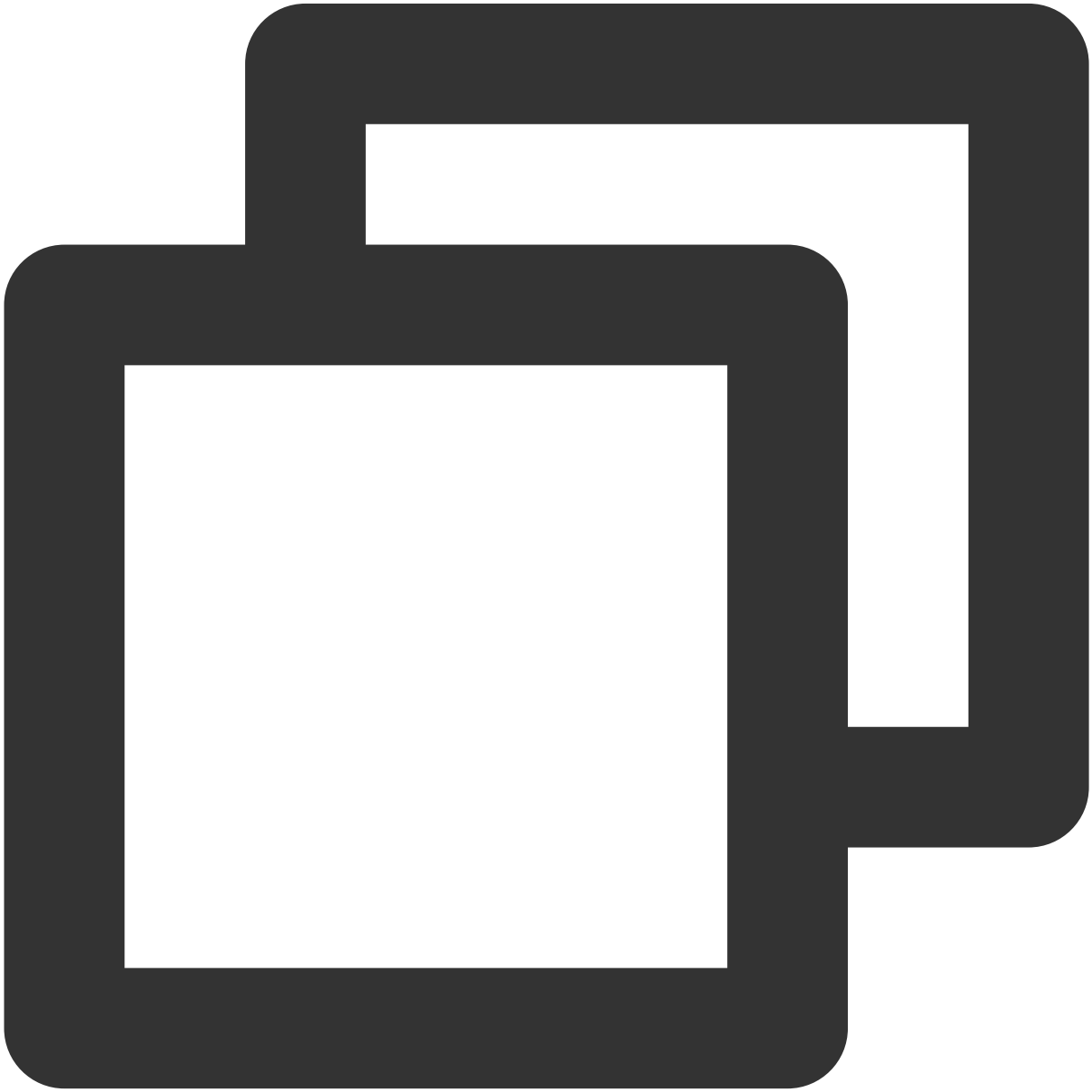
Schemas and Data Types



```
col_type
  : primitive_type
  | nested_type

primitive_type
  : boolean
  | int/integer
  | long/bigint
  | float
  | double
  | decimal(p,s), p=最大位数, s=最大小数点位数, s<=p<=38
```

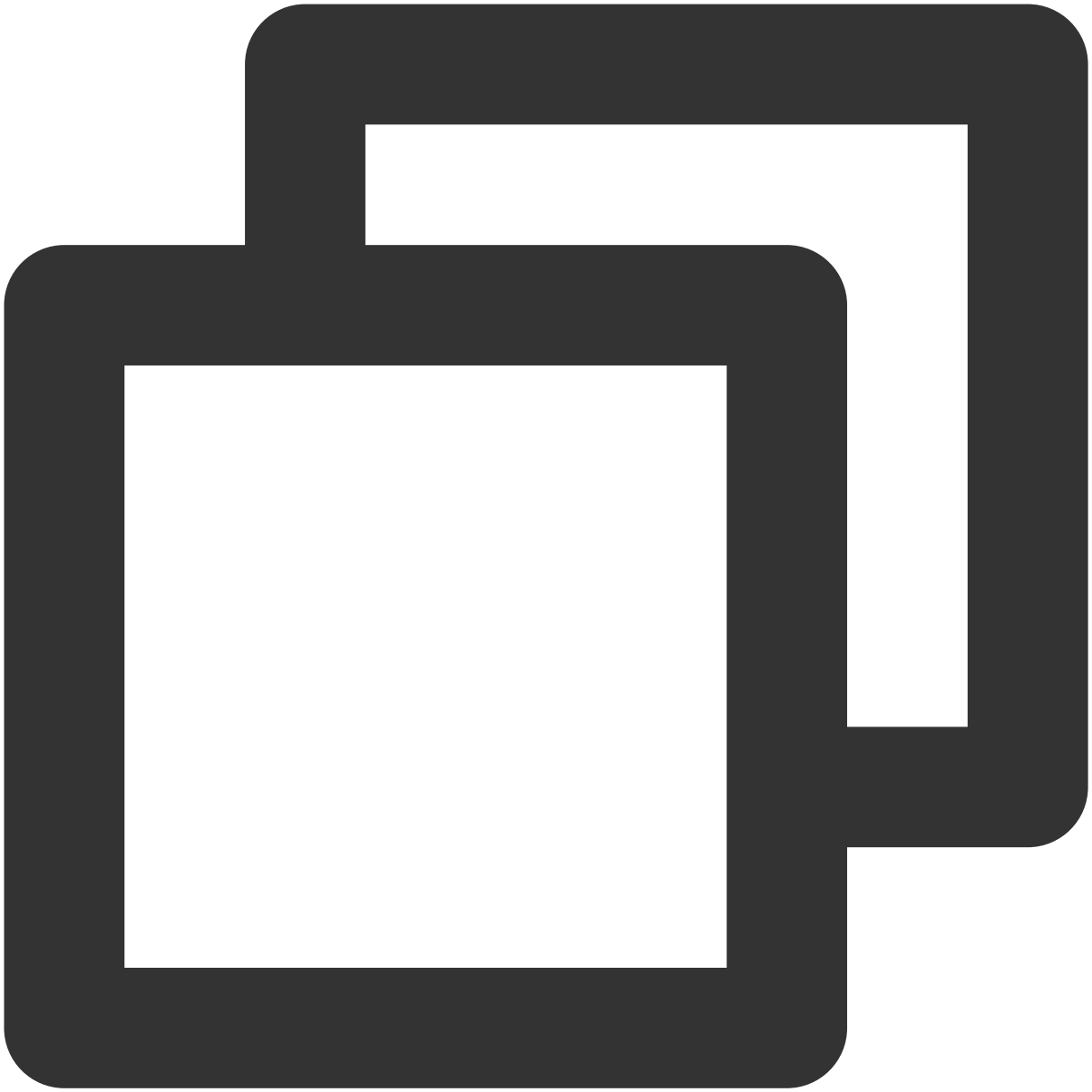
```
    | date
    | timestamp, timestamp with timezone, 不支持time和without timezone
    | string, 也可对应Iceberg uuid类型
    | binary, 也可对应Iceberg fixed类型

  nested_type
    : struct
    | list
    | map
```

Partition Transforms



```
transform
  : identity, 支持任意类型, DLC不支持该转换
  | bucket[N], hash mod N分桶, 支持col_type: int,long, decimal, date, timestamp, stri
  | truncate[L], L截取分桶, 支持col_type: int,long,decimal,string
  | years, 年份, 支持col_type: date,timestamp
  | months, 月份, 支持col_type: date,timestamp
  | days/date, 日期, 支持col_type: date,timestamp
  | hours/date_hour, 小时, 支持col_type: timestamp
```

示例



```
CREATE TABLE dempts(
    id bigint COMMENT 'id number',
    num int,
    eno float,
    dno double,
    cno decimal(9,3),
    flag  boolean,
    data string,
    ts_year timestamp,
    date_month date,
    bno binary,
```

```
    point struct<x: double, y: double>,
    points array<struct<x: double, y: double>>,
    pointmaps map<struct<x: int>, struct<a: int>>
    )
COMMENT 'table documentation'
PARTITIONED BY (bucket(16,id), years(ts_year), months(date_month), identity(bno),
```

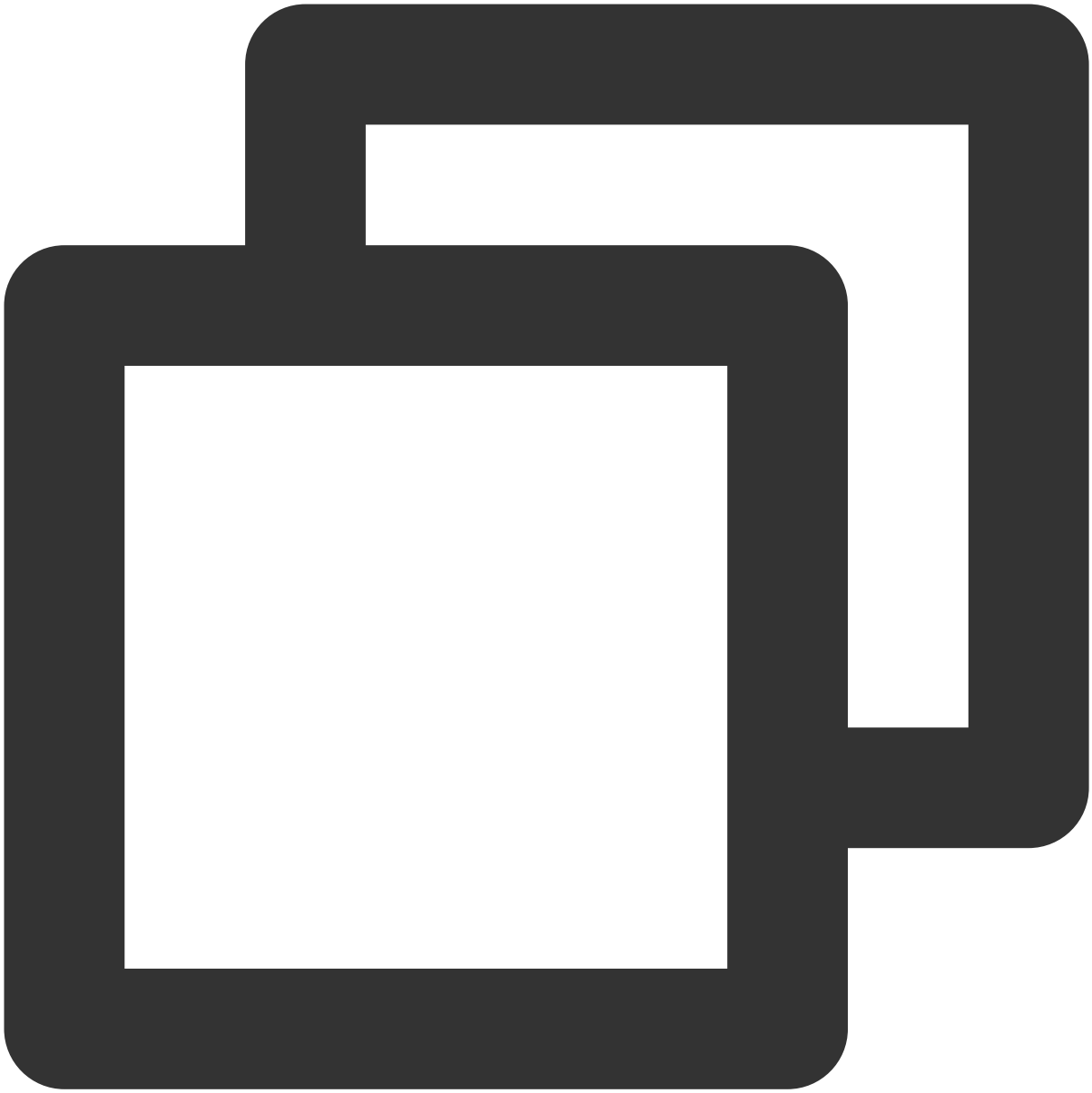
REPLACE TABLE AS SELECT

最近更新时间：2024-08-07 17:12:37

说明

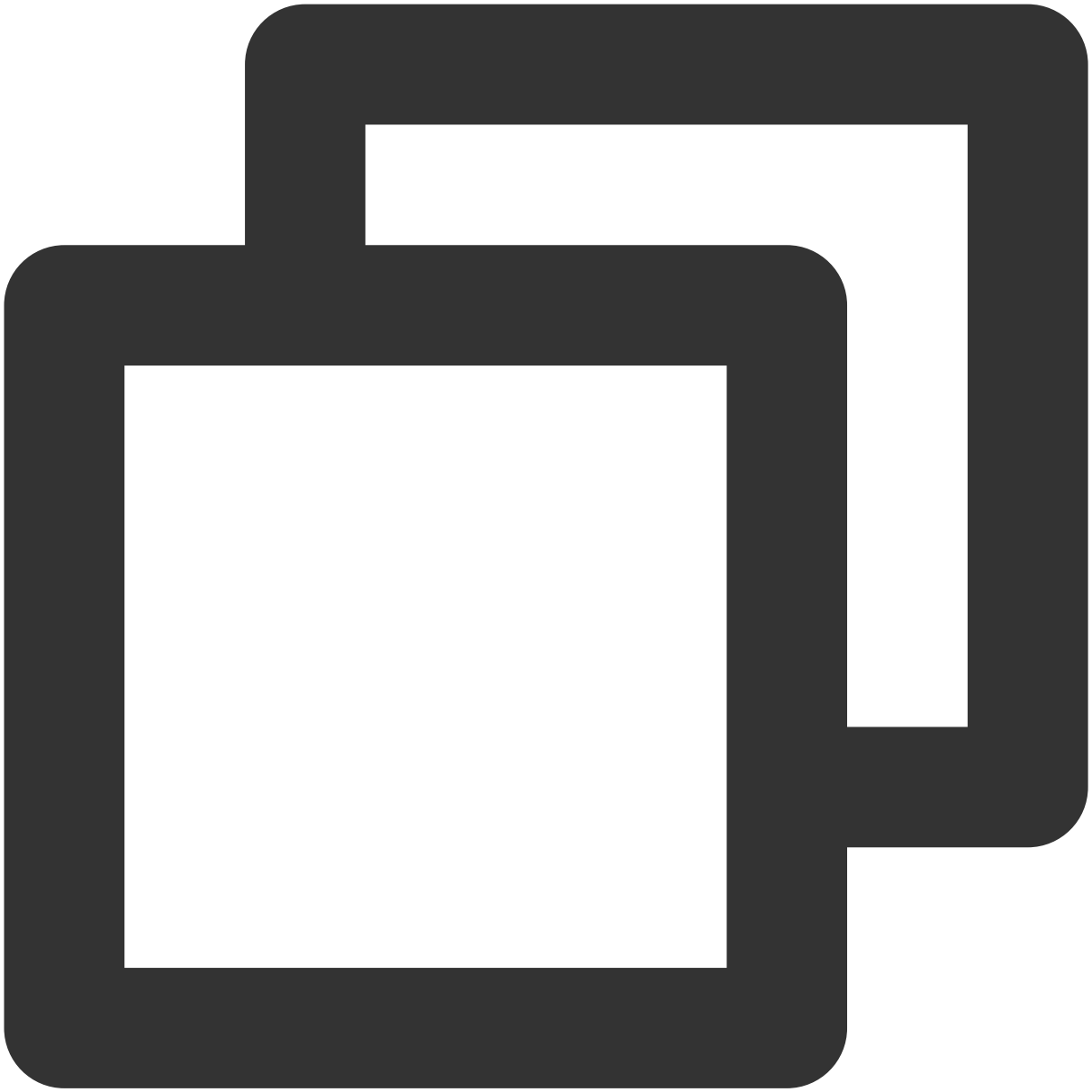
支持内核：SparkSQL。
适用表类型：外部 Iceberg 表、原生 Iceberg 表。
用途：保留表 history 情况下，从目标表更新替换表快照 snapshot。

语法



```
CREATE [OR REPLACE] TABLE table_identifier
USING iceberg
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
    [ LOCATION path ]
    [ TBLPROPERTIES ( property_name=property_value, ... ) ]
AS select_statement
```

示例



```
CREATE OR REPLACE TABLE dempts_replace
USING iceberg
COMMENT 'table create as replace'
PARTITIONED BY (eno, dno)
TBLPROPERTIES ('write.format.default'='avro')
LOCATION '/warehouse/db_001/dempts_replace'
AS SELECT * from dempts;
```

版权所有：腾讯云计算（北京）有限责任公司

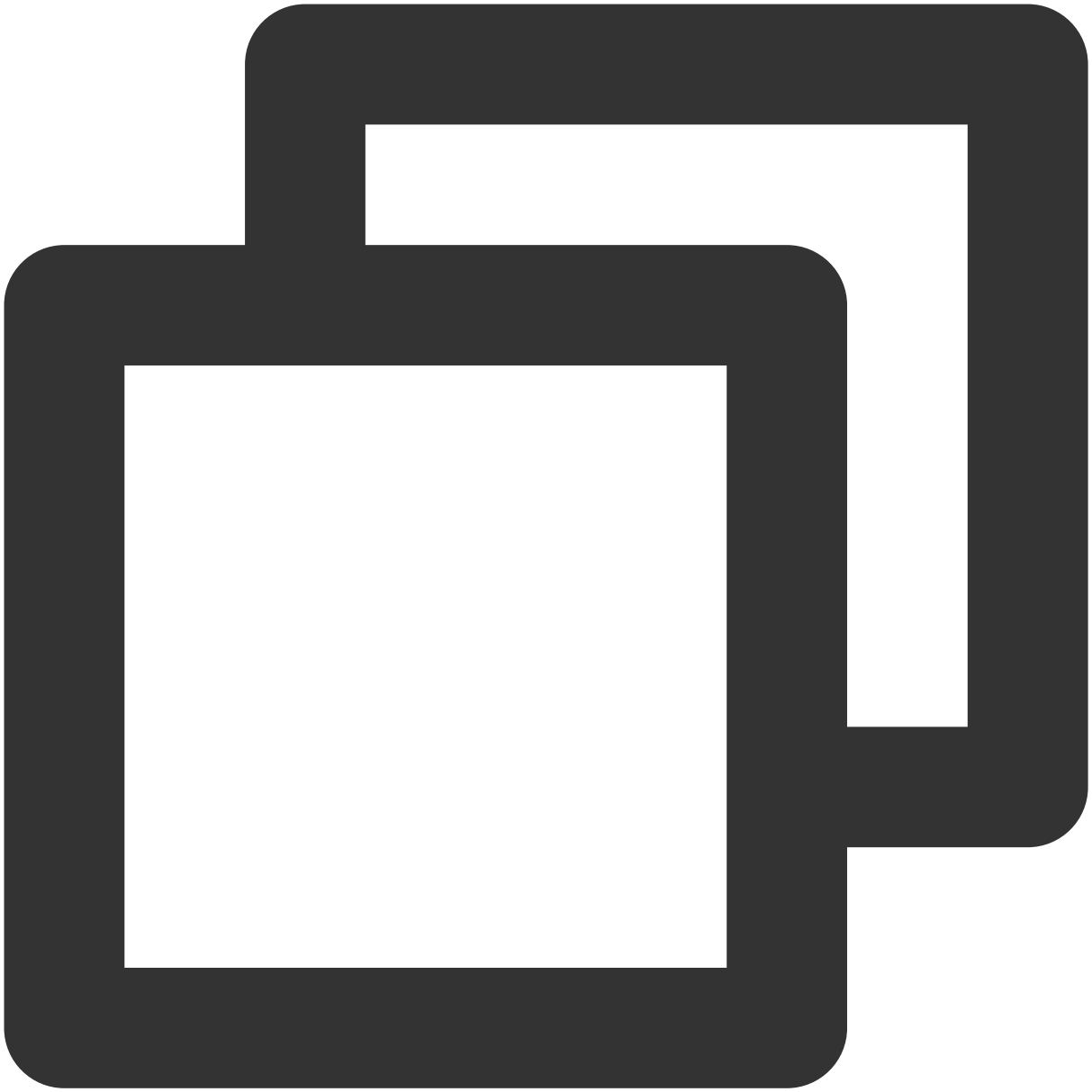
SHOW TABLES

最近更新时间：2024-08-07 17:12:57

说明

支持内核：Presto、SparkSQL。
适用表范围：原生表、外部表。
用途：列出数据库中的所有基表和视图。

语法



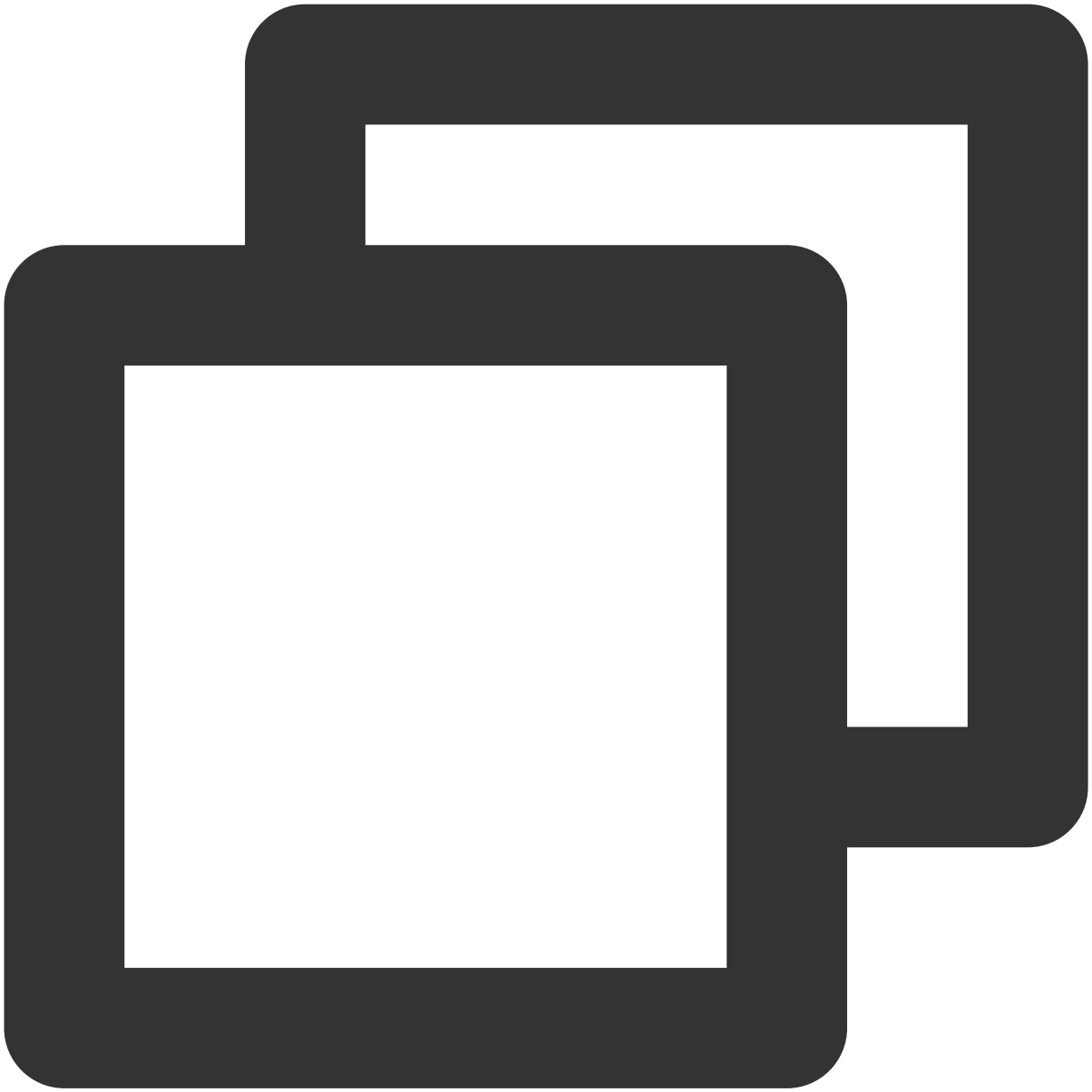
```
SHOW TABLES [IN database_name] ['regular_expression']
```

参数

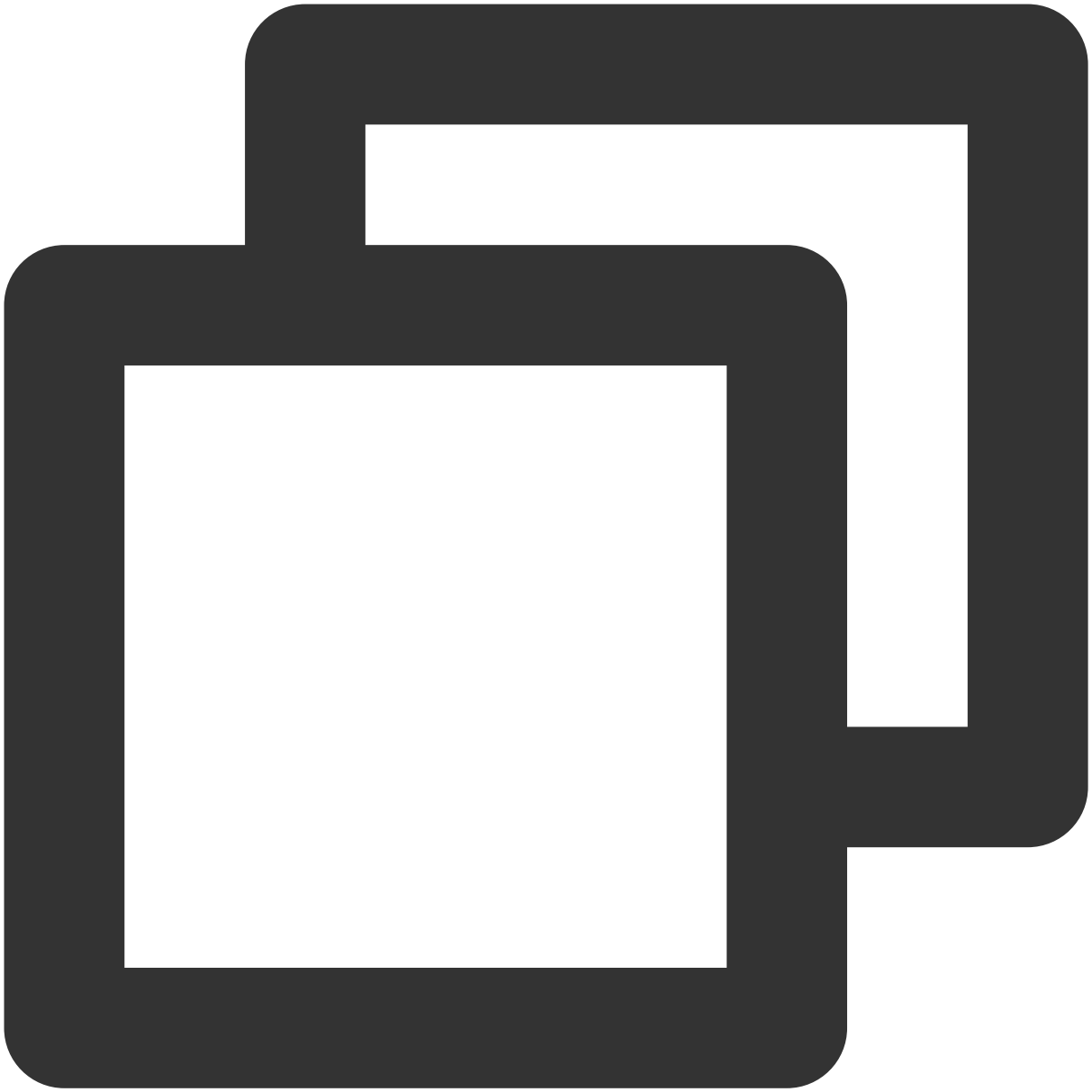
`[IN database_name]` ：指定将从中列出表的数据库名称。如果省略，则假定当前上下文中的数据库。

`['regular_expression']` ：将表列表筛选为与指定的常规表达式匹配的表。只能使用表示任意字符的通配符 *，或表示字符之间。

示例



```
SHOW TABLES IN sampledb;
```



```
SHOW TABLES IN sampledb '*flights*';
```

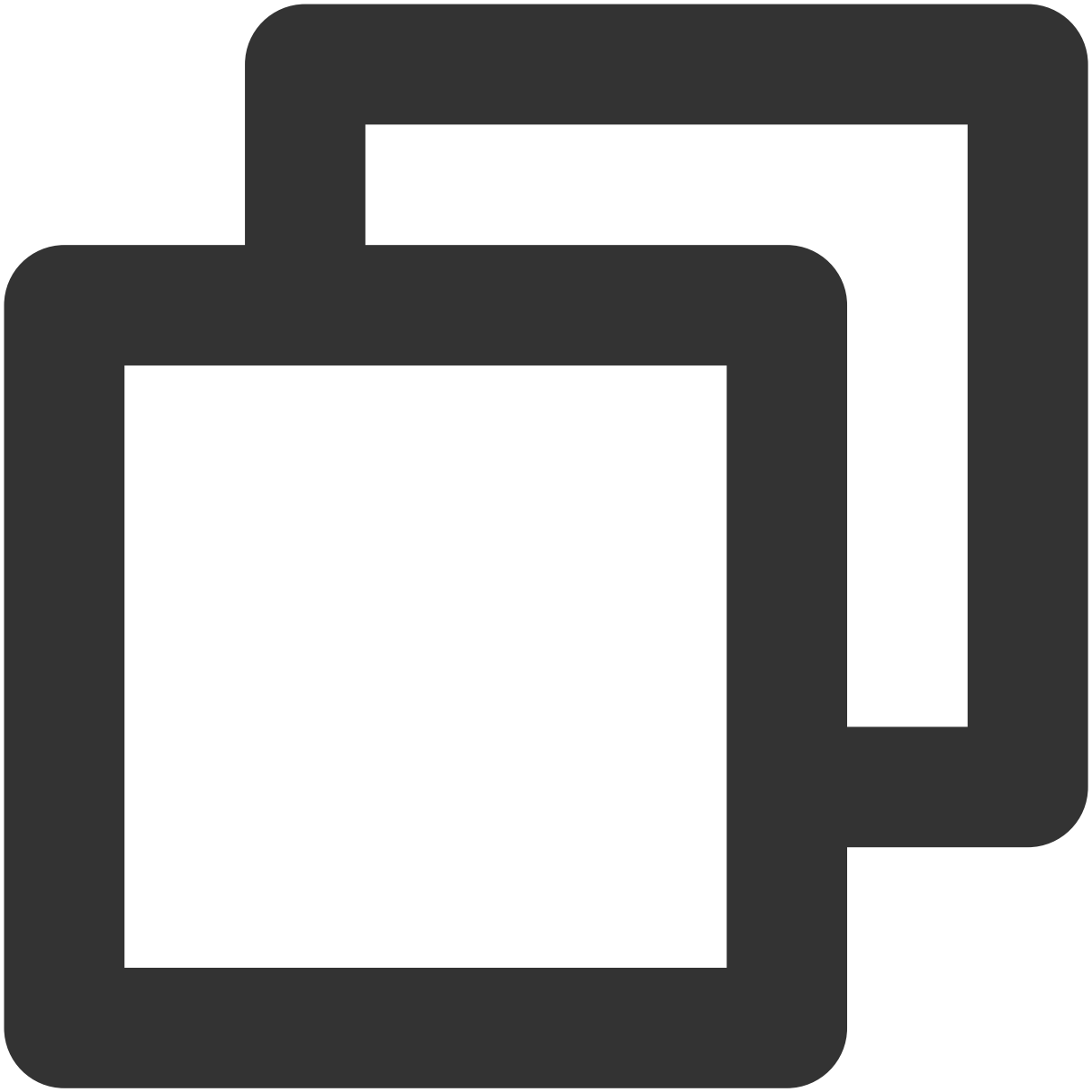
SHOW CREATE TABLE

最近更新时间：2024-08-07 17:13:14

说明

支持内核：Presto、SparkSQL。
适用表范围：原生表、外部表。
用途：分析存在的表 table_name，查询创建信息。

语法



```
SHOW CREATE TABLE [catalog_name.][db_name.]table_name
```

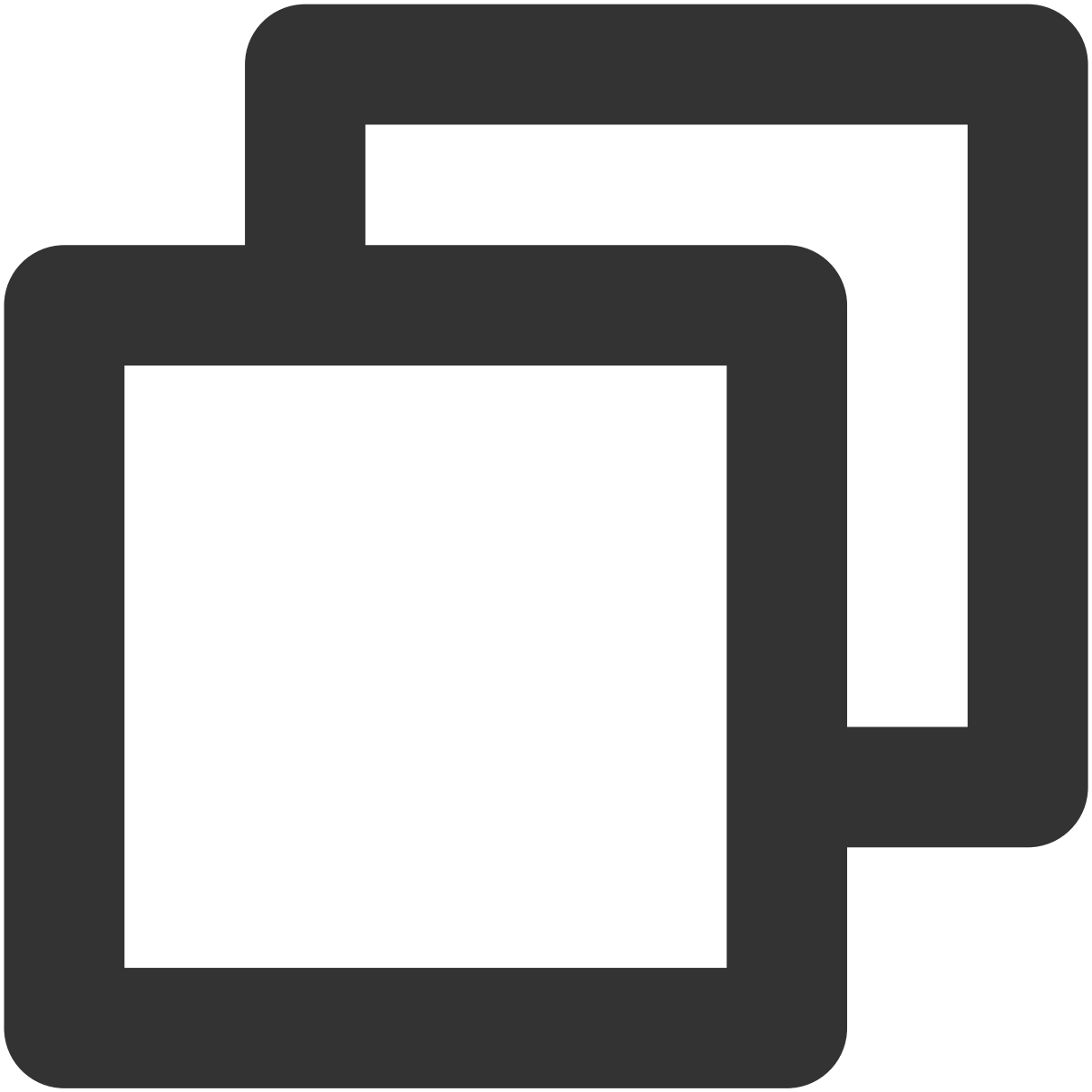
参数

`TABLE [db_name.]table_name`

`db_name` ：数据库名称。

`table_name` ：表名称。

示例



```
SHOW CREATE TABLE tbl;
```

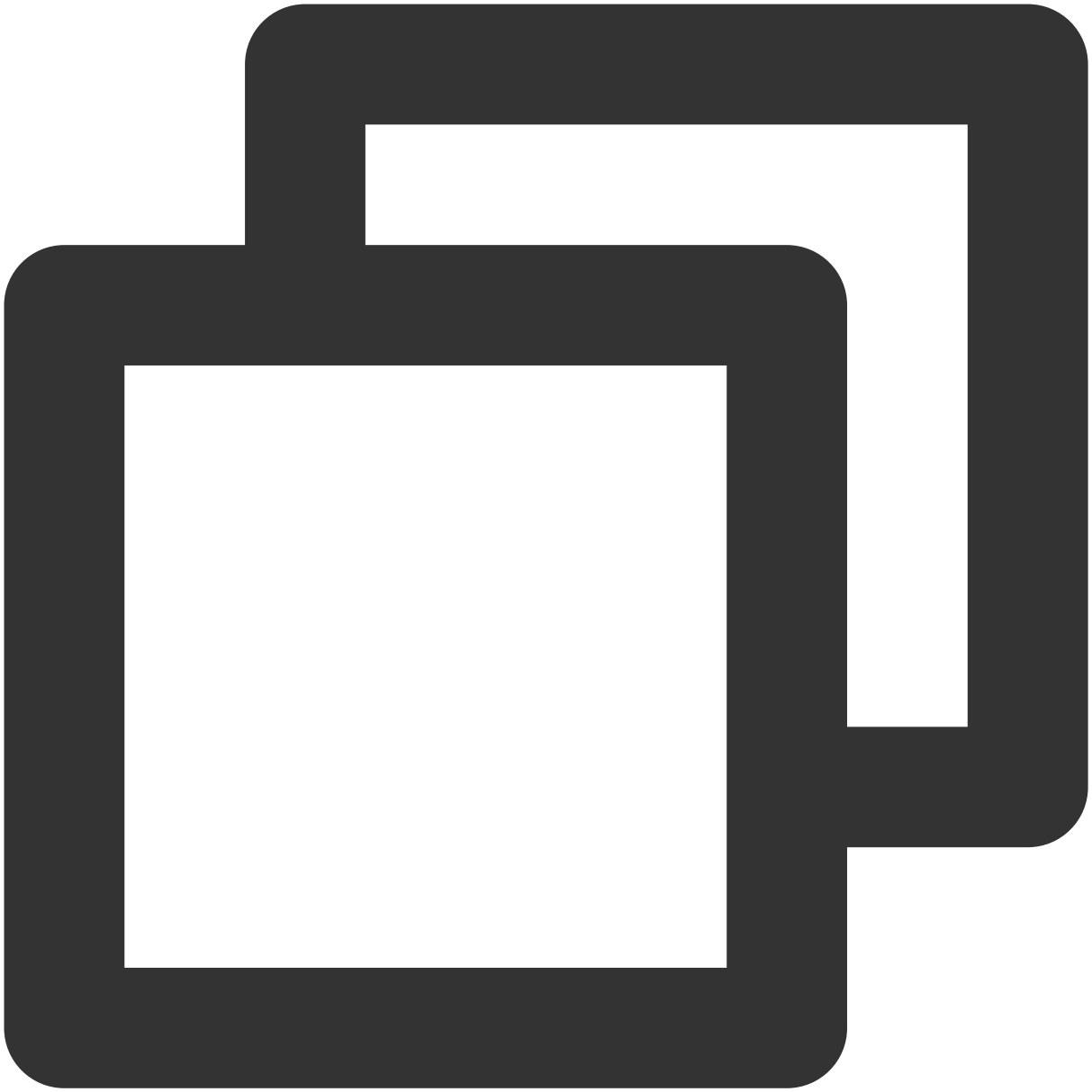
SHOW TBLPROPERTIES

最近更新时间：2024-08-07 17:13:27

说明

支持内核：Presto、SparkSQL。
适用表范围：原生表、外部表。
用途：列出命名表的表属性。

语法

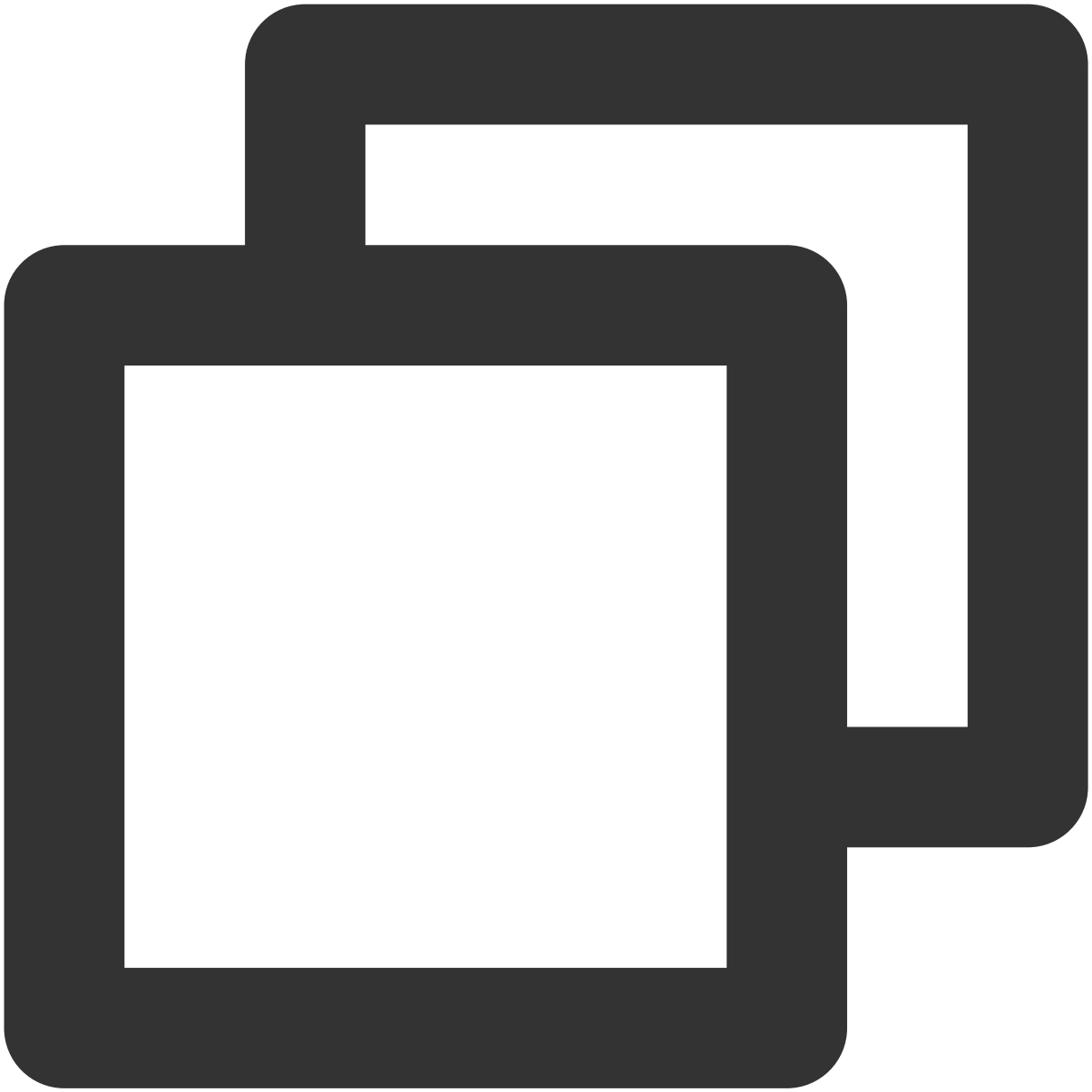


```
SHOW TBLPROPERTIES table_name [('property_name')]
```

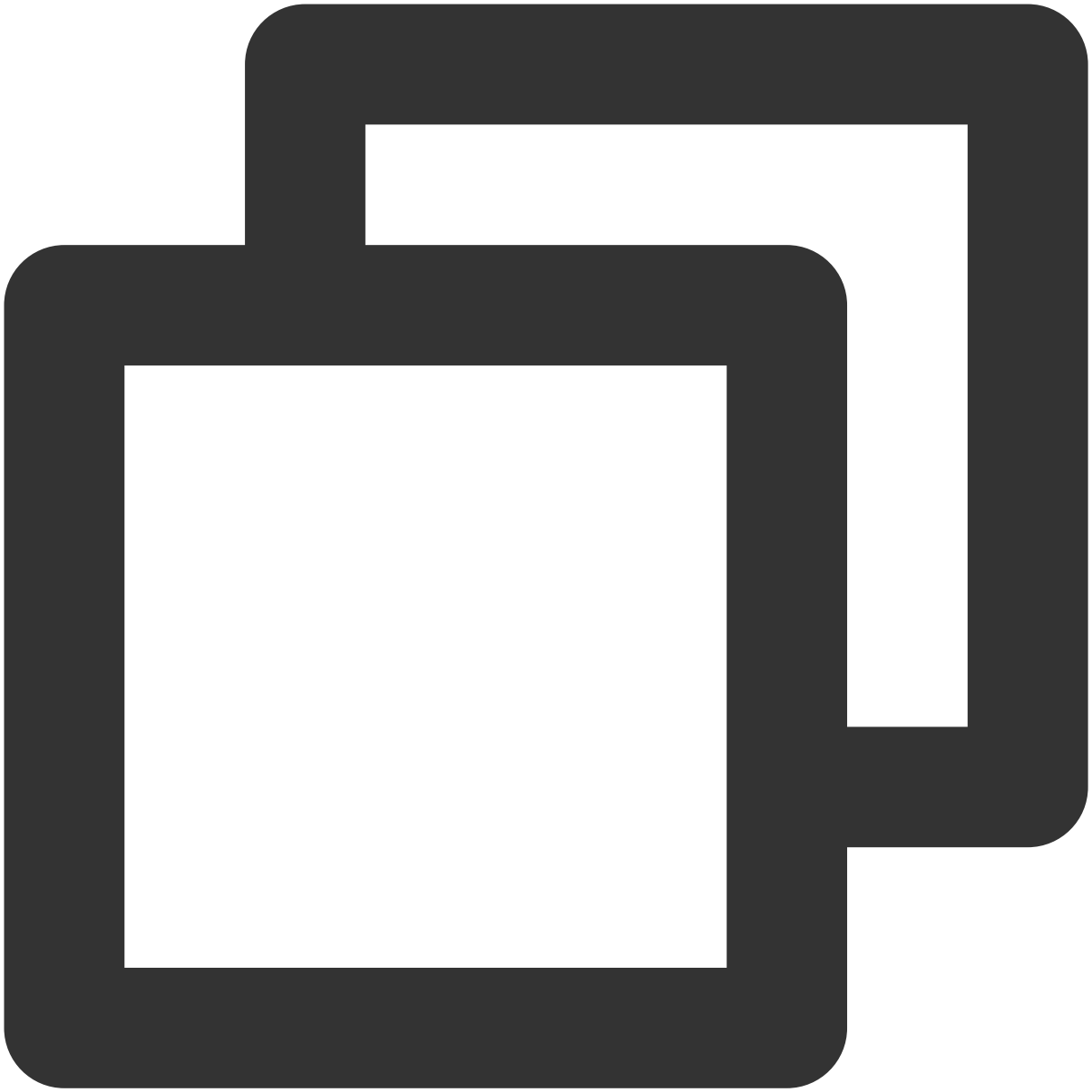
参数

`[('property_name')]` ：如果包含，则只列出属性 name 和 value 值。

示例



```
SHOW TBLPROPERTIES tb1;
```



```
SHOW TBLPROPERTIES orders('tb1');
```

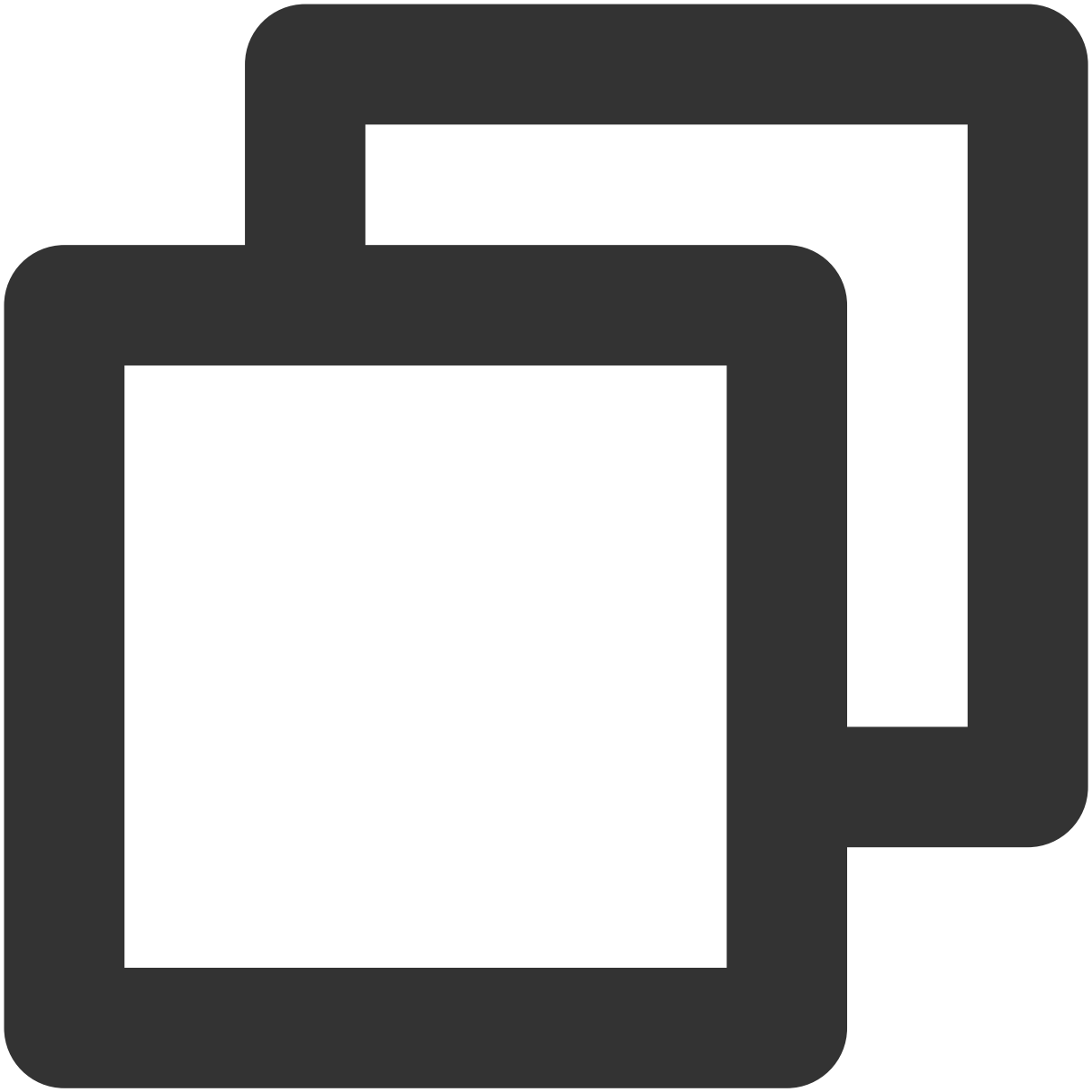
DESCRIBE TABLE

最近更新时间：2024-08-07 17:13:40

说明

支持内核：Presto、SparkSQL。
适用表范围：原生表、外部表。
用途：查看数据表列信息及元数据信息。

标准语法



```
DESCRIBE [EXTENDED | FORMATTED] [db_name.]table_name [PARTITION partition_spec];
```

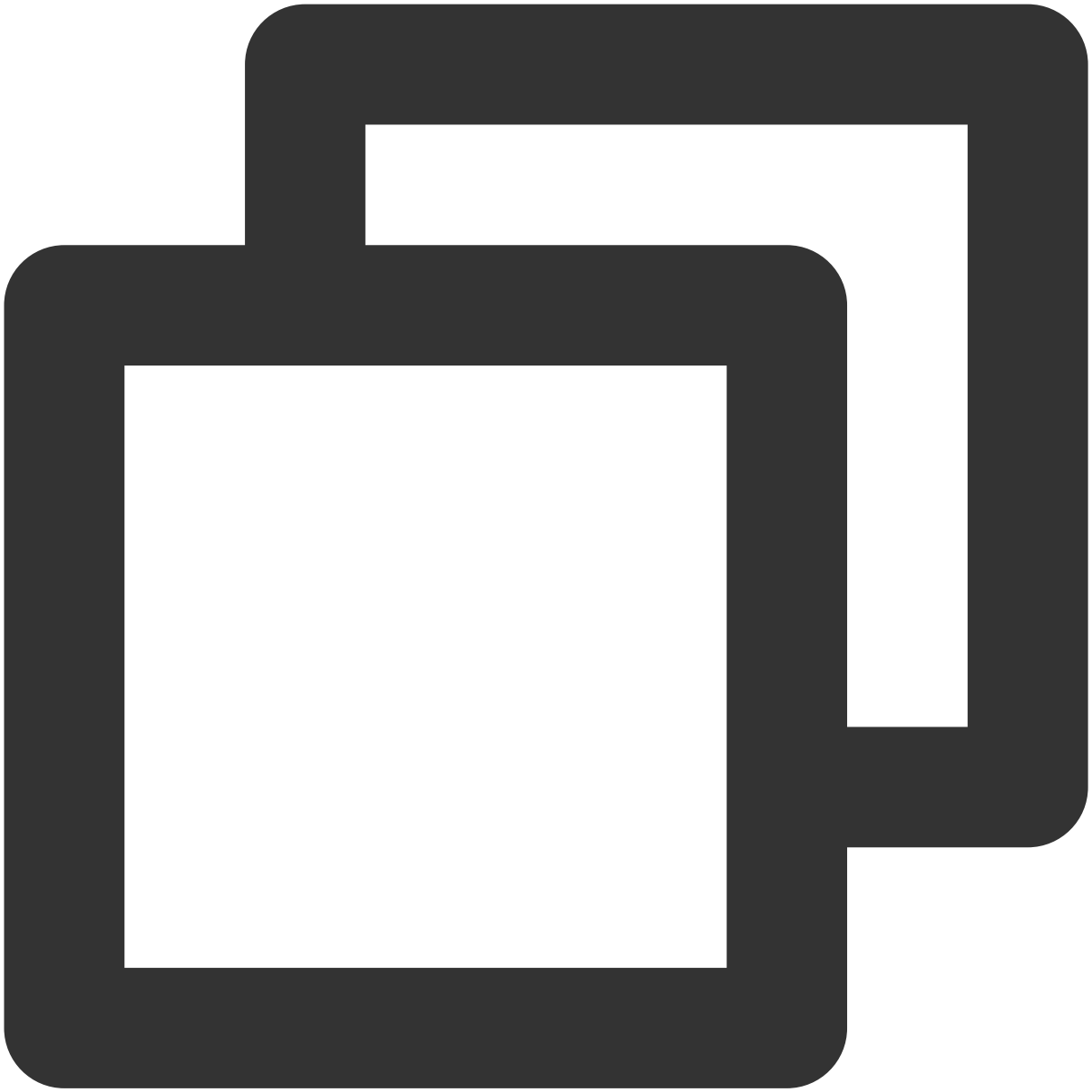
参数

`[EXTENDED | FORMATTED]` ：指定表的格式化形式。

`table_name` ：需要的表名字。

`[PARTITION partition_spec]` ：该表的分区列表。

示例



```
DESCRIBE tbl;
DESCRIBE FORMATTED tbl PARTITION (date_id = '2019-01-07');
```

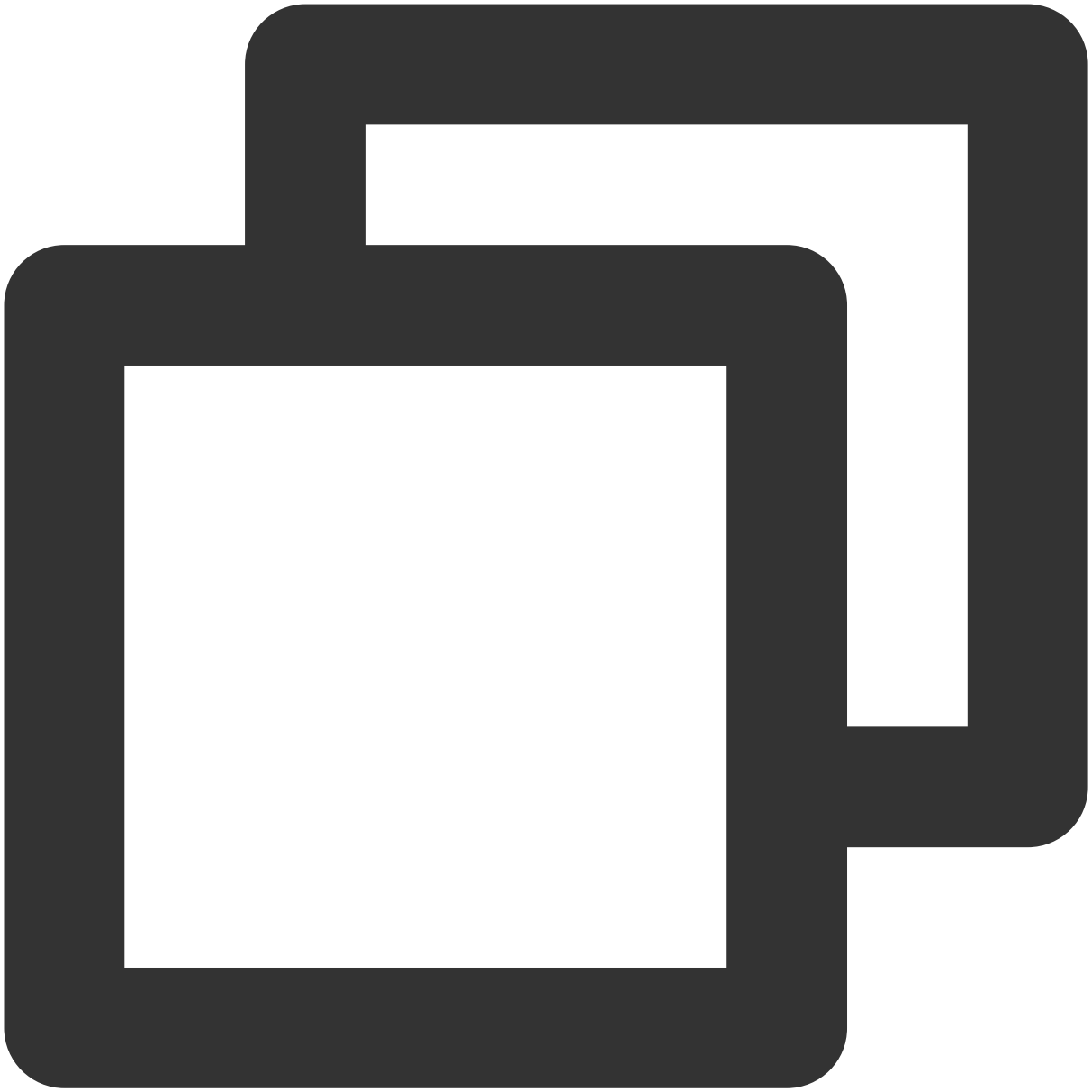
SHOW COLUMNS IN TABLE

最近更新时间：2024-08-07 17:55:41

说明

支持内核：Presto、SparkSQL。
适用表范围：原生表、外部表。
用途：查看数据表的列信息。

语法

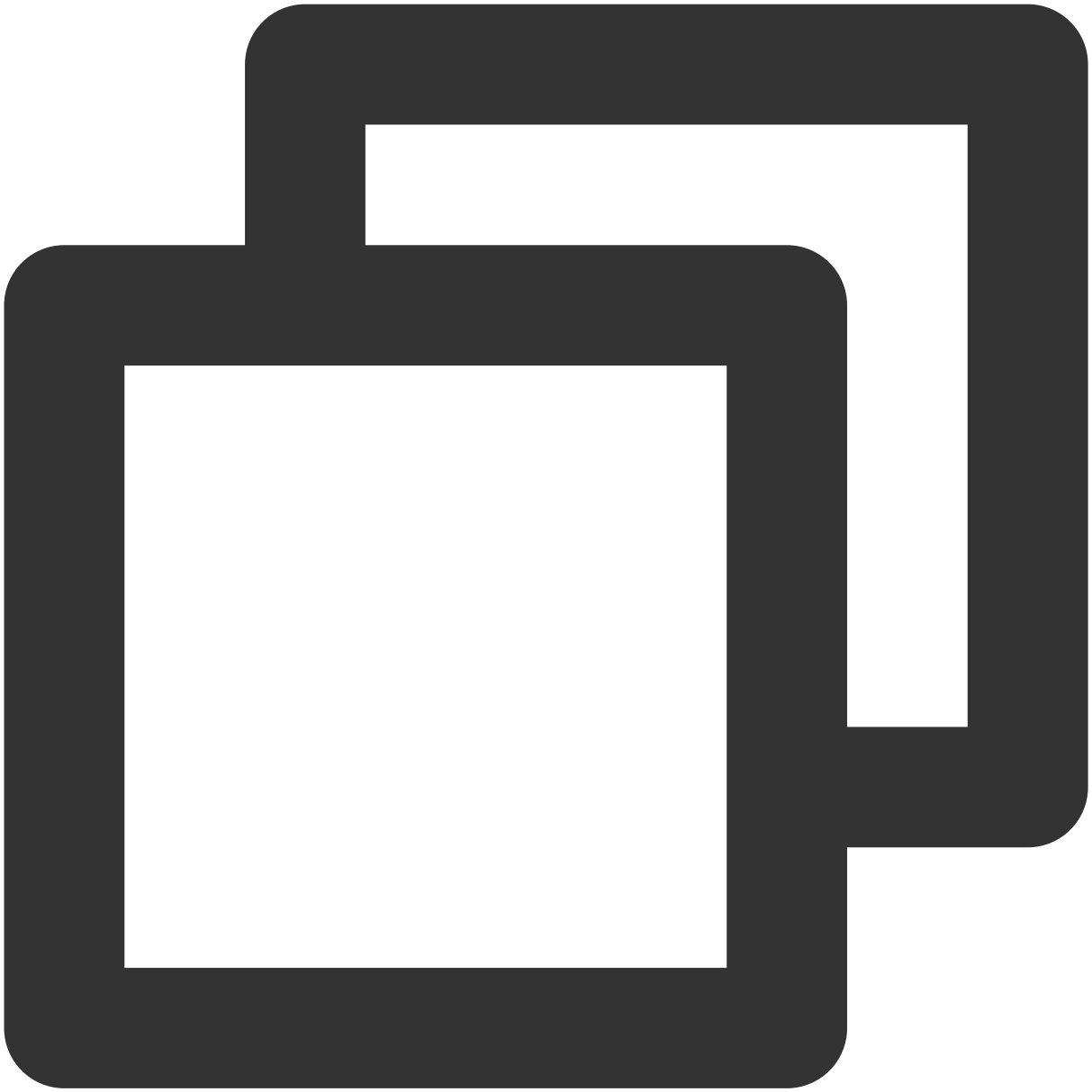


```
SHOW COLUMNS IN table_name
```

参数

`table_name` ：数据表名称。

示例



```
SHOW COLUMNS IN clicks;
```

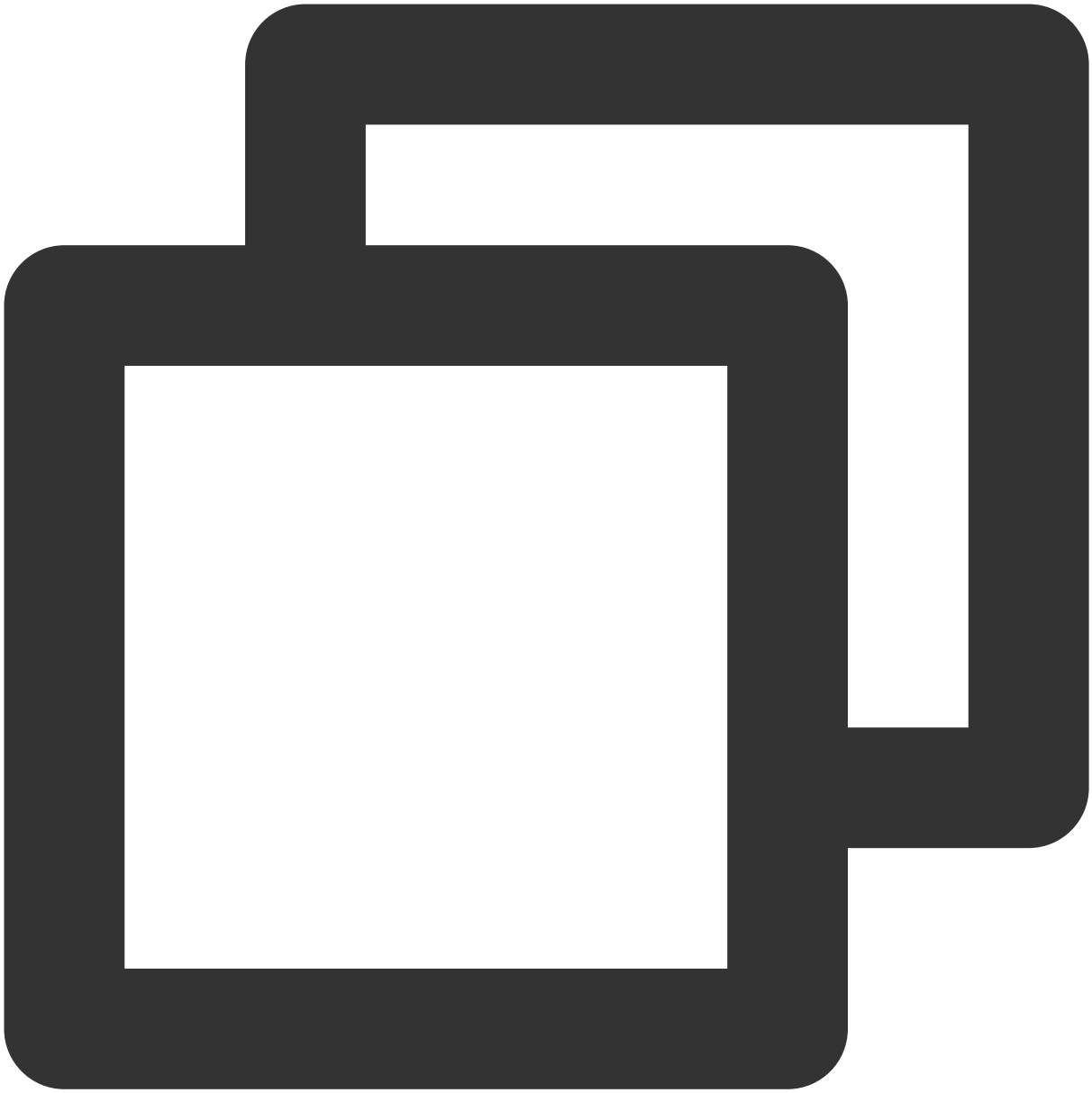
ALTER TABLE
ALTER TABLE ADD COLUMNS

最近更新时间：2024-08-07 17:15:08

说明

支持内核：Presto、SparkSQL。
适用表范围：原生 Iceberg 表、外部表。
用途：修改数据表的属性。

标准语法

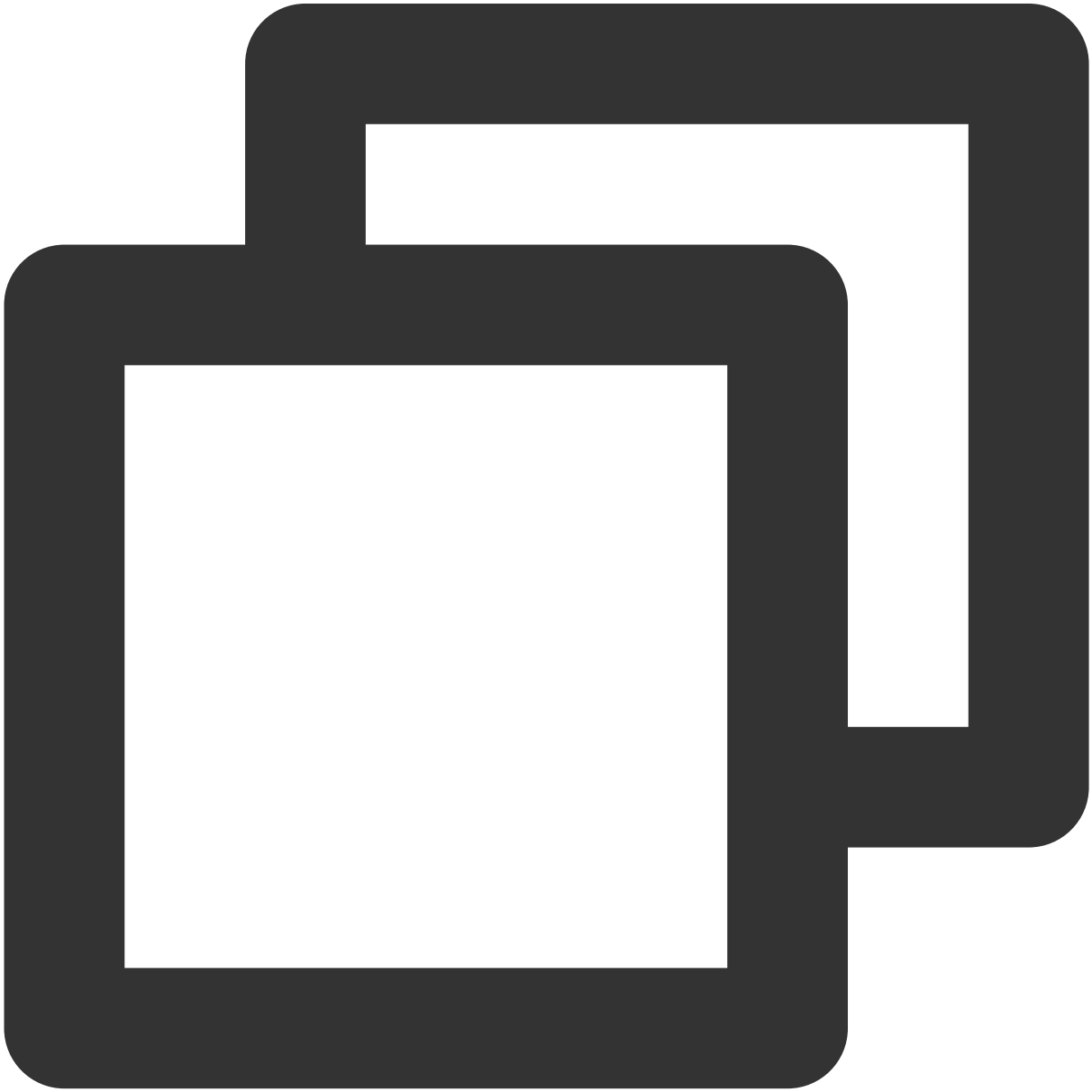


```
ALTER TABLE table_name
 [PARTITION
   (partition_col1_name = partition_col1_value
   [,partition_col2_name = partition_col2_value][,...])]
  ADD COLUMNS (col_name data_type) [RESTRICT | CASCADE]
```

参数

`table_name` ：需要的表名字。
`partition_col1_name`：分区名。
`partition_col1_value` ：分区值。
`col_name` ：添加的列名。
`data_type` ：添加的列类型。

示例



```
ALTER TABLE events ADD COLUMNS (eventowner string);

ALTER TABLE events ADD COLUMNS (eventowner string) CASCADE;

//ALTER TABLE PARTITION ADD COLUMNS语法仅支持DLC原生表
ALTER TABLE events PARTITION (year='2021') ADD COLUMNS (event string);
```

注意

如果创建表的时候采用了 `ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'` 这种格式存储的表时，在创建表之后就不可以进行增加列情况，如果使用 JsonSerDe 方式创建表的时候请注意尽量确认表的结构，如果必须要增加列可以考虑删除表然后重建。

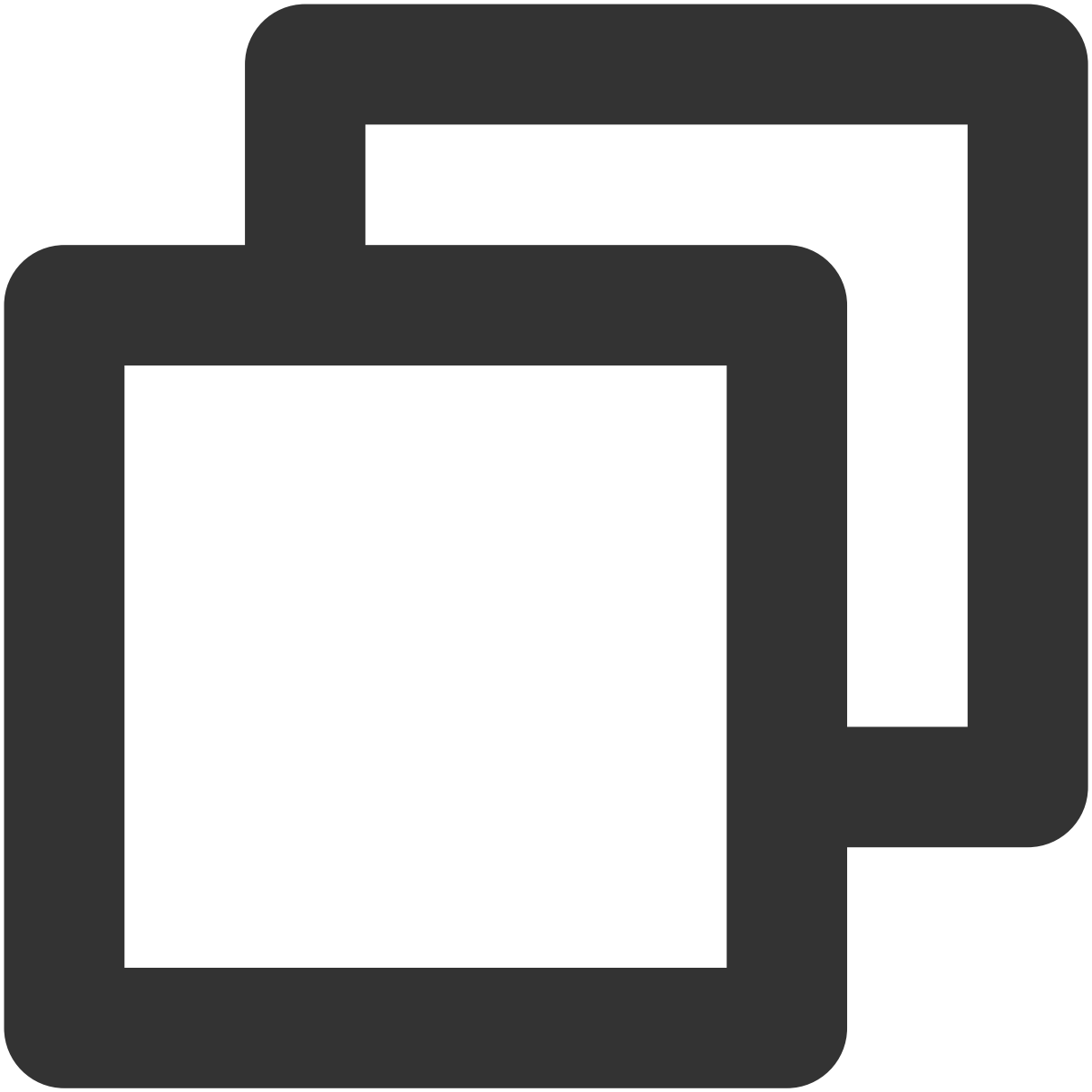
ALTER TABLE ADD COLUMN AFTER/FIRST

最近更新时间：2024-08-07 17:15:22

说明

支持内核：Presto、SparkSQL。
适用表范围：原生 Iceberg 表、外部表。
用途：为数据表添加列。

标准语法



```
ALTER TABLE table_name ADD COLUMN column_name1 column_type [COMMENT col_comment] [F
```

参数

`table_name` ：需要修改的表名字。

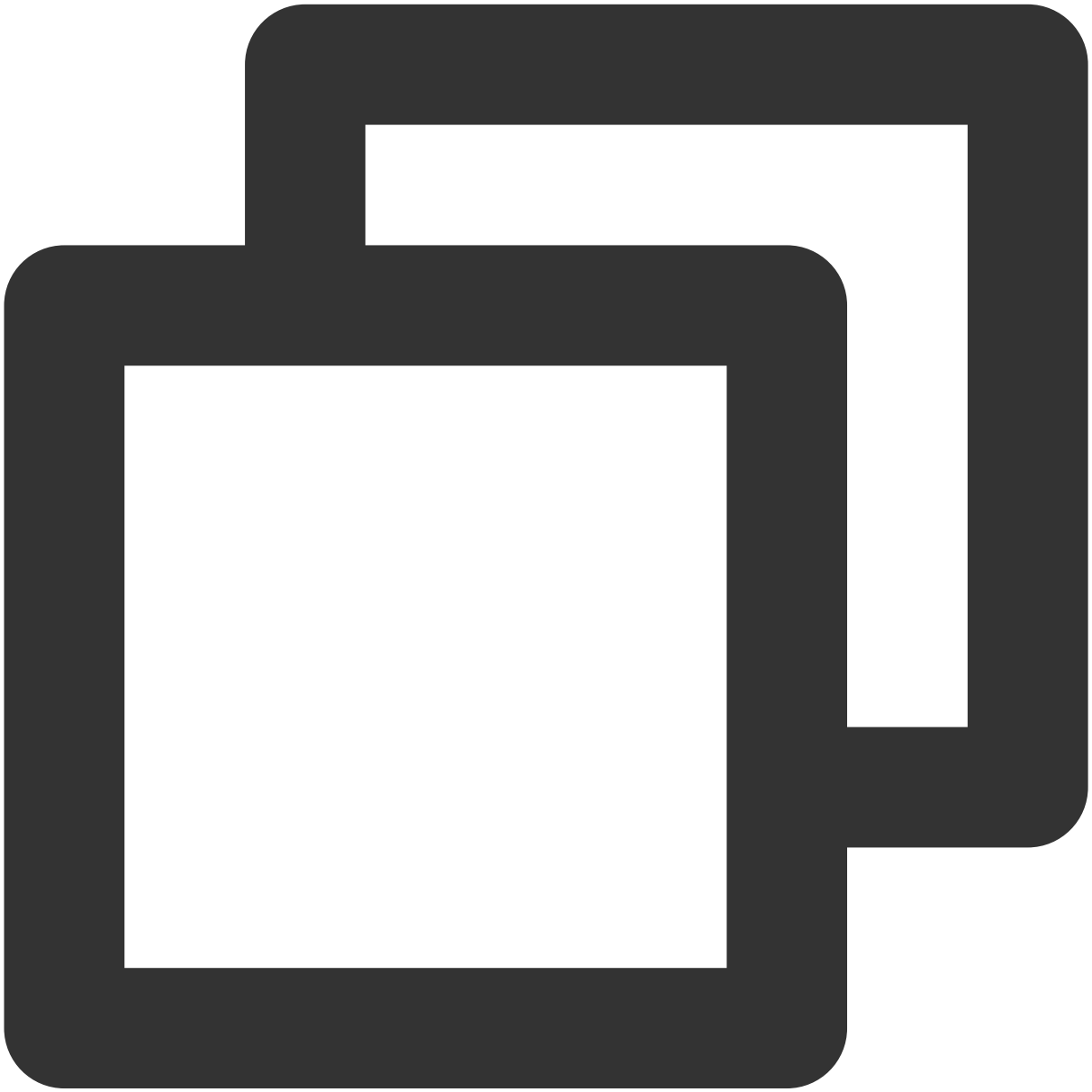
`column_name1` ：需要添加的列。

`column_type` ：添加列的类型。

`col_comment`：添加列的注释。

`column_name2`：添加的列放置到这个列的后面。

示例



```
ALTER TABLE `TBL` ADD COLUMN `COL2` STRING COMMENT 'test' AFTER `COL1`

ALTER TABLE `TBL` ADD COLUMN `COL2` STRING COMMENT 'test' FIRST
```

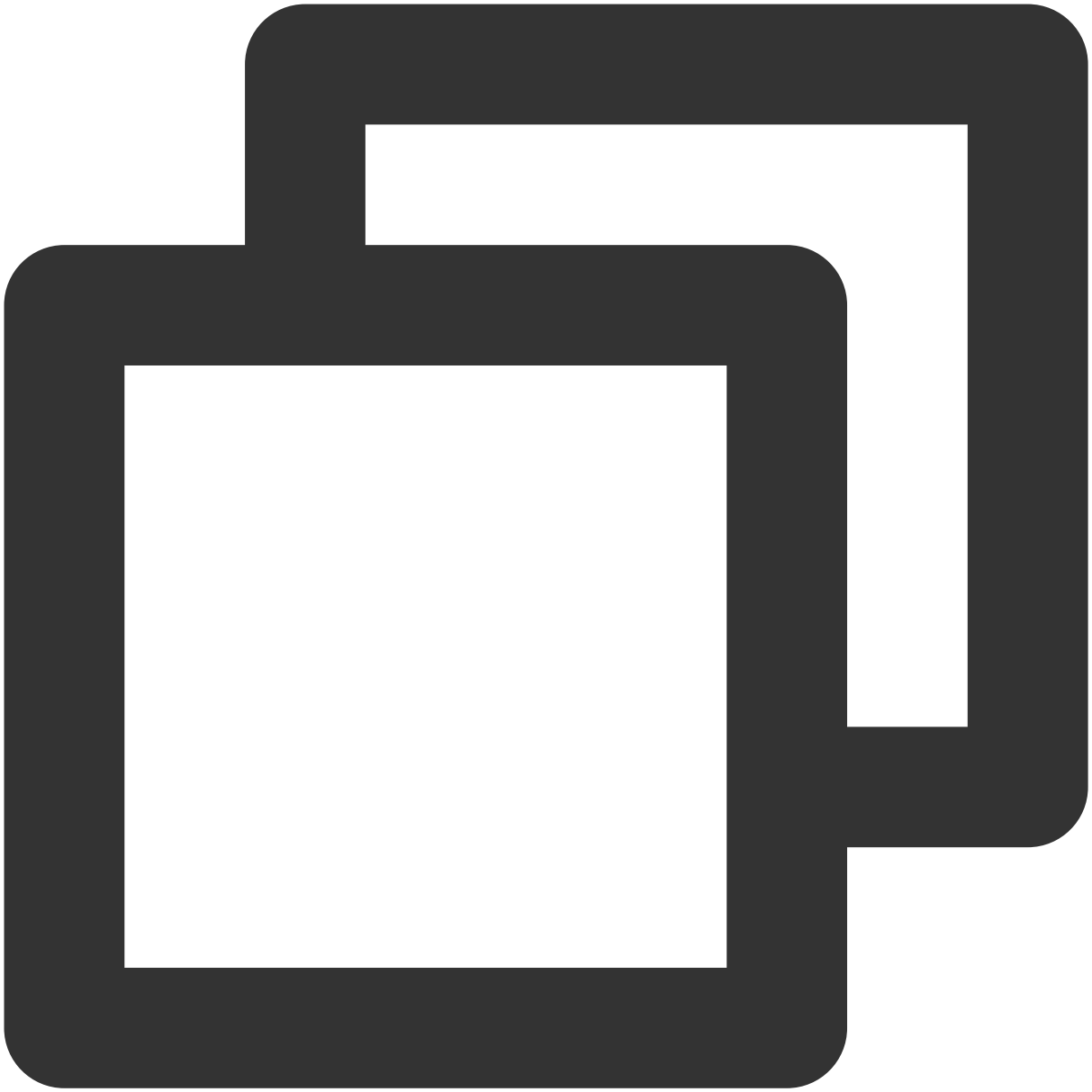
ALTER TABLE DROP COLUMN

最近更新时间：2024-08-07 17:15:50

说明

支持内核：Presto、SparkSQL。
适用表范围：原生 Iceberg 表、外部表。
用途：删除数据表的某字段。

标准语法



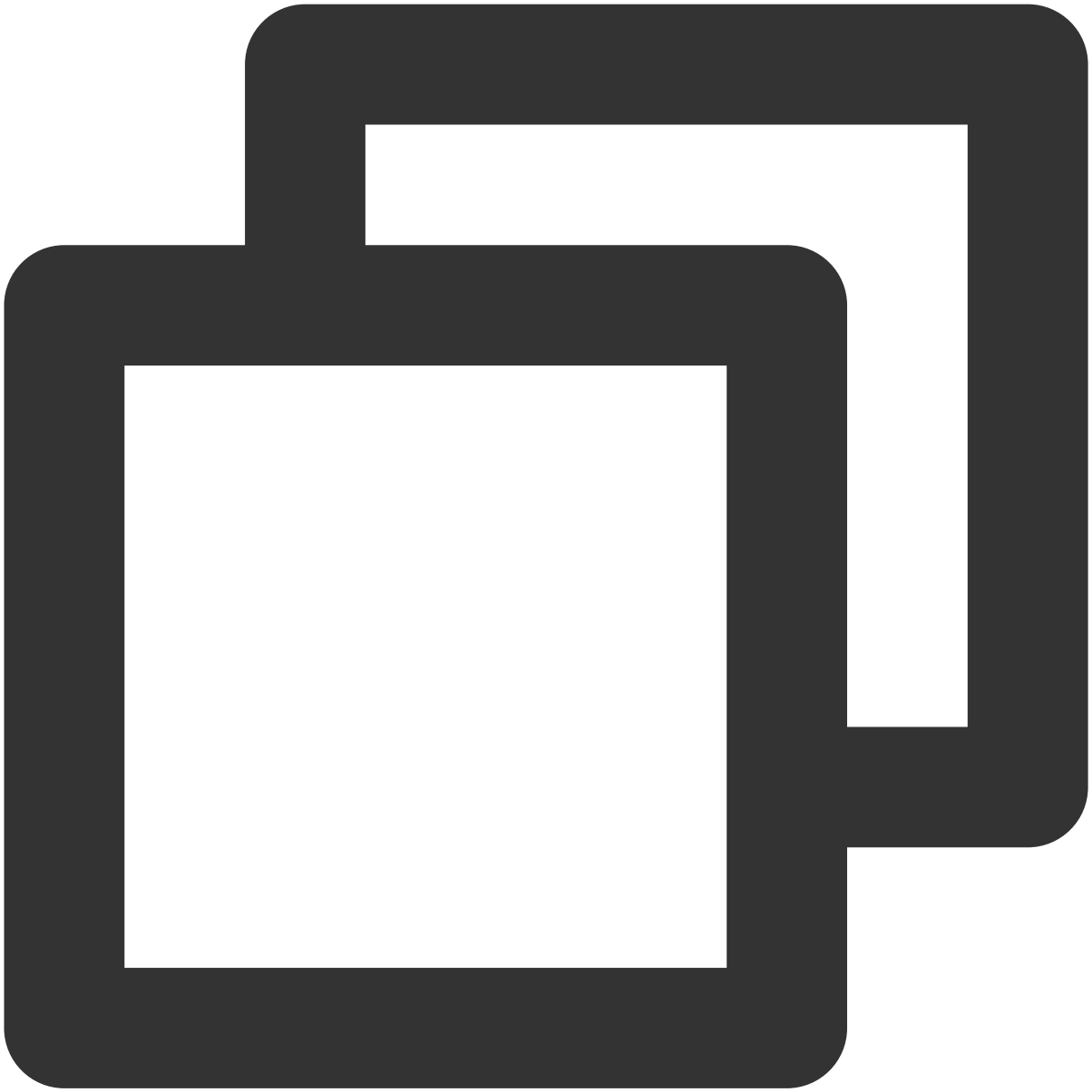
```
ALTER TABLE table_name DROP COLUMN column_name
```

参数

`table_name` ：需要修改的表名字。

`column_name` ：需要删除的列名称。

示例



```
ALTER TABLE `TBL` DROP COLUMN `COL2`
```

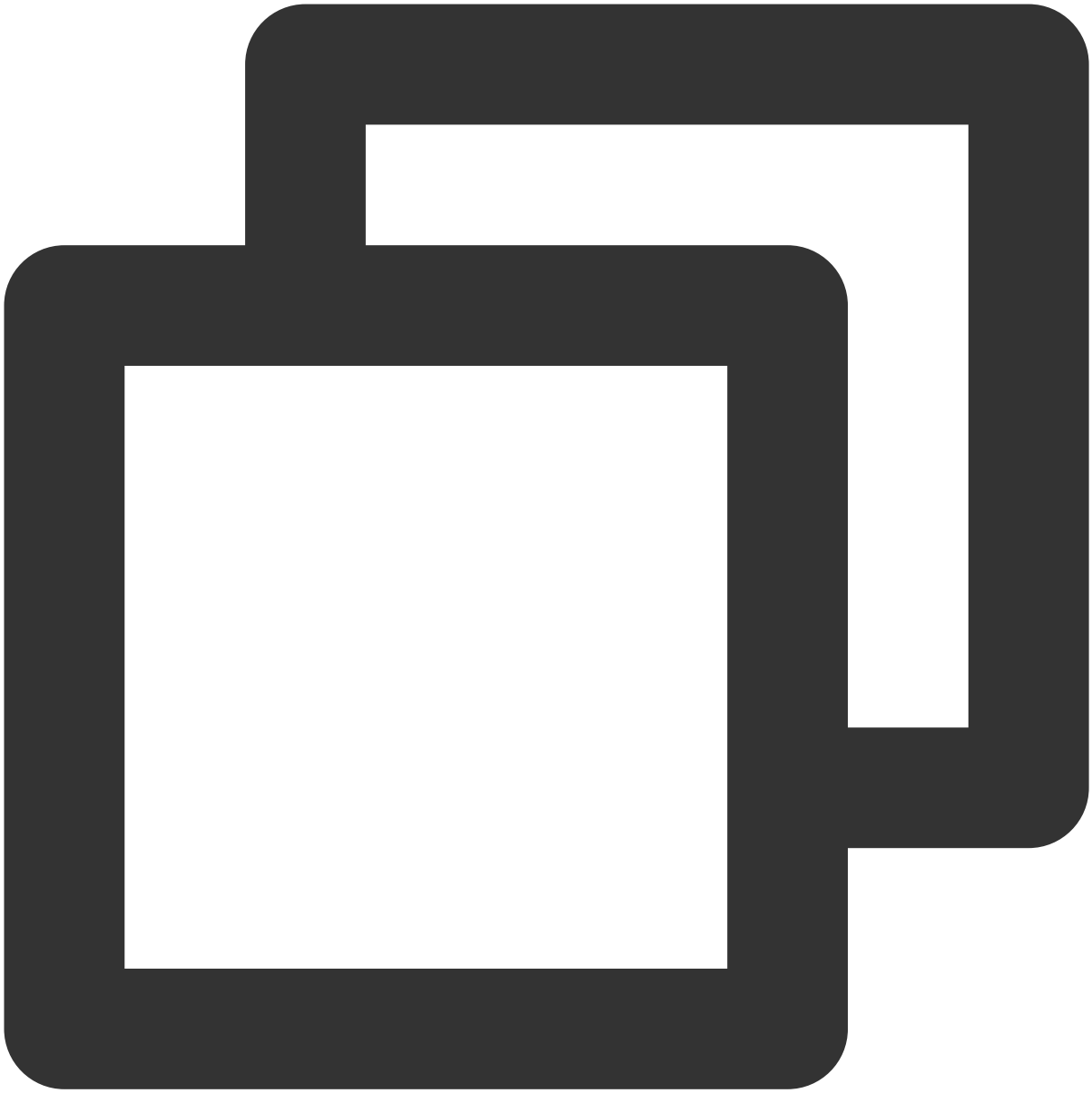
ALTER TABLE ADD PARTATION

最近更新时间：2024-08-07 17:16:01

说明

支持内核：Presto、SparkSQL。
适用表范围：外部表。
用途：为数据表创建一个或多个分区列。

语法



```
ALTER TABLE table_name ADD [IF NOT EXISTS]
  PARTITION (partition_spec)
  [PARTITION (partition_spec) ...]

partition_spec:
  : partition_column = partition_col_value, partition_column = partition_col_value,
```

参数

`table_name` ：需要的表名字。
`partition_column` ：分区名。
`partition_col_value` ：分区值。

示例

```
ALTER TABLE tbl ADD PARTITION (p1=1, p2='a');

ALTER TABLE tbl ADD IF NOT EXISTS PARTITION (P1 = 1) PARTITION (P2 = 2);
```

SHOW PARTITIONS

最近更新时间：2024-08-07 17:16:20

列出表中的所有分区。

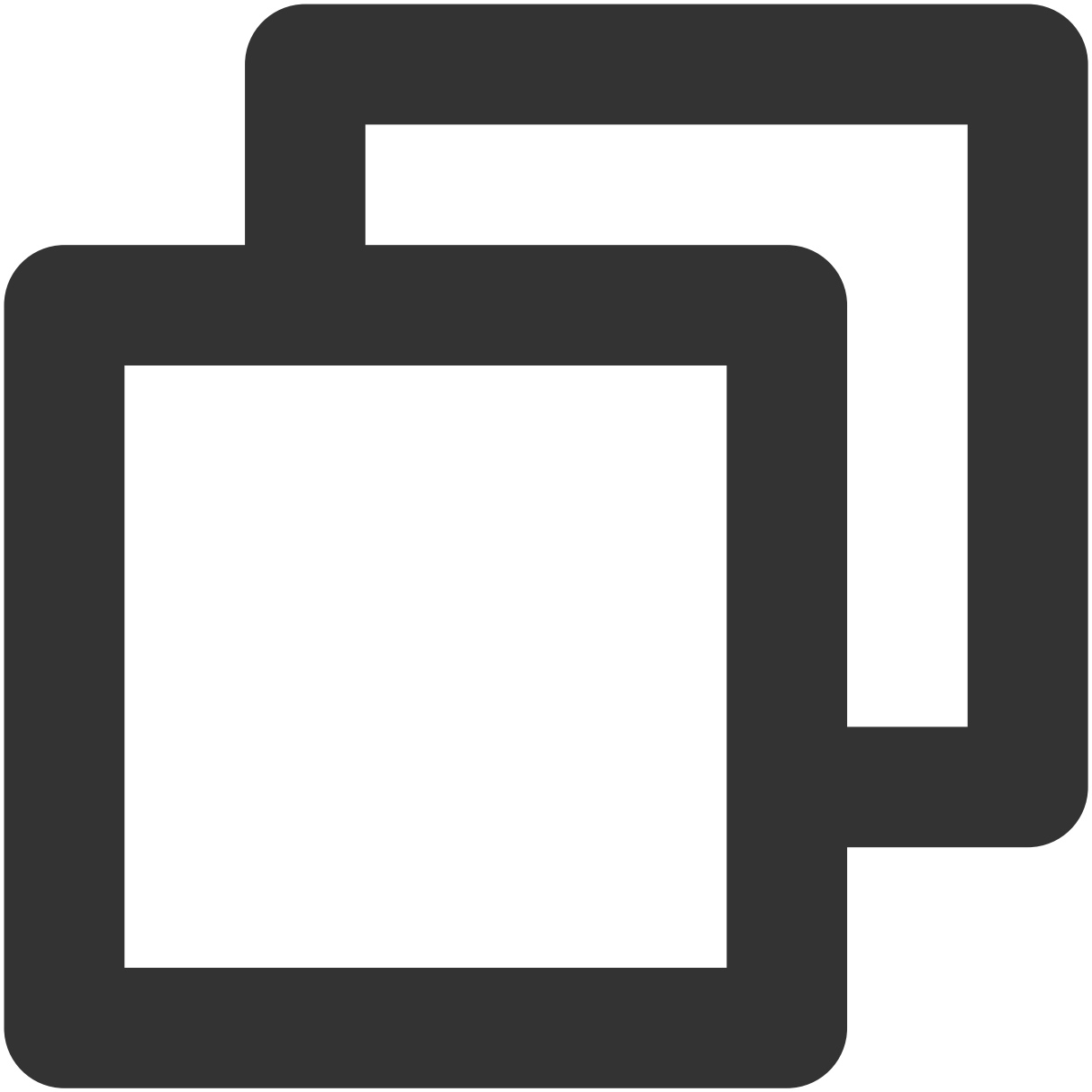
语法

```
SHOW PARTITIONS [db_name.]table_name [PARTITION(partition_spec)];
```

参数

`TABLE [db_name.]table_name` ：表名。
`[PARTITION(partition_spec)]` ：分区列，可以有多个分区列条件。

示例



```
SHOW PARTITIONS db01.table PARTITION(ds='2010-03-03', hr='12');
```

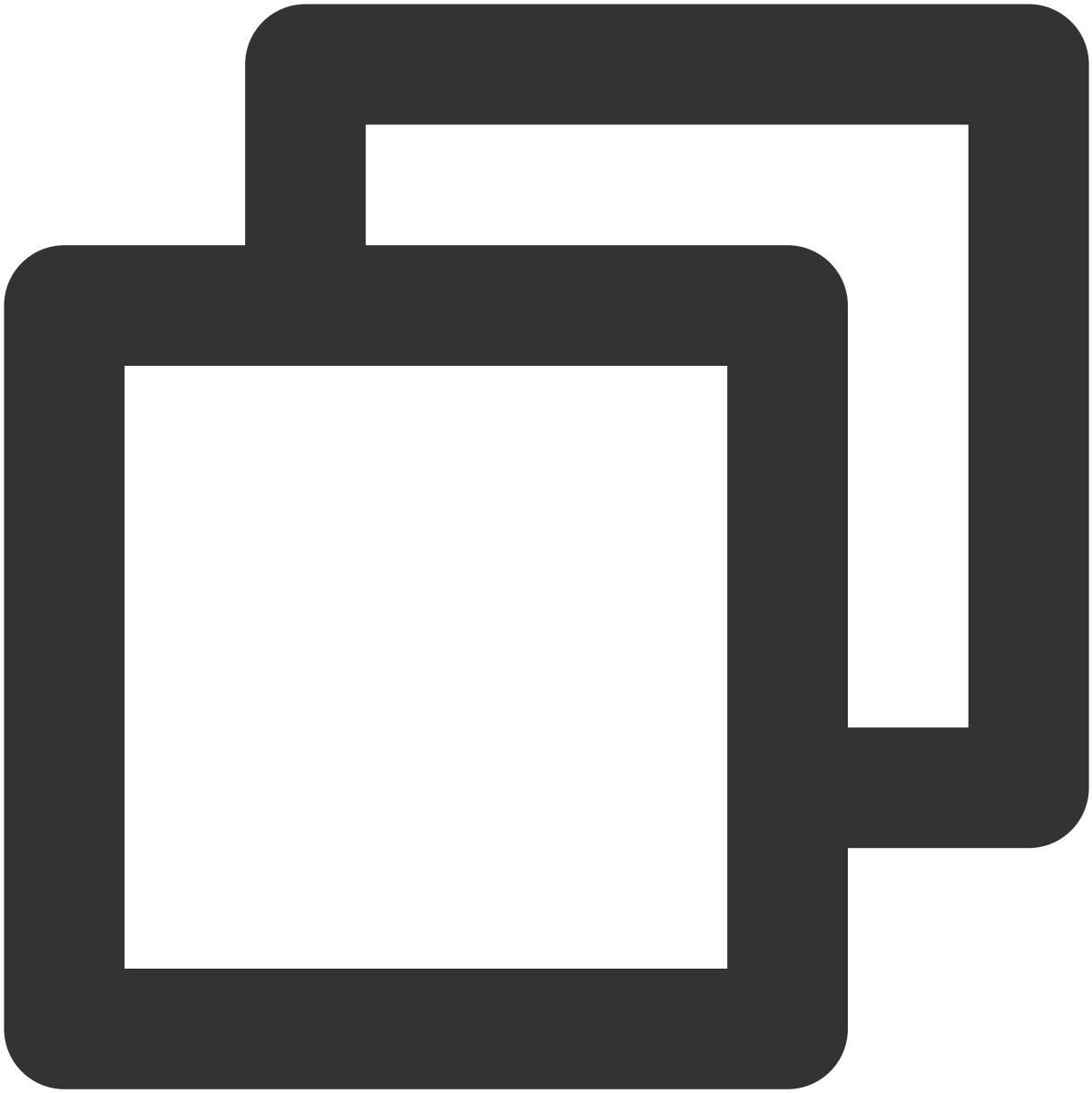
ALTER TABLE DROP PARTITION

最近更新时间：2024-08-07 17:16:32

说明

支持内核：Presto、SparkSQL。
适用表范围：外部表。
用途：删除数据表的某个分区列。

标准语法



```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION (partition_spec) [,PARTITION (par

partition_spec:
  : partition_column = partition_col_value, partition_column = partition_col_value,
```

参数

`table_name` ：需要的表名字。
`partition_column` ：分区名。
`partition_col_value` ：分区值。

示例

```
ALTER TABLE page_view DROP PARTITION (dt='2008-08-08', country='us')
```

```
ALTER TABLE `page_view` DROP
PARTITION (`dt` = '2008-08-08', `country` = 'us'),
PARTITION (`dt` = '2008-08-08', `country` = 'us'),
PARTITION (`dt` = '2008-08-08', `country` = 'us')
```

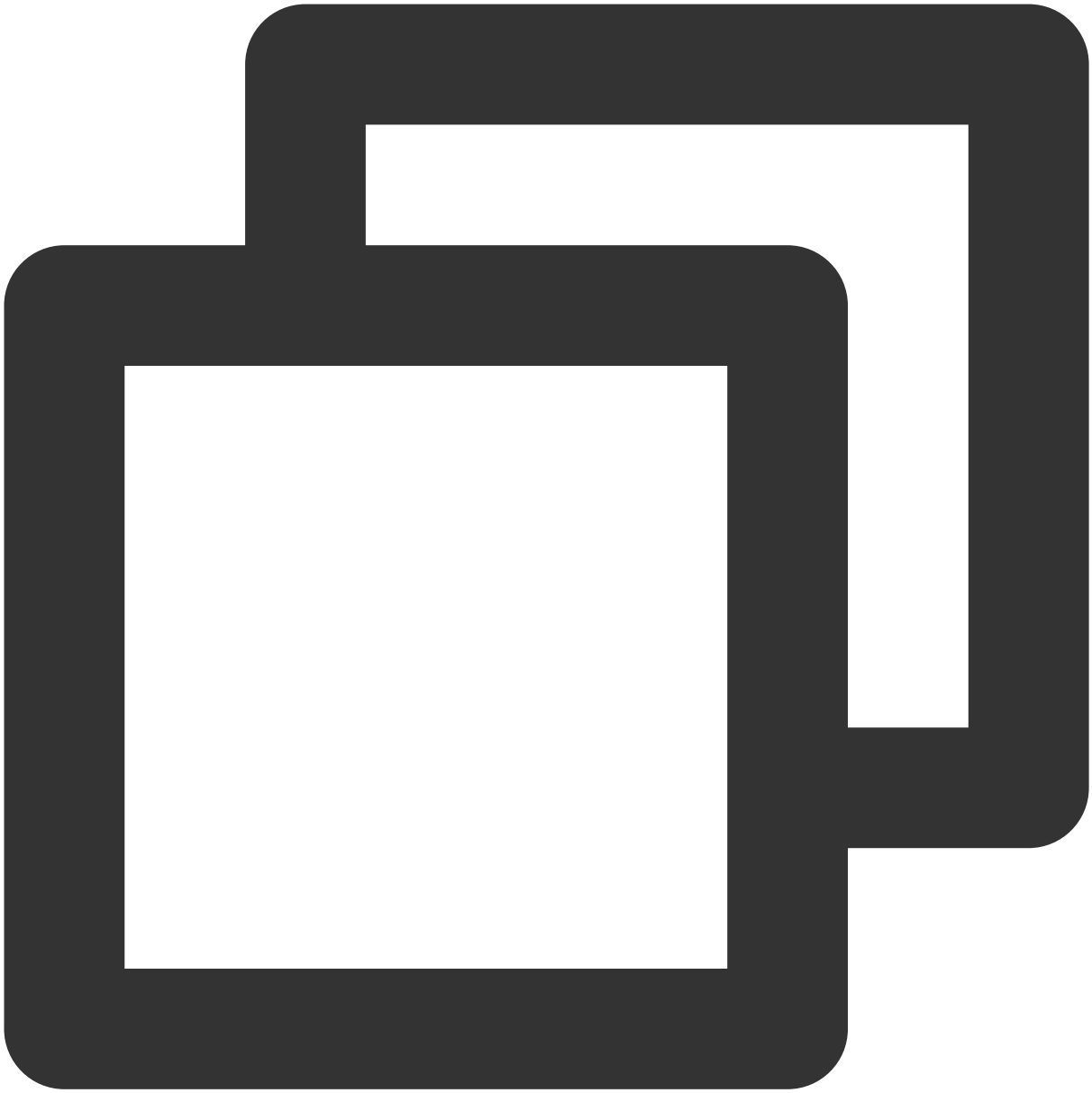
ALTER TABLE ADD PARTITION FIELD

最近更新时间：2024-08-07 17:16:46

说明

支持内核：Presto、SparkSQL。
适用表范围：原生 Iceberg 表、外部表。
用途：为数据表新增单个分区字段。

标准语法



```
ALTER TABLE table_name ADD PARTITION partition_column | hidden_partition_spec [AS a

hidden_partition_spec:
    Supported transformations are:
        years(ts): partition by year
        months(ts): partition by month
        days(ts) or date(ts): equivalent to dateint partitioning
        hours(ts) or date_hour(ts): equivalent to dateint and hour partitioning
        bucket(N, col): partition by hashed value mod N buckets
        truncate(L, col): partition by value truncated to L
            Strings are truncated to the given length
```

```
              Integers and longs truncate to bins: truncate(10, i) produces partition
```

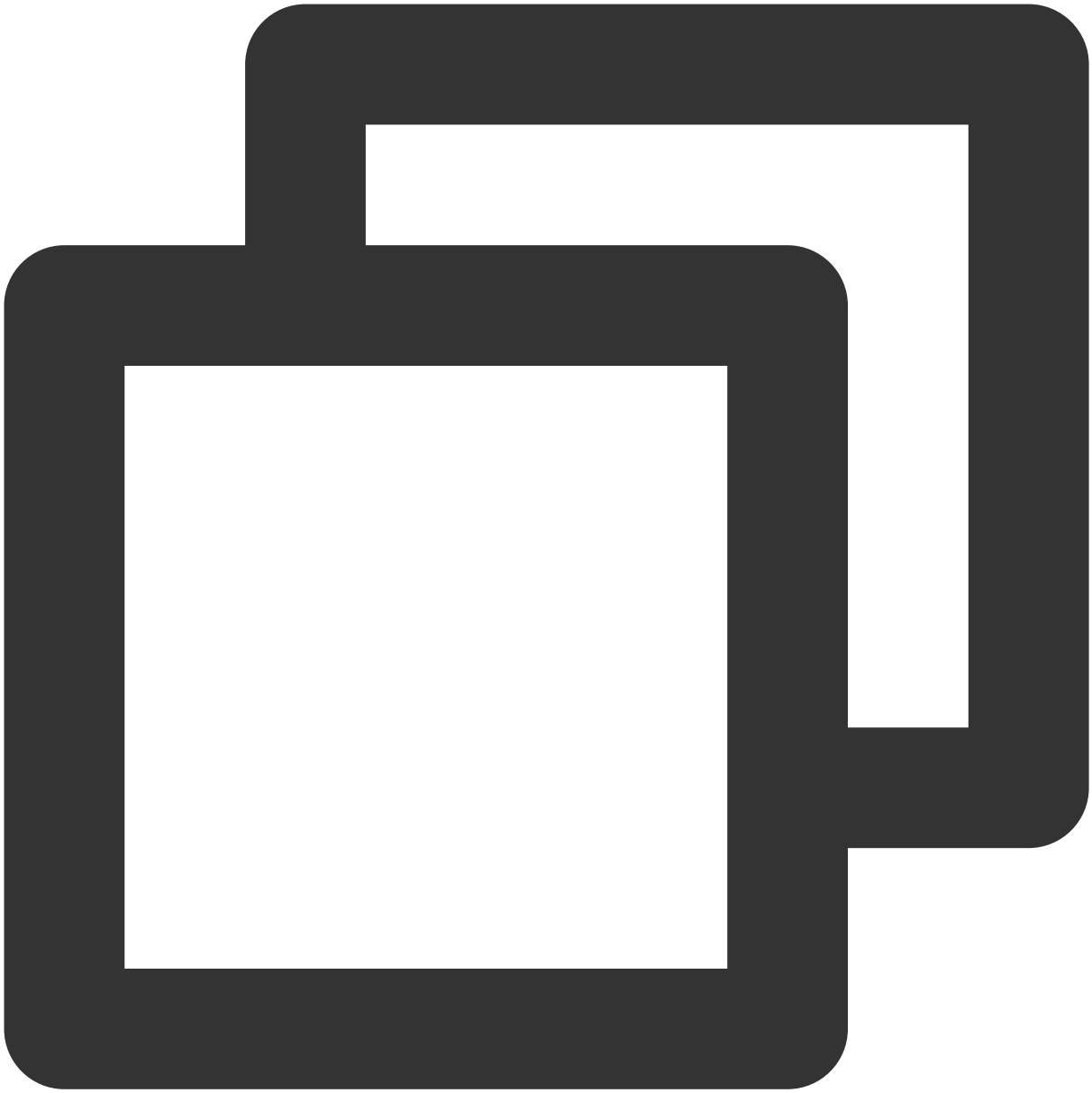
参数说明

`table_name` ：需要的表名字。

`partition_column` ：分区列。

`alias` ：分区列增加的别名。

示例



```
ALTER TABLE prod.db.sample ADD PARTITION FIELD bucket(16, id)
ALTER TABLE prod.db.sample ADD PARTITION FIELD truncate(data, 4)
ALTER TABLE prod.db.sample ADD PARTITION FIELD years(ts)
-- use optional AS keyword to specify a custom name for the partition field
ALTER TABLE prod.db.sample ADD PARTITION FIELD bucket(16, id) AS shard
```

ALTER TABLE DROP PARTITION FIELD

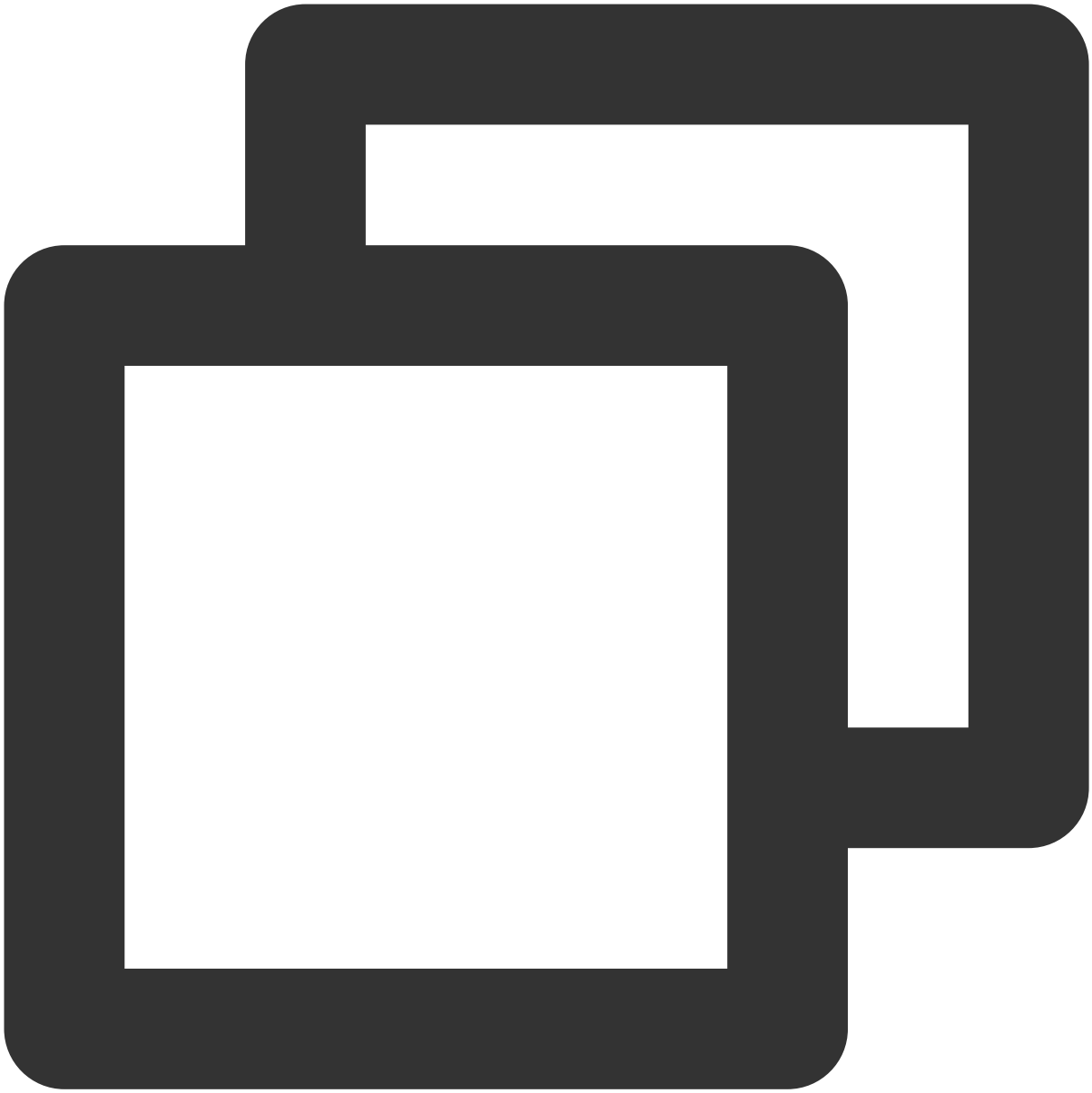
最近更新时间：2024-08-07 17:17:06

说明

支持内核：Presto、SparkSQL。
适用表范围：原生 Iceberg 表、外部表。
用途：删除数据表的某个分区字段。

标准语法

版权所有：腾讯云计算（北京）有限责任公司



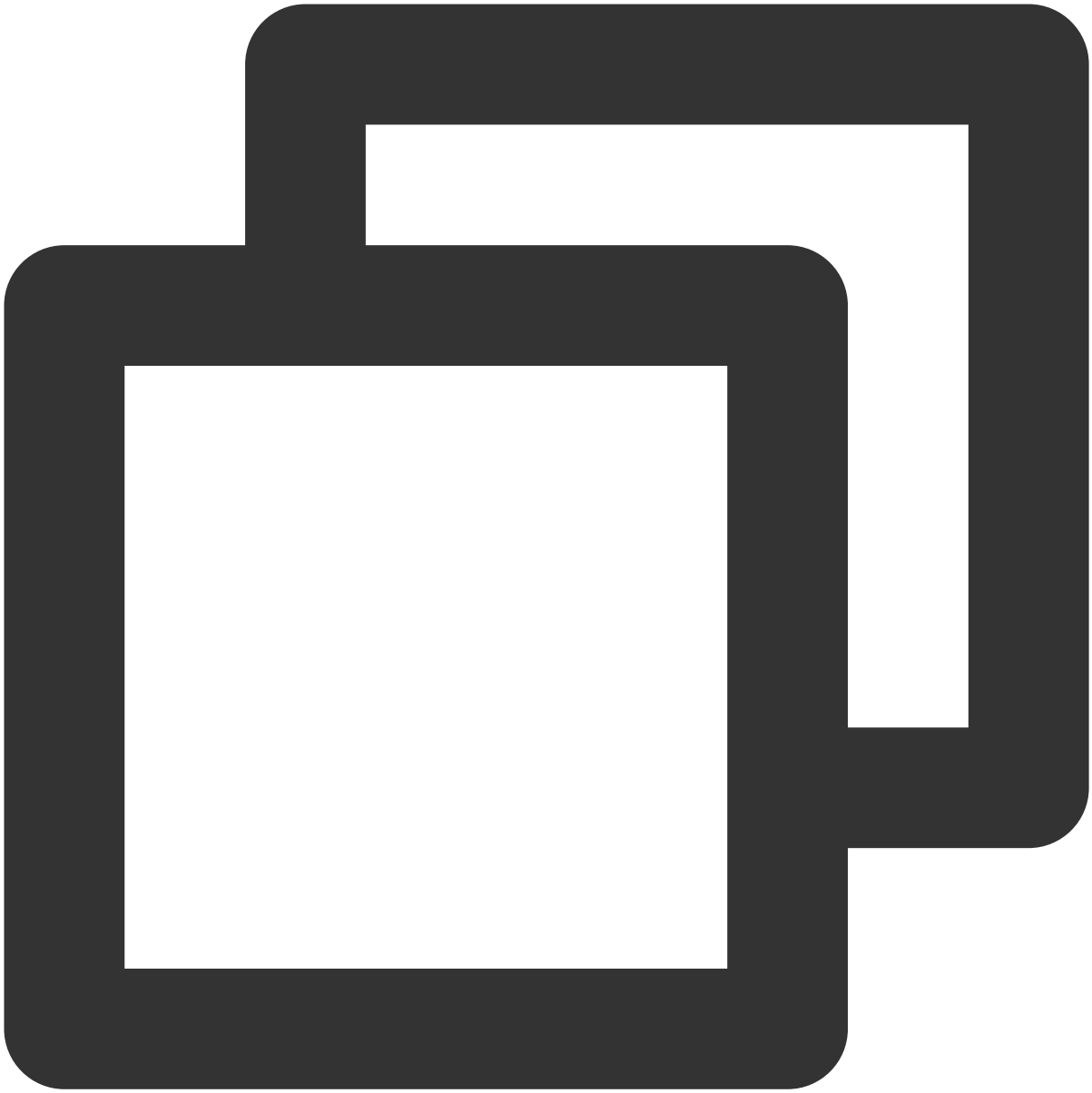
```
ALTER TABLE table_name ADD PARTITION partition_column | hidden_partition_spec

hidden_partition_spec:
    Supported transformations are:
        years(ts): partition by year
        months(ts): partition by month
        days(ts) or date(ts): equivalent to dateint partitioning
        hours(ts) or date_hour(ts): equivalent to dateint and hour partitioning
        bucket(N, col): partition by hashed value mod N buckets
        truncate(L, col): partition by value truncated to L
            Strings are truncated to the given length
```

```
              Integers and longs truncate to bins: truncate(10, i) produces partition
```

参数说明

`table_name` ：需要的表名字。
`partition_column` ：分区列。

示例



```
ALTER TABLE prod.db.sample DROP PARTITION FIELD catalog
ALTER TABLE prod.db.sample DROP PARTITION FIELD bucket(16, id)
ALTER TABLE prod.db.sample DROP PARTITION FIELD truncate(data, 4)
ALTER TABLE prod.db.sample DROP PARTITION FIELD years(ts)
ALTER TABLE prod.db.sample DROP PARTITION FIELD shard
```

版权所有：腾讯云计算（北京）有限责任公司

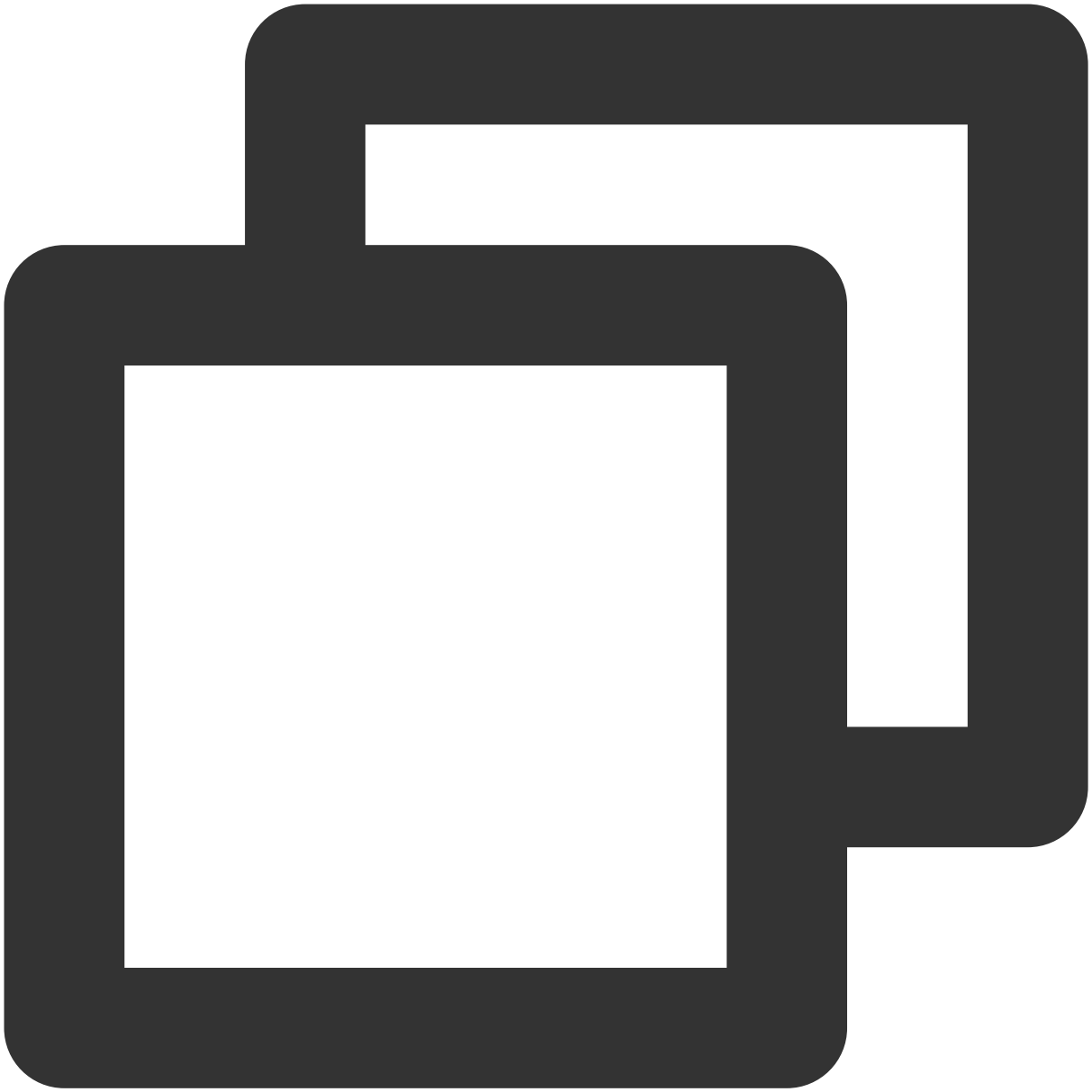
ALTER TABLE ... RENAME COLUMN

最近更新时间：2024-08-07 17:17:23

说明

支持内核：SparkSQL。
适用表类型：外部 Iceberg 表、原生 Iceberg 表。
用途：变更字段名称。

语法



```
ALTER TABLE table_identifier
RENAME COLUMN old_column_name TO new_column_name
```

参数

`table_identifier` ：数据表名称。
`old_column_name` ：需变更的字段名称。

`new_column_name` ：变更后的字段名称。

示例

```
alter table iceberg_rename rename column id to id_2
```

ALTER TABLE SET TBLPROPERTIES

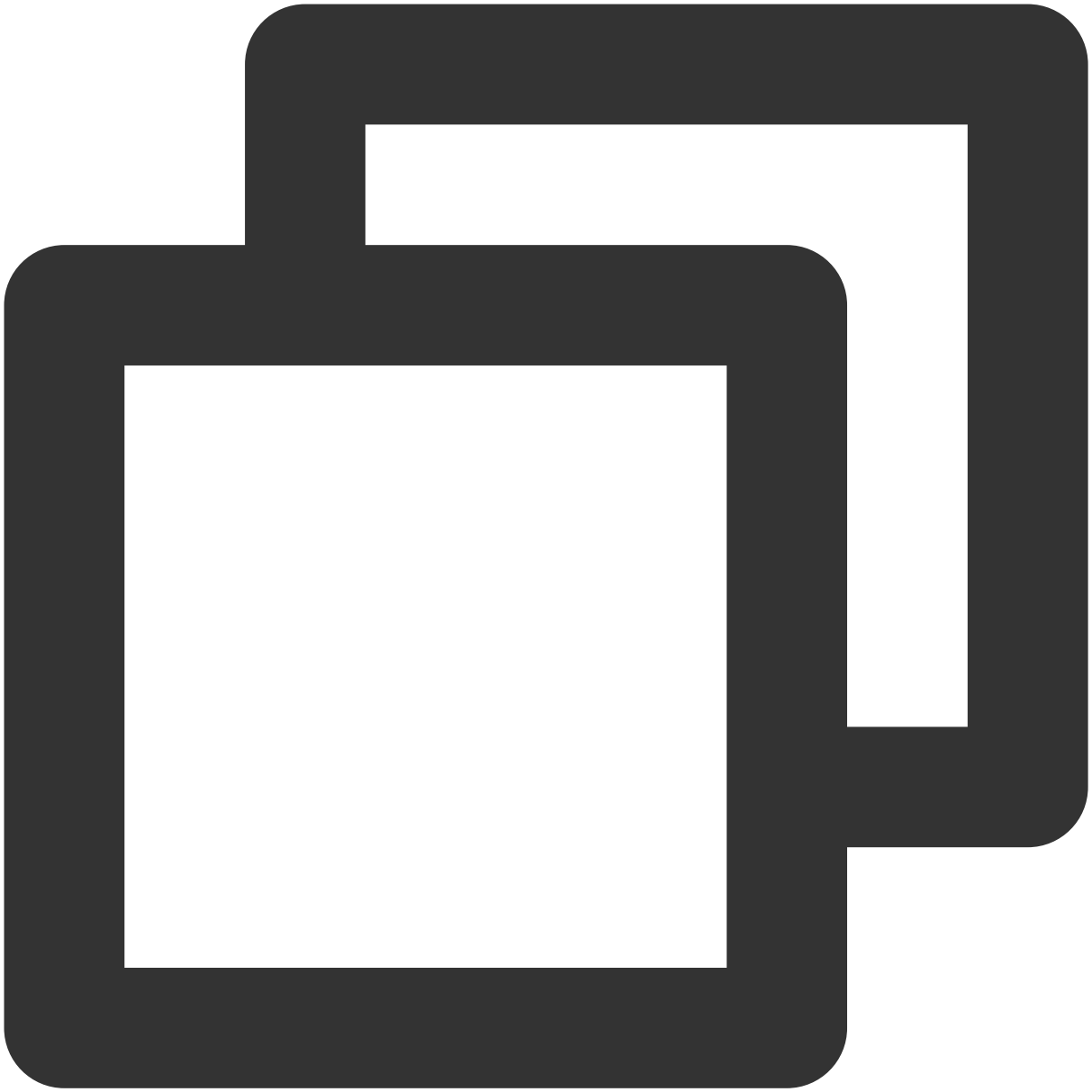
最近更新时间：2024-08-07 17:17:38

说明

支持内核：Presto、SparkSQL。
适用表范围：原生Iceberg表、外部表。
用途：更新/删除数据表属性。

SET：更新属性配置

语法



```
ALTER TABLE table_name
SET TBLPROPERTIES (property_name=property_value, ...)
```

参数

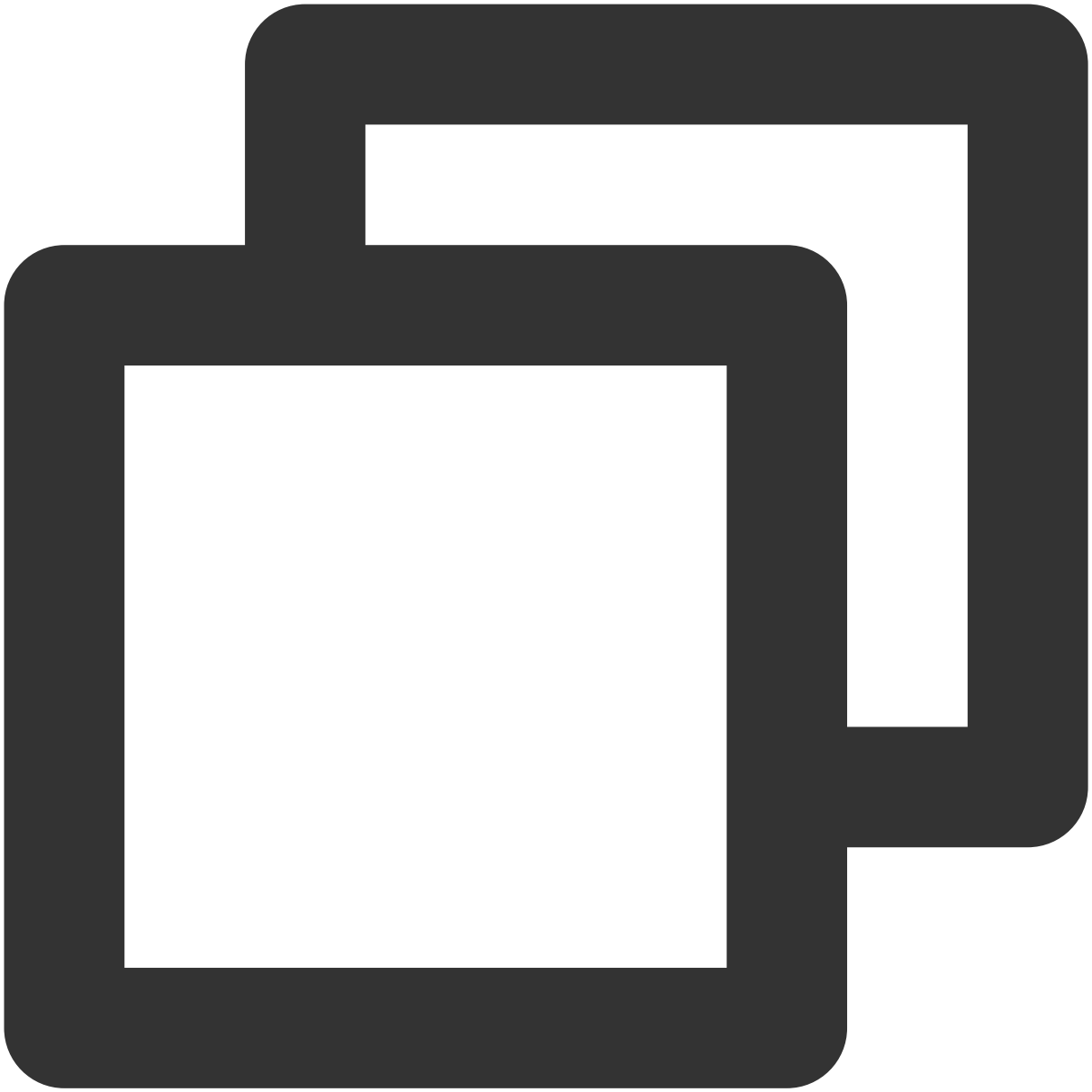
`table_name` ：需要的表名字。

`property_name` ：需要修改的属性名称。

`property_value` ：需要修改的属性值。

版权所有：腾讯云计算（北京）有限责任公司

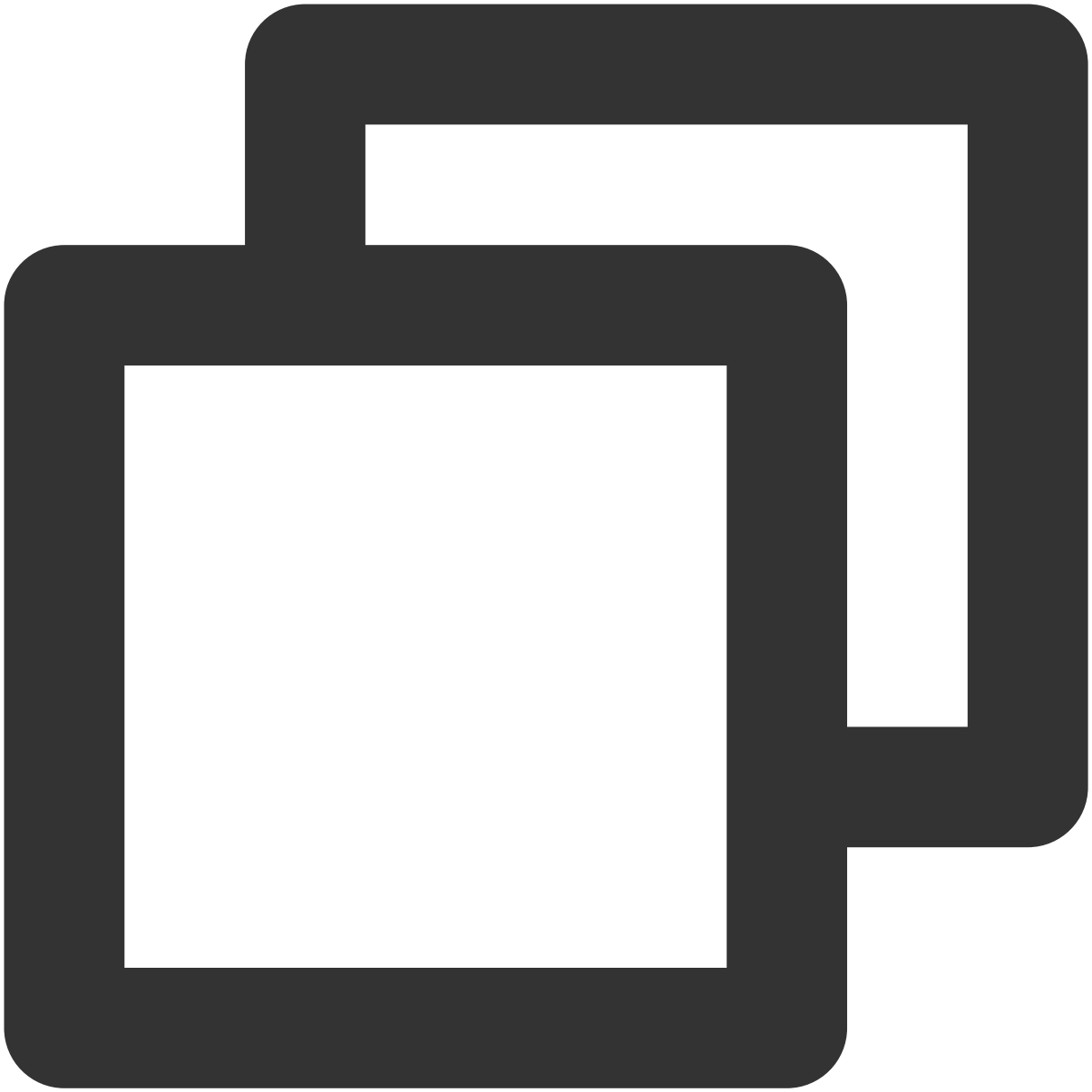
示例



```
ALTER TABLE orders SET TBLPROPERTIES ('notes'="Please don't drop this table.");
```

UNSET：删除属性配置

语法



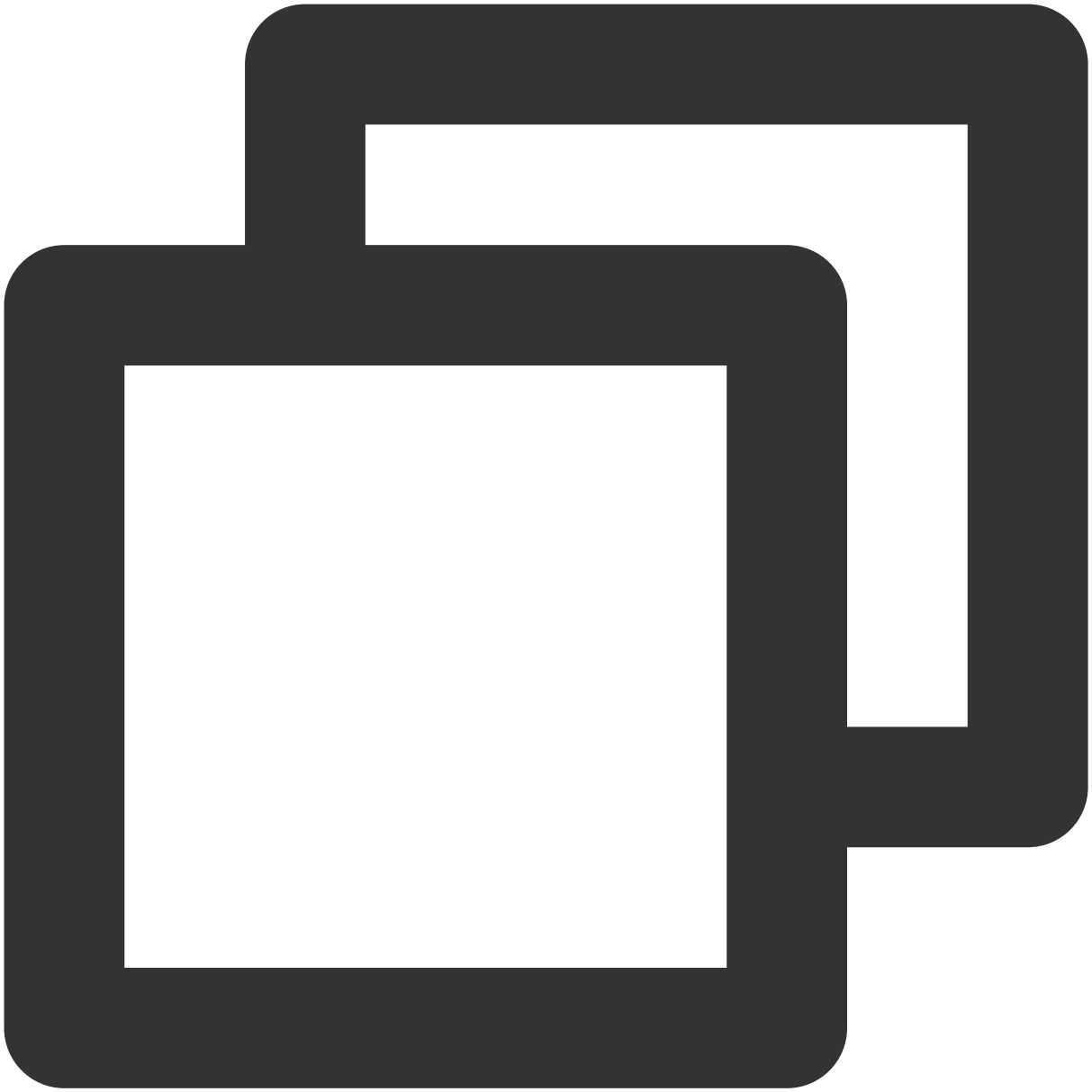
```
ALTER TABLE table_name
UNSET TBLPROPERTIES (property_name, ...)
```

参数

`table_name` ：需要的表名字。

`property_name` ：需要修改的属性名称。

示例



```
ALTER TABLE dempts UNSET TBLPROPERTIES ('read.split.target-size')
```

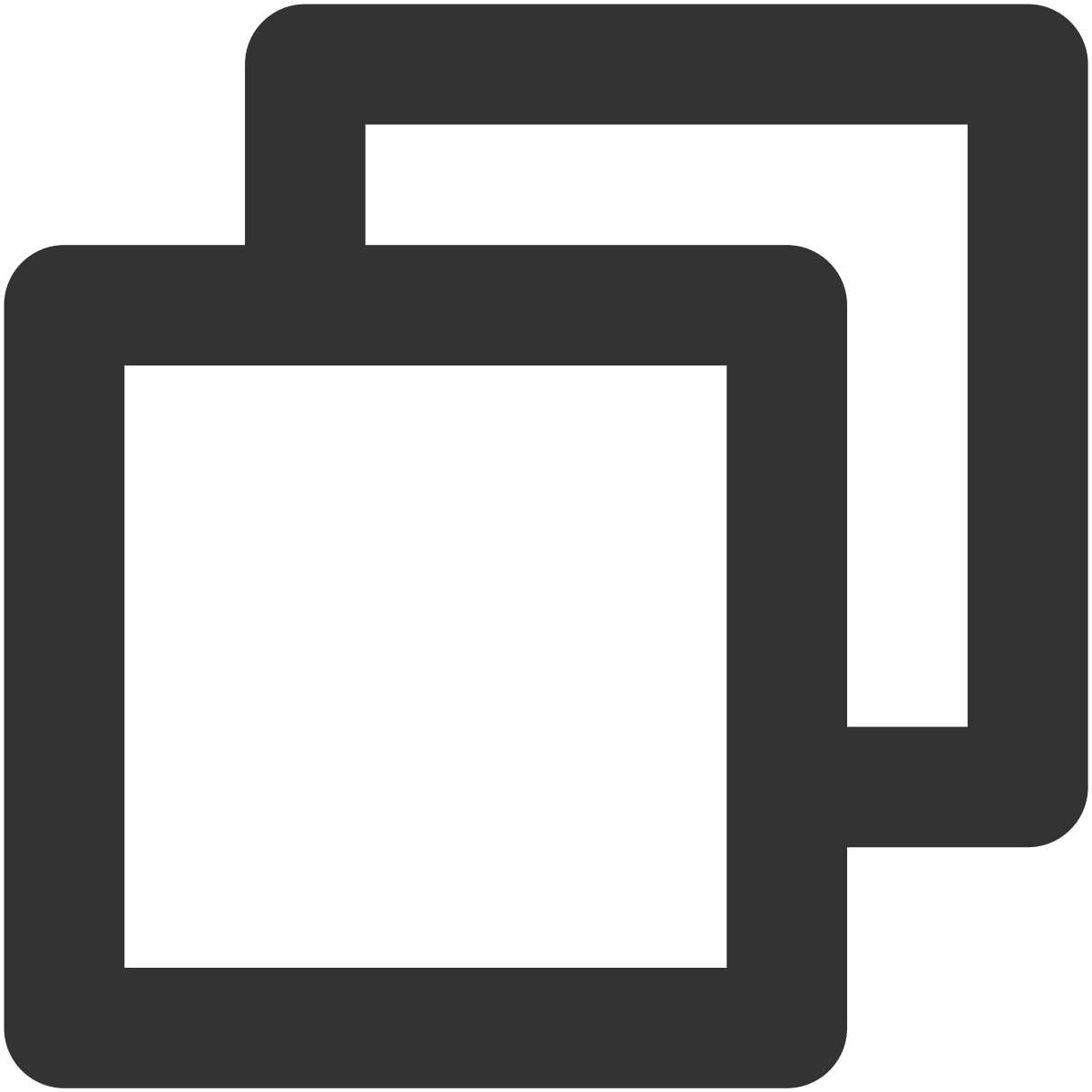
ALTER TABLE SET LOCATION

最近更新时间：2024-08-07 17:17:57

说明

支持内核：Presto、SparkSQL。
适用表范围：原生Iceberg表、外部表。
用途：修改数据表的存储路径。

标准语法



```
ALTER TABLE table_name [ PARTITION (partition_spec) ] SET LOCATION 'new location';
```

参数

`table_name` ：数据表的名称。

`PARTITION (partition_spec)` ：指定分区列。

partition_col_name：分区列名。

partition_col_value：分区列的值。

`'new location'` ：新的表或者分区在 tencent cos 上的位置。

示例

```
ALTER TABLE tbl PARTITION (a='1', b='2') SET LOCATION '/path/to/part/ways';

ALTER TABLE tbl SET LOCATION '/path/to/part/ways';
```

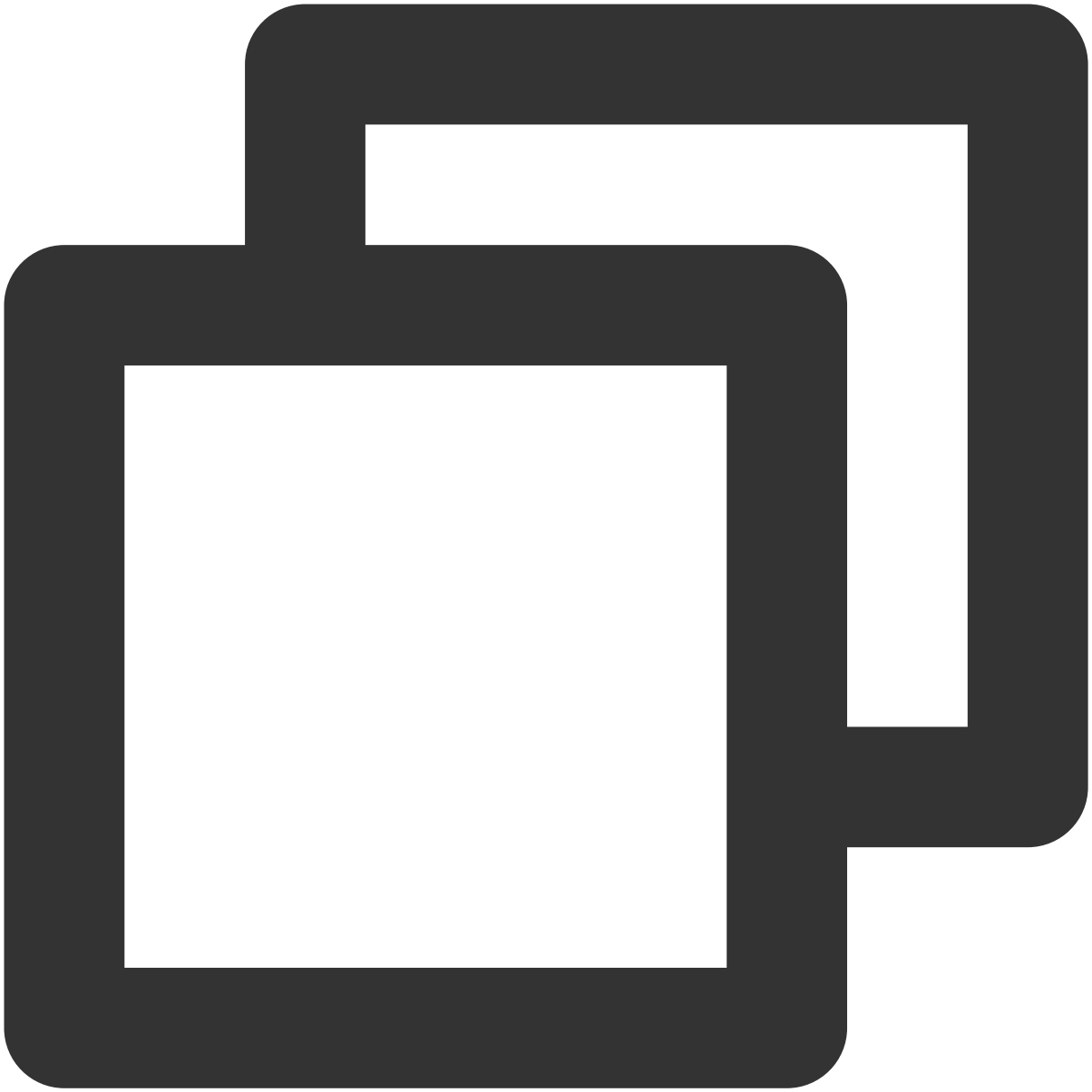
ALTER TABLE ... WRITE ORDERED BY

最近更新时间：2024-08-07 17:18:13

说明

支持内核：SparkSQL。
适用表范围：外部 Iceberg 表、原生 Iceberg 表。
用途：设置表数据插入时的排序方式。

语法



```
ALTER TABLE table_identifier
WRITE [LOCALLY] ORDERED BY
{col_name [ASC|DESC] [NULLS FIRST|LAST]}[, ...]
```

参数

`table_identifier` ：数据表名称

示例

```
ALTER TABLE dempts WRITE ORDERED BY category, id;
-- use optional ASC/DEC keyword to specify sort order of each field (default ASC)
ALTER TABLE dempts WRITE ORDERED BY category ASC, id DESC;
-- use optional NULLS FIRST/NULLS LAST keyword to specify null order of each field
ALTER TABLE dempts WRITE ORDERED BY category ASC NULLS LAST, id DESC NULLS FIRST;
-- To order within each task, not across tasks
ALTER TABLE dempts WRITE LOCALLY ORDERED BY category, id;
```

版权所有：腾讯云计算（北京）有限责任公司

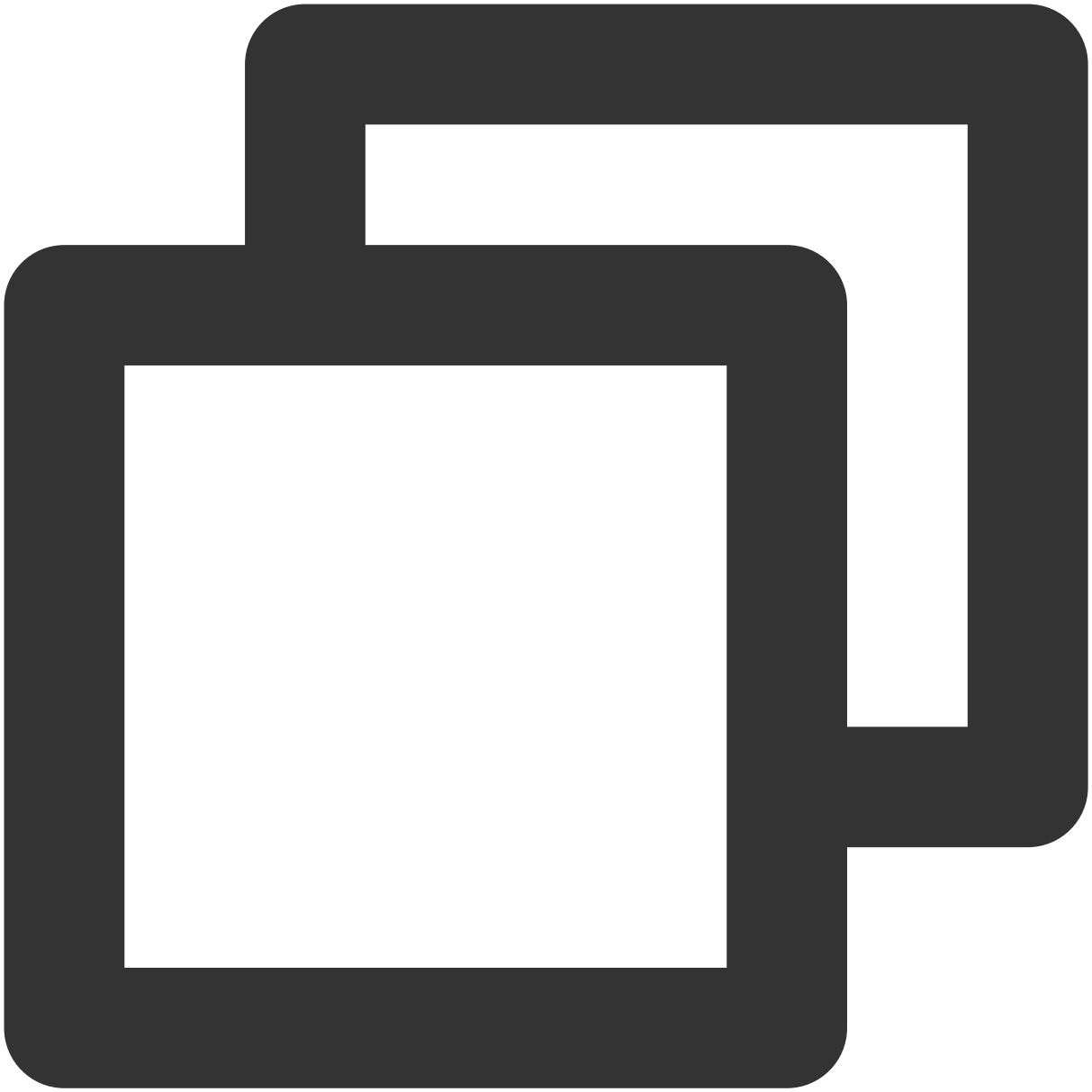
ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION

最近更新时间：2024-08-07 17:18:28

说明

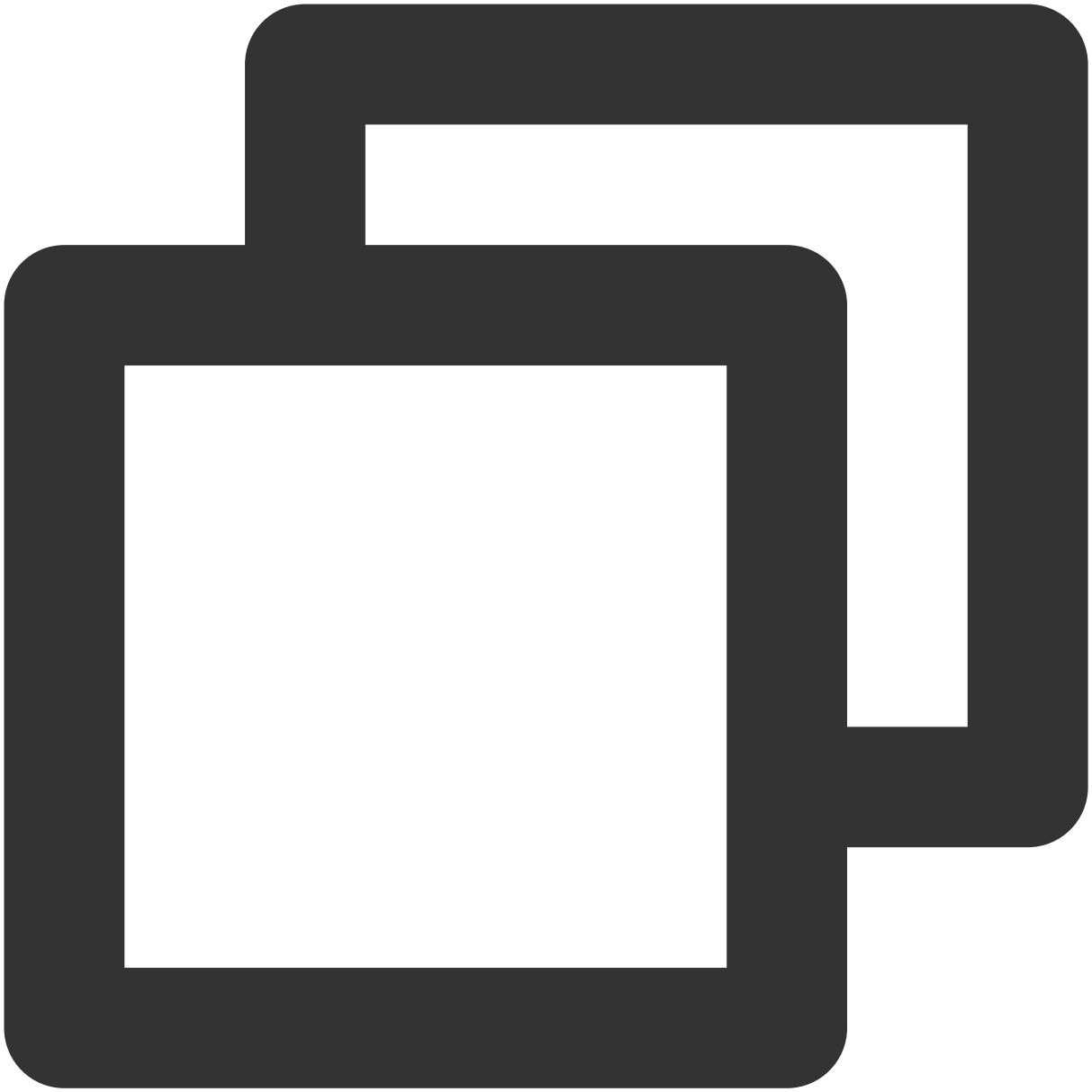
支持内核：SparkSQL。
适用表类型：外部 Iceberg 表、原生 Iceberg 表。
用途：修改分区表的数据分配策略。

语法



```
ALTER TABLE table_identifier
WRITE DISTRIBUTED BY PARTITION
[ LOCALLY ORDERED BY
{col_name [ASC|DESC] [NULLS FIRST|LAST]}[, ...]]
```

示例



```
ALTER TABLE dempts WRITE DISTRIBUTED BY PARTITION;
ALTER TABLE dempts WRITE DISTRIBUTED BY PARTITION LOCALLY ORDERED BY id;
```

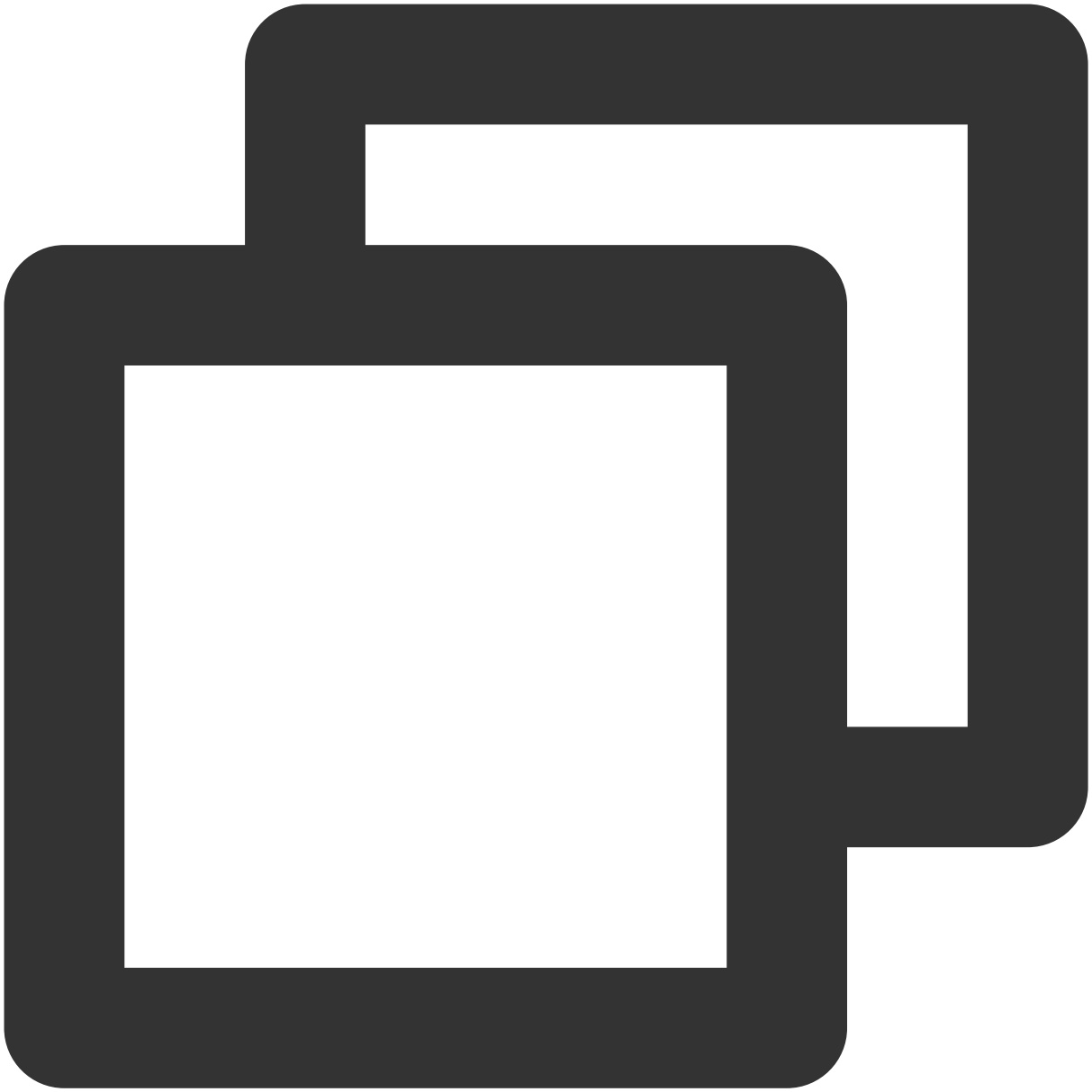
ALTER TABLE ... SET IDENTIFIER FIELDS

最近更新时间：2024-08-07 17:19:12

说明

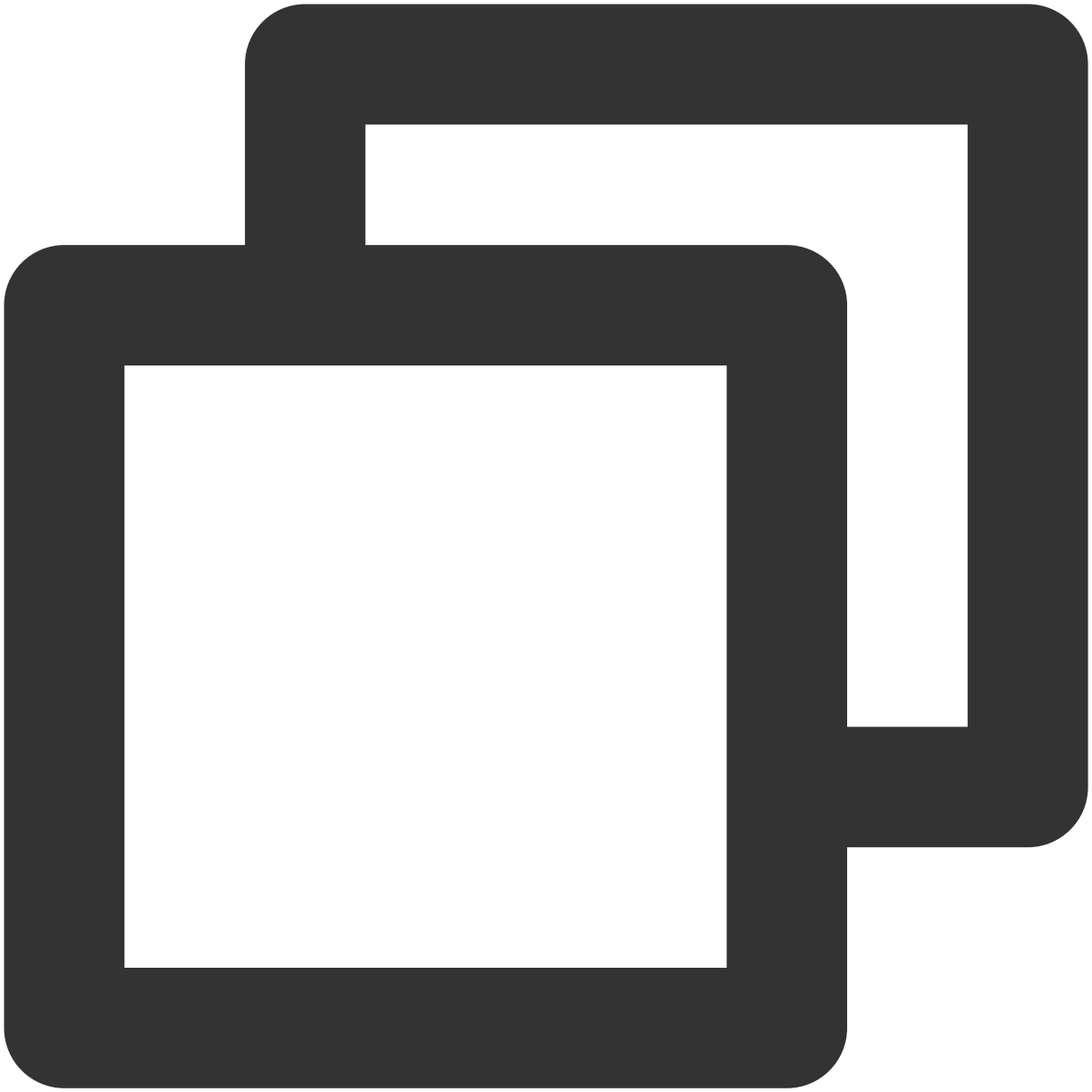
支持内核：SparkSQL。
适用表类型：外部 Iceberg 表、原生 Iceberg 表。
用途：添加 identifier fields 属性。

语法



```
ALTER TABLE dempts SET IDENTIFIER FIELD empno, name
```

示例



```
ALTER TABLE tb1 SET IDENTIFIER FIELDS id, location.lon
```

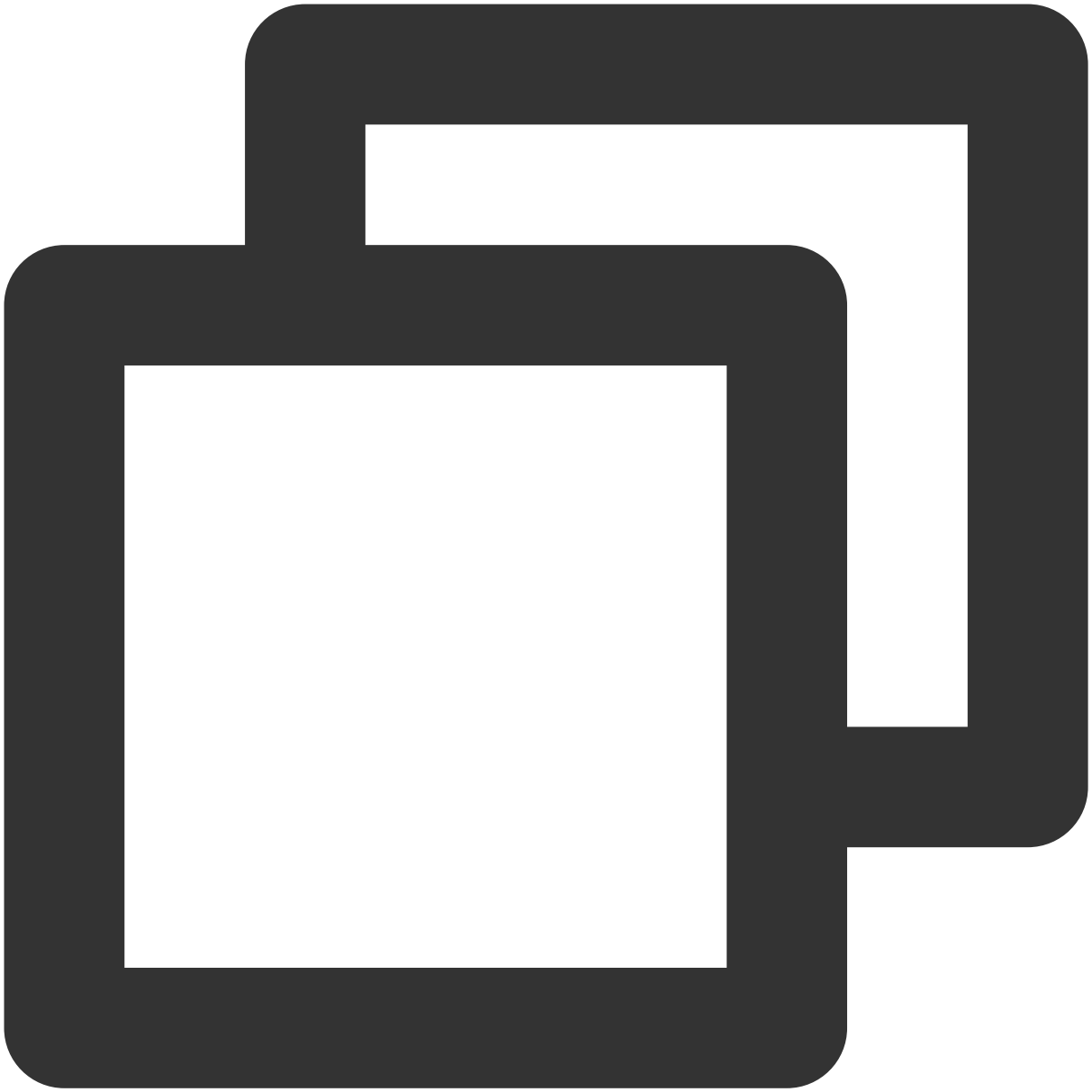
ALTER TABLE ... DROP IDENTIFIER FIELDS

最近更新时间：2024-08-07 17:19:33

说明

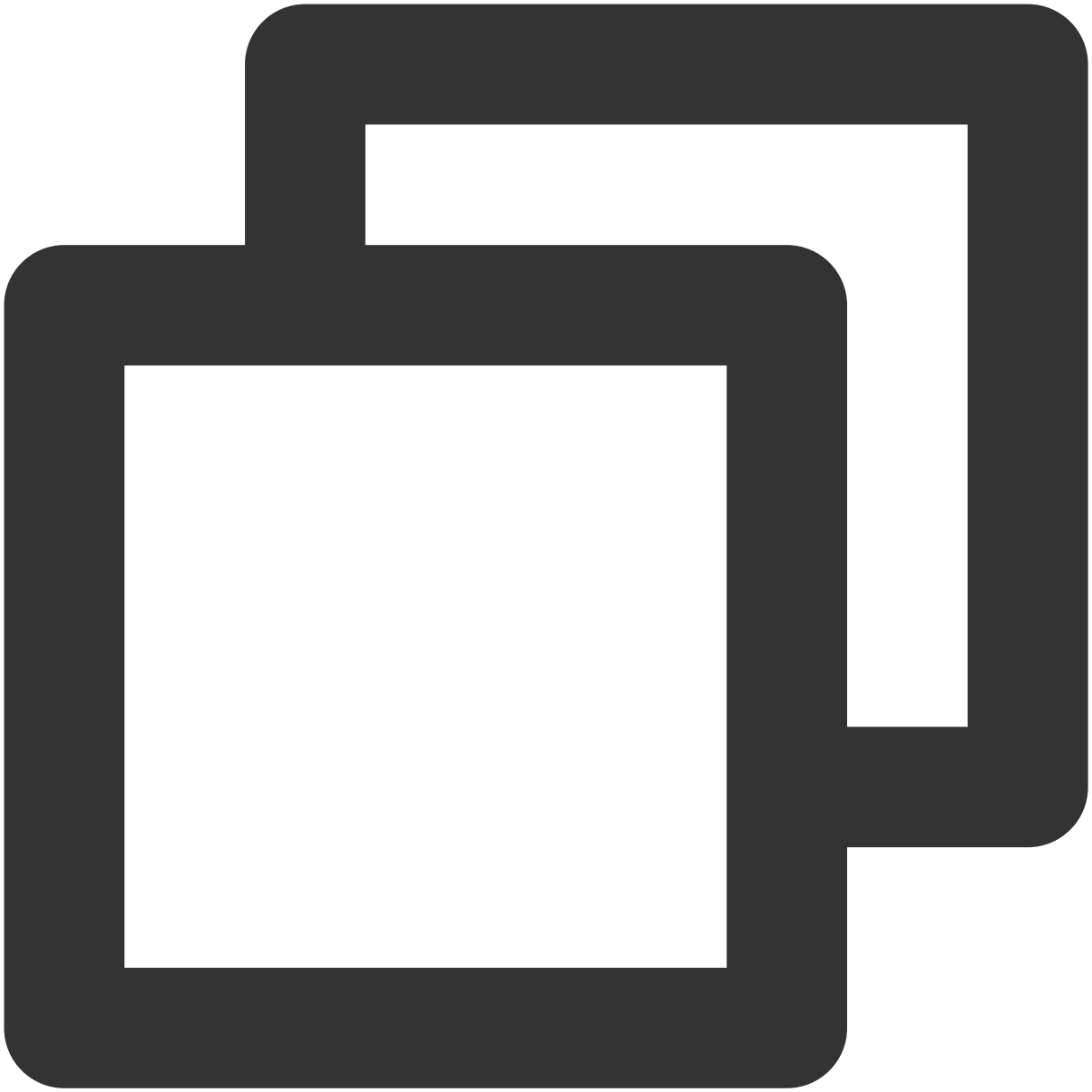
支持内核：SparkSQL。
适用表类型：外部 Iceberg 表、原生 Iceberg 表。
用途：删除 identifier fields 属性。

语法



```
ALTER TABLE dempts DROP IDENTIFIER FIELD empno, name
```

示例



```
ALTER TABLE tb1 DROP IDENTIFIER FIELDS id, location.lon
```

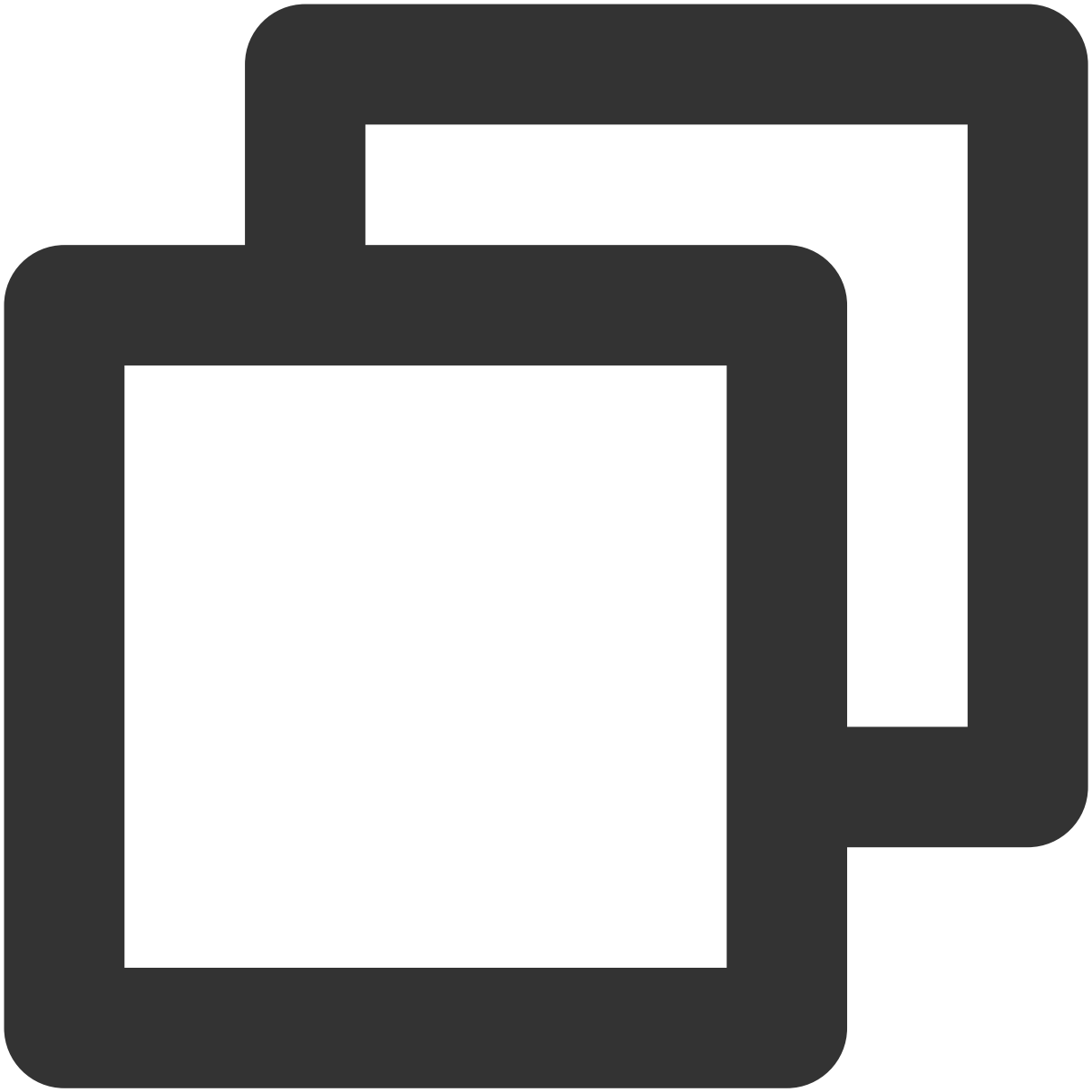
MSCK REPAIR TABLE

最近更新时间：2024-08-07 17:19:45

说明

支持内核：Presto、SparkSQL。
适用表范围：原生 Iceberg 表、外部表。
用途：更新数据表的分区信息。

标准语法

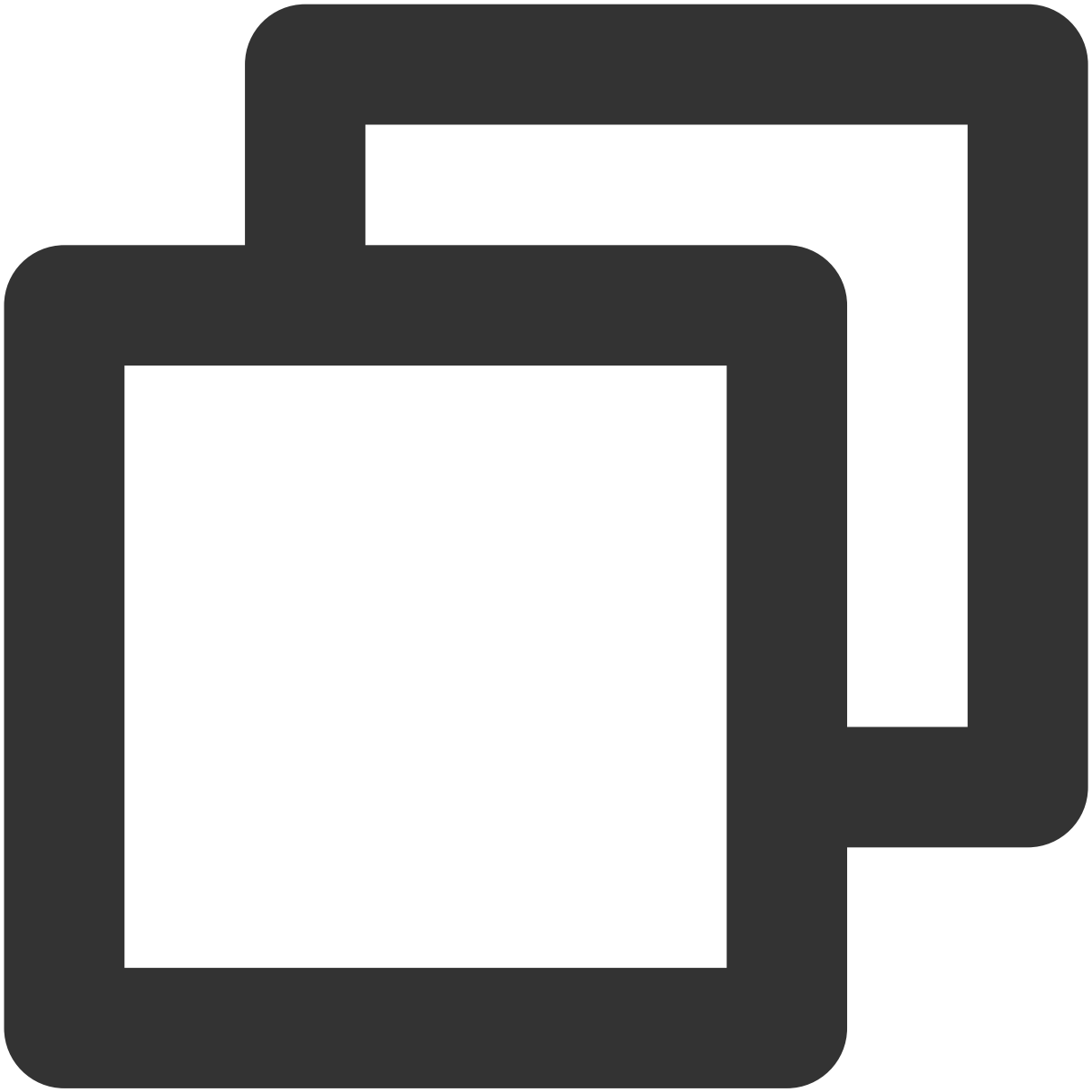


```
MSCK REPAIR TABLE table_identifier
```

参数

`table_identifier` ：表的名称。

示例



```
MSCK REPAIR TABLE t1
```

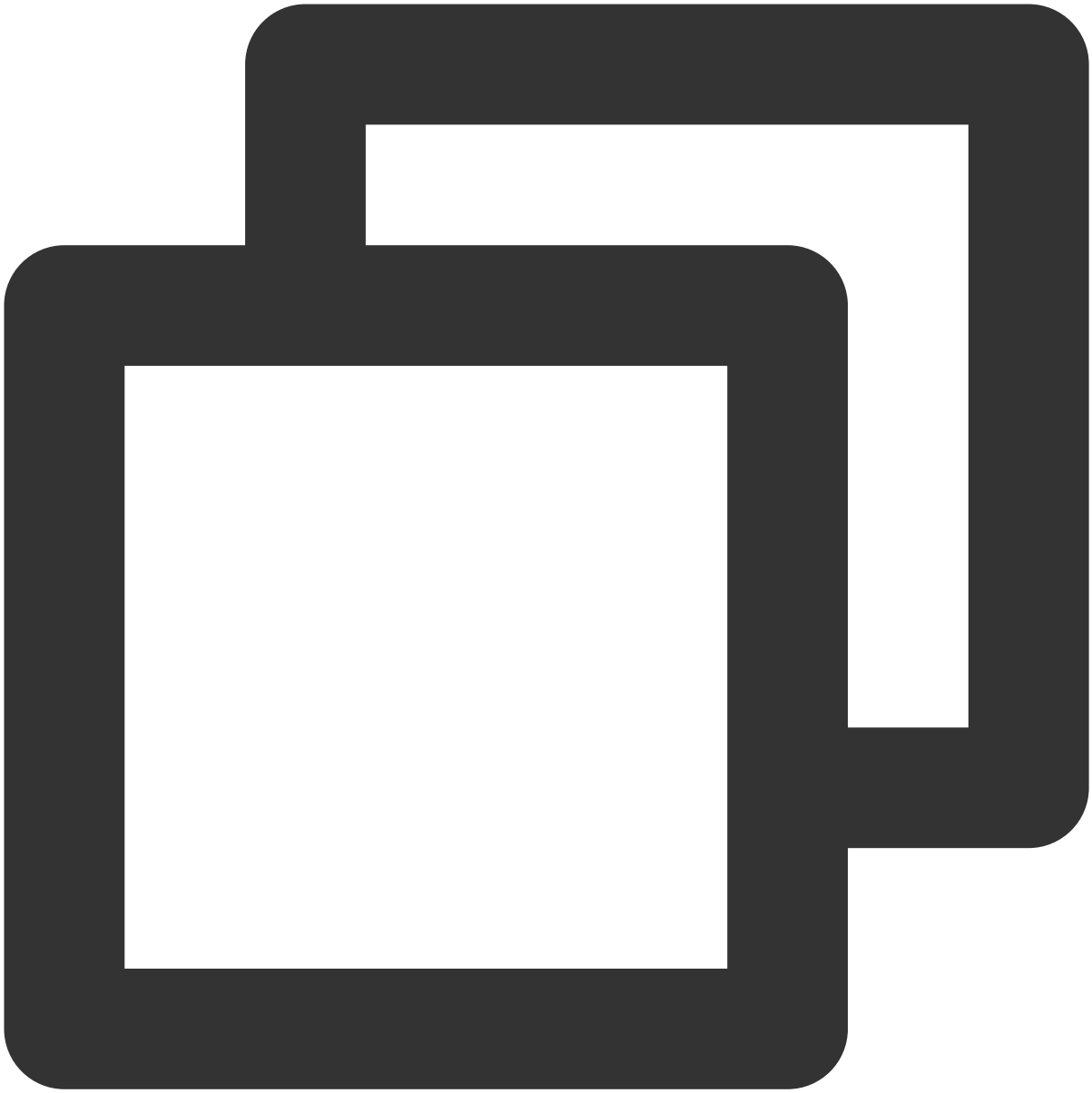
ANALYZE TABLES

最近更新时间：2024-08-07 17:20:01

说明

支持内核：Presto、SparkSQL。
用途：支持对数据库表进行统计。

语法



```
ANALYZE TABLES [ { FROM | IN } database_name ] COMPUTE STATISTICS [ NOSCAN ]

ANALYZE TABLE table_identifier
[ PARTITION ( partition_col_name [ = partition_col_val ] [ , ... ] ) ]
COMPUTE STATISTICS [ NOSCAN | FOR COLUMNS col [ , ... ] | FOR ALL COLUMNS ]
```

参数

`database_name` ：需要计算统计信息的表所在的数据库。
`table_identifier` ：需要计算统计信息的表名。
`partition_col_name` ：需要计算统计信息的分区列名。
`partition_col_value` ：需要计算统计信息的分区列的值。

示例

```
ANALYZE TABLE students COMPUTE STATISTICS
```

```
ANALYZE TABLE students COMPUTE STATISTICS FOR COLUMNS name
ANALYZE TABLE db.students COMPUTE STATISTICS FOR COLUMNS name
ANALYZE TABLE students COMPUTE STATISTICS NOSCAN
ANALYZE TABLE students COMPUTE STATISTICS FOR all COLUMNS
ANALYZE TABLE db.students COMPUTE STATISTICS FOR all COLUMNS
ANALYZE TABLE students PARTITION (student_id) COMPUTE STATISTICS
ANALYZE TABLE students PARTITION (student_id = 111111) COMPUTE STATISTICS
ANALYZE TABLE db.students PARTITION (student_id = 111111, name = 'test') COMPUTE ST

ANALYZE TABLES COMPUTE STATISTICS
ANALYZE TABLES COMPUTE STATISTICS NOSCAN
ANALYZE TABLES from school_db COMPUTE STATISTICS NOSCAN
ANALYZE TABLES IN school_db COMPUTE STATISTICS NOSCAN
```

版权所有：腾讯云计算（北京）有限责任公司

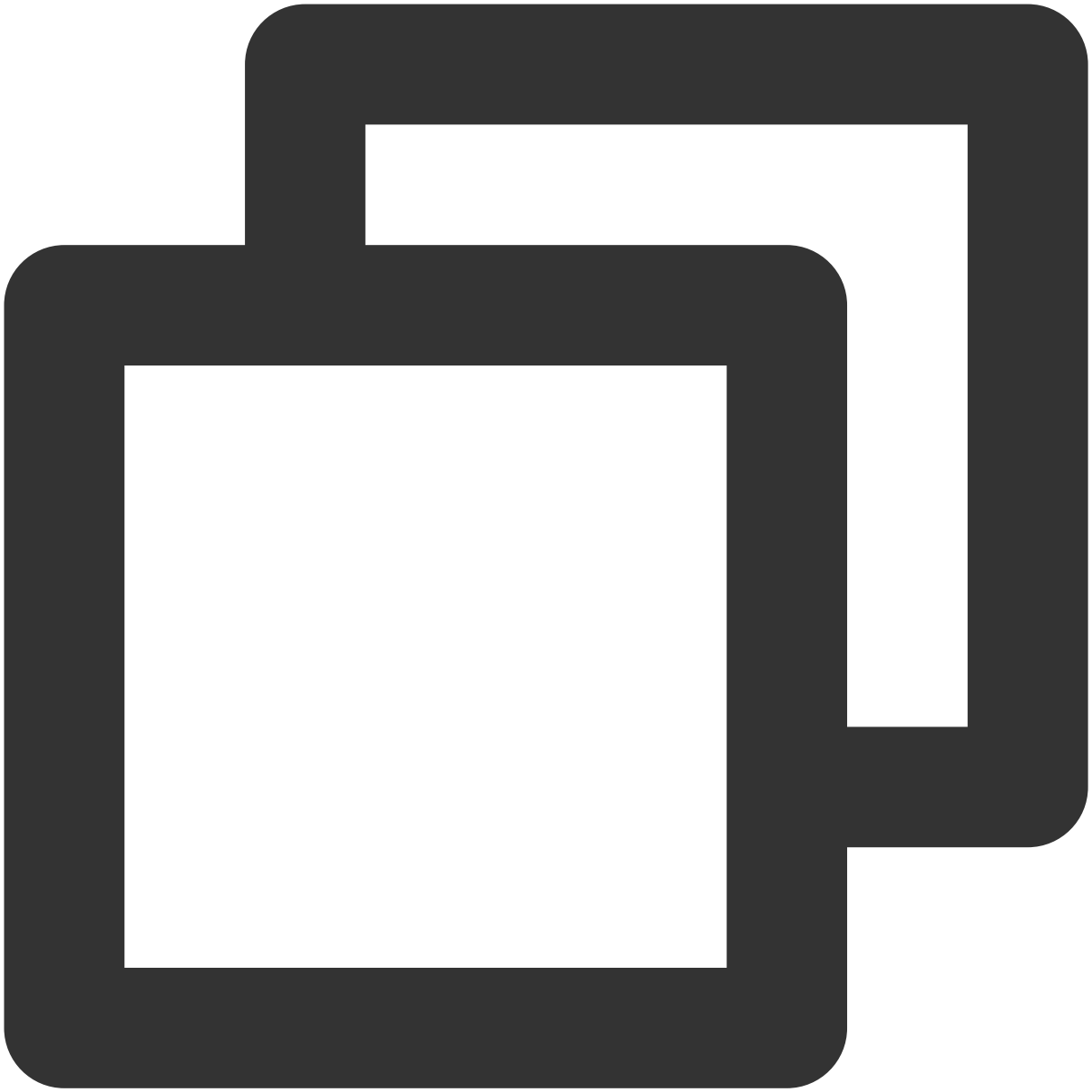
DROP TABLE

最近更新时间：2024-08-07 17:21:11

说明

支持内核：Presto、SparkSQL。
适用表范围：原生表、外部表。
用途：移除元数据表。

语法



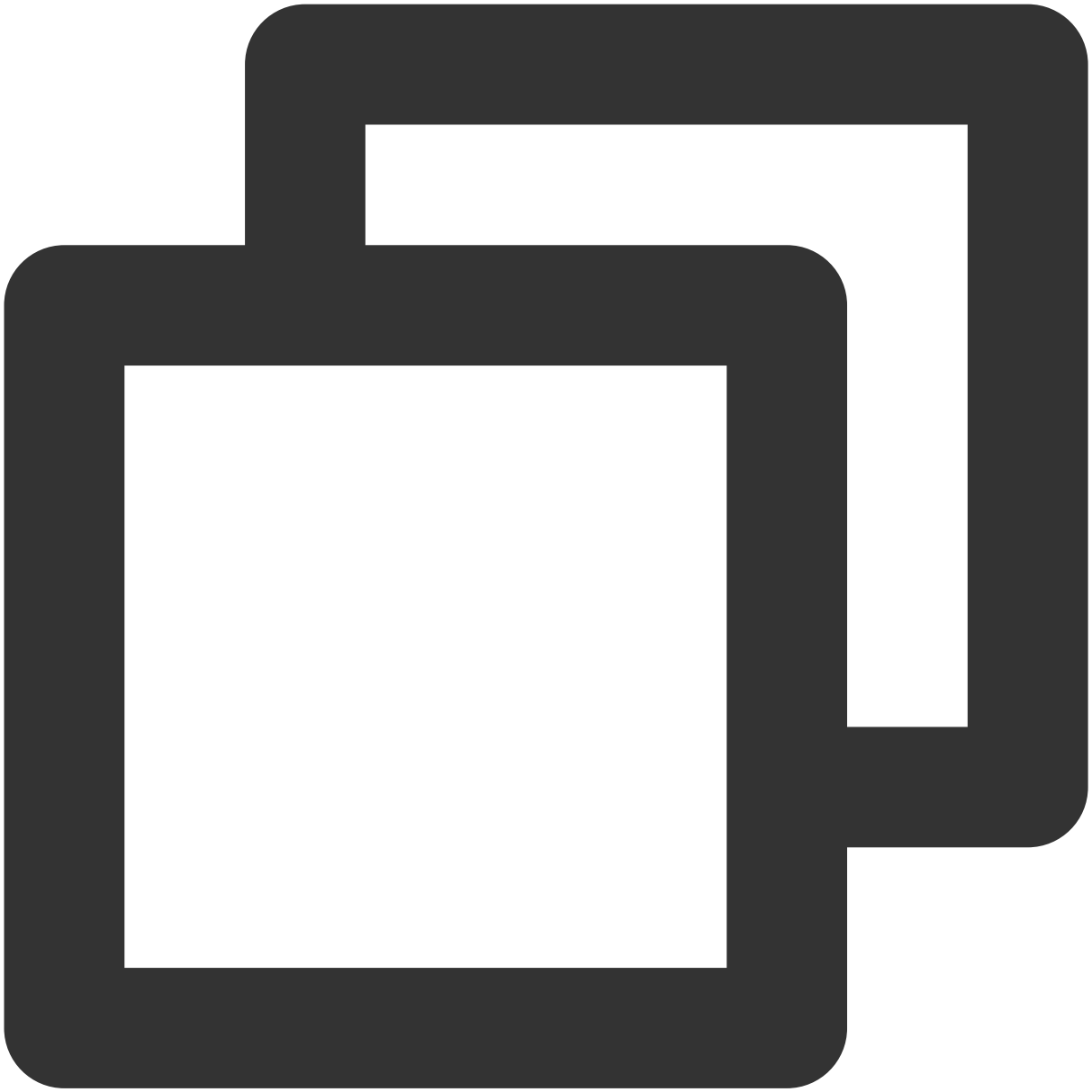
```
DROP TABLE [IF EXISTS] table_name;
```

参数

`IF EXISTS` ：可选，如果存在的意思。

`table_name` ：表名。

示例



```
DROP TABLE tbl
DROP TABLE IF EXISTS tbl
```

EXPLAIN

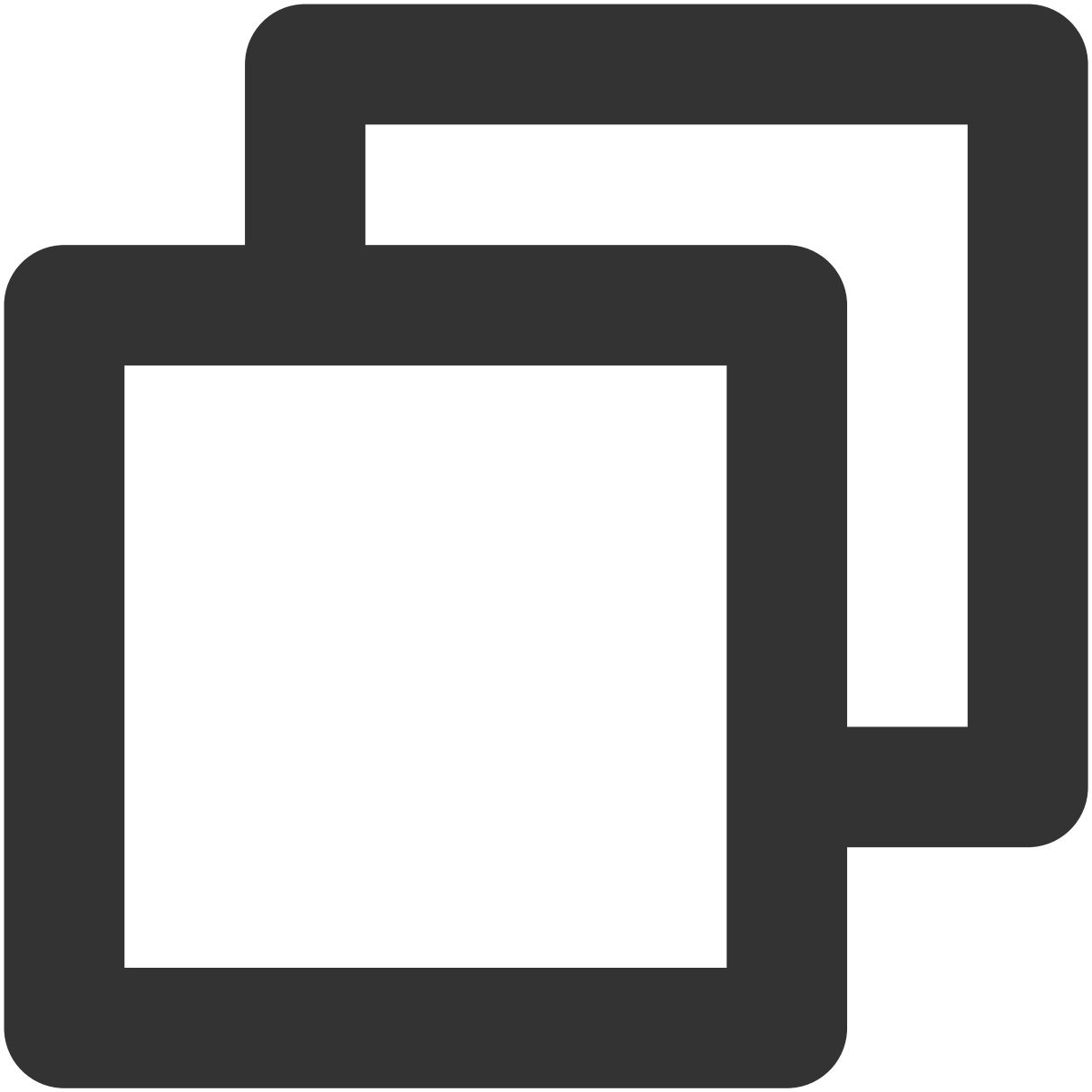
最近更新时间：2024-08-07 17:21:25

说明

支持内核：Presto、SparkSQL。
支持表类型：原生表、外部表。
用途：展示执行 sql 的逻辑或物理计划。

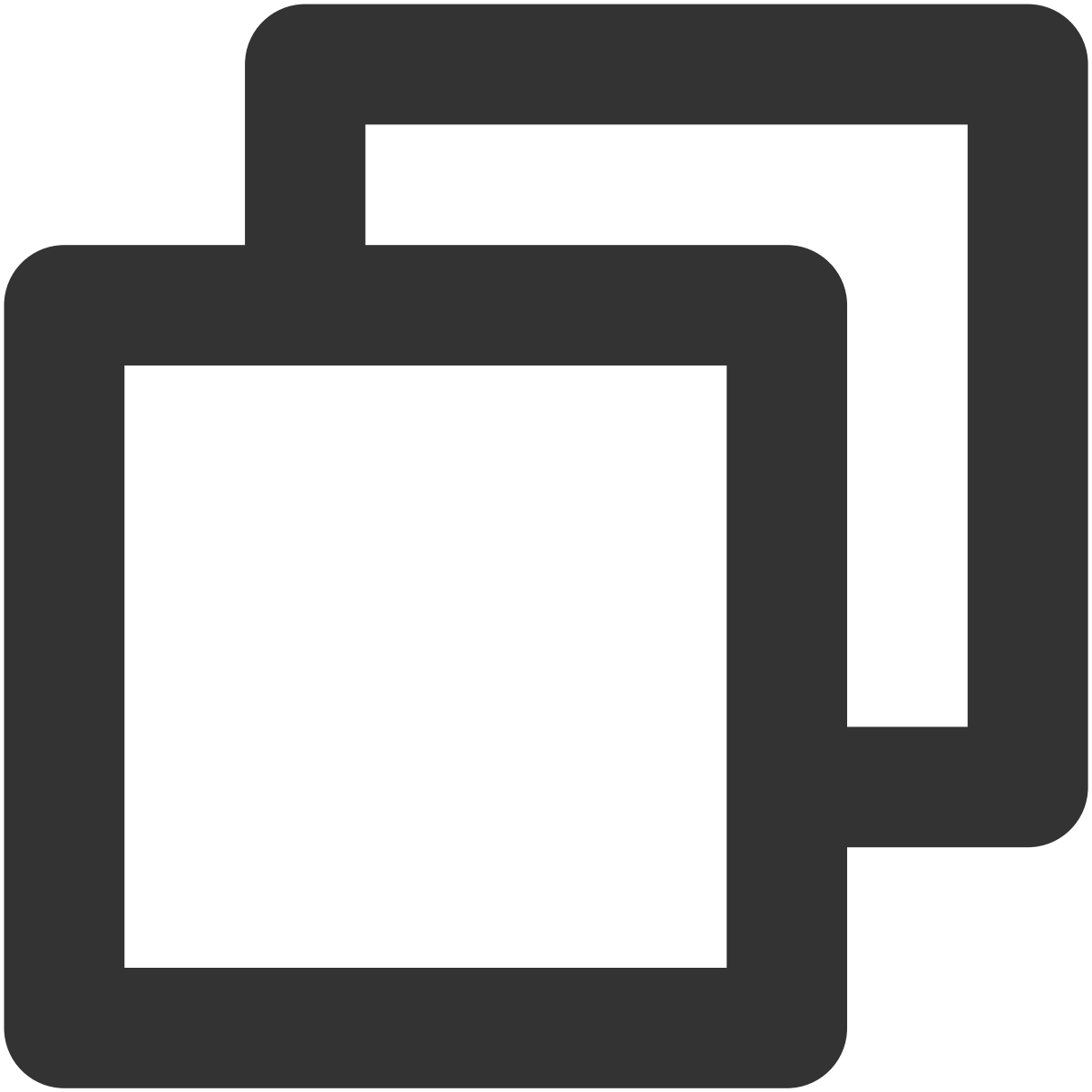
语法

Presto



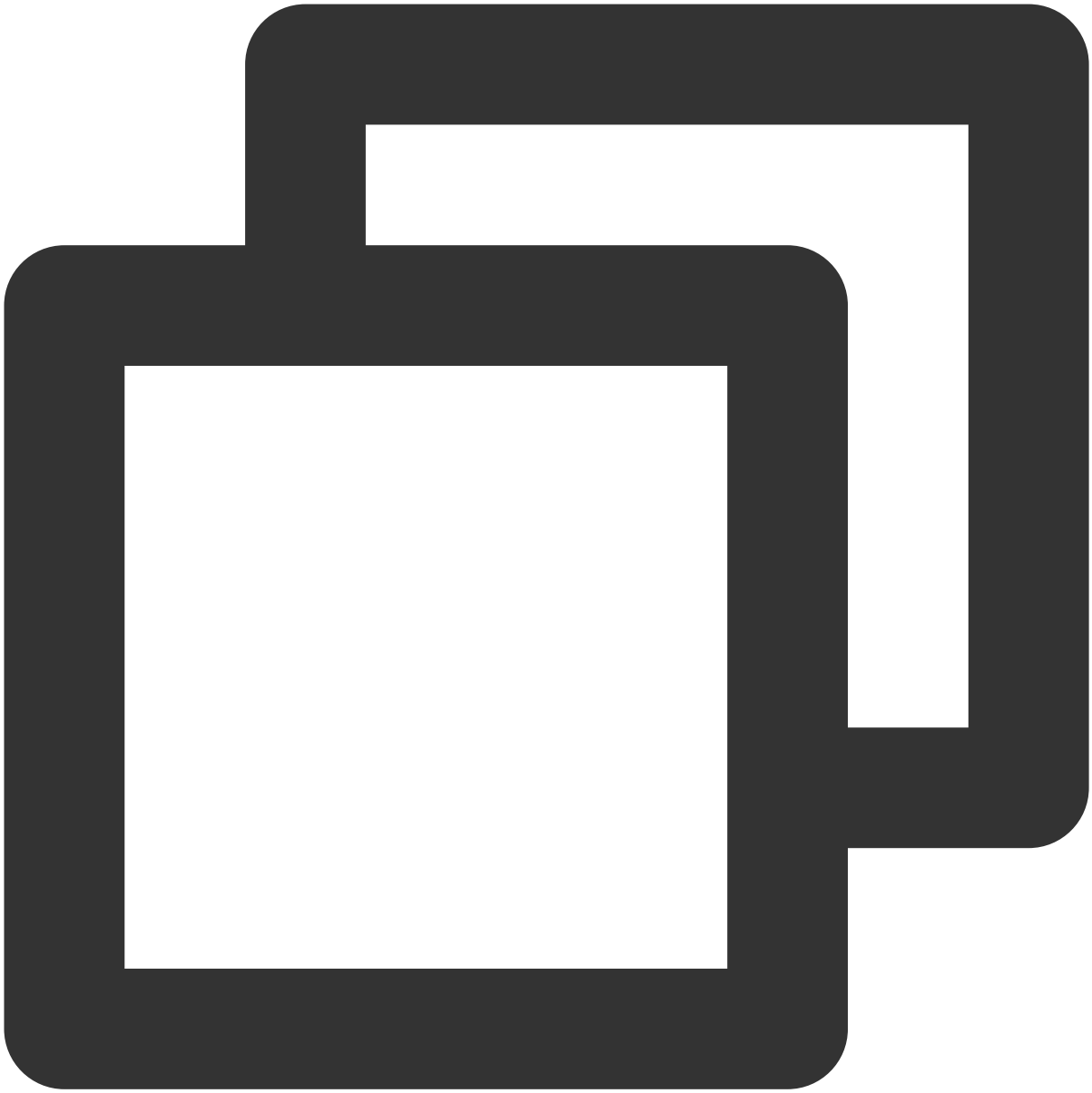
```
EXPLAIN [ ( option [, ...] ) ] statement
-- where option can be one of:
-- FORMAT { TEXT | GRAPHVIZ | JSON }
-- TYPE { LOGICAL | DISTRIBUTED | VALIDATE | IO }
```

SparkSQL



```
EXPLAIN [ EXTENDED | CODEGEN | COST | FORMATTED ] statement

EXPLAIN ANALYZE
EXPLAIN ANALYZE [VERBOSE] statement
```

示例



```
-- presto
EXPLAIN (TYPE VALIDATE) SELECT regionkey, count(*) FROM nation GROUP BY 1;
EXPLAIN (TYPE IO, FORMAT JSON) INSERT INTO test_nation SELECT * FROM nation WHERE r

-- EXPLAIN ANALYZE
EXPLAIN ANALYZE SELECT count(*), clerk FROM orders WHERE orderdate > date '1995-01-
EXPLAIN ANALYZE VERBOSE SELECT count(clerk) OVER() FROM orders WHERE orderdate > da
```

CALL STATEMENT

最近更新时间：2024-08-07 17:21:47

这里只有在使用 Spark 中使用 Iceberg SQL 扩展 时，存储过程才可用。

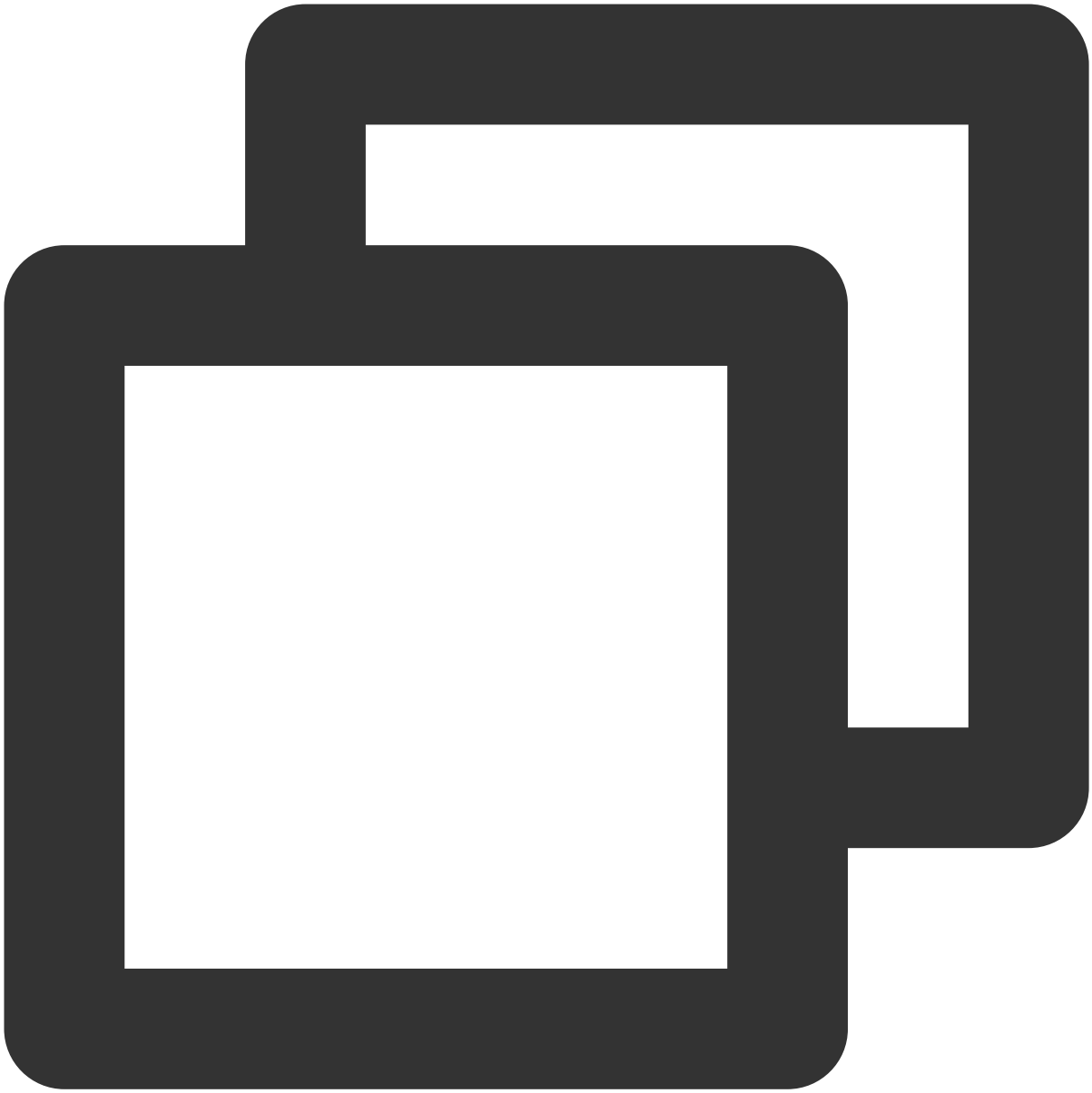
语法

```
CALL expression
```

参数

`expression` ：函数表达式。

示例



```
CALL catalog_name.`system`.procedure_name(arg_name_2 => arg_2, arg_name_1 => arg_1)

#当按位置传递参数时，如果它们是可选的，则只能省略结束参数。
CALL catalog_name.system.procedure_name(arg_1, arg_2, ... arg_n)

#将当前快照设置为db.sample1：
CALL catalog_name.system.set_current_snapshot('db.sample', 1)
```

更多使用

Spark Procedures。

版权所有：腾讯云计算（北京）有限责任公司

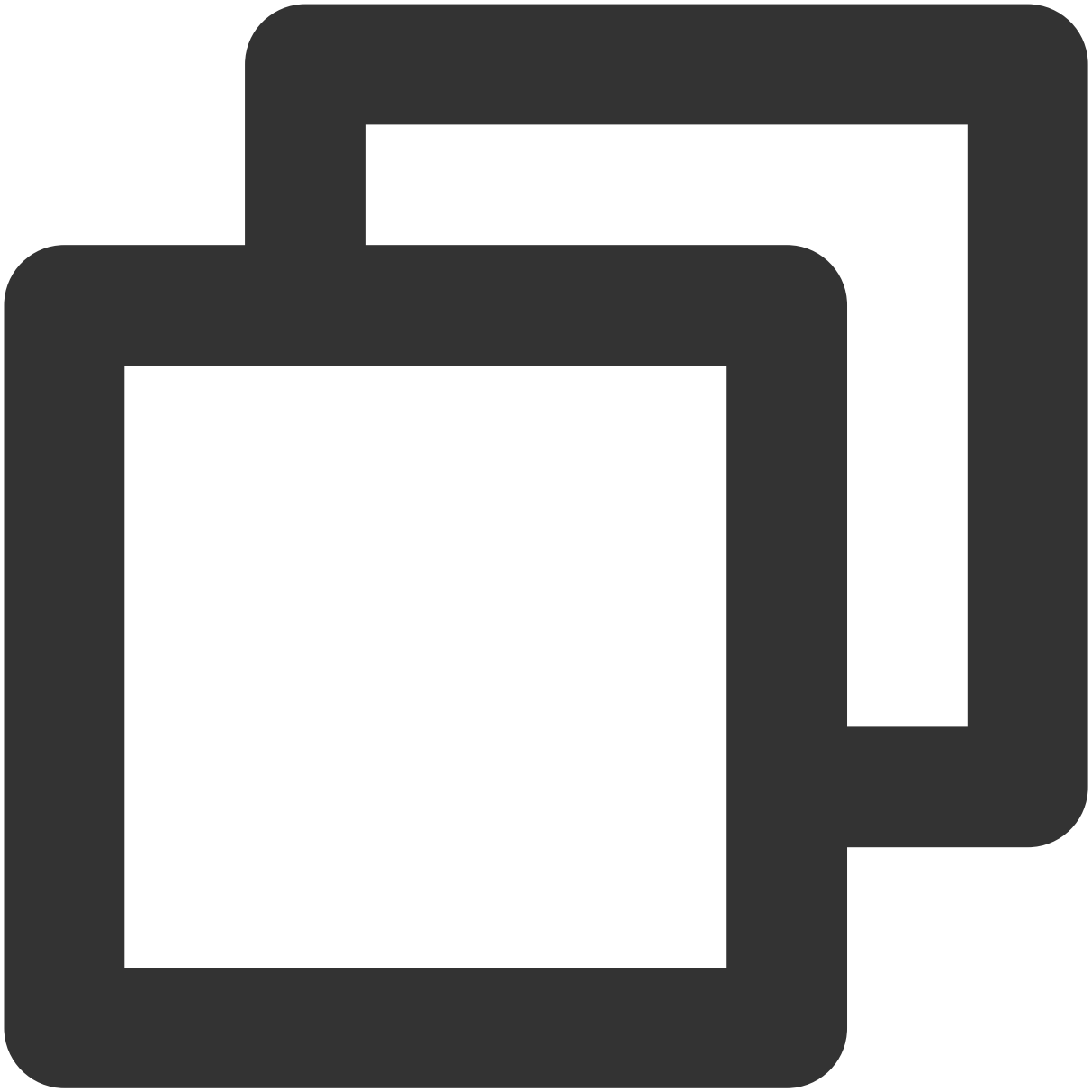
CREATE VIEW AS

最近更新时间：2024-08-07 17:22:17

说明

支持内核：Presto、SparkSQL。
用途：创建视图。

标准语法



```
CREATE [ OR REPLACE ] VIEW [IF NOT EXISTS] view_name
    [(column_name [COMMENT 'column_comment'][, ...])]
    [COMMENT 'view_comment']
  AS select_statement
```

参数

`[IF NOT EXISTS]` ：不存在则创建。

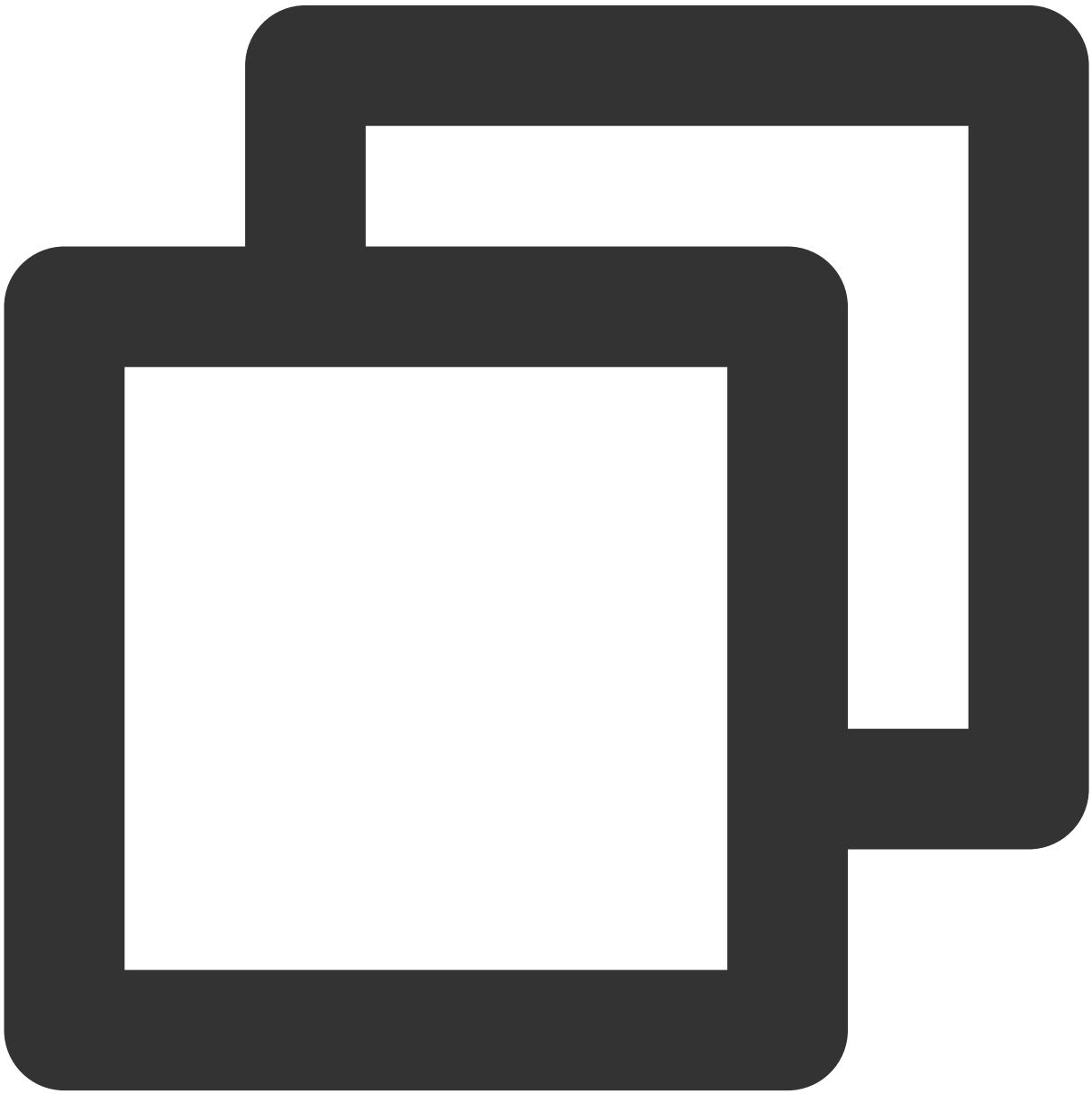
`view_name` ：视图名。

`[(column_name [COMMENT 'column_comment'][, ...])]` ：列的名字，同时后面可以带上列的注释。

`[COMMENT 'view_comment']` ：视图的注释。

`select_statement` ：查询语句。

示例



```
create or replace view db1.v1 as select x,y from tbl;
```

```
create view test_view (id comment 'test c1', name_length comment 'test name c2') as
```

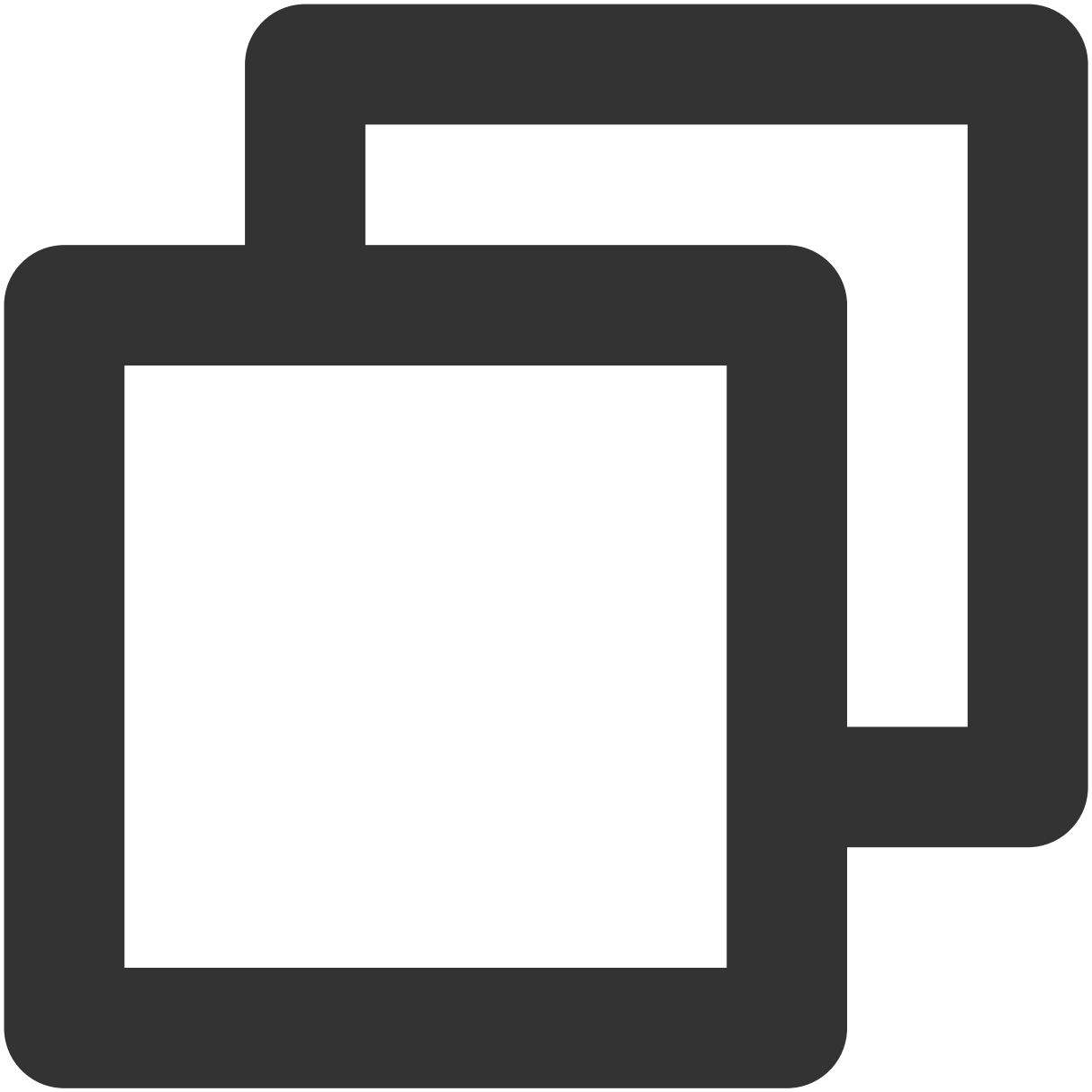
SHOW VIEWS

最近更新时间：2024-08-07 17:22:31

说明

支持内核：Presto、SparkSQL。
用途：列出指定数据库中的视图，如果省略数据库名称，则列出当前数据库中的视图。

语法



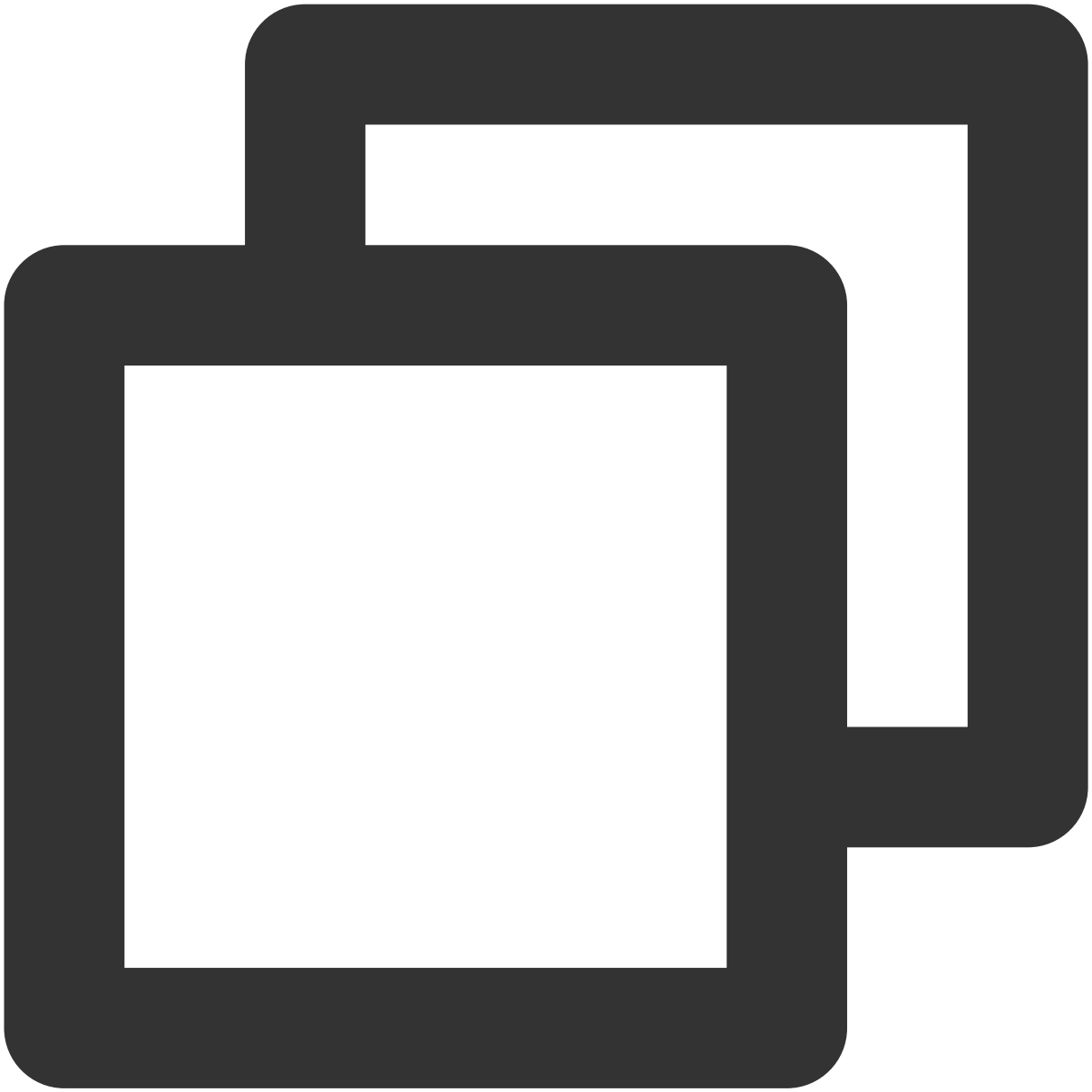
```
SHOW VIEWS [IN database_name] LIKE ['regular_expression']
```

参数

`[IN database_name]` ：指定要从中列出视图的数据库名称。如果省略，则假定当前上下文中的数据库。

`-` ：将视图列表筛选为与指定的常规表达式匹配的视图。只能使用表示任意字符的通配符*，或表示字符之间可供选择。

示例



```
SHOW VIEWS;

SHOW VIEWS IN db01 LIKE 'view*';
```

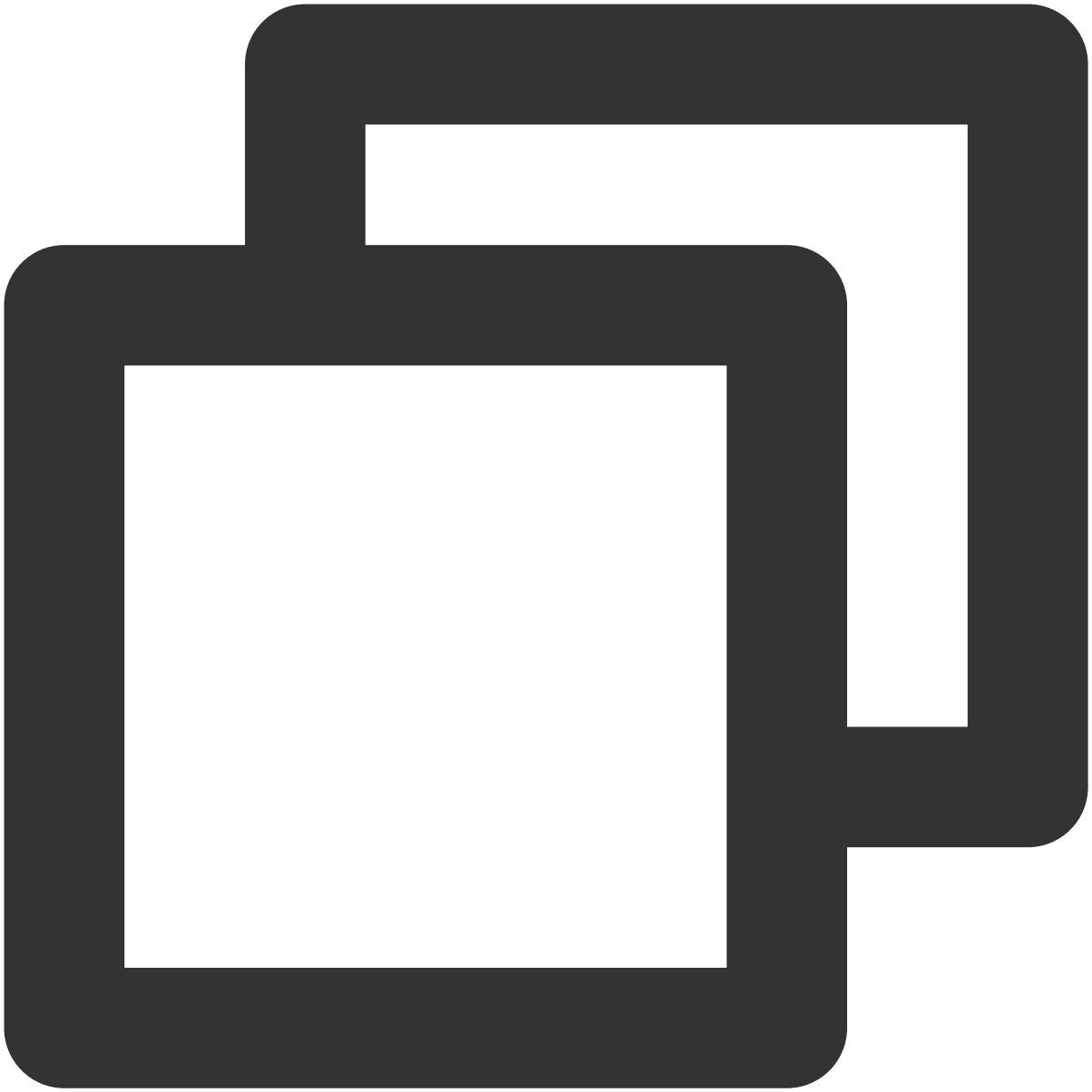
DESCRIBE VIEW

最近更新时间：2024-08-07 17:22:45

说明

支持内核：Presto、SparkSQL。
用途：查看视图属性。

语法

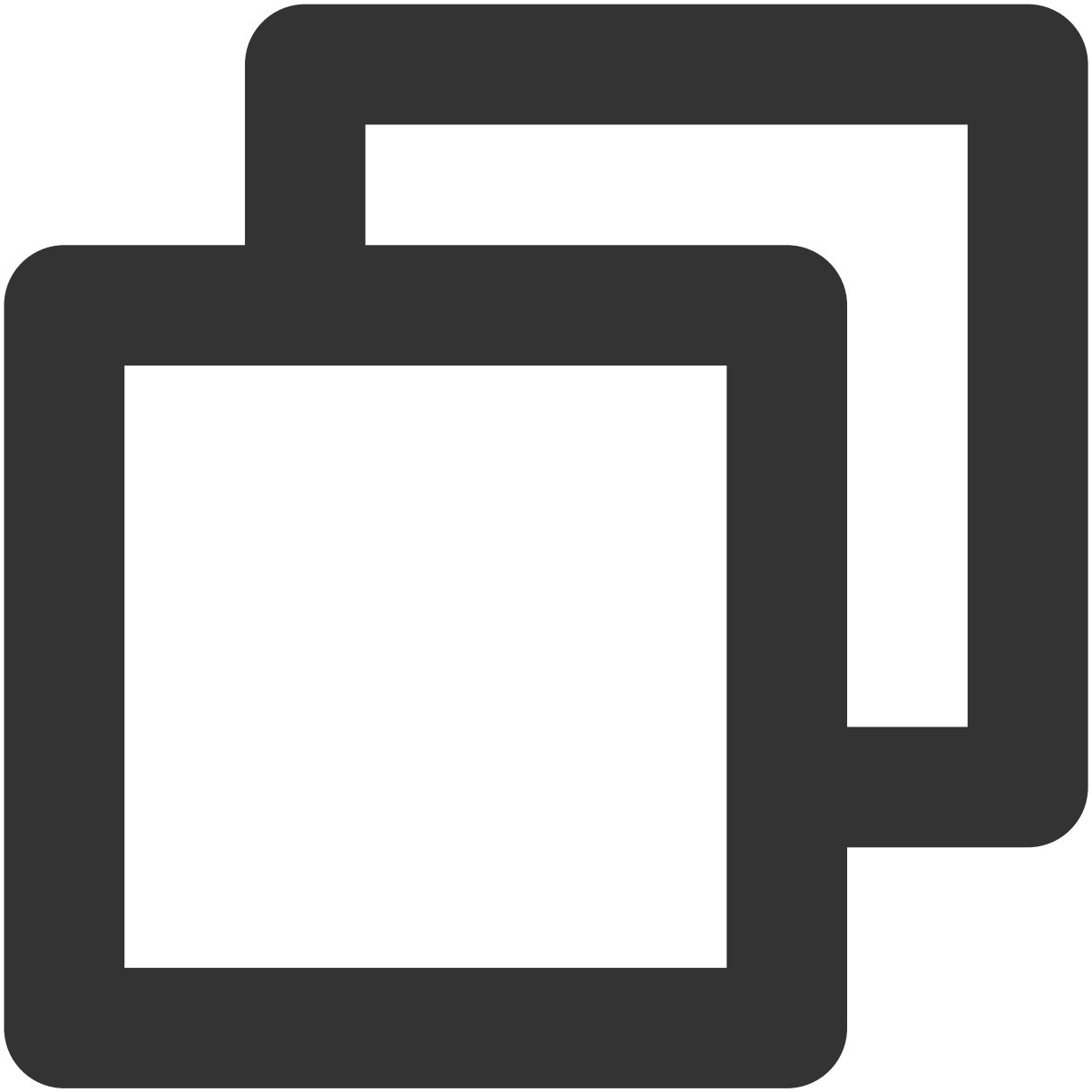


```
DESCRIBE [view_name]
```

参数

view_name：视图名。

示例



```
DESCRIBE view1;
```

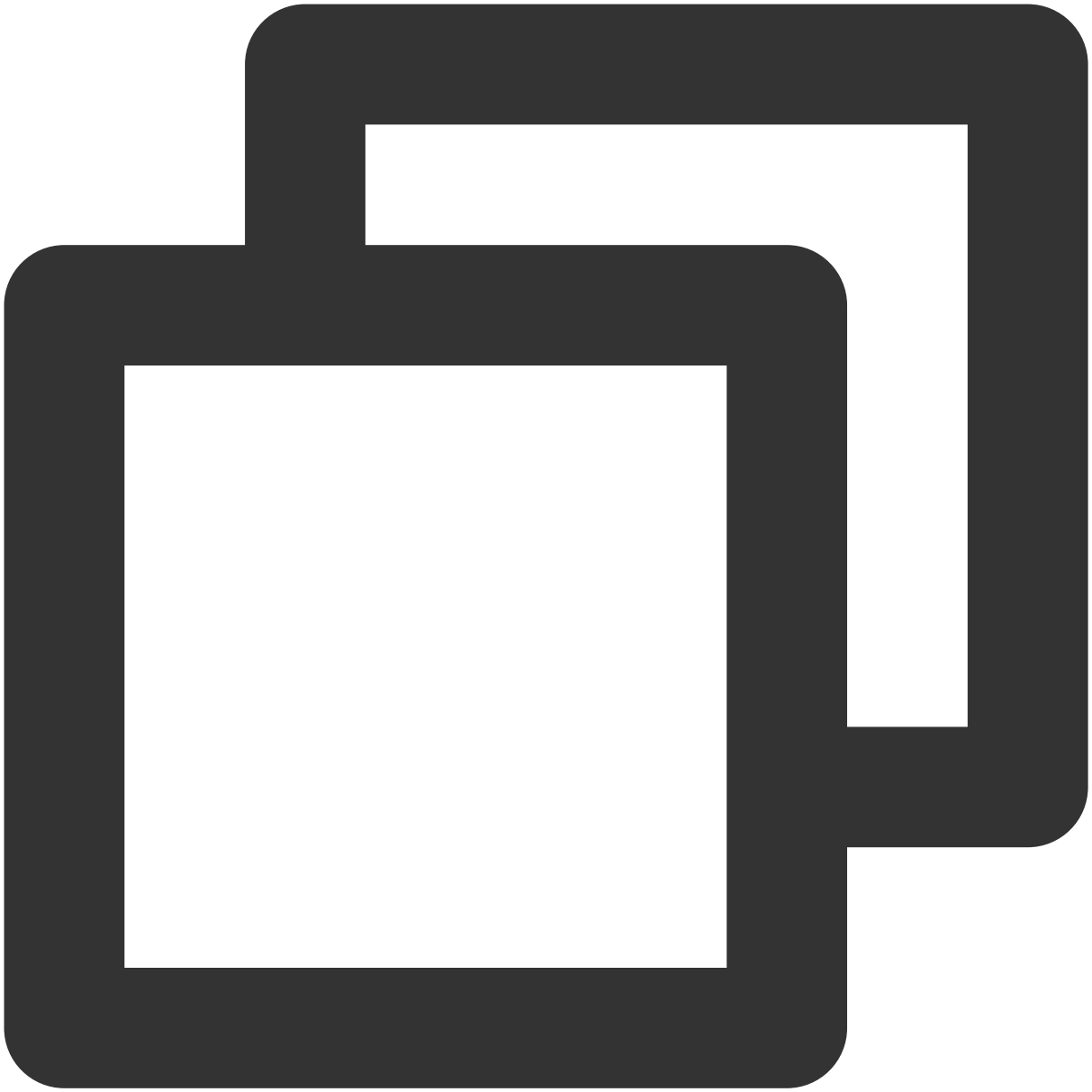
SHOW CREATE VIEW

最近更新时间：2024-08-07 17:23:02

说明

支持内核：Presto、SparkSQL。
用途：展示创建视图的语句。

语法

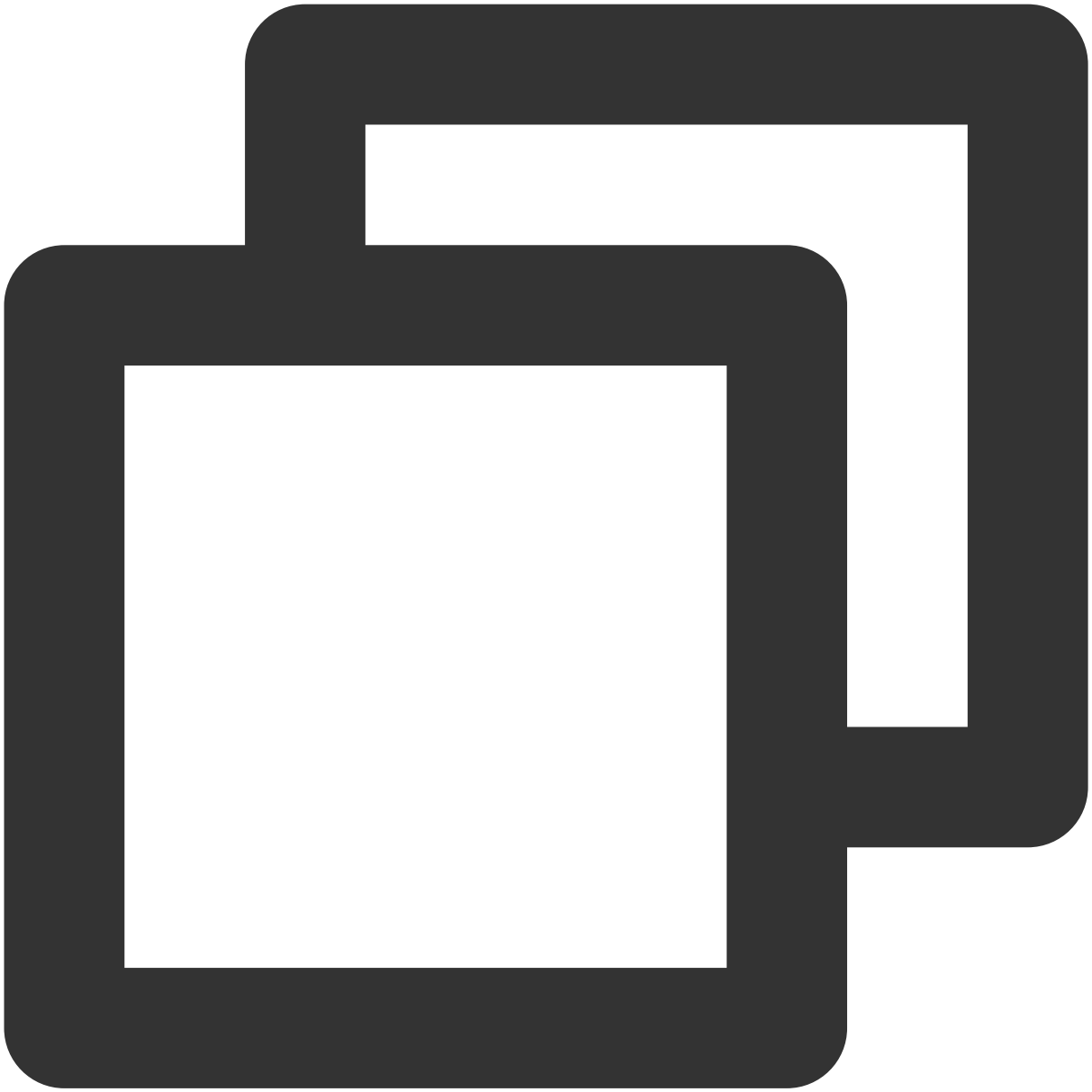


```
SHOW CREATE VIEW view_name
```

参数

`view_name` ：视图名称。

示例



```
SHOW CREATE VIEW orders_by_date
```

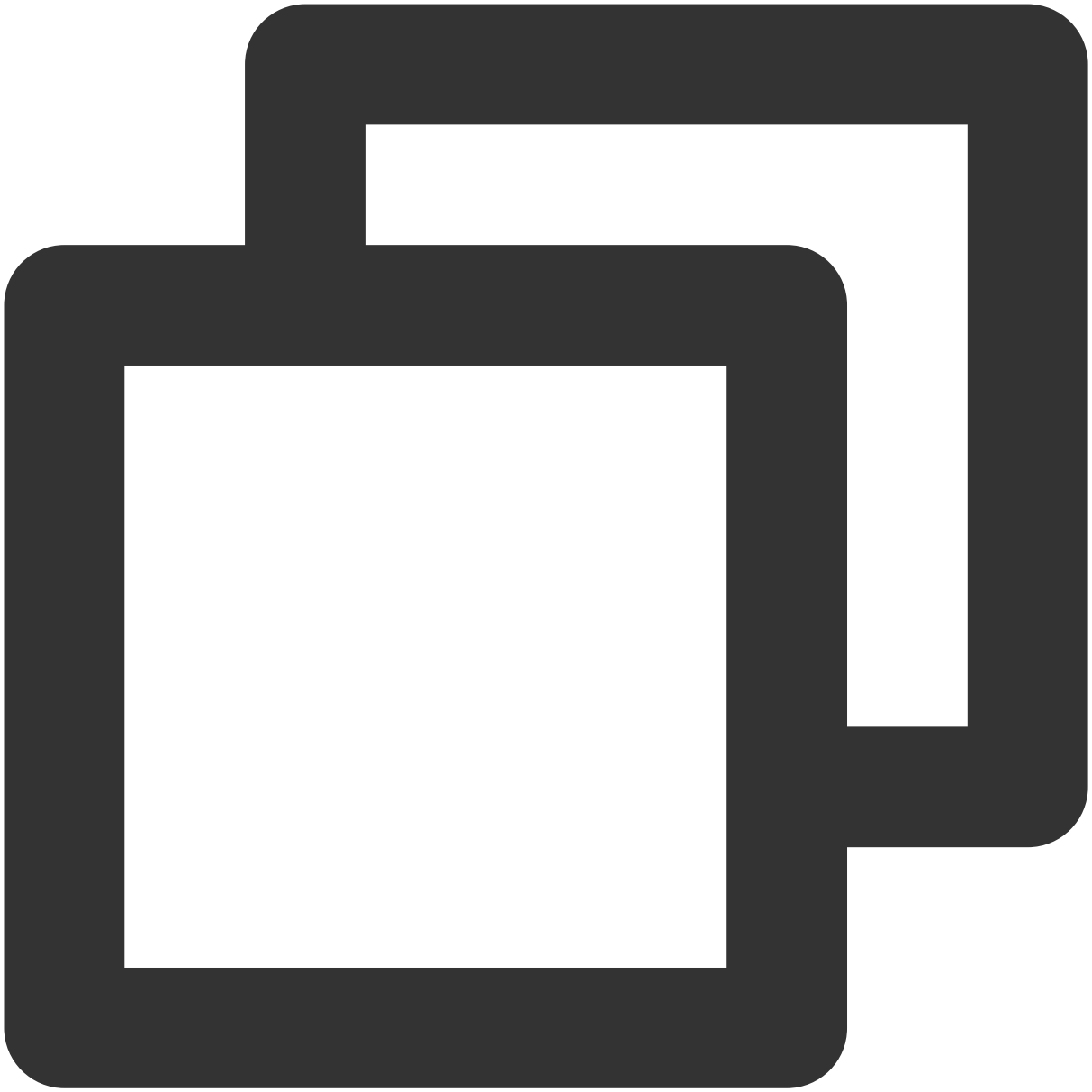
SHOW COLUMNS IN VIEW

最近更新时间：2024-08-07 17:23:16

基础说明

支持内核：Presto、SparkSQL。
用途：查看视图的列信息。

标准语法

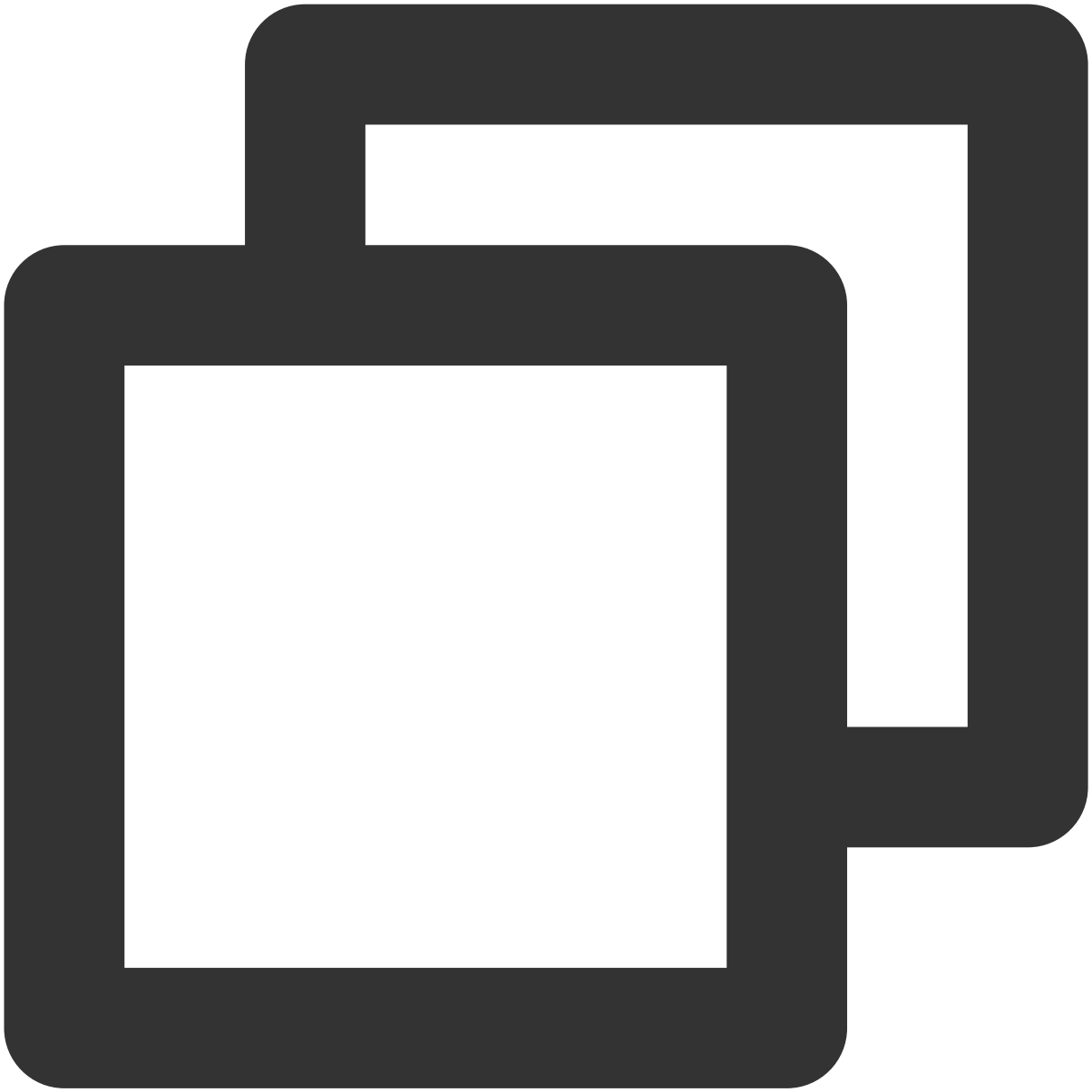


```
SHOW COLUMNS IN view_name;
```

参数

`view_name` ：视图名称。

示例



```
SHOW COLUMNS IN view_test
```

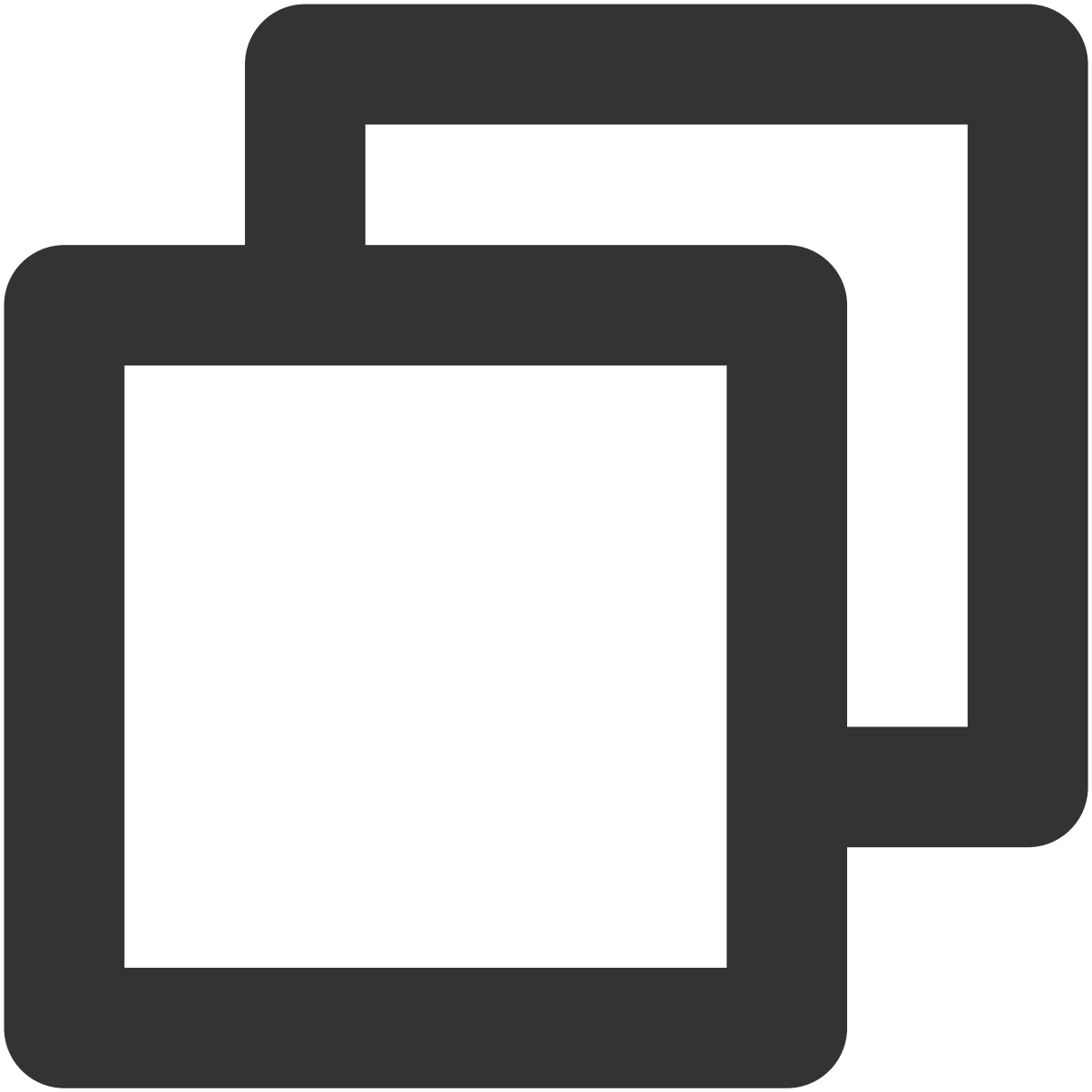
ALTER VIEW
ALTER VIEW RENAME TO

最近更新时间：2024-08-07 17:23:41

说明

支持内核：Presto、SparkSQL。
用途：变更视图名称。

标准语法

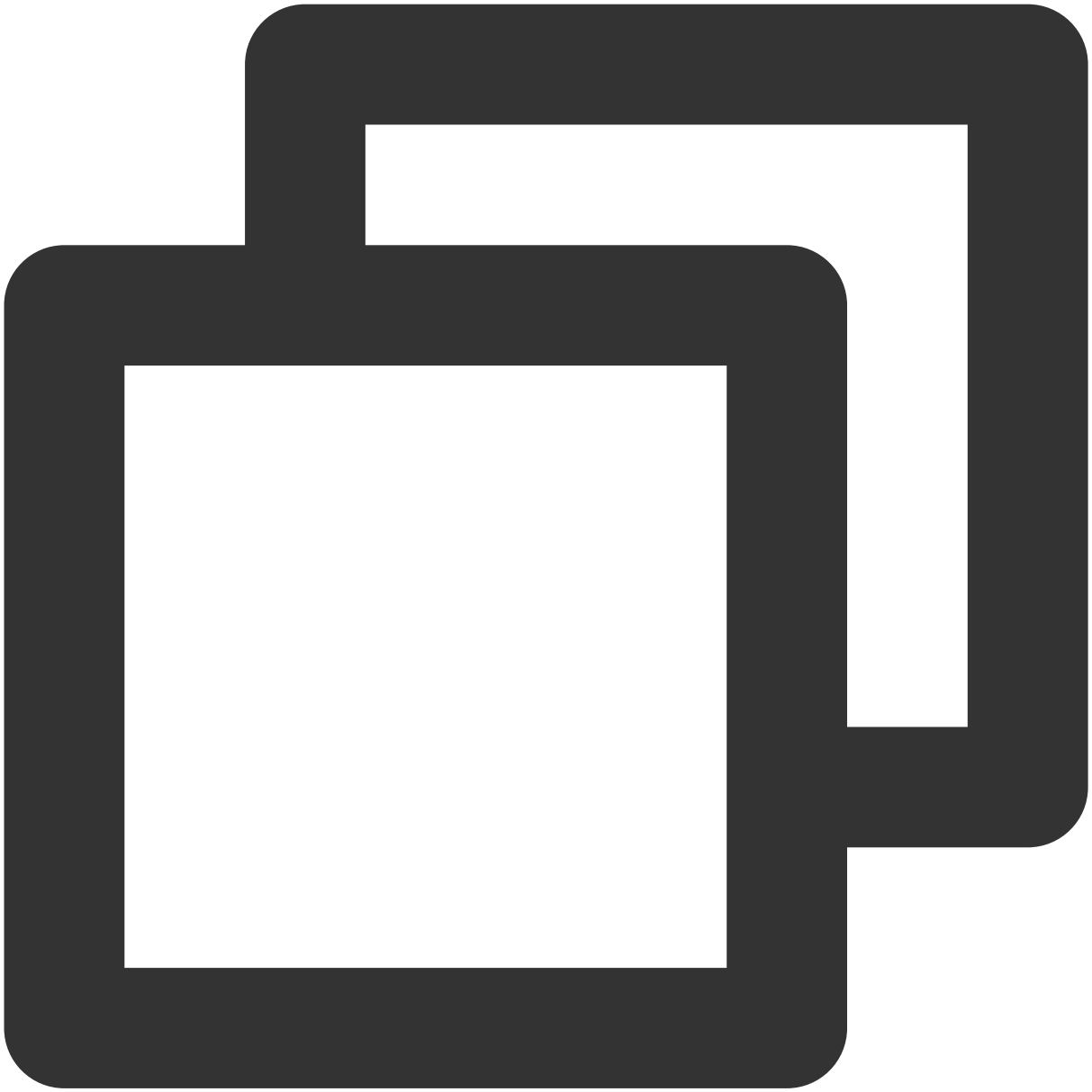


```
ALTER VIEW old_view_identifier RENAME TO new_view_identifier
```

参数

`old_view_identifier` ：修改前视图的名称。
`new_view_identifier` ：修改后视图的名称。

示例



```
alter view old_view rename to new_view
```

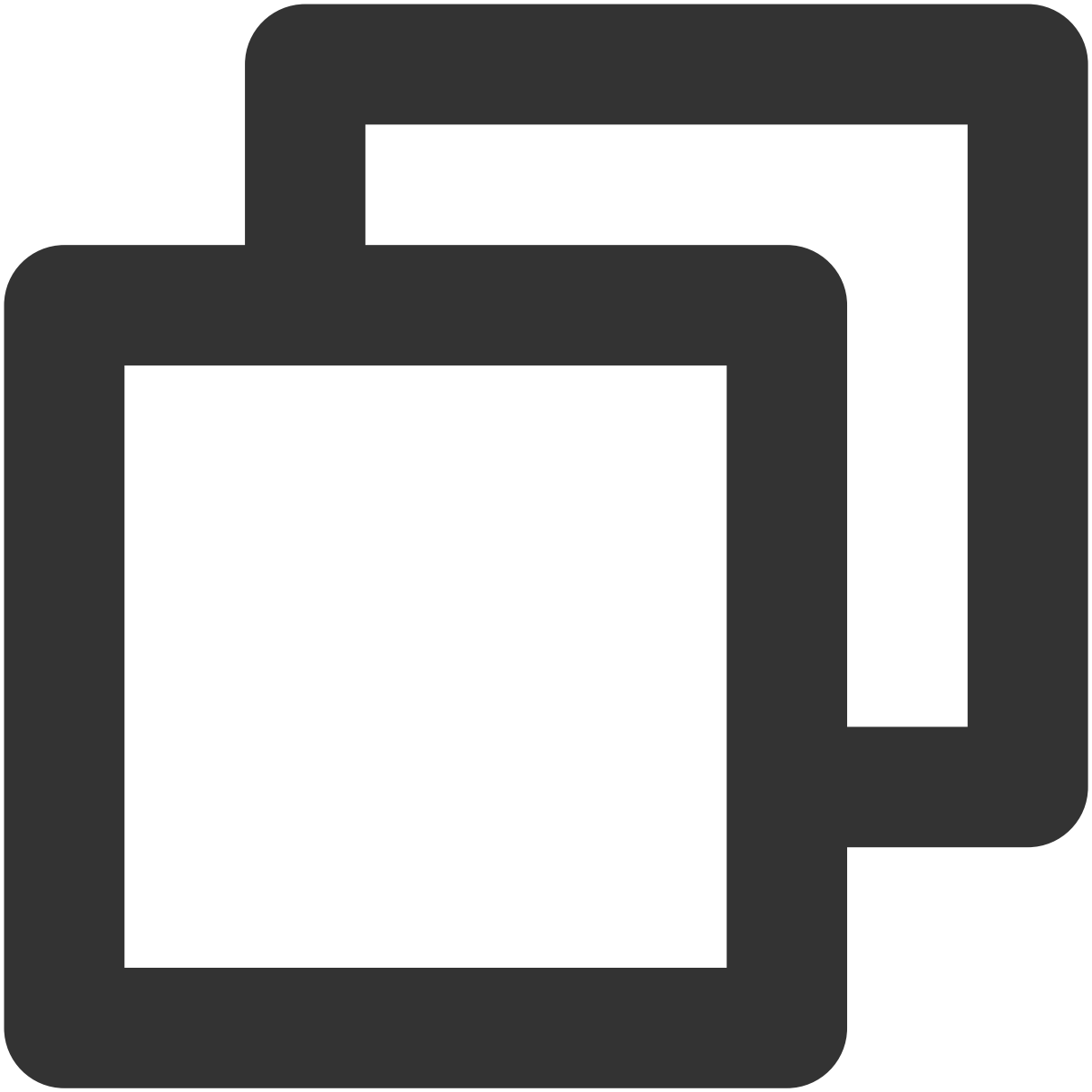
ALTER VIEW SET TBLPROPERTIES

最近更新时间：2024-08-07 17:24:06

说明

支持内核：Presto、SparkSQL。
用途：修改视图属性。

标准语法



```
ALTER VIEW view_identifier SET TBLPROPERTIES ( property_key = property_val [ , ...
```

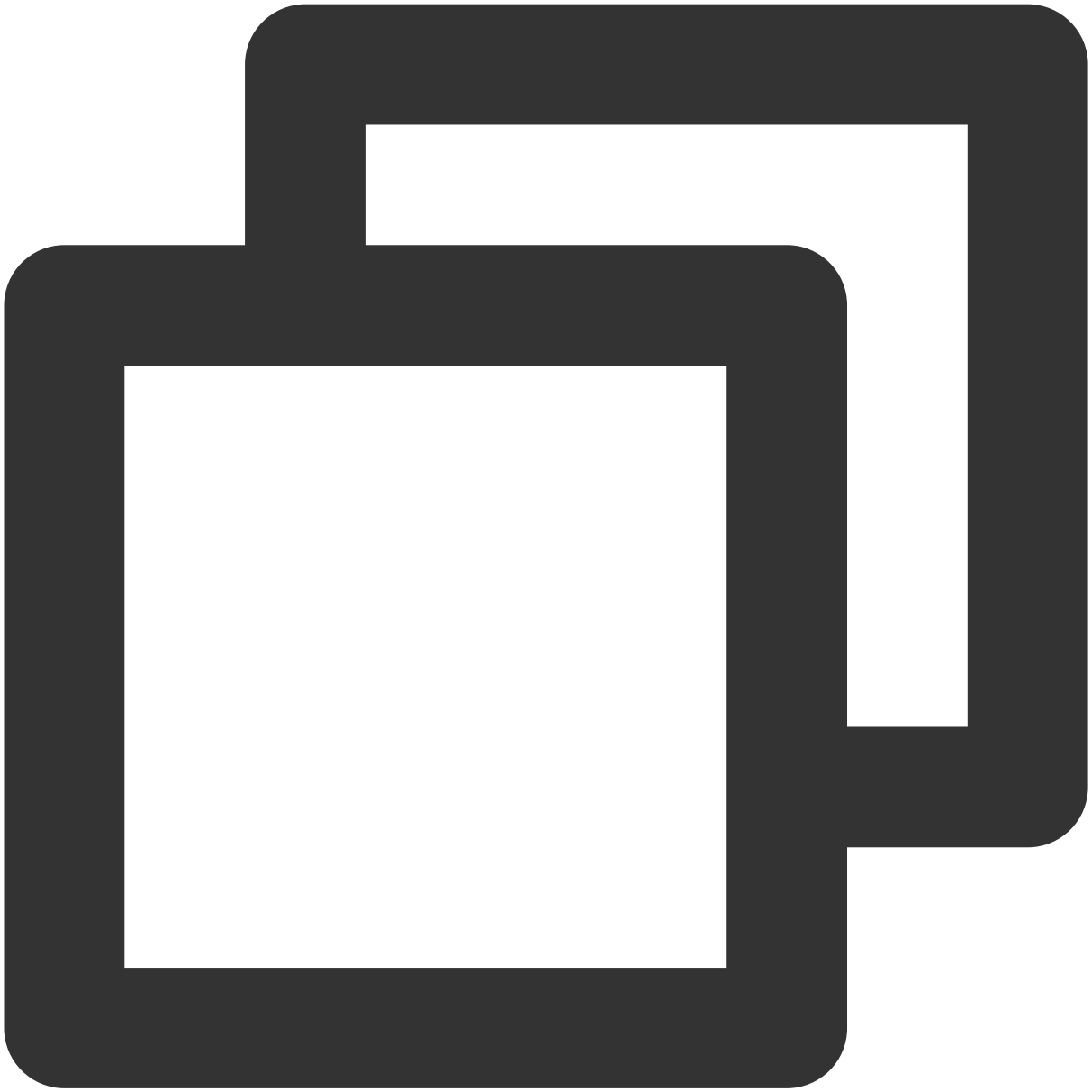
参数

`view_identifier` ：需要修改属性视图的名称。

`property_key` ：属性名称。

`property_val` ：属性值。

示例



```
alter view view1 set tblproperties('comment' = 'view1')

alter view view1 set tblproperties('property1' = 'value1', 'property2' = 'value2')
```

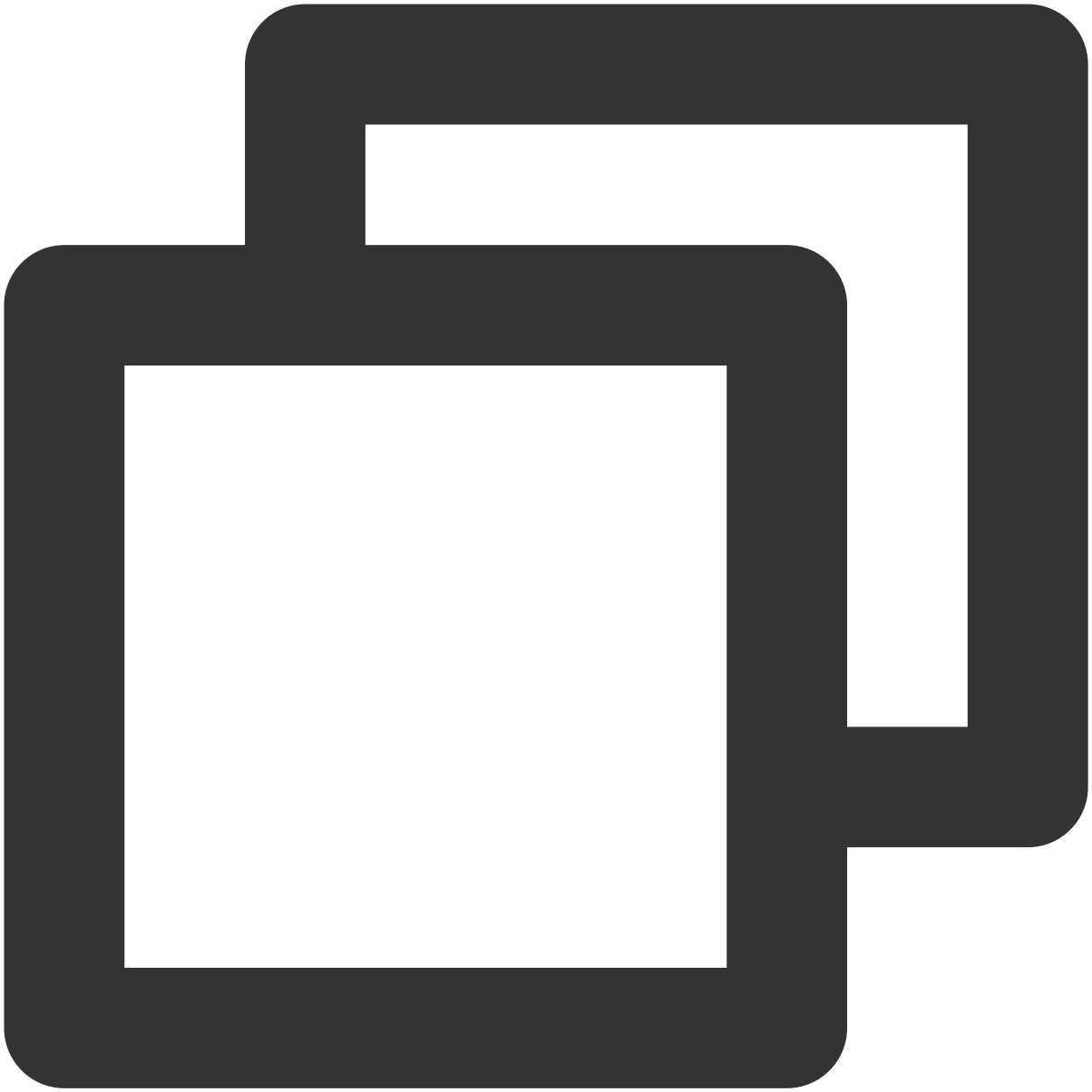
DROP VIEW

最近更新时间：2024-08-07 17:24:20

说明

支持内核：Presto、SparkSQL。
用途：删除视图。

语法



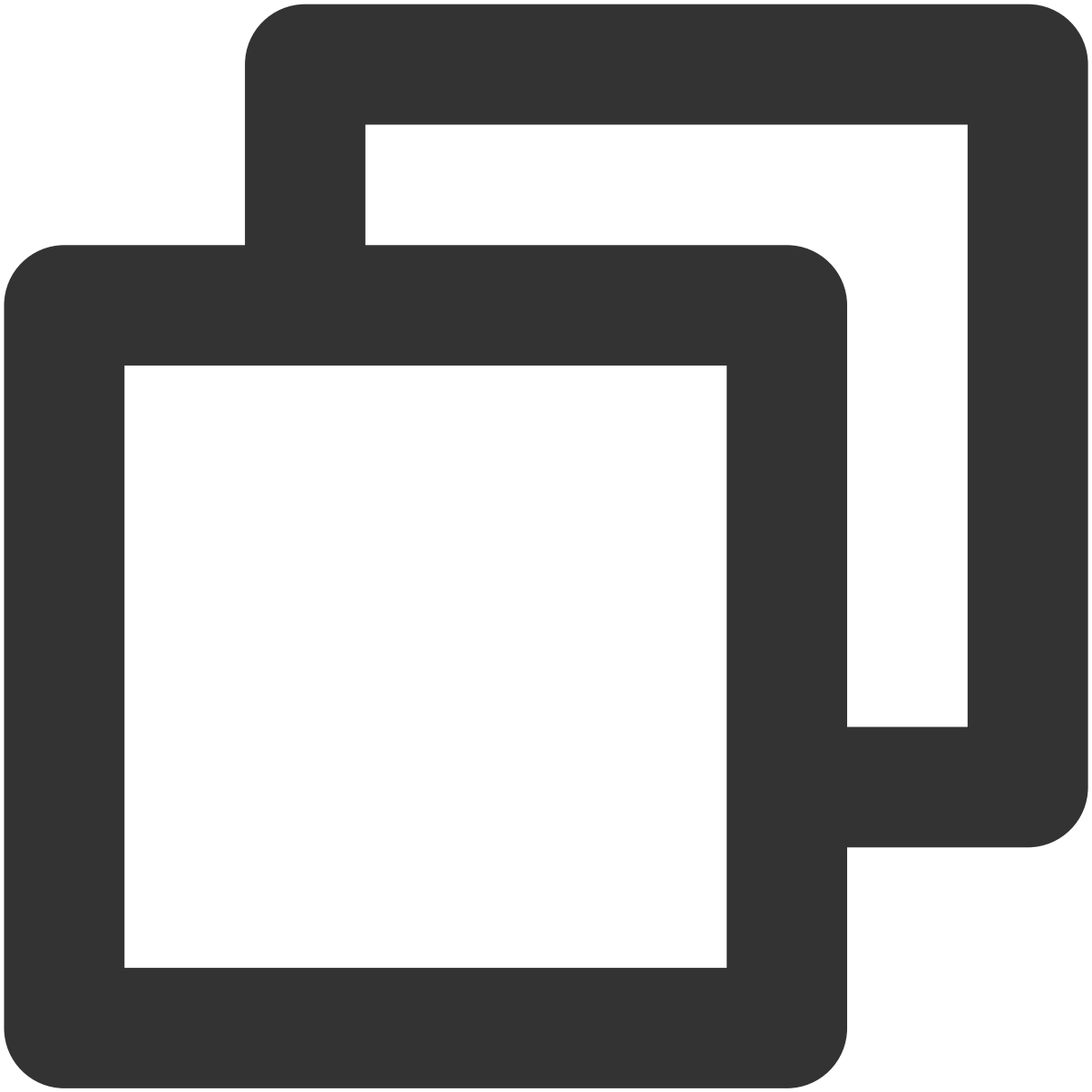
```
DROP VIEW [ IF EXISTS ] view_name;
```

参数

`IF EXISTS` ：可选，如果存在的意思。

`view_name` ：视图名称。

示例



```
DROP VIEW orders_by_date;
DROP VIEW IF EXISTS orders_by_date;
```

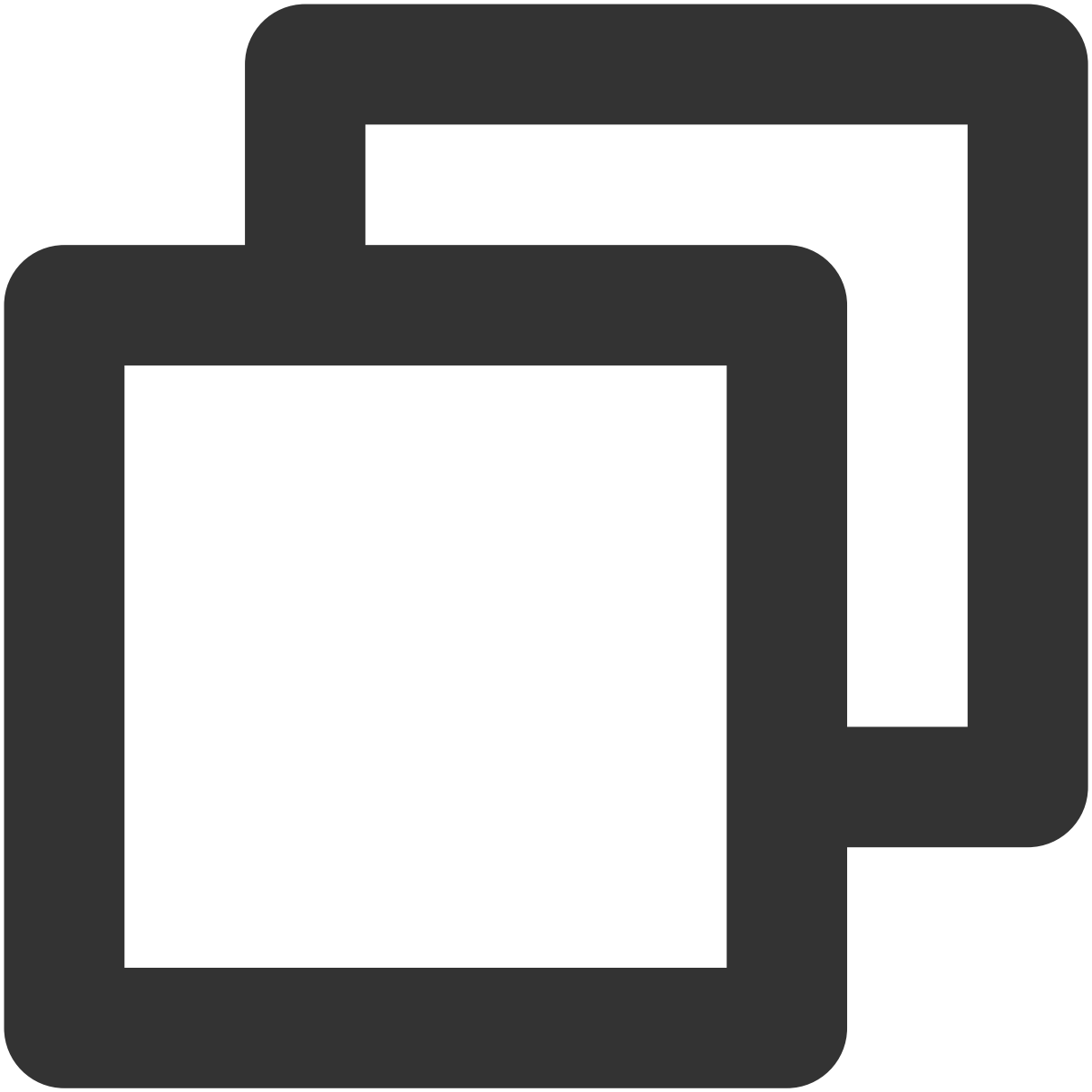
CREATE FUNCTION

最近更新时间：2024-08-07 17:24:31

说明

支持内核：Presto、SparkSQL。
用途：创建一个由类名实现的函数。

语法

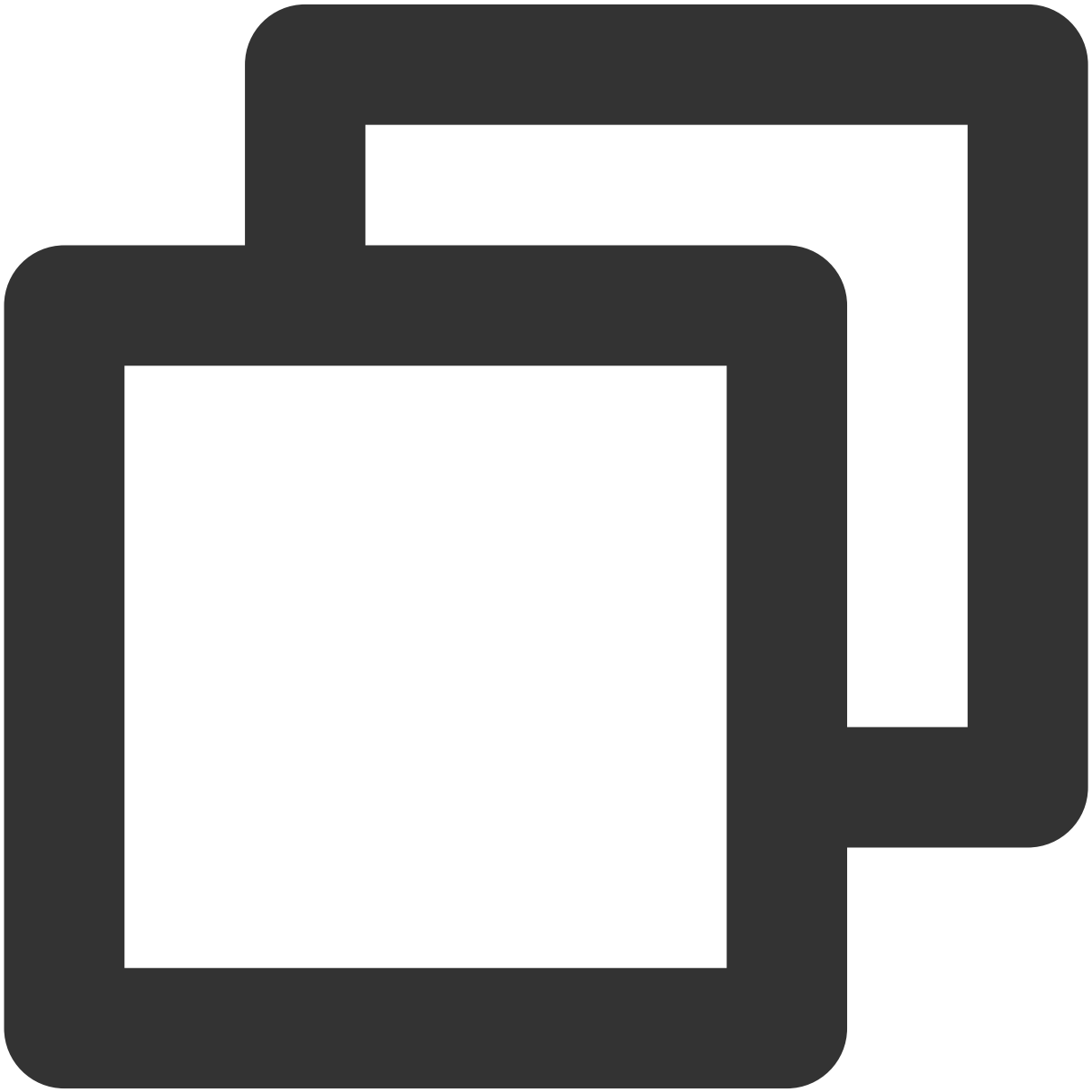


```
CREATE  FUNCTION [db_name.]function_name AS class_name
 [USING JAR|FILE|ARCHIVE 'file_uri' [, JAR|FILE|ARCHIVE 'file_uri'] ];

-- 函数后缀名为"_udtf"时候，会被识别为UDTF函数
CREATE FUNCTION [db_name].function_udtf AS class_name
 [USING JAR|FILE|ARCHIVE 'file_uri' [, JAR|FILE|ARCHIVE 'file_uri'] ];
```

参数

`[db_name.]function_name` ：函数名称，创建函数的时候后指定命名空间在 `db_name` 下。
`class_name` ：函数的实现类。
`USING JAR|FILE|ARCHIVE 'file_uri'` ：函数资源的路径。

示例



```
CREATE FUNCTION `MYFUNC` AS 'myclass' USING JAR 'hdfs:///path/to/jar'
```

```
CREATE FUNCTION `MYFUNC` AS 'myclass' USING JAR 'hdfs:///path/to/jar', FILE 'file:/
```

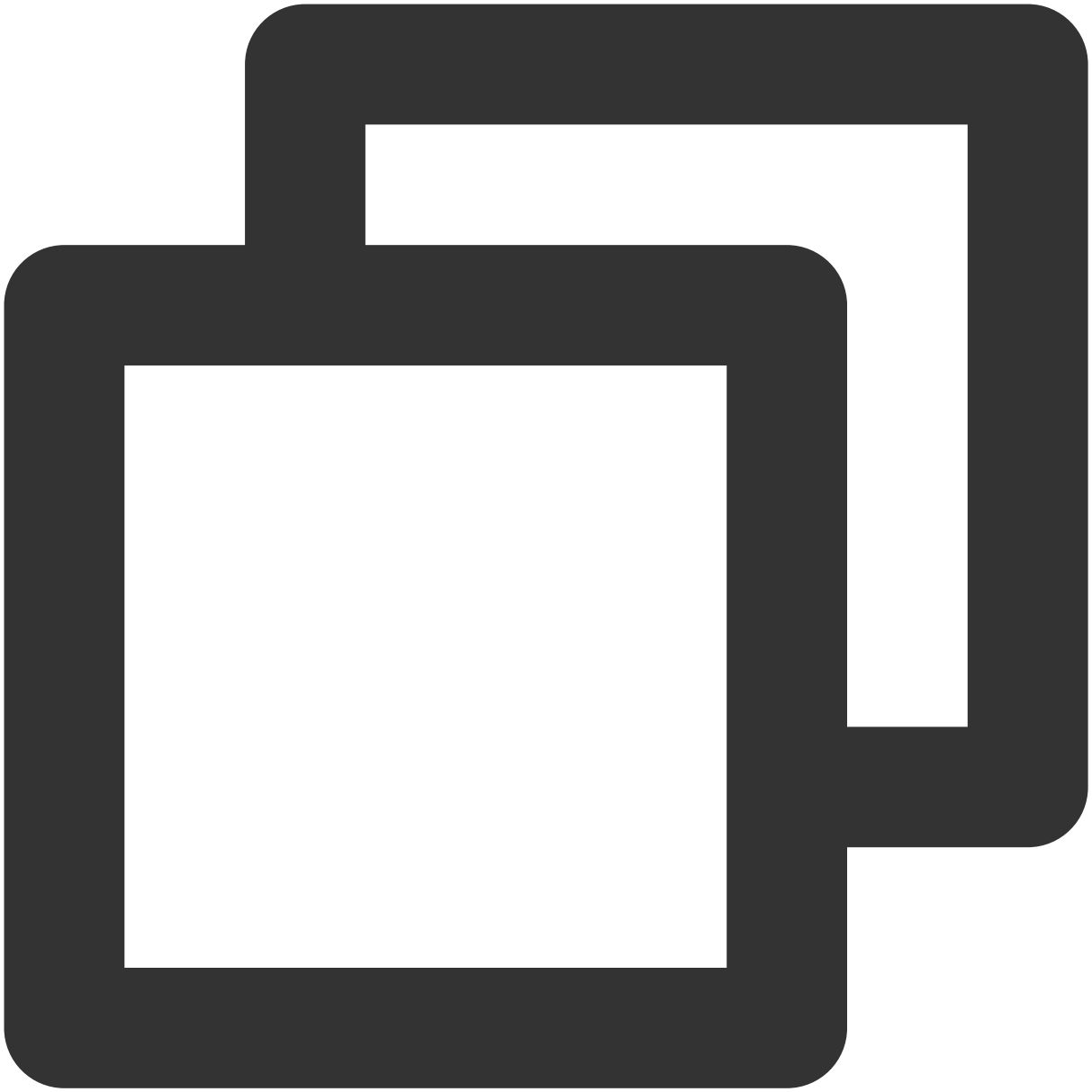
SHOW FUNCTION

最近更新时间：2024-08-07 17:24:50

说明

支持内核：Presto、SparkSQL。
用途：显示函数创建语法。

语法



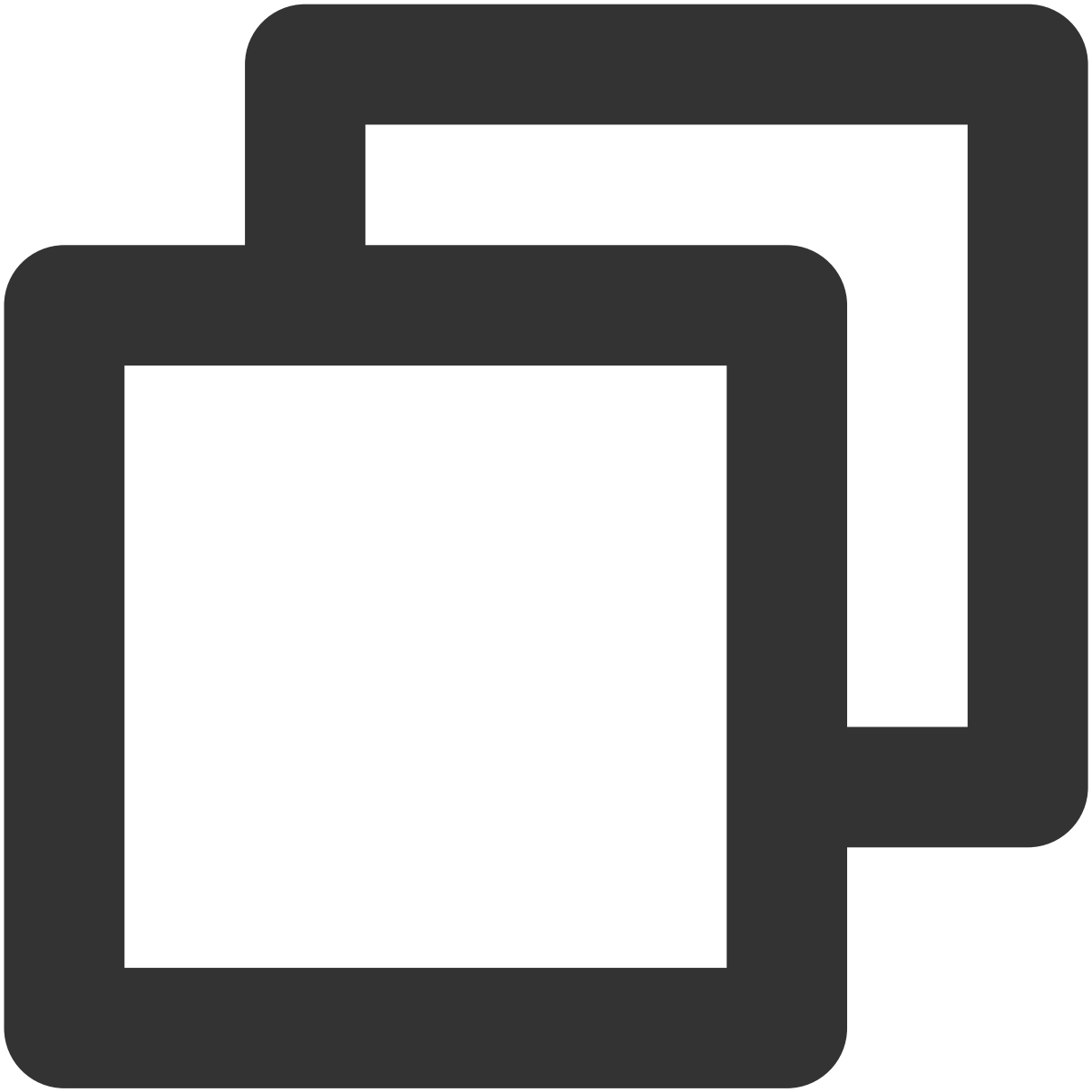
```
SHOW FUNCTIONS [ [ LIKE ] { function_name | regex_pattern }
```

参数

`function_name` ：函数名称。

`regex_pattern` ：需要过滤出函数名称的正则表达式。

示例



```
SHOW FUNCTIONS

SHOW FUNCTIONS trim;

SHOW FUNCTIONS LIKE 't*'
SHOW FUNCTIONS LIKE 'yea*|windo*'
SHOW FUNCTIONS LIKE 't[a-z][a-z][a-z]'
```

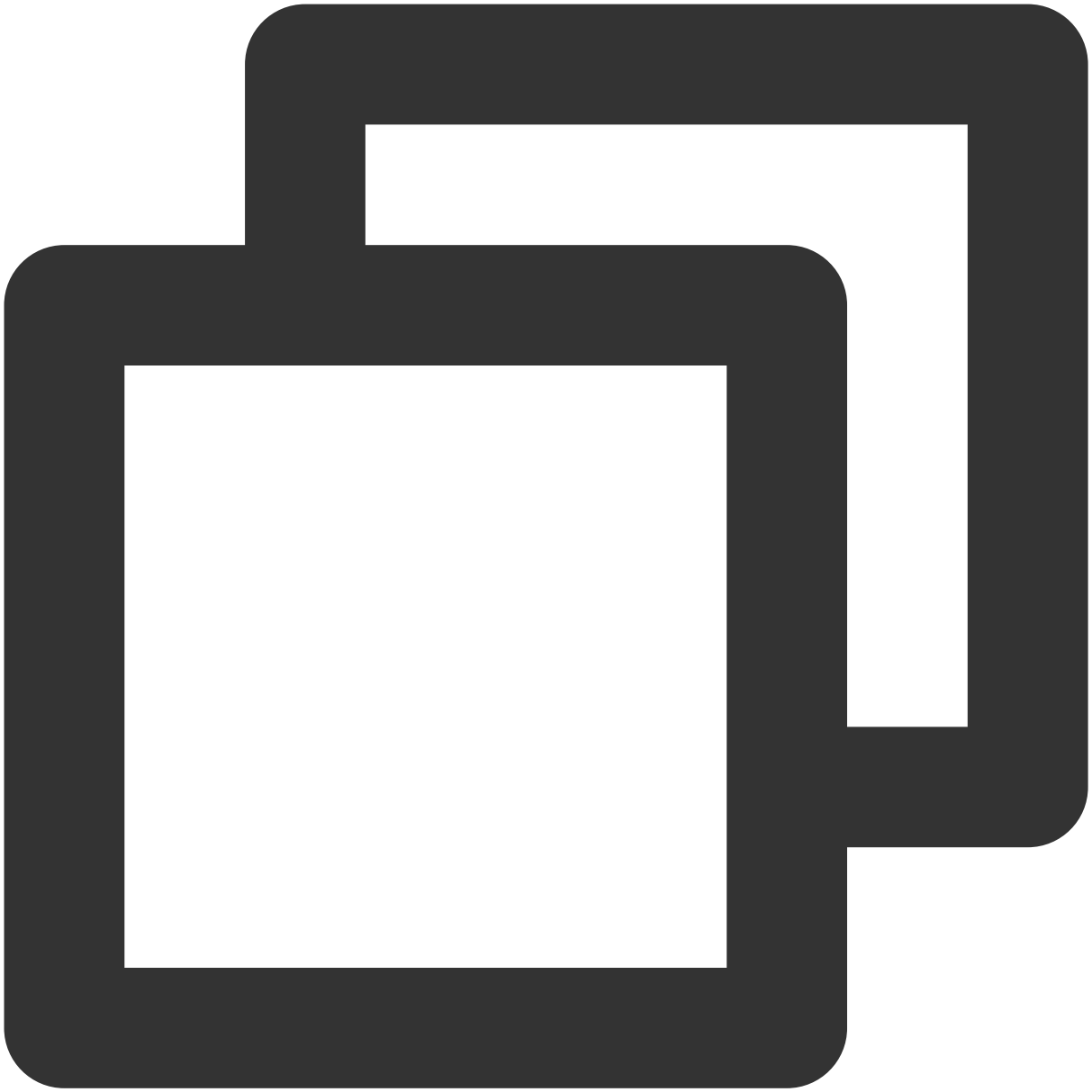
DROP FUNCTION

最近更新时间：2024-08-07 17:25:08

说明

支持内核：Presto、SparkSQL。
用途：删除自定义的函数。

语法

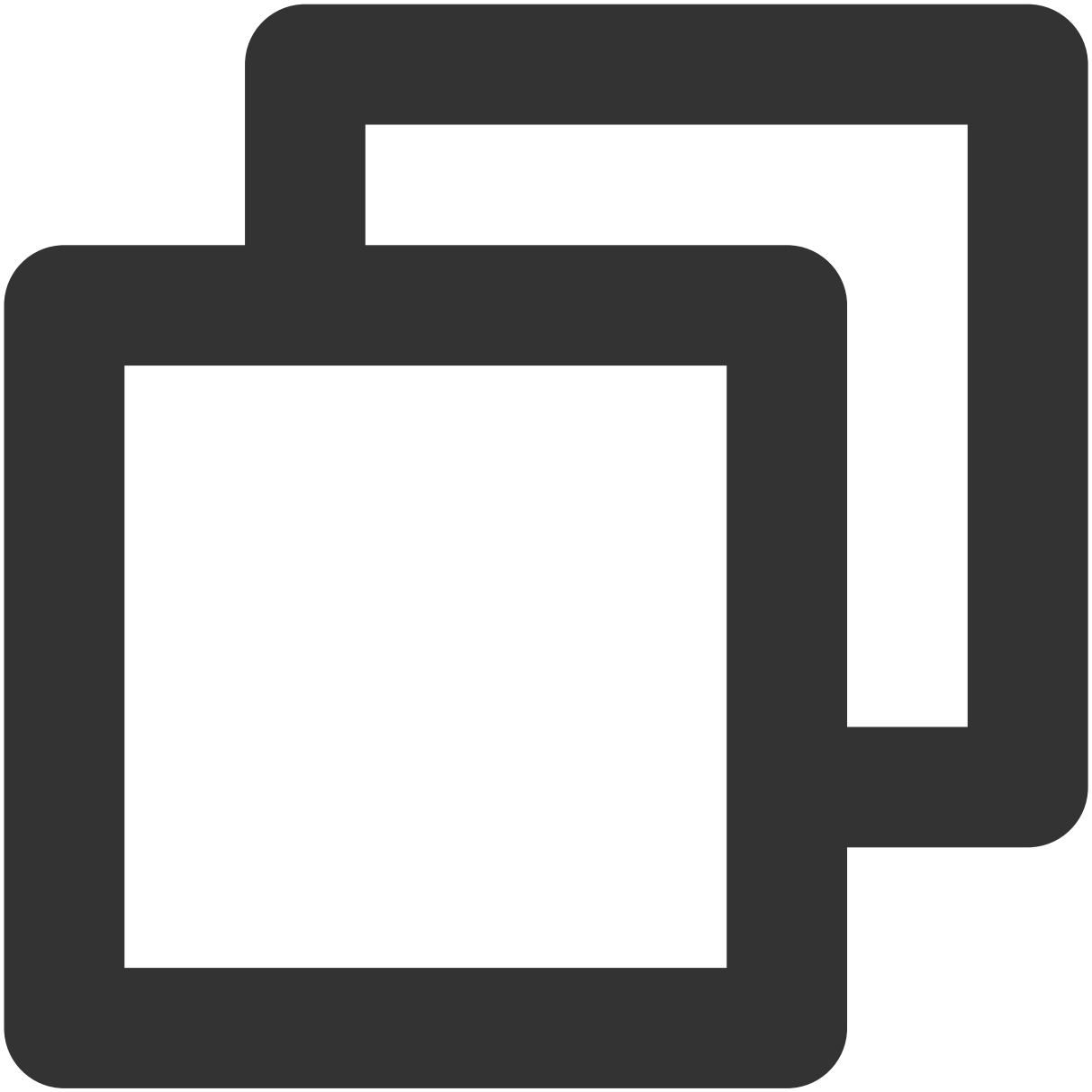


```
DROP FUNCTION [IF EXISTS] function_name
```

参数

`function_name` ：需要删除的函数名称。

示例



```
DROP FUNCTION IF EXISTS `FUNC`

DROP FUNCTION `FUNC
```

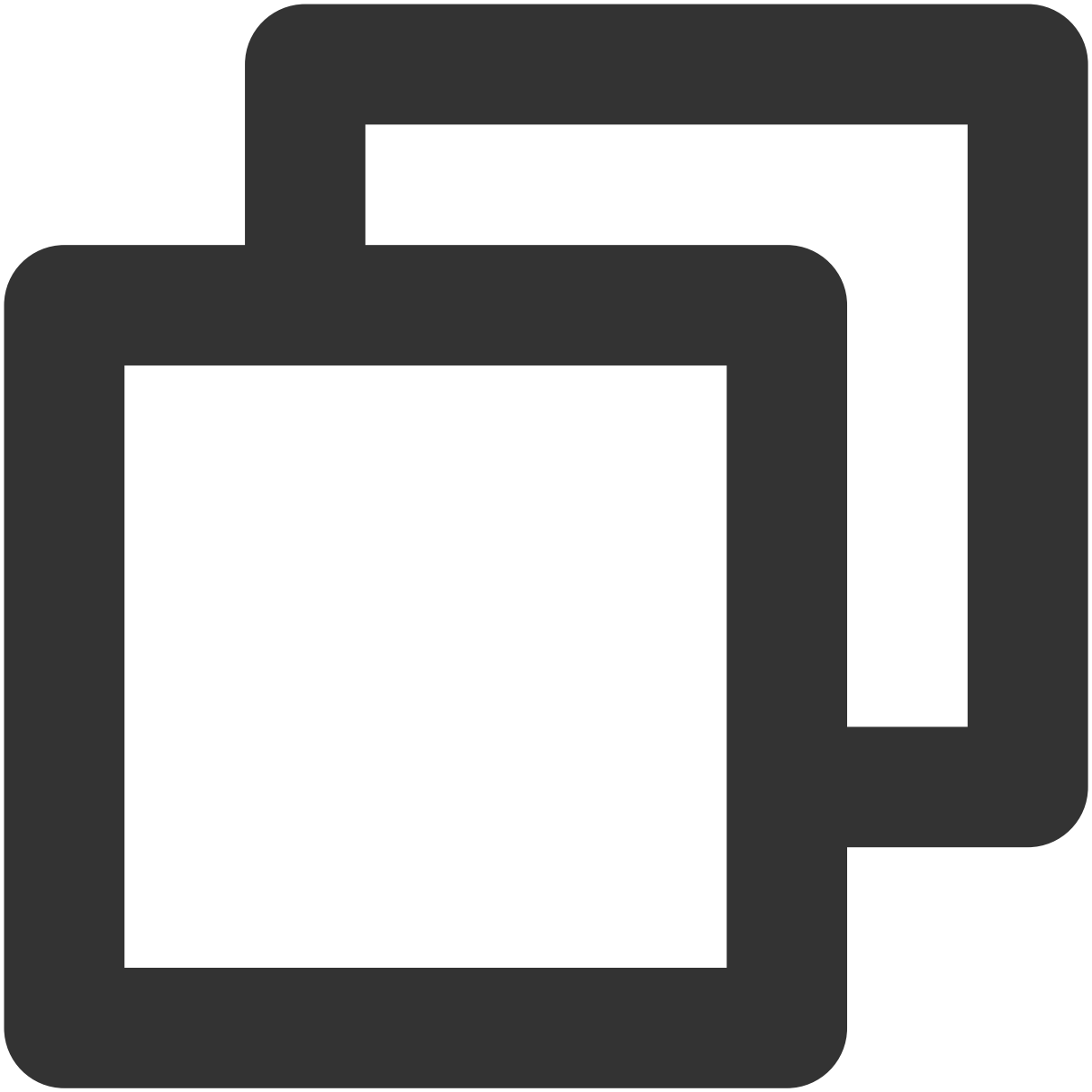
DML 语法
INSERT STATEMENT

最近更新时间：2024-08-07 17:25:46

插入一条新行记录到一个表中。如果指定了列名列表，则它们必须与查询生成的列名列表完全匹配。表中不在列名列表中的每一列都将填充一个空值。如果未指定列名列表，则查询生成的列必须与插入的表中的列完全匹配。

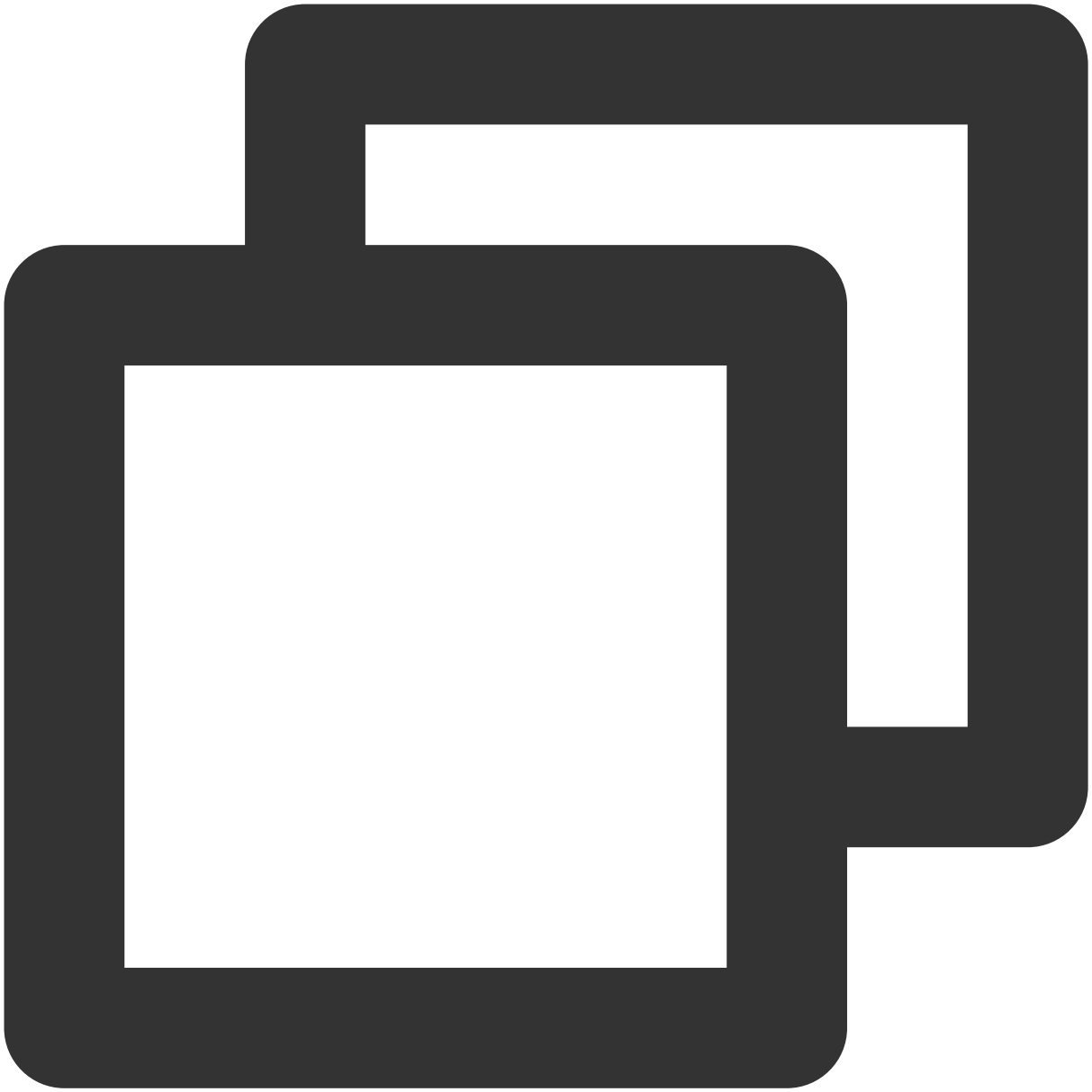
语法

Presto：



```
INSERT INTO table_name [ ( column [, ... ] ) ] query
```

Spark：



```
INSERT INTO table_identifier [ partition_spec ] [ ( column_list ) ]
  { VALUES ( { value | NULL } [ , ... ] ) [ , ( ... ) ] | query }
```

参数

`[ partition_spec ]` ：分区列和值。例如 dt='2021-06-01'。

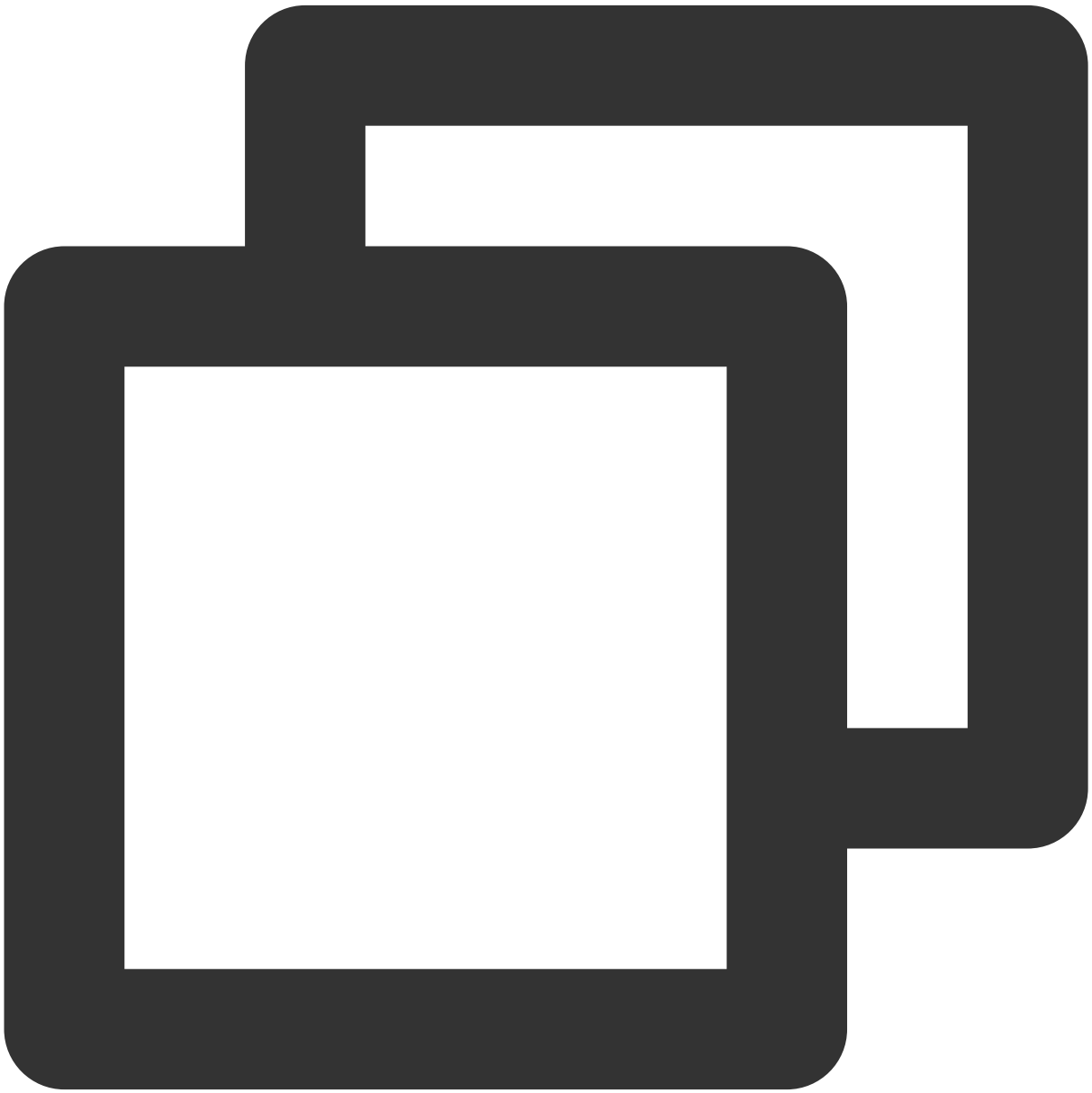
`[ ( column [, ... ] ) ]` ：列的所有。

`[table_name] | table_identifier` ：表名。

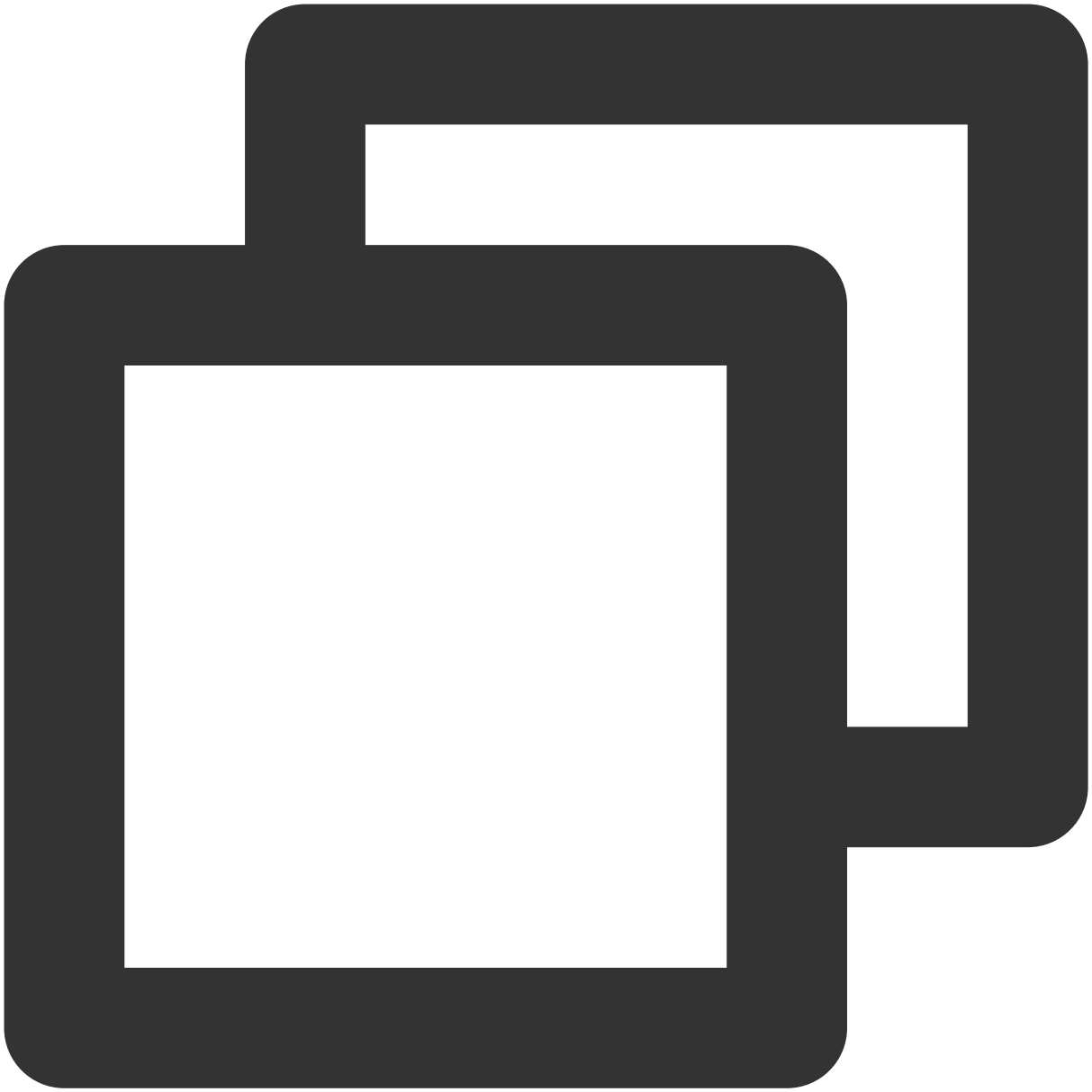
`[query]` ：一个通用 Select 查询语句。

示例

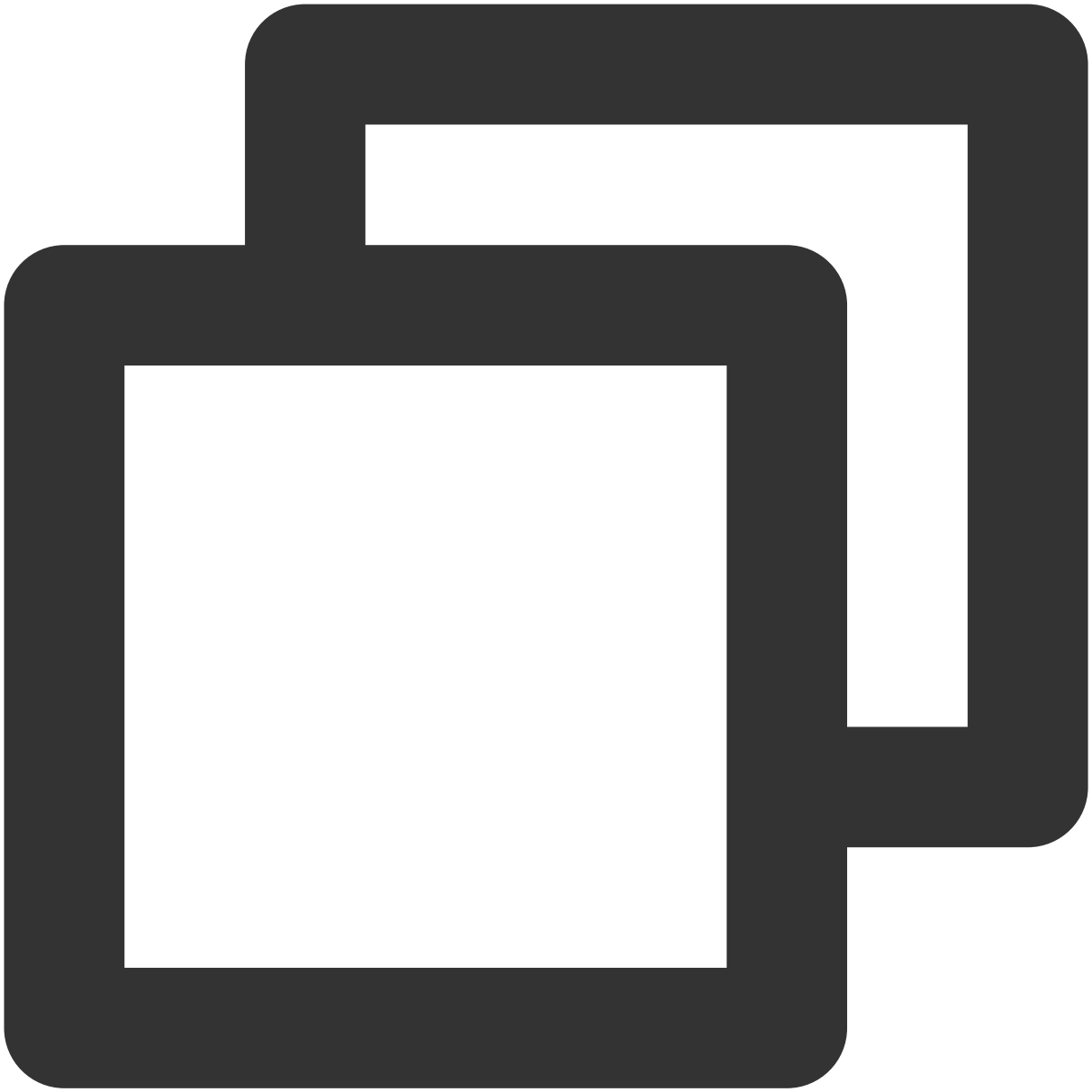
Presto 和 Spark 通用插入示例：



```
INSERT INTO orders SELECT * FROM new_orders;
```



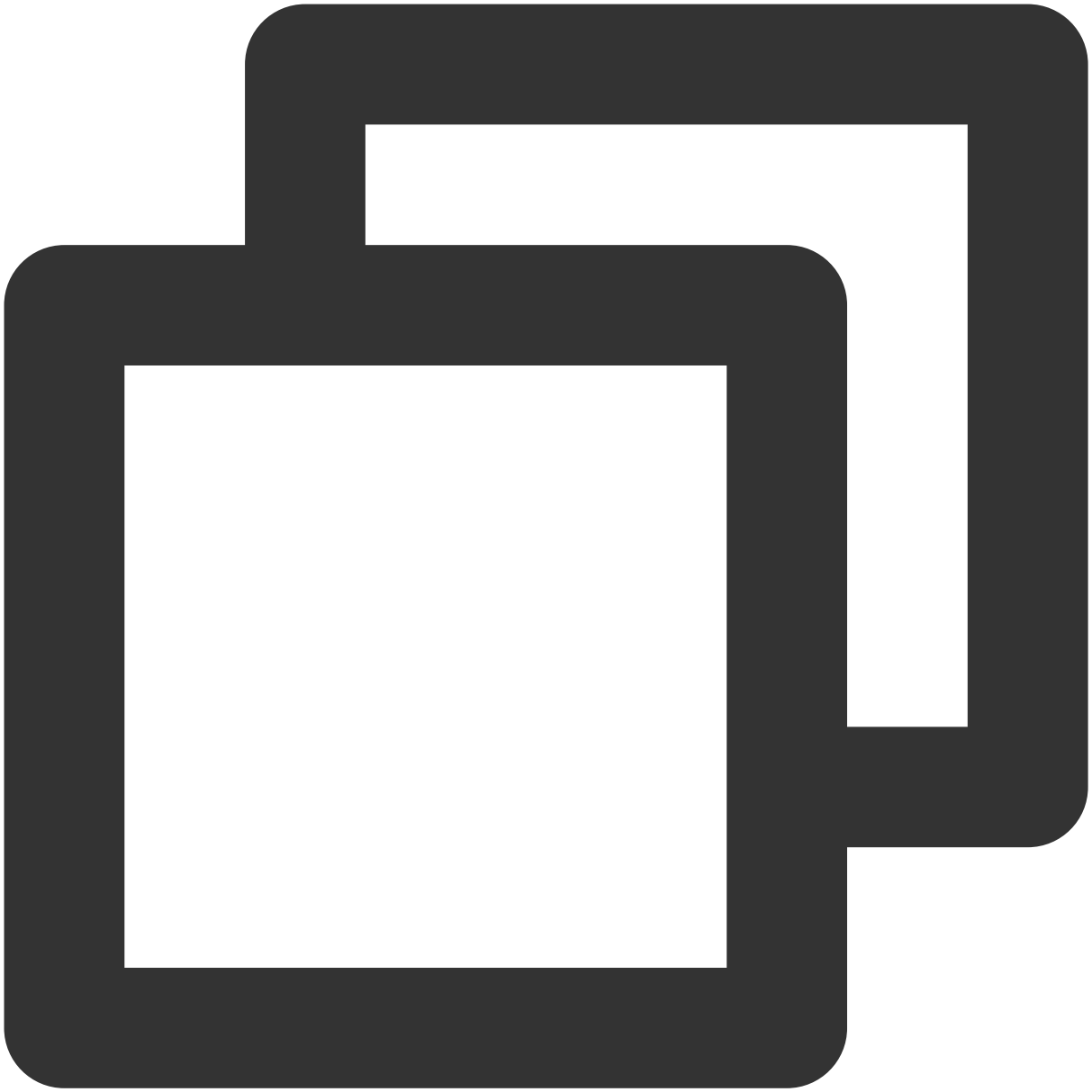
```
INSERT INTO cities VALUES (1, 'China');
```



```
INSERT INTO nation (nationkey, name, regionkey, comment)
VALUES (26, 'POLAND', 3, 'no comment');
```

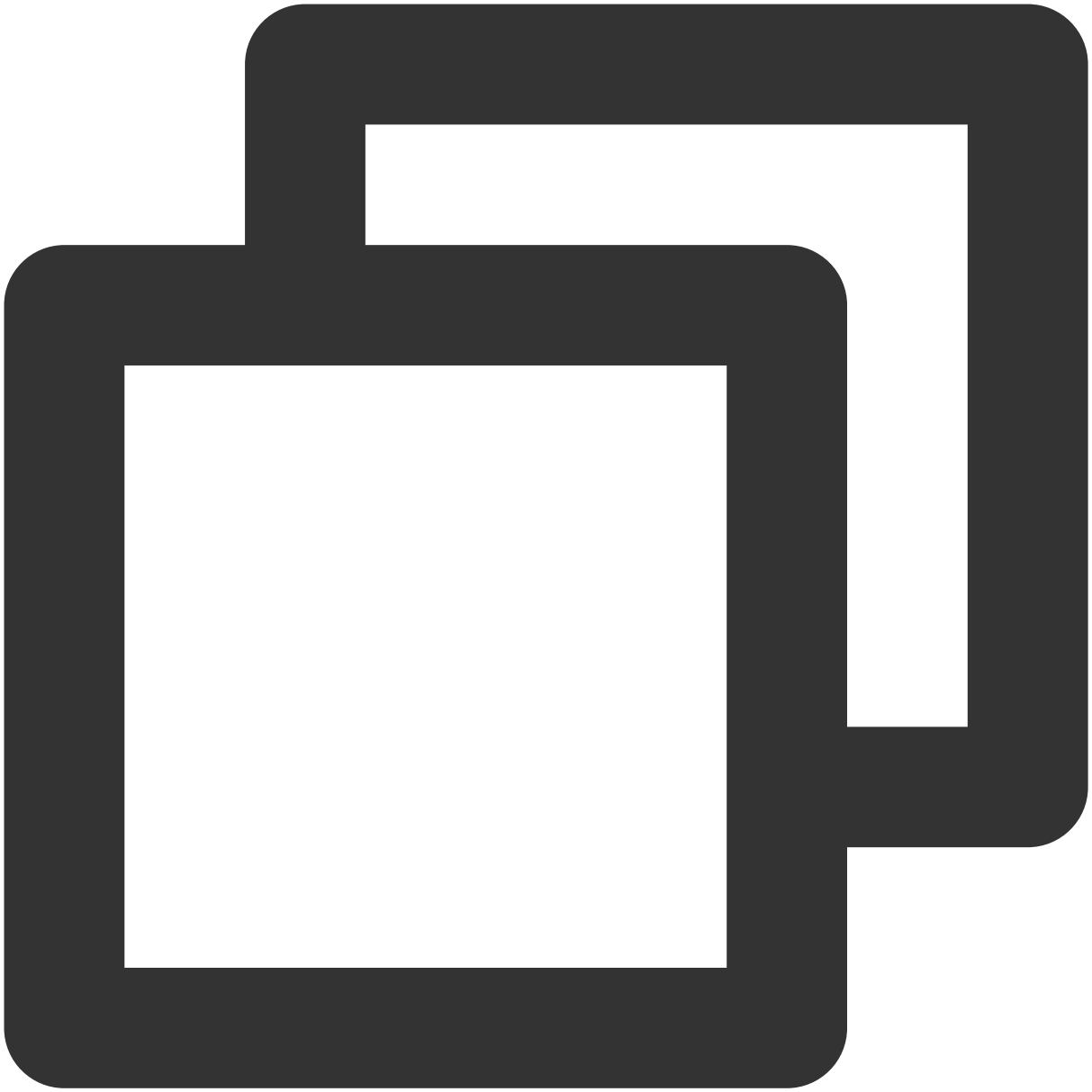
Spark 样例：

插入分区使用一个 select 查询：



```
INSERT INTO students PARTITION (student_id = 444444) SELECT name, address FROM pers
```

插入分区：



```
INSERT INTO students PARTITION (student_id = 11215017) (address, name) VALUES  ('Sh
```

限制

Presto 不支持插入分区的操作，如果需要插入分区的话可以采用 spark 引擎执行。

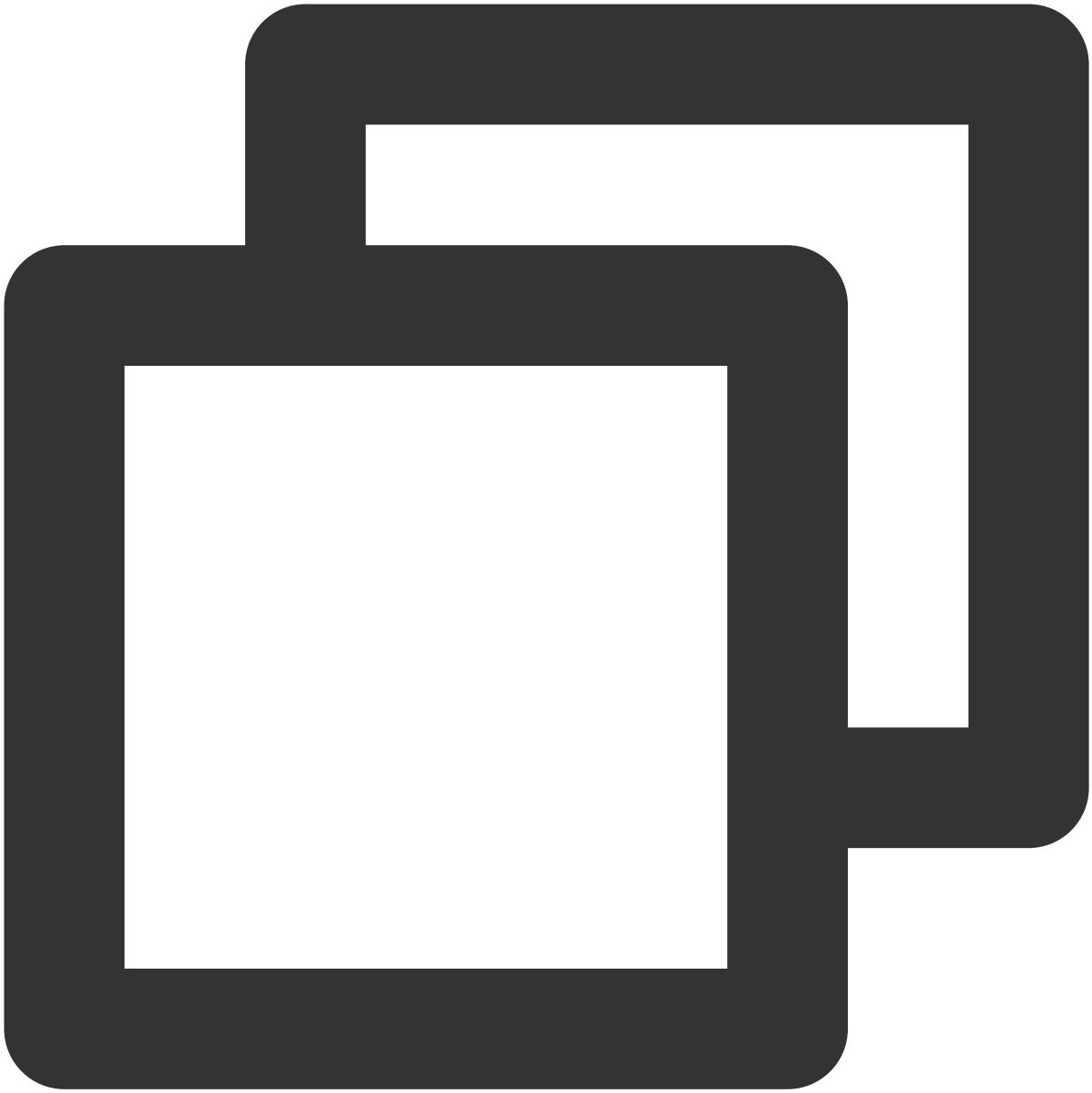
INSERT INTO

最近更新时间：2024-08-07 17:25:53

说明

支持内核：Presto、SparkSQL。
适用表范围：原生 Iceberg 表、外部表。
用途：支持将在源表上运行的 SELECT 查询结果作为新行插入到目标表中。

语法



```
[ WITH with_query [ , ... ] ]
INSERT {INTO [<TABLE>]| TABLE} table_identifier [ partition_spec ] [ ( column_list
    { VALUES ( { value | NULL } [ , ... ] ) [ , ( ... ) ] | query }
```

参数

`table_identifier` ：指定表名，支持三段式，例如： `catalog.database.table` 。
`partition_spec` ：分区列和值。例如 dt='2021-06-01'。

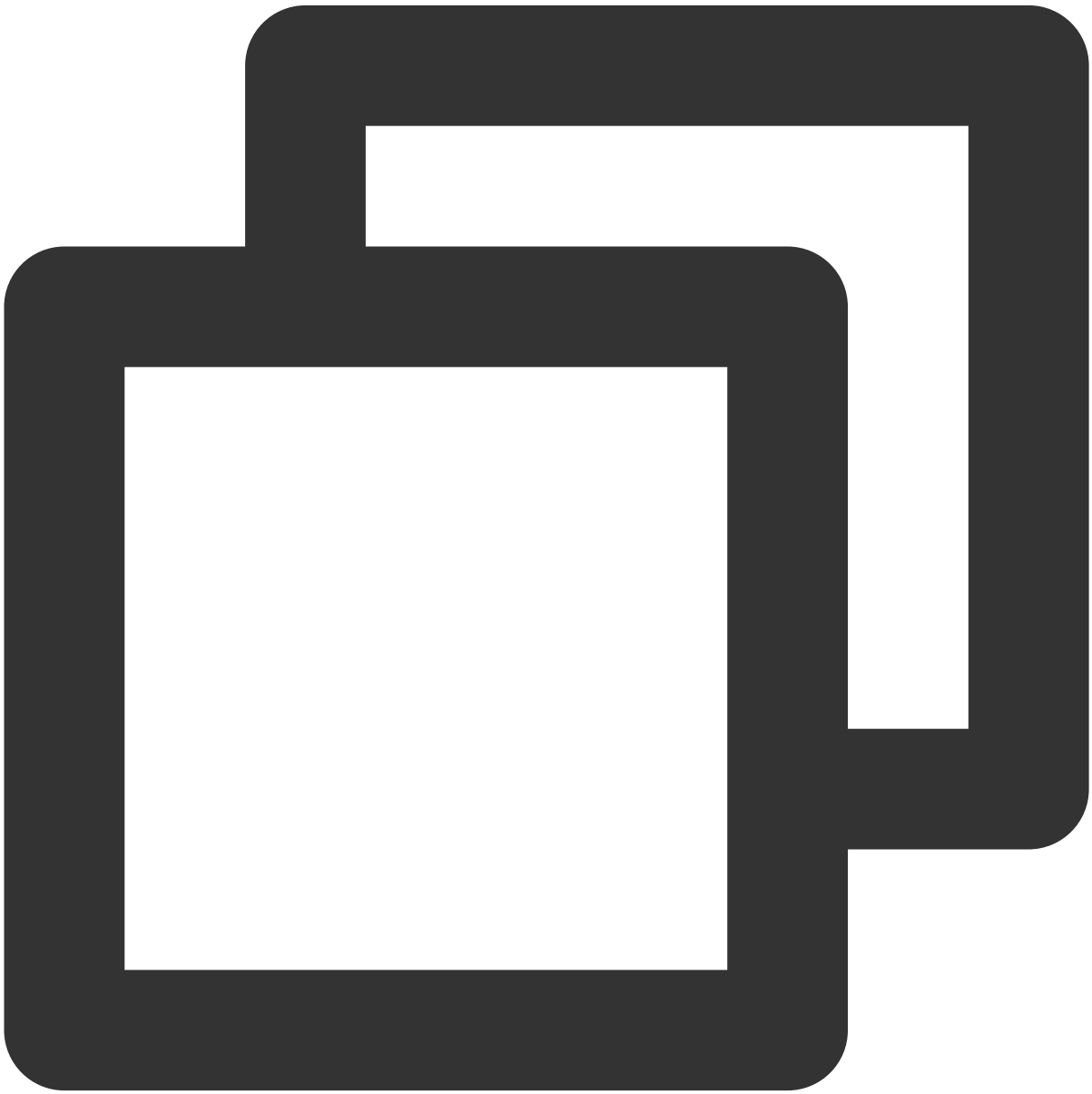
`column_list` ：列的所有。

`query` ：一个通用 Select 查询语句。

1.1 `a SELECT statement`

1.2 `a TABLE statement`

示例



```
INSERT INTO orders SELECT * FROM new_orders;
```

```
INSERT INTO cities VALUES (1, 'China');
INSERT INTO nation (nationkey, name, regionkey, comment)
VALUES (26, 'POLAND', 3, 'no comment');

-- INSERT INTO partition
INSERT INTO students PARTITION (student_id = 444444) SELECT name, address FROM pers
INSERT INTO students PARTITION (student_id = 11215017) (address, name) VALUES  ('Sh

-- Insert Using a TABLE Statement
INSERT INTO students TABLE visiting_students;

-- with
WITH `tmp1` AS ((SELECT *
FROM `catalog1`.`db1`.`tbl1`)), `tmp2` AS ((SELECT *
FROM `tbl2`))
INSERT INTO `catalog1`.`db2`.`tbl1`
(SELECT `col1`, `col2`
FROM `tmp1` `a`
INNER JOIN `tmp2` `b` ON `a`.`col1` = `b`.`col2`)


INSERT INTO `catalog1`.`db2`.`tbl1`
WITH `tmp1` AS ((SELECT *
FROM `catalog1`.`db1`.`tbl1`)), `tmp2` AS ((SELECT *
FROM `tbl2`))
(SELECT `col1`, `col2`
FROM `tmp1` `a`
INNER JOIN `tmp2` `b` ON `a`.`col1` = `b`.`col2`)
```

版权所有：腾讯云计算（北京）有限责任公司

INSERT OVERWRITE

最近更新时间：2024-08-07 17:26:00

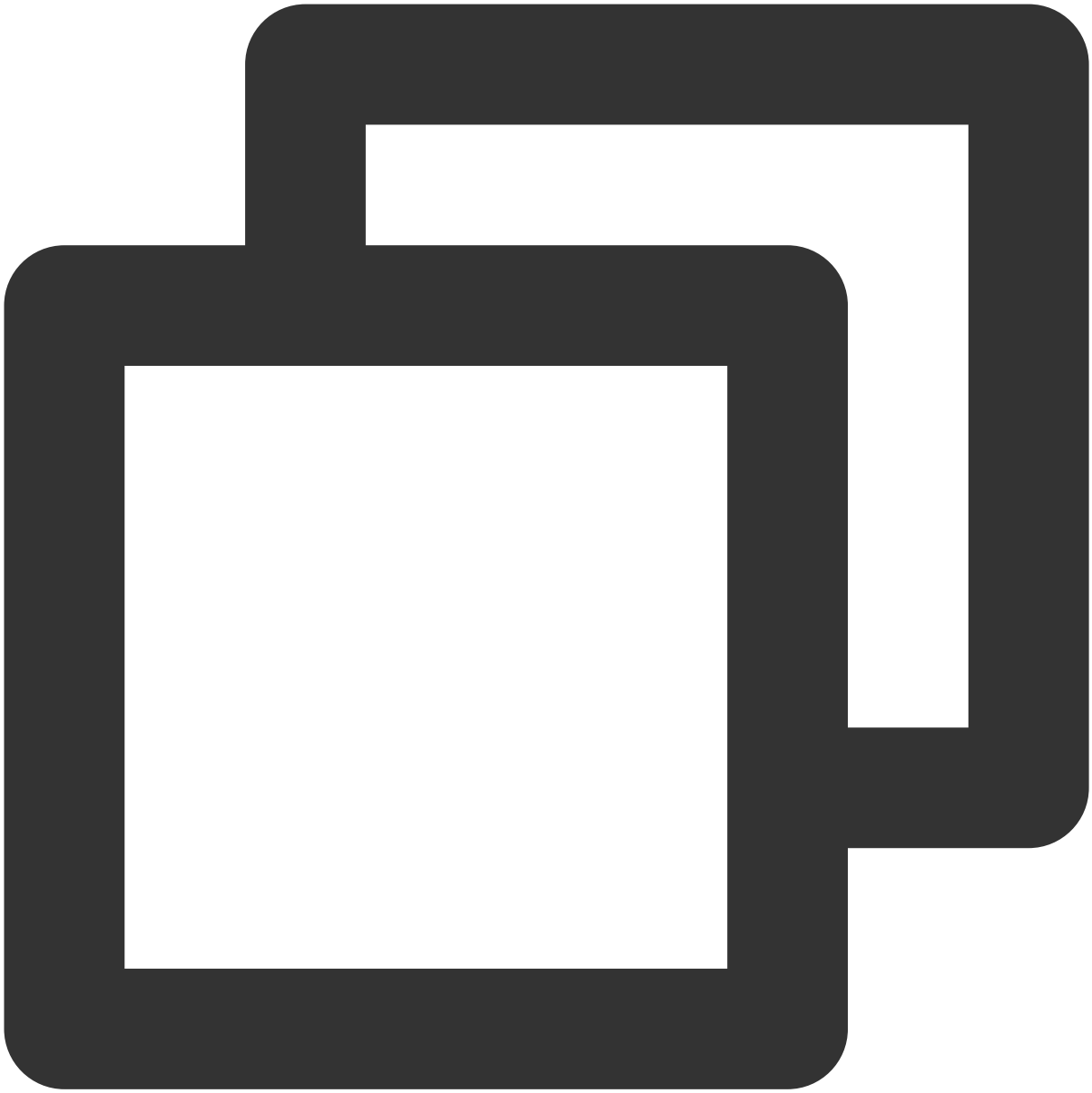
说明

支持内核：Presto、SparkSQL。
适用表范围：原生 Iceberg 表、外部表。
用途：行级数据插入操作。

说明

Presto 仅支持在 Hive 数据源的分区表上执行 insert overwrite，非分区表以及 Iceberg 数据源的表暂时不支持这个用法。

语法



```
INSERT OVERWRITE table_identifier [ partition_spec ] [ ( column_list ) ]
    { VALUES ( { value | NULL } [ , ... ] ) [ , ( ... ) ] | query
```

参数

`table_identifier` ：指定表名，支持三段式，例如：catalog.database.table

`partition_spec` ：分区列和值。例如 dt='2021-06-01'。

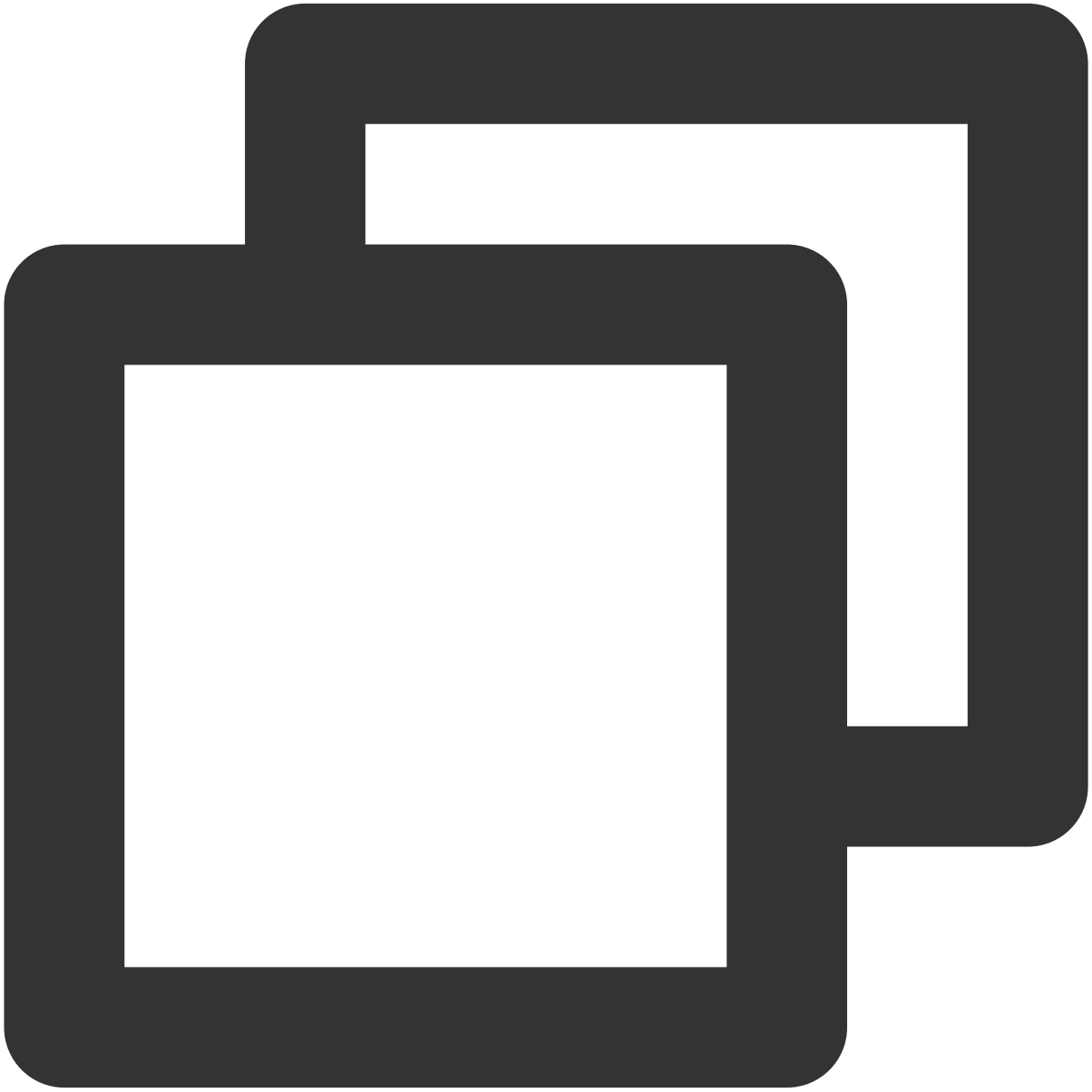
`column_list` ：列的所有。

`query` ：一个通用 Select 查询语句。

1.1 a SELECT statement

1.2 a TABLE statement

示例



```
-- Insert Using a VALUES Clause
```

```
INSERT OVERWRITE students VALUES
    ('Ashua Hill', '456 Erica Ct, Cupertino', 111111),
    ('Brian Reed', '723 Kern Ave, Palo Alto', 222222);

-- Insert Using a SELECT Statement
INSERT OVERWRITE students PARTITION (student_id = 222222)
    SELECT name, address FROM persons WHERE name = "Dora Williams"

-- Insert Using a TABLE Statement
INSERT OVERWRITE students TABLE visiting_students

-- Insert with a column list
INSERT OVERWRITE students (address, name, student_id) VALUES
    ('Hangzhou, China', 'Kent Yao', 11215016)

-- Insert with both a partition spec and a column list
INSERT OVERWRITE students PARTITION (student_id = 11215016) (address, name) VALUES
    ('Hangzhou, China', 'Kent Yao Jr.')
```

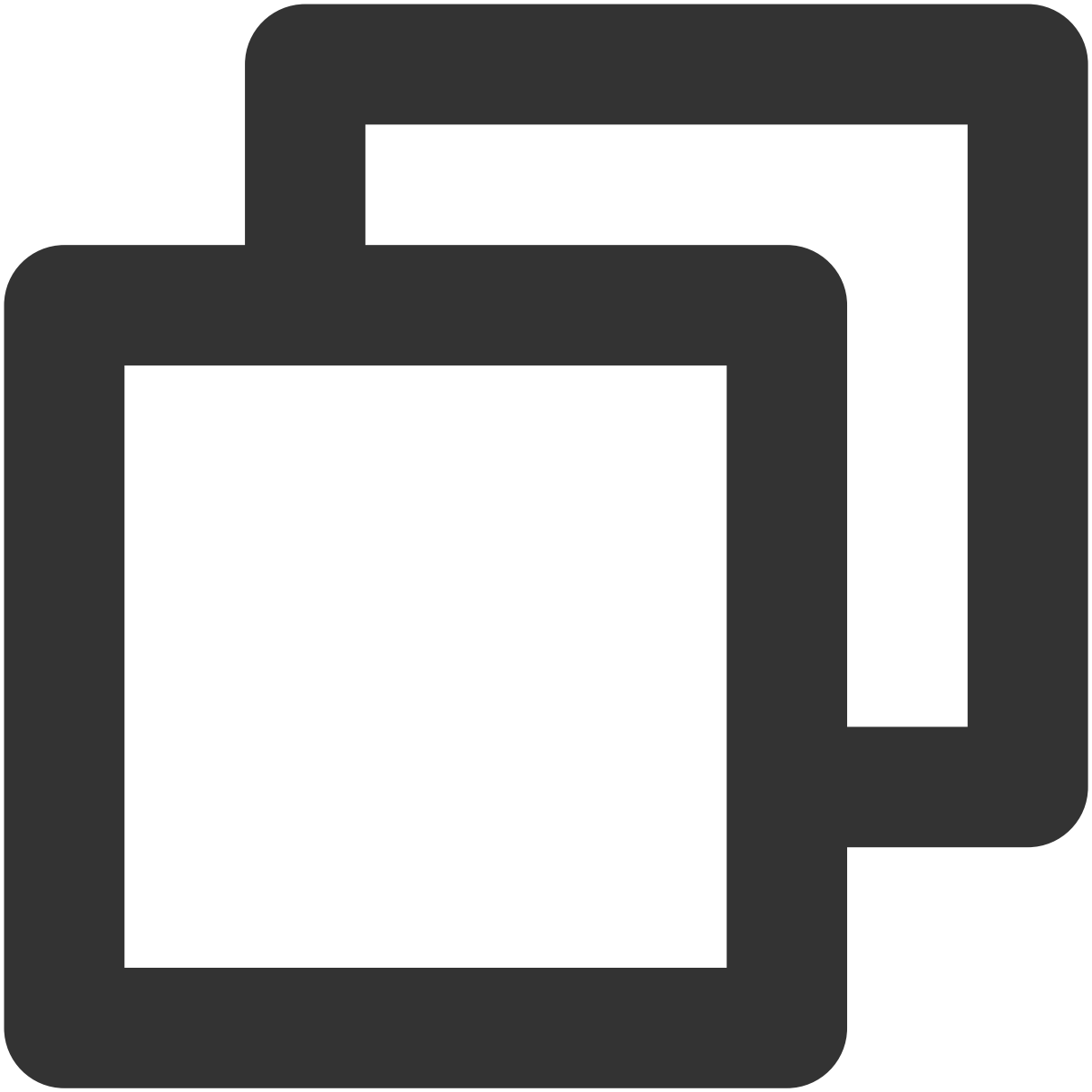
MERGE INTO

最近更新时间：2024-08-07 17:26:21

说明

支持内核：SparkSQL。
适用表范围：原生 Iceberg 表、外部表。
用途：行级数据更新操作，可用于替换 INSERT OVERWRITE 操作。

语法



```
MERGE INTO tablePrimary1 [ [ AS ] alias ]
USING tablePrimary2
ON booleanExpression
[ WHEN MATCHED (AND matchedCond=booleanExpression)? THEN DELETE ]*
[ WHEN MATCHED (AND matchedCond=booleanExpression)? THEN UPDATE SET assign [, assig
[ WHEN NOT MATCHED (AND notMatchedCond=booleanExpression)? THEN INSERT VALUES '(' v
```

参数

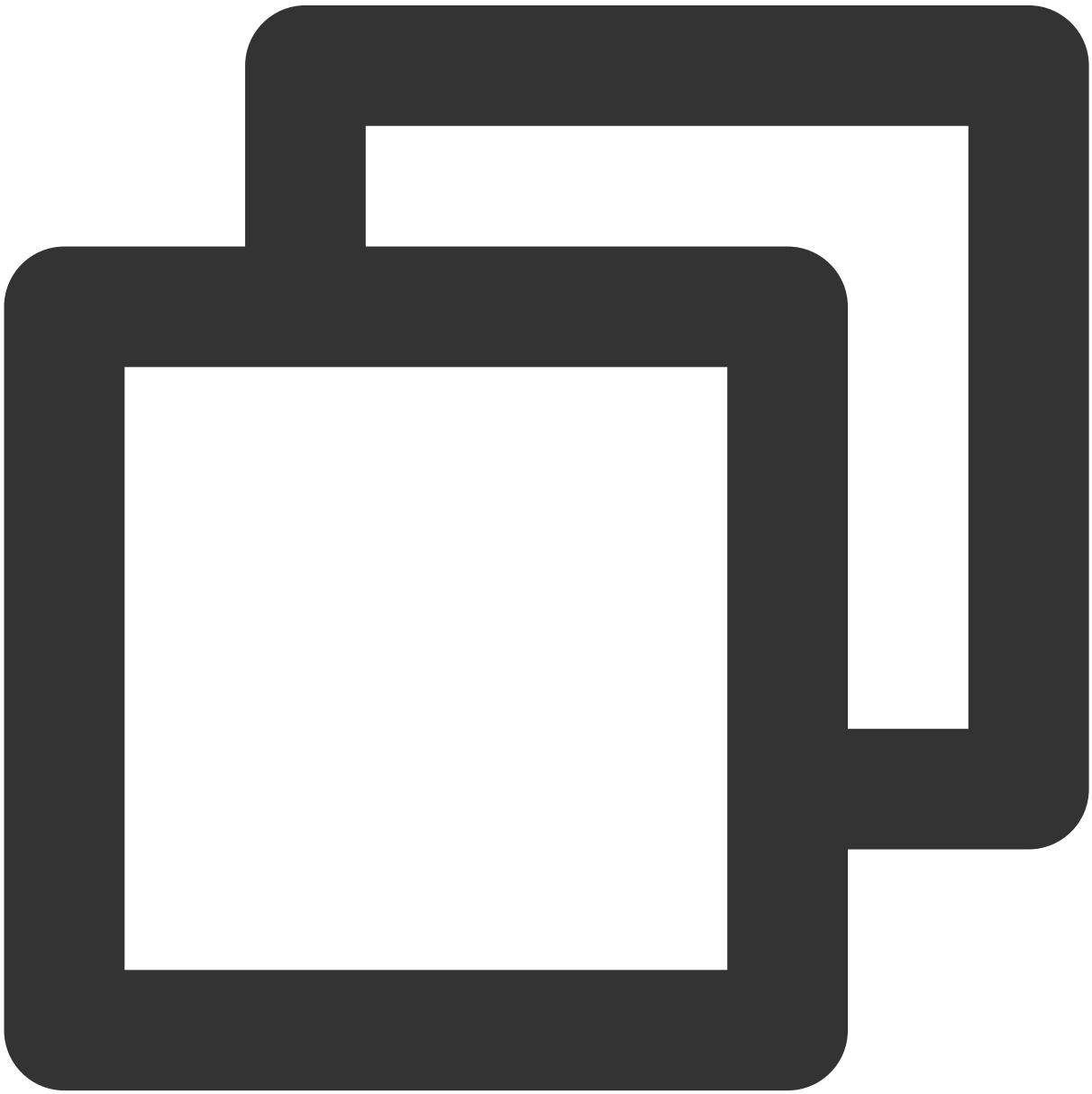
`tablePrimary1` ：指定表名，支持三段式，例如：catalog.database.table。

`alias` ：别名。

`tablePrimary2` ：可以是表名，也可以是子查询。

`booleanExpression` ：布尔表达式。

示例



```
MERGE INTO catalog1.db2.tbl1 t
```

```
USING catalog1.db1.tbl1
ON t.col1 = tbl1.col1
WHEN MATCHED AND t.col1 = 14 THEN DELETE

MERGE INTO catalog1.db2.tbl1 t
USING (SELECT col1 FROM catalog1.db1.tbl1) s
ON t.col1 = s.col1
WHEN MATCHED AND t.col1 = 14 THEN UPDATE SET col1 = 2


MERGE INTO catalog1.db2.tbl1 t
USING (SELECT col1 FROM catalog1.db1.tbl1) s
ON t.col1 = s.col1
WHEN MATCHED AND t.col1 = 12 THEN UPDATE SET col1 = 0
WHEN MATCHED AND t.col1 = 13 THEN UPDATE SET col1 = 1
WHEN MATCHED AND t.col1 = 14 THEN UPDATE SET col1 = 2
WHEN MATCHED AND t.col1 = 15 or s.col1 = 16 THEN UPDATE SET col1 = t.col1 + 1
WHEN MATCHED AND t.col1 not in (12, 13, 14, 15) THEN UPDATE SET col1 = 4
WHEN NOT MATCHED AND t.col1 = 12 THEN INSERT (col1) VALUES (s.col1)
WHEN NOT MATCHED AND t.col1 = 13 THEN INSERT (col1) VALUES (s.col1 + 1)
WHEN NOT MATCHED AND t.col1 = 14 THEN INSERT (col1) VALUES (s.col1 + 2)


MERGE INTO catalog1.db2.tbl1 t
USING (SELECT col1, col2 FROM catalog1.db1.tbl1) s
ON t.col1 = s.col1
WHEN MATCHED AND t.col1 = fun1(s.col2) THEN DELETE
WHEN MATCHED AND t.col1 = db2.fun1(s.col2) THEN DELETE
WHEN MATCHED AND (t.col1 = length(s.col2) or t.col1 = catalog2.db2.fun3(s.col2)) TH
WHEN NOT MATCHED AND t.col1 = 12 THEN INSERT (col1) VALUES (db2.fun2(s.col2))
```

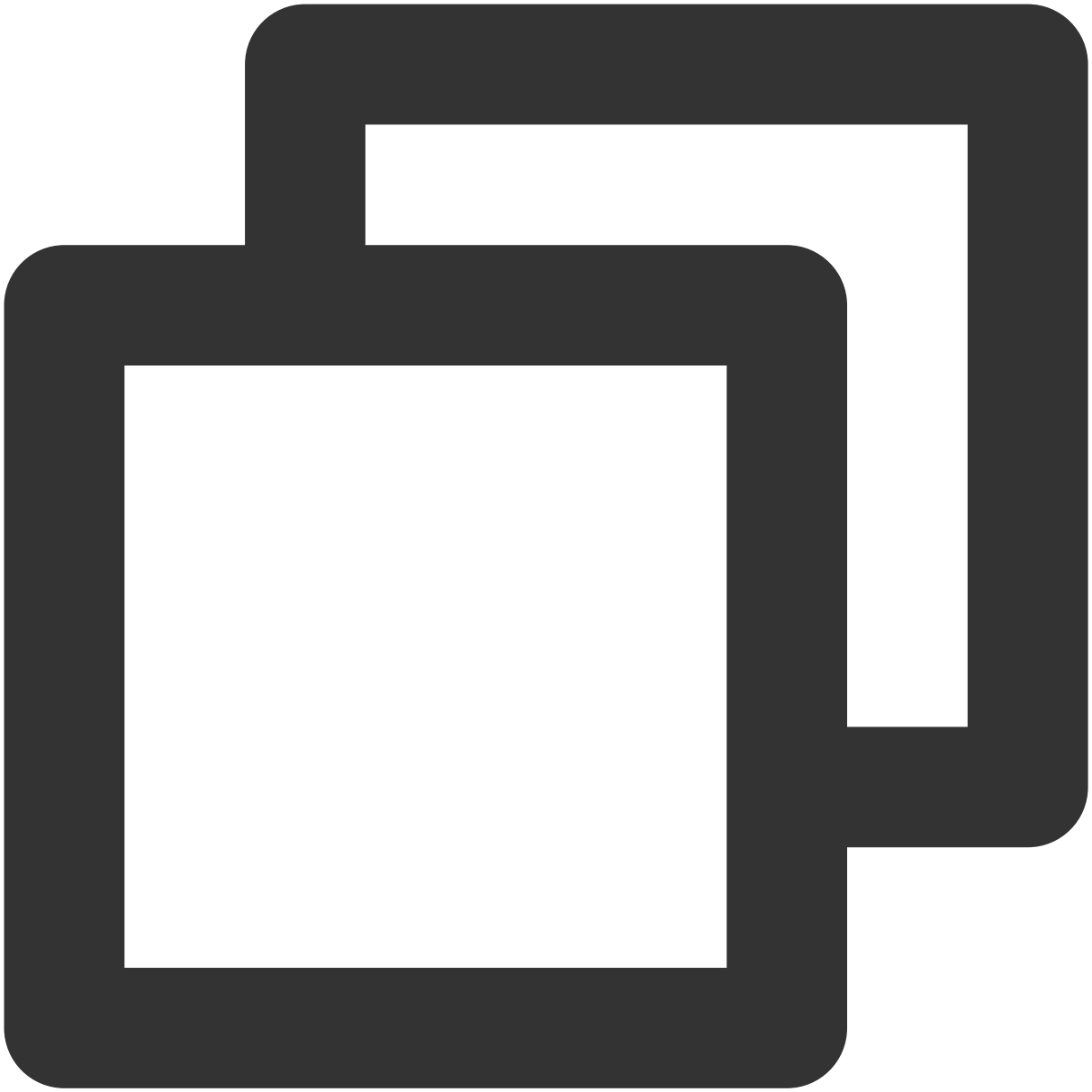
UPDATE

最近更新时间：2024-08-07 17:26:39

说明

支持内核：SparkSQL。
适用表范围：原生 Iceberg 表。
用途：更新数据表中的指定行。

语法



```
UPDATE tablePrimary
SET assign [, assign ]*
[ WHERE booleanExpression ]
```

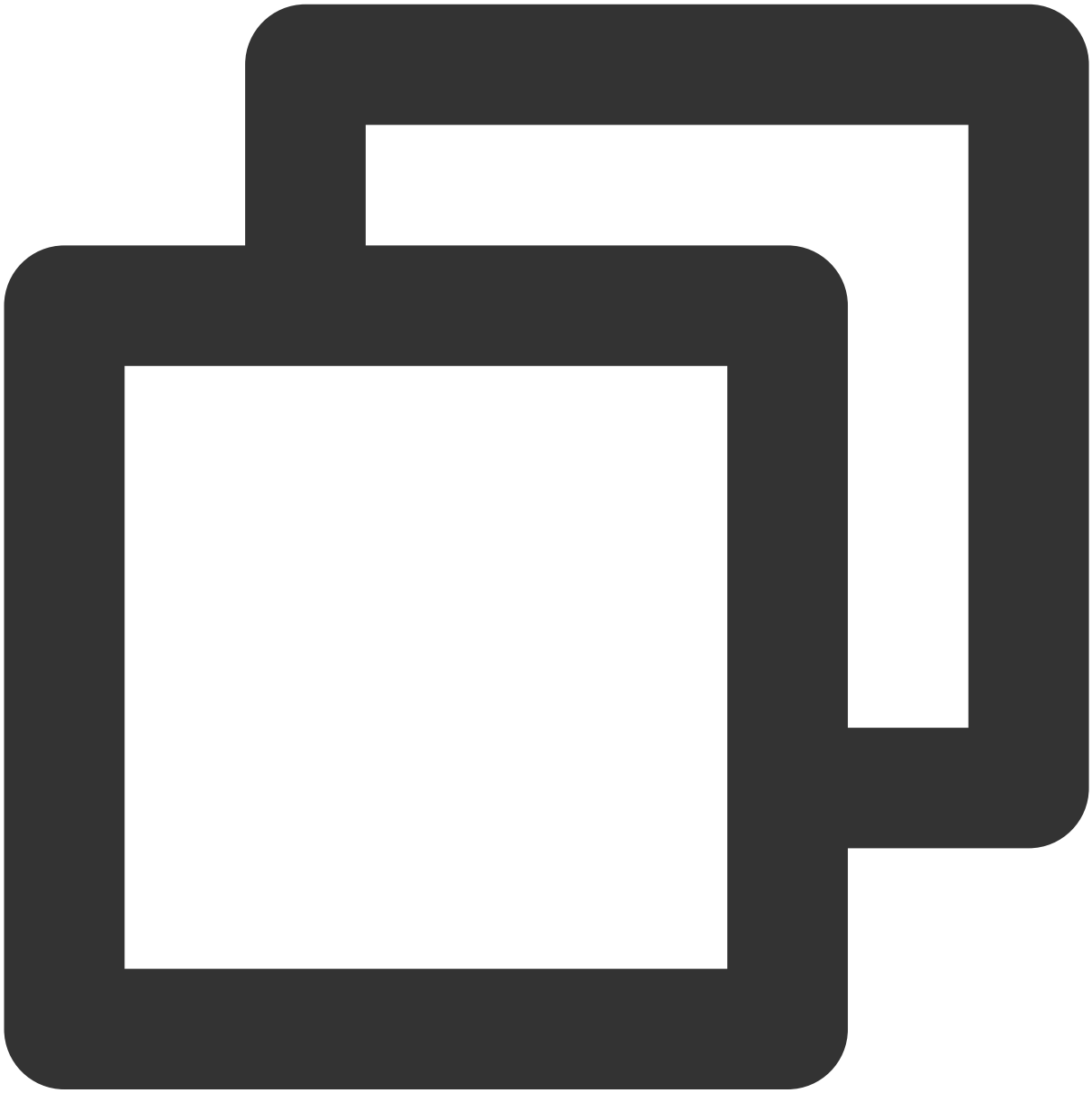
参数

`tablePrimary` ：指定表名，支持三段式，例如：catalog.database.table。

`assign` ：需要修改的式子，例如：col1 = 'new_data'。

`booleanExpression` ：布尔表达式。

示例



```
UPDATE prod.db.table
SET c1 = 'update_c1', c2 = 'update_c2'
WHERE ts >= '2020-05-01 00:00:00' and ts < '2020-06-01 00:00:00'

UPDATE prod.db.all_events
```

```
SET session_time = 0, ignored = true
WHERE session_time < (SELECT min(session_time) FROM prod.db.good_events)

UPDATE prod.db.orders AS t1
SET order_status = 'returned'
WHERE EXISTS (SELECT oid FROM prod.db.returned_orders WHERE t1.oid = oid)
```

版权所有：腾讯云计算（北京）有限责任公司

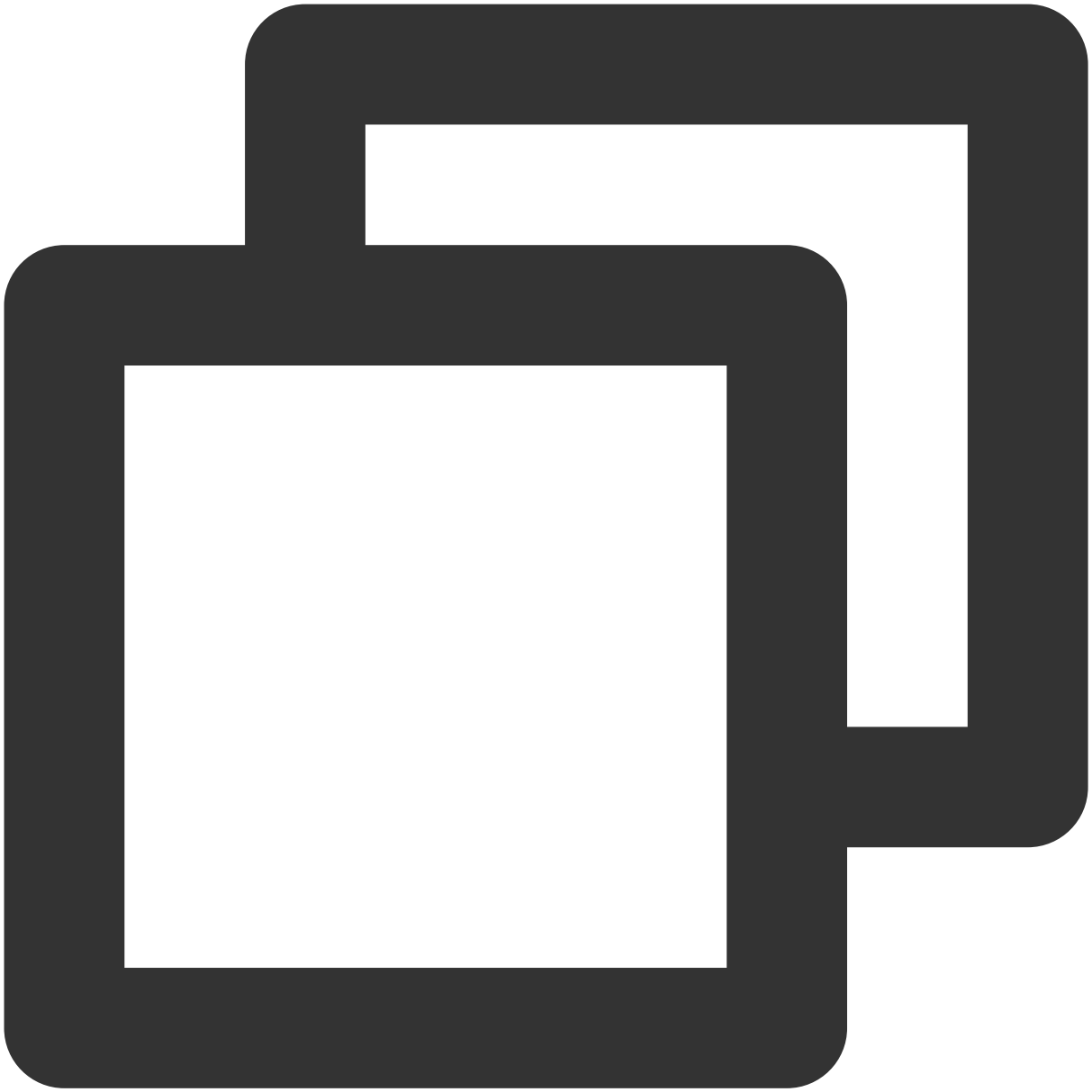
DELETE STATEMENT

最近更新时间：2024-08-07 17:26:52

说明

支持内核：SparkSQL。
适用表范围：原生 Iceberg 表。
用途：删除数据表中的指定行。

语法

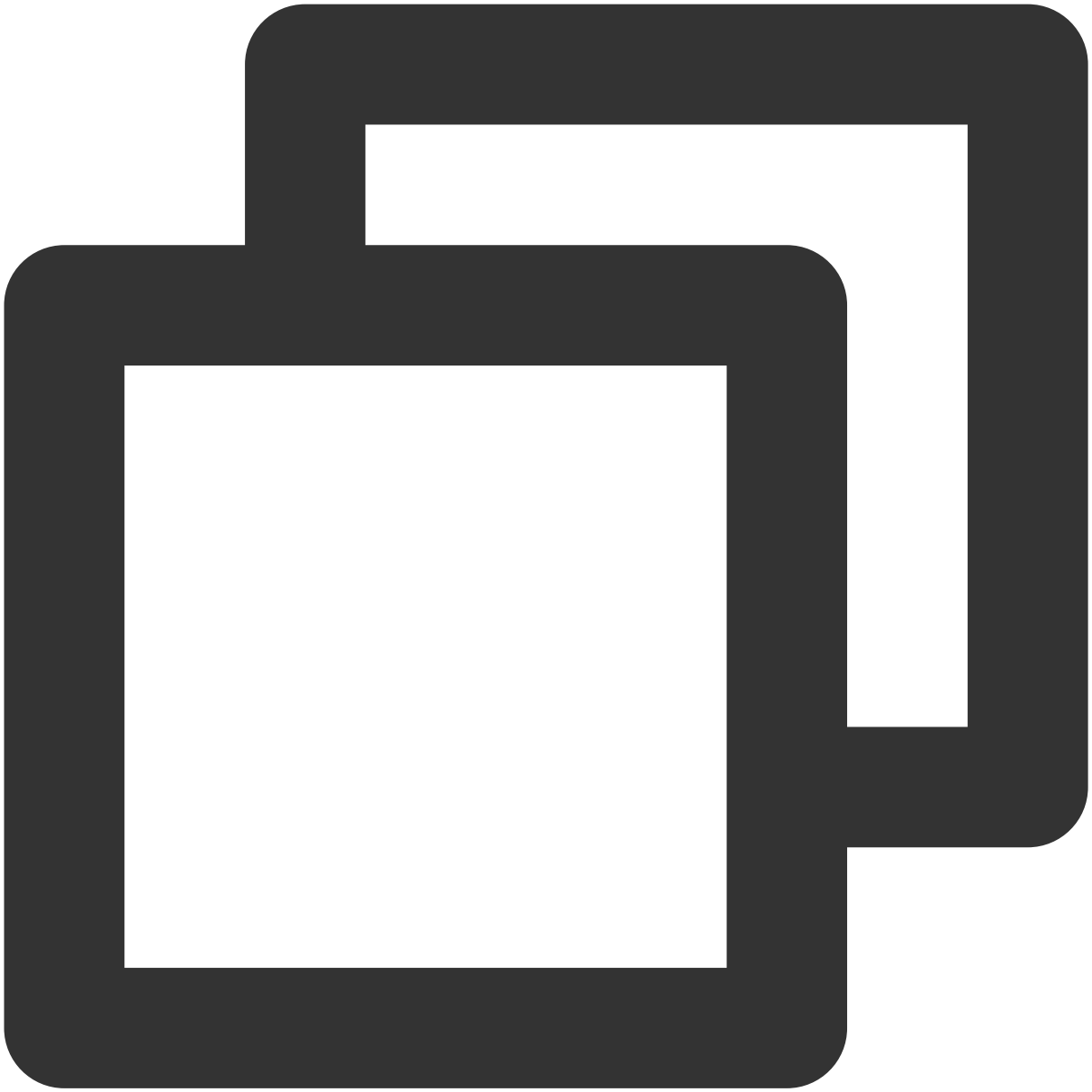


```
DELETE FROM table_name [ [ AS ] alias ]
[ WHERE booleanExpression ]
```

参数说明

`table_identifier` ：指定表名，支持三段式，例如：catalog.database.table

示例



```
DELETE FROM lineitem WHERE shipmode = 'AIR';

DELETE FROM lineitem
WHERE orderkey IN (SELECT orderkey FROM orders WHERE priority = 'LOW');

DELETE FROM orders;
```

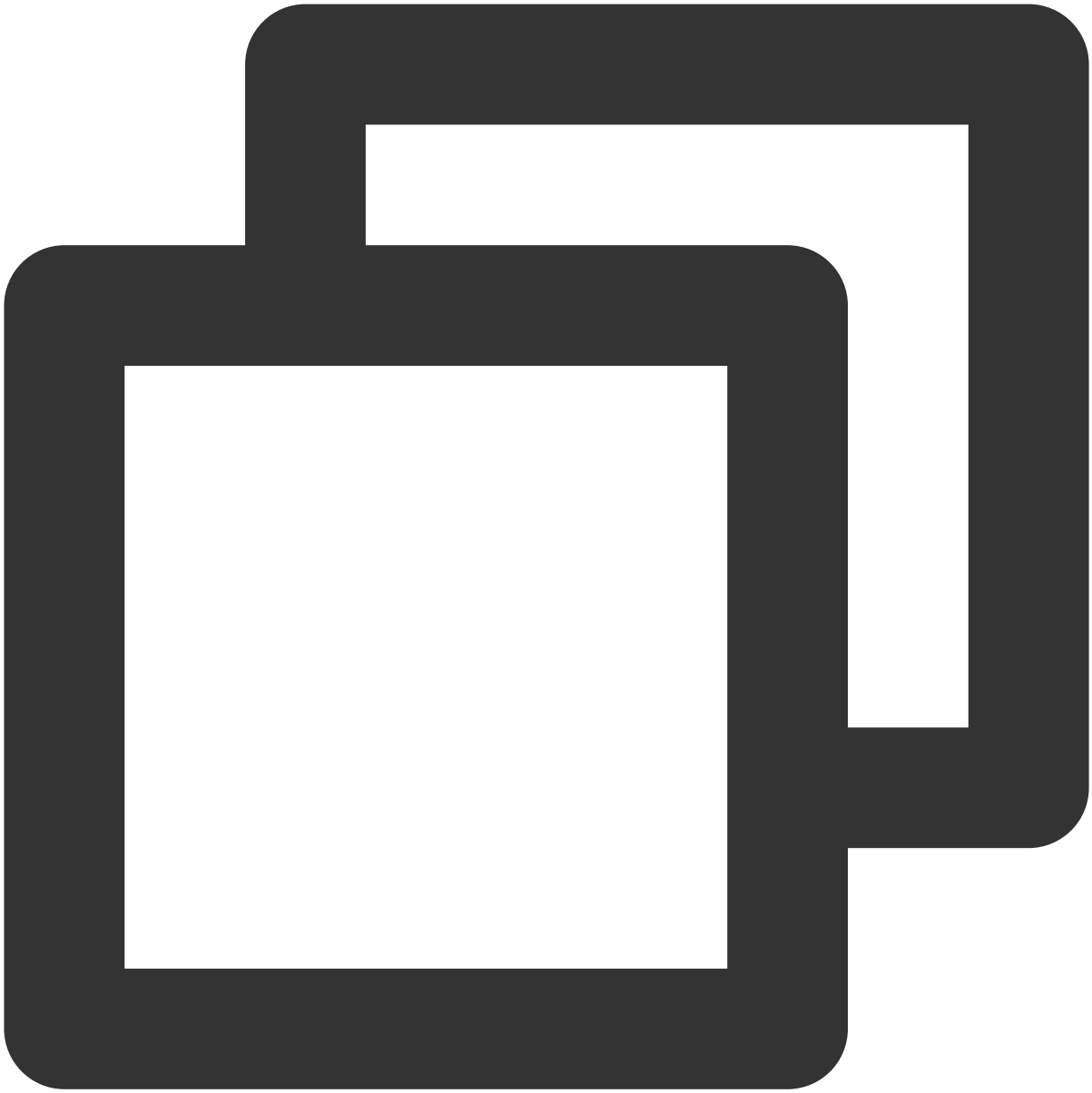
TABLE METADATA

最近更新时间：2024-08-07 17:27:39

说明

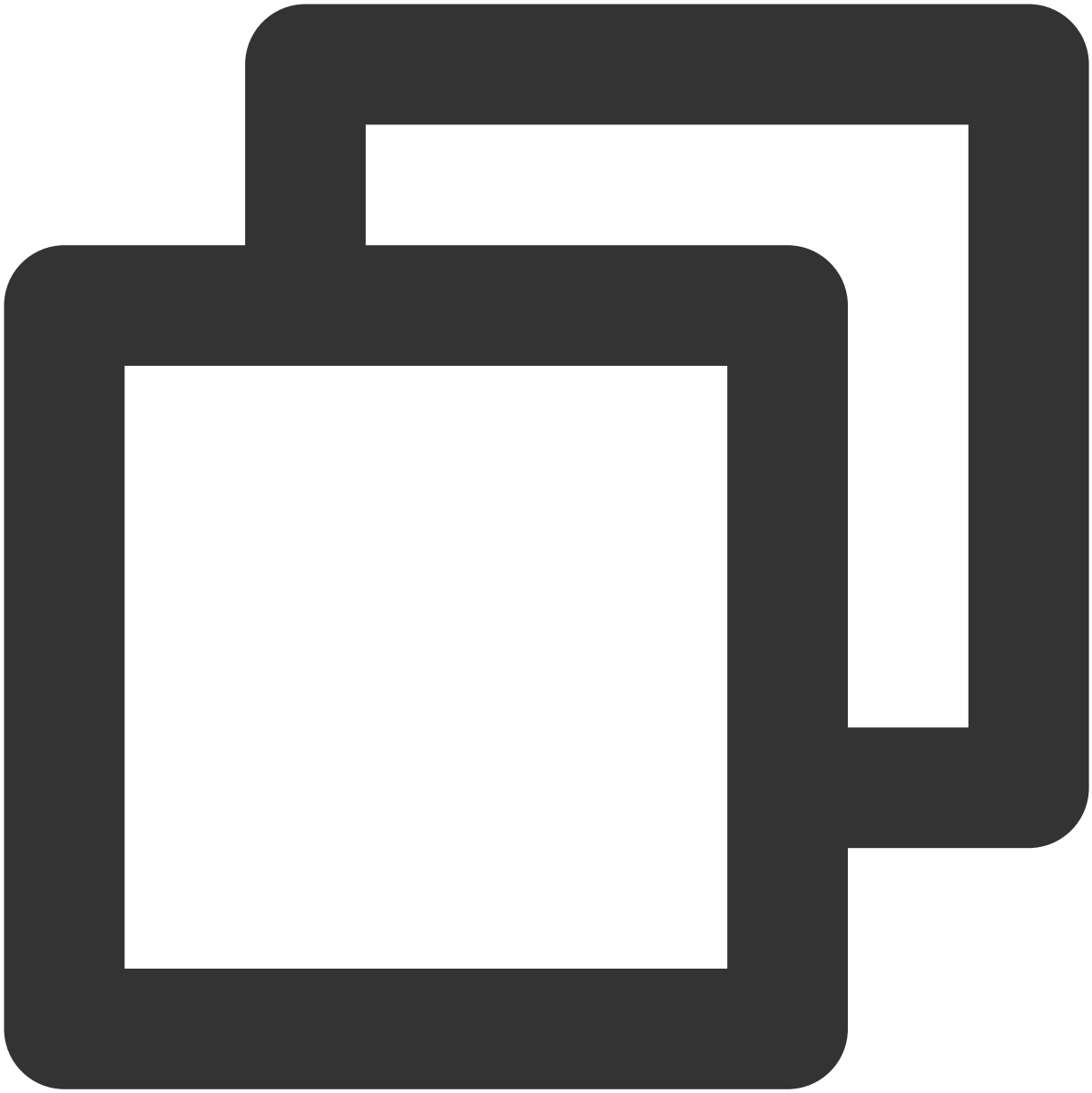
支持内核：SparkSQL。
适用表类型：原生 Iceberg 表。
用途：支持四段式的 Iceberg 表元数据查询，包括：history、snapshots、files、manifests。

语法



```
SELECT select_expr (, select_expr)*
FROM `Catalog`.`db`.`tableName${history|snapshots|files|manifests|partitions|all_da
[WHERE where_condition]
[LIMIT [offset,] rows]
```

示例



```
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$history` ORDER BY snapshot_id
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$snapshots` ORDER BY snapshot_i
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$files` ORDER BY file_size_in_b
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$manifests` ORDER BY length LIM
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$partitions` LIMIT 10;
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$all_data_files` LIMIT 10;
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$all_manifests` LIMIT 10;
```

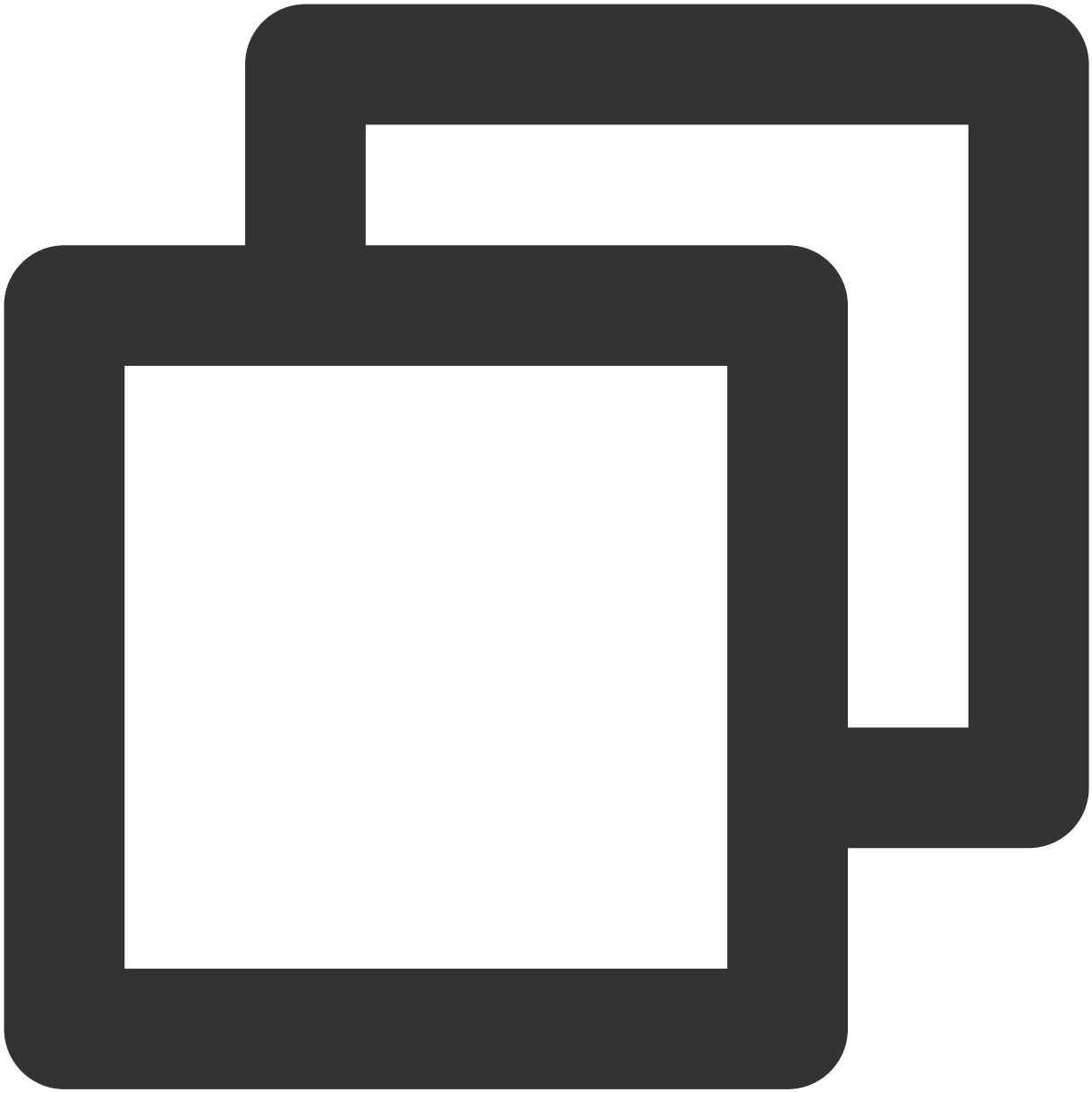
版权所有：腾讯云计算（北京）有限责任公司

DQL 语法
SELECT STATEMENT

最近更新时间：2024-08-07 17:27:58

`SELECT` 语句：从零个或多个表中检索数据行。

语法



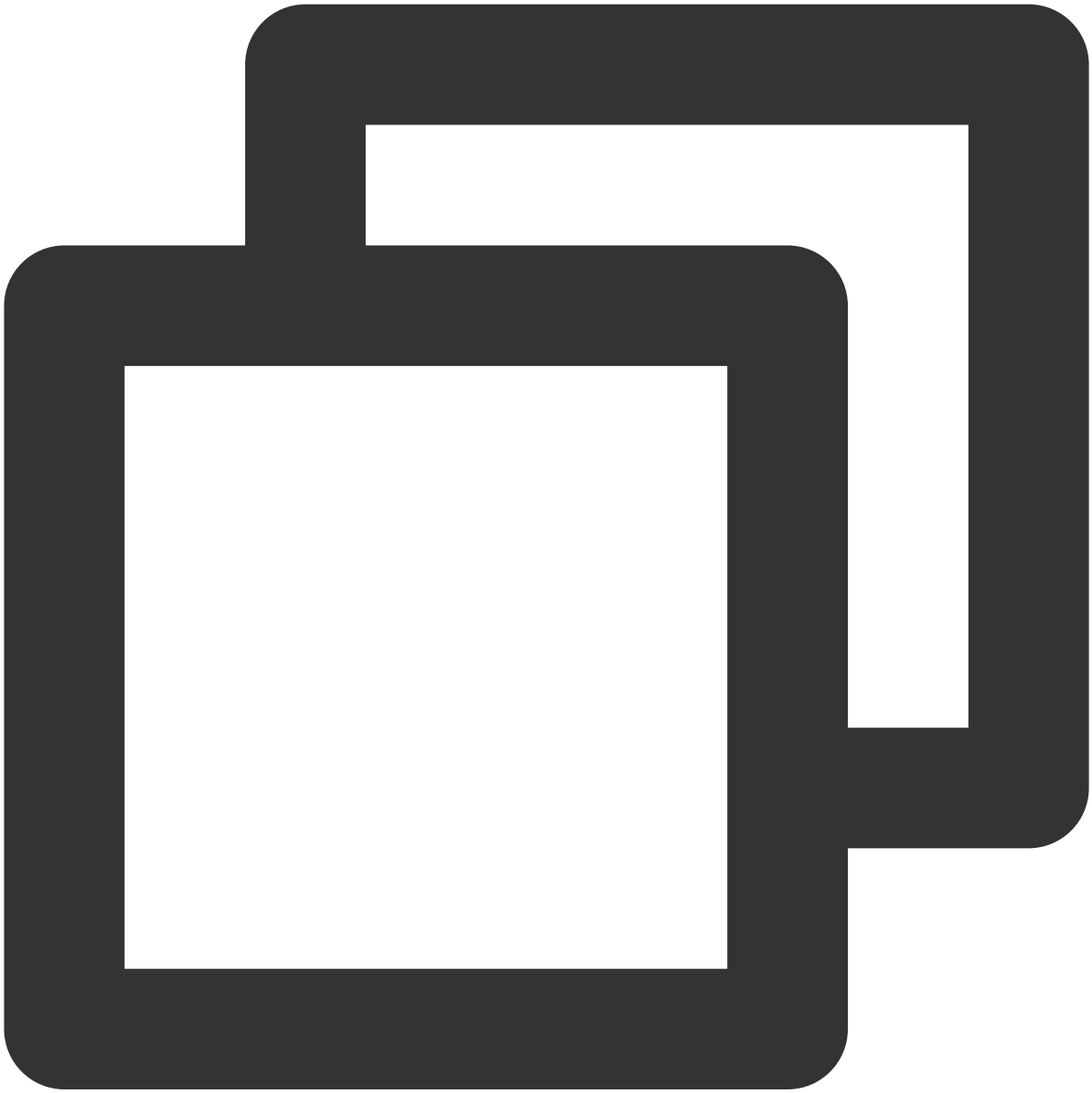
```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition ]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]
[ LIMIT [ count | ALL ] ]
```

版权所有：腾讯云计算（北京）有限责任公司

参数

[WITH with_query [, ...]]

可使用 WITH 来展平嵌套查询或简化子查询。 `with_query` 语法如下：



```
subquery_table_name [ ( column_name [, ...] ) ] AS (subquery)
```

subquery_table_name 是临时表的唯一名称，该临时表用于定义 WITH 子句子查询的结果。每个 subquery 都必须具有一个可在 FROM 子句中引用的表名称。

column_name [, ...] 是可选的输出列名称列表。列名称数目必须等于或小于 subquery 定义的列数。

subquery 是任意查询语句。

[ALL | DISTINCT] select_expr

ALL 和 DISTINCT 选项指定是否应返回重复的行。如果没有给出这些选项，则默认值为 ALL（返回所有匹配的行）。DISTINCT 指定从结果集中删除重复行。

FROM from_item [, ...]

from_item 可以是视图、表、子查询，如果多表 join，支持的 join 类型如下：

`[ INNER ] JOIN`

`LEFT [ OUTER ] JOIN`

`RIGHT [ OUTER ] JOIN`

`FULL [ OUTER ] JOIN`

`CROSS JOIN`

`ON join_condition` ，如果使用 `join_condition` ，您可以为多个表中的联接键指定列名称；如果使用 `join_column` ，则要求 join_column 在两个表中都存在。

[WHERE condition]

根据您指定的 condition 筛选结果，返回满足条件的结果集。

[GROUP BY [ALL | DISTINCT] grouping_expressions [, ...]]

GROUP BY 表达式可以按照指定的列名对输出进行分组。

[HAVING condition]

与聚合函数和 GROUP BY 子句一起使用。控制哪些组处于选中状态，从而消除不满足 condition 的组。此筛选在计算组和聚合之后发生。

[{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] union_query]

`UNION` 、 `INTERSECT` 和 `EXCEPT` 将多个结果组合在一起， `UNION` 将第一个查询生成的行与第二个查询生成的行组合在一起。为了消除重复，UNION 构建一个散列表，这会消耗内存。为了更好的性能，建议使用 UNION ALL。

`INTERSECT` 仅返回第一个和第二个查询的结果中存在的行。

`EXCEPT` 返回第一个查询结果中的行，不包括第二个查询找到的行。

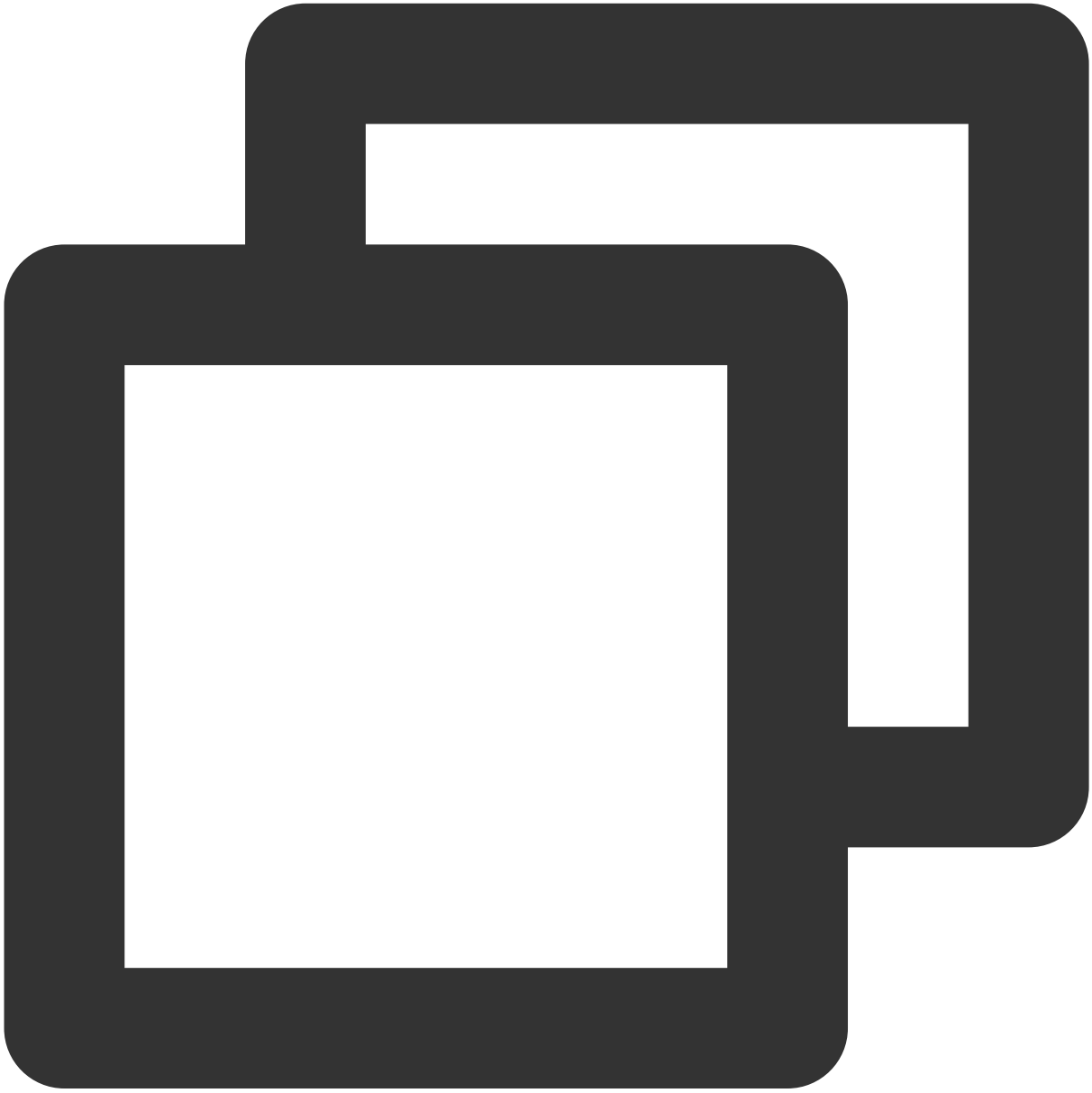
[ORDER BY expression [ASC | DESC] [NULLS FIRST | NULLS LAST] [, ...]]

按一个或多个输出 expression 对结果集进行排序，当子句包含多个表达式时，将根据第一个 `expression` 对结果集进行排序。然后，第二个 `expression` 应用于具有第一个表达式中的匹配值的行，以此类推。

示例

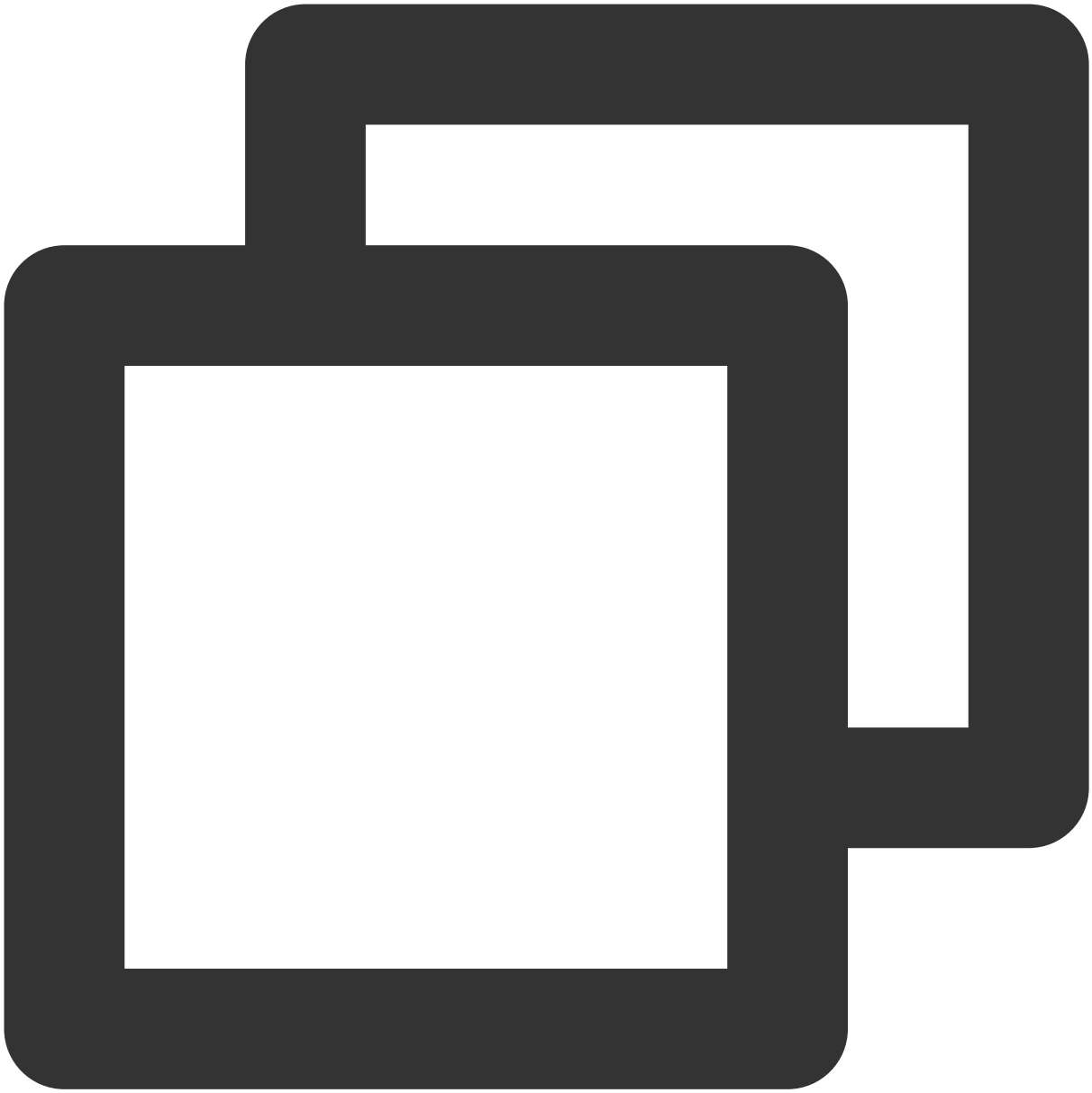
WITH Clause

WITH 子句定义在查询中使用的命名关系。它允许展平嵌套查询或简化子查询。例如，以下查询是等效的：



```
WITH x AS (SELECT a, MAX(b) AS b FROM t GROUP BY a)
SELECT a, b FROM x;
```

也可以跟多个子查询：

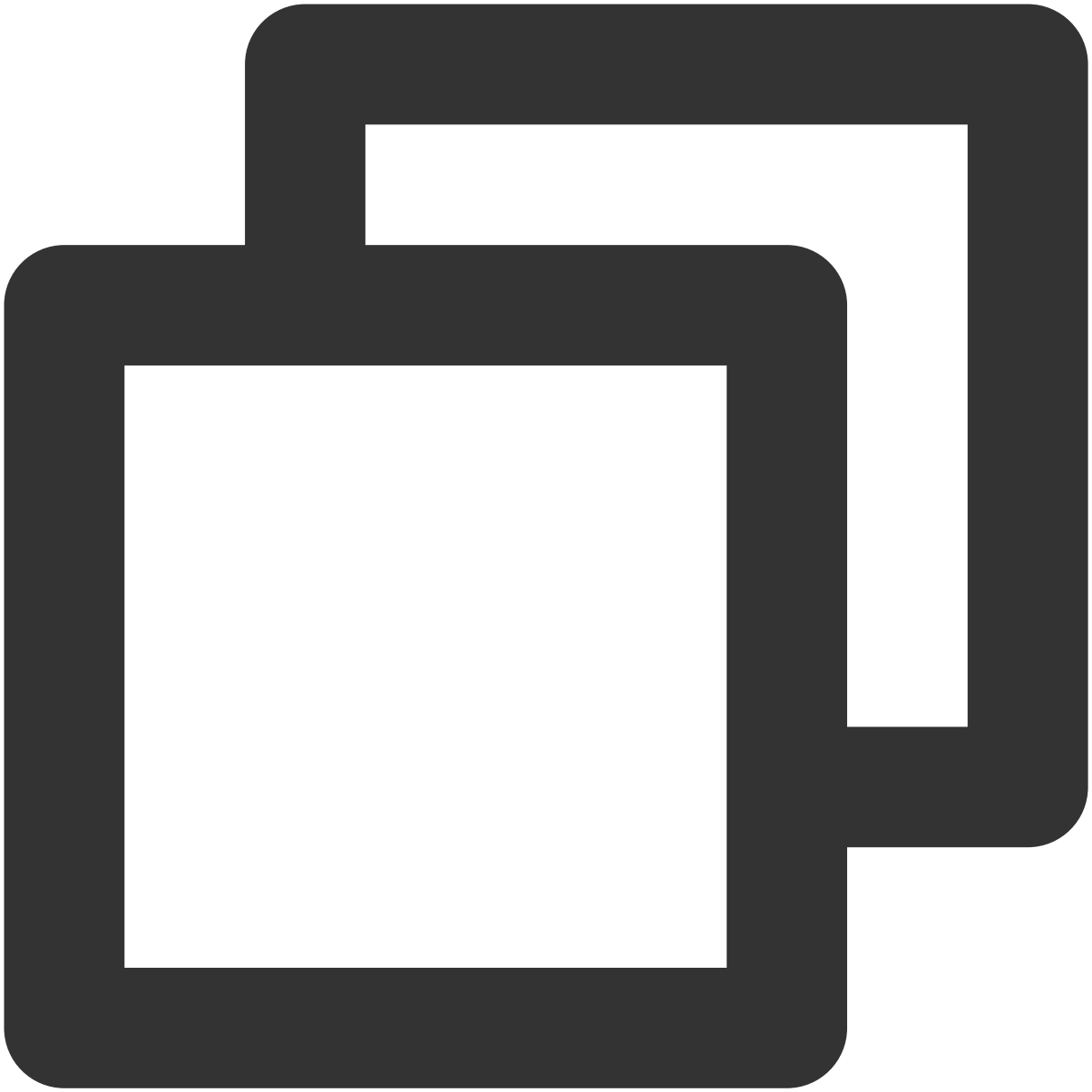


```
WITH
  t1 AS (SELECT a, MAX(b) AS b FROM x GROUP BY a),
  t2 AS (SELECT a, AVG(d) AS d FROM y GROUP BY a)
SELECT t1.*, t2.*
FROM t1
JOIN t2 ON t1.a = t2.a;
```

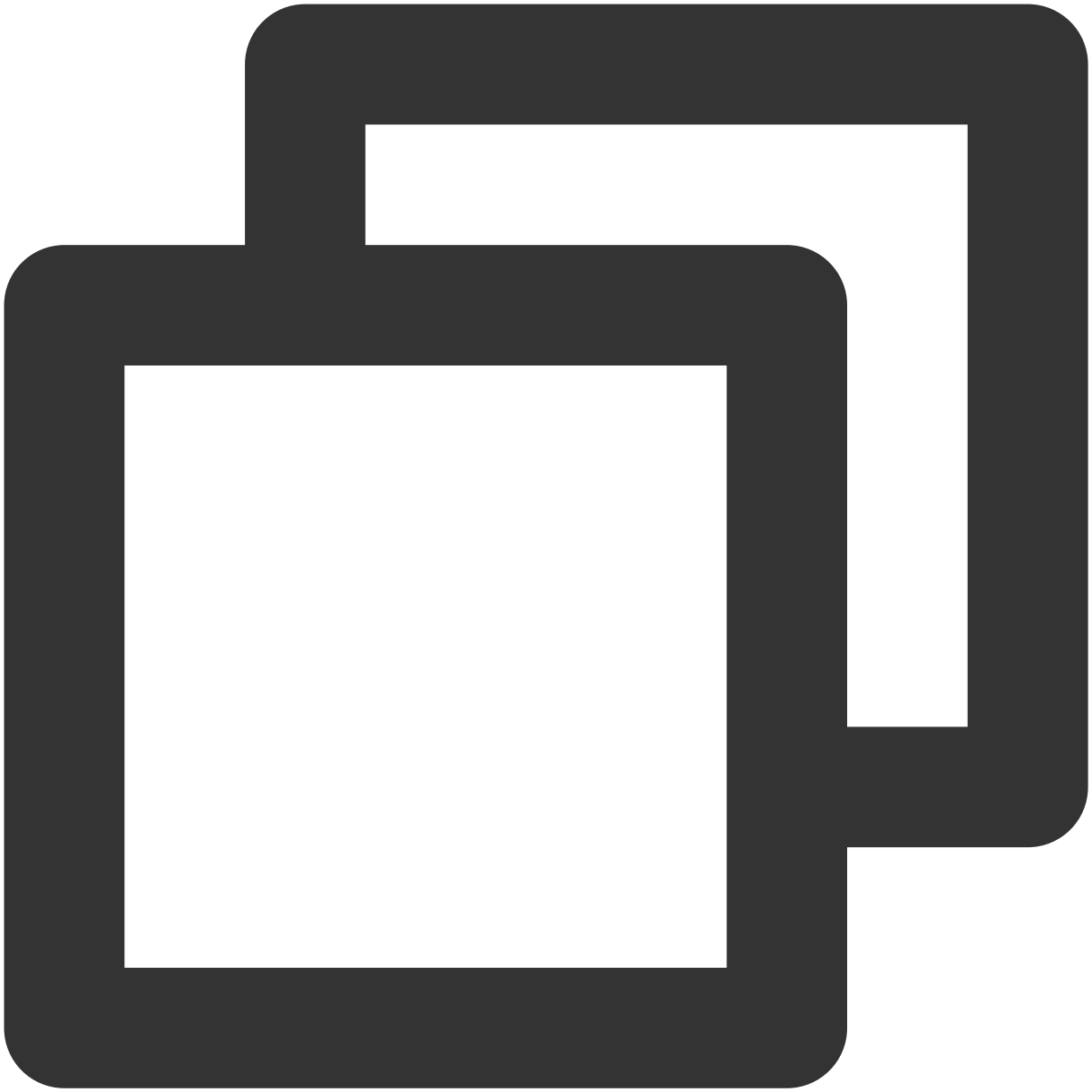
GROUP BY Clause

GROUPBY 子句将 SELECT 语句的输出分成包含匹配值的行组。简单的 GROUPBY 子句可以包含由输入列组成的任何表达式，也可以是按位置选择输出列的序数：

```
SELECT count(*), nationkey FROM customer GROUP BY 2;
```



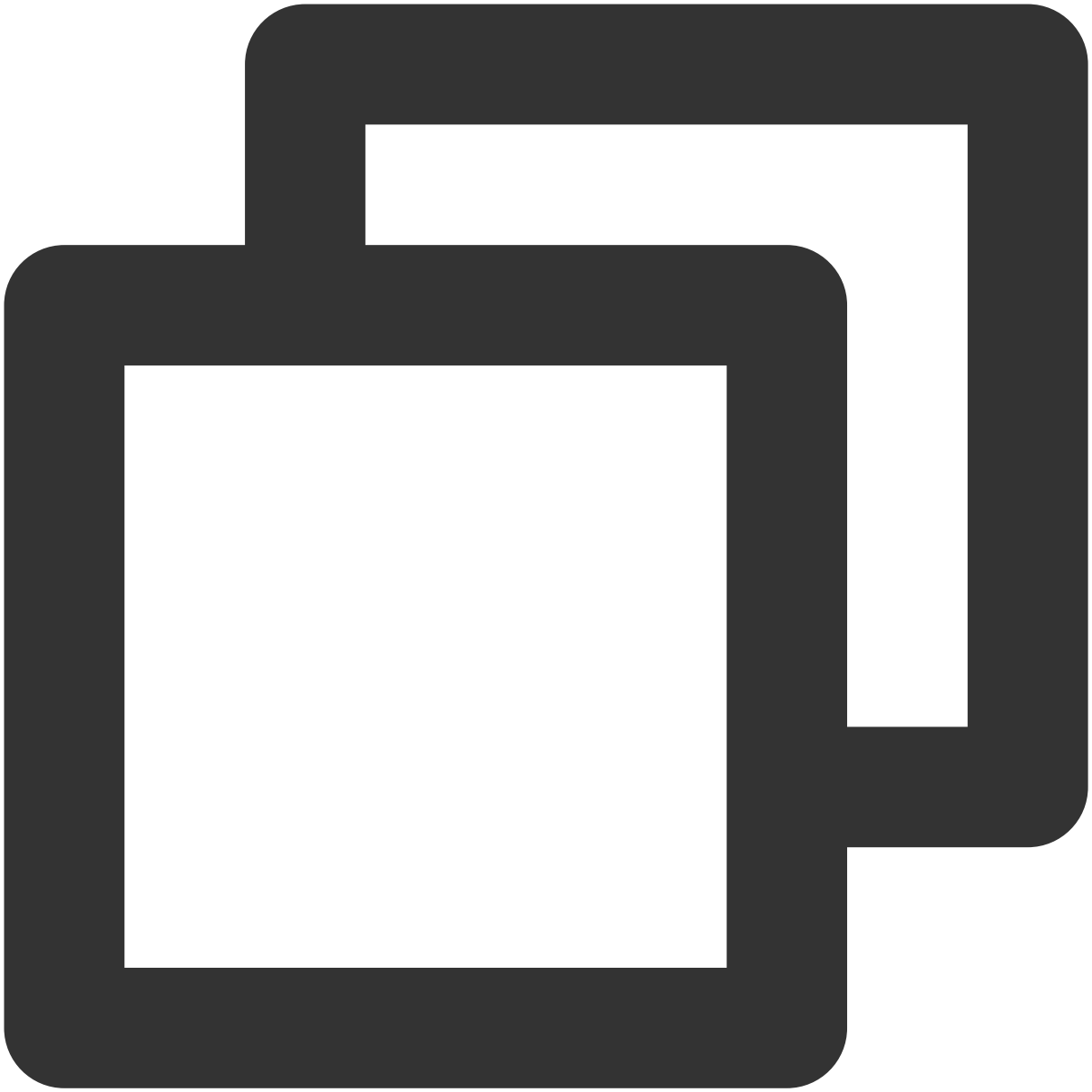
```
SELECT count(*), nationkey FROM customer GROUP BY nationkey;
```



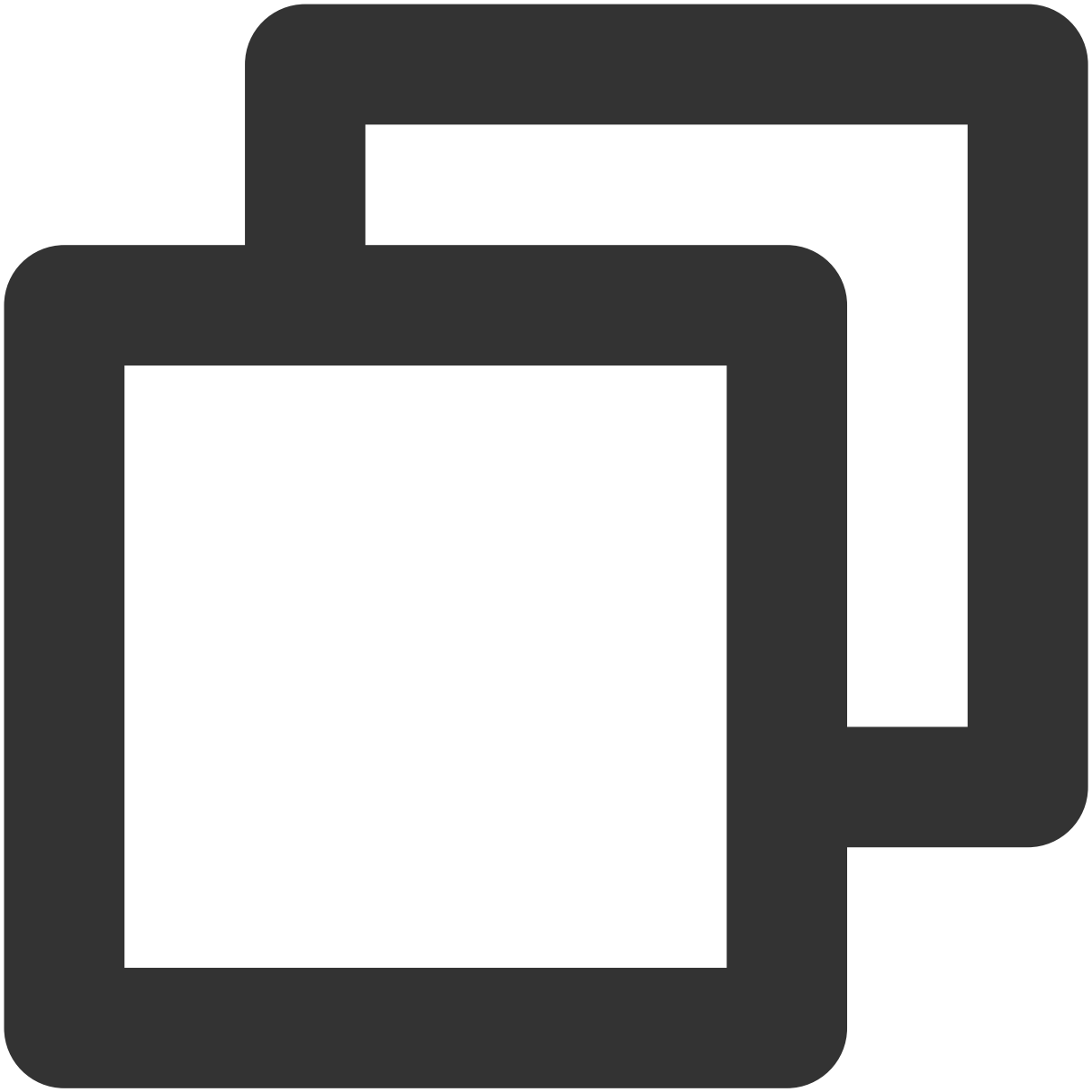
```
SELECT count(*) FROM customer GROUP BY mktsegment;
```

GROUPING SETS

分组集允许用户指定要分组的列的多个列表。不属于给定分组列子列表的列被设置为空。



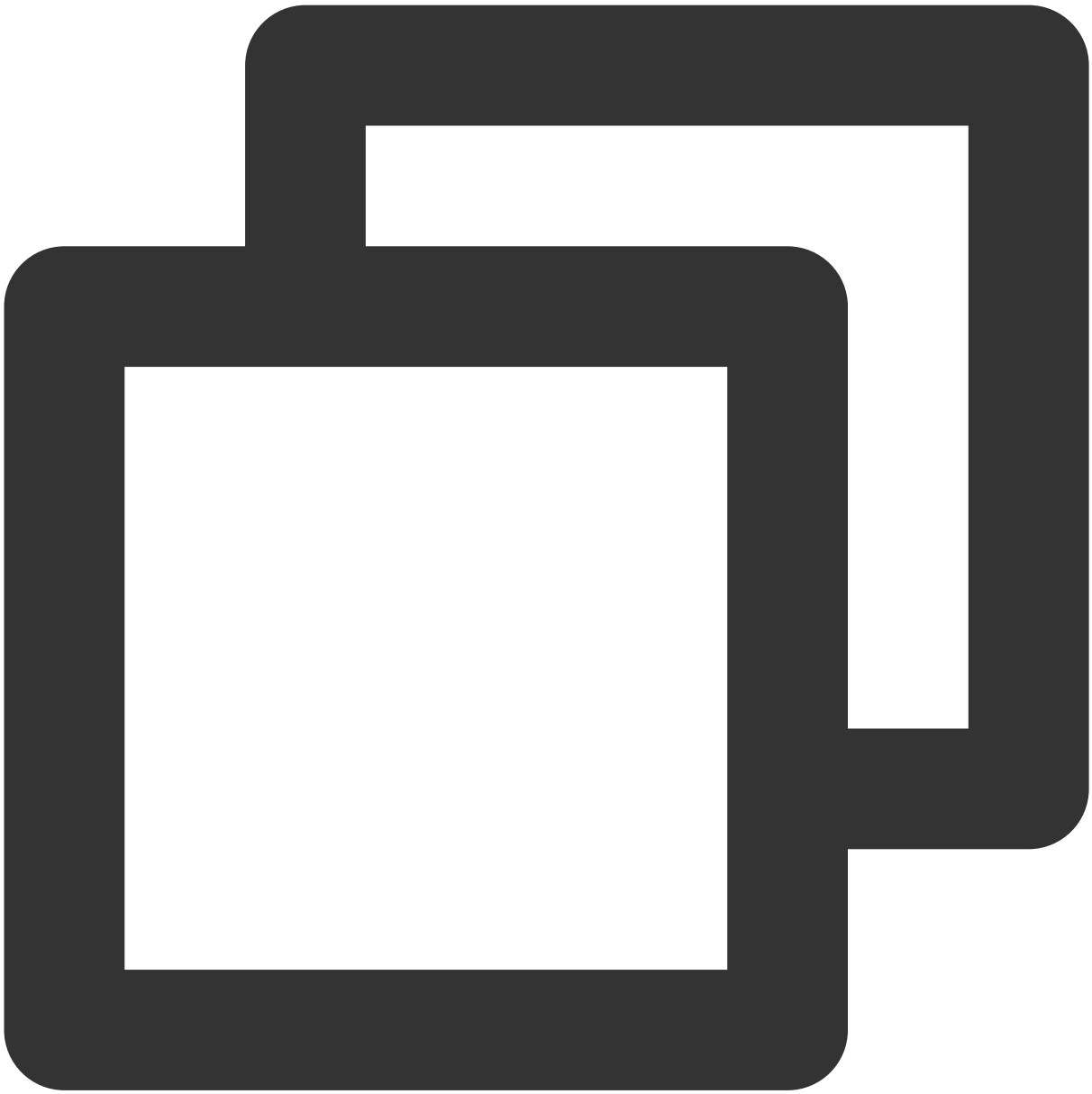
```
SELECT origin_state, origin_zip, destination_state, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state),
    (origin_state, origin_zip),
    (destination_state));
```



```
SELECT origin_state, NULL, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state
UNION ALL
SELECT origin_state, origin_zip, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state, origin_zip
UNION ALL
SELECT NULL, NULL, destination_state, sum(package_weight)
FROM shipping GROUP BY destination_state;
```

HAVING Clause

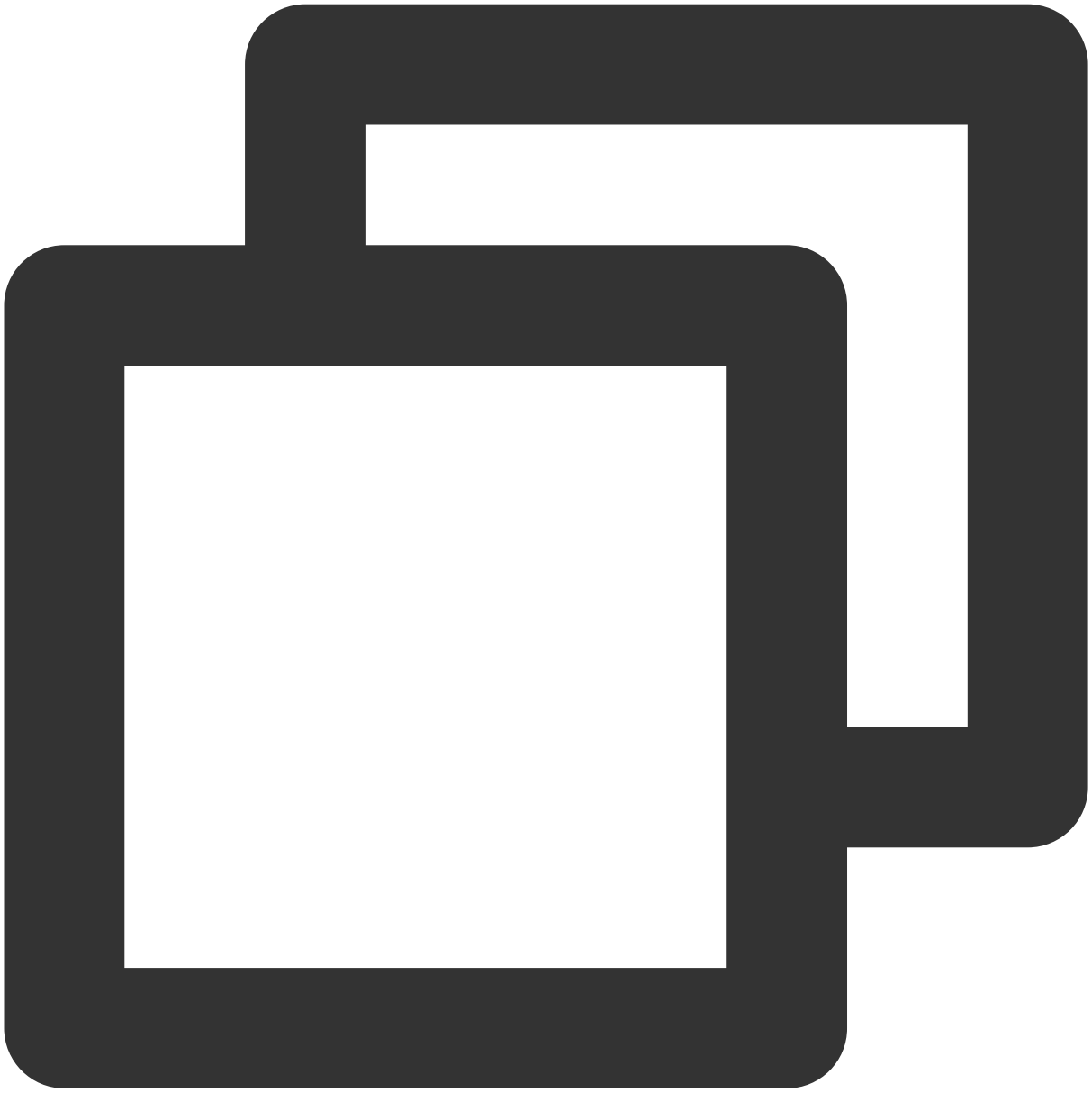
HAVING 子句与 aggregate 函数和 groupby 子句一起使用，以控制选择哪些组。HAVING 子句消除了不满足给定条件的组。



```
SELECT count(*), mktsegment, nationkey,
       CAST(sum(acctbal) AS bigint) AS totalbal
FROM customer
GROUP BY mktsegment, nationkey
HAVING sum(acctbal) > 5700000
ORDER BY totalbal DESC;
```

IN

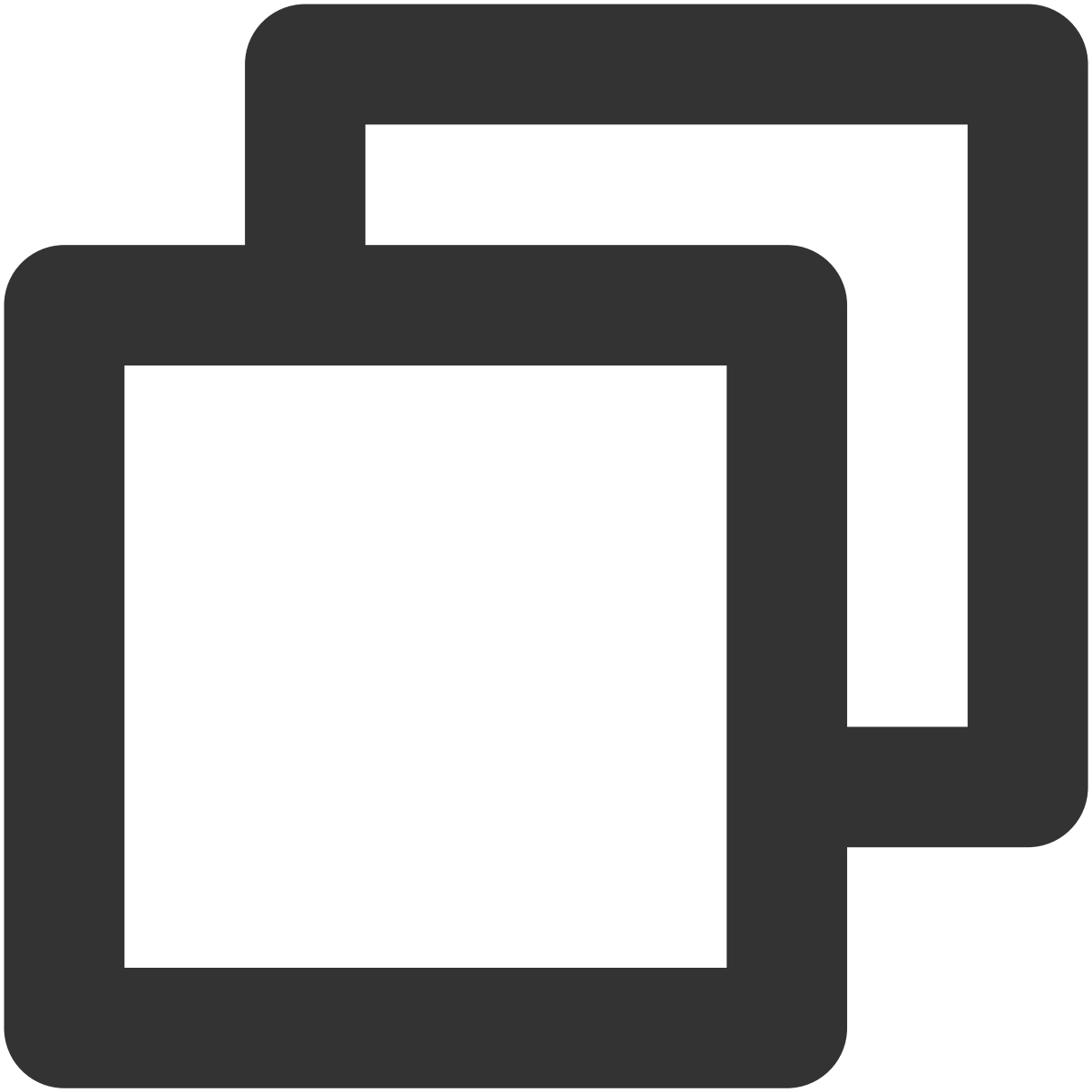
IN 操作符允许您在 WHERE 子句中规定多个值。



```
SELECT name
FROM nation
WHERE regionkey IN (SELECT regionkey FROM region)
```

EXISTS

EXISTS 运算符用于判断查询子句是否有记录，如果有一条或多条记录存在返回 True，否则返回 False。



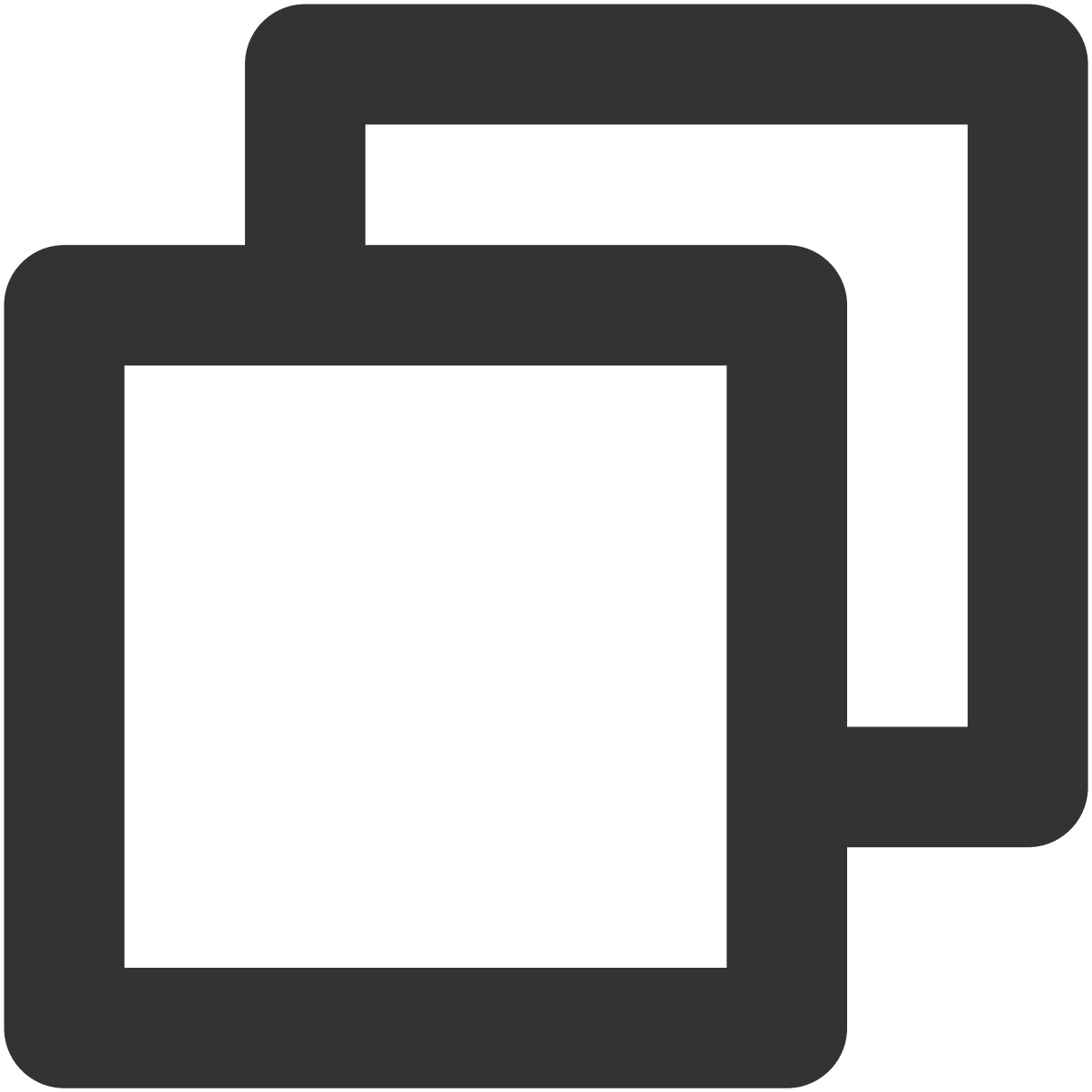
```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition)
```

USING

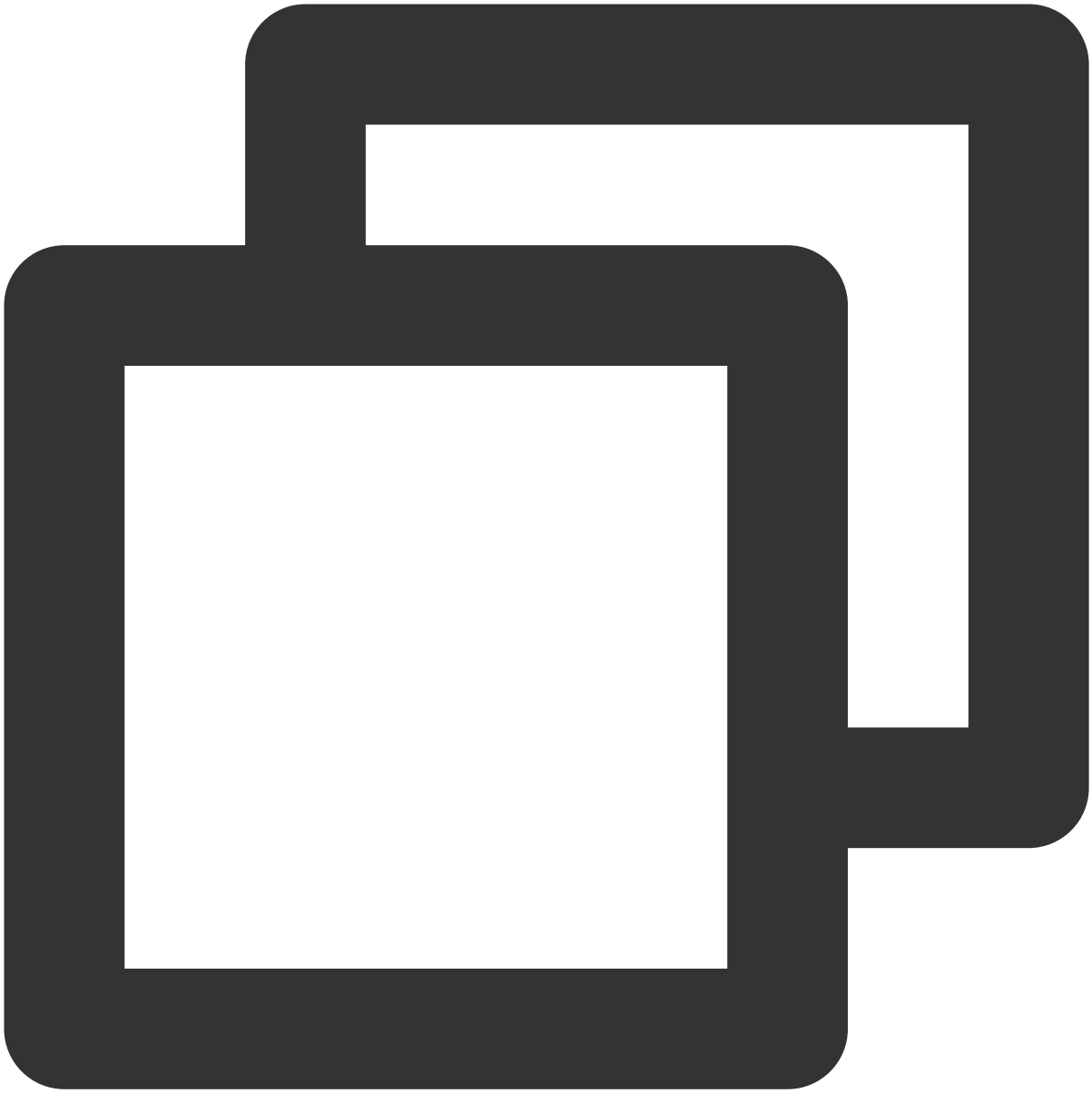
用 using 关键字进行简化。

查询必须是相等条件连接。

等值连接中的列必须具有相同的名称和数据类型。



```
SELECT *
FROM table_1
JOIN table_2
USING (key_A, key_B)
```

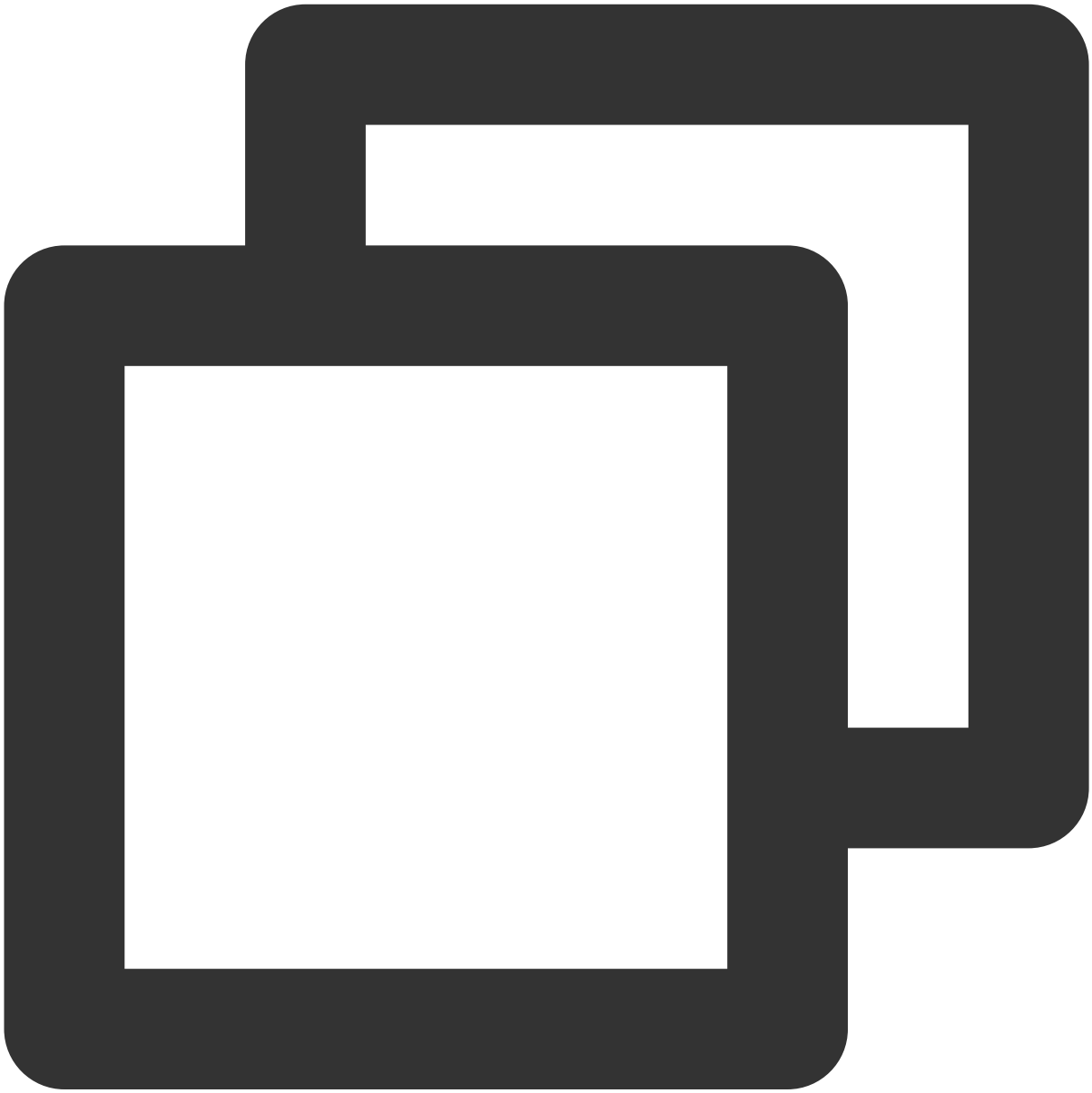


```
SELECT *
FROM (
    VALUES
        (1, 3, 10),
        (2, 4, 20)
) AS table_1 (key_A, key_B, y1)
LEFT JOIN (
    VALUES
        (1, 3, 100),
        (2, 4, 200)
) AS table_2 (key_A, key_B, y2)
```

```
USING (key_A, key_B);
```

CROSS JOIN

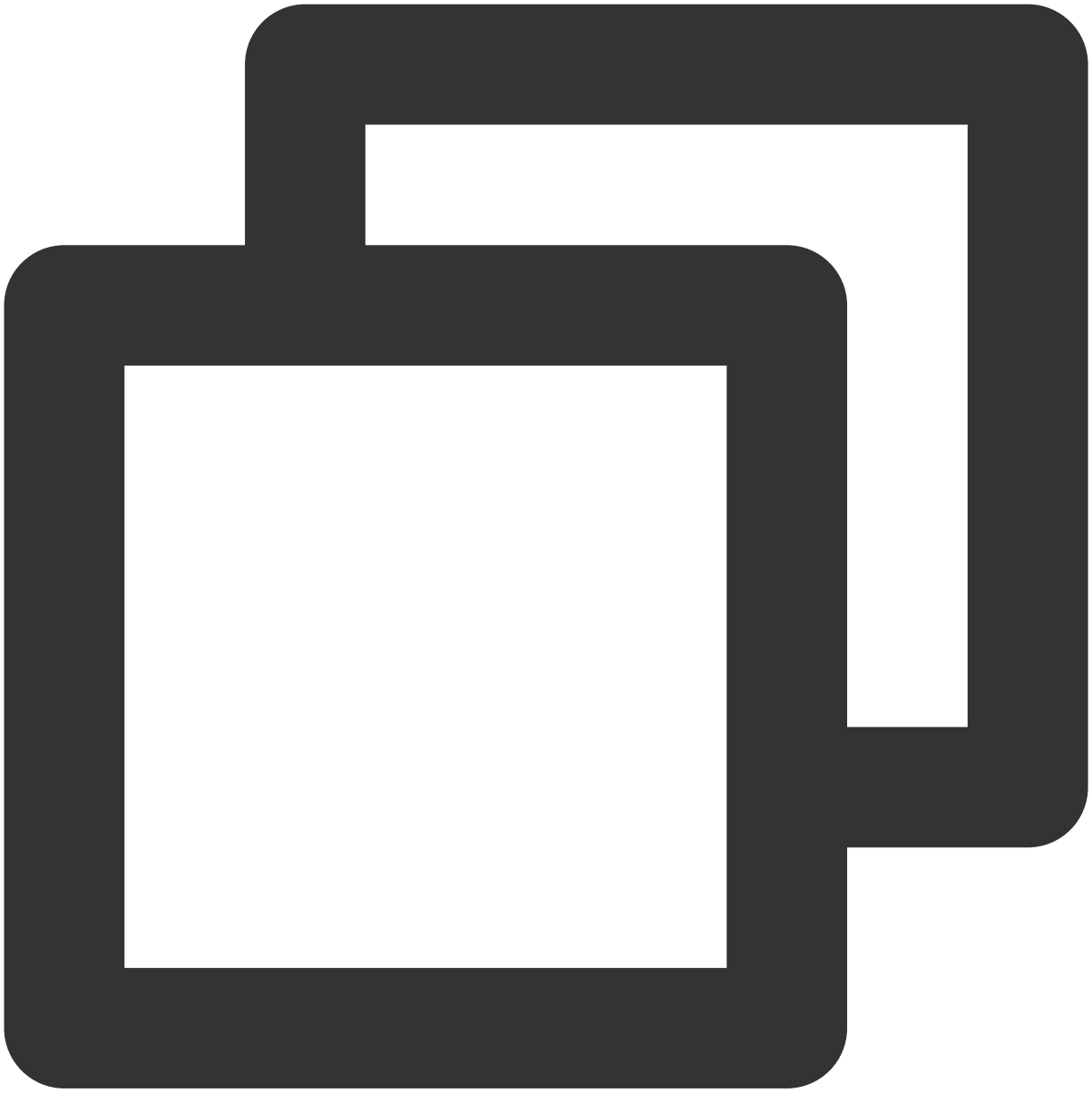
交叉连接返回两个关系的笛卡尔积（所有组合）。



```
SELECT *
FROM nation
CROSS JOIN region
```

LIMIT Clause

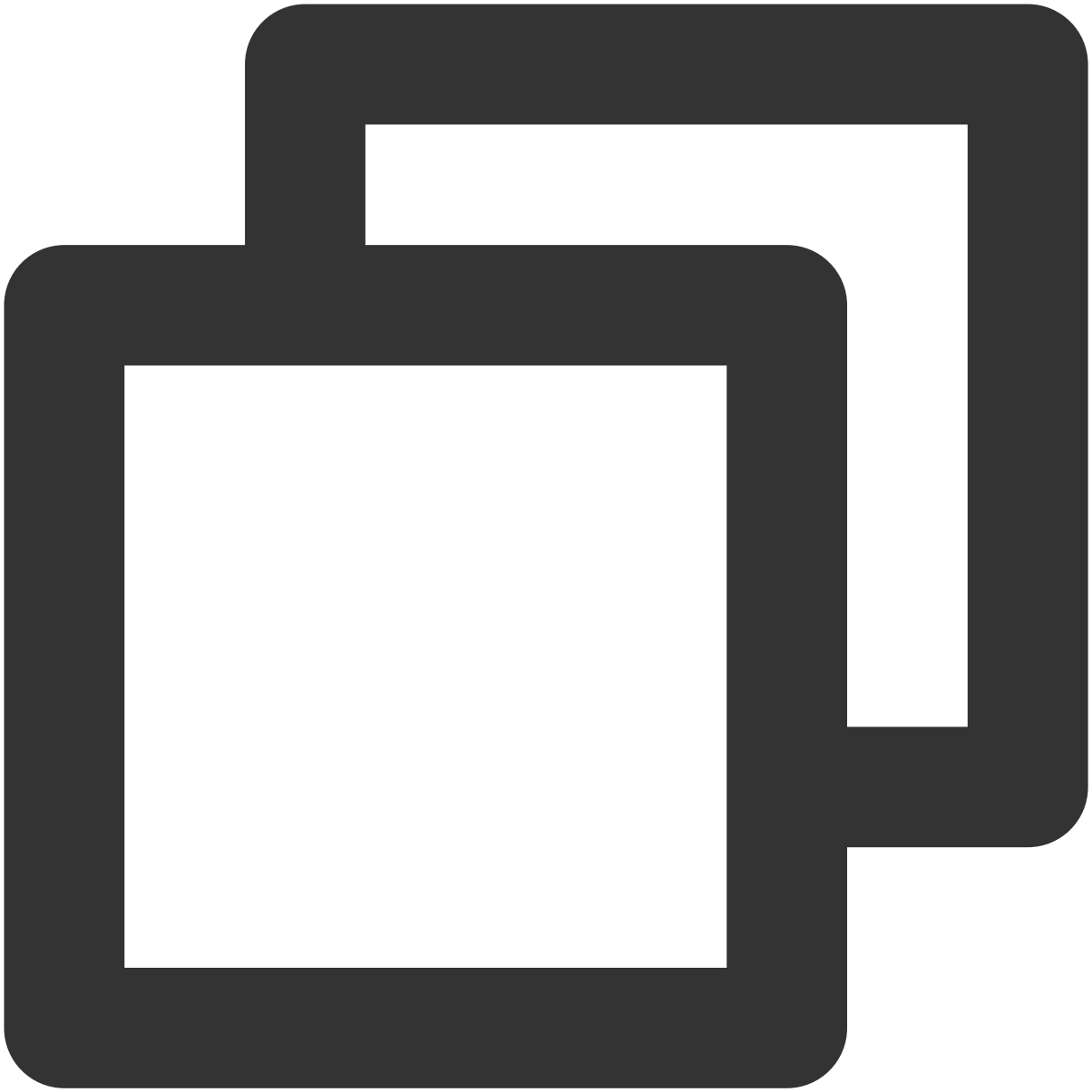
LIMIT 子句限制结果集中的行数。



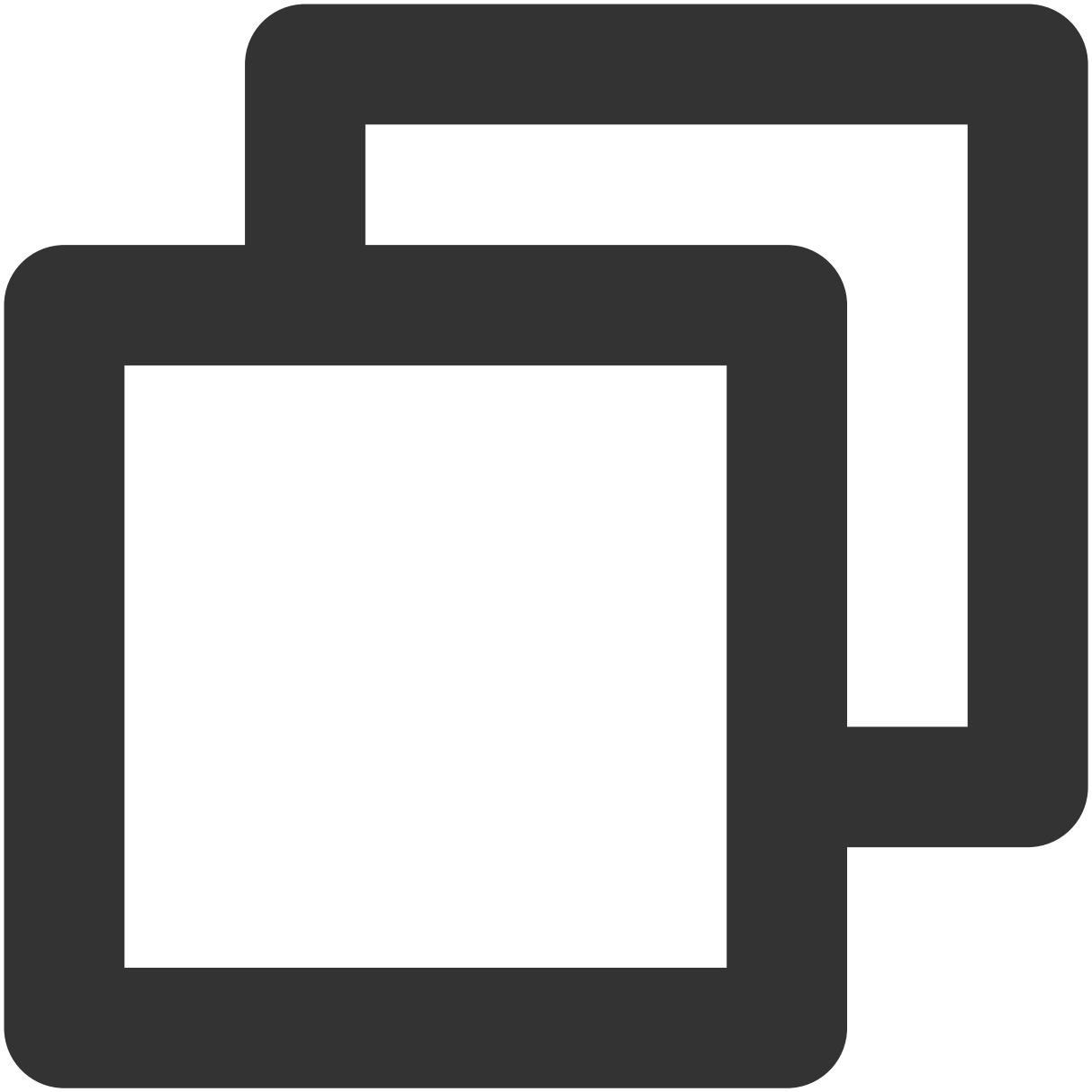
```
SELECT orderdate FROM orders LIMIT 5
```

ORDER BY Clause

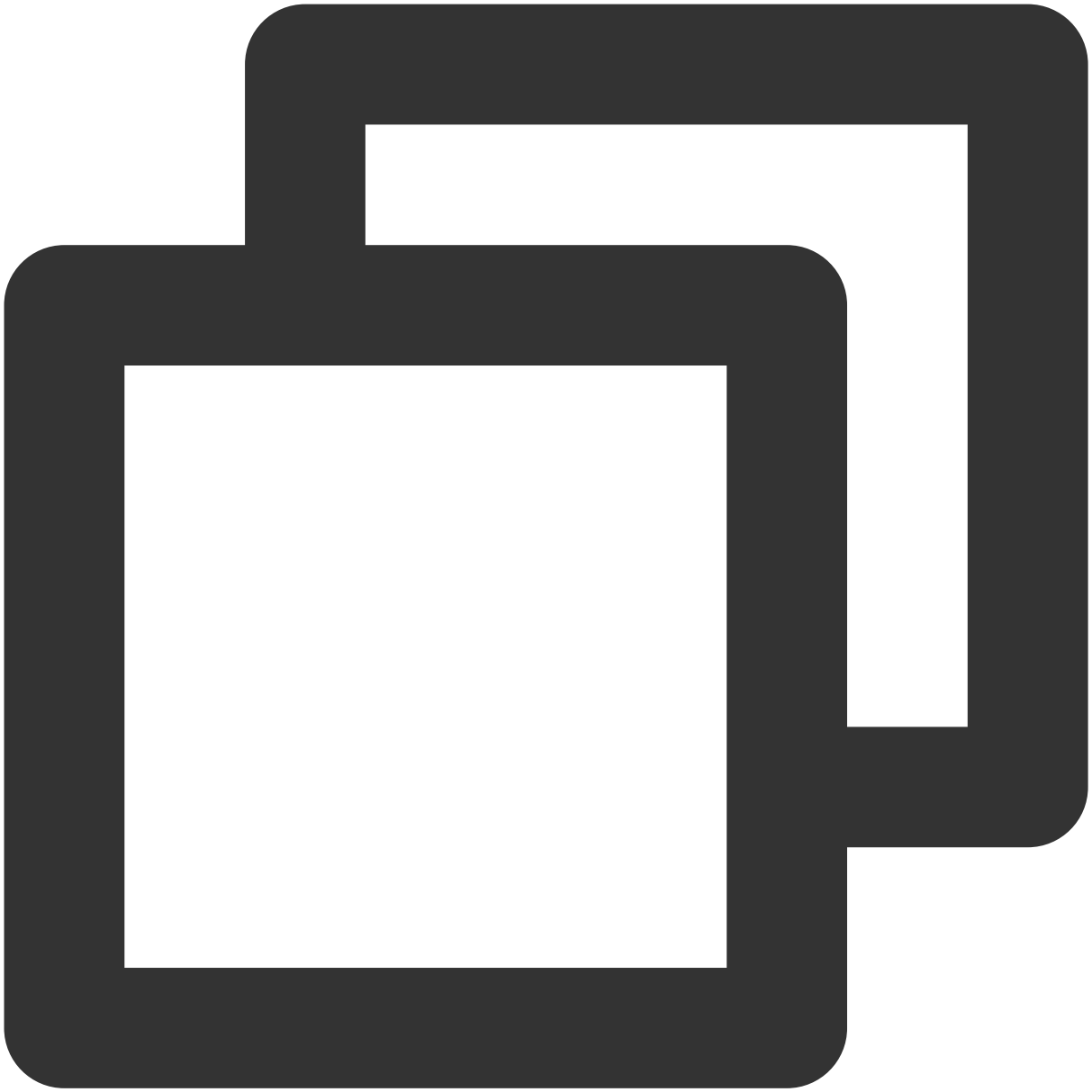
ORDER BY 子句用于按一个或多个输出表达式对结果集进行排序。



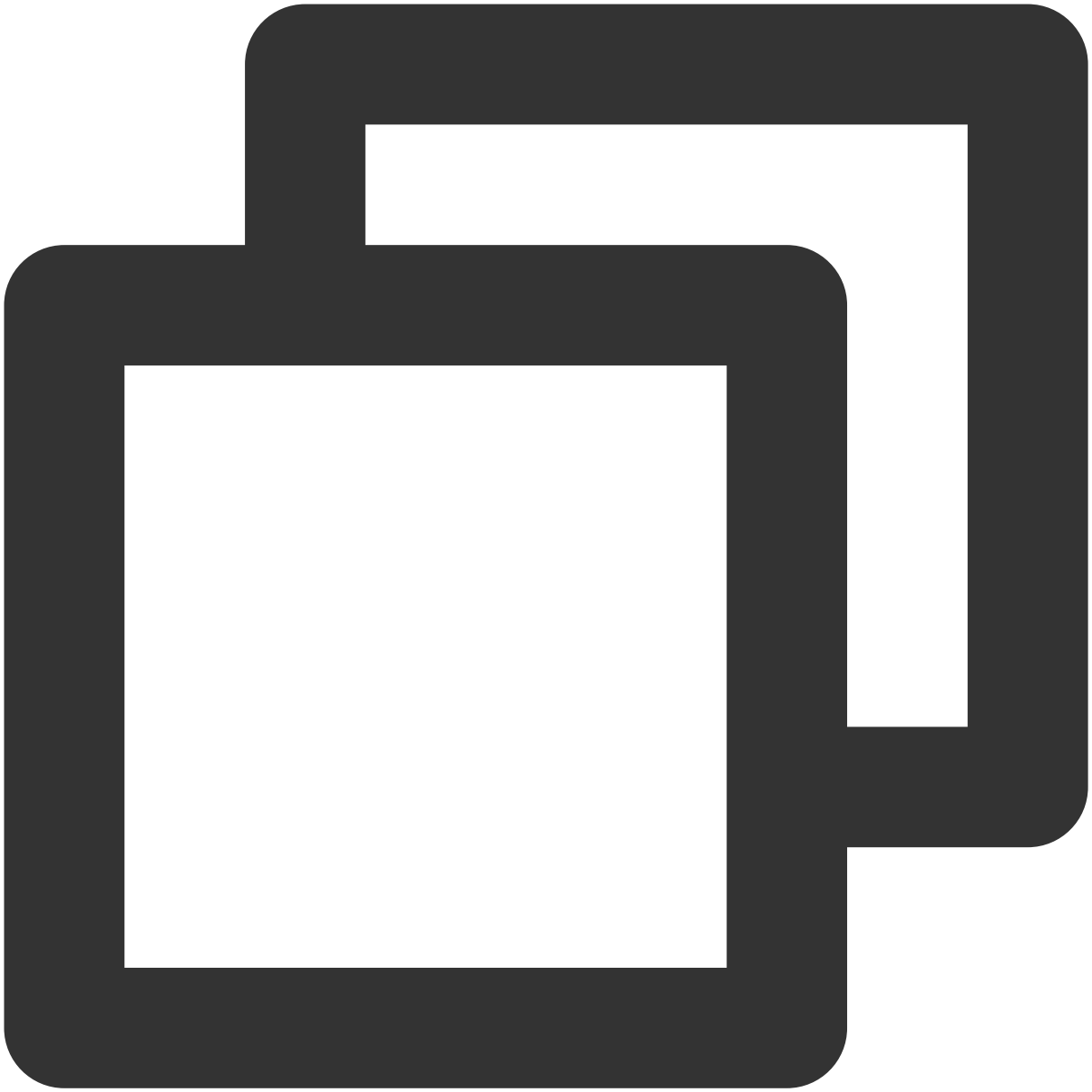
```
语法：ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...]
```



```
SELECT name, age FROM person ORDER BY age
```



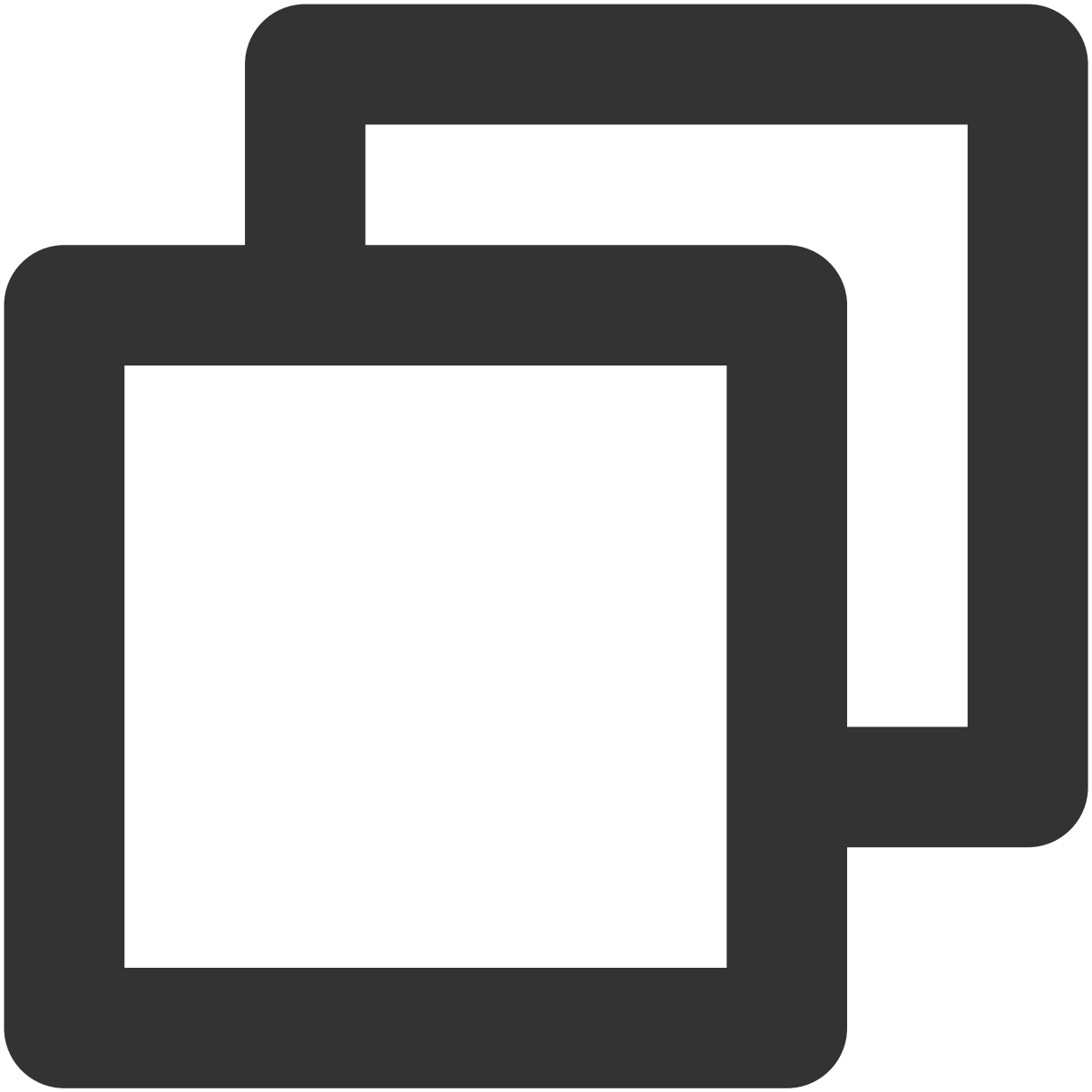
```
SELECT * FROM student
ORDER BY student_id
```



```
SELECT * FROM student
ORDER BY student_id,student_name
```

EXCEPT

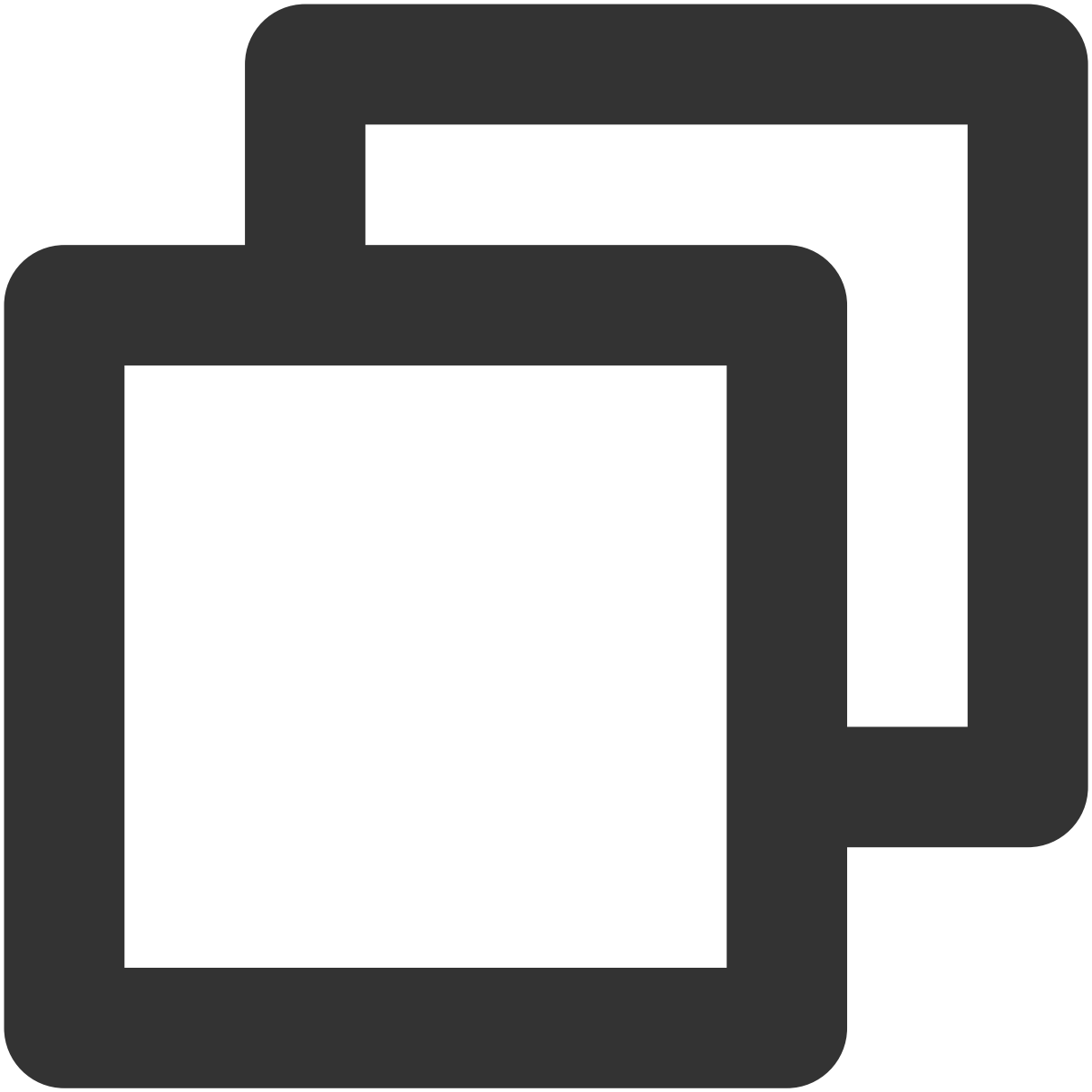
EXCEPT 子句/操作符用于合并两个 SELECT 语句，并从那些没有被第二个 SELECT 语句返回的第一个 SELECT 语句返回行。这意味着 EXCEPT 仅返回行，在第二个 SELECT 语句不可用。
正如使用 UNION 操作，同样的规则时，使用 EXCEPT 操作符适用。



```
SELECT * FROM (VALUES 13, 42)
EXCEPT
SELECT 13
```

INTERSECT

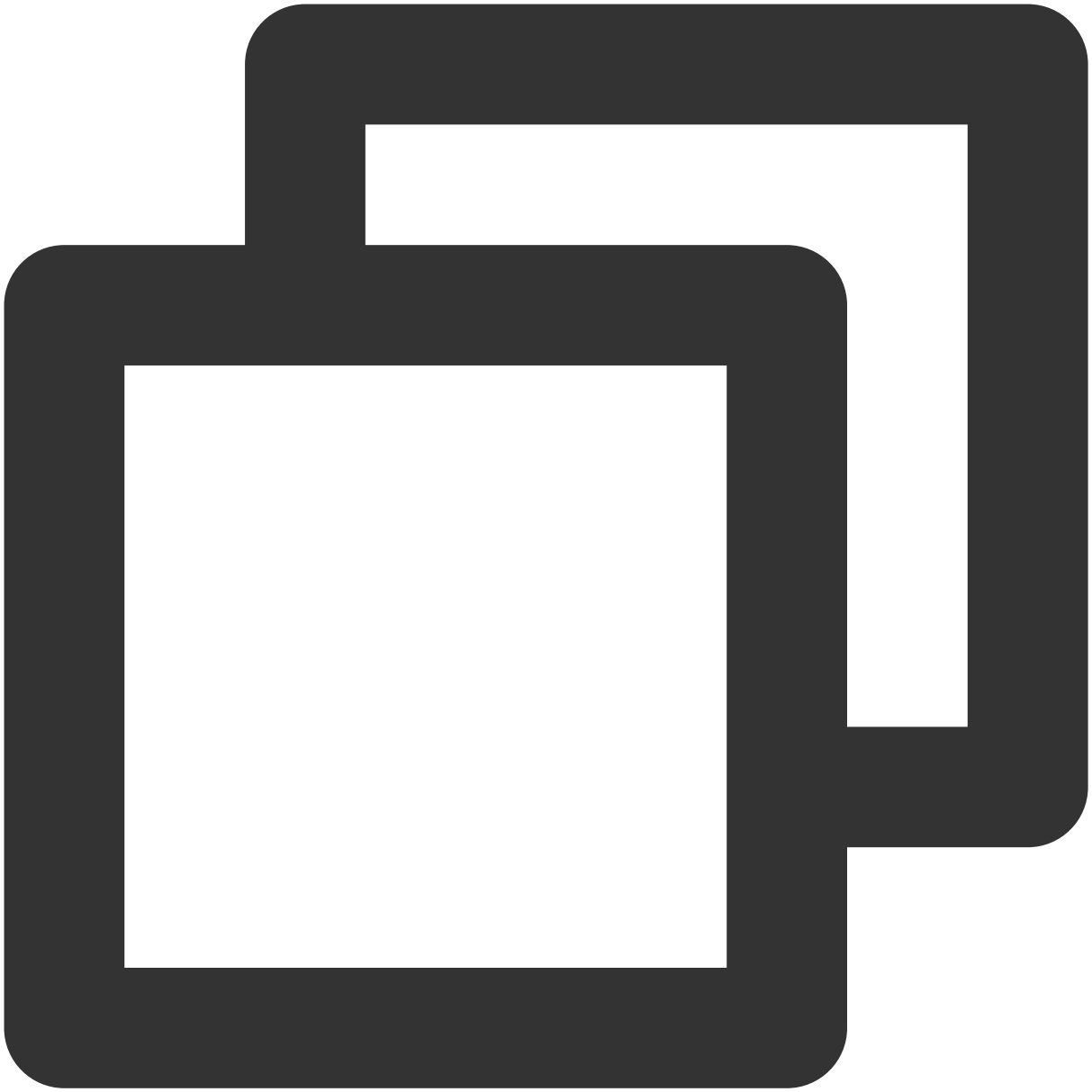
从 SELECT 语句返回两个或多个结果集的不同行。



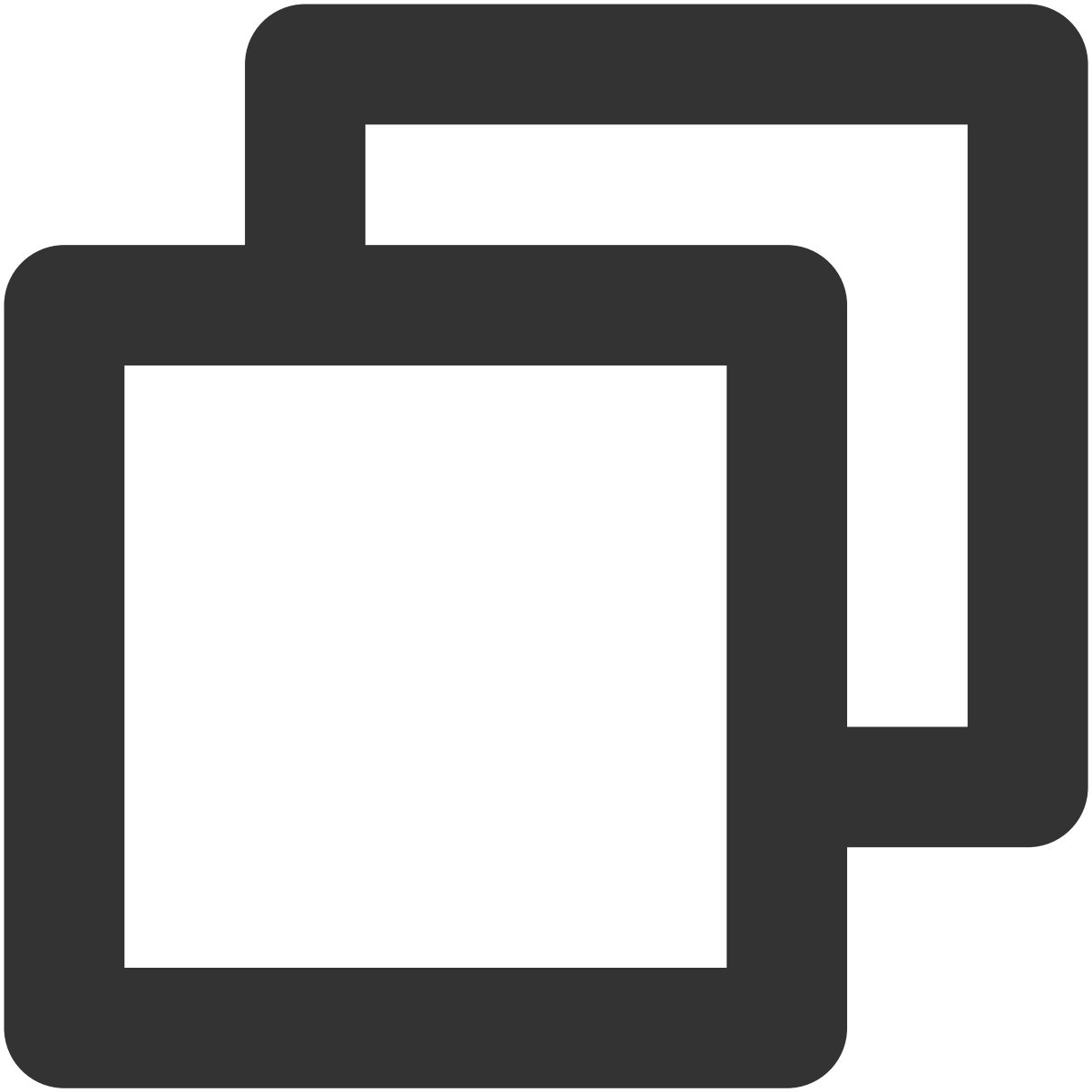
```
SELECT * FROM (VALUES 13, 42)
INTERSECT
SELECT 1
```

UNION

将两个或多个 SELECT 语句的结果集合并到一个结果集中。要保留结果集中的重复行，请使用 `UNION ALL` 运算符。



```
SELECT 13
UNION
SELECT 42
```

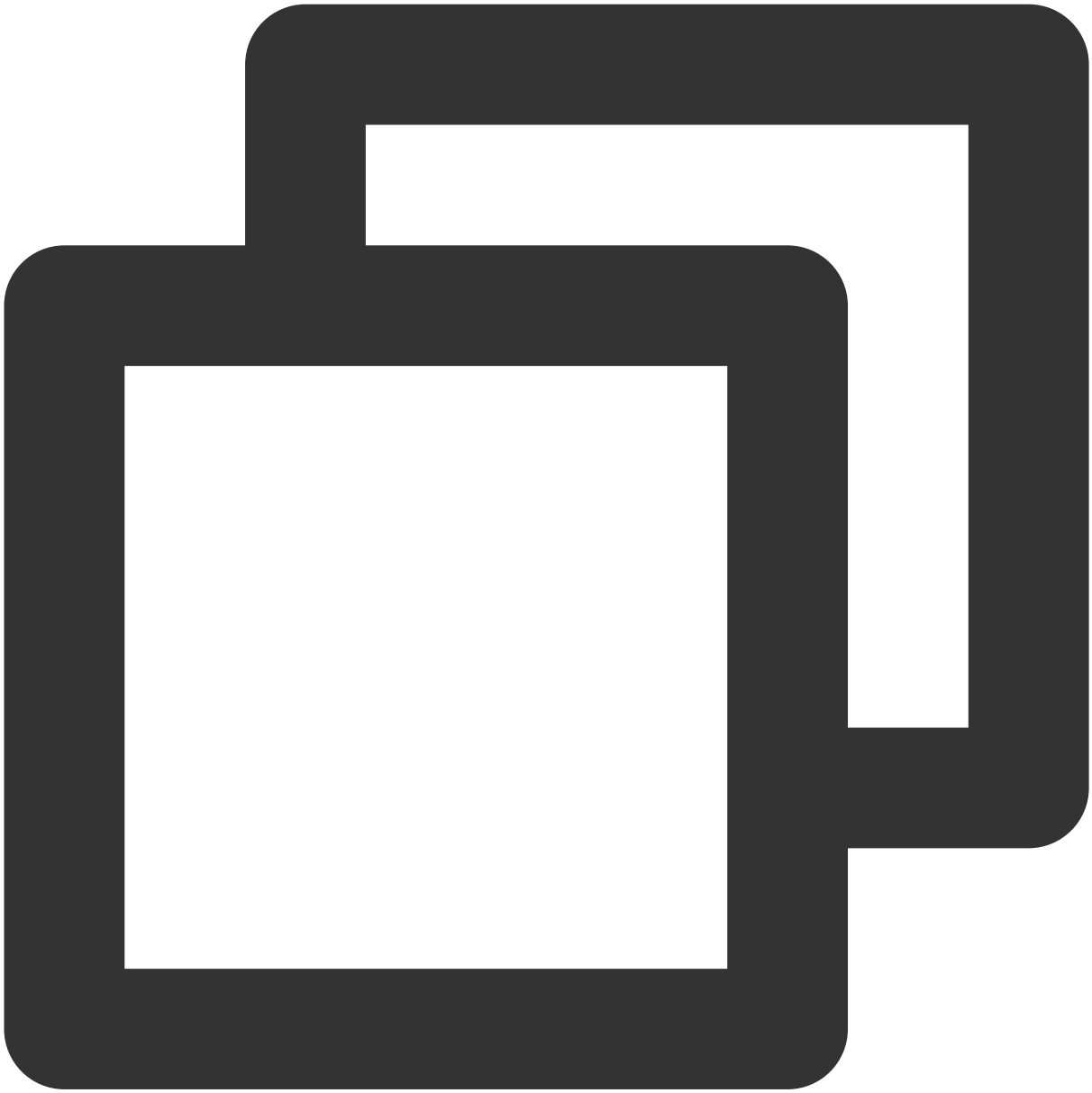


```
SELECT id FROM a
UNION ALL
SELECT id FROM b;
```

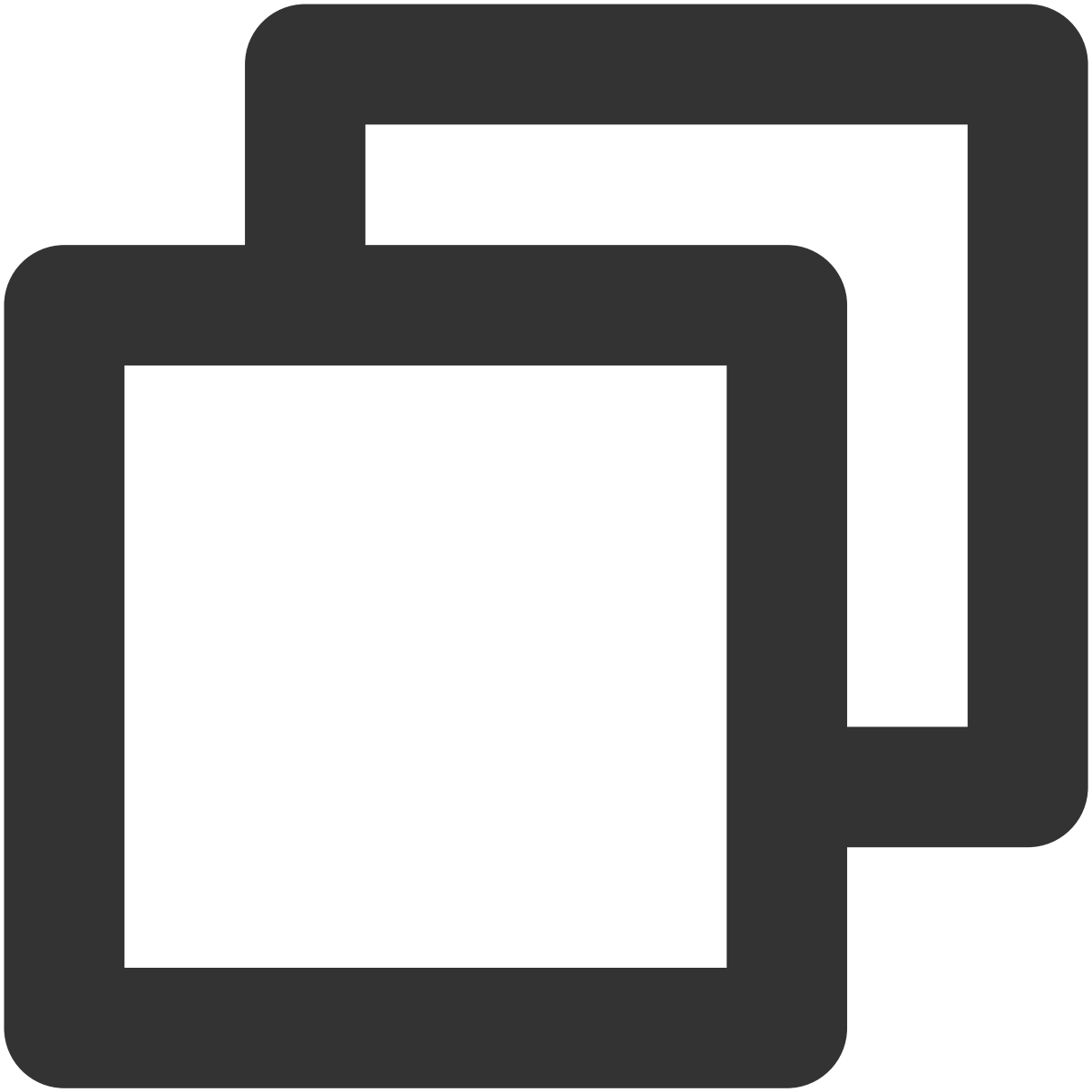
TABLESAMPLE

`BERNOULLI` ：选择每一行作为表样本，概率为样本百分比。使用 Bernoulli 方法对表进行采样时，将扫描表的所有物理块并跳过某些行（基于采样百分比与运行时计算的随机值之间的比较）。结果中包含一行的概率独立于任何其他行。这不会减少从磁盘读取采样表所需的时间。如果进一步处理采样输出，可能会影响总查询时间。

`SYSTEM` ：这种采样方法将表划分为逻辑数据段，并以此粒度对表进行采样。此采样方法要么从特定数据段中选择所有行，要么跳过它（基于采样百分比和运行时计算的随机值之间的比较）。在系统采样中选择的行将取决于所使用的连接器。例如，当与 Hive 一起使用时，它取决于数据在 HDFS 上的布局方式。这种方法不能保证独立的抽样概率。



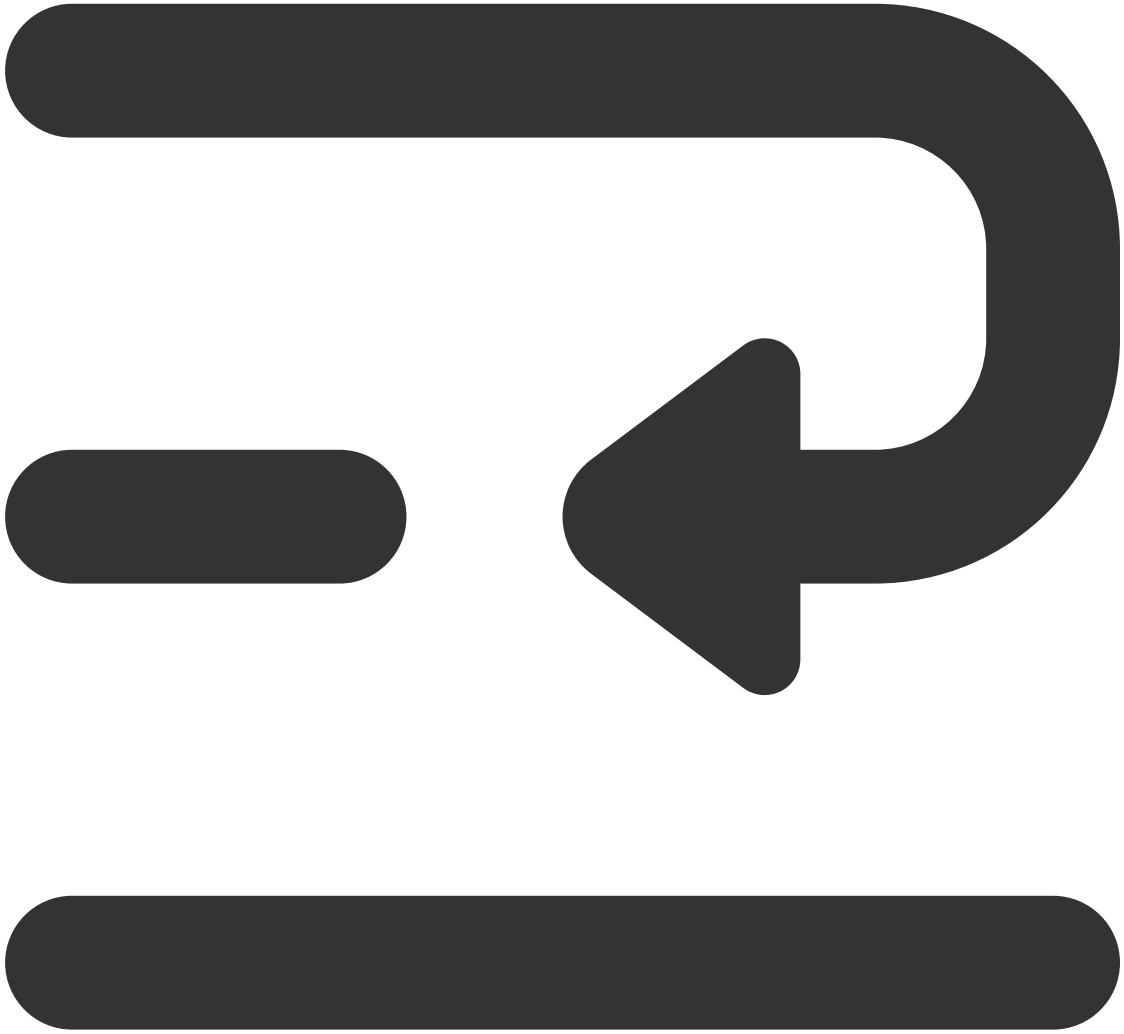
```
SELECT *
FROM users TABLESAMPLE BERNOULLI (50);
```



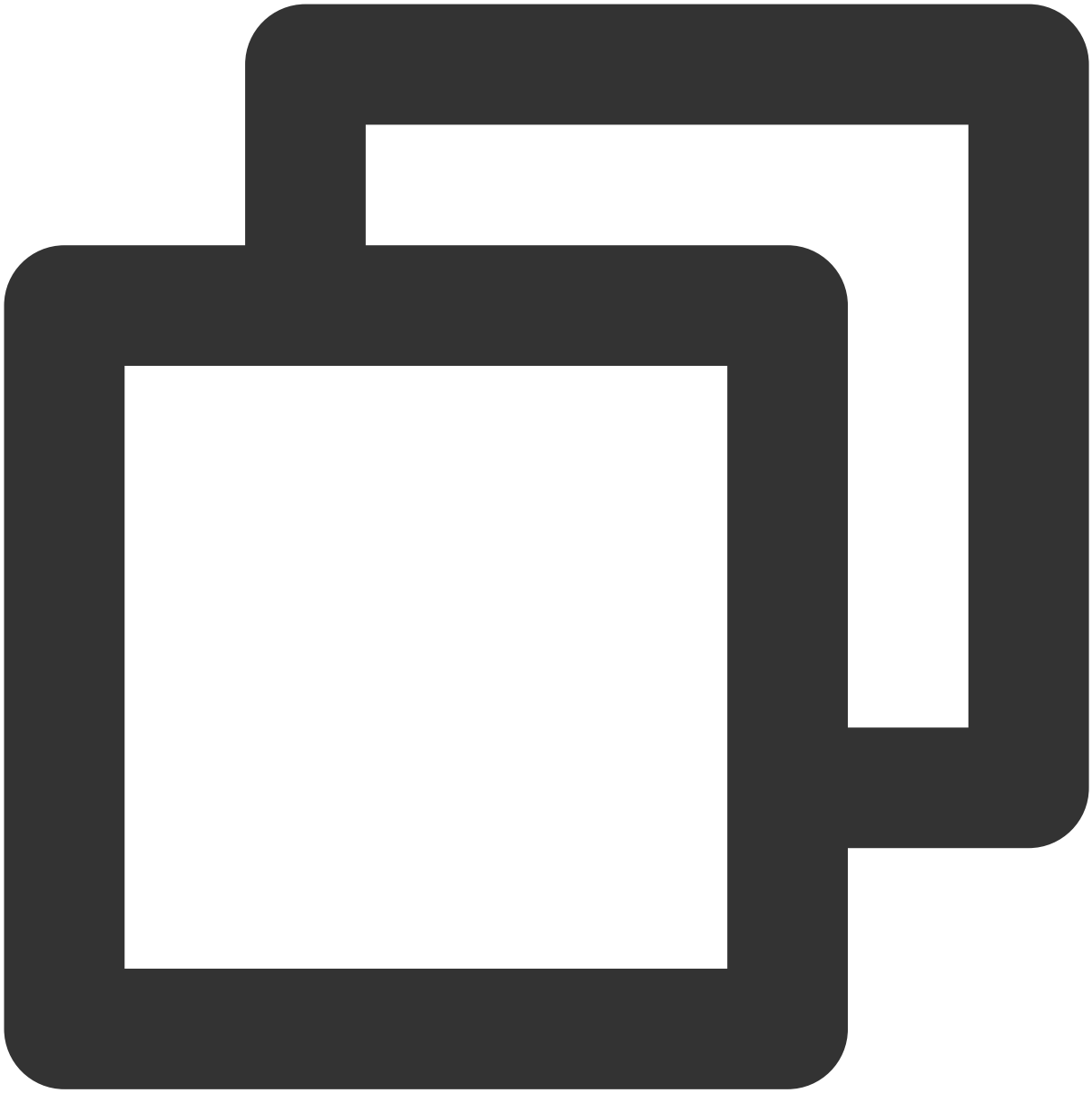
```
SELECT *
FROM users TABLESAMPLE SYSTEM (75);
```

PIVOT Clause

根据特定的列返回聚合值。

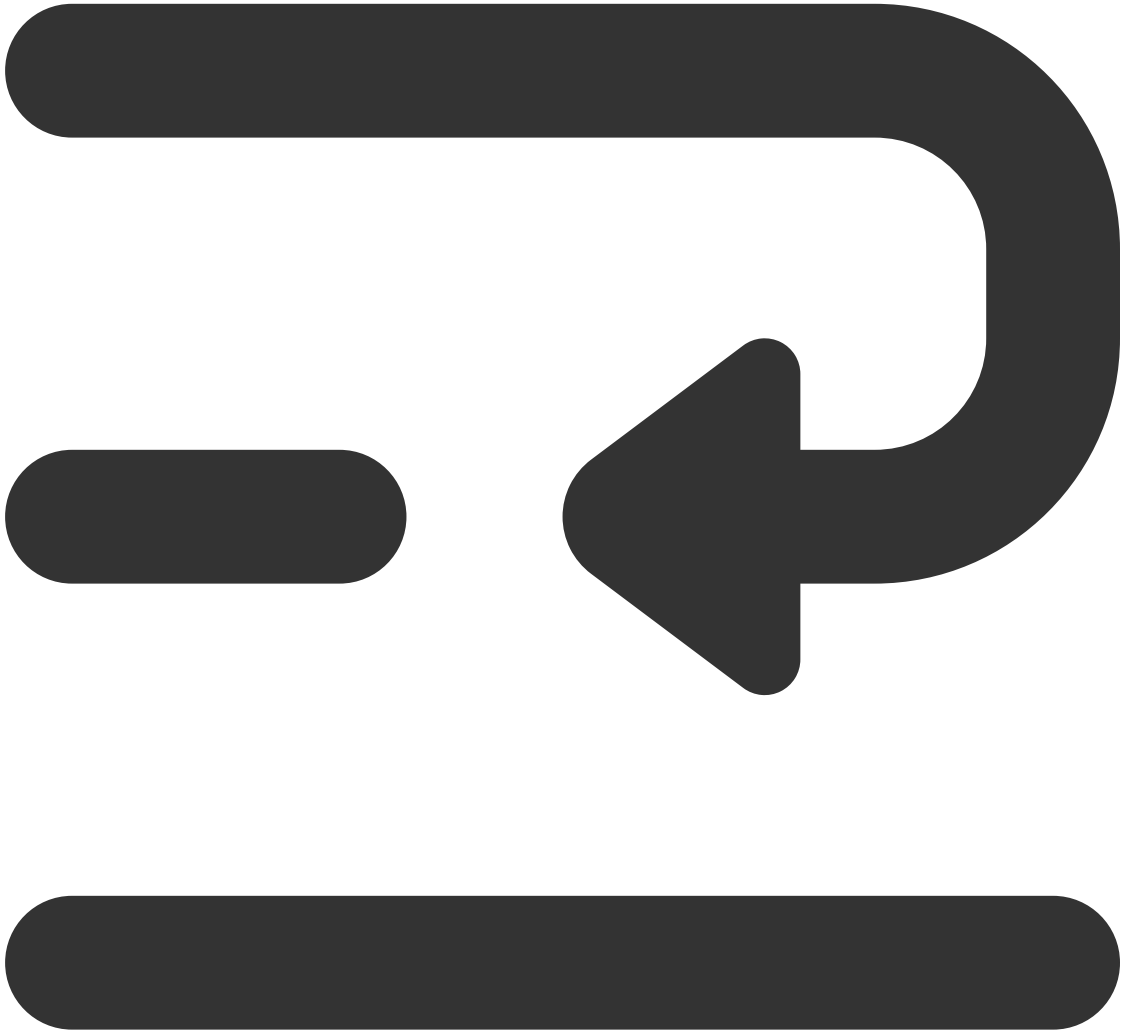


版权所有：腾讯云计算（北京）有限责任公司

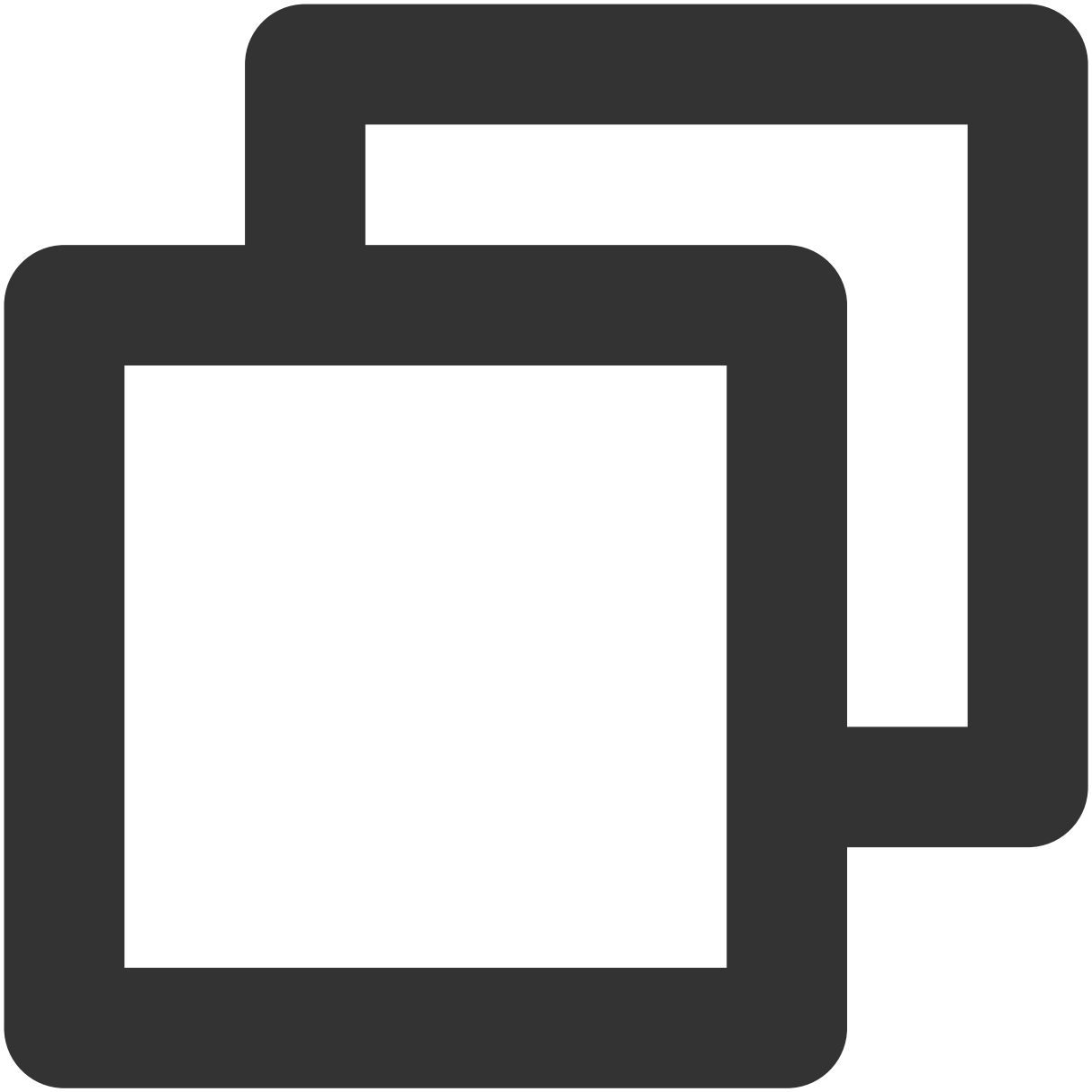


```
SELECT * FROM person
    PIVOT (
      SUM(age) AS a, AVG(class) AS c
      FOR name IN ('John' AS john, 'Mike' AS mike)
     );
```

Lateral View Clause



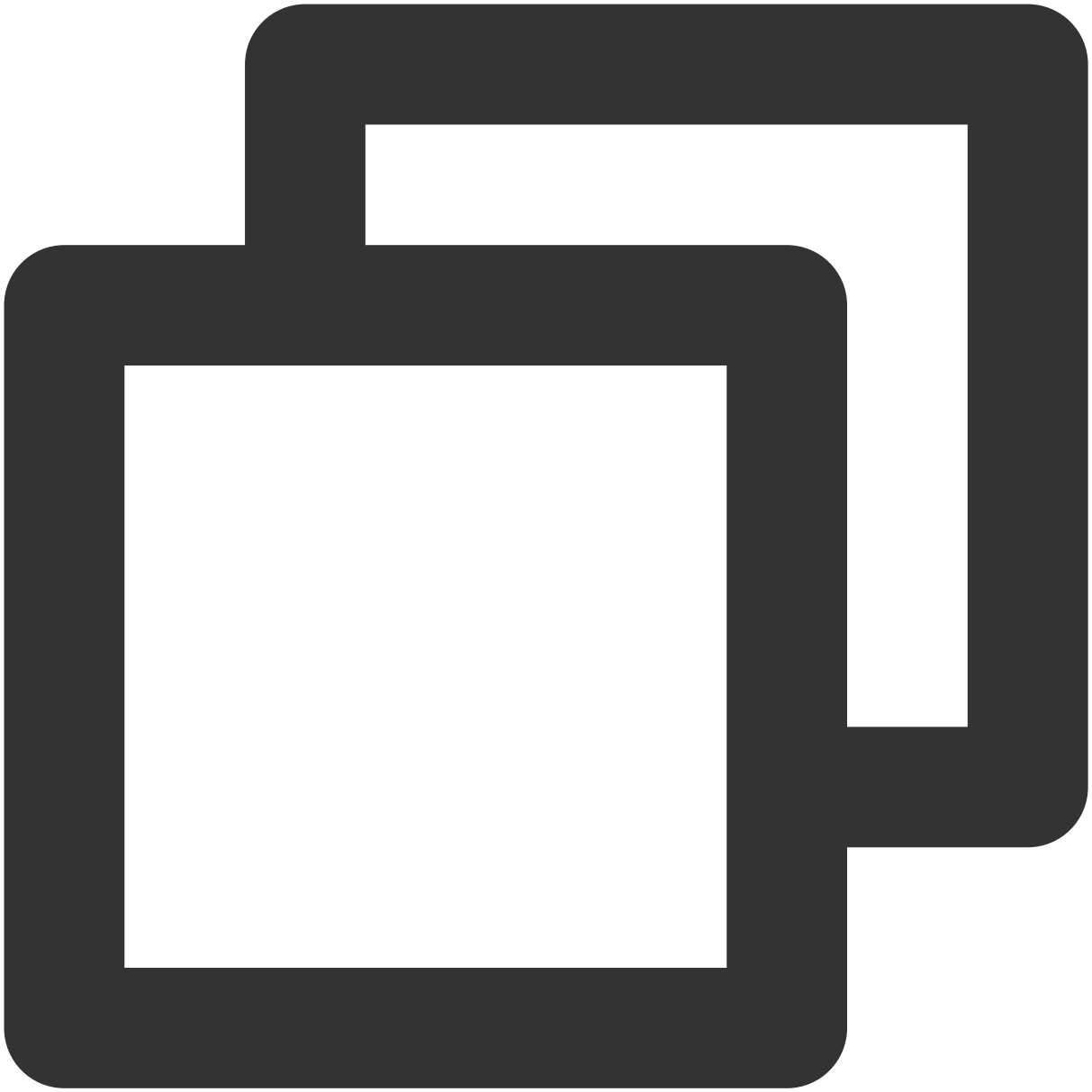
版权所有：腾讯云计算（北京）有限责任公司



```
LATERAL VIEW [ OUTER ] generator_function ( expression [ , ... ] ) [ table_alias ]
```

转义

要转义单引号，请在其前面加上另一个单引号，如以下示例所示：



```
Select 'dlc''test'
```

创建表的时候指定转义符或者逃逸符，例如使用下列方式创建：

```
CREATE EXTERNAL TABLE IF NOT EXISTS `csv_test_2222` (
      `_c0`  STRING,
      `_c1`  INTEGER,
      `_c2`  INTEGER,
      `_c3`  INTEGER
    )
    ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH SERDEPROPERT
      'separatorChar' = '''',
      'quoteChar' = ''''
    )
    STORED AS `textfile`
```

```
    LOCATION 'cosn://dlc-nj-1258469122/csv/100M/
```

Iceberg 表语法

DDL 语法

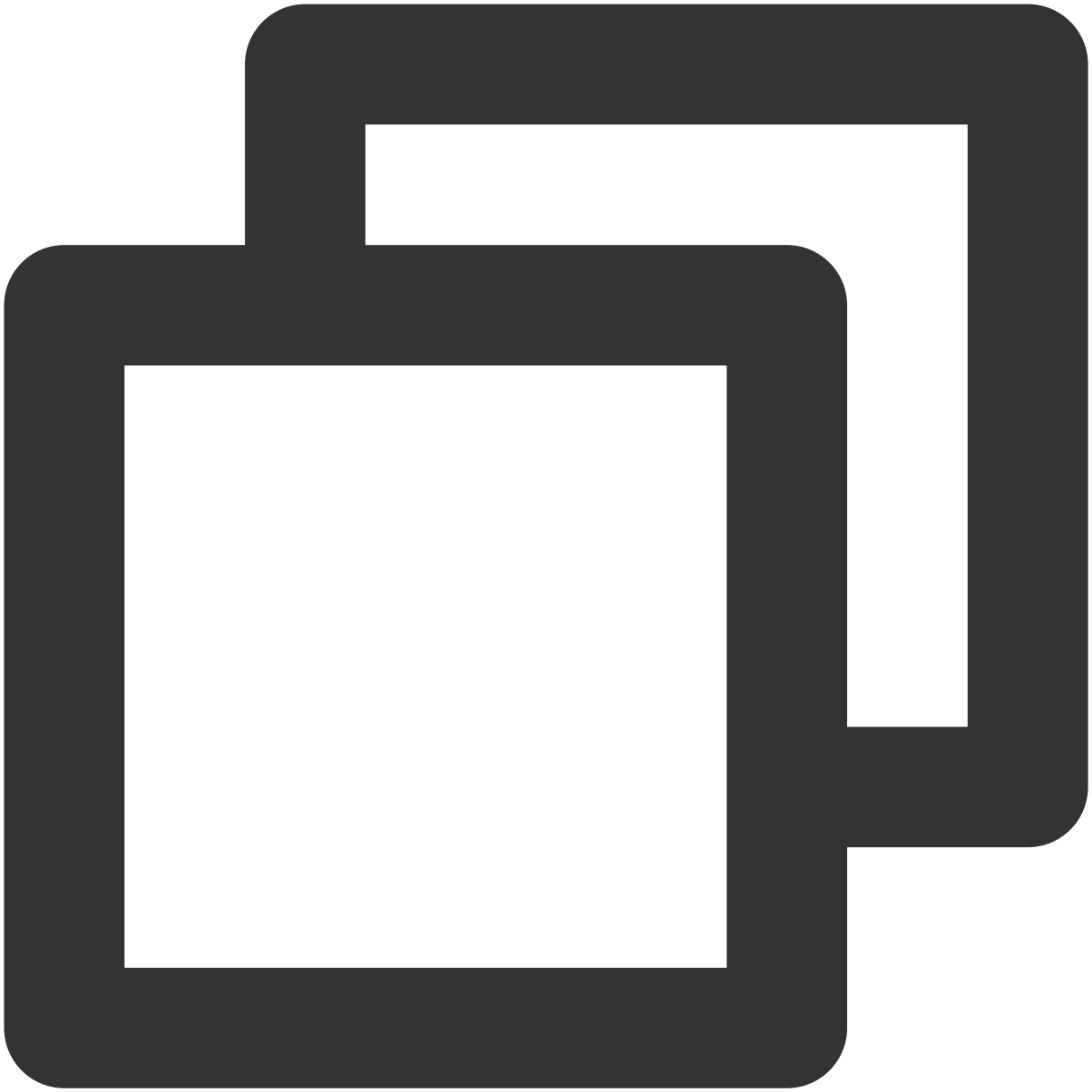
最近更新时间：2024-08-07 17:28:29

说明：

以下语法说明为 DLC 原生表语法，DLC 原生表默认为 Iceberg 表。如果您使用 Iceberg 外部表，在 DDL 语法上会有细节差异，可参考文档 Iceberg 外部表与原生表语法差异。

CREATE TABLE

语法



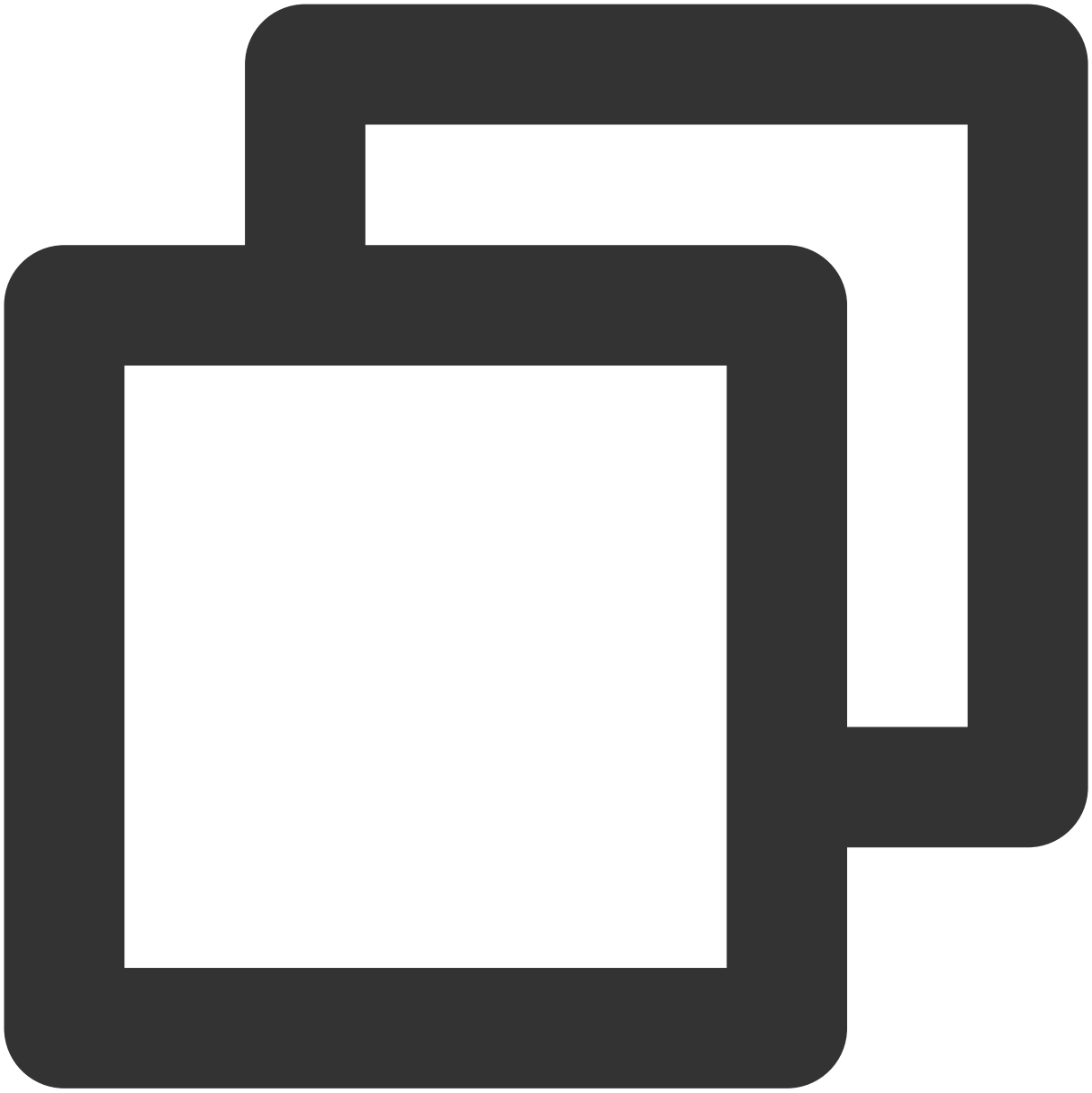
```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
( col_name[:] col_type [ COMMENT col_comment ], ... )
[ COMMENT table_comment ]
[ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
```

参数

table_identifier

`table_identifier` 支持三段式： `catalog.db.name` 。

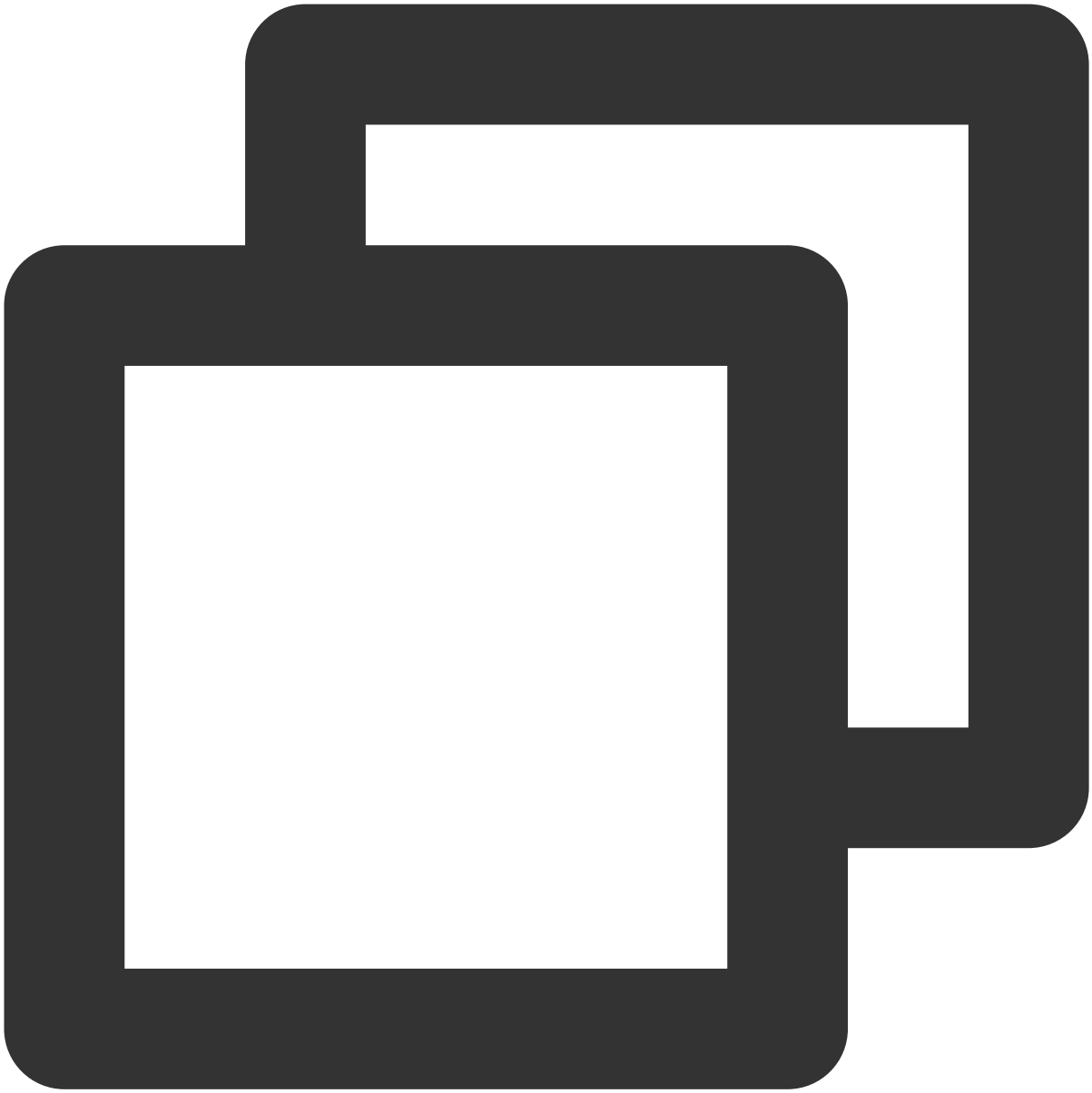
Schemas and Data Types



```
col_type
: primitive_type
  | nested_type

primitive_type
: boolean
| int/integer
| long/bigint
| float
```

```
| double
| decimal(p,s), p=最大位数, s=最大小数点位数, s<=p<=38
| date
| timestamp, timestamp with timezone, 不支持time和without timezone
| string, 也可对应Iceberg uuid类型
| binary, 也可对应Iceberg fixed类型

nested_type
: struct
| list
| map
```

Partition Transforms



```
transform
: identity, 支持任意类型, DLC不支持该转换
| bucket[N], hash mod N分桶, 支持col_type: int,long, decimal, date, timestamp, string
| truncate[L], L截取分桶, 支持col_type: int,long,decimal,string
| years, 年份, 支持col_type: date,timestamp
| months, 月份, 支持col_type: date,timestamp
| days/date, 日期, 支持col_type: date,timestamp
| hours/date_hour, 小时, 支持col_type: timestamp
```

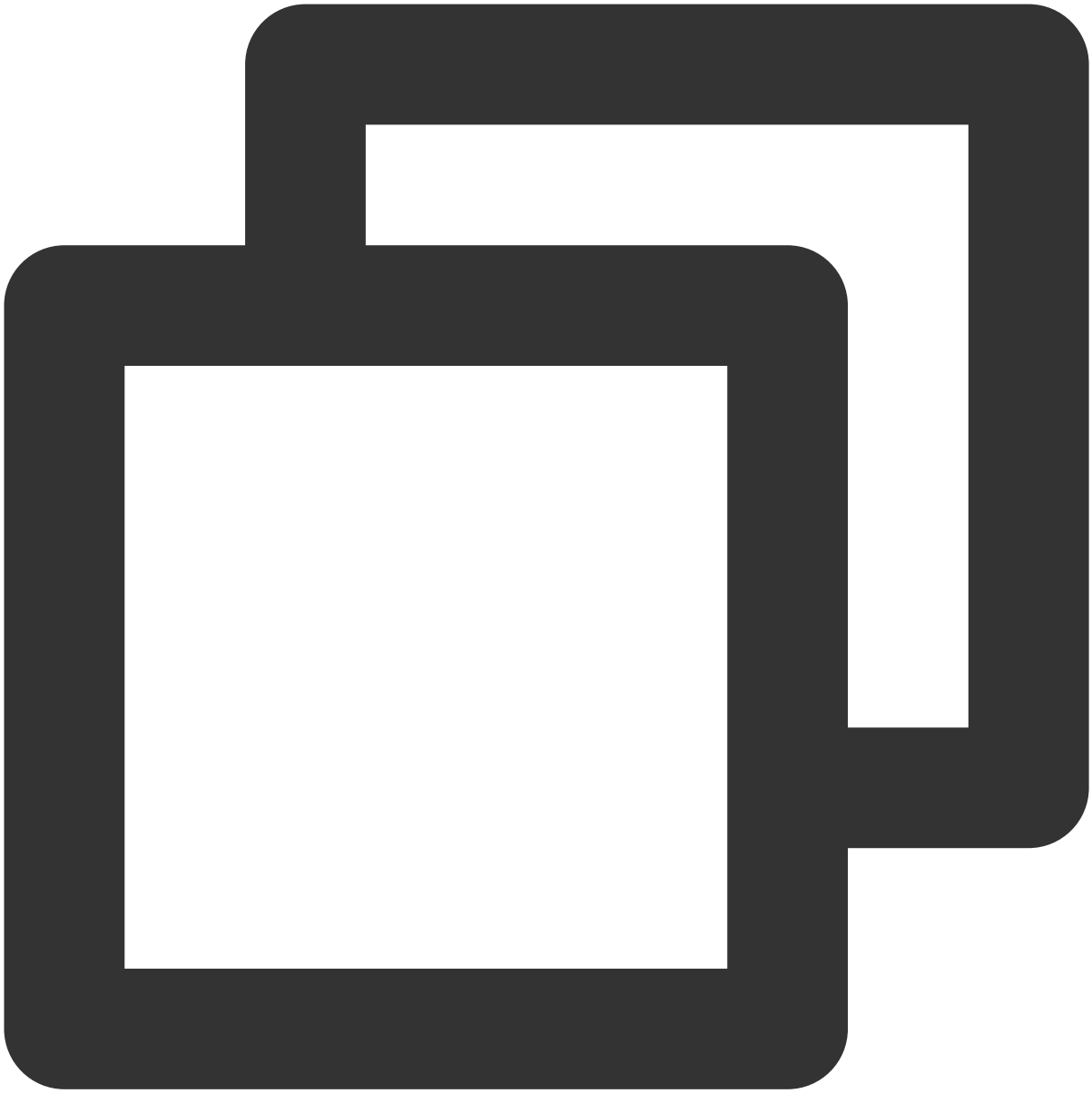
示例

```
CREATE TABLE dempts(
    id bigint COMMENT 'id number',
    num int,
    eno float,
    dno double,
    cno decimal(9,3),
    flag  boolean,
    data string,
    ts_year timestamp,
    date_month date,
    bno binary,
```

```
    point struct<x: double, y: double>,
    points array<struct<x: double, y: double>>,
    pointmaps map<struct<x: int>, struct<a: int>>
)
COMMENT 'table documentation'
PARTITIONED BY (bucket(16,id), years(ts_year), months(date_month), identity(bno), b
```

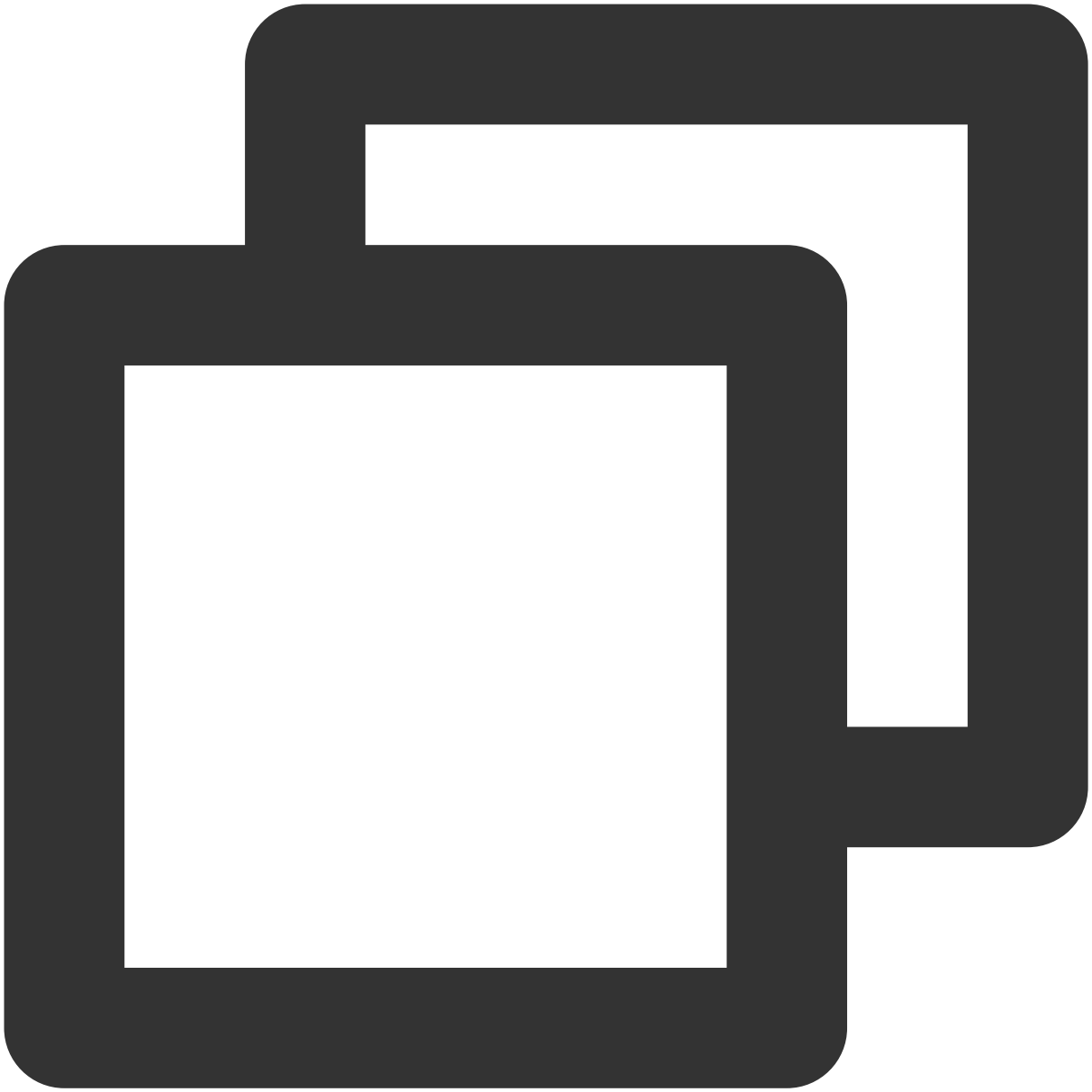
CREATE TABLE AS SELECT

语法



```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
    [ TBLPROPERTIES ( property_name=property_value, ... ) ]
AS select_statement
```

示例

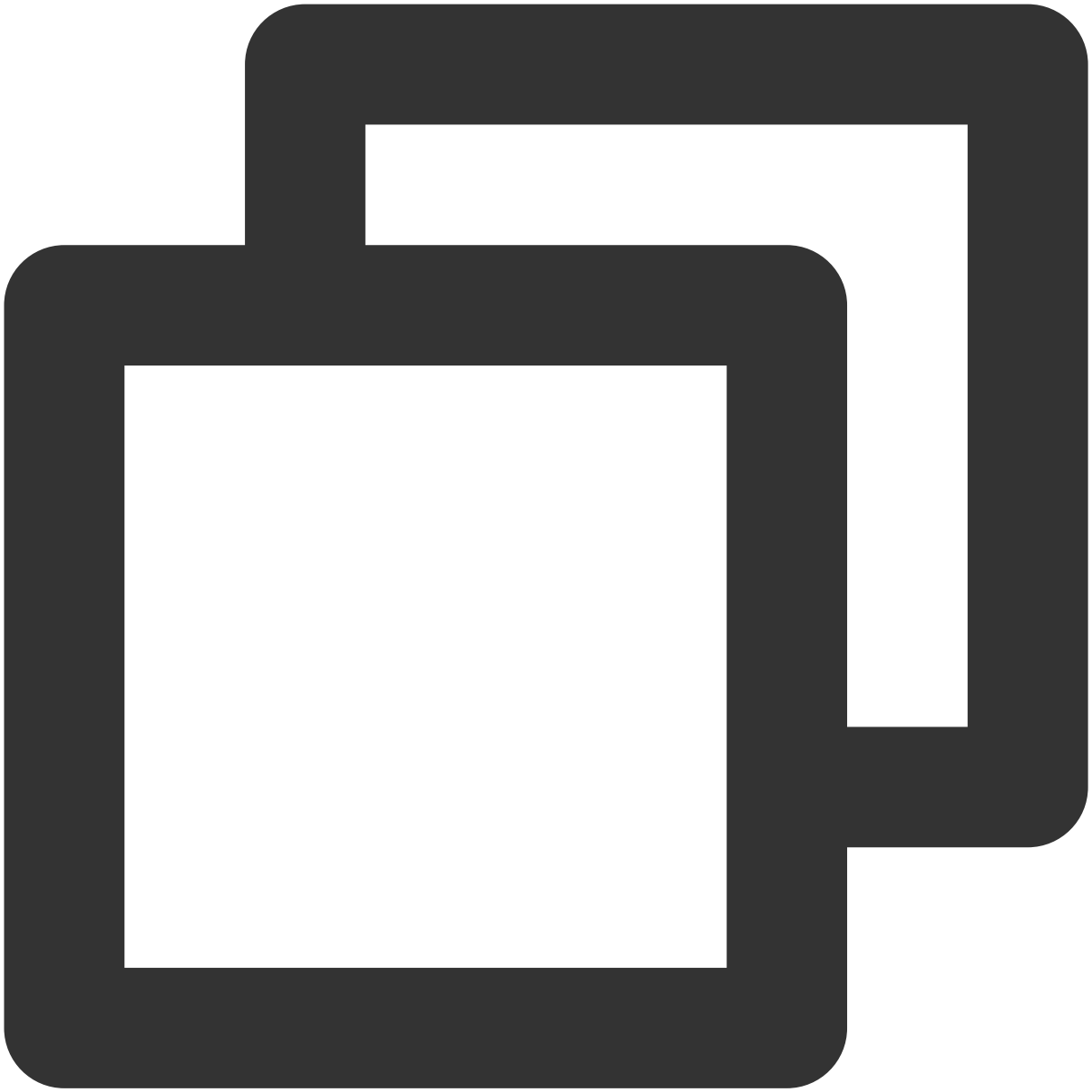


```
CREATE TABLE dempts_copy
COMMENT 'table create as select'
PARTITIONED BY (eno, dno)
AS SELECT * from dempts;
```

REPLACE TABLE AS SELECT

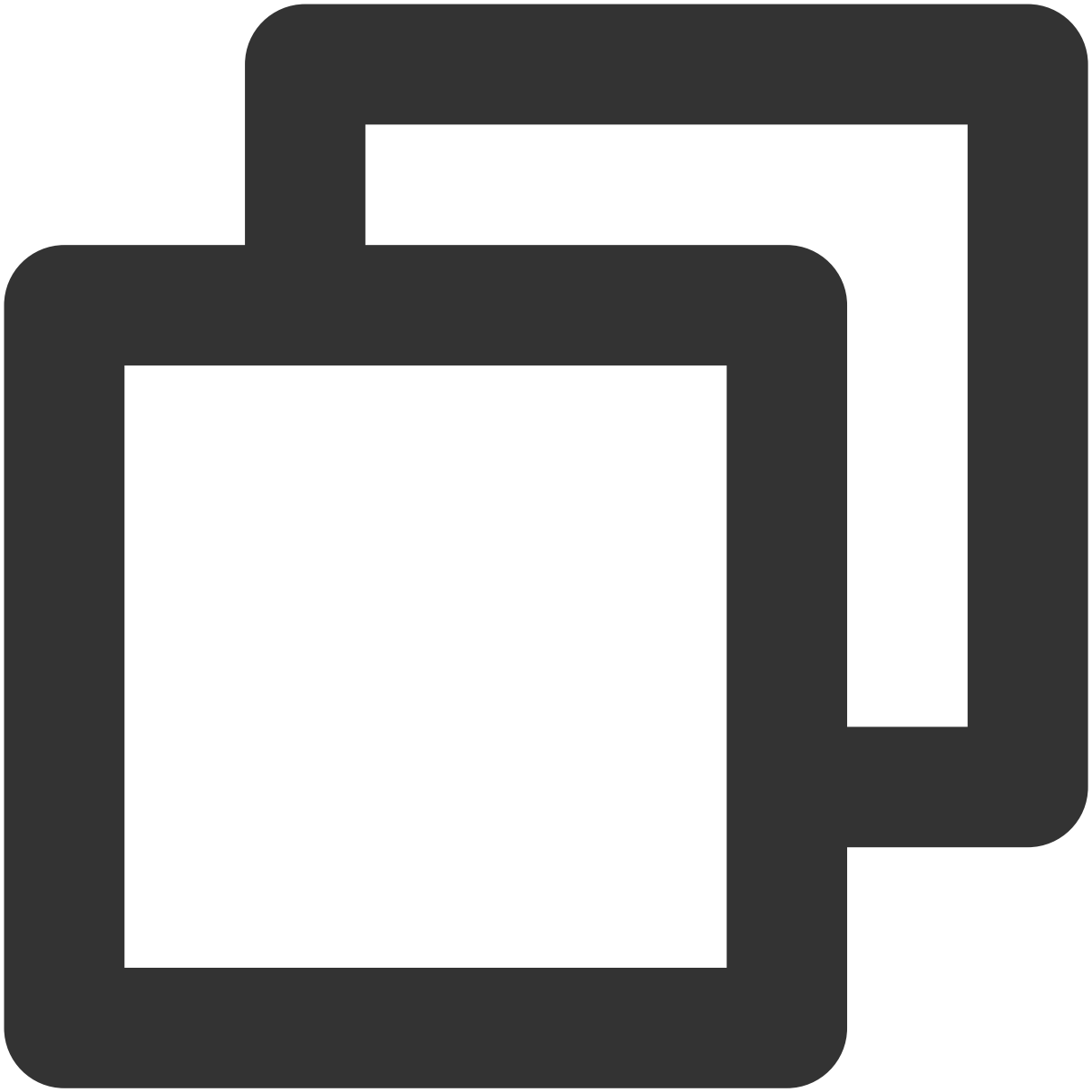
保留表历史 History 的情况下，用 SELECT 查询的结果生成一个快照 Snapshot 来更新表。

语法



```
CREATE [OR REPLACE] TABLE table_identifier
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
AS select_statement
```

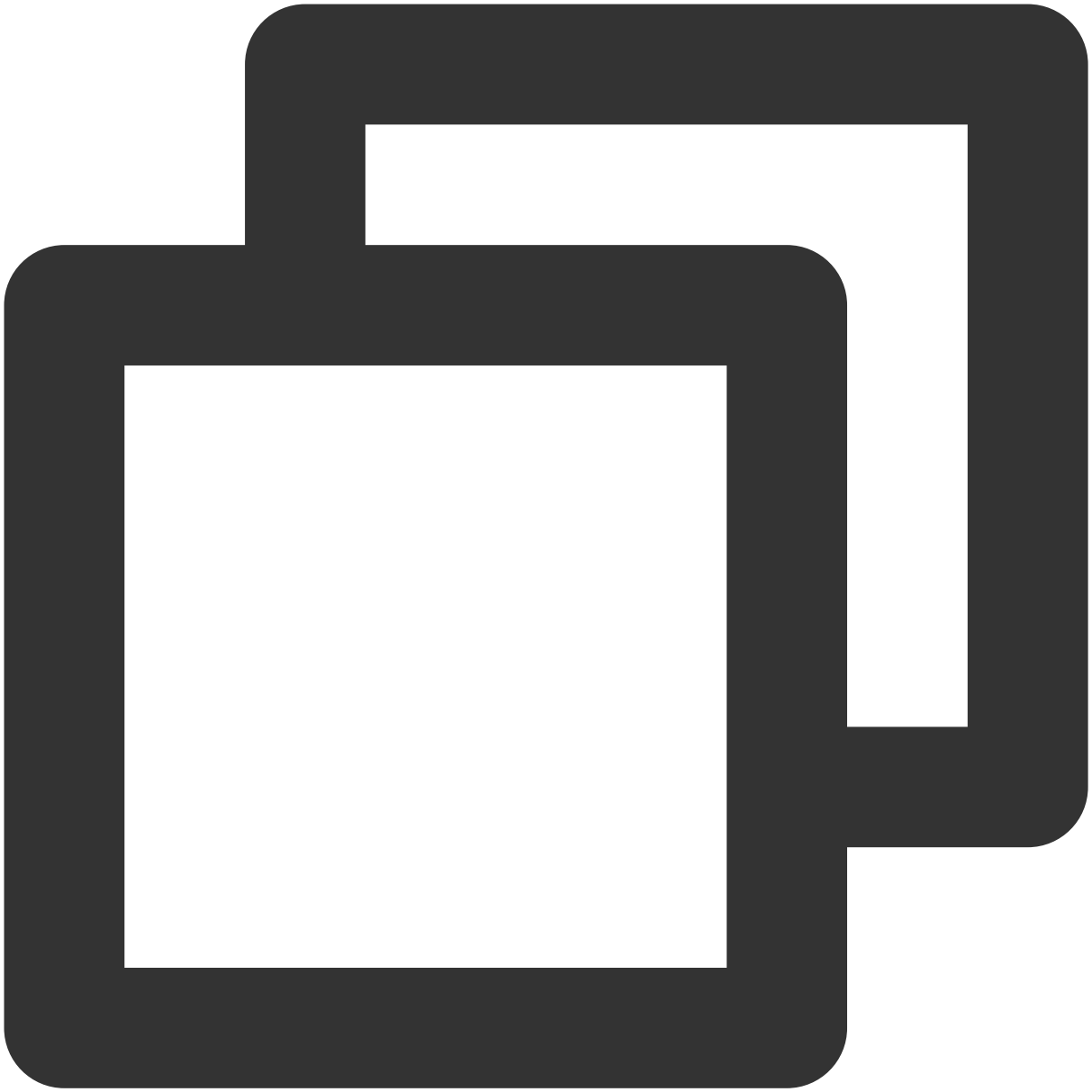
示例



```
CREATE OR REPLACE TABLE dempts_replace
    COMMENT 'table create as replace'
    PARTITIONED BY (eno, dno)
AS SELECT * from dempts;
```

DROP TABLE

语法



```
DROP TABLE [ IF EXISTS ] table_identifier
```

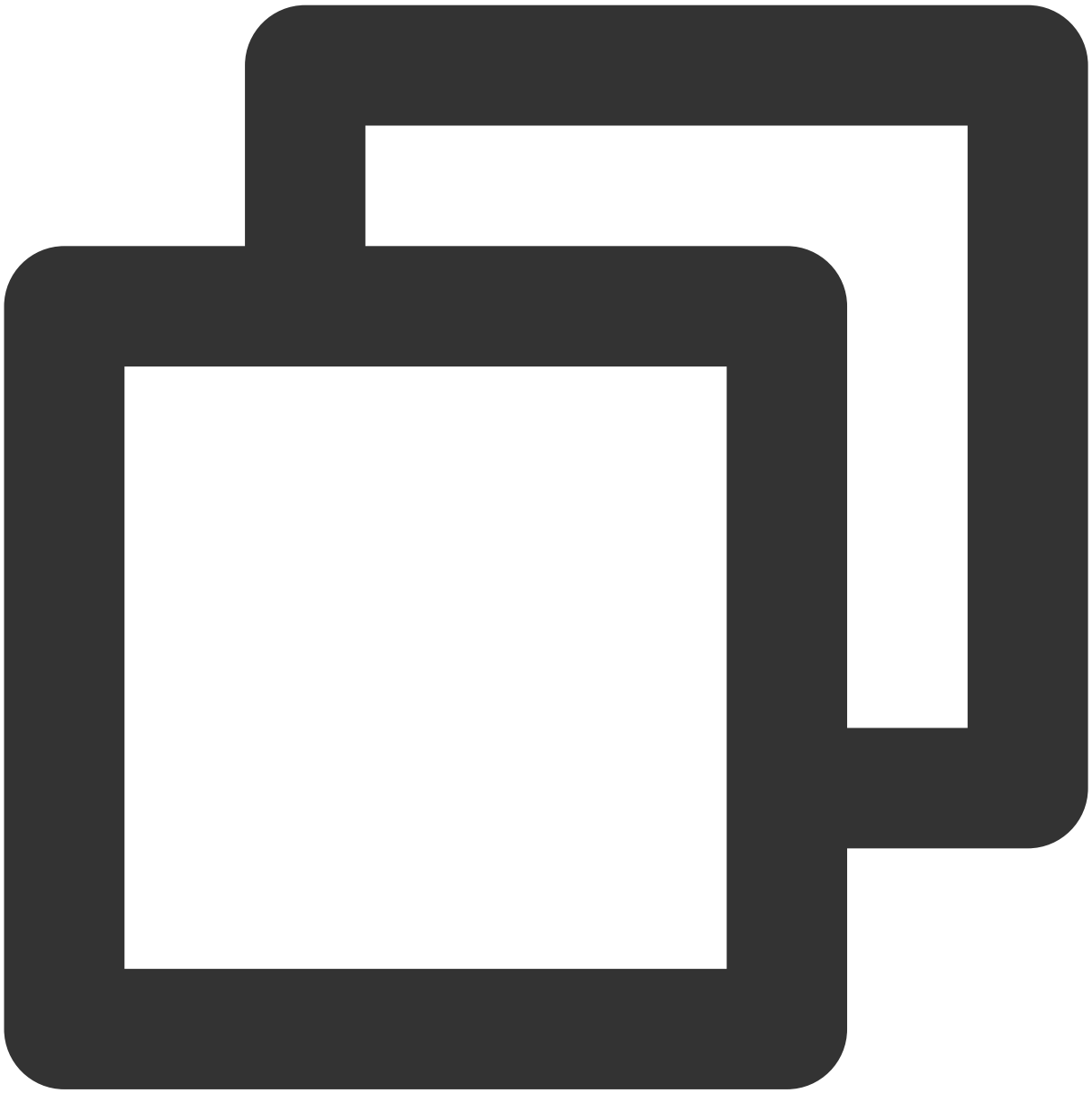
ALTER TABLE

变更表语法

ALTER TABLE ... RENAME TO

变更表名称

语法

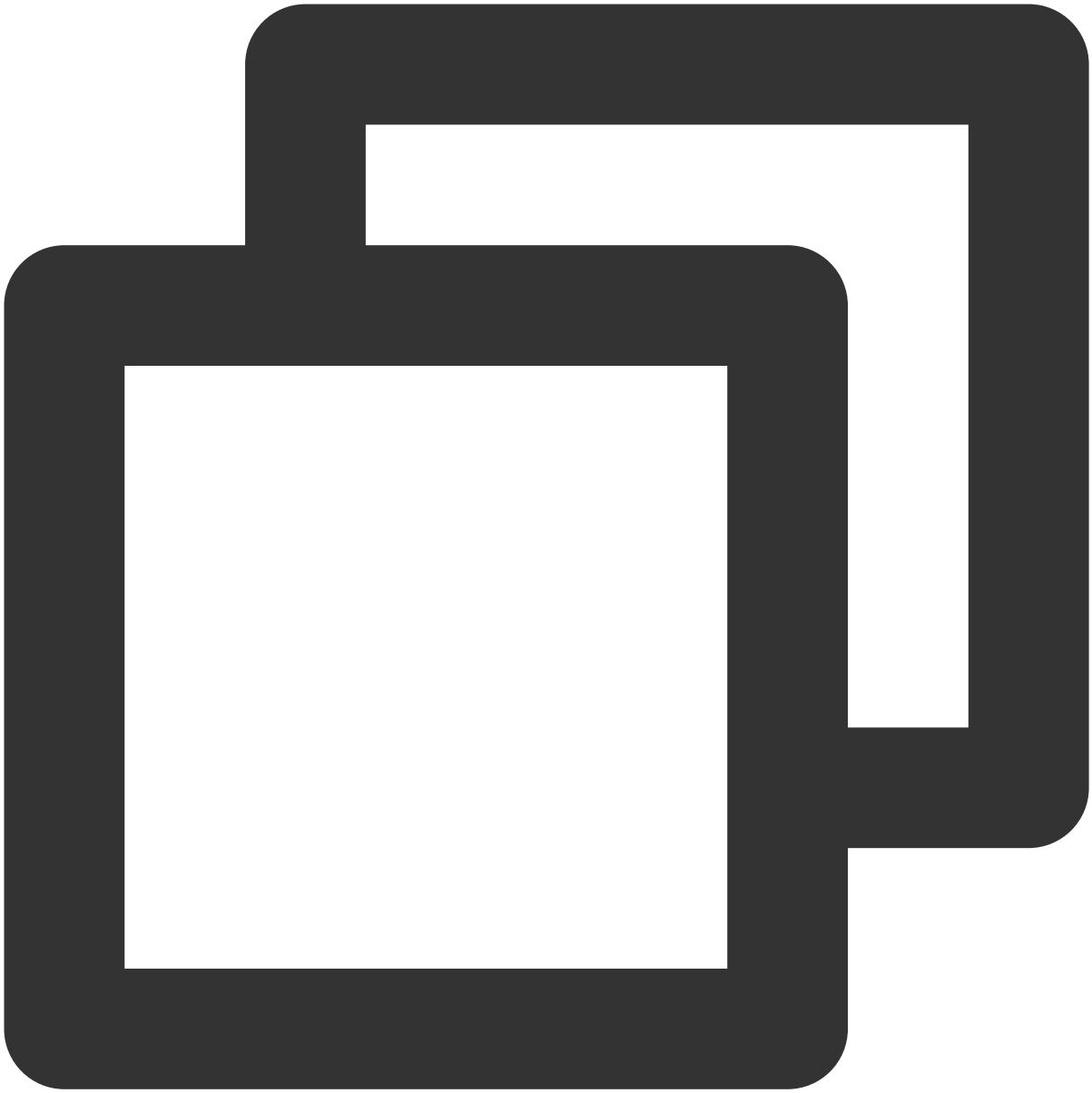


```
ALTER TABLE table_identifier RENAME TO new_table_identifier
```

ALTER TABLE ... SET / UNSET TBLPROPERTIES

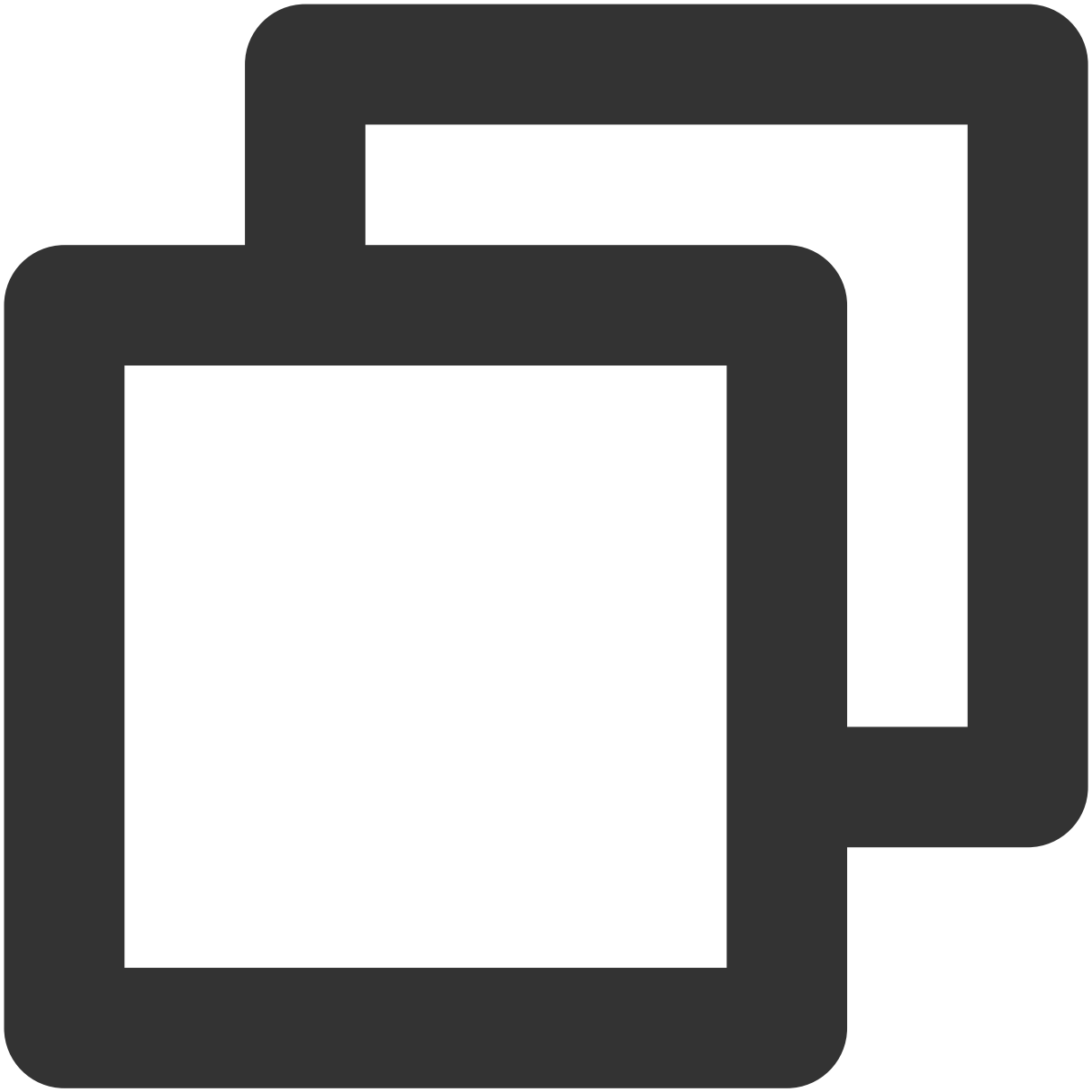
更新/删除数据表属性

语法



```
-- SET 更新属性配置
ALTER TABLE table_identifier
SET TBLPROPERTIES (property_name=property_value, ...)

-- UNSET 删除属性配置
ALTER TABLE table_identifier
UNSET TBLPROPERTIES (property_name, ...)
```

示例

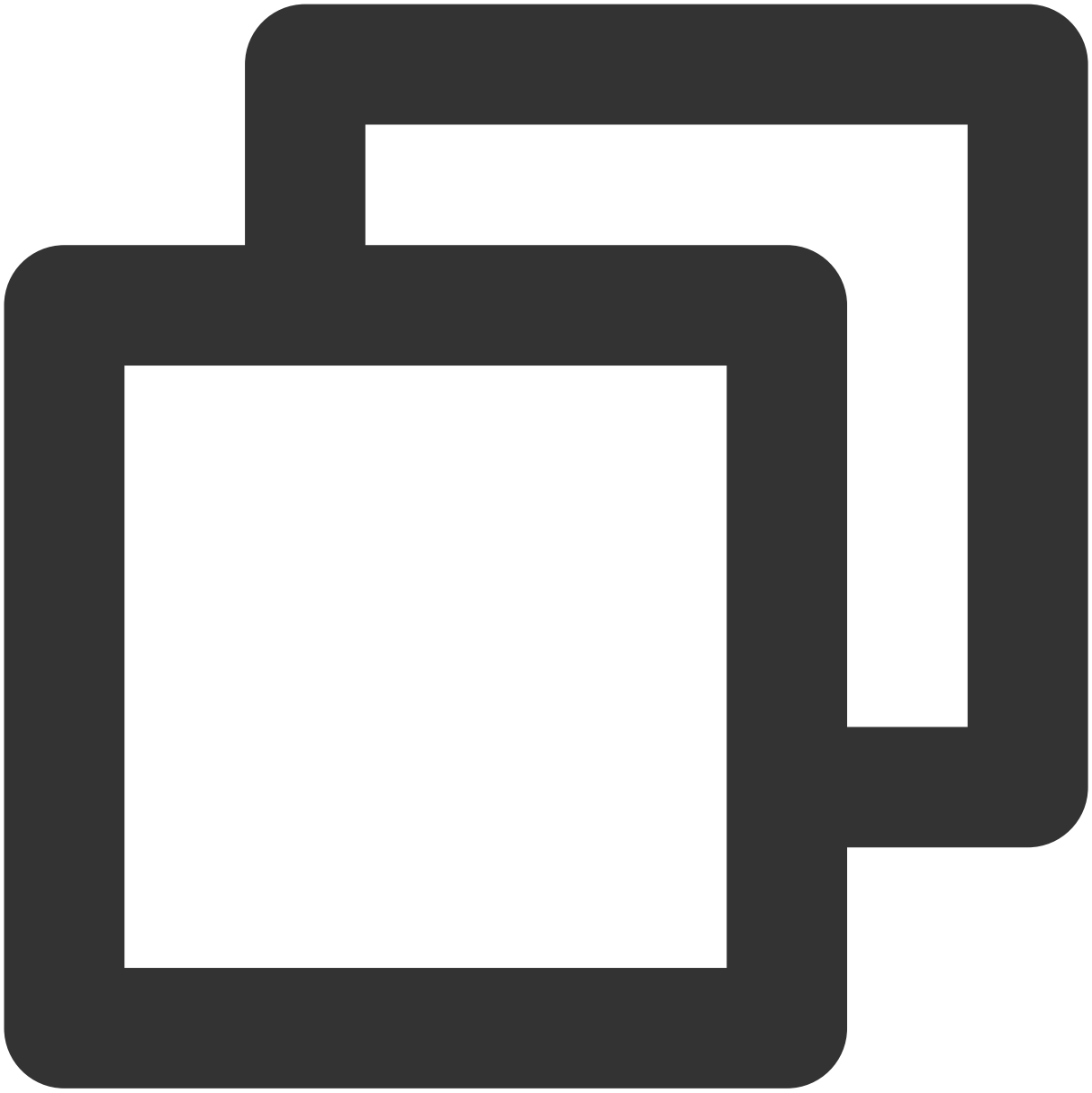


```
-- SET 更新属性配置
ALTER TABLE dempts SET TBLPROPERTIES ('read.split.target-size'='268435456');

-- UNSET 删除属性配置
ALTER TABLE dempts UNSET TBLPROPERTIES ('read.split.target-size'='268435456');
```

ALTER TABLE ... WRITE ORDERED BY

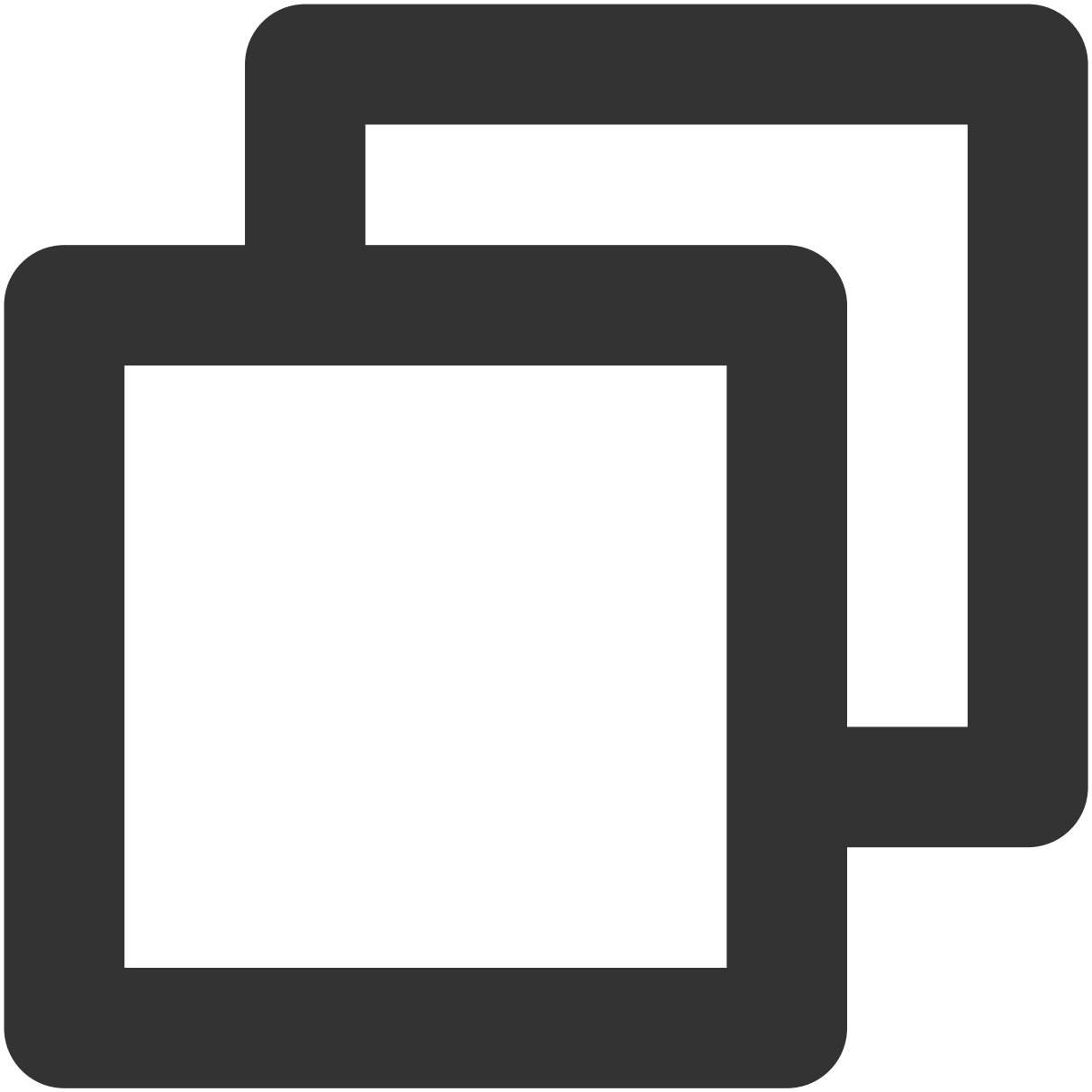
设置表数据插入时的排序方式

语法



```
ALTER TABLE table_identifier
WRITE [LOCALLY] ORDERED BY
{col_name [ASC|DESC] [NULLS FIRST|LAST]}[, ...]
```

示例

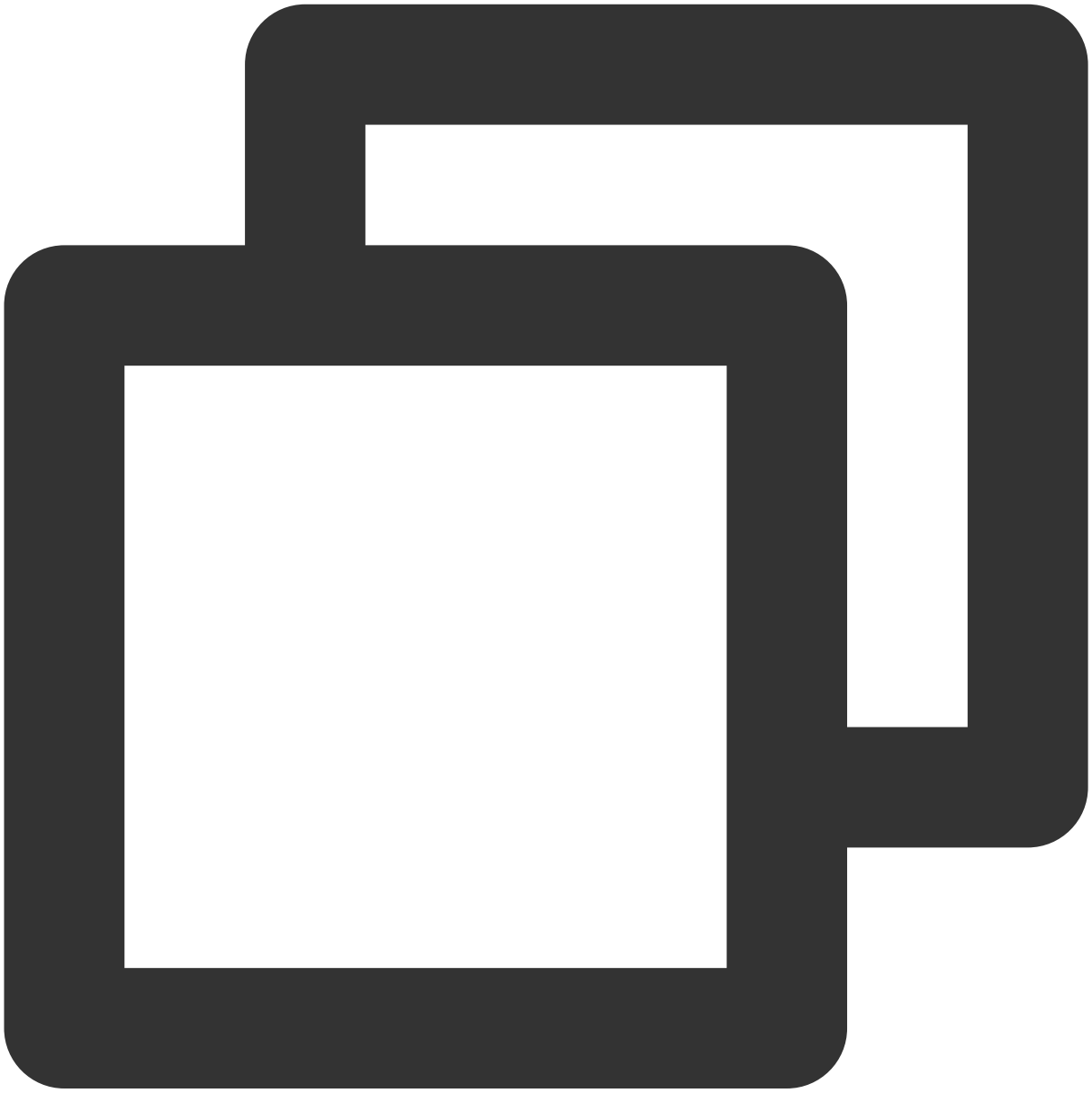


```
ALTER TABLE dempts WRITE ORDERED BY category, id;

-- use optional ASC/DEC keyword to specify sort order of each field (default ASC)
ALTER TABLE dempts WRITE ORDERED BY category ASC, id DESC;

-- use optional NULLS FIRST/NULLS LAST keyword to specify null order of each field
ALTER TABLE dempts WRITE ORDERED BY category ASC NULLS LAST, id DESC NULLS FIRST;

-- To order within each task, not across tasks
ALTER TABLE dempts WRITE LOCALLY ORDERED BY category, id;
```

ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION

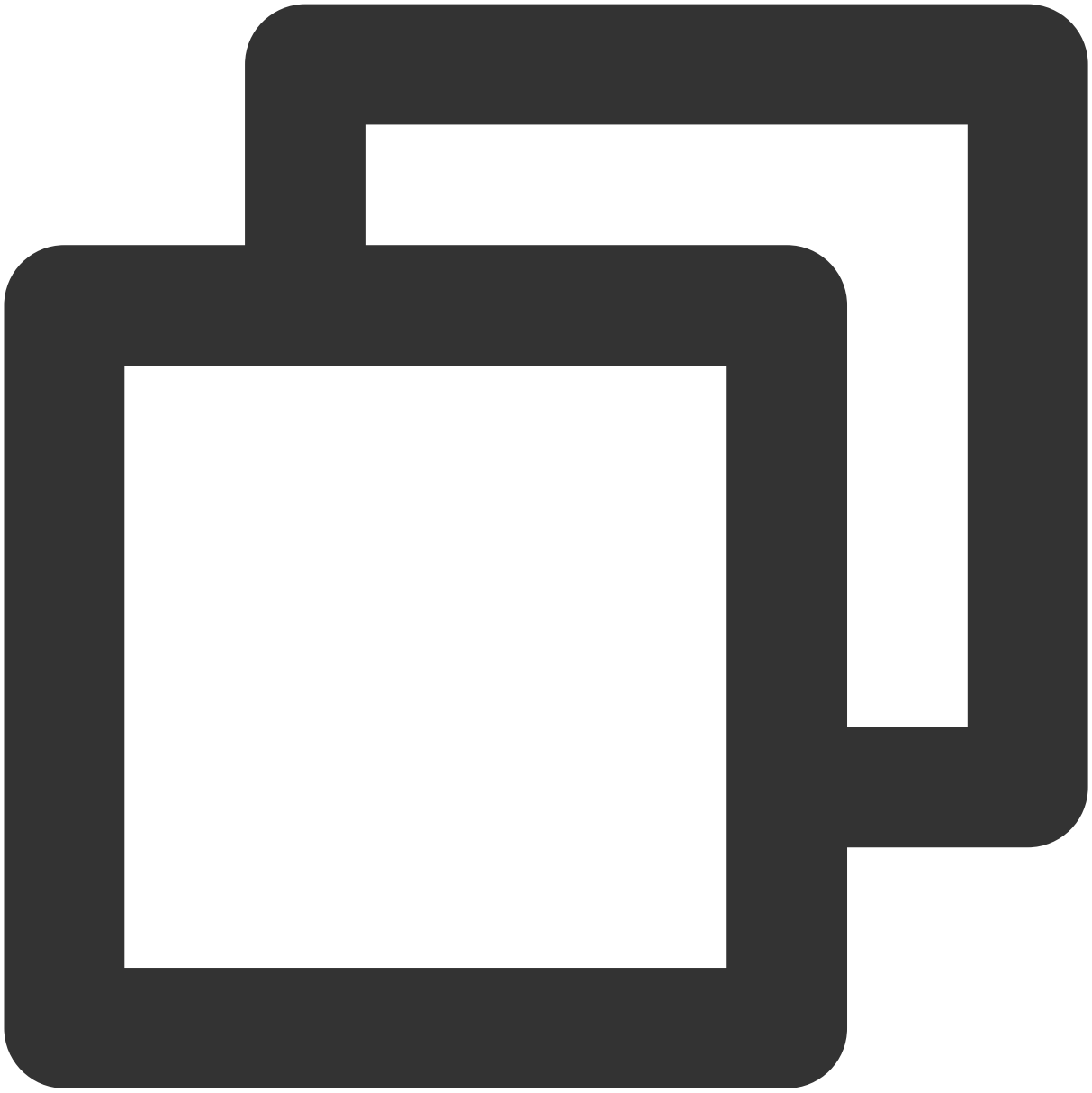
修改分区表的数据分配策略

语法



```
ALTER TABLE table_identifier
WRITE DISTRIBUTED BY PARTITION
[ LOCALLY ORDERED BY
{col_name [ASC|DESC] [NULLS FIRST|LAST]}[, ...]]
```

示例



```
ALTER TABLE dempts WRITE DISTRIBUTED BY PARTITION;
ALTER TABLE dempts WRITE DISTRIBUTED BY PARTITION LOCALLY ORDERED BY id;
```

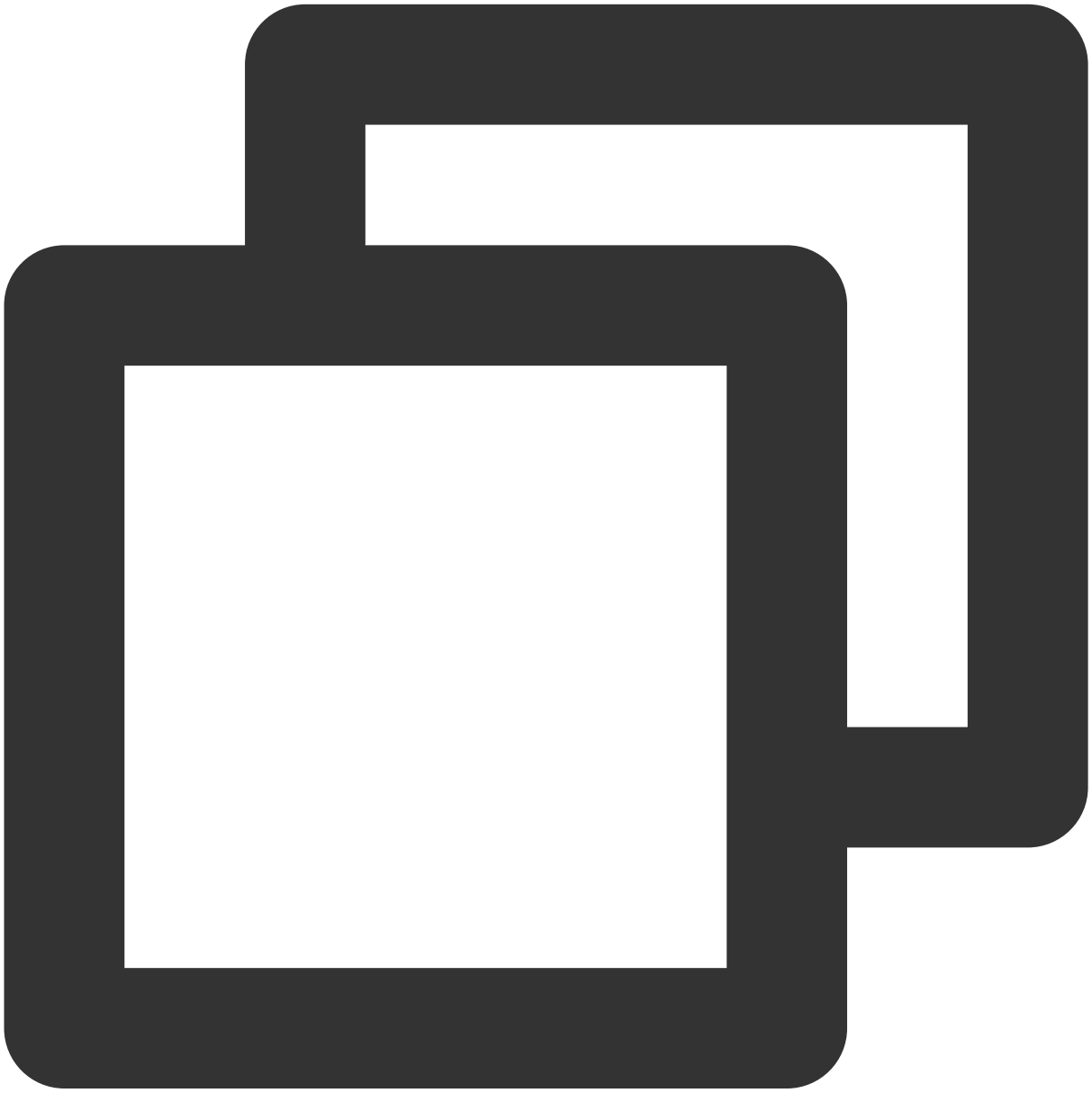
ALTER TABLE COLUMNS

变更字段语法

ALTER TABLE ... ADD COLUMNS

新增多个字段

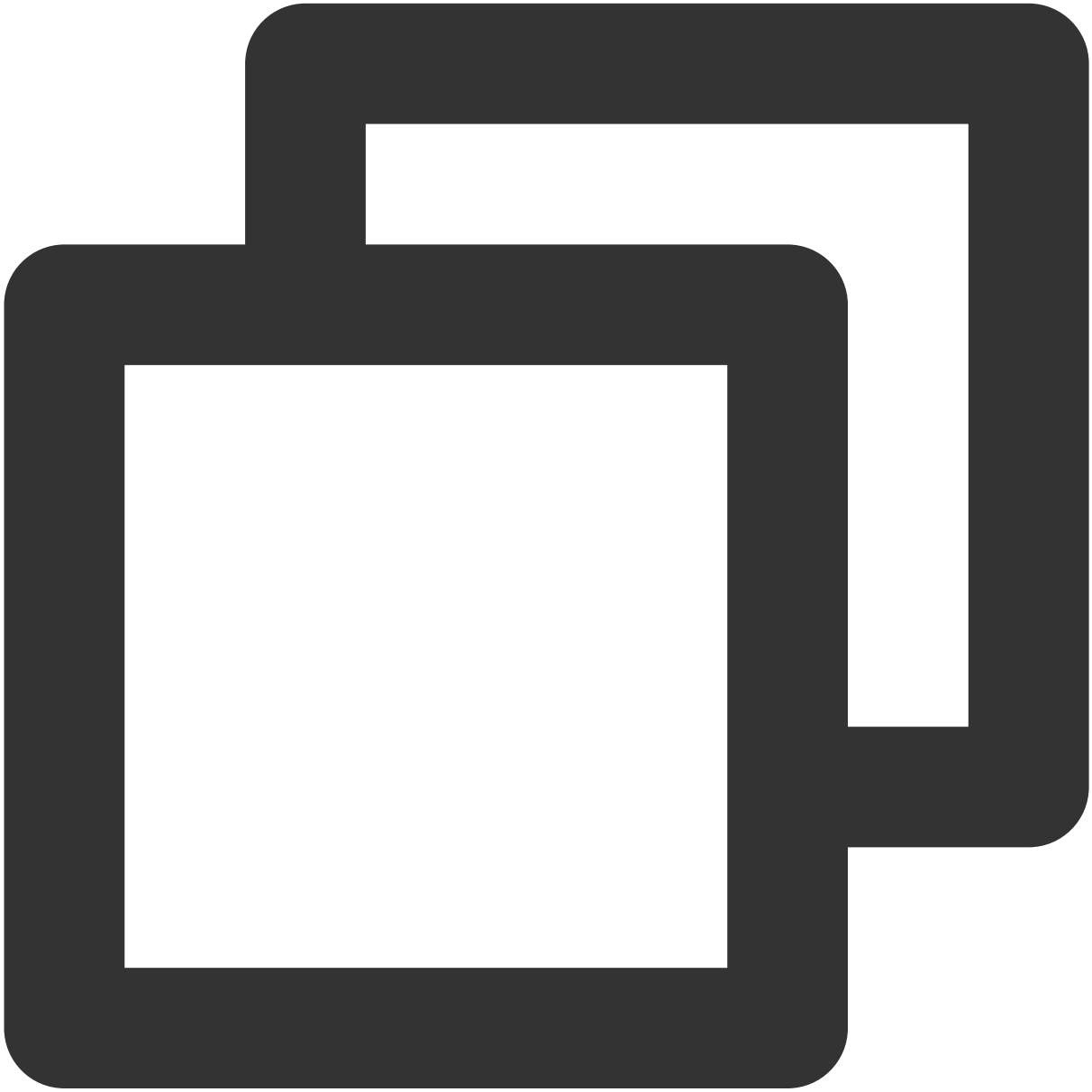
语法



```
-- 新增多个字段
ALTER TABLE table_identifier
ADD COLUMNS (col_name col_type [COMMENT col_comment], ...)

-- 新增单个字段
```

```
ALTER TABLE table_identifier
ADD COLUMN col_name col_type [COMMENT col_comment]
[FIRST | AFTER target_col_name]
```

示例



```
-- 新增多个字段
ALTER TABLE dempts
ADD COLUMNS (
    new_column_1 string comment 'new_column_1 docs',
    new_column_2 int comment 'new_column_2 docs'
```

```
);

-- 新增单个字段
ALTER TABLE dempts
ADD COLUMN new_column_3 string comment 'new_column docs';
```

ALTER TABLE ... RENAME COLUMN

变更字段名称

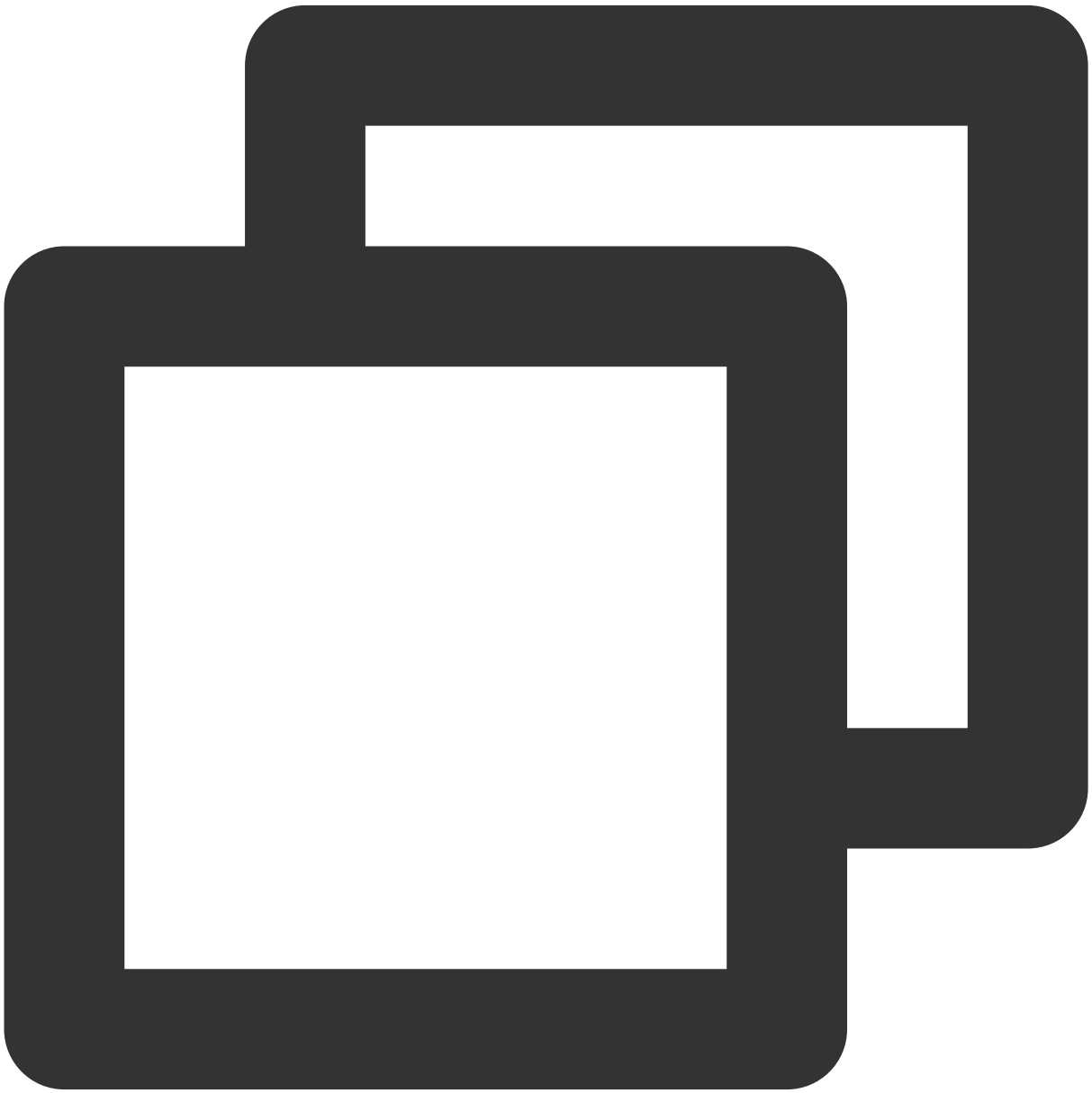
语法

```
ALTER TABLE table_identifier
RENAME COLUMN old_column_name TO new_column_name
```

ALTER TABLE ... ALTER COLUMN

变更字段类型/备注信息

语法



```
ALTER TABLE table_identifier
```

```
ALTER COLUMN col_name
{TYPE new_col_type | COMMENT col_comment}
```

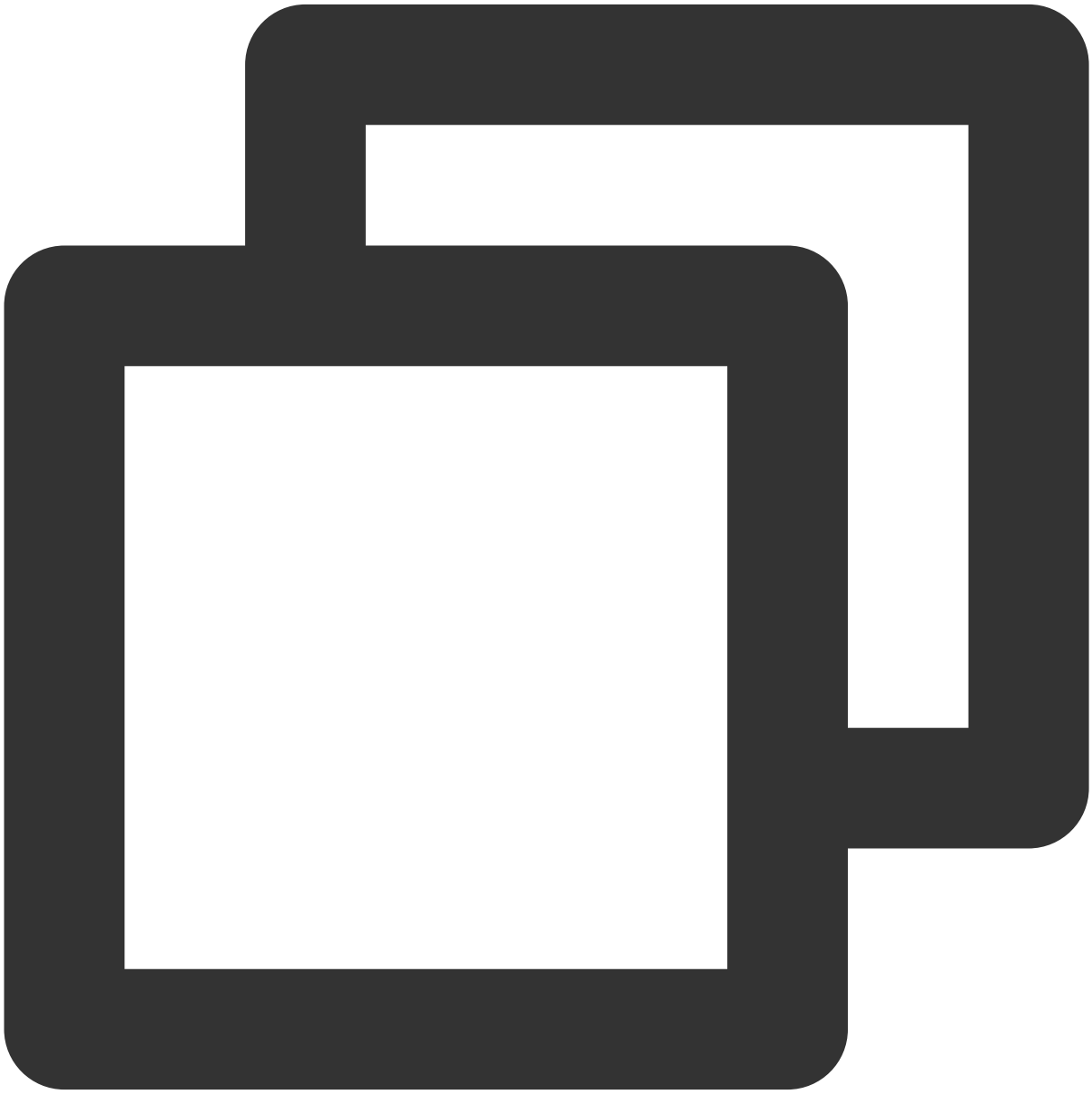
目前Iceberg TYPE 变更，仅支持字段类型安全扩展：

int/integer -> long/bigint

float -> double

decimal(P,S) -> decimal(P2,S)，其中P2 > P，即精准度提升

示例



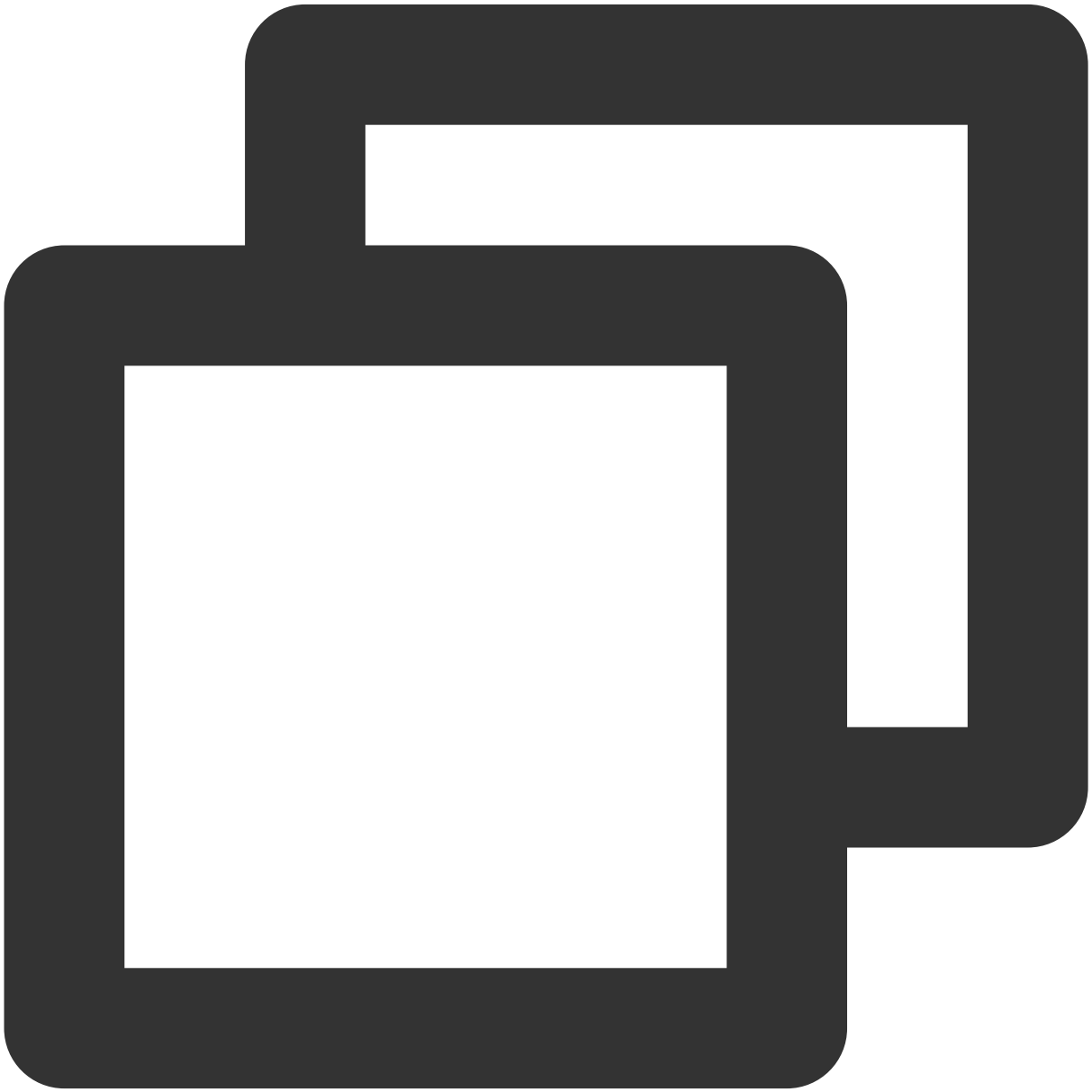
```
ALTER TABLE dempts ALTER COLUMN new_column_2 TYPE bigint;
```

```
ALTER TABLE dempts ALTER COLUMN new_column_2 comment 'alter docs';
```

ALTER TABLE ... DROP COLUMN

删除表字段

语法



```
ALTER TABLE table_identifier  DROP COLUMN column_name
```

ALTER TABLE PARTITIONS

ALTER TABLE ... ADD PARTITION FIELD

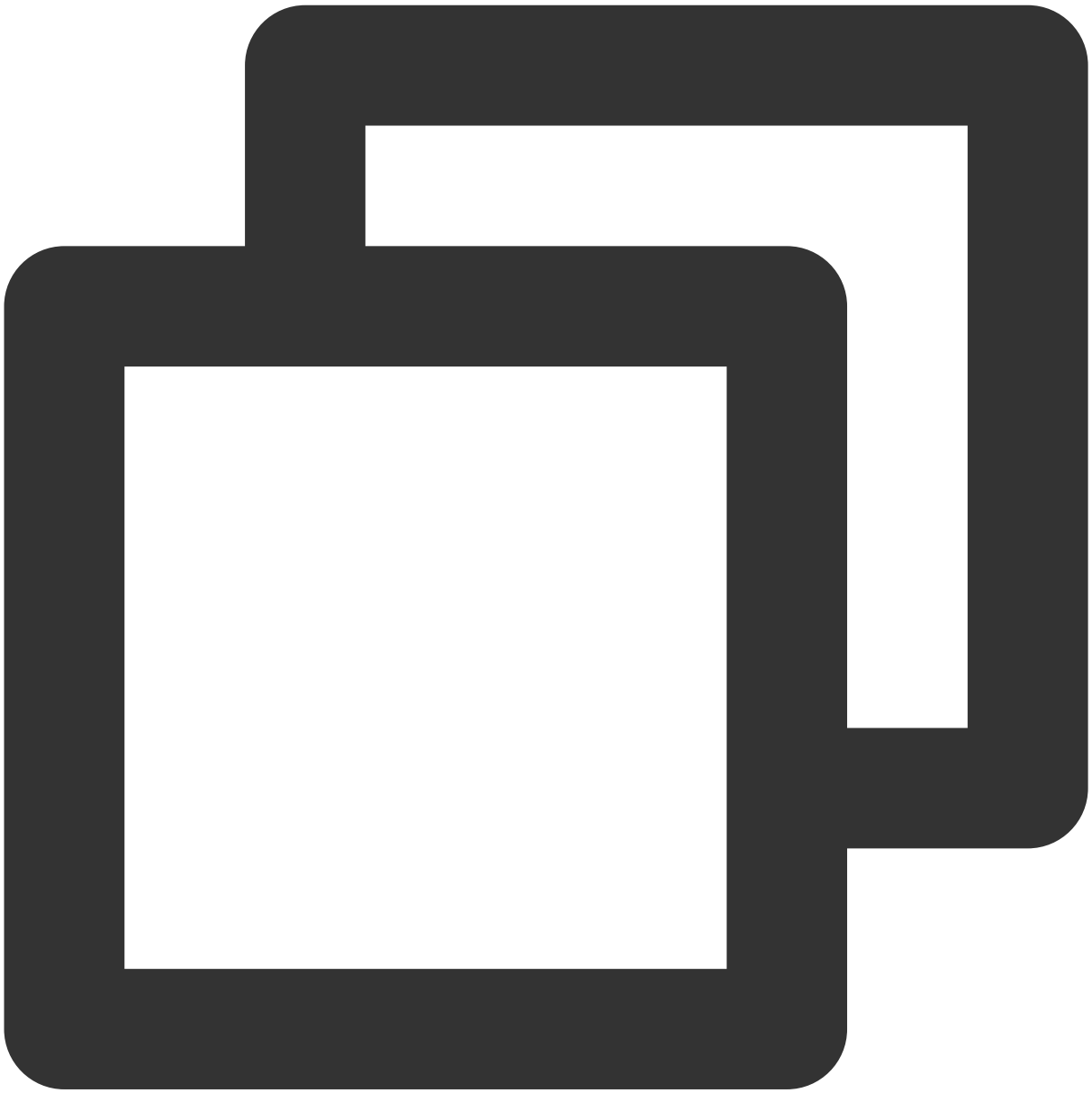
新增单个分区字段

语法

```
ALTER TABLE table_identifier
ADD PARTITION FIELD col_name|transform (col_name) [AS alias]
```

transform 参考 CREATE TABLE 说明。

示例

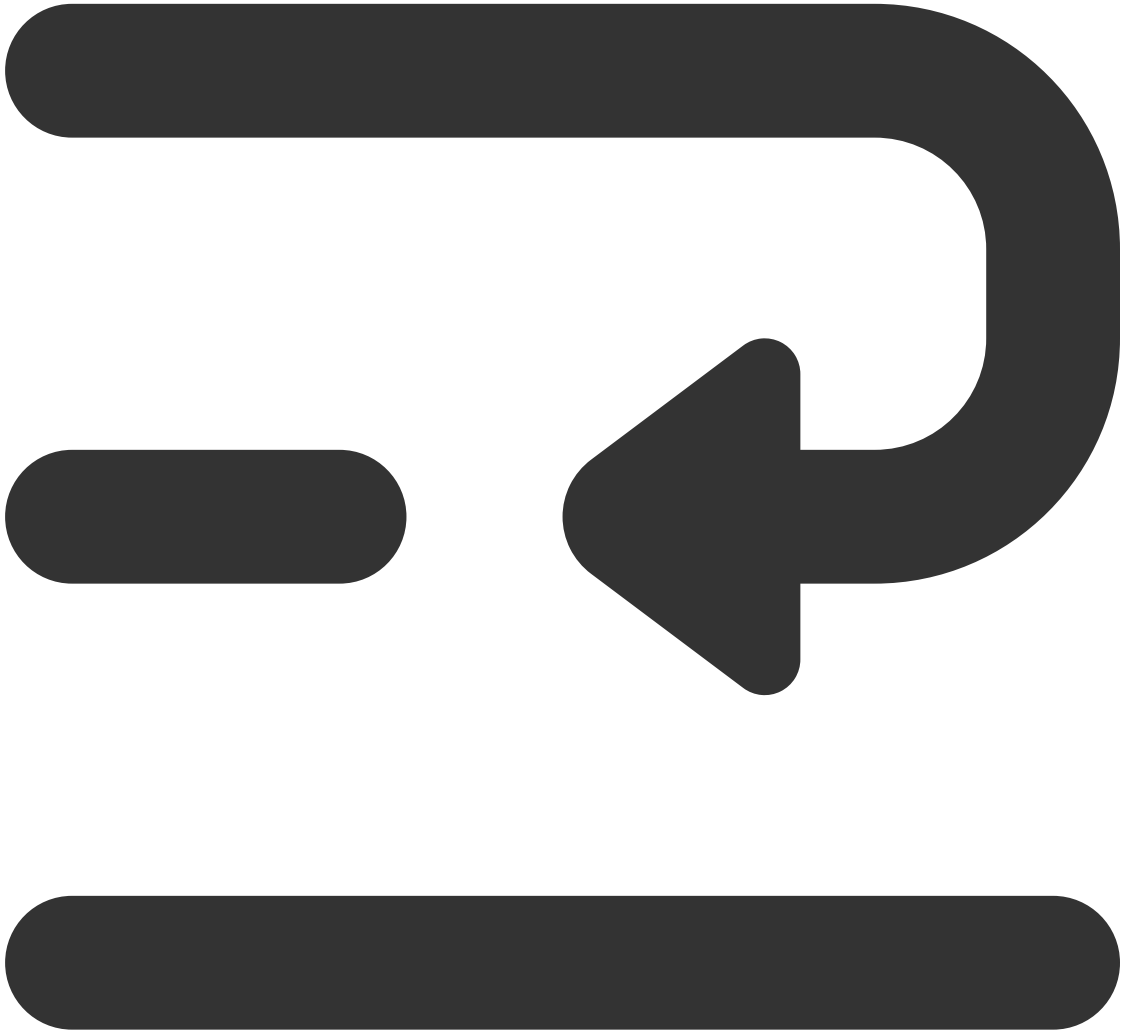


```
ALTER TABLE dempts ADD PARTITION FIELD new_column_1;
ALTER TABLE dempts ADD PARTITION FIELD bucket(3,new_column_2);
```

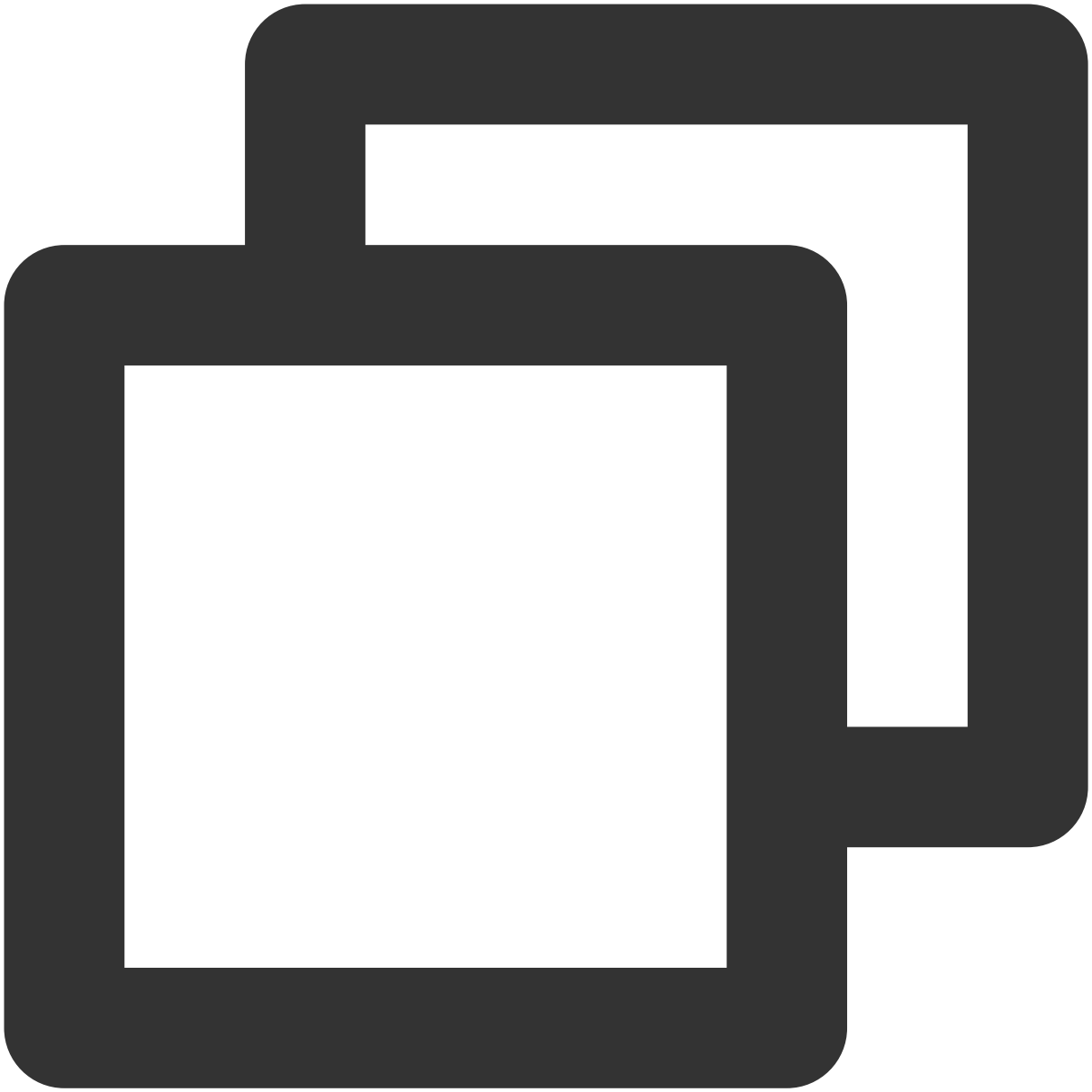
ALTER TABLE ... REPLACE PARTITION FIELD

替换单个分区字段

语法



版权所有：腾讯云计算（北京）有限责任公司

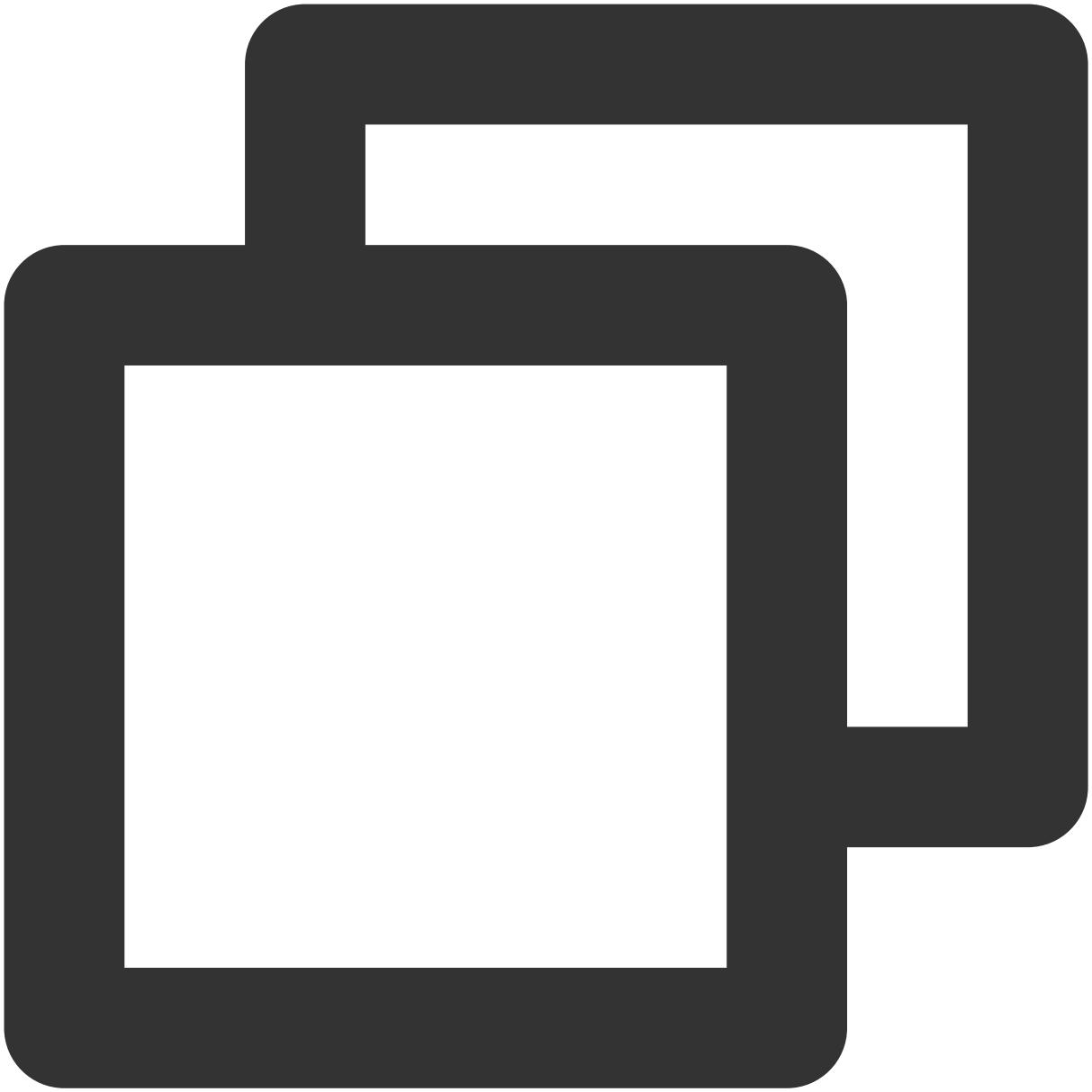


```
ALTER TABLE table_identifier REPLACE PARTITION FIELD col_name|transform (col_name)
```

ALTER TABLE ... DROP PARTITION FIELD

删除单个分区字段

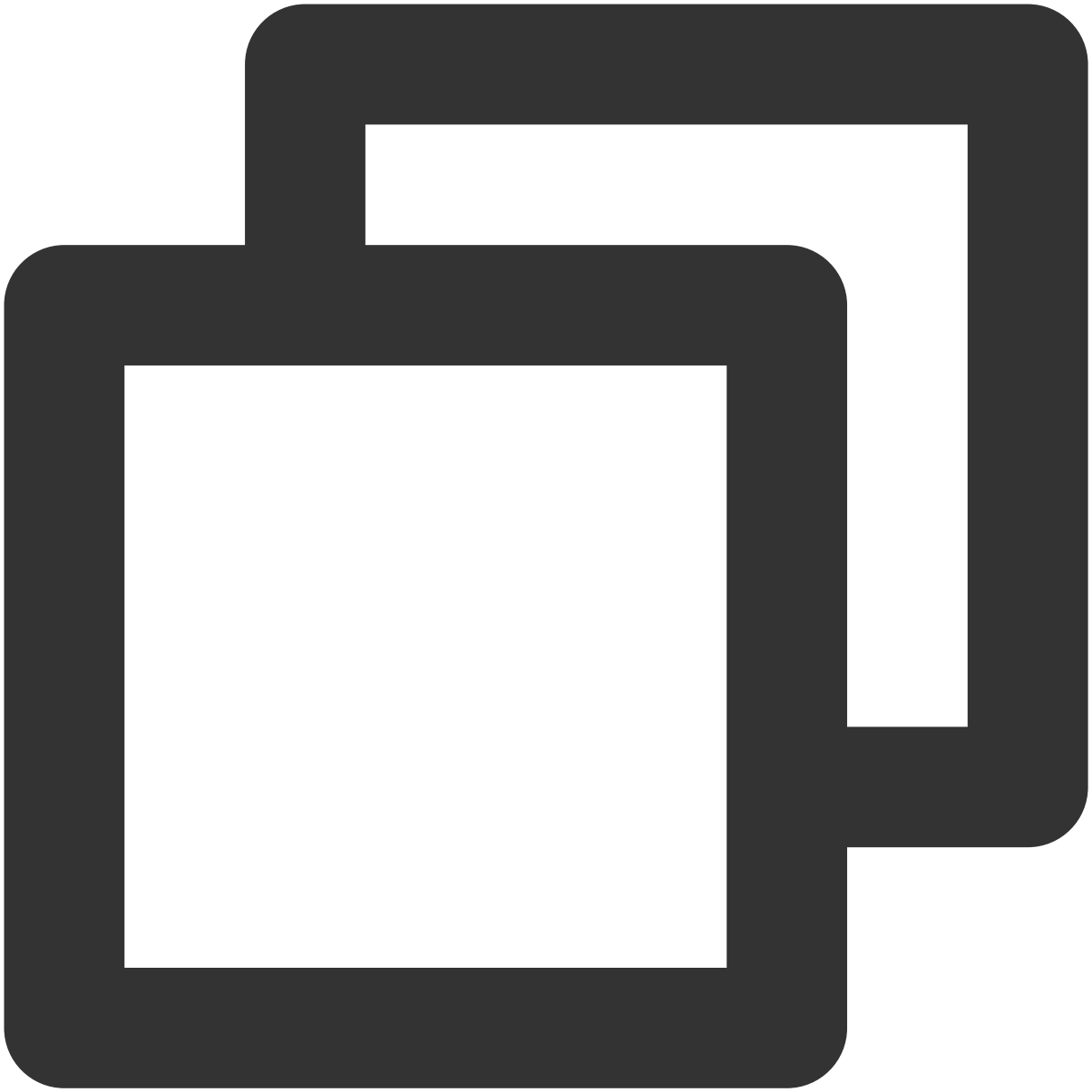
语法



```
ALTER TABLE table_identifier
DROP PARTITION FIELD col_name|transform (col_name);
```

transform 参考 CREATE TABLE 说明

示例

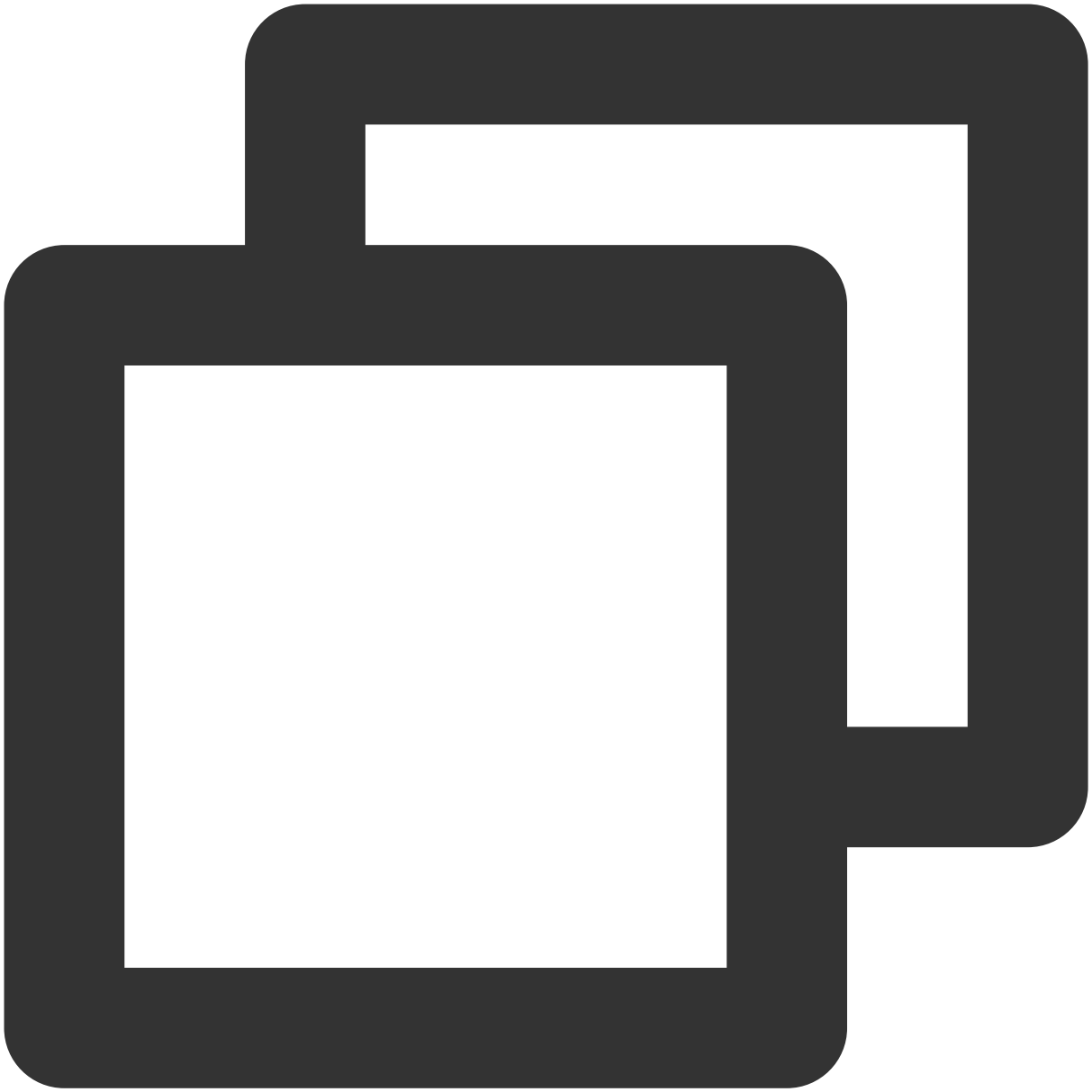


```
ALTER TABLE dempts DROP PARTITION FIELD new_column_1;
ALTER TABLE dempts DROP PARTITION FIELD bucket(3,new_column_2);
```

ALTER TABLE ... SET IDENTIFIER FIELDS

添加 identifier fields 属性

语法

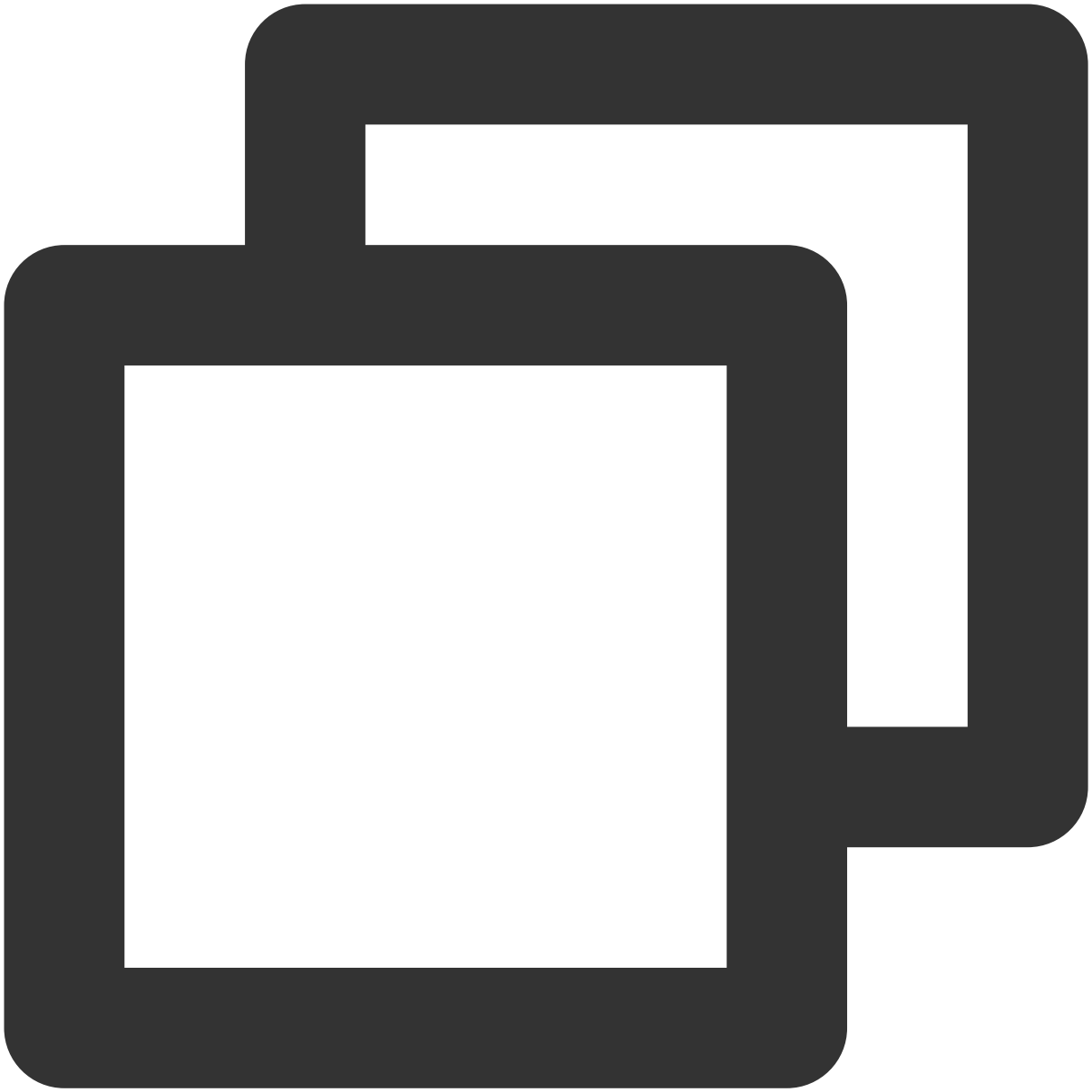


```
ALTER TABLE dempts SET IDENTIFIER FIELDS empno, name
```

ALTER TABLE .. DROP IDENTIFIER FIELDS

添加 identifier fields 属性

语法



```
ALTER TABLE dempts DROP IDENTIFIER FIELDS empno, name
```

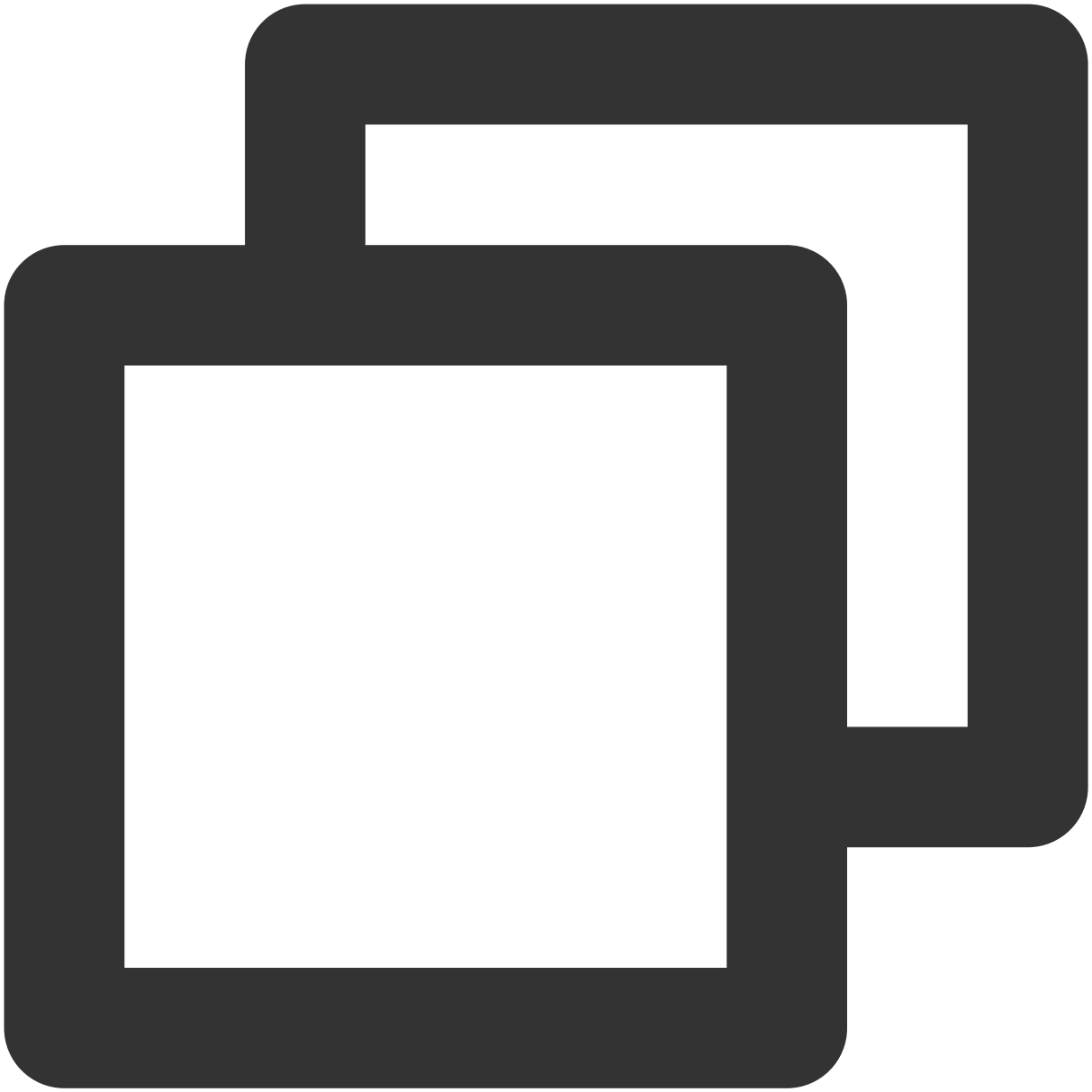
DML 语法

最近更新时间：2024-08-07 17:28:42

INSERT OVERWRITE | INTO

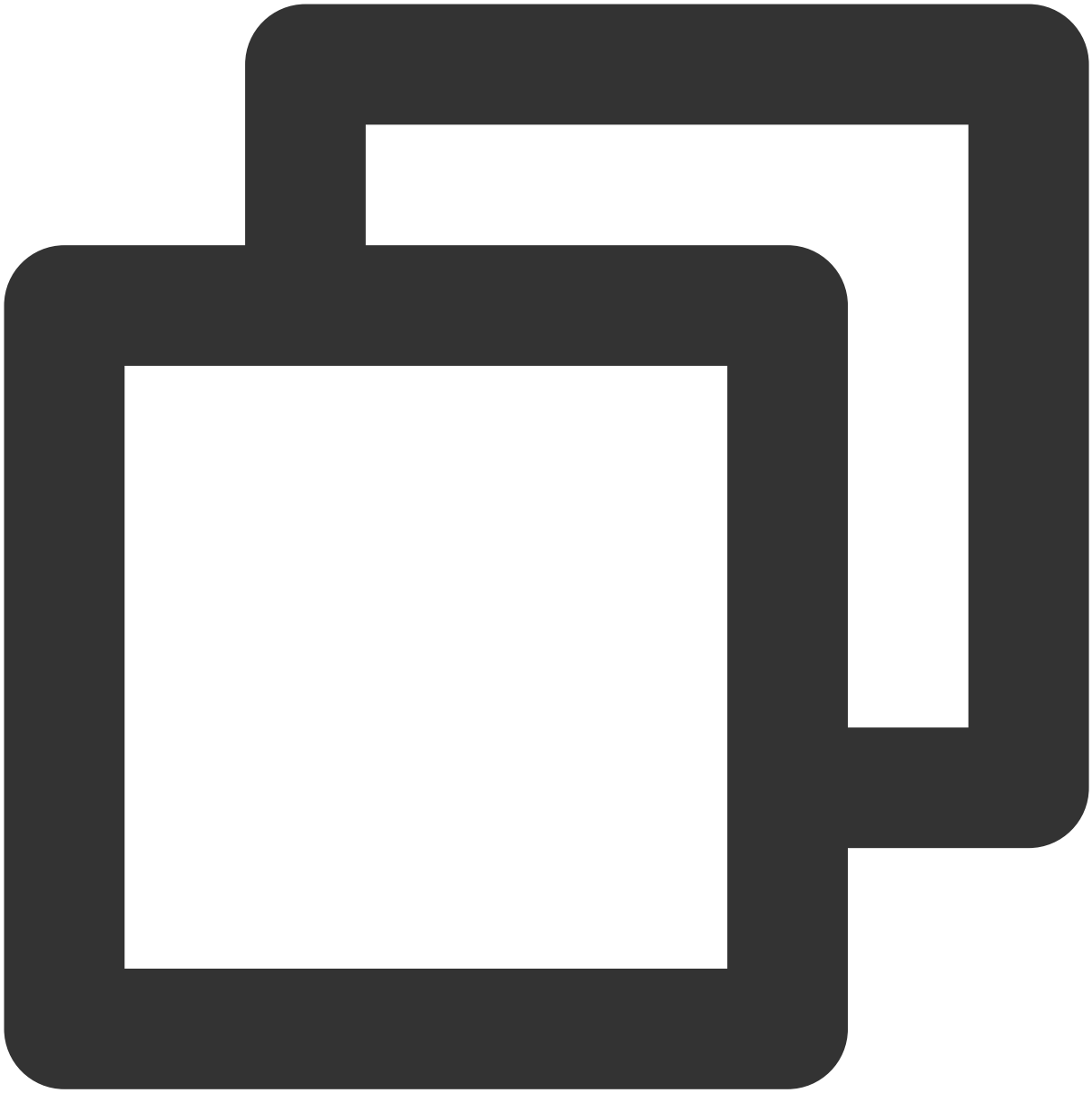
行级数据插入操作

语法



```
INSERT { OVERWRITE | INTO } [ TABLE ] table_name
[ PARTITION clause ]
{ VALUES (column_values,...), (column_values,...)...
| SELECT select_expr}
```

示例



```
CREATE TABLE IF NOT EXISTS `table_01` (
    `id` INTEGER,
    `num` int,
    `name` STRING
) USING `iceberg`

INSERT INTO table_01 PARTITION(name='21') VALUES (1,2), (2,3);
INSERT INTO TABLE table_01 VALUES (3,2,'abc'), (4,3,'abd');
```

MERGE INTO

行级数据更新操作，可用于替换 INSERT OVERWRITE 操作

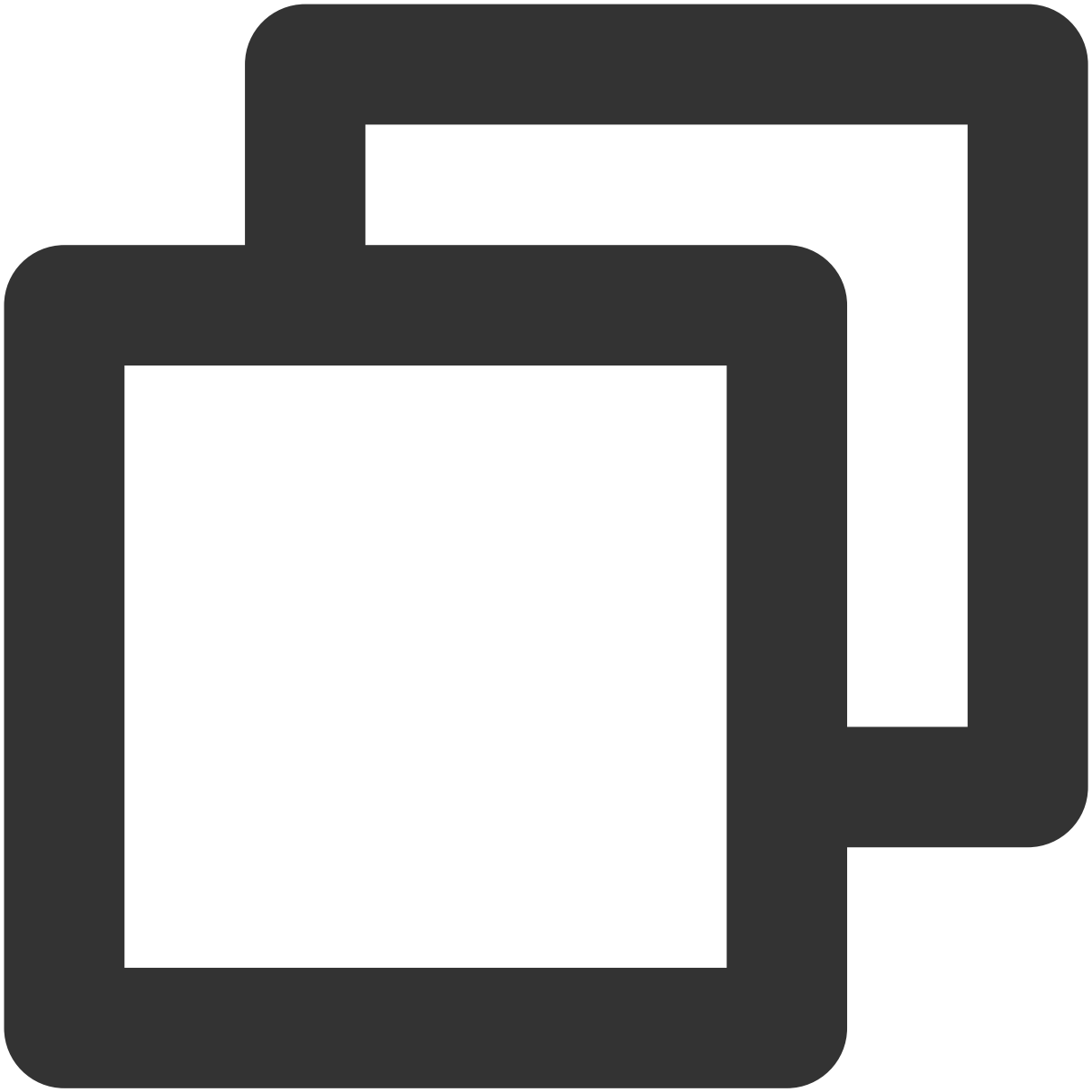
语法

```
MERGE INTO target_table_name [target_alias]
USING source_table_reference [source_alias]
ON merge_condition
[ WHEN MATCHED [ AND condition ] THEN matched_action ] [...]
```

```
[ WHEN NOT MATCHED [ AND condition ] THEN not_matched_action ] [...]

matched_action
{ DELETE |
UPDATE SET * |
UPDATE SET { column1 = value1 } [, ...] }

not_matched_action
{ INSERT * |
INSERT (column1 [, ...] ) VALUES (value1 [, ...])
```

DELETE FROM

语法

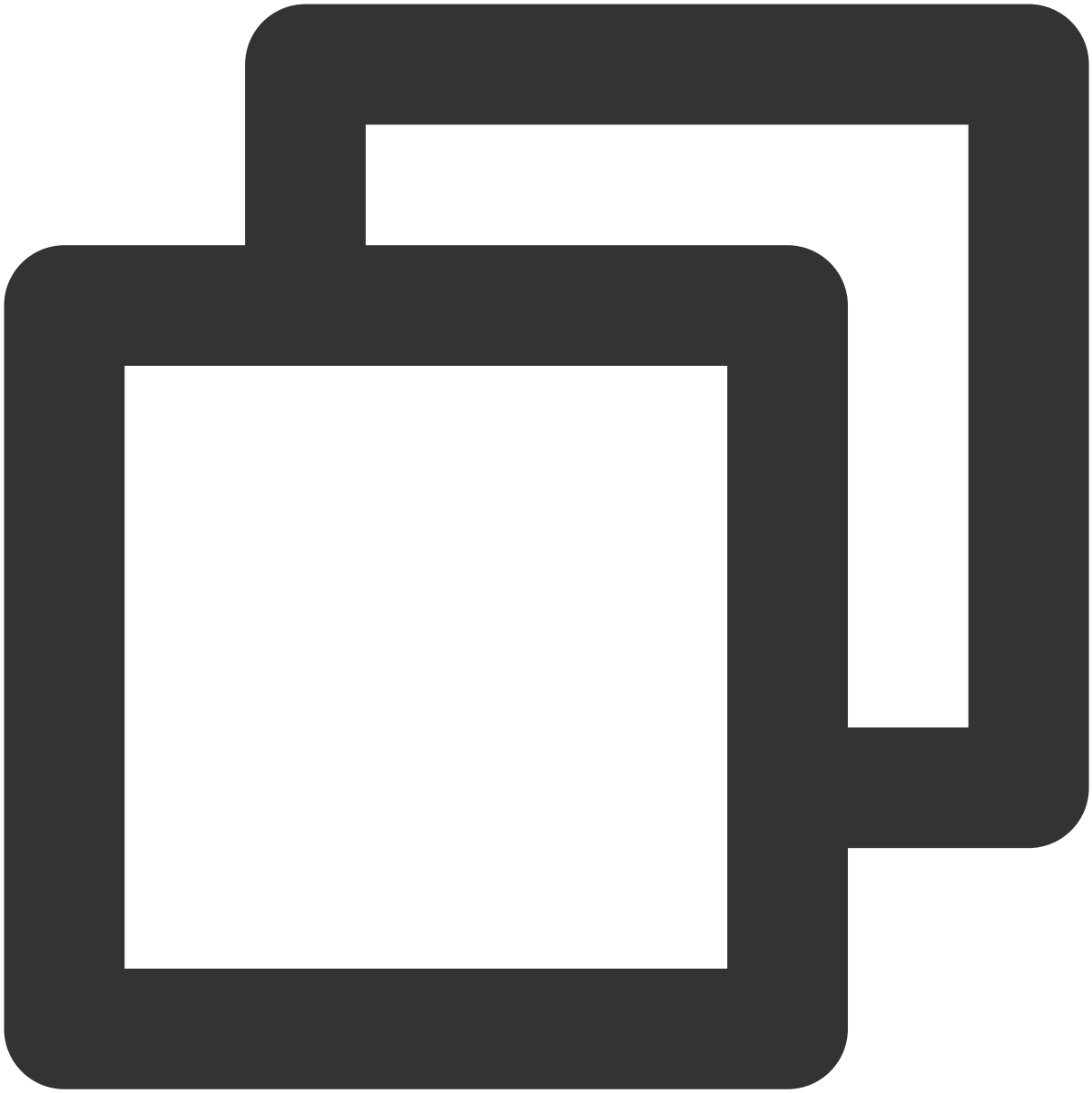


```
DELETE FROM table_name [table_alias] [WHERE predicate]
```

UPDATE

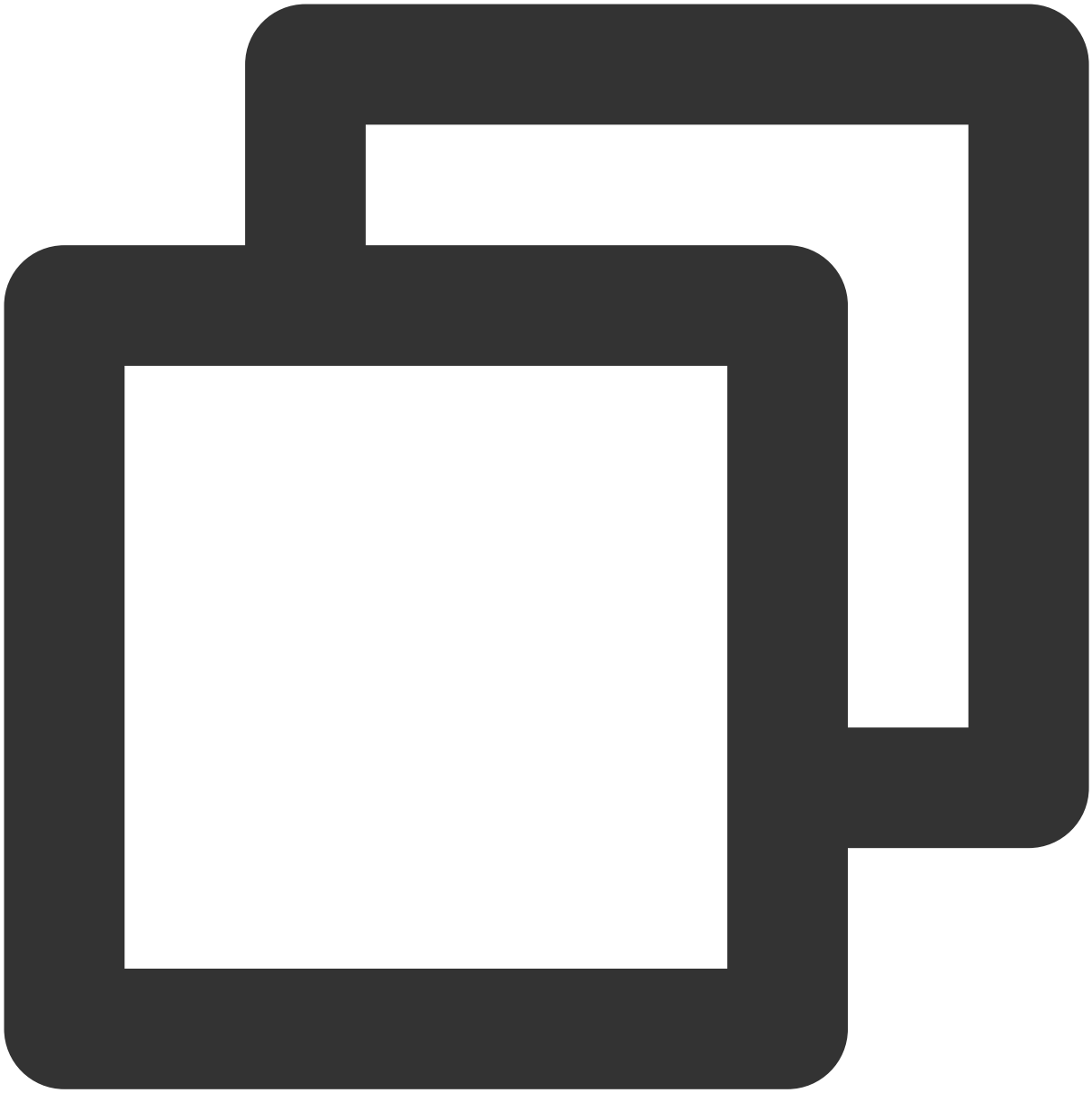
从Spark 3.1起支持UPDATE操作

语法



```
UPDATE table_identifier [table_alias]
SET { { column_name | field_name } = expr } [, ...]
[WHERE clause]
```

示例



```
UPDATE dempts SET c1 = 'update_c1', c2 = 'update_c2'
WHERE ts >= '2020-05-01 00:00:00' and ts < '2020-06-01 00:00:00'

UPDATE dempts SET session_time = 0, ignored = true
WHERE session_time < (SELECT min(session_time) FROM prod.db.good_events)

UPDATE dempts AS t1 SET order_status = 'returned'
WHERE EXISTS (SELECT oid FROM prod.db.returned_orders WHERE t1.oid = oid)
```

版权所有：腾讯云计算（北京）有限责任公司

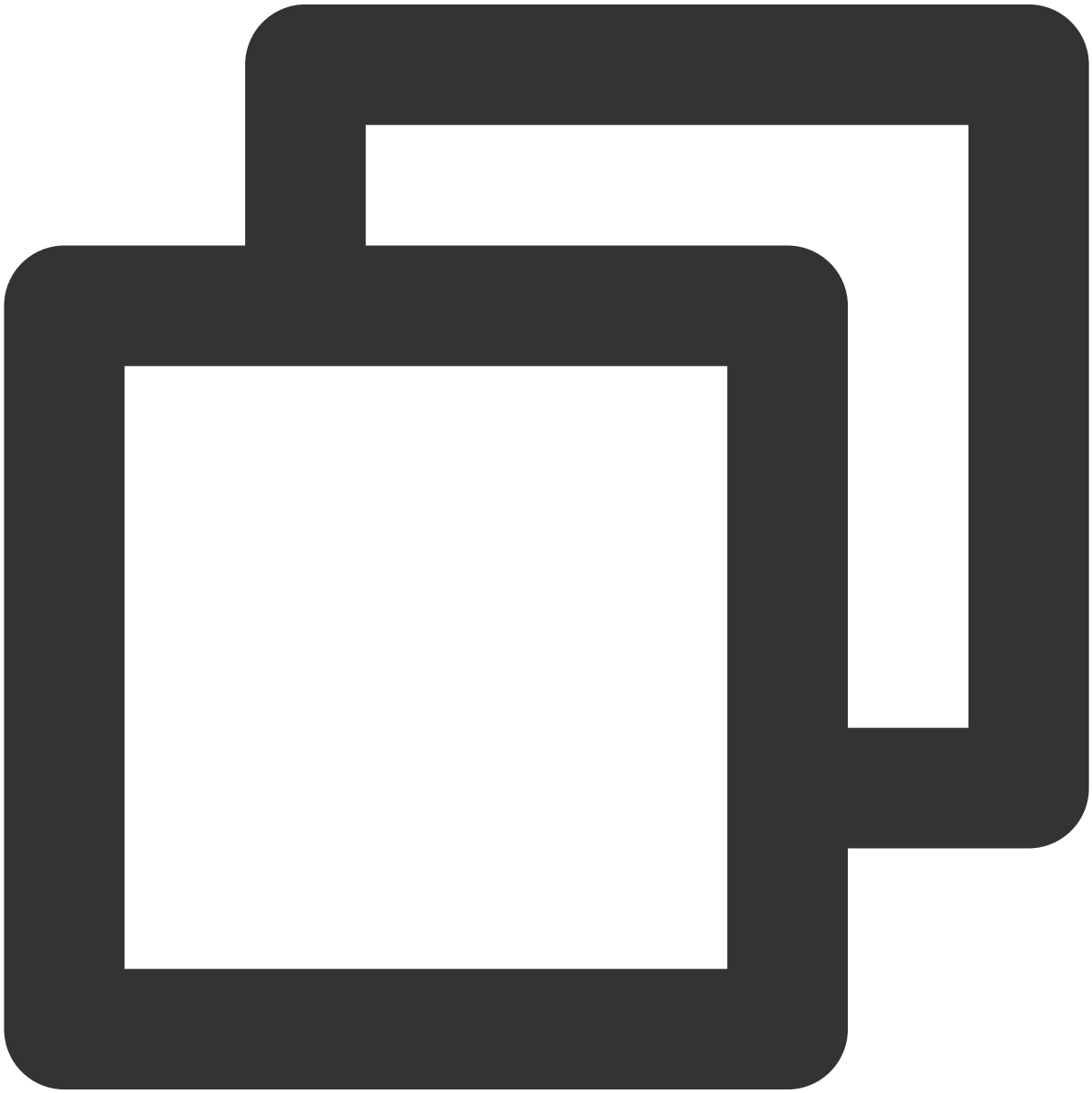
版权所有：腾讯云计算（北京）有限责任公司

DQL 语法

最近更新时间：2024-08-07 17:28:55

SELECT

语法



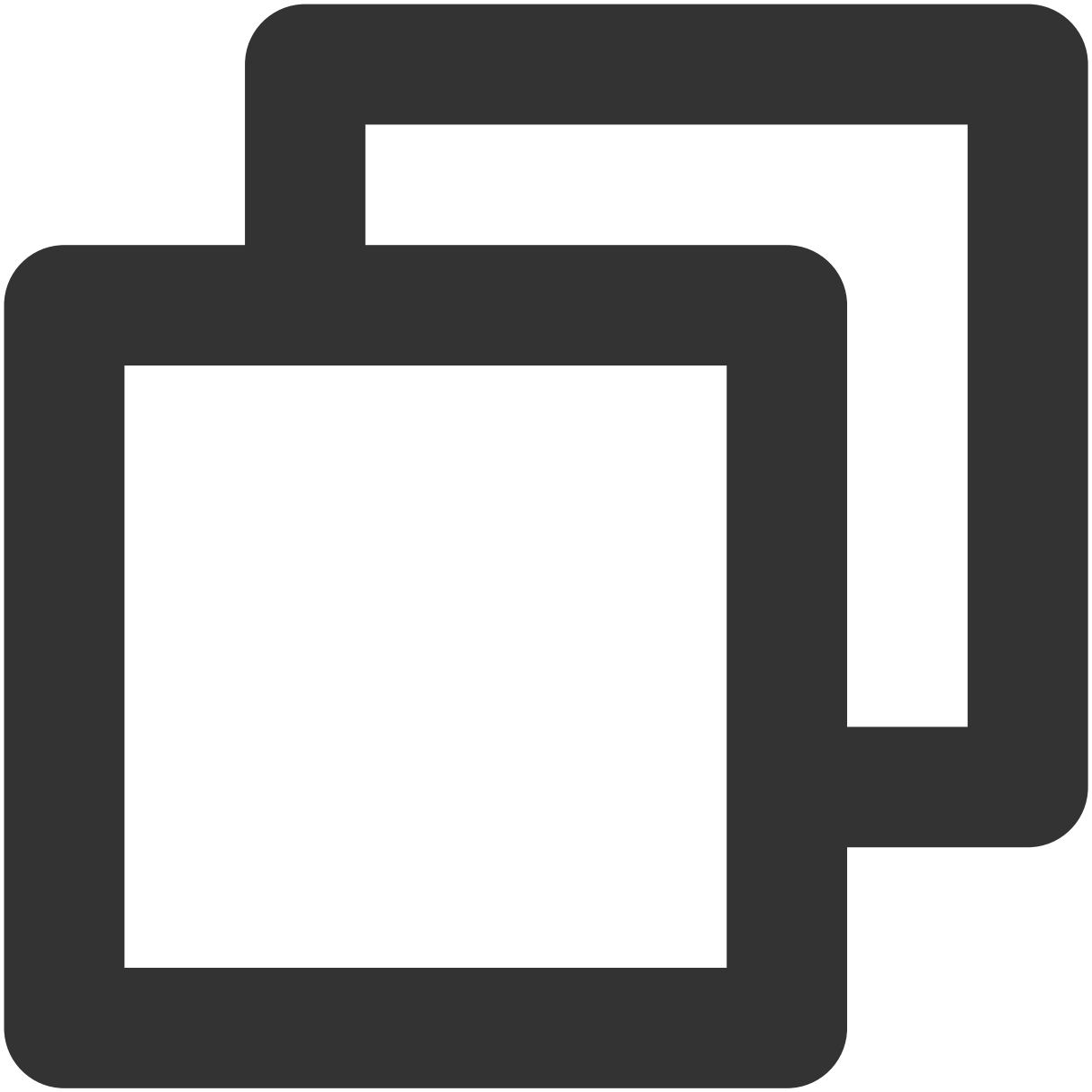
```
SELECT [ hints ] [ ALL | DISTINCT ] { named_expression | star_clause } [, ...]
```

```
FROM from_item [, ...]
    [ LATERAL VIEW clause ]
    [ PIVOT clause ]
    [ WHERE clause ]
    [ GROUP BY clause ]
    [ HAVING clause]
    [ QUALIFY clause ]

from_item
{ table_name [ TABLESAMPLE clause ] [ table_alias ] |
    JOIN clause |
    [ LATERAL ] table_valued_function [ table_alias ] |
    VALUES clause |
    [ LATERAL ] ( query ) [ TABLESAMPLE clause ] [ table_alias ] }

named_expression
    expression [ column_alias ]

star_clause
    [ { table_name | view_name } . ] *
```

TABLE METADATA

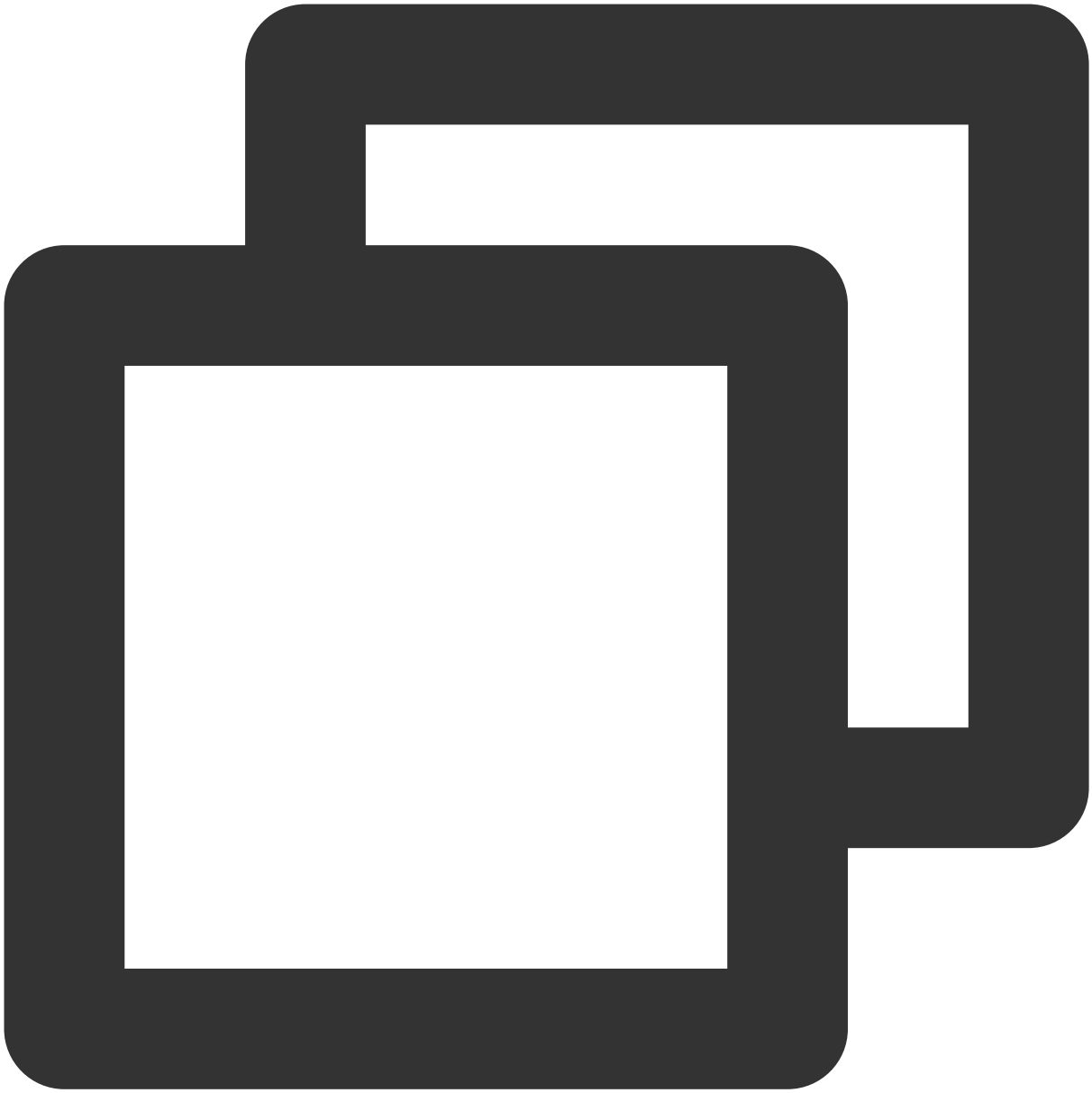
支持四段式的 Iceberg 表元数据查询，包括：`history`、`snapshots`、`files`、`manifests`。

语法



```
SELECT select_expr (, select_expr)*
FROM `Catalog`.`db`.`tableName${history|snapshots|files|manifests|partitions|all_da
[WHERE where_condition]
[LIMIT [offset,] rows]
```

示例



```
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$history` ORDER BY snapshot_id
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$snapshots` ORDER BY snapshot_i
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$files` ORDER BY file_size_in_b
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$manifests` ORDER BY length LIM
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$partitions` LIMIT 10;
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$all_data_files` LIMIT 10;
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$all_manifests` LIMIT 10;
```

版权所有：腾讯云计算（北京）有限责任公司

TIME TRAVEL

FOR SYSTEM_TIME AS OF/ TIMESTAMP AS OF

Spark3.3以上支持，支持字符串和 Unix 时间戳两种格式

示例

```
SELECT empno FROM sales.emp FOR SYSTEM_TIME AS OF '1986-10-26 01:21:00';
SELECT empno FROM sales.emp FOR SYSTEM_TIME AS OF 12324235546;
SELECT empno FROM sales.emp TIMESTAMP AS OF '1986-10-26 01:21:00';
```

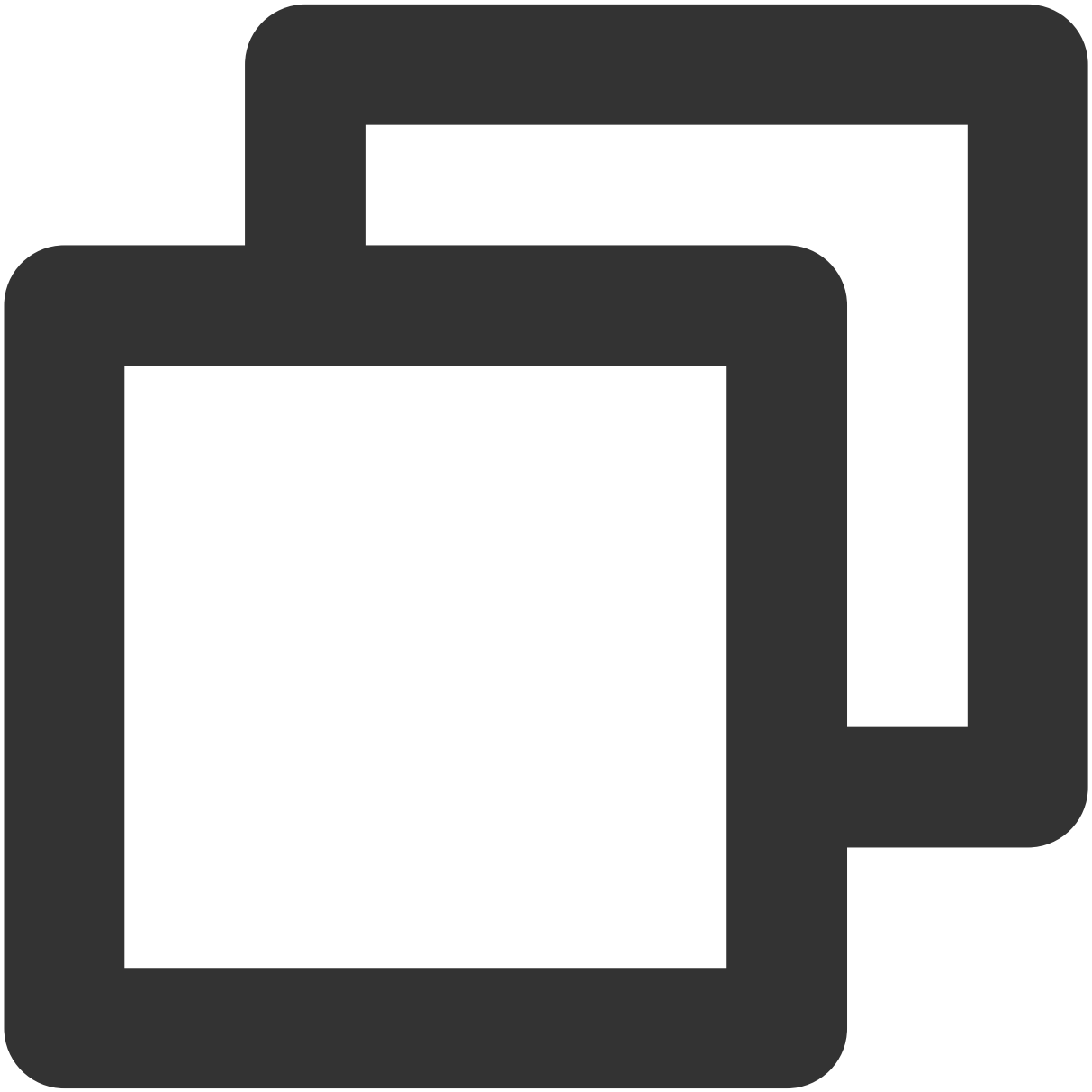
版权所有：腾讯云计算（北京）有限责任公司

```
SELECT empno FROM sales.emp TIMESTAMP AS OF 11111;
```

FOR SYSTEM_VERSION AS OF/ VERSION AS OF (Spark 3.3支持)

支持字符串和 snapshot id

示例

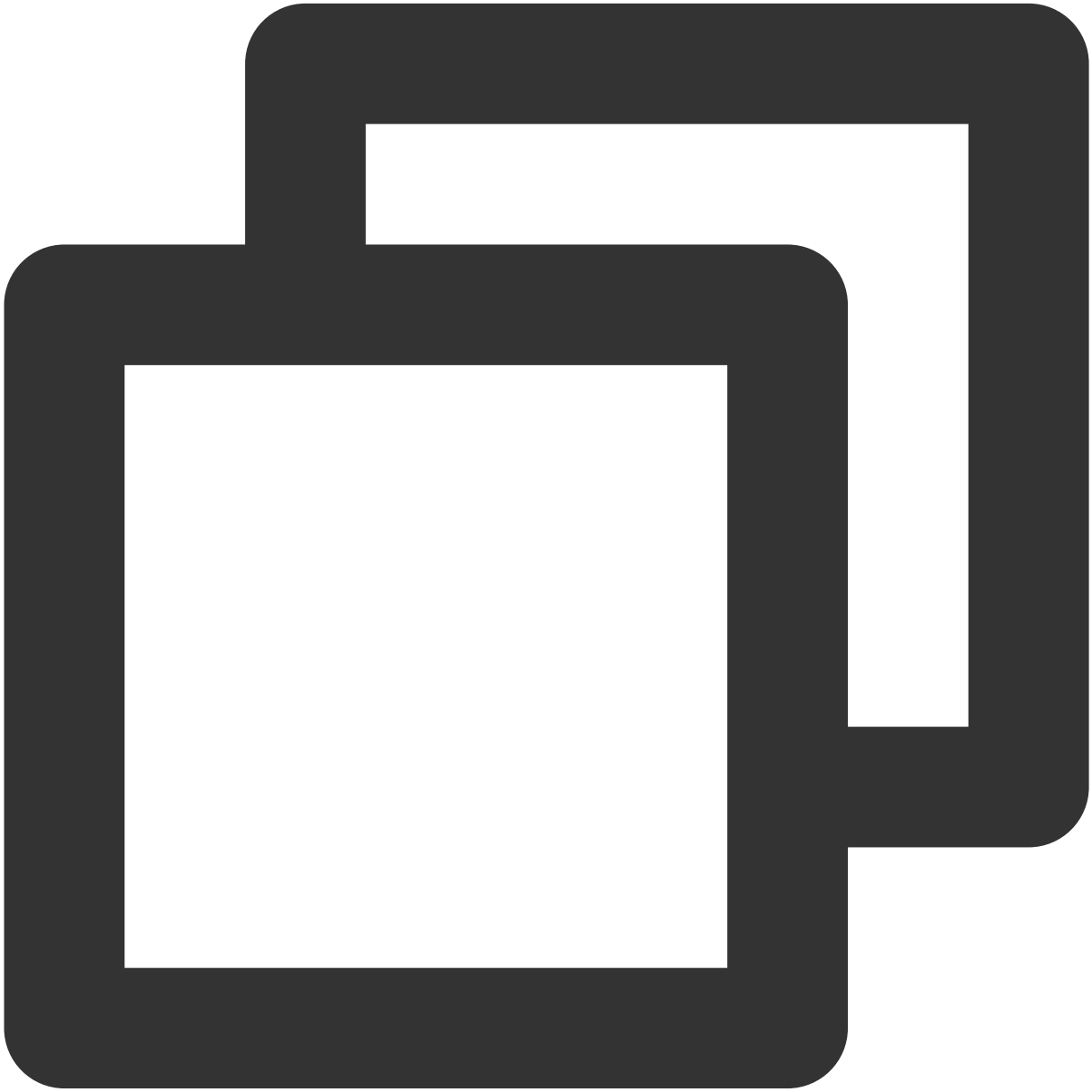


```
SELECT empno FROM sales.emp VERSION AS OF 'Snapshot123456789';
SELECT empno FROM sales.emp VERSION AS OF 11111;
SELECT empno FROM sales.emp FOR SYSTEM_VERSION AS OF 'Snapshot123456789';
```

```
SELECT empno FROM sales.emp FOR SYSTEM_VERSION AS OF 11111;
```

AT_TIMESTAMP_XXXX

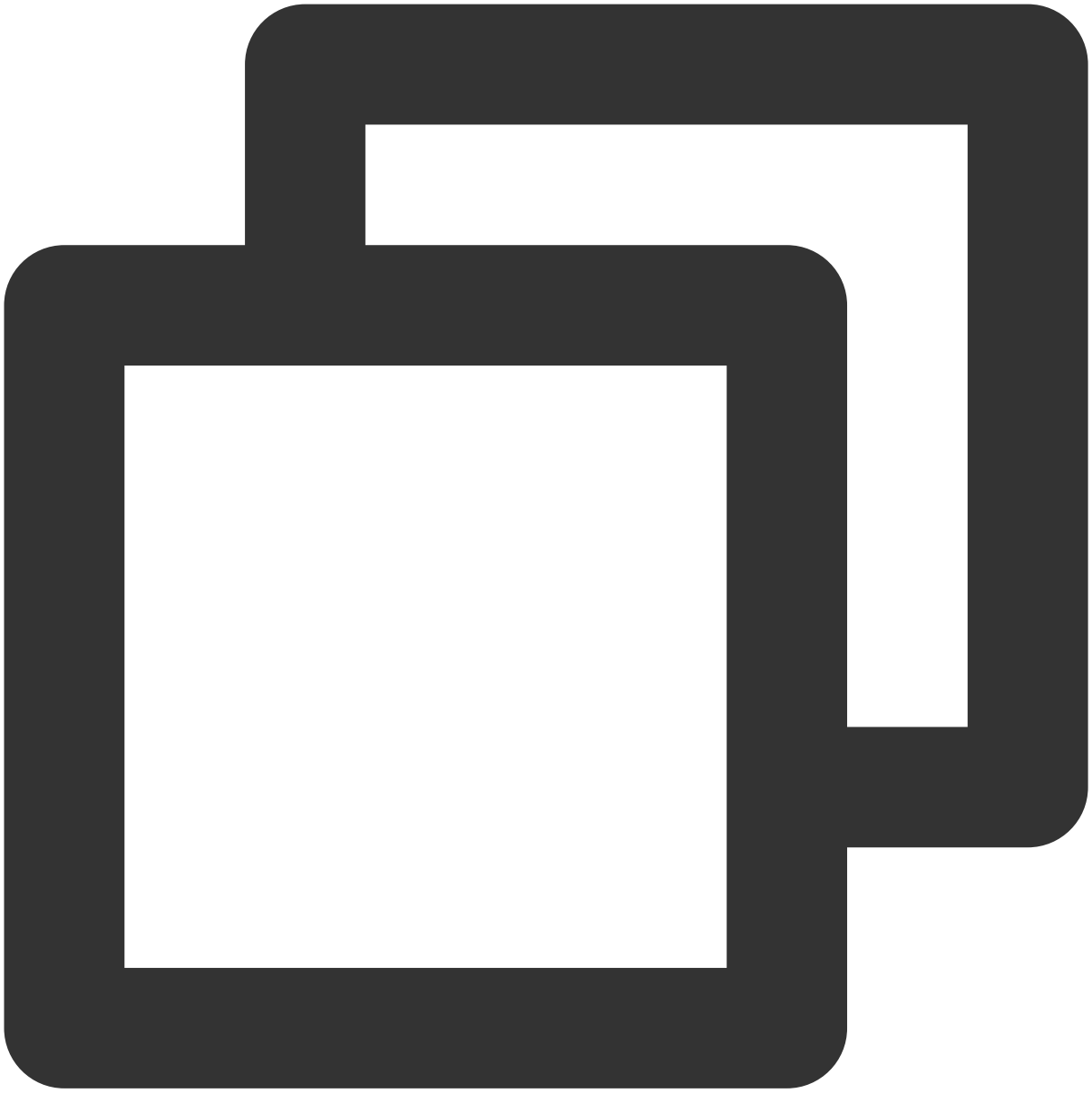
示例



```
SELECT * FROM `sales`.`emp$at_timestamp_1111` ;
```

SNAPSHOT_ID_XXXX

示例



```
SELECT * FROM `sales`.`emp$snapshot_id_1111`
```

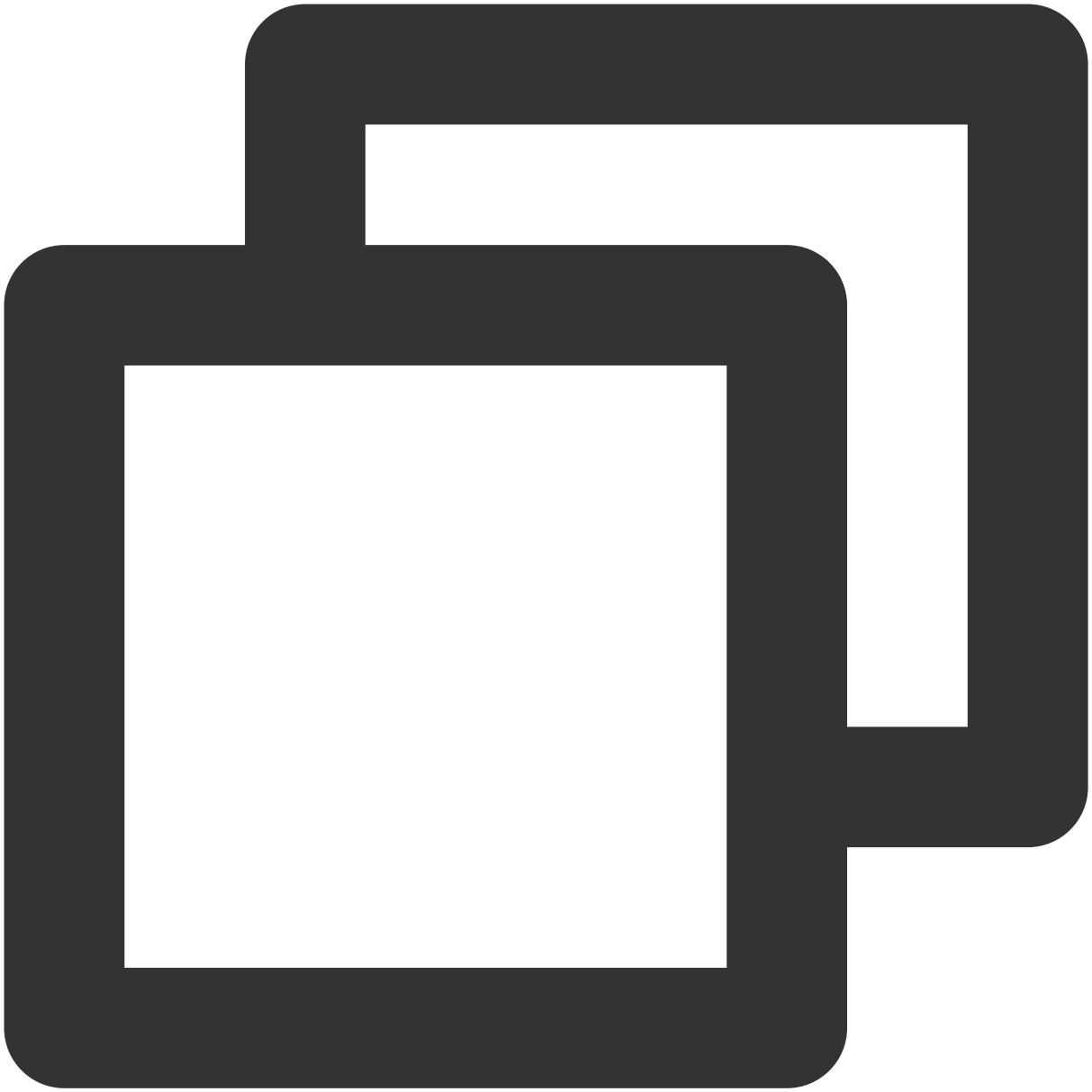
Procedure

最近更新时间：2024-08-07 17:29:17

说明

支持内核：SparkSQL。
适用表类型：外部 Iceberg 表、原生 Iceberg 表。

基本句法



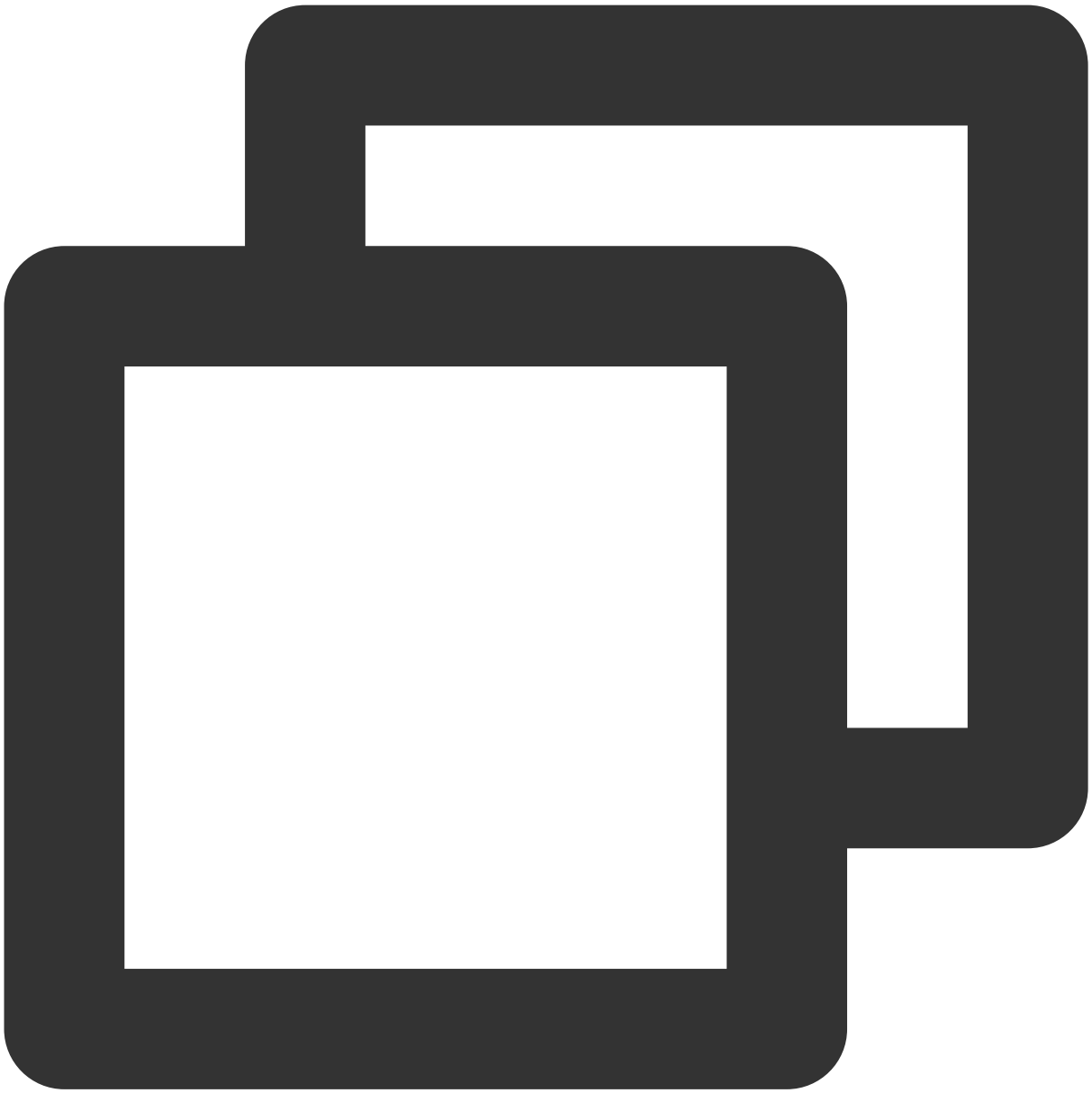
```
CALL catalog_name.system.procedure_name(arg_name_2 => arg_2, arg_name_1 => arg_1);

CALL catalog_name.system.procedure_name(arg_1, arg_2, ... arg_n);
```

Snapshot 管理

rollback_to_snapshot

回滚快照到指定版本。

输入参数：表名和版本号。

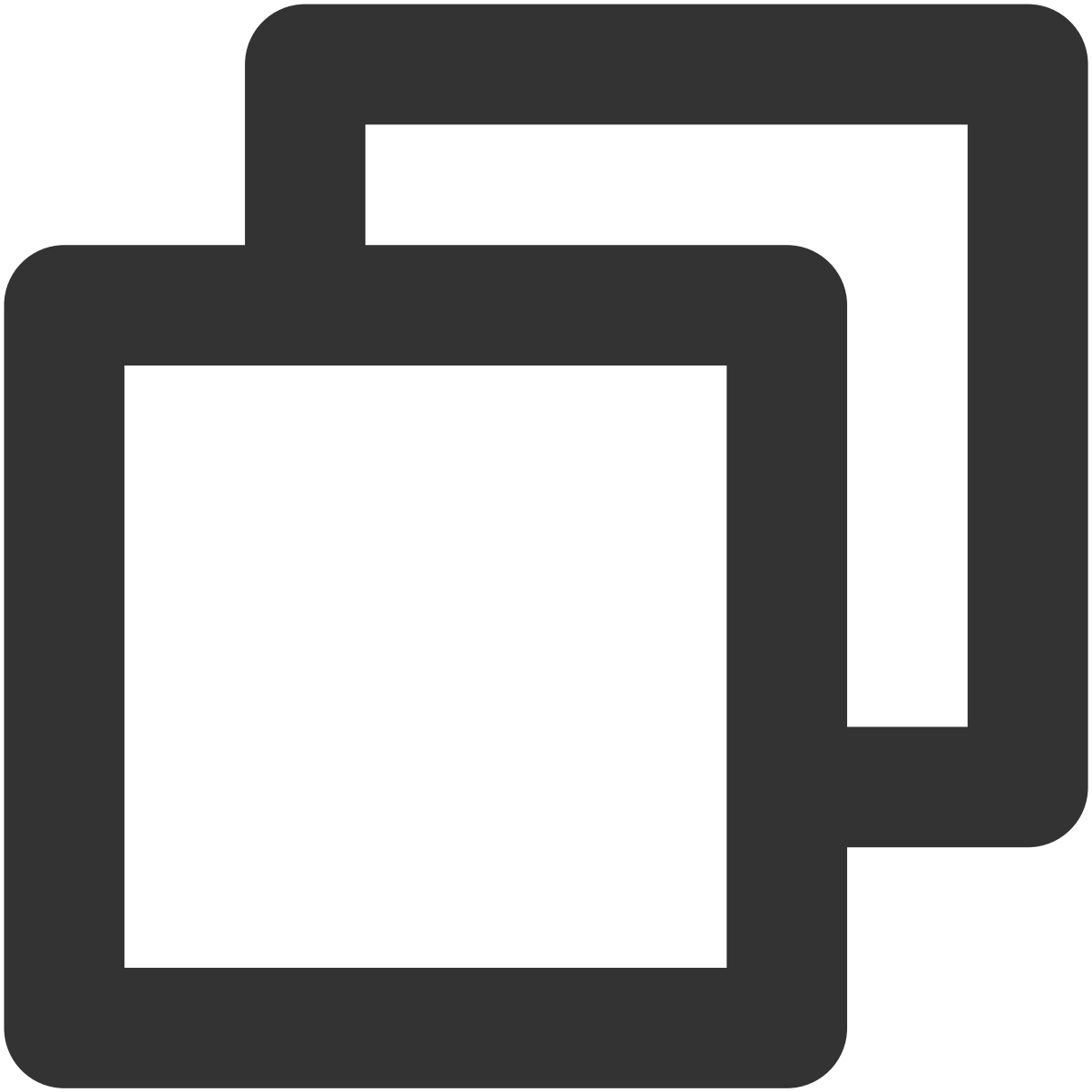


```
CALL `DataLakeCatalog`.`system`.rollback_to_snapshot('validation.dempts', 1);
```

rollback_to_timestamp

回滚快照到指定时间戳。

输入参数：表名和时间戳。

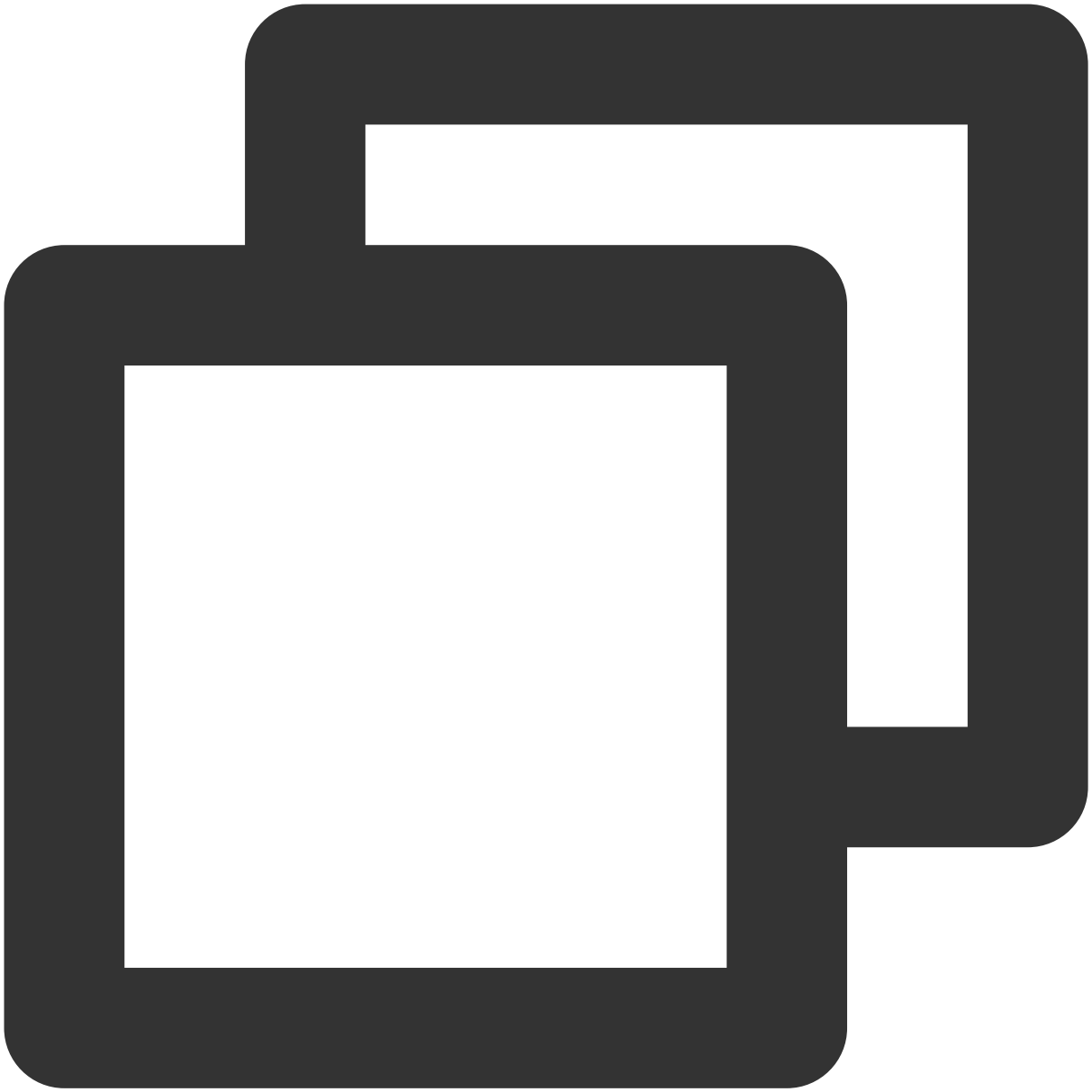


```
CALL `DataLakeCatalog`.`system`.rollback_to_timestamp('validation.dempts', TIMESTAM
```

set_current_snapshot

设置当前快照版本。

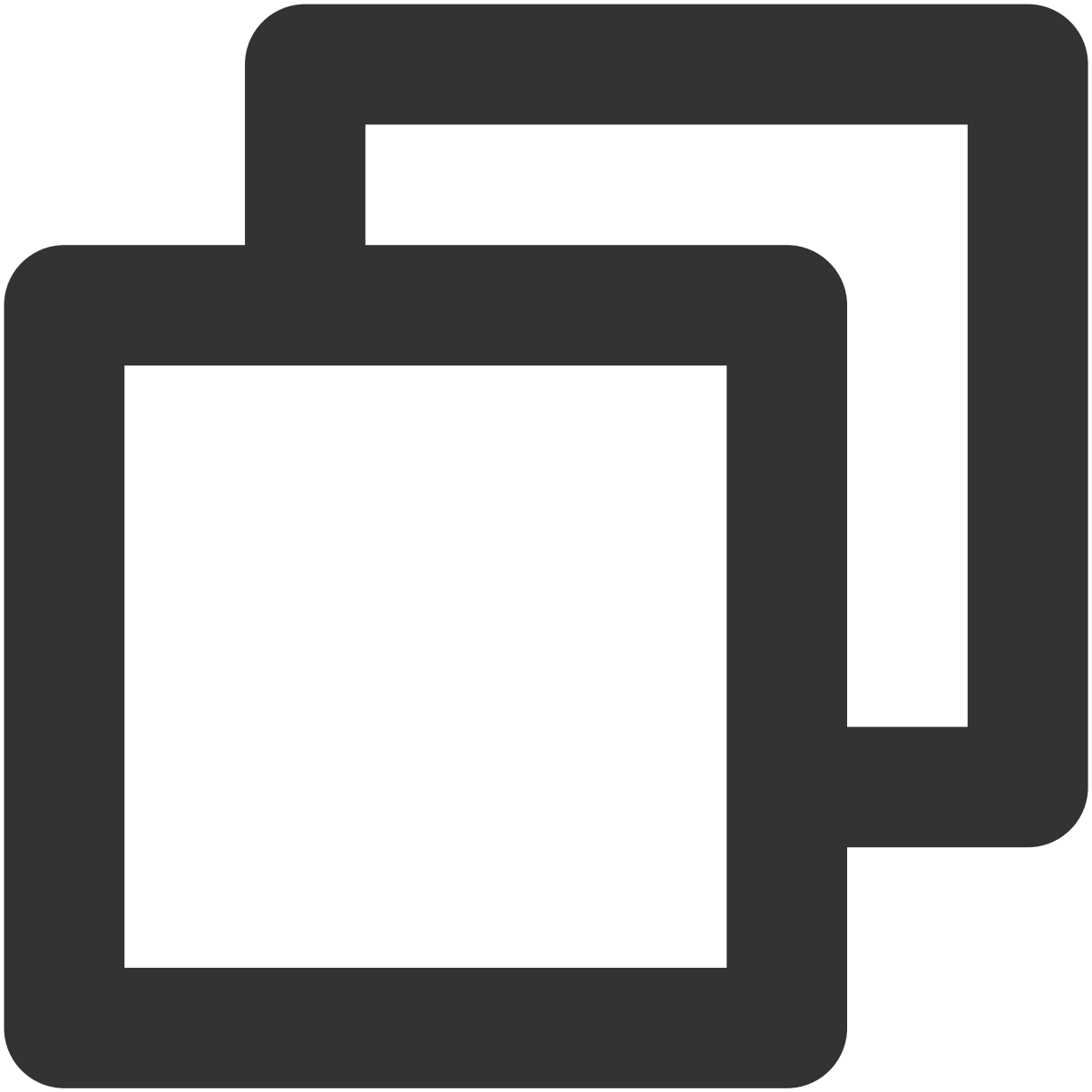
输入参数：表名和版本号。



```
CALL `DataLakeCatalog`.`system`.set_current_snapshot('validation.dempts', 1);
```

cherrypick_snapshot

从指定快照版本 cherrypick 到当前快照。

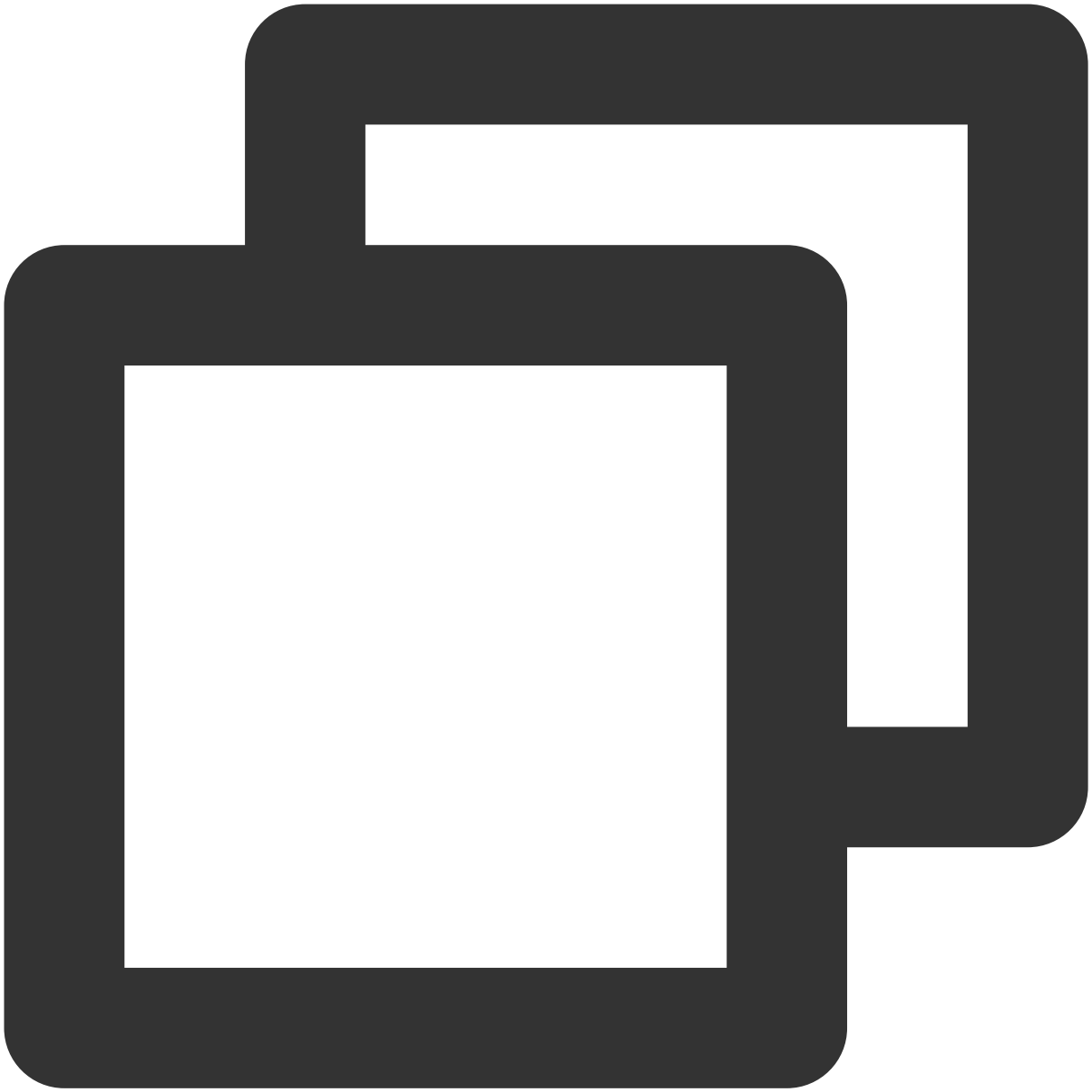


```
CALL `DataLakeCatalog`.`system`.cherrypick_snapshot('validation.dempts', 1);
CALL `DataLakeCatalog`.`system`.cherrypick_snapshot(snapshot_id => 1, table => 'my_
```

Metadata 管理

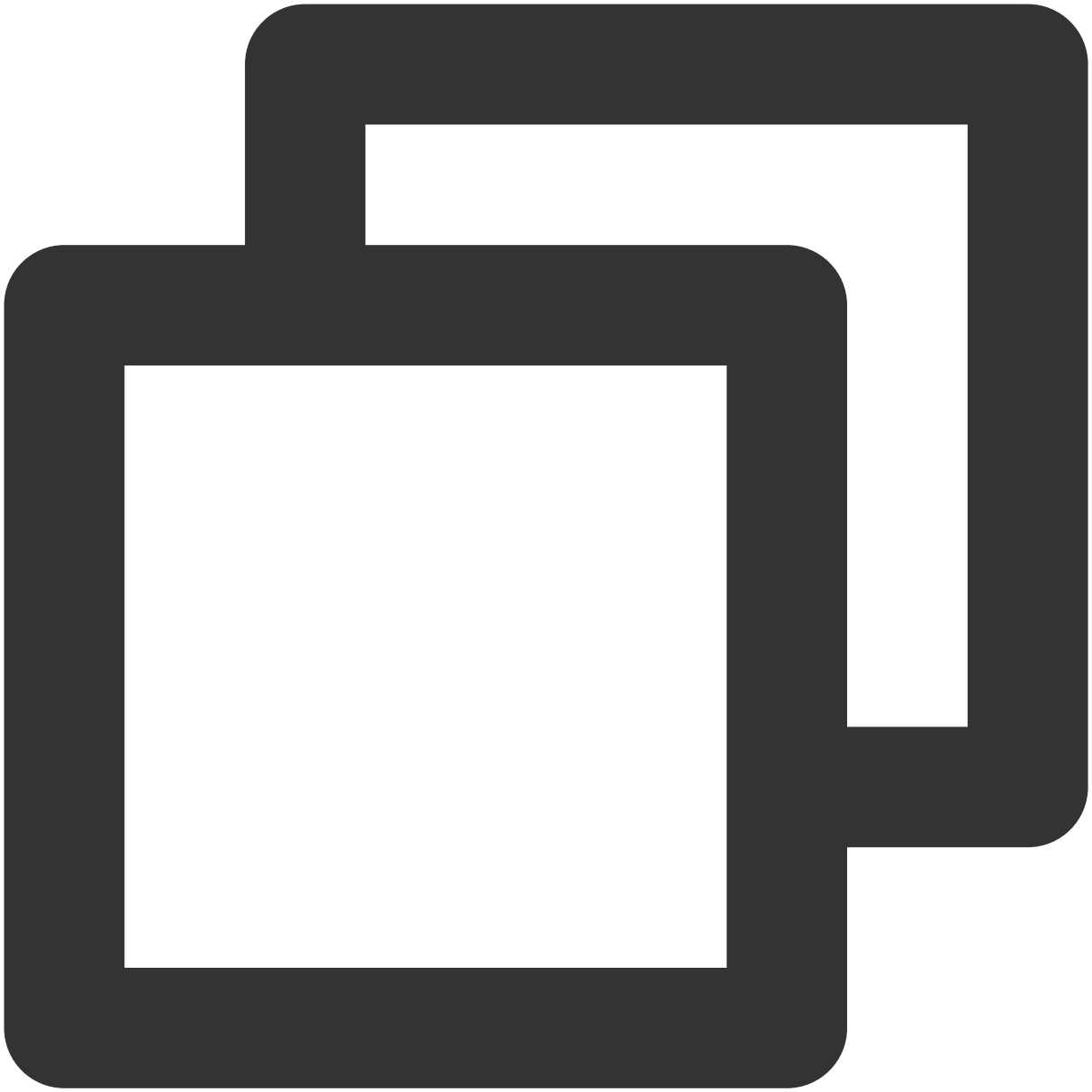
expire_snapshots

清理过期快照，减少小文件数。



```
CALL `Catalog`.`system`.expire_snapshots(table_name, [older_than], [retain_last], [
```

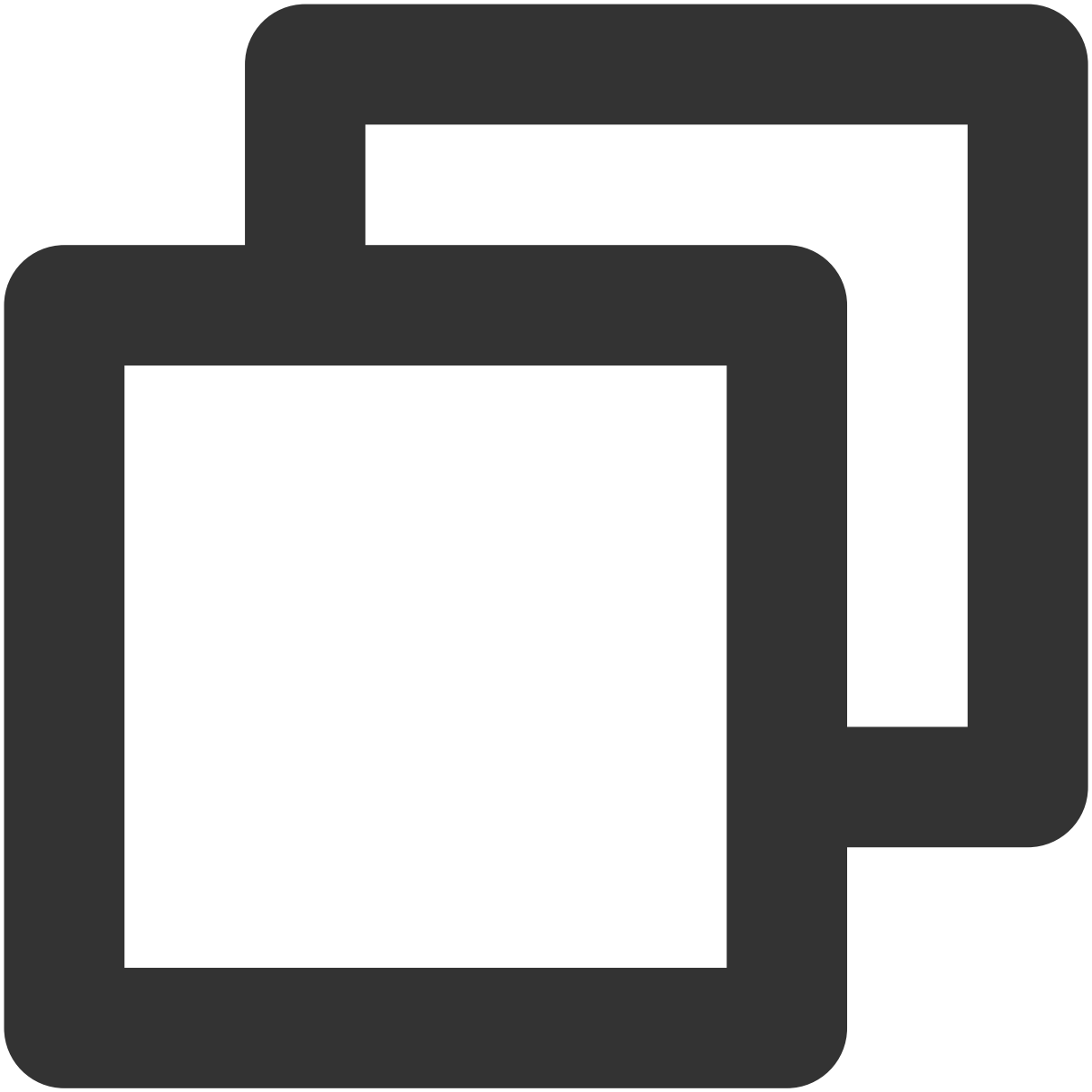
示例：



```
CALL `DataLakeCatalog`.`system`.expire_snapshots('validation.dempts', TIMESTAMP '20
```

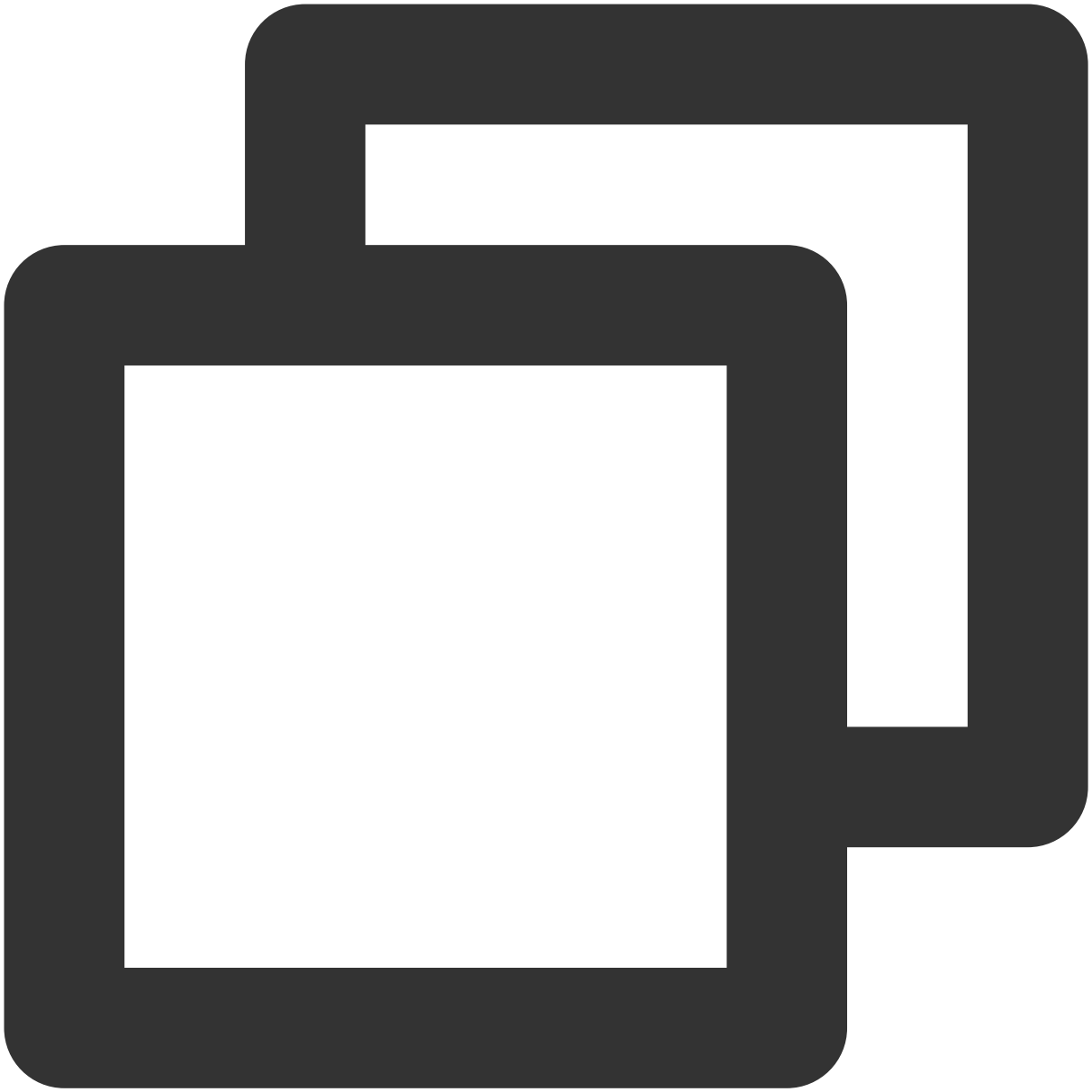
remove_orphan_files

移除不再被引用元数据文件。



```
CALL `Catalog`.`system`.remove_orphan_files(table_name, [older_than], [location], [
```

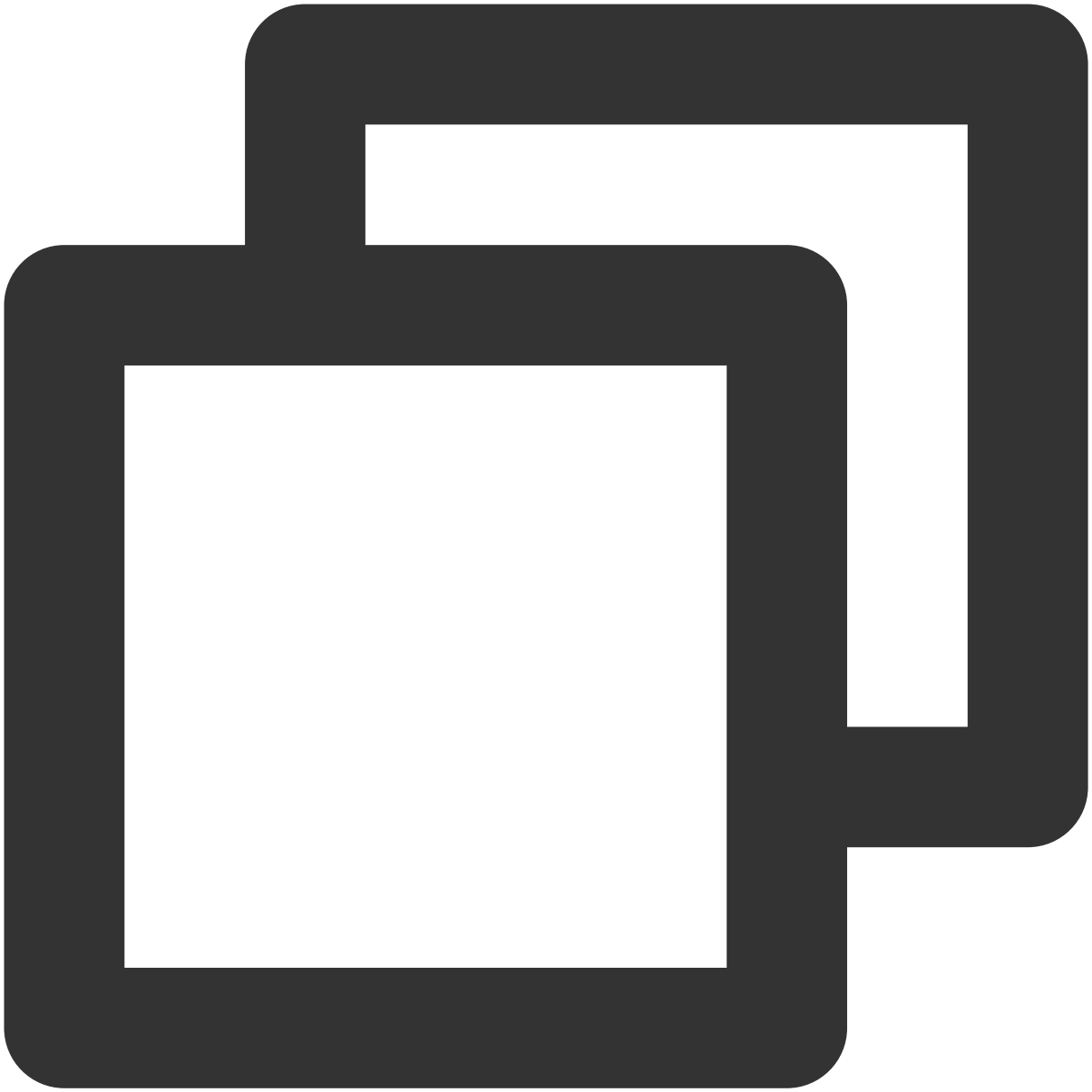
示例：



```
CALL `DataLakeCatalog`.`system`.remove_orphan_files(`table`=>'validation.dempts', d
CALL `DataLakeCatalog`.`system`.remove_orphan_files(`table`=>'validation.dempts', `
CALL `DataLakeCatalog`.`system`.remove_orphan_files('validation.dempts', TIMESTAMP
```

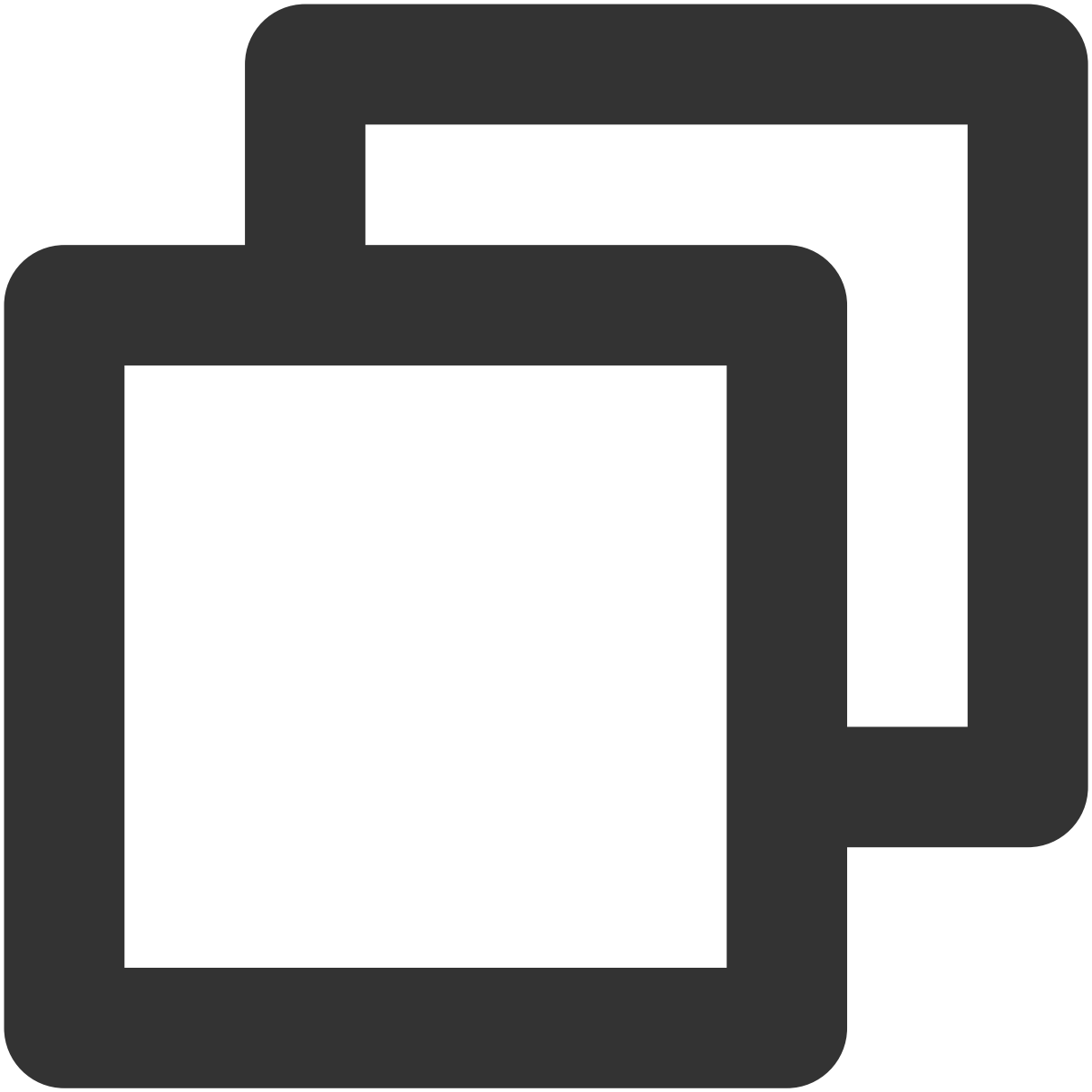
remove_orphan_files

移除不再被引用元数据文件。



```
CALL `Catalog`.`system`.remove_orphan_files(table_name, [older_than], [location], [
```

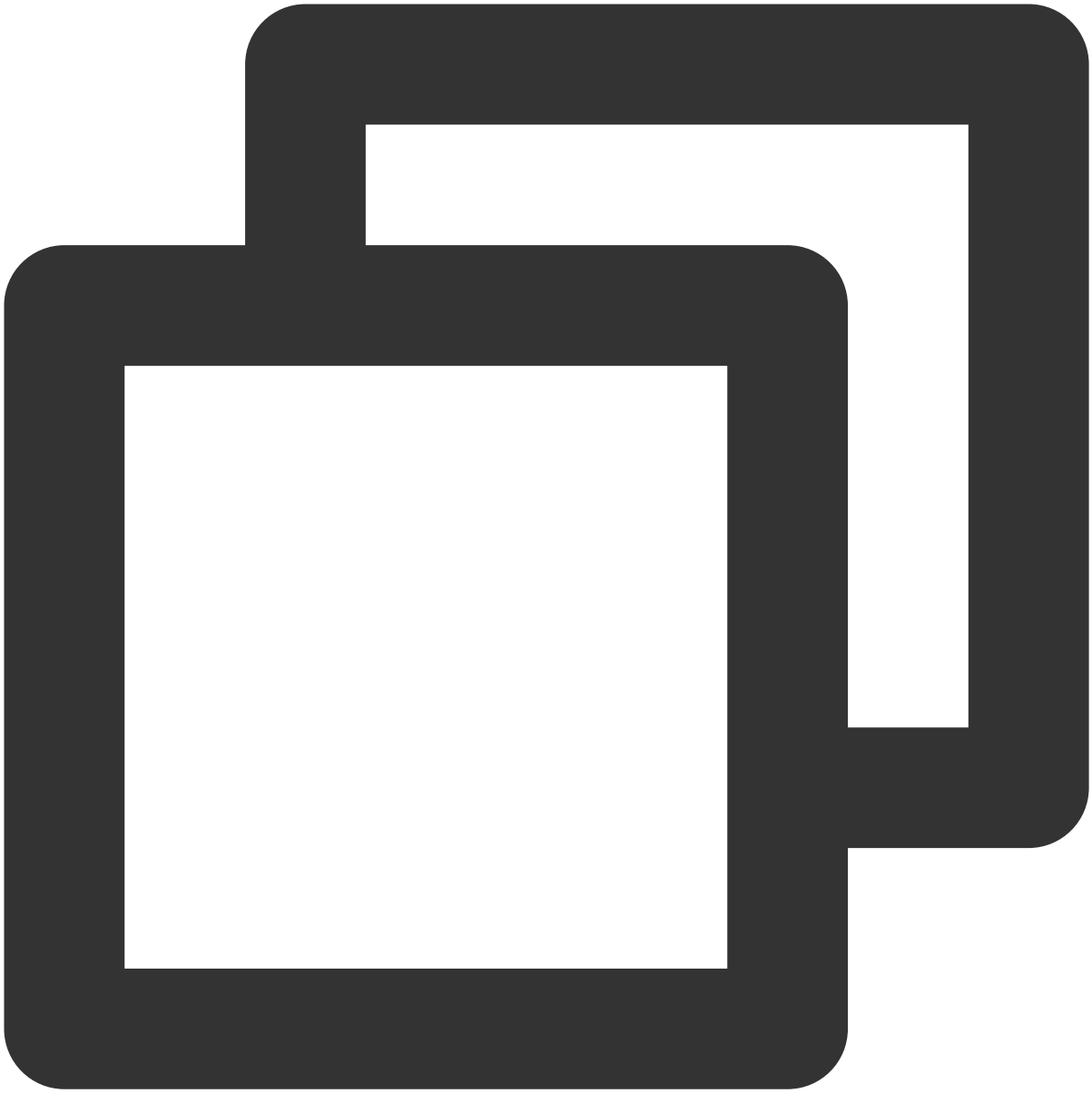
示例：



```
CALL `DataLakeCatalog`.`system`.remove_orphan_files(`table`=>'validation.dempts', d
CALL `DataLakeCatalog`.`system`.remove_orphan_files(`table`=>'validation.dempts', `
CALL `DataLakeCatalog`.`system`.remove_orphan_files('validation.dempts', TIMESTAMP
```

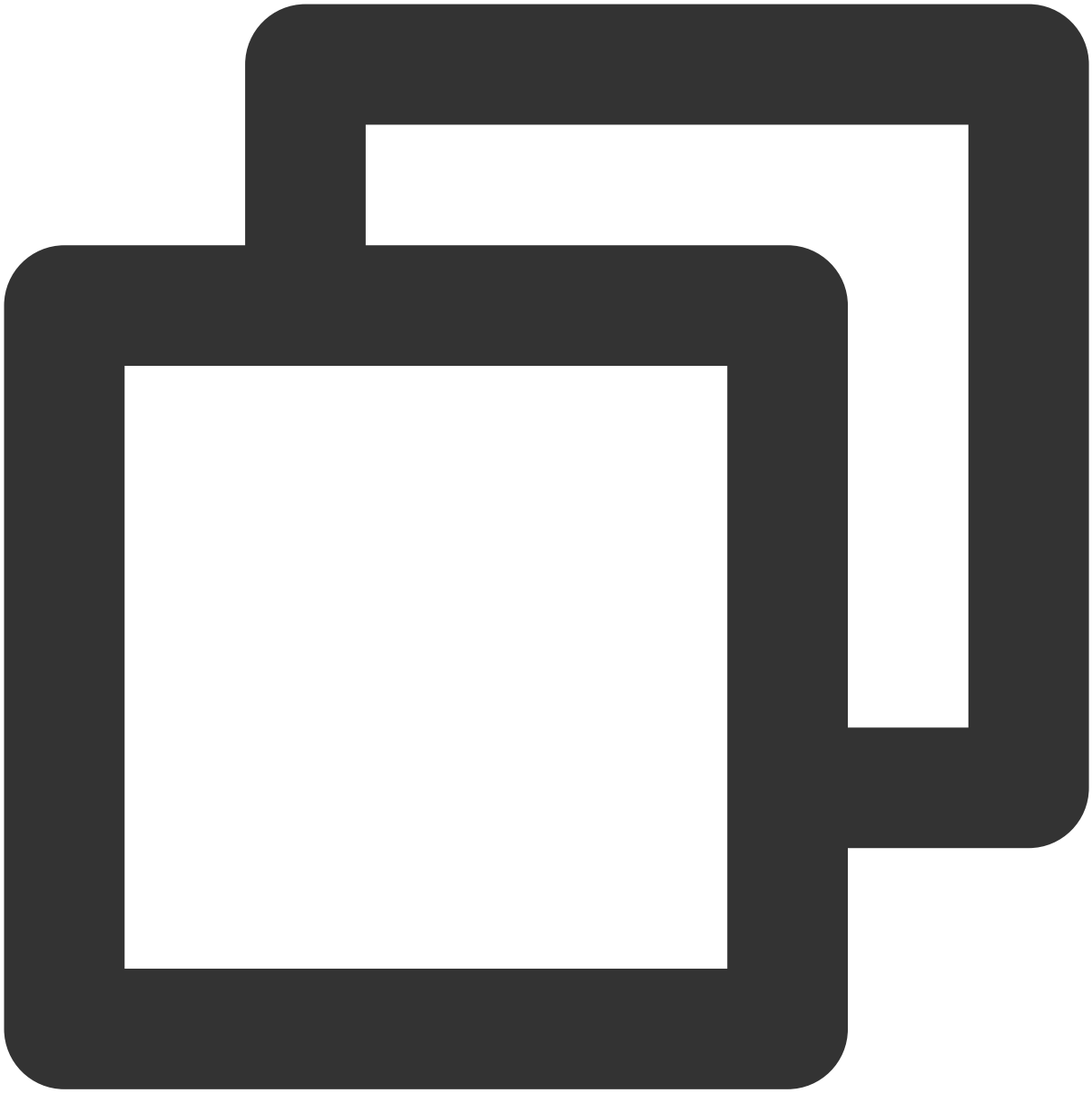
rewrite_data_files

数据文件合并重写，即小数据文件合并。



```
CALL `Catalog`.`system`.rewrite_data_files(table_name, [strategy], [sort_order], [o
```

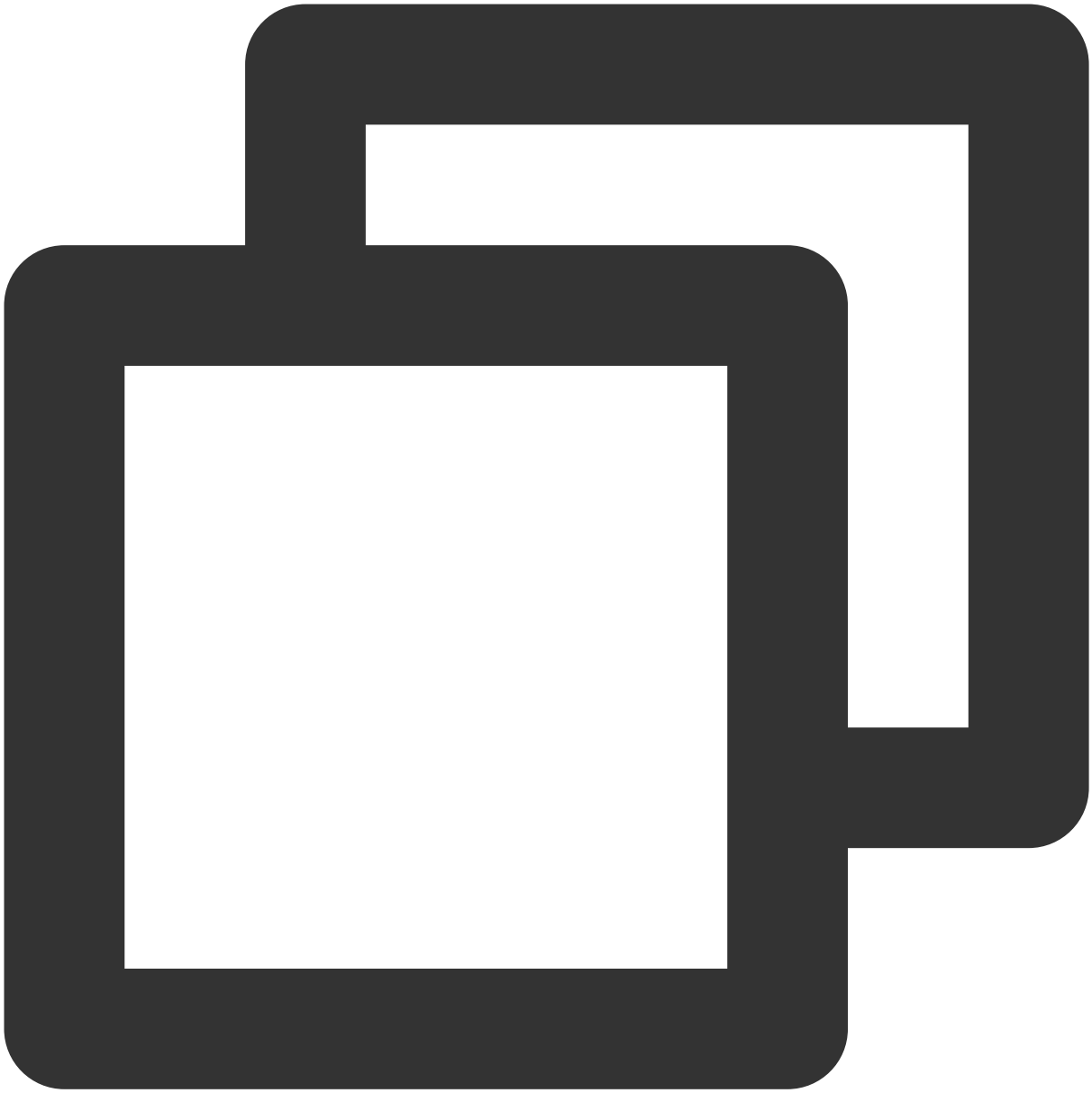
示例：



```
CALL `DataLakeCatalog`.`system`.rewrite_data_files('validation.dempts');
CALL `DataLakeCatalog`.`system`.rewrite_data_files(`table`=>'validation.dempts', `s
CALL `DataLakeCatalog`.`system`.rewrite_data_files(`table`=>'validation.dempts', `o
CALL `DataLakeCatalog`.`system`.rewrite_data_files(`table`=>'validation.dempts', `w
```

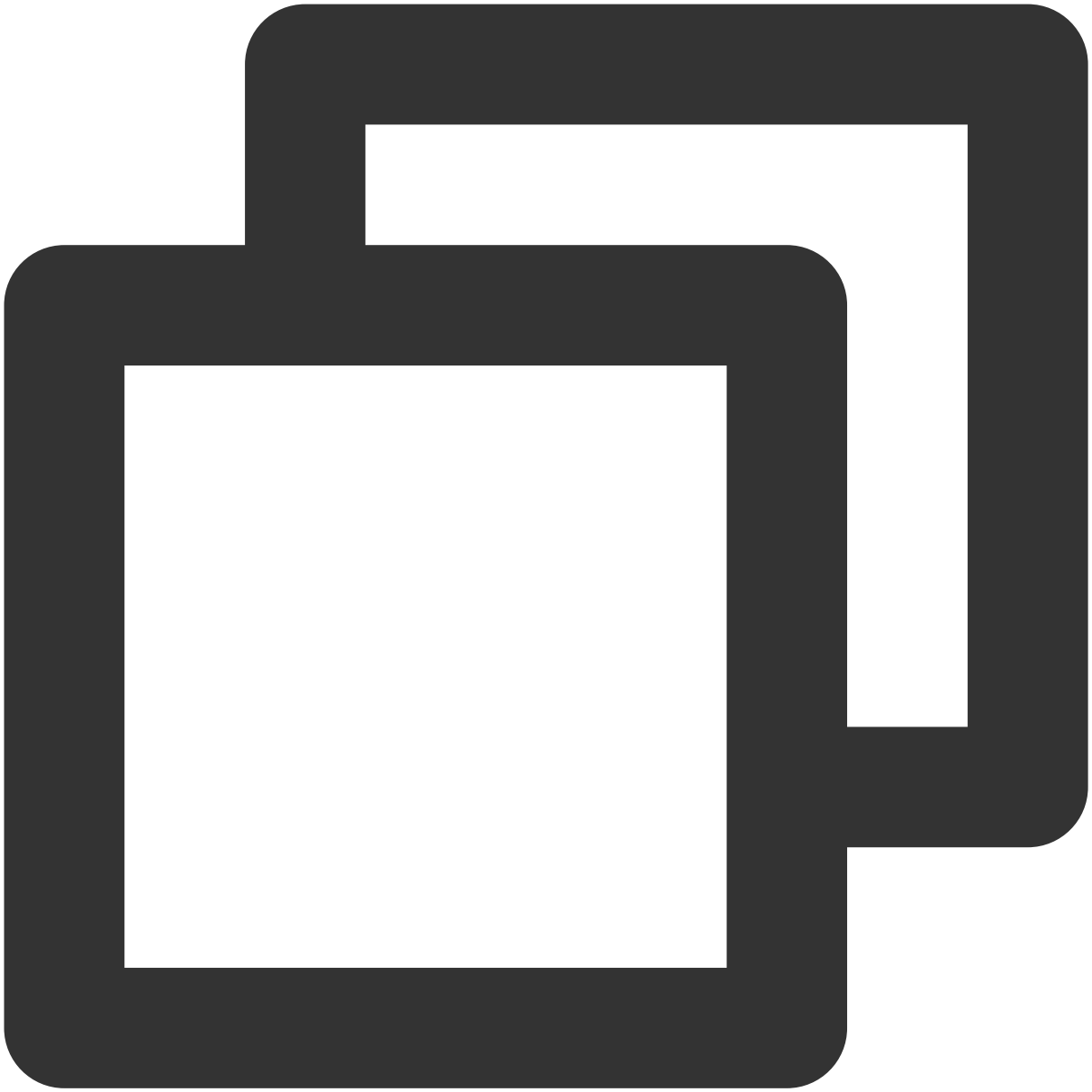
rewrite_manifests

manifests 文件合并重写。



```
CALL `Catalog`.`system`.rewrite_manifests(table_name, [using_caching]);
```

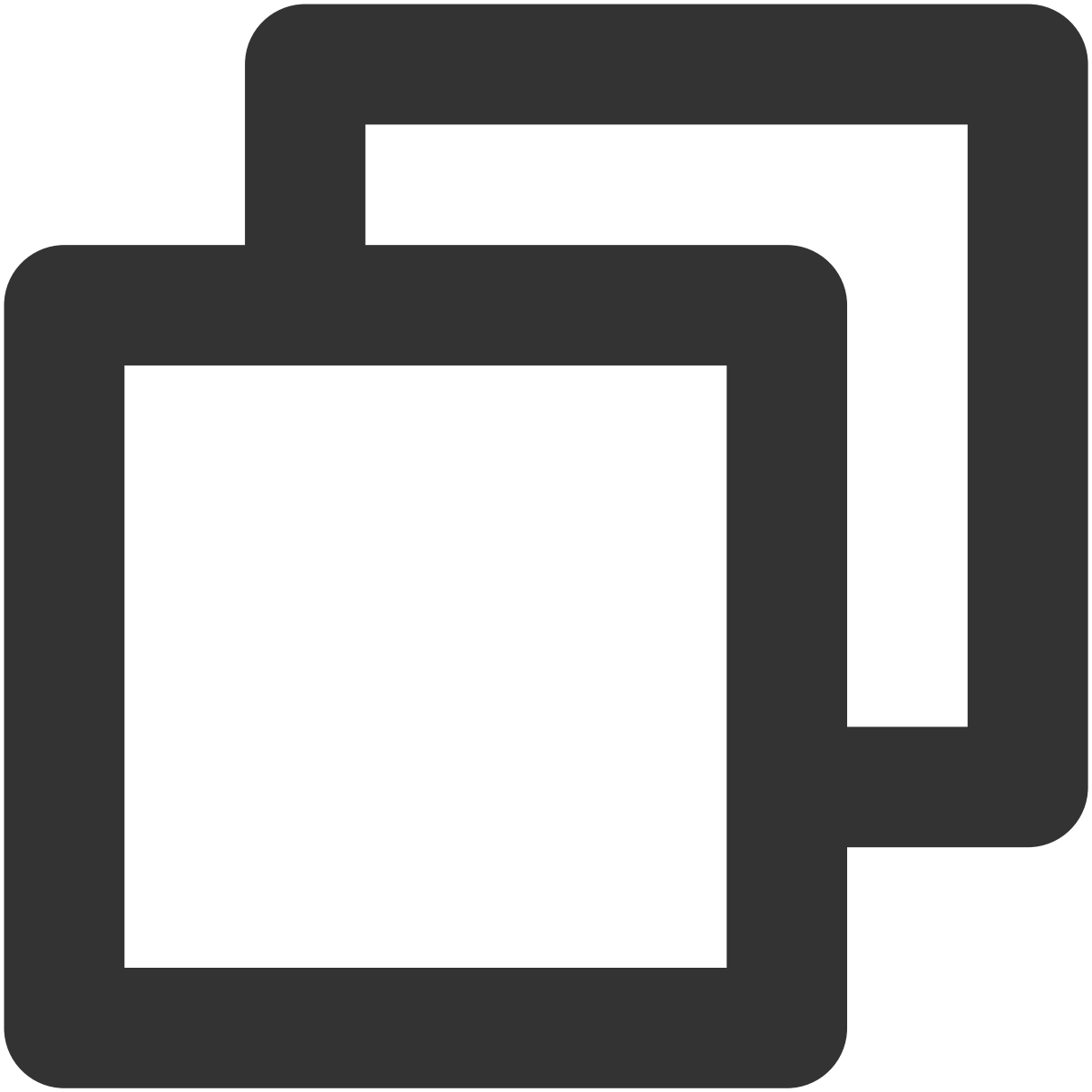
示例：



```
CALL `DataLakeCatalog`.`system`.rewrite_manifests('validation.dempts');
CALL `DataLakeCatalog`.`system`.rewrite_manifests('validation.dempts', FALSE);
```

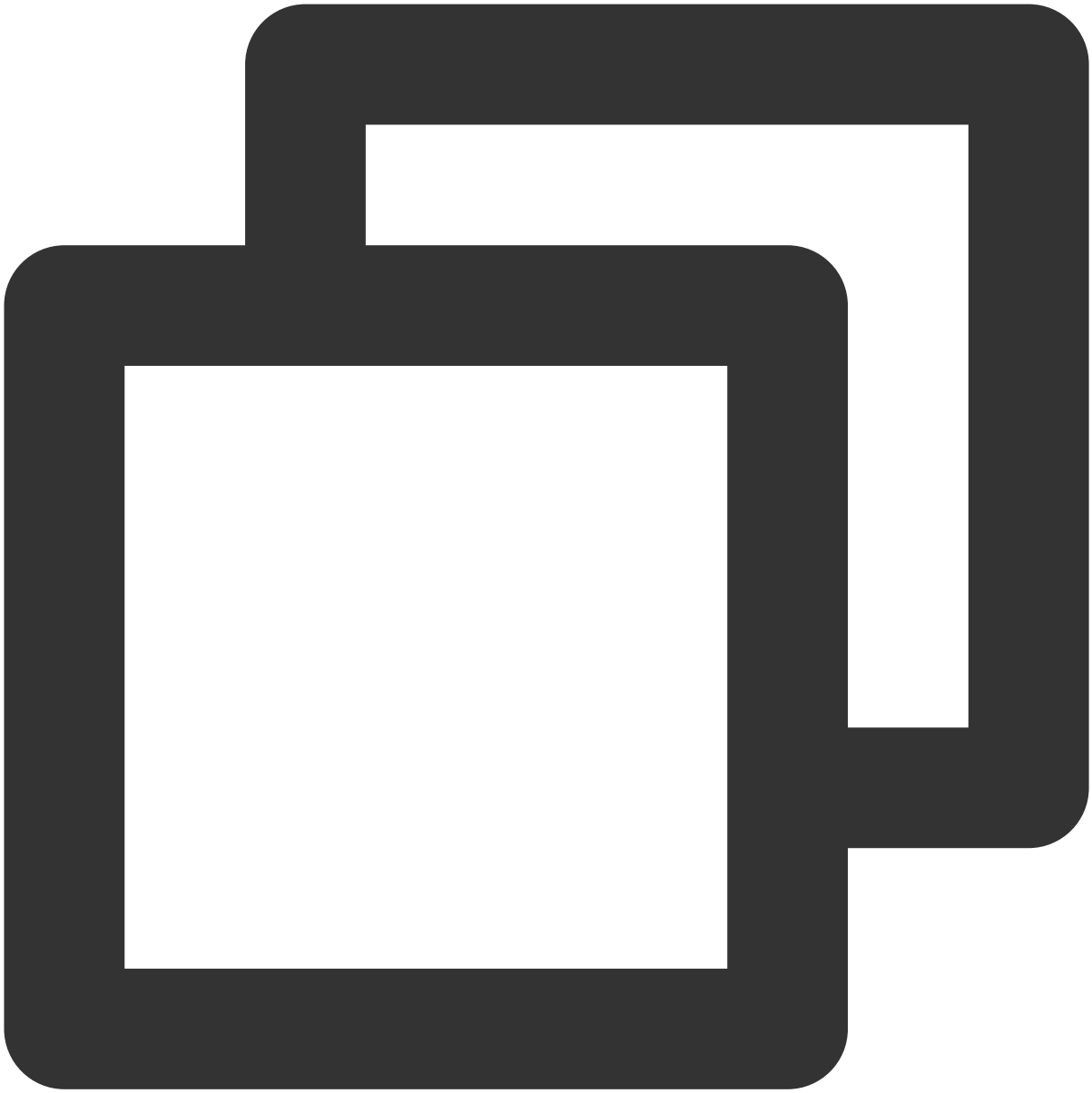
ancestors_of

获取快照的血缘信息。



```
CALL `Catalog`.`system`.ancestors_of(table_name, [snapshot_id]);
```

示例：



```
CALL `DataLakeCatalog`.`system`.ancestors_of('validation.dempts');
CALL `DataLakeCatalog`.`system`.ancestors_of('validation.dempts', 1);
```

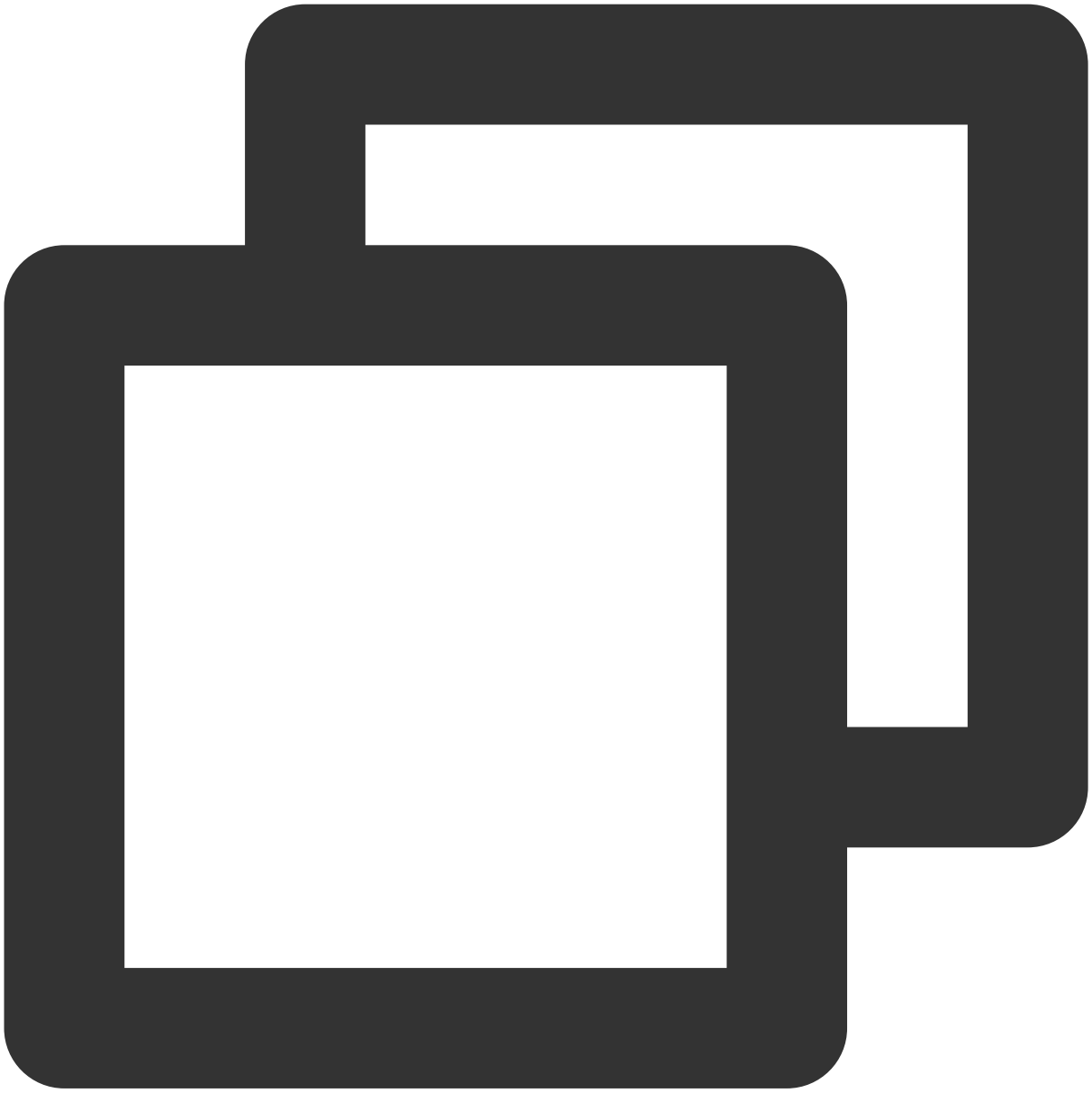
数据表迁移管理

注意

原表必须为 Hive 表或 Spark 表。

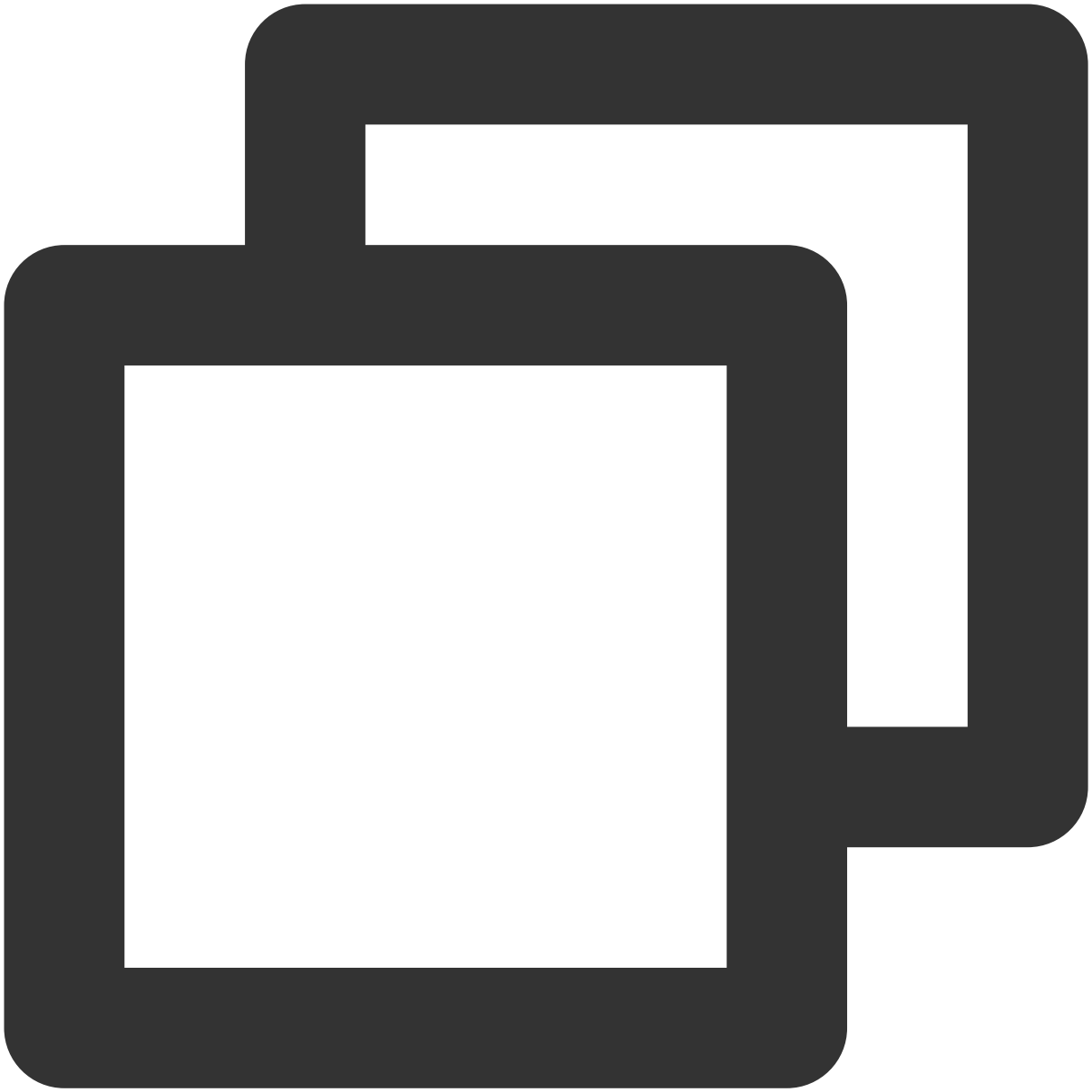
snapshot

基于原始表创建轻量级的临时表，临时表直接复用原始表快照。



```
CALL `Catalog`.`system`.snapshot(source_table, table, [location], [properties]);
```

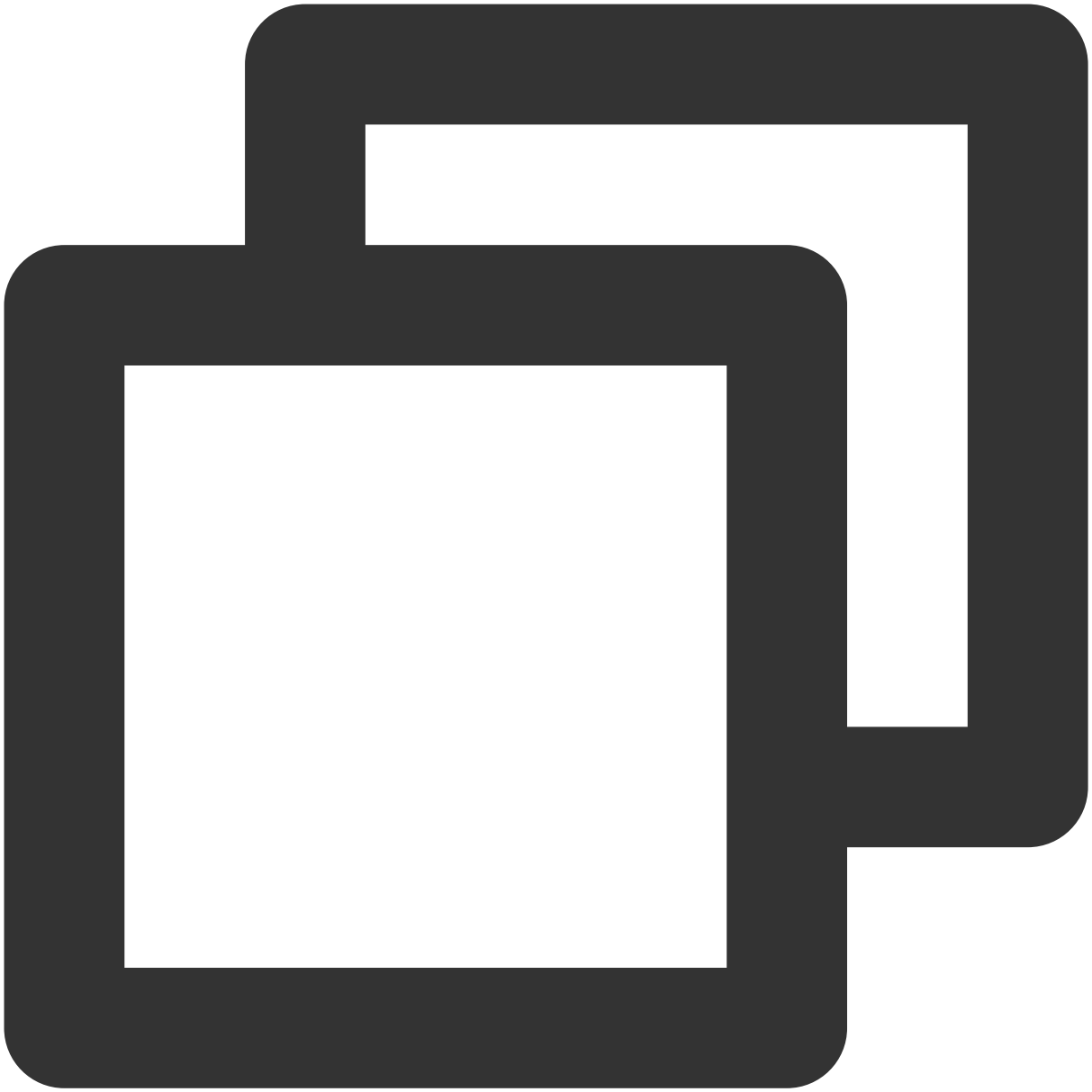
示例：



```
CALL `DataLakeCatalog`.`system`.snapshot('validation.table_01', 'validation.snap');
CALL `DataLakeCatalog`.`system`.snapshot('validation.table_01', 'validation.snap2',
```

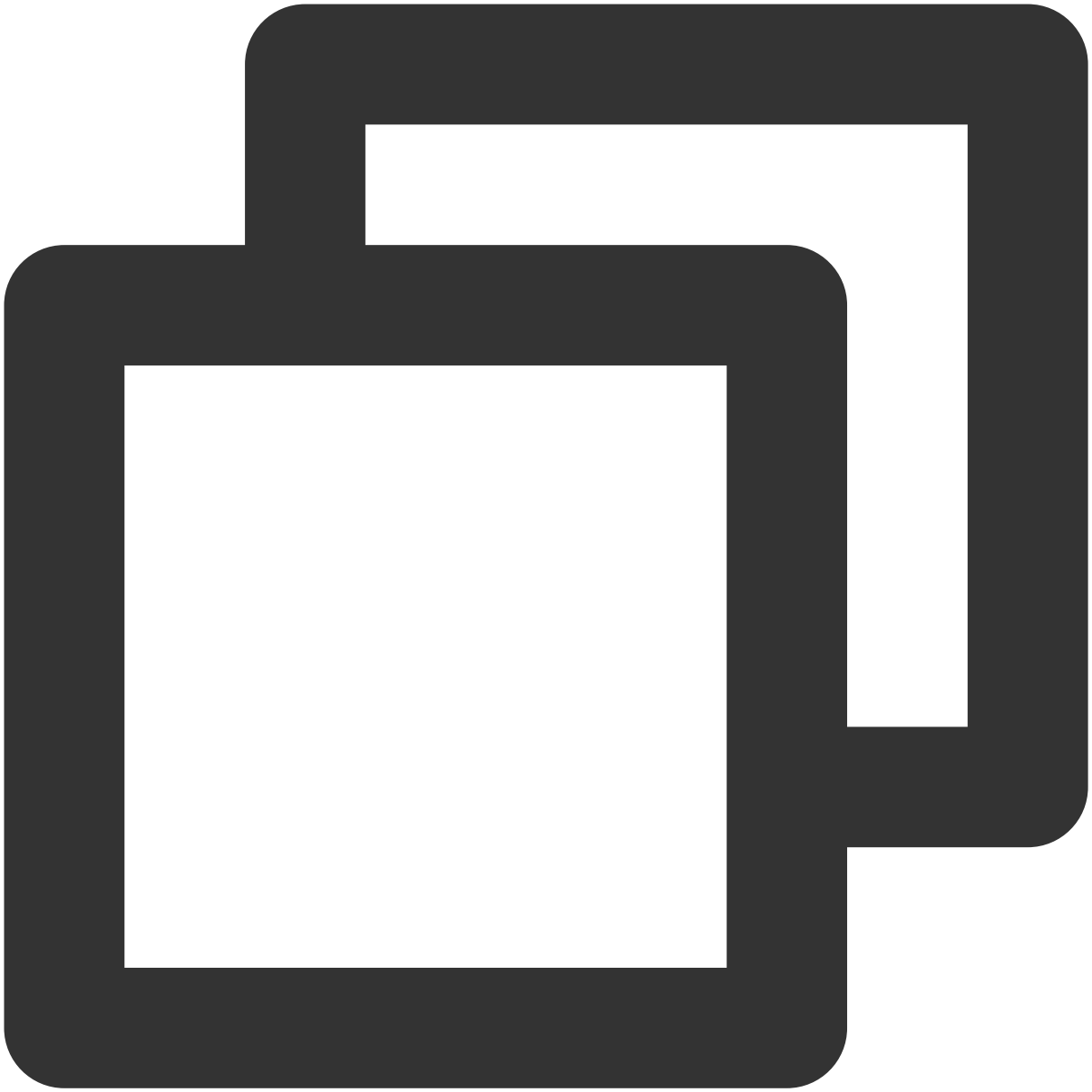
migrate

更新替换表属性。



```
CALL `Catalog`.`system`.migrate(table, [properties]);
```

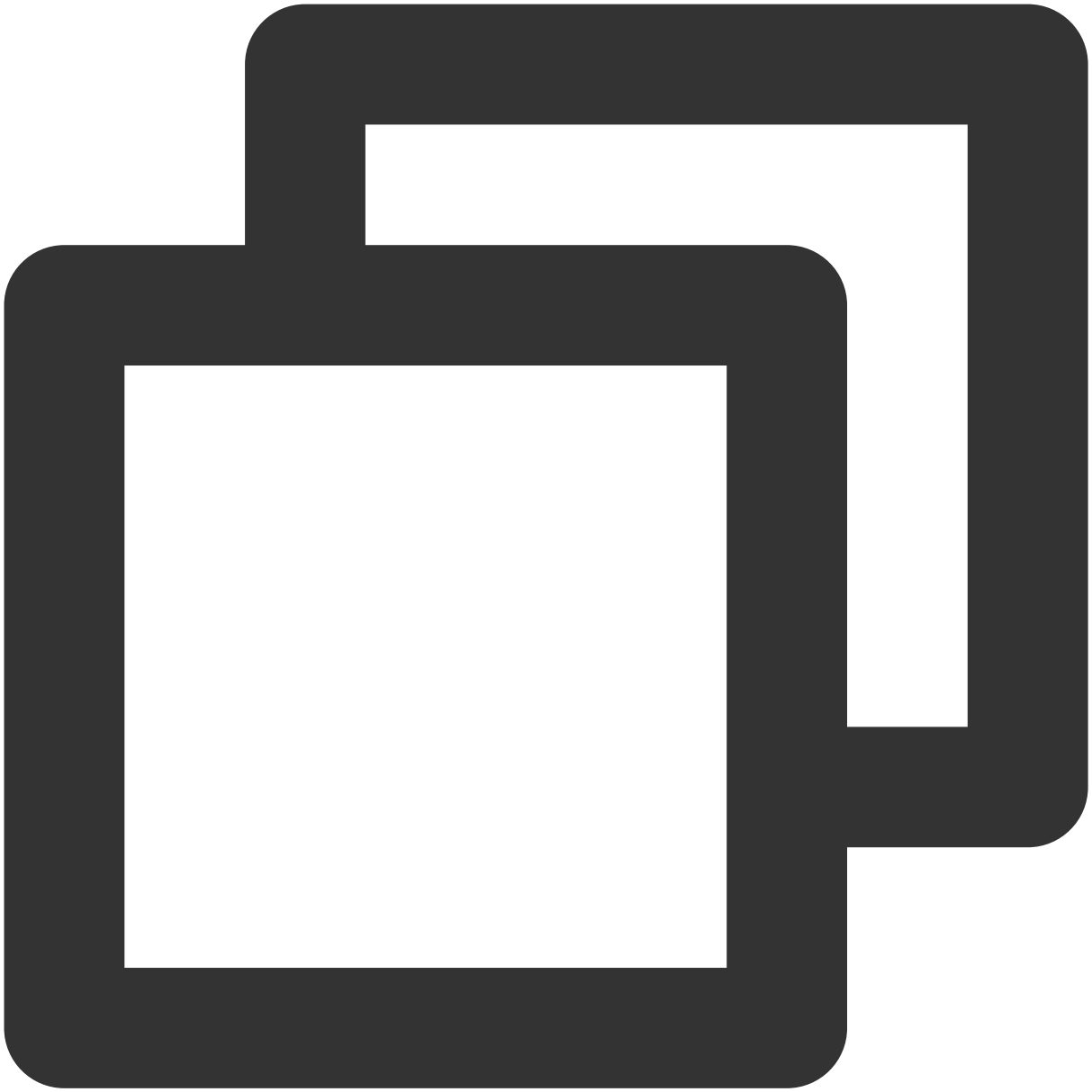
示例：



```
CALL `DataLakeCatalog`.`system`.migrate('validation.table_01');
CALL `DataLakeCatalog`.`system`.migrate('validation.table_01', map('data', 'name'))
```

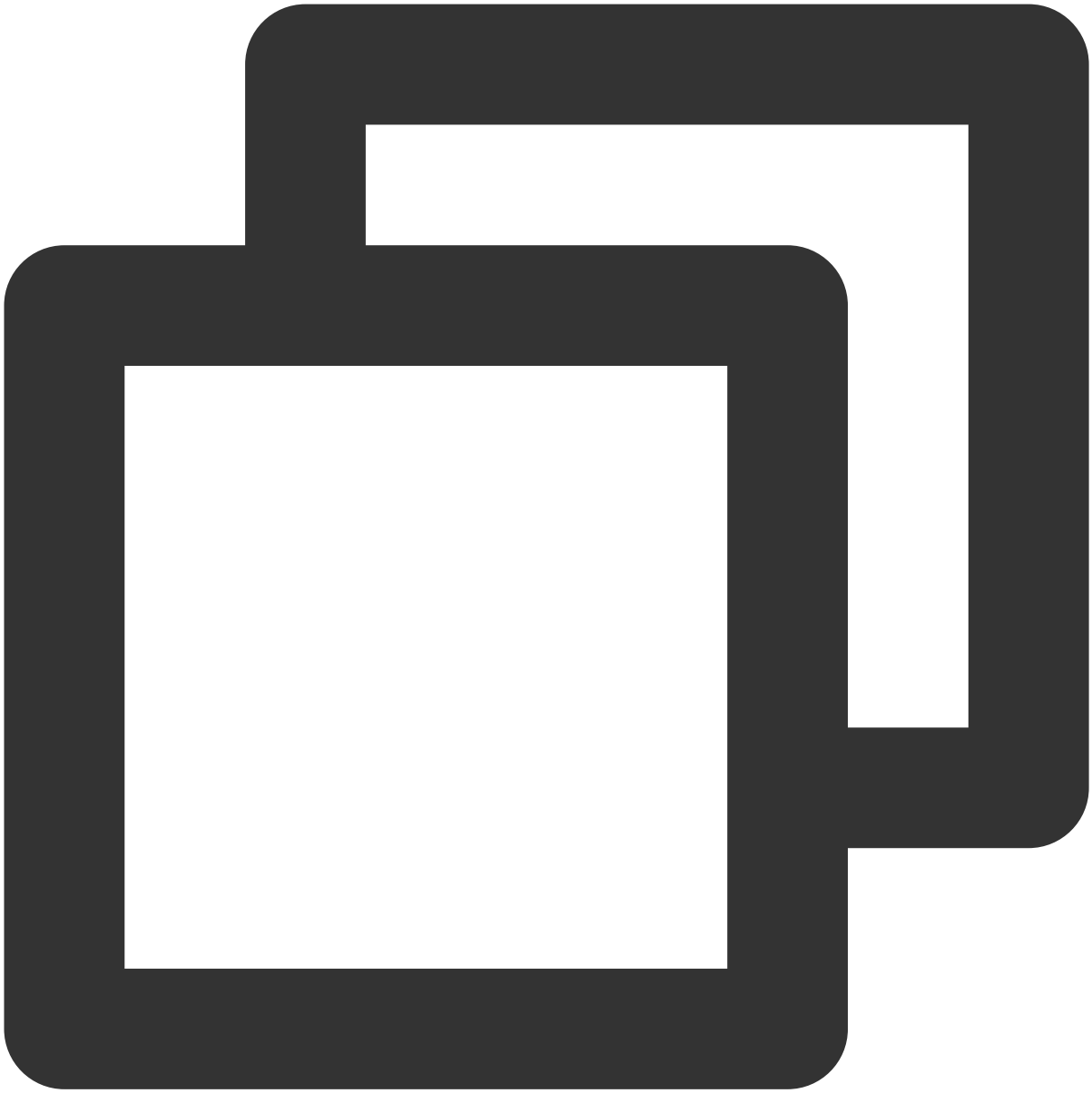
add_files

直接从 hive 中加载数据文件，可指定数据文件到指定分区。



```
CALL `Catalog`.`system`.add_files(table, source_table, [partition_filter]);
```

示例：



```
CALL `DataLakeCatalog`.`system`.add_files(`table`=>'validation.table_02', `source_t
CALL `DataLakeCatalog`.`system`.add_files(`table`=>'validation.table_02', `source_t
```

Iceberg 外部表与原生表语法差异

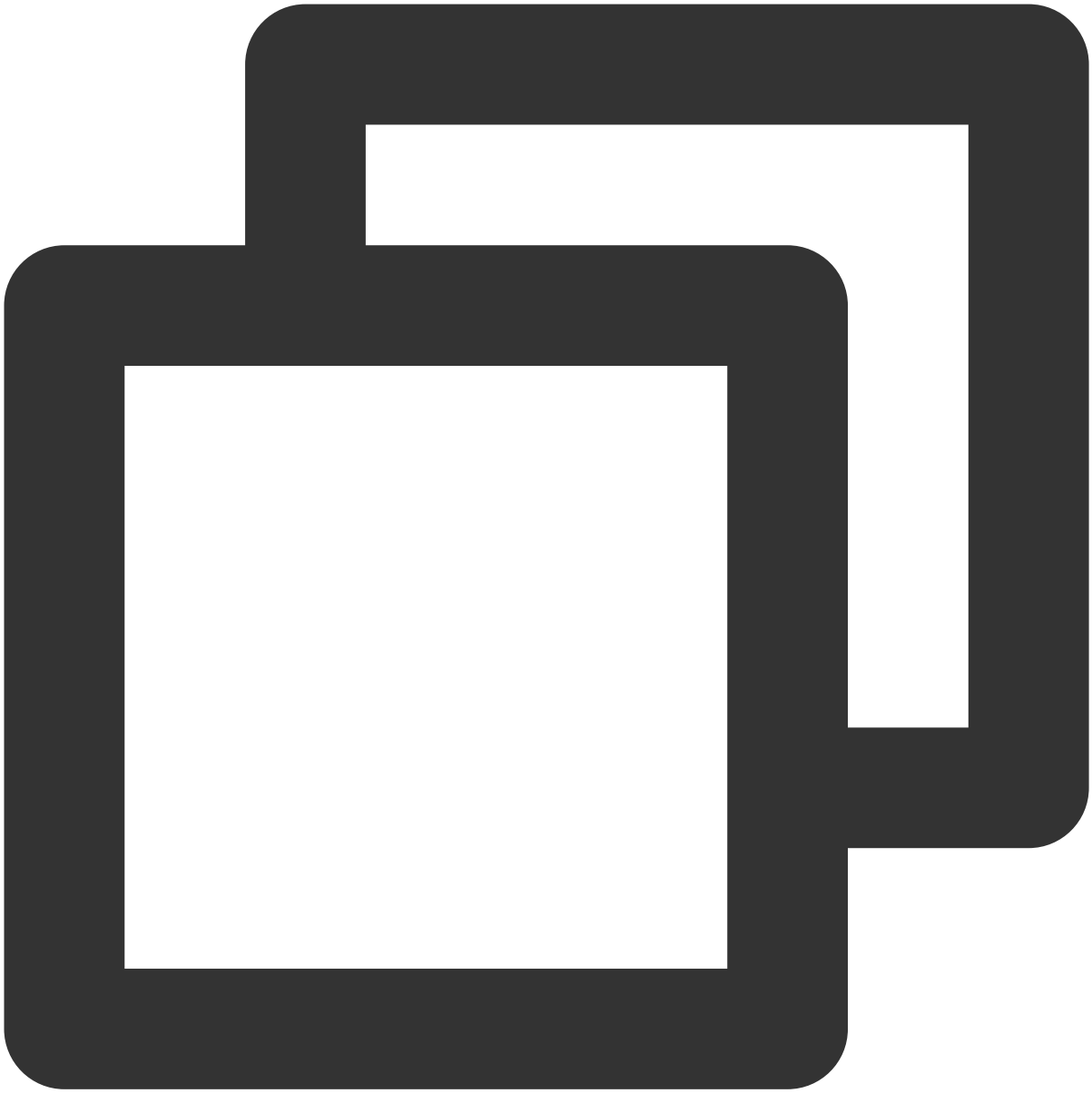
最近更新时间：2024-08-07 17:29:32

数据湖计算 DLC 使用的 Iceberg 语法版本为0.13.1，详情语法说明可参考 Iceberg 官网文档。
当您在使用 Iceberg 外部表时，SQL 语法与 Iceberg 原生表存在以下差异。

CREATE TABLE

原生表

语法



```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
    ( col_name[:] col_type [ COMMENT col_comment ], ... )
[ COMMENT table_comment ]
[ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
```

示例

```
CREATE TABLE dempts(
    id bigint COMMENT 'id number',
    num int,
    eno float,
    dno double,
    cno decimal(9,3),
    flag  boolean,
    data string,
    ts_year timestamp,
    date_month date,
    bno binary,
```

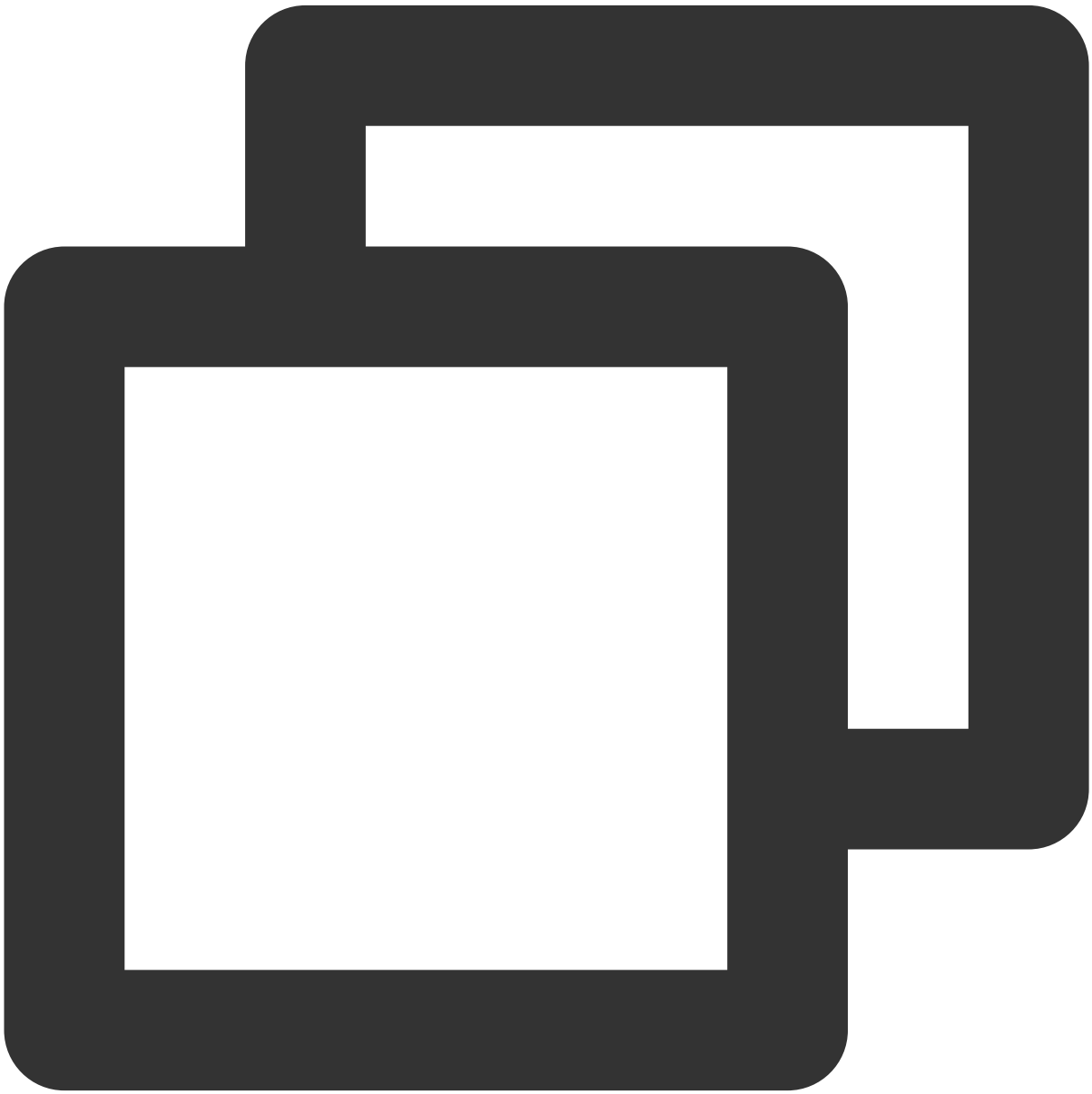
```
    point struct<x: double, y: double>,
    points array<struct<x: double, y: double>>,
    pointmaps map<struct<x: int>, struct<a: int>>
    )
COMMENT 'table documentation'
PARTITIONED BY (bucket(16,id), years(ts_year), months(date_month), identity(bno),
```

外部表

语法

```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
    ( col_name[:] col_type [ COMMENT col_comment ], ... )
USING iceberg
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
    [ LOCATION path ]
    [ TBLPROPERTIES ( property_name=property_value, ... ) ]
```

示例



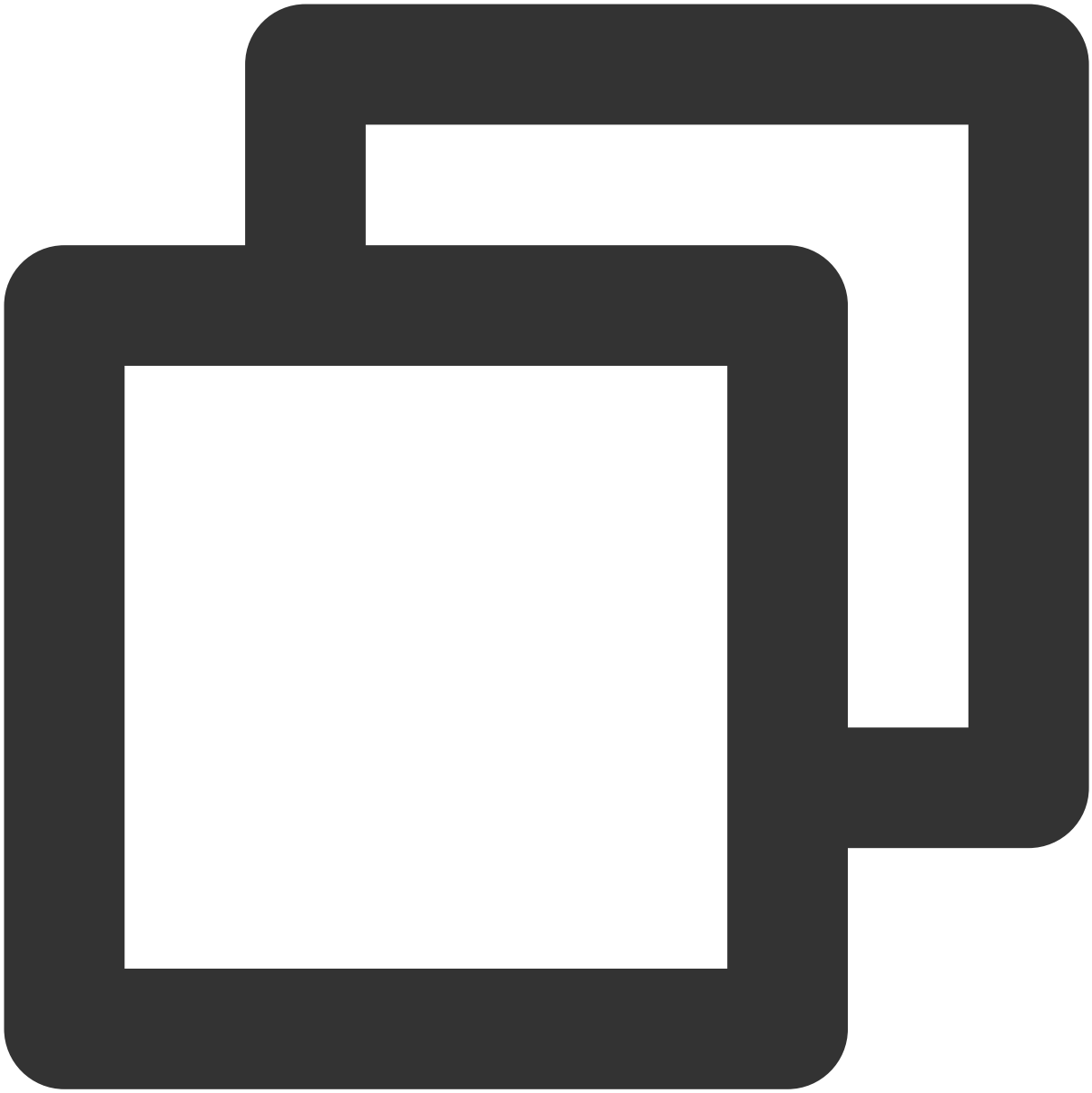
```
CREATE TABLE dempts(
```

```
    id bigint COMMENT 'id number',
    num int,
    eno float,
    dno double,
    cno decimal(9,3),
    flag  boolean,
    data string,
    ts_year timestamp,
    date_month date,
    bno binary,
    point struct<x: double, y: double>,
    points array<struct<x: double, y: double>>,
    pointmaps map<struct<x: int>, struct<a: int>>
    )
USING iceberg
COMMENT 'table documentation'
PARTITIONED BY (bucket(16,id), years(ts_year), months(date_month), identity(bno),
LOCATION 'cosn://rickytest-1305424723/channing-test/loc'
TBLPROPERTIES ('write.format.default'='orc');
```

CREATE TABLE AS SELECT

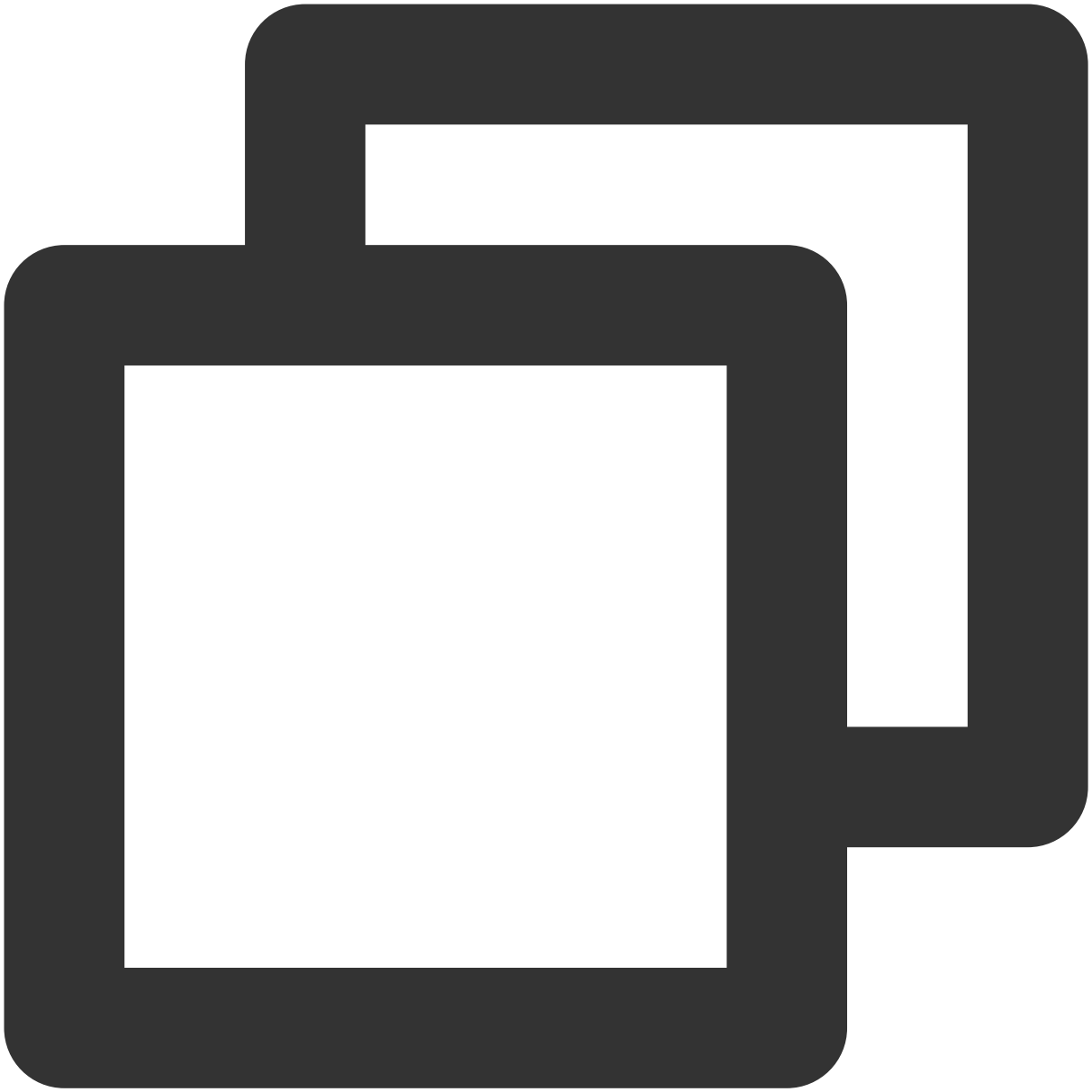
原生表

语法



```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
AS select_statement
```

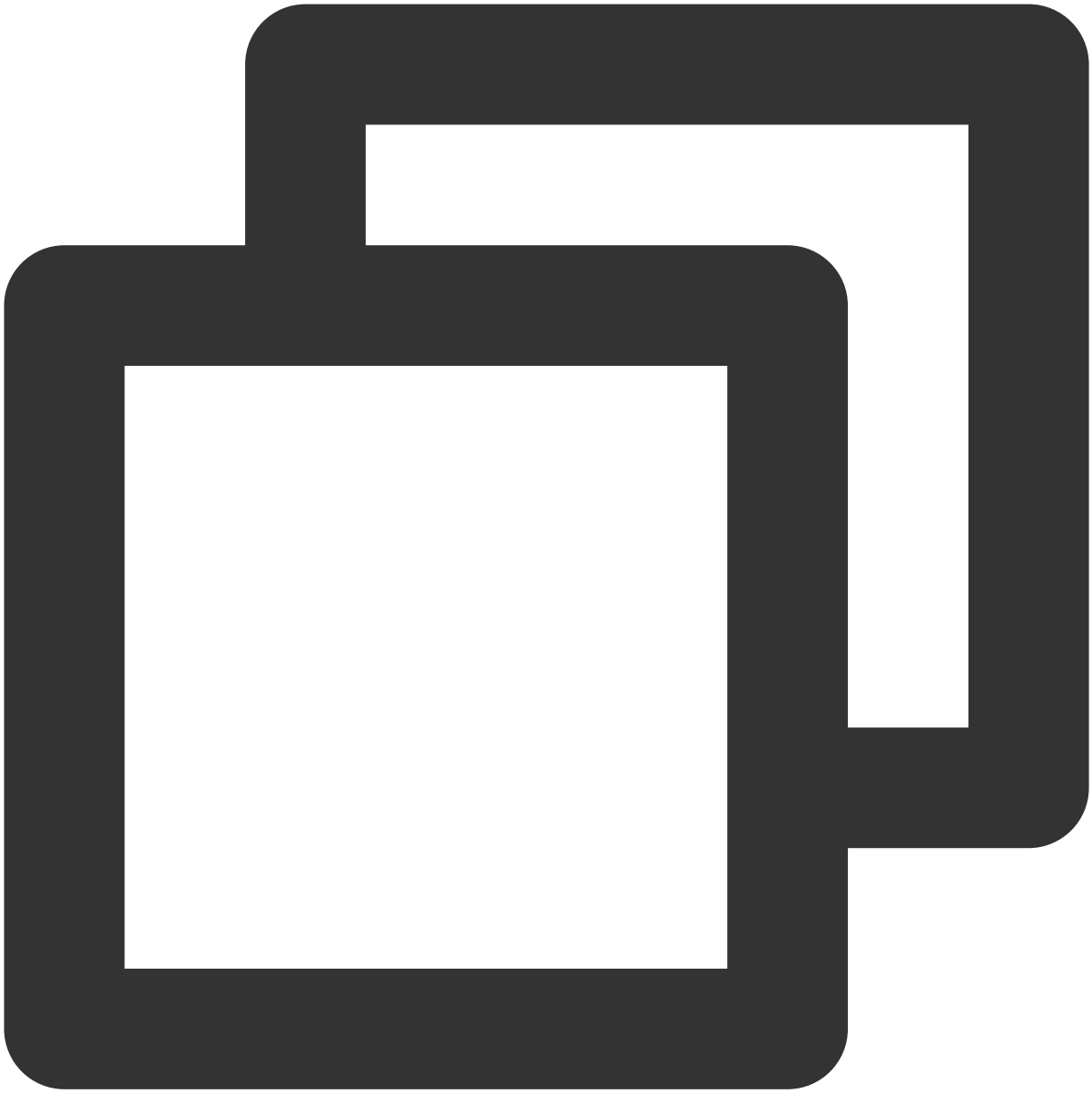
示例



```
CREATE TABLE IF NOT EXISTS dempts_copy
COMMENT 'table create as select'
PARTITIONED BY (eno, dno)
AS SELECT * from dempts;
```

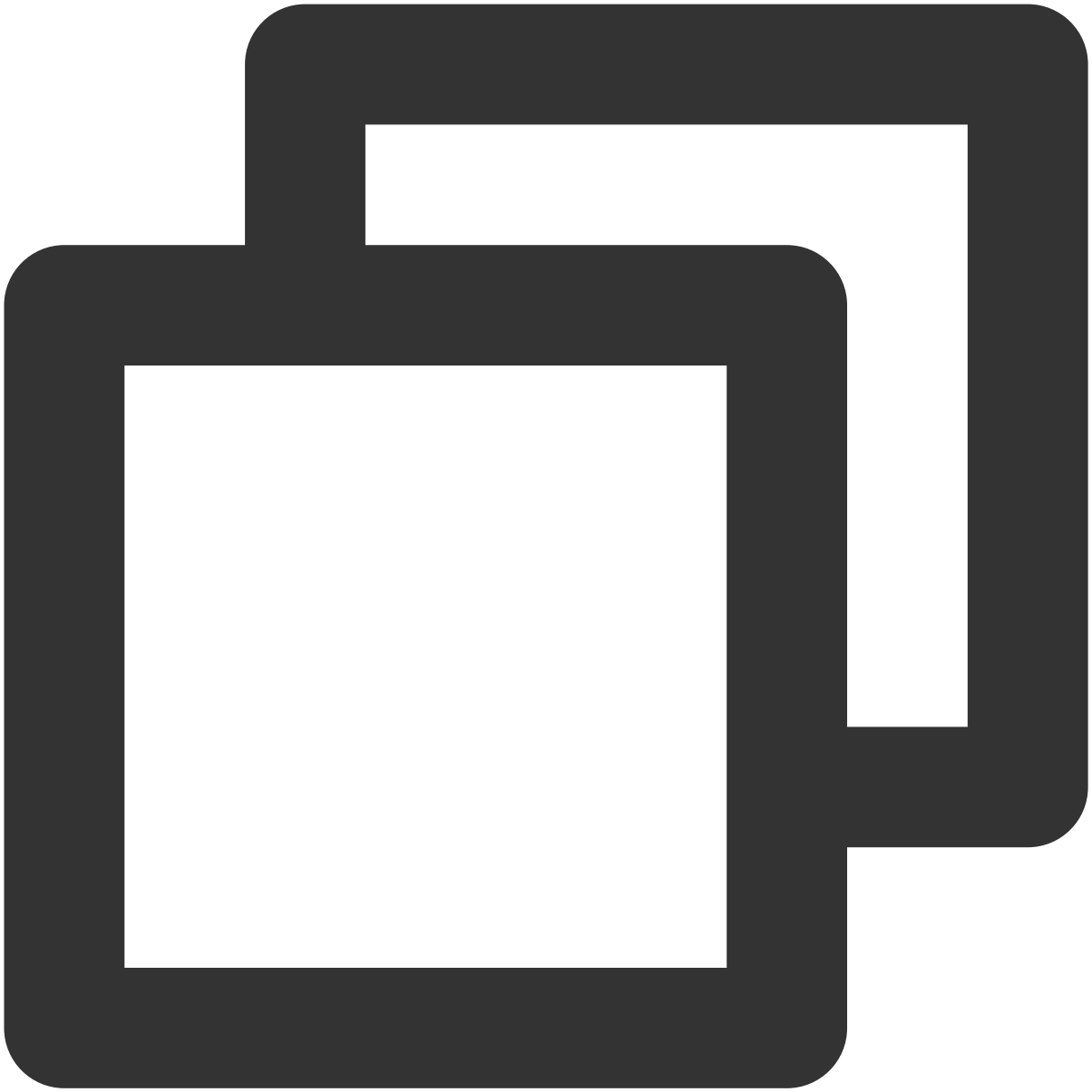
外部表

语法



```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
USING iceberg
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
    [ LOCATION path ]
    [ TBLPROPERTIES ( property_name=property_value, ... ) ]
AS select_statement
```

示例

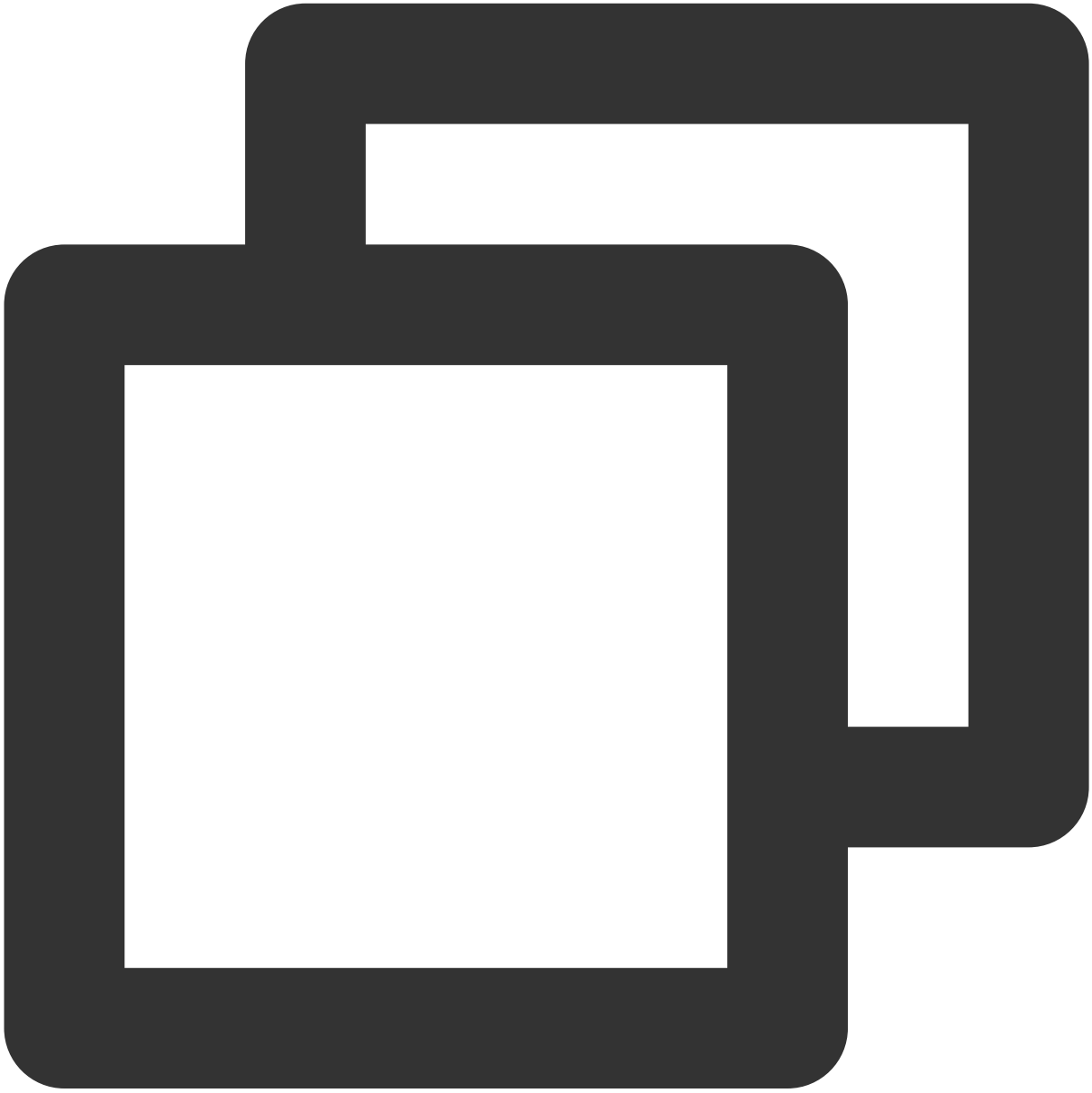


```
CREATE TABLE dempts_copy
USING iceberg
COMMENT 'table create as select'
PARTITIONED BY (eno, dno)
LOCATION 'cosn://rickytest-1305424723/channing-test/loc'
TBLPROPERTIES ('write.format.default'='avro')
AS SELECT * from dempts;
```

REPLACE TABLE AS SELECT

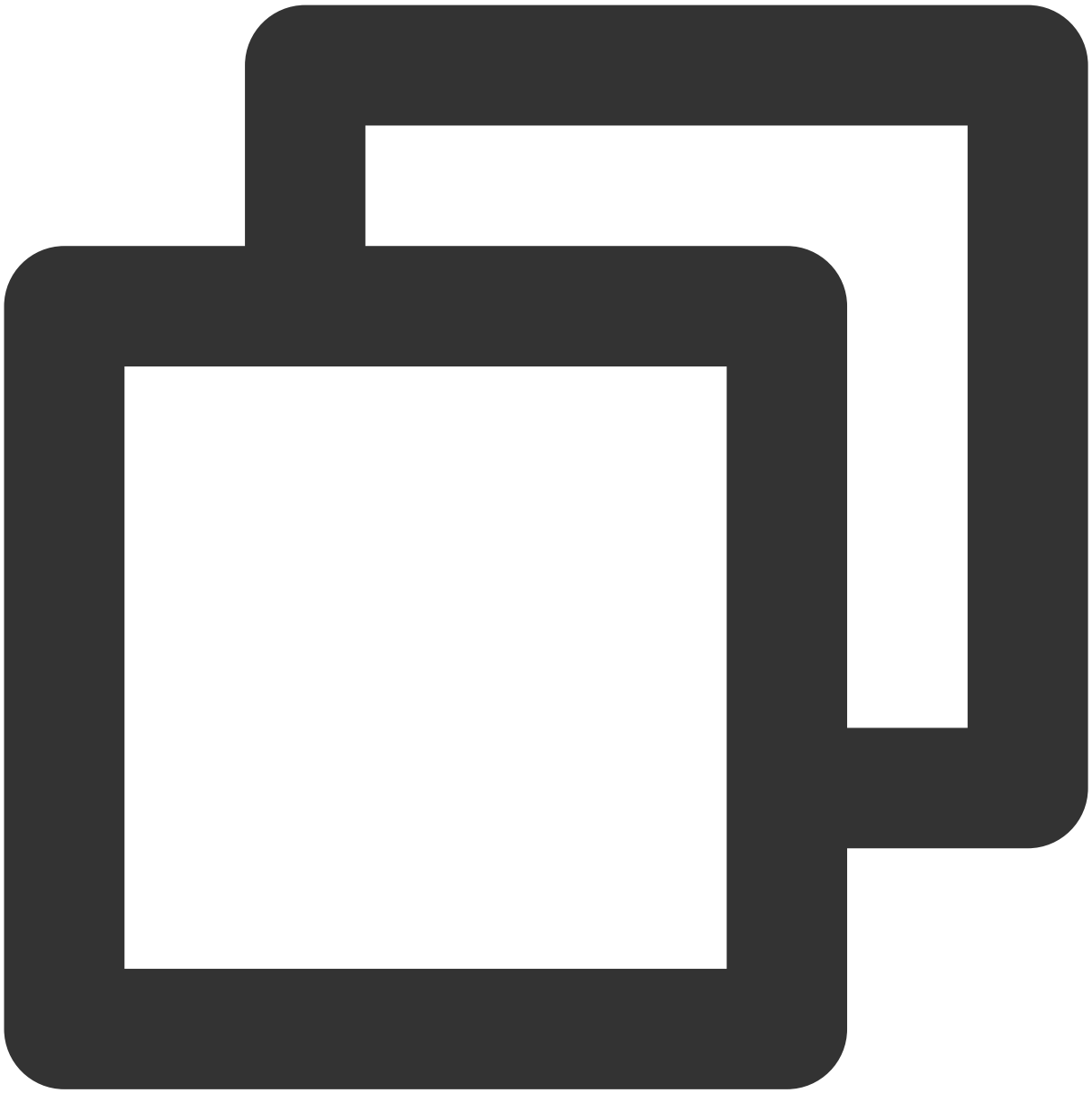
原生表

语法



```
CREATE OR REPLACE TABLE table_identifier
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
AS select_statement
```

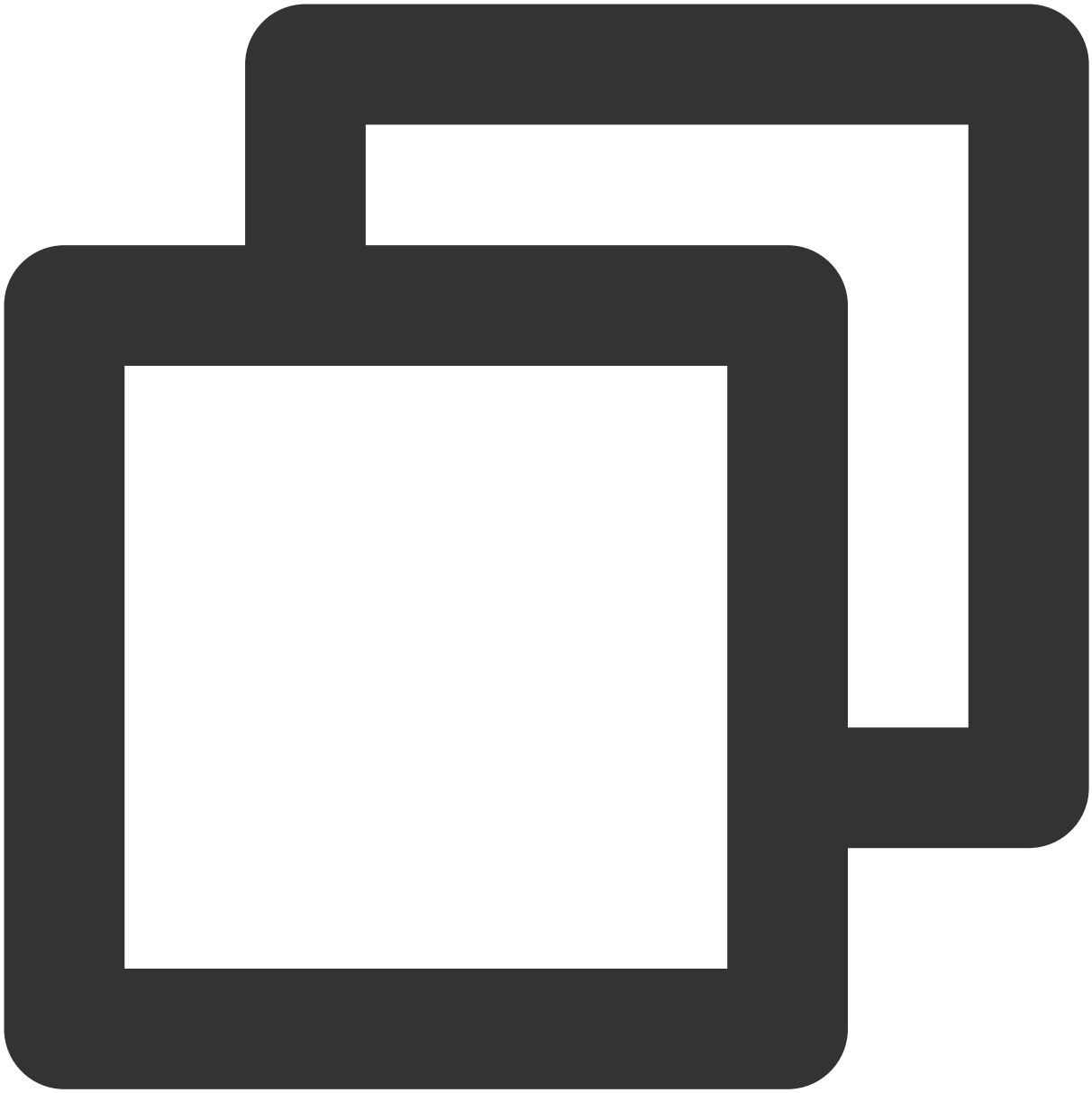
示例



```
CREATE OR REPLACE TABLE dempts_replace
COMMENT 'table create as replace'
PARTITIONED BY (eno, bucket(10, num))
AS SELECT * from dempts;
```

外部表

语法



```
CREATE [OR REPLACE] TABLE table_identifier
USING iceberg
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
    [ LOCATION path ]
    [ TBLPROPERTIES ( property_name=property_value, ... ) ]
AS select_statement
```

示例

```
CREATE OR REPLACE TABLE dempts_replace
USING iceberg
COMMENT 'table create as replace'
PARTITIONED BY (eno, dno)
LOCATION 'cosn://rickytest-1305424723/channing-test/loc'
TBLPROPERTIES ('write.format.default'='avro')
AS SELECT * from dempts;
```

Procedure

注意

迁移原表必须为 Hive 表或 Spark 表。

原生表

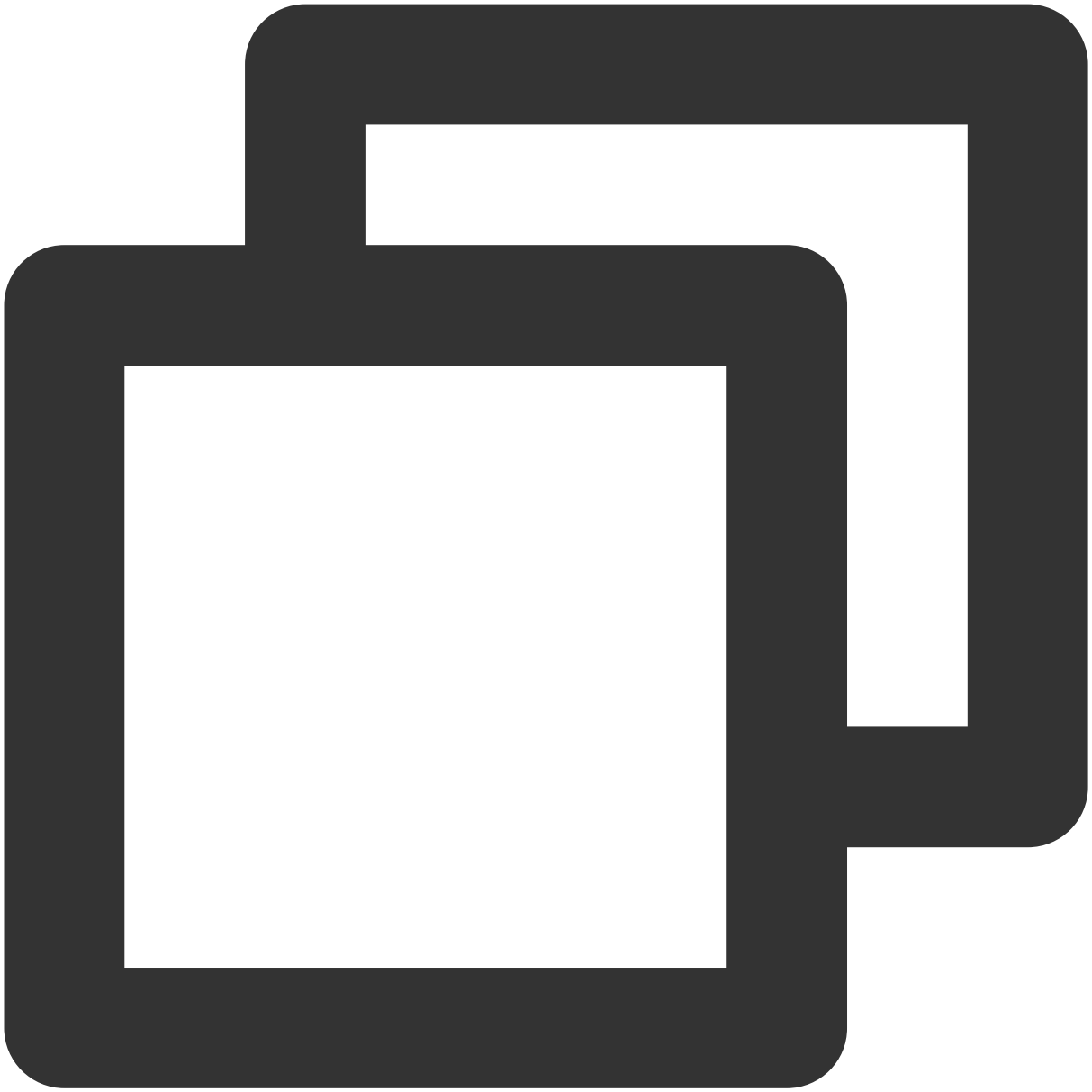
暂不支持。

外部表

snapshot

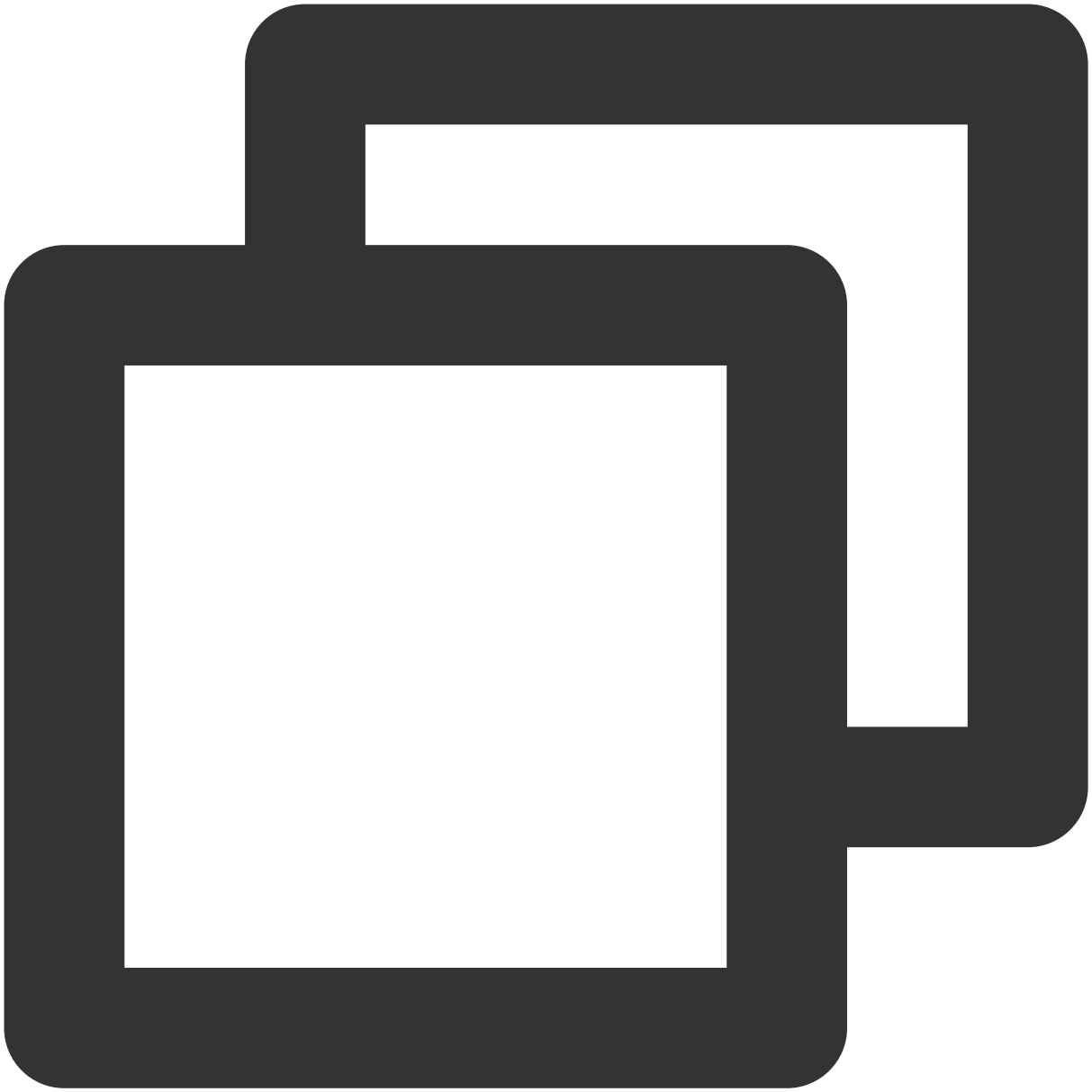
基于原始表创建轻量级的临时表，临时表直接复用原始表快照。

语法



```
CALL `Catalog`.`system`.snapshot(source_table, table, [location], [properties]);
```

示例

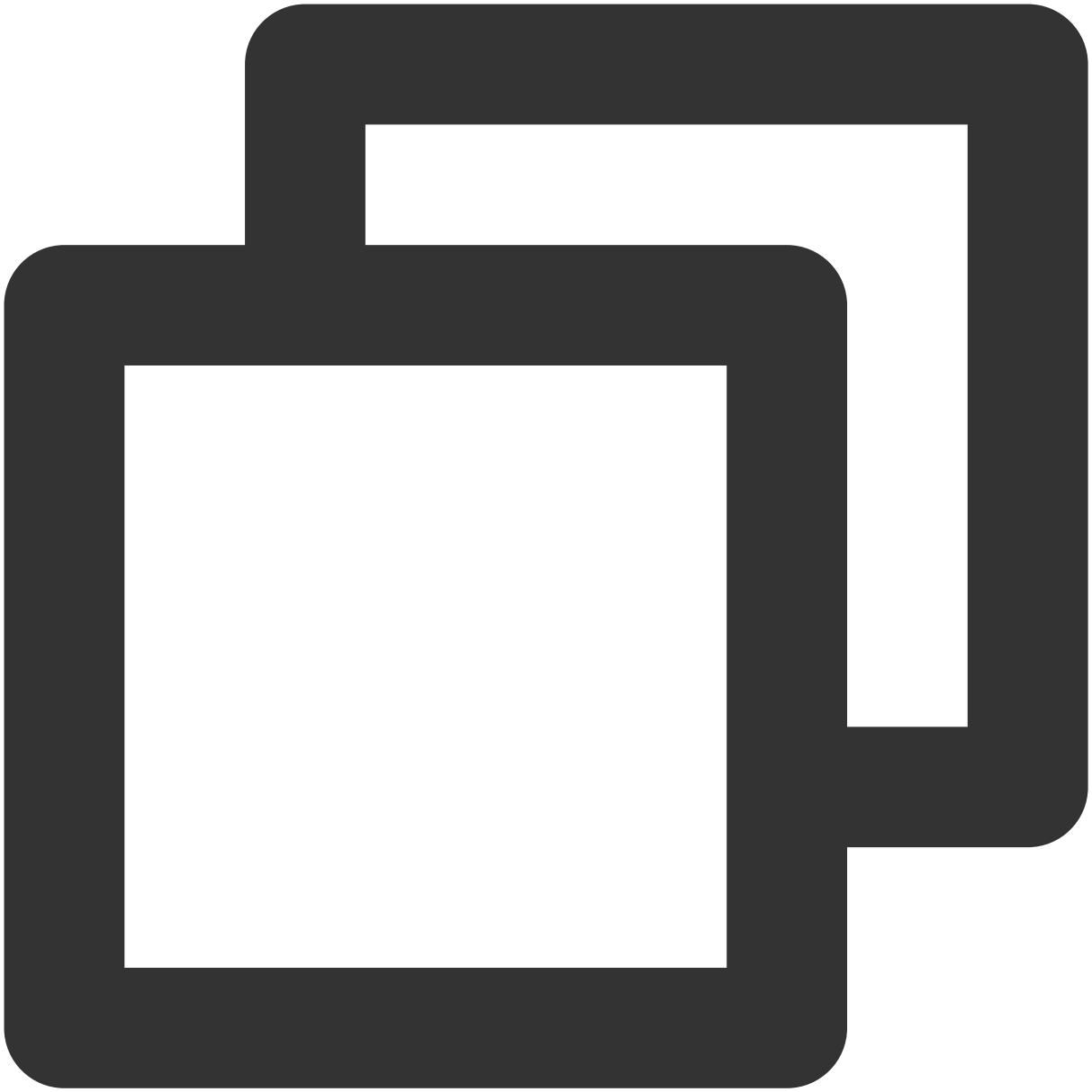


```
CALL `DataLakeCatalog`.`system`.snapshot('validation.table_01', 'validation.snap');
CALL `DataLakeCatalog`.`system`.snapshot('validation.table_01', 'validation.snap2',
```

call

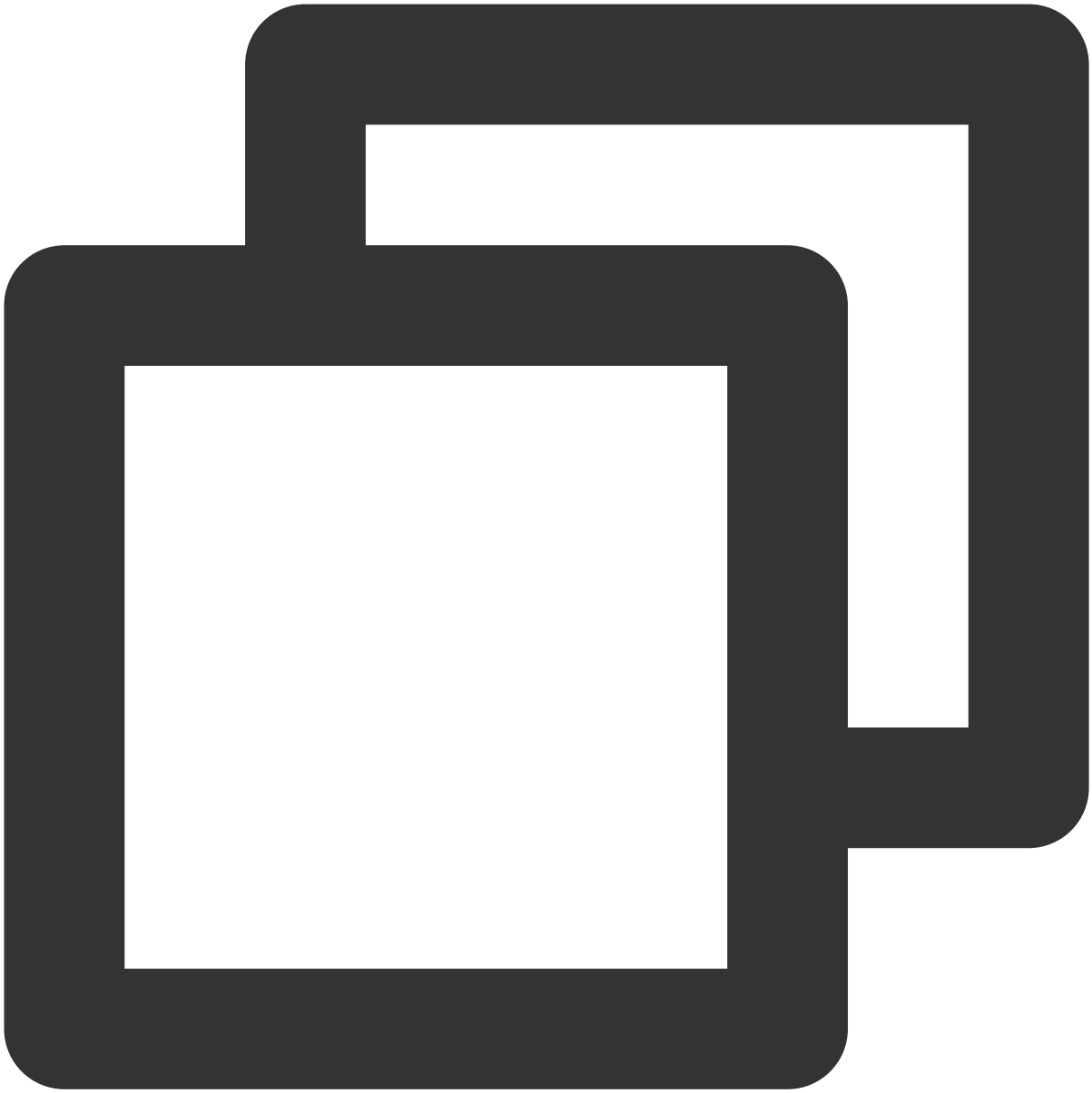
更新替换表属性。

语法



```
CALL `Catalog`.`system`.migrate(table, [properties]);
```

示例

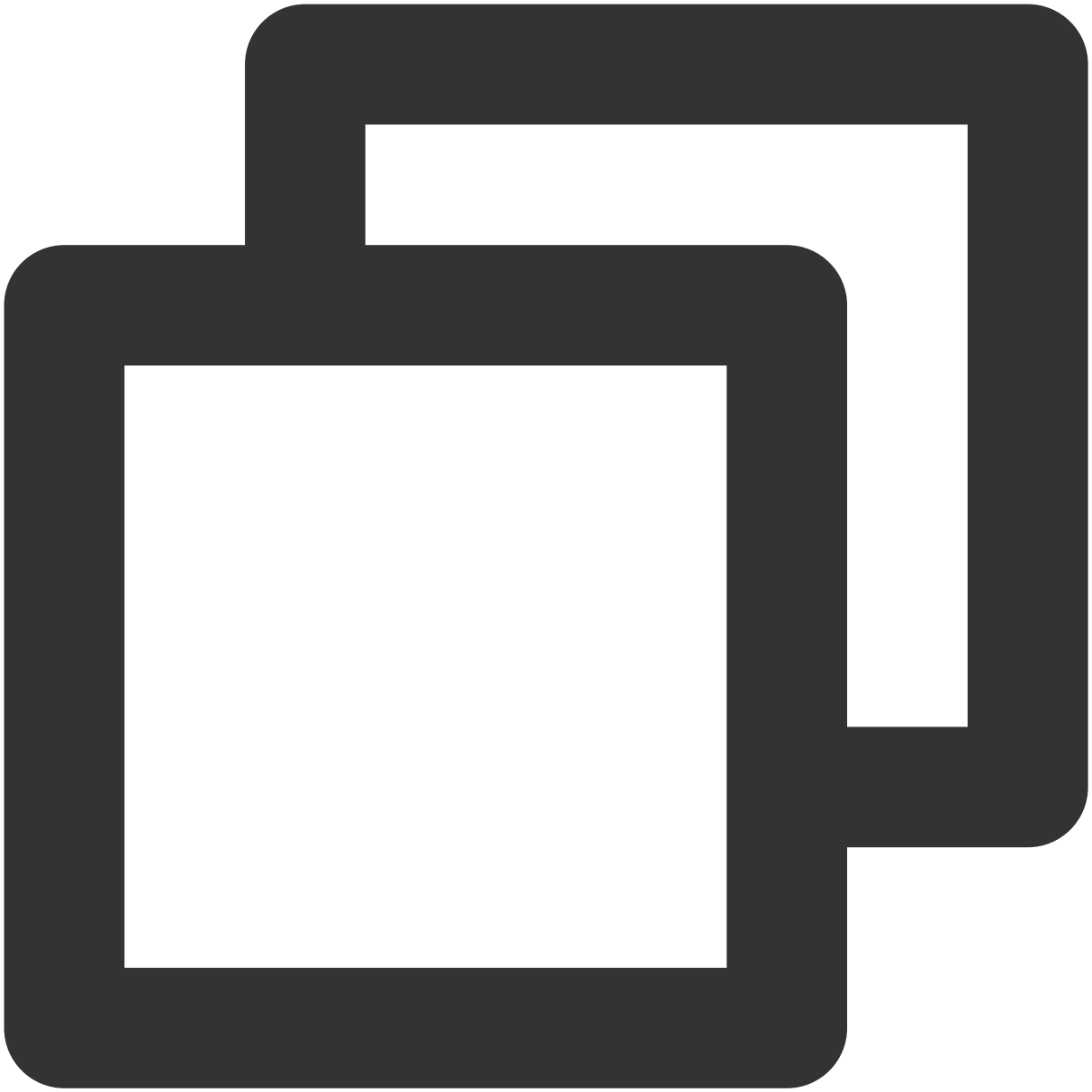


```
CALL `DataLakeCatalog`.`system`.migrate('validation.table_01');
CALL `DataLakeCatalog`.`system`.migrate('validation.table_01', map('data', 'name'))
```

add_files

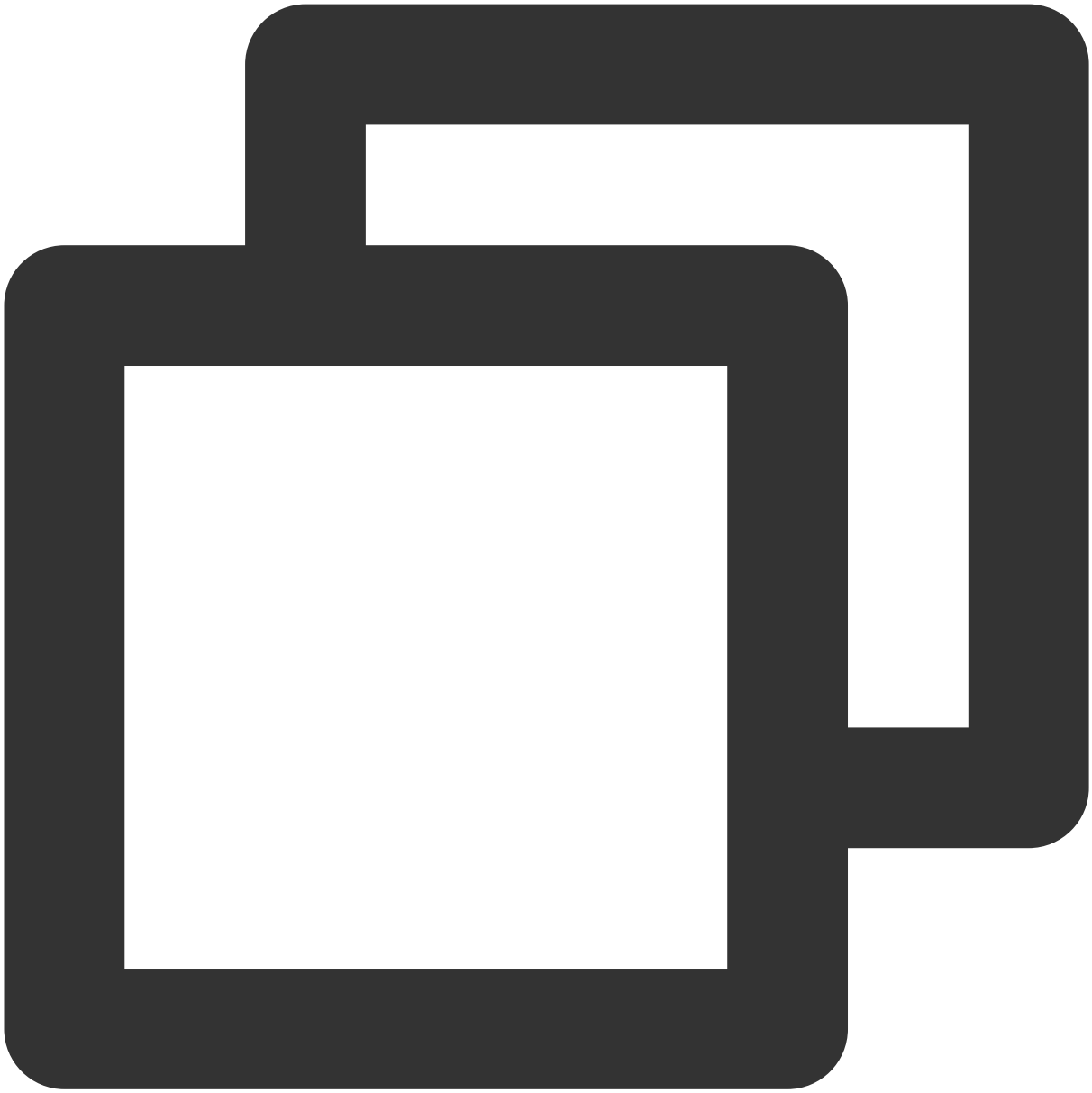
直接从 hive 中加载数据文件，可指定数据文件到指定分区。

语法



```
CALL `Catalog`.`system`.add_files(table, source_table, [partition_filter]);
```

示例



```
CALL `DataLakeCatalog`.`system`.add_files(`table`=>'validation.table_02', `source_t
CALL `DataLakeCatalog`.`system`.add_files(`table`=>'validation.table_02', `source_t
```

物化视图语法

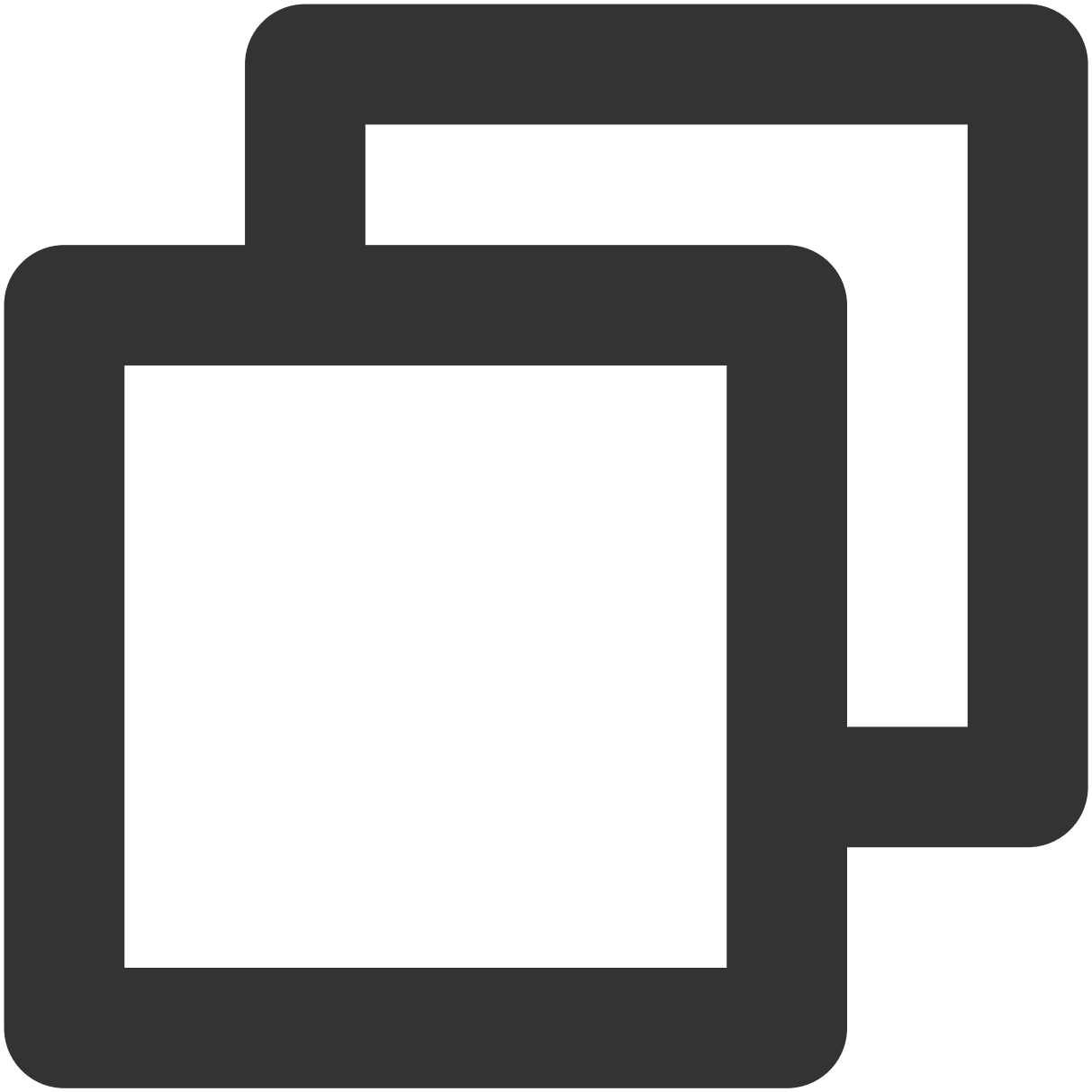
最近更新时间：2024-08-07 17:29:48

说明：

以下为物化视图支持的语法，仅支持 Presto 和 SparkSQL 引擎。详细使用指南可以参考物化视图文档。

CREATE MATERIALIZED

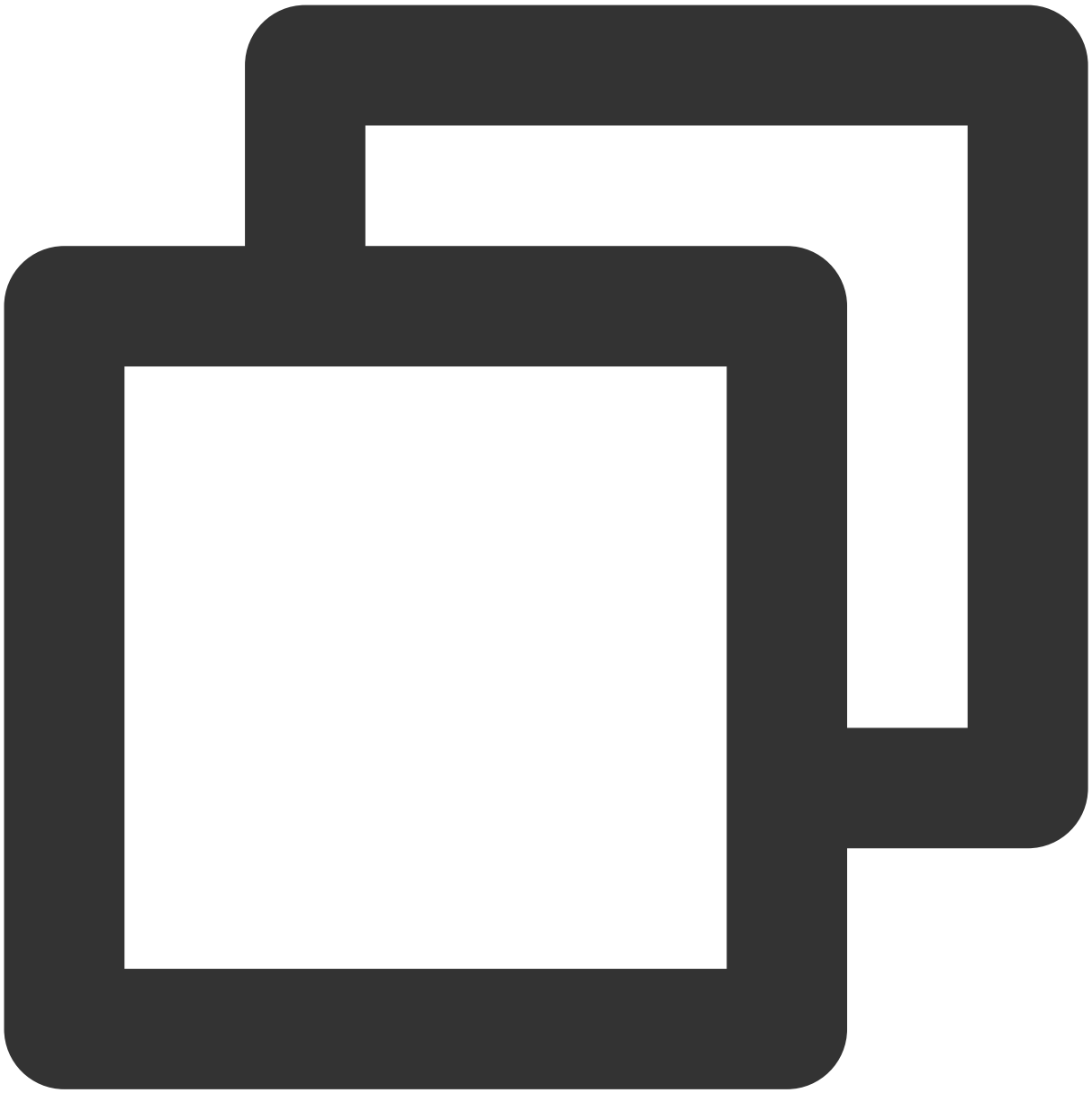
创建物化视图：



```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] mv_identifier
    [ENABLE | DISABLE REWRITE]
    [COMMENT mv_comment]
    [WITH META LINK]
    [TBLPROPERTIES (key1=value1, key2=value2,...)]
    [AS] select_query
```

DROP MATERIALIZED

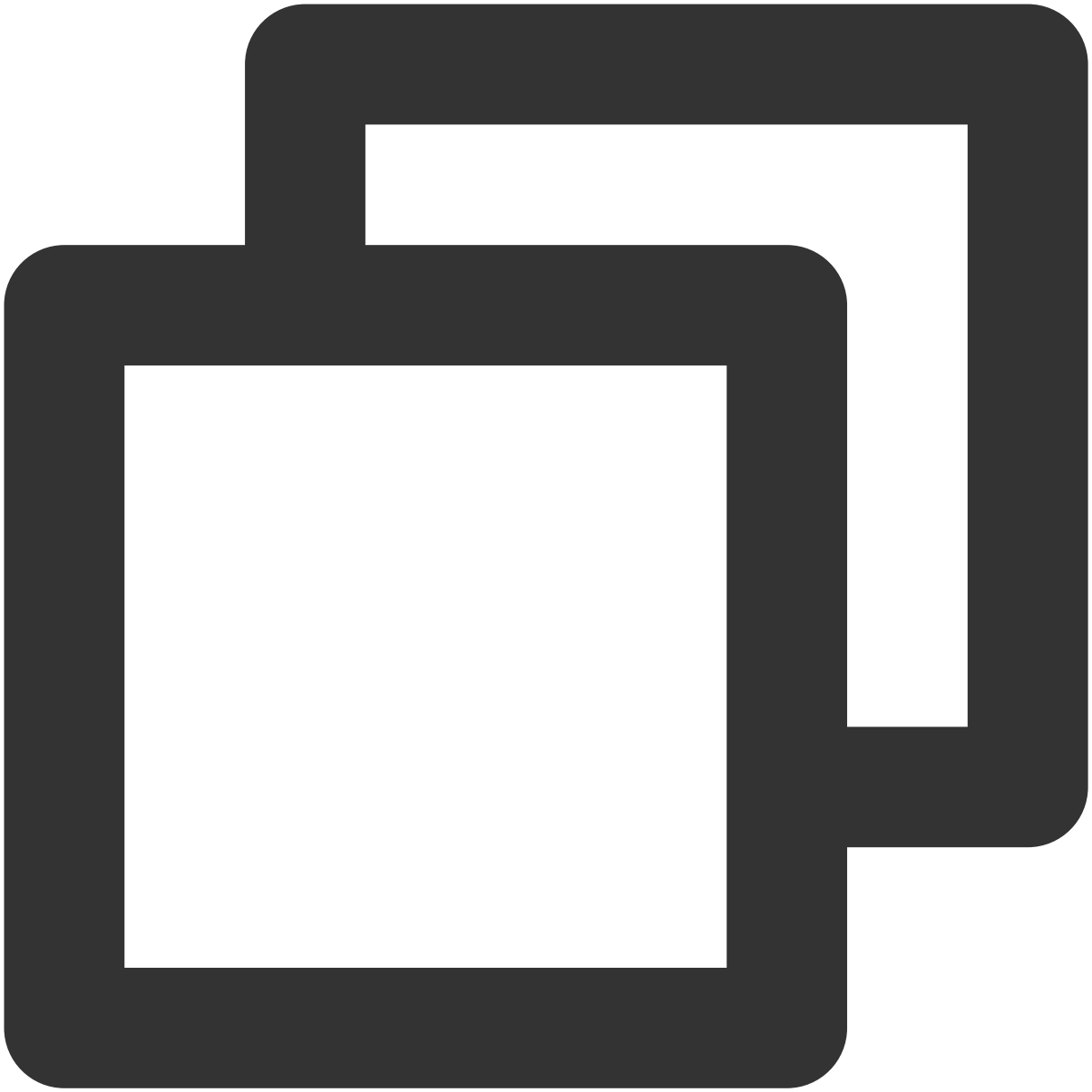
删除物化视图：



```
DROP MATERIALIZED VIEW [IF EXISTS] mv_identifier
```

DESCRIBE MATERIALIZED

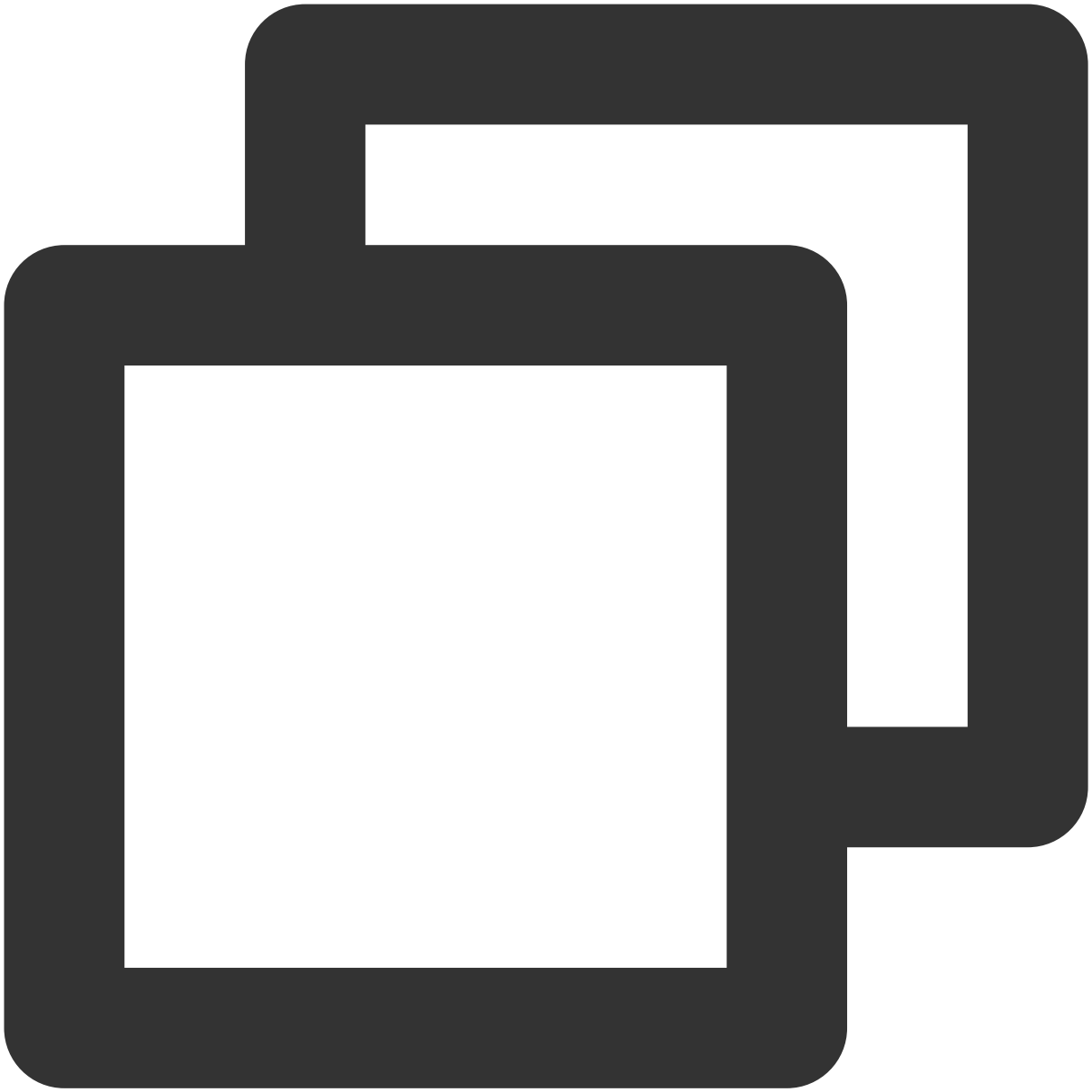
描述物化视图：



```
[DESCRIBE | DESC] MATERIALIZED VIEW mv_identifier
```

REFRESH MATERIALIZED

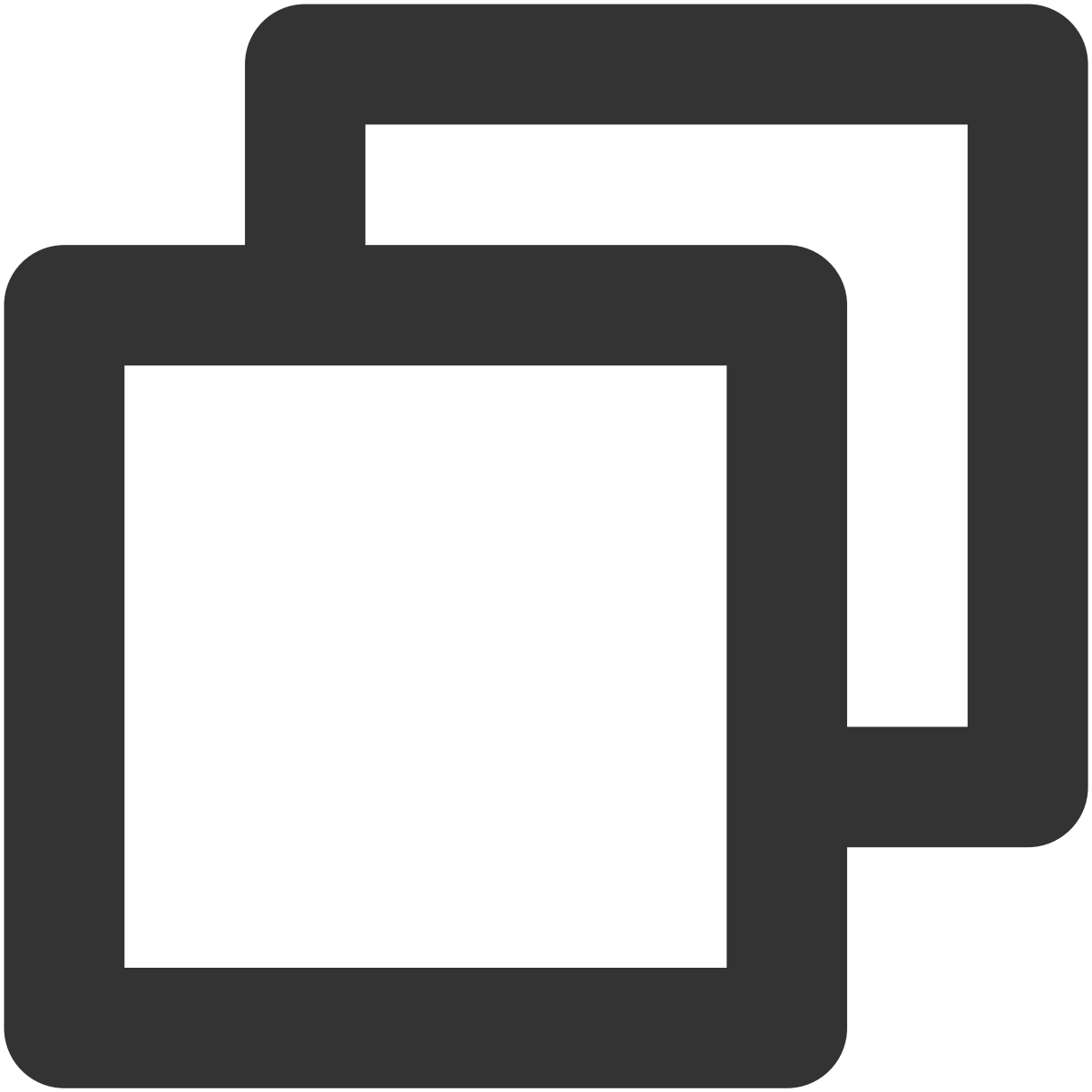
刷新物化视图：



```
REFRESH MATERIALIZED VIEW mv_identifier
```

SHOW MATERIALIZED VIEW JOBS

显示物化视图的执行任务列表，可以了解到物化视图的刷新历史和状态。



```
SHOW MATERIALIZED VIEW JOBS IN mv_identifier
```

SQL 隐式转换

最近更新时间：2024-08-07 17:30:08

为了便于 SQL 编写，快速从 Hive/Spark 语法切换到 DLC 统一的 SQL，我们提供了 SQL 的隐式转化能力，转化列表如下：

I：implicit cast，E: explicit cast，--: not allowed

From-To	null	boolean	tinyint	smallint	int	bigint	decimal	float/real	double	inte
null	I	I	I	I	I	I	I	I	I	I
boolean	--	I	E	E	E	E	E	E	E	--
tinyint	--	E	I	I	I	I	I	I	I	E
smallint	--	E	I	I	I	I	I	I	I	E
int	--	E	I	I	I	I	I	I	I	E
bigint	--	E	I	I	I	I	I	I	I	E
decimal	--	E	I	I	I	I	I	I	I	E
float/real	--	E	I	I	I	I	I	I	I	--
double	--	E	I	I	I	I	I	I	I	--
interval	--	--	E	E	E	E	E	--	--	I
date	--	--	--	--	--	--	--	--	--	--
time	--	--	--	--	--	--	--	--	--	--
timestamp	--	--	E	E	E	E	E	E	E	--
[var]char	--	E	I	I	I	I	I	I	I	I
[var]binary	--	--	--	--	--	--	--	--	--	--

版权所有：腾讯云计算（北京）有限责任公司

函数

统一函数

统一函数概览

最近更新时间：2024-08-07 17:30:49

数据湖计算 DLC 支持通过统一函数在不同内核使用，同时兼容 Spark、Presto，具体函数及支持的内核可参见下表。

数据湖计算 DLC 同时支持 Presto 的内置函数，支持列表及开启方式可参见 Presto 内置函数。

函数	Spark	Presto
ABS	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：与入参一致	出参： double
ACOS	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
ACOSH	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
ADD_MONTHS	入参：date|timestamp|string, int	入参：date|timestamp|strin
	出参：date	出参：date
AES_DECRYPT	-	入参：(binary, string|binary)
		出参： binary
AES_ENCRYPT	-	入参：(string|binary, string|
		出参： binary
AND	入参：boolean AND boolean	boolean AND boolean
	出参：boolean	出参：boolean
ANY	入参：boolean	入参：boolean
	出参：boolean	出参：boolean
ANY_MATCH	入参：(array<T>, lambda function)	-

	出参：boolean	
APPROX_COUNT_DISTINCT	入参：(bigint|double|decimal|date|timestamp,	-
	double|array<double>[, bigint])	
	出参：与第二个输入参数相同	
APPROX_PERCENTILE	入参：(bigint|double|decimal|, array<double>|double[, int])	入参：(bigint|double|decima
	出参：double|array	出参：double|array
ARG_MAX	入参：（col1, col2 | expr(col2)）	-
	出参：col2或expr(col2)的类型	
ARG_MIN	入参：（col1, col2 | expr(col2)）	
	出参：col2或expr(col2)的类型	
ARRAY	入参：(T, ...)	入参：(T, ...)
	出参：array<T>	出参：array<T>
ARRAYS_OVERLAP	入参：(array<T>, array<T>)	入参：(array<T>, array<T>
	出参：boolean	出参：boolean
ARRAYS_ZIP	入参：(array<T>, array< U> , ...)	-
	出参：array<struct<T, U, ...>>	
ARRAY_CONTAINS	入参：(array<T>, T)	入参：(array<T>, T)
	出参：boolean	出参：boolean
ARRAY_DISTINCT	入参：array<T>	入参：array<T>
	出参：array<T>	出参：array<T>
ARRAY_EXCEPT	入参：(array<T>, array<T>)	入参：(array<T>, array<T>
	出参：array<T>	出参：array<T>
ARRAY_INTERSECT	入参：(array<T>, array<T>)	入参：(array<T>, array<T>
	出参：array<T>	出参：array<T>

ARRAY_JOIN	入参：(array<T>, string[, string])	入参：(array<T>, string[, str
	出参：string	出参：string
ARRAY_MAX	入参：array<bigint|double|decimal>	入参：array<bigint|double|d
	出参：bigint|double|decimal	出参：bigint|double|decima
ARRAY_MIN	入参：array<bigint|double|decimal>	入参：array<bigint|double|d
	出参：bigint|double|decimal	出参：bigint|double|decima
ARRAY_POSITION	入参：(array<T>, bigint)	入参：(array<T>, bigint)
	出参：bigint	出参：bigint
ARRAY_REMOVE	入参：(array<T>, T)	入参：(array<T>, T)
	出参：array<T>	出参：array<T>
ARRAY_REPEAT	入参：(array<T>, bigint)	入参：(array<T>, bigint)
	出参：array<T>	出参：array<T>
ARRAY_SORT	入参：(array<T>, function(T, T)->integer)	入参：(array<T>)
	出参：array<T>	出参：array<T>
ARRAY_UNION	入参：(array<T>, array<T>)	入参：(array<T>, array<T>)
	出参：array<T>	出参：array<T>
ASCII	入参：string	入参：string
	出参：int	出参：int
ASIN	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
ASINH	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
ASSERT_TRUE	入参：boolean	入参：boolean
	出参：null|抛出异常	出参：null|抛出异常
ATAN	入参：bigint|double|decimal	入参：bigint|double|decima

	出参：double	出参：double
ATAN2	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decima bigint|double|decimal)
	出参：double	出参：double
ATANH	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
AVG	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参： double
BASE64	入参：string|binary	入参：binary
	出参：string	出参： string
BIGINT	强制类型转换为bigint	强制类型转换为bigint
BIN	入参：bigint	入参：bigint
	出参：string	出参： string
BINARY	强制类型转换为 binary	强制类型转换为 binary
BIT_AND	入参：int	入参：int
	出参：int	出参：int
BIT_COUNT	入参：int|boolean	入参：int|boolean
	出参：int	出参：int
BIT_GET	入参：(int, int)	入参：(int, int)
	出参：int	出参：int
BIT_LENGTH	入参：string	入参：string
	出参：int	出参：int
BIT_OR	入参：int	入参：int
	出参：int	出参：int
BIT_XOR	入参：int	入参：int

	出参：int	出参：int
BOOLEAN	强制类型转换为 boolean	强制类型转换为 boolean
BOOL_AND	入参：boolean	入参：boolean
	出参：boolean	出参：boolean
BOOL_OR	入参：boolean	入参：boolean
	出参：boolean	出参：boolean
BROUND	入参：(bigint|double|decimal, int)	入参：(bigint|double|decima
	出参：int	出参：int
BTRIM	入参：(string[, string])	入参：(string[, string])
	出参：string	出参：string
CARDINALITY	入参：(array|map)	入参：(array|map)
	出参：int	出参：int
CAST	入参：(<expr> AS T)	入参：(<expr> AS T)
	出参：T	出参：T
CBRT	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
CEIL	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：bigint	出参：bigint
CEILING	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：bigint	出参：bigint
CHAR	入参：int	入参：int
	出参：string	出参：string
CHARACTER_LENGTH	入参：string|binary	入参：string
	出参：int	出参：int
CHAR_LENGTH	入参：string|binary	入参：string

	出参：int	出参：int
CHR	入参：int|double	入参：int|double
	出参：string	出参： string
CLUSTER_SAMPLE	入参：（bigint [, bigint]）	-
	出参：boolean	
COALESCE	入参：(T, T, ...)	入参：(T, T, ...)
	出参：T	出参：T
COLLECT_LIST	入参：T	入参：T
	出参：array<T>	出参： array<T>
COLLECT_SET	入参：T	入参：T
	出参：array<T>	出参： array<T>
CONCAT	入参：string|array	入参(string|binary, string|bin
	出参：与输入参数一致	出参：string
CONCAT_WS	入参：(string, [string | array<string>]+)	入参：(string, [string | array
	出参：string	出参：string
CONTEXT_NGRAMS	-	入参：(array<array<string> int, int)
		出参：array<struct<string, c
CONV	入参：(bigint|double|decimal|string, int, int)	入参：(bigint|double|decima
	出参：string	出参： string
CORR	入参：bigint|double|decimal,	入参：bigint|double|decima
	出参：double	出参： double
	出参：double	
COS	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double

COSH	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
COT	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
COUNT	入参：T	入参：T
	出参：bigint	出参： bigint
COUNT_IF	入参：boolean	-
	出参：int	
COUNT_MIN_SKETCH	入参：(int|binary|string, double, double, int)	-
	出参：binary	
COVAR_POP	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参： double
COVAR_SAMP	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参： double
CRC32	入参：string|binary	入参：string|binary
	出参：bigint	出参：bigint
CUME_DIST	入参：无	入参：无
	出参：double	出参：double
CURRENT_CATALOG	入参：无	-
	出参：string	
CURRENT_DATABASE	入参：无	-
	出参：string	
CURRENT_DATE	入参：无	入参：无
	出参：date	出参：date
CURRENT_TIMESTAMP	入参：无	入参：无

	出参：timestamp	出参：timestamp
CURRENT_TIMEZONE	入参：无	入参：无
	出参：string	出参：string
CURRENT_USER	入参：无	入参：无
	出参：string	出参：string
DATE	入参：string	入参：string
	出参：date	出参：date
DATEDIFF	入参：(string|timestamp|date, string|timestamp|date)	入参：(string|timestamp|dat string|timestamp|date)
	出参：int	出参：int
DATE_ADD	入参：(string|date|timestamp, int)	入参：(string|date|timestam
	出参：date	出参： date
DATEADD	入参：(date/timestamp, int, string)	-
	出参：timestamp	
DATE_FORMAT	入参：(string|date|timestamp, string)	入参：(string|date|timestam
	出参：string	出参：string
DATE_FROM_UNIX_DATE	入参：int	入参：int
	出参：date	出参：date
DATE_PART	入参：(string, string|date|timestamp)	入参：(string, string|date|tin
	出参：bigint|double|decimal	出参：bigint|double|decima
DATE_SUB	入参：(string|date|timestamp)	入参：(string|date|timestam
	出参：date	出参：date
DATE_TRUNC	入参：(string, string)	入参：(string, string)
	出参：timestamp	出参：timestamp
DAY	入参：string|timestamp|date	入参：string|timestamp|date

	出参：int	出参：int
DAYOFMONTH	入参：string|timestamp|date	入参：string|timestamp|date
	出参：int	出参：int
DAYOFWEEK	入参：string|timestamp|date	string|timestamp|date
	出参：int	出参：int
DAYOFYEAR	入参：string|timestamp|date	string|timestamp|date
	出参：int	出参：int
DECIMAL	cast as decimal	cast as decimal
DECODE	入参：(binary, string) | (int, int, string[, int, string]...[, default])	入参：(binary, string)
	出参：	出参： string
DEGREES	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
DENSE_RANK	入参：无	入参：无
	出参：int	出参：int
DIV	入参：bigint|double|decimal	-
	出参：int	
DOUBLE	cast as double	cast as double
E	入参：无	入参：无
	出参：double	出参：double
ELEMENT_AT	入参：(array|map, int)	入参：(array, int)
	出参：与array元素|map value一致	出参：与array元素一致
ELT	入参：(int, T, U,...)	入参：(int, string, string, ...)
	出参：依输入而定	出参： string
ENCODE	入参：(string|binary, string)	入参：(string, string)
	出参：binary	出参： binary

EVERY	入参：T	入参：T
	出参：boolean	出参：boolean
EXP	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
EXPLODE	入参：array<T>	-
	出参：T	
EXPLODE_OUTER	入参：array<T>	-
	出参：T	
EXPM1	入参：bigint|double|decimal	-
	出参：double	
FACTORIAL	入参：int	入参：int
	出参：bigint	出参： bigint
FIELD	-	入参：(T, T, T, ...)
		出参：int
FIND_IN_SET	入参：(string, string)	入参：(string, string)
	出参：int	出参：int
FIRST_VALUE	入参：(T[, boolean])	-
	出参：T	
FLATTEN	入参：array<array<T>>	入参：array<array<T>>
	出参：array<T>	出参：array<T>
FLOAT	cast as float	cast as float
FLOOR	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：bigint	出参：bigint
FORMAT_NUMBER	入参：(bigint|double|decimal, int|string)	入参：(bigint|double|decima

	出参：string	出参：string
FORMAT_STRING	入参：(string, T, ...)	入参：(string, T, ...)
	出参：string	出参：string
FROM_CSV	入参：(string, string[, map])	-
	出参：struct	
FROM_JSON	入参：(string, string[, map])	-
	出参：struct	
FROM_UNIXTIME	入参：(bigint[, string])	入参：(bigint[, string])
	出参：string	出参：string
FROM_UTC_TIMESTAMP	入参：(string, string)	入参：(bigint|double|decimal|times string)
	出参：timestamp	出参： timestamp
GETBIT	入参：(int, int)	入参：(int, int)
	出参：int	出参：int
GET_IDCARD_AGE	入参：string	-
	出参：int	
GET_IDCARD_BIRTHDAY	入参：string	-
	出参：date	
GET_IDCARD_SEX	入参：string	-
	出参：string	
GREATEST	入参：(bigint|double|decimal, bigint|double|decimal, ...)	入参：(bigint|double|decima bigint|double|decimal, ...)
	出参：与输入一致	出参：与输入一致
GROUPING	入参：T	T
	出参：int	出参：int
GROUPING_ID	入参：[T[, T...]]	-

	出参：int	
HASH	入参：(T, T,...)	入参：(T, T,...)
	出参：int	出参：int
HEX	入参：int|string|binary	入参：int|string|binary
	出参：string	出参：string
HISTOGRAM_NUMERIC	-	入参：bigint|double|decima
		出参：array<struct<'x', 'y'>>
HOUR	入参：string|timestamp	入参：string|timestamp
	出参：int	出参：int
HYPOT	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
IF	入参：(boolean, T, T)	入参：(boolean, T, T)
	出参：T	出参：T
IFNULL	入参：(T, U)	-
	出参：T|U	
IN	入参：(T, T,...)	入参：(T, T,...)
	出参：boolean	出参：boolean
INITCAP	入参：string	入参：string
	出参：string	出参：string
INLINE	入参：array	-
	出参：table	
INLINE_OUTER	入参：array	-
	出参：table	
INPUT_FILE_BLOCK_LENGTH	入参：无	-
	出参：int	

INPUT_FILE_BLOCK_START	入参：无	-
	出参：int	
INPUT_FILE_NAME	入参：无	-
	出参：string	
INSTR	入参：(string, string)	入参：(string, string)
	出参：int	出参：int
INT	强制类型转换为 int	强制类型转换为 int
ISNAN	入参：T	入参：T
	出参：boolean	出参：boolean
ISNOTNULL	入参：T	入参：T
	出参：boolean	出参：boolean
ISNULL	入参：T	入参：T
	出参：boolean	出参：boolean
JSON_ARRAY_LENGTH	入参：string	入参：string
	出参：int	出参：int
JSON_OBJECT_KEYS	入参：string	入参：string
	出参：array	出参：array
JSON_TUPLE	入参：(string, ..., string)	-
	出参：string...string	
KEYVALUE	入参：(string [, string, string], string)	
	出参：string	
KURTOSIS	入参：bigint|double|decimal	-
	出参：double	
LAG	入参：(T[, int[, string]])	入参：(T[, int[, string]])
	出参: T	出参：T

LAST_DAY	入参：string|date|timestamp	入参：string|date|timestamp
	出参：string	出参：string
LAST_VALUE	入参：(T[, boolean])	-
	出参：T	
LCASE	入参：string	入参：string
	出参：string	出参：string
LEAD	入参：(T[, int[, string]])	入参：(T[, int[, string]])
	出参：T	出参：T
LEAST	入参：(bigint|double|decimal, bigint|double|decimal, ...)	(bigint|double|decimal, bigint|double|decimal, ...)
	出参：与输入一致	出参：与输入一致
LEFT	入参：(string, int)	入参：(string, int)
	出参：string	出参：string
LENGTH	入参：string	入参：string
	出参：int	出参：int
LEVENSHTEIN	入参：(string, string)	入参：(string, string)
	出参：int	出参：int
LIKE	入参：(string, string) | string LIKE string ESCAPE string	入参：(string, string)
	出参：boolean	出参：boolean
LN	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
LOCATE	入参：(string, string[, int])	入参：(string, string[, int])
	出参：int	出参：int
LOG	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decima bigint|double|decimal)
	出参：double	出参：double

LOG10	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
LOG1P	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
LOG2	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
LOWER	入参：string	入参：string
	出参：string	出参：string
LPAD	入参：(string, int[, string])	入参：(string, int, string)
	出参：string	出参：string
LTRIM	入参：string	入参：string
	出参：string	出参：string
MAKE_DATE	入参：(int, int, int)	入参：(int, int, int)
	出参：date	出参：date
MAKE_TIMESTAMP	入参：(int, int, int, int, int, int[, string])	入参：(int, int, int, int, int, int
	出参：timestamp	出参：timestamp
MAP	入参：(K1, V1, K2, V2,...)	入参：(array<K>, array<V>
	出参：map	出参：map
MAP_CONCAT	入参：(map, ...)	入参：(map, ...)
	出参：map	出参：map
MAP_ENTRIES	入参：map	入参：map
	出参：array	出参：array
MAP_FROM_ARRAYS	入参：(array, array)	入参：(array, array)
	出参：map	出参：map
MAP_FROM_ENTRIES	入参：array	入参：array

	出参：map	出参：map
MAP_KEYS	入参：map	入参：map
	出参：array	出参：array
MAP_UINON_SUM	入参：map	-
	出参：map	
MAP_VALUES	入参：map	入参：map
	出参：array	出参：array
MAX	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：与入参一致	出参： double
MAX_BY	入参：T, bigint|double|decimal	入参：T, bigint|double|decir
	出参：T	出参：T
MAX_PT	入参：const string	-
	出参: string	
MD5	入参：string|binary	入参：string|binary
	出参：string	出参： string
MEAN	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
MIN	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：与入参一致	出参： double
MINUTE	入参：(stirng|timestamp)	入参：(stirng|timestamp)
	出参：int	出参：int
MIN_BY	入参：(T, bigint|double|decimal)	入参：(T, bigint|double|deci
	出参：T	出参：T
MOD	入参：(bigint|double|decimal, bigint|double|decimal) | bigint|double|decimal MOD bigint|double|decimal	入参：(bigint|double|decima bigint|double|decimal)

	出参：int|double	出参：int|double
MONOTONICALLY_INCREASING_ID	入参：无	-
	出参：bigint	
MONTH	入参：date|timestamp|string	入参：date|timestamp|string
	出参：int	出参：int
MONTHS_BETWEEN	入参：(date|timestamp|string, date|timestamp|string[, boolean])	入参：(date|timestamp|string, date|timestamp|string)
	出参：double	出参：double
NAMED_STRUCT	入参：(K1, V1, K2, V2, ...)	-
	出参：struct	
NANVL	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decimal, bigint|double|decimal)
	出参：double	出参：double
NEGATIVE	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：bigint|double|decimal	出参： int | double
NEXT_DAY	入参：(string|date|timestamp, string)	入参：(string|date|timestamp
	出参：date	出参：string
NGRAMS	-	入参：(array<array<string>
		出参：array<struct<string,
NOW	入参：无	入参：无
	出参：timestamp	出参：timestamp
NTH_VALUE	入参：(T[, int])	入参：(T[, int])
	出参：T	出参：T
NTILE	入参：int	入参：int
	出参：int	出参： int
NULLIF	入参：(T, T)	入参：(T, T)

	出参：T	出参：T
NVL	入参：(T, T)	入参：(T, T)
	出参：T	出参：T
NVL2	入参：(T, T, T)	入参：(T, T, T)
	出参：T	出参：T
OCTET_LENGTH	入参：string	入参：string
	出参：int	出参：int
OR	入参：expr1 OR expr2	入参：expr1 OR expr2
	出参：boolean	出参：boolean
OVERLAY	入参：(string PLACING string FROM int[FOR int])	-
	出参：	
PARSE_URL	入参：(string, string[, string])	入参：(string, string[, string]
	出参：string	出参： string
PERCENTILE	入参：(bigint|double|decimal, bigint|double|decimal|array, int)	入参：(bigint|double|decima bigint|double|decimal|array,
	出参：bigint|double|decimal|array	出参：double|array
PERCENTILE_APPROX	入参：(bigint|double|decimal, bigint|double|decimal | array[, int])	入参：(bigint|double|decima bigint|double|decimal[, int])
	出参：double|array	出参：double|array
PERCENT_RANK	入参：无	入参：无
	出参：double	出参：double
PI	入参：无	入参：无
	出参：double	出参：double
PMOD	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decima bigint|double|decimal)
	出参：与入参一致	出参：与入参一致

POSITION	入参：(string, string[, int]) | (string IN string)	入参：(string, string[, int]) | (
	出参：int	出参：int
POSITIVE	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：与入参一致	出参：与入参一致
POSEXPLODE	入参：array	-
	出参：table	
POSEXPLODE_OUTER	入参：array	-
	出参：table	
POW	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decima bigint|double|decimal)
	出参：double	出参：double
POWER	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decima bigint|double|decimal)
	出参：double	出参：double
PRINTF	入参：(string, T, ...)	入参：(string, T, ...)
	出参：string	出参：string
QUARTER	入参：string|date|timestamp	入参：string|date|timestamp
	出参：int	出参：int
RADIANS	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
RAISE_ERROR	入参：string	入参：string
	出参：抛出异常	出参：抛出异常
RAND	入参：[int]	入参：[int]
	出参：double	出参：double
RANDN	入参：[int]	入参：[long]
	出参：double	出参：double

版权所有：腾讯云计算（北京）有限责任公司

RANDOM	入参：[int]	入参：[int]
	出参：double	出参：double
RANK	入参：无	入参：无
	出参：int	出参：int
REGEXP	入参：(string, string)	入参：(string, string)
	出参：boolean	出参：boolean
REGEXP_COUNT	入参：(string, string)	-
	出参：int	
REGEXP_EXTRACT	入参：(string, string[, int])	入参：(string, string, int)
	出参：int	出参：int
REGEXP_EXTRACT_ALL	入参：(string, string[, int])	入参：(string, string[, int])
	出参：array	出参：array
REGEXP_LIKE	入参：(string, string)	入参：(string, string)
	出参：boolean	出参：boolean
REGEXP_REPLACE	入参：(string, string, string[, int])	入参：(string, string, string)
	出参：string	出参：string
REGEXP_SUBSTR	入参：(string, string)	-
	出参：string	
REPEAT	入参：(string, int)	入参：(string, int)
	出参：string	出参：string
REPLACE	入参：(string, string[, string])	入参：(string, string, string)
	出参：string	出参：string
REVERSE	入参：string | array	入参：string
	出参：string	出参：string
RIGHT	入参：(string, int)	入参：(string, int)

	出参：string	出参：string
RINT	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
RLIKE	入参：(string, string)	入参：(string, string)
	出参：boolean	出参：boolean
ROUND	入参：(double, int)	入参：(double, int)
	出参：double	出参：double
ROW_NUMBER	入参：无	入参：无
	出参：int	出参：int
RPAD	入参：(string, int[, string])	入参：(string, int, string)
	出参：string	出参：string
RTRIM	入参：string	入参：string
	出参：string	出参：string
SAMPLE	入参：(long [, long [, cols...]])	-
	出参：boolean	
SCHEMA_OF_CSV	入参：(string[, map])	
	出参：struct	
SCHEMA_OF_JSON	入参：(string[, map])	-
	出参：	
SECOND	入参：string|timestamp	入参：string|timestamp
	出参：int	出参：int
SENTENCES	(string[, string, string])	入参：(string[, string, string]
	出参：array<array<string>>	出参：array<array<string>>
SEQUENCE	入参：(int|bigint|date|timestamp, int|bigint|date|timestamp, int)	入参：(int|bigint|date|timest int|bigint|date|timestamp, int
	出参：与入参一致	出参：与入参一致

SHA	入参：string|binary	入参：string|binary
	出参：string	出参：string
SHA1	入参：string|binary	入参：string|binary
	出参：string	出参：string
SHA2	入参：(string, int)	入参：(string, int)
	出参：string	出参：string
SHIFTLEFT	入参：(int|bigint, int)	入参：(int|biging, int)
	出参：int|bigint	出参：int|bigint
SHIFTRIGHT	入参：(int|bigint, int)	入参：(int|bigint, int)
	出参：int|bigint	出参：int|bigint
SHIFTRIGHTUNSIGNED	入参：(int|bigint, int)	入参：(int|bigint, int)
	出参：int|bigint	出参：int|bigint
SHUFFLE	入参：array	入参：array
	出参：array	出参：array
SIGN	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
SIGNUM	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
SIN	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
SINH	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
SIZE	入参：array | map	入参：array | map
	出参：int	出参：int
SKEWNESS	入参：bigint|double|decimal	-

	出参：double	
SLICE	入参：(array, int, int)	入参：(array, int, int)
	出参：array	出参：array
SMALLINT	cast as smallint	cast as smallint
SOME	入参：boolean	入参：boolean
	出参：boolean	出参：boolean
SORT_ARRAY	入参：(array[, boolean])	入参：array
	出参：array	出参：array
SOUNDEX	入参：string	入参：string
	出参：string	出参：string
SPACE	入参：string	入参：string
	出参：string	出参：string
SPARK_PARTITION_ID	入参：无	-
	出参：int	
SPLIT	入参：(string, string, int)	入参：(string, string)
	出参：string	出参：string
SQRT	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
STACK	入参：(T, U, ...)	-
	出参：table	
STD	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
STDDEV	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参： double
STDDEV_POP	入参：bigint|double|decimal	入参：bigint|double|decima

	出参：double	出参： double
STDDEV_SAMP	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参： double
STRING	cast as string	cast as string
STRUCT	入参：(T, U, ...)	入参：(T, U, ...)
	出参：struct	出参：struct
STR_TO_MAP	入参：(string[, string[, string]])	入参：(string[, string, string]
	出参：map	出参：map
SUBSTR	入参：(string, int[, int]) | (string FROM int [FOR int])	入参：(string|binary, int[, int FROM int [FOR int])
	出参：string	出参： string
SUBSTRING	入参：(string, int[, int]) | (string FROM int [FOR int])	入参：(string|binary, int[, int FROM int [FOR int])
	出参：string	出参： string
SUBSTRING_INDEX	入参：(string, string, int)	入参：(string, string, int)
	出参：string	出参：string
SUM	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：与输入一致	出参： double
TAN	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
TANH	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参：double
TIMESTAMP	入参：string	入参：string
	出参：timestamp	出参：timestamp
TIMESTAMP_ADD	入参：（date/timestamp, int, string）	
	出参：timestamp	

TIMESTAMP_MICROS	入参：bigint	入参：bigint
	出参：timestamp	出参：timestamp
TIMESTAMP_MILLIS	入参：bigint	入参：bigint
	出参：timestamp	出参：timestamp
TIMESTAMP_SECONDS	入参：bigint	入参：bigint
	出参：timestamp	出参：timestamp
TIMEZONE_HOUR	入参：string	-
	出参：int	
TIMEZONE_MINUTE	入参：string	-
	出参：int	
TINYINT	cast as tinyint	cast as tinyint
TO_CHAR	入参：(boolean|int|long|double|decimal [, string])	
	出参：string	
TO_CSV	入参：(struct[, map])	-
	出参：string	
TO_DATE	入参：(string[, string])	入参：string
	出参：date	出参： date
TO_JSON	入参：(string[, map])	-
	出参：string	
TO_TIMESTAMP	入参：(string[, string])	入参：(string)
	出参：timestamp	出参：timestamp
TO_UNIX_TIMESTAMP	入参：(string[, string])	入参：(string[, string])
	出参：timestamp	出参：timestamp
TO_UTC_TIMESTAMP	入参：(string[, string])	入参：(bigint|double|decimal|date| string)

	出参：timestamp	出参： timestamp
TRANS_ARRAY	入参：(int, string, cols...)	-
	出参：与cols类型相同	
TRANS_COLS	入参：(int, cols...)	
	出参：(int, cols）	
TRANSLATE	入参：(string, string, string)	-
	出参：string	
TRIM	入参：string	入参：string
	出参：string	出参： string
TRUNC	入参：(string|date|timestamp, string)	入参：(string|date|timestam
	出参：string	出参： string
TRY_ADD	入参：(bigint|double|decimal|date|timestamp|, bigint|double|decimal|date|timestamp)	入参：(bigint|double|decima bigint|double|decimal)
	出参：与入参一致	出参：与入参一致
TRY_DIVIDE	入参：(bigint|double|decimal, bigint|double|decimal)	-
	出参：double	
TYPEOF	入参：T	入参：T
	出参：string	出参：string
UCASE	入参：string	入参：string
	出参：string	出参：string
UNBASE64	入参：string	入参：string
	出参：string	出参：string
UNHEX	入参：string	入参：string
	出参：binary	出参：binary

UNIX_DATE	入参：string|timestamp|date	入参：string|timestamp|date
	出参：int	出参：int
UNIX_MICROS	入参：timestamp	入参：timestamp
	出参：bigint	出参：bigint
UNIX_MILLIS	入参：timestamp	入参：timestamp
	出参：bigint	出参：bigint
UNIX_SECONDS	入参：timestamp	入参：timestamp
	出参：bigint	出参：bigint
UNIX_TIMESTAMP	入参：[date|timestamp|string[, string]]	入参：(date|timestamp|strin
	出参：bigint	出参： bigint
UPPER	入参：string	入参：string
	出参：string	出参：string
URL_DECODE	入参：string	-
	出参：string	
UUID	入参：无	入参：无
	出参：string	出参：string
VARIANCE	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参： double
VAR_POP	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参： double
VAR_SAMP	入参：bigint|double|decimal	入参：bigint|double|decima
	出参：double	出参： double
VERSION	入参：无	入参：无
	出参：string	出参：string
WEEKDAY	入参：date|timestamp|string	入参：date|timestamp|string

	出参：int	出参：int
WEEKOFYEAR	入参：date|timestamp|string	入参：date|timestamp|string
	出参：int	出参：int
WIDTH_BUCKET	入参：(bigint|double|decimal, bigint|double|decimal, bigint|double|decimal, int)	入参：(bigint|double|decima bigint|double|decimal, bigint int)
	出参：int	出参：int
WINDOW	入参：(date|timestamp, string[, string[, string]])	-
	出参：table	
XPATH	入参：(string, string)	入参：(string, string)
	出参：array	出参：array
XPATH_BOOLEAN	入参：(string, string)	入参：(string, string)
	出参：boolean	出参：boolean
XPATH_DOUBLE	入参：(string, string)	入参：(string, string)
	出参：double	出参：double
XPATH_FLOAT	入参：(string, string)	入参：(string, string)
	出参：float	出参：float
XPATH_INT	入参：(string, string)	入参：(string, string)
	出参：int	出参：int
XPATH_LONG	入参：(string, string)	入参：(string, string)
	出参：bigint	出参：bigint
XPATH_NUMBER	入参：(string, string)	入参：(string, string)
	出参：double	出参：double
XPATH_STRING	入参：(string, string)	入参：(string, string)
	出参：string	出参：string
XXHASH64	入参：(T, U, ...)	入参：(T, U, ...)

版权所有：腾讯云计算（北京）有限责任公司

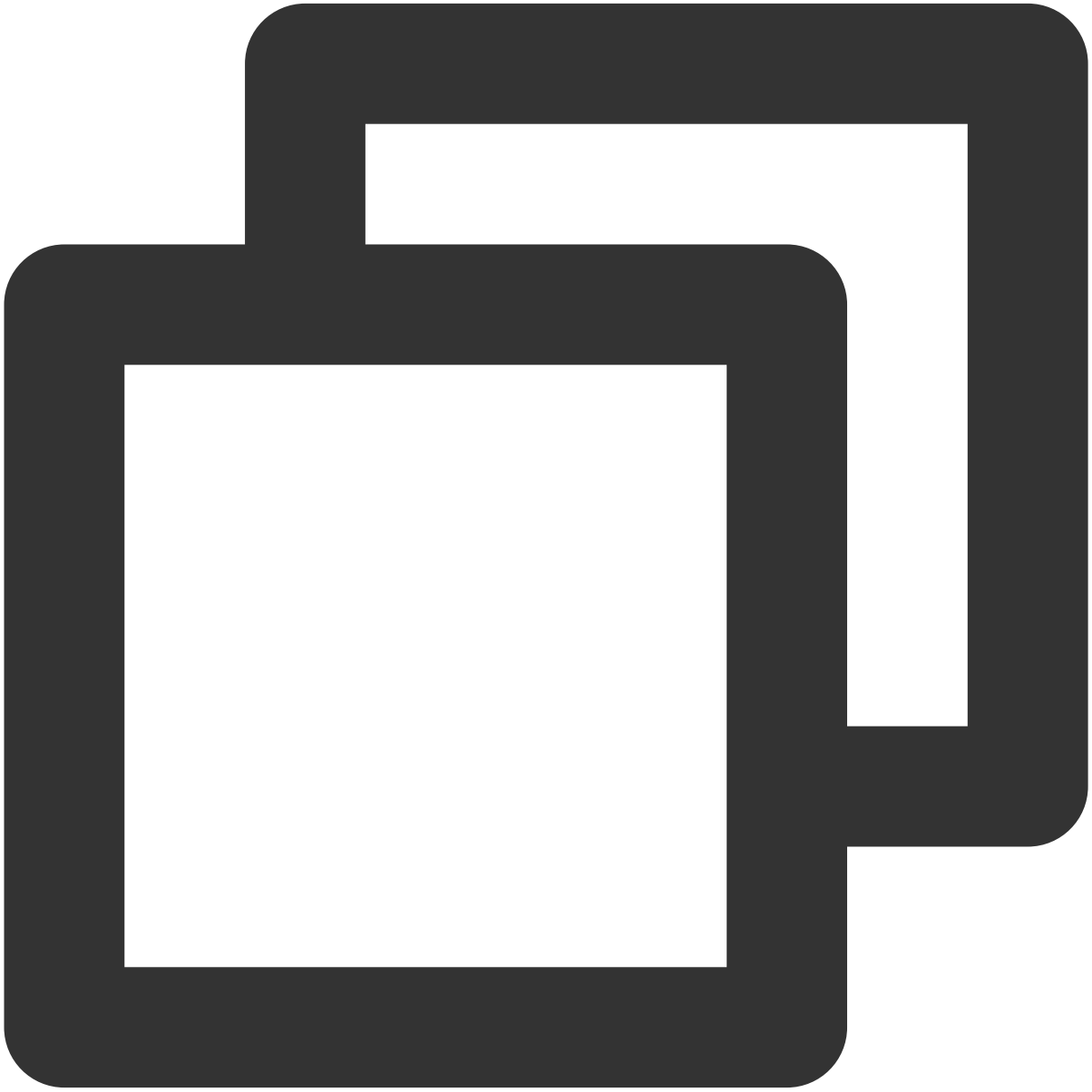
	出参：bigint	出参：bigint
YEAR	入参：date|timestamp|string	入参：date|timestamp|string
	出参：int	出参：int

二进制函数

最近更新时间：2024-08-07 17:31:01

CRC32

函数语法：



```
CRC32(<expr> string|binary)
```

支持引擎： `SPARKSQL` ， `PRESTO`

使用说明：使用CRC32算法计算表达式的循环冗余校验值。

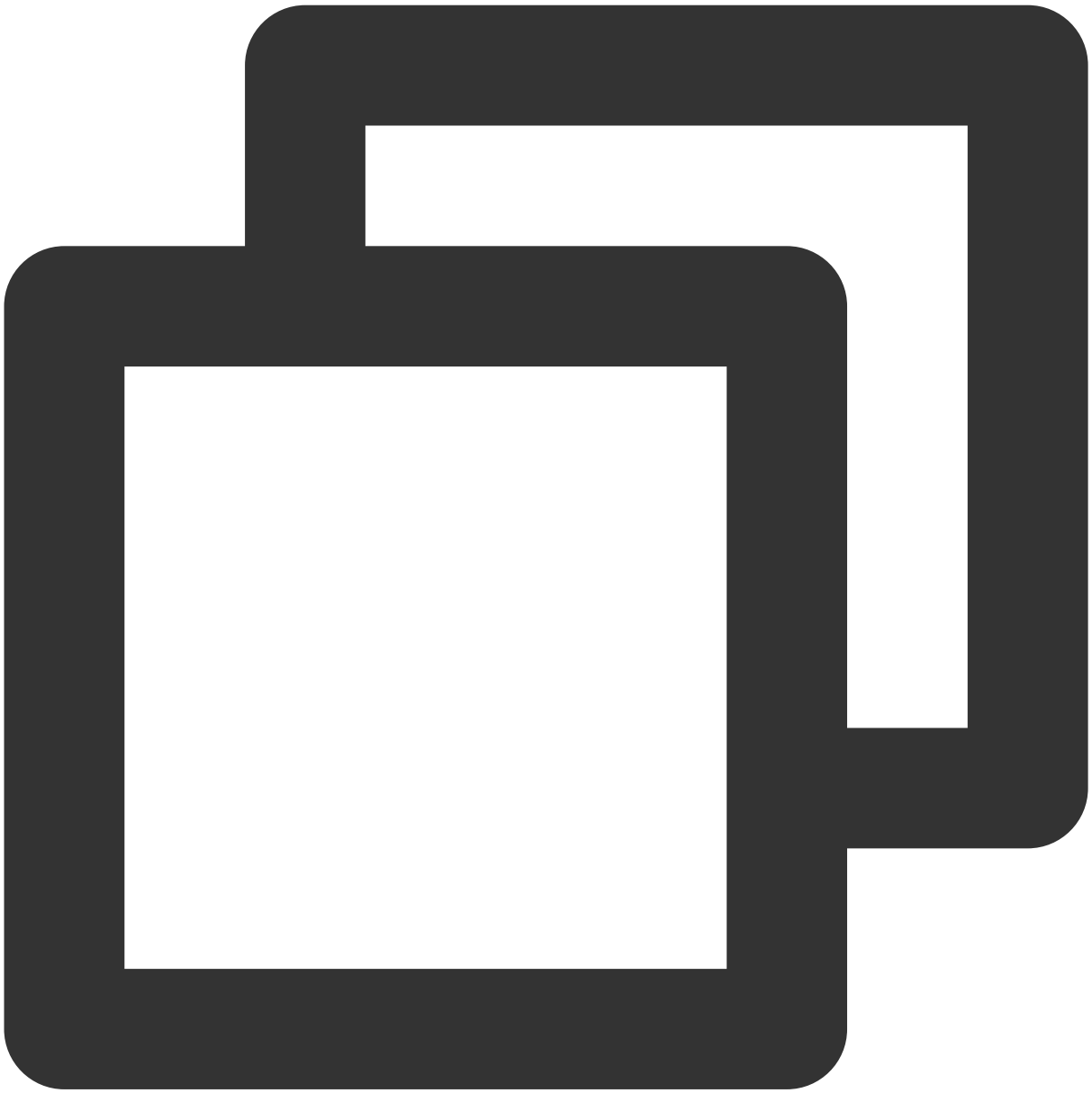
返回类型： `bigint`

示例：

```
> select crc32('tencent');
820633257
```

MD5

函数语法：



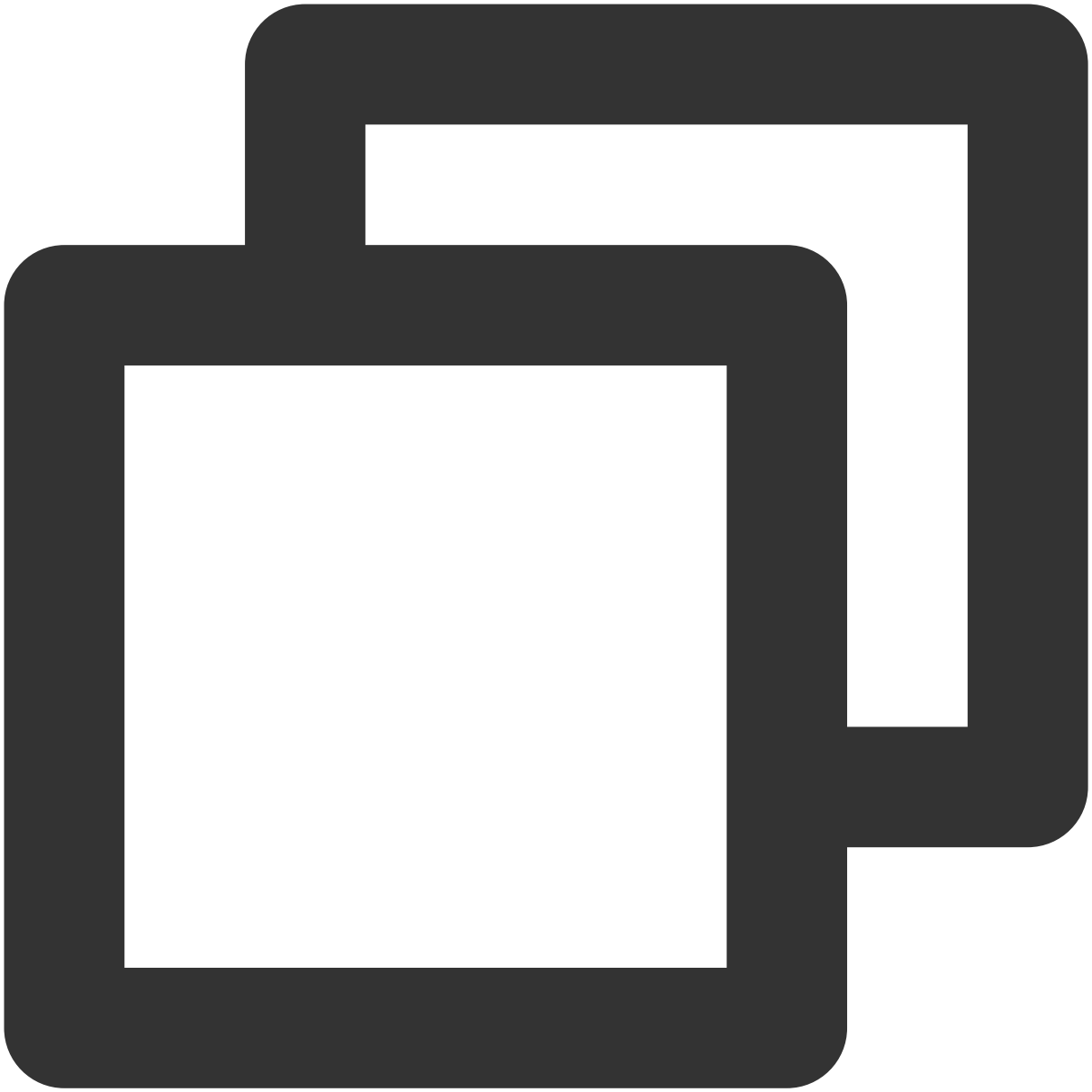
```
MD5(<expr string|binary)
```

支持引擎：SPARKSQL, PRESTO

使用说明：以十六进制字符串形式返回MD5 128位校验和。

返回类型：string

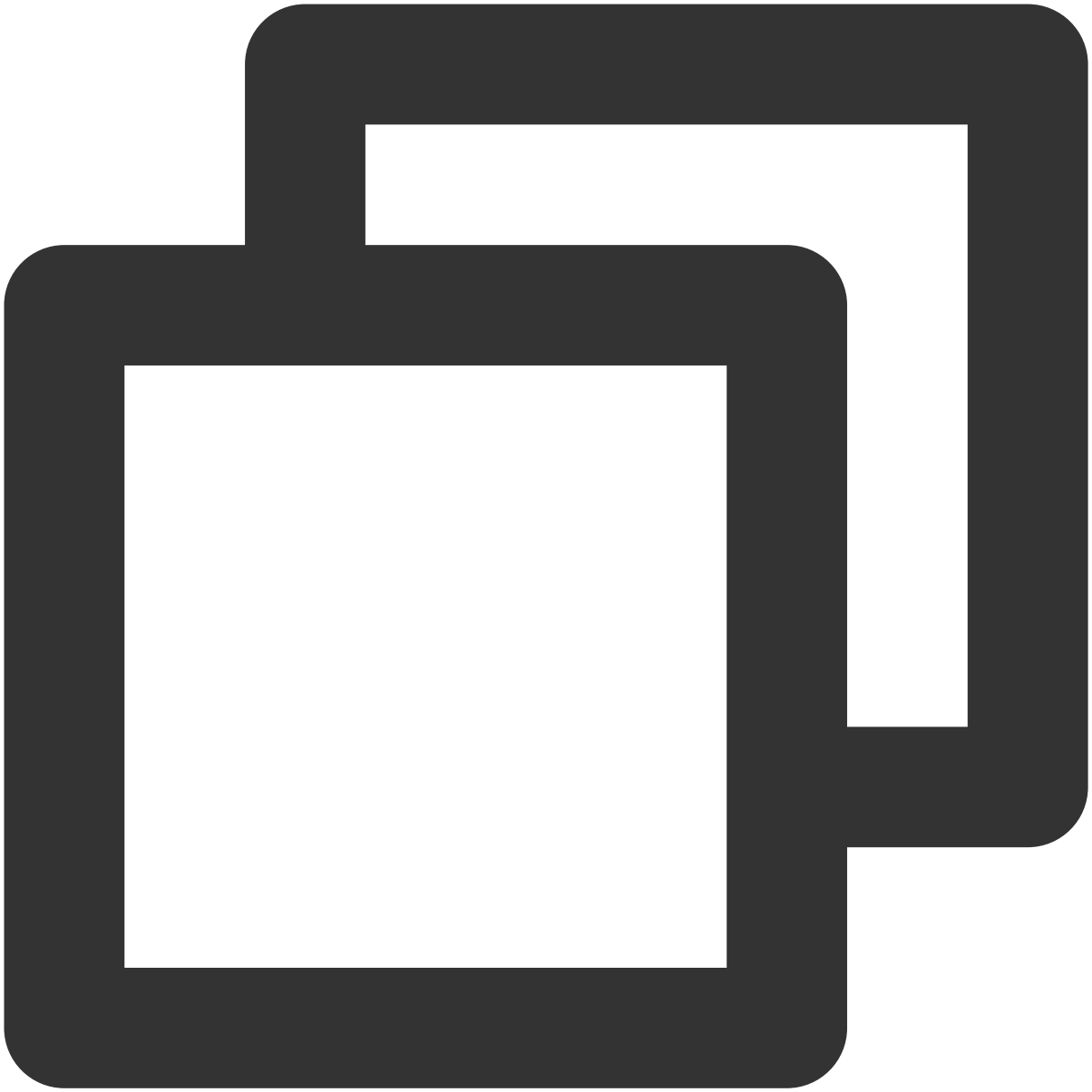
示例：



```
> select md5('tencent');
3da576879001c77b442b9f8ef95c09d6
```

HASH

函数语法：



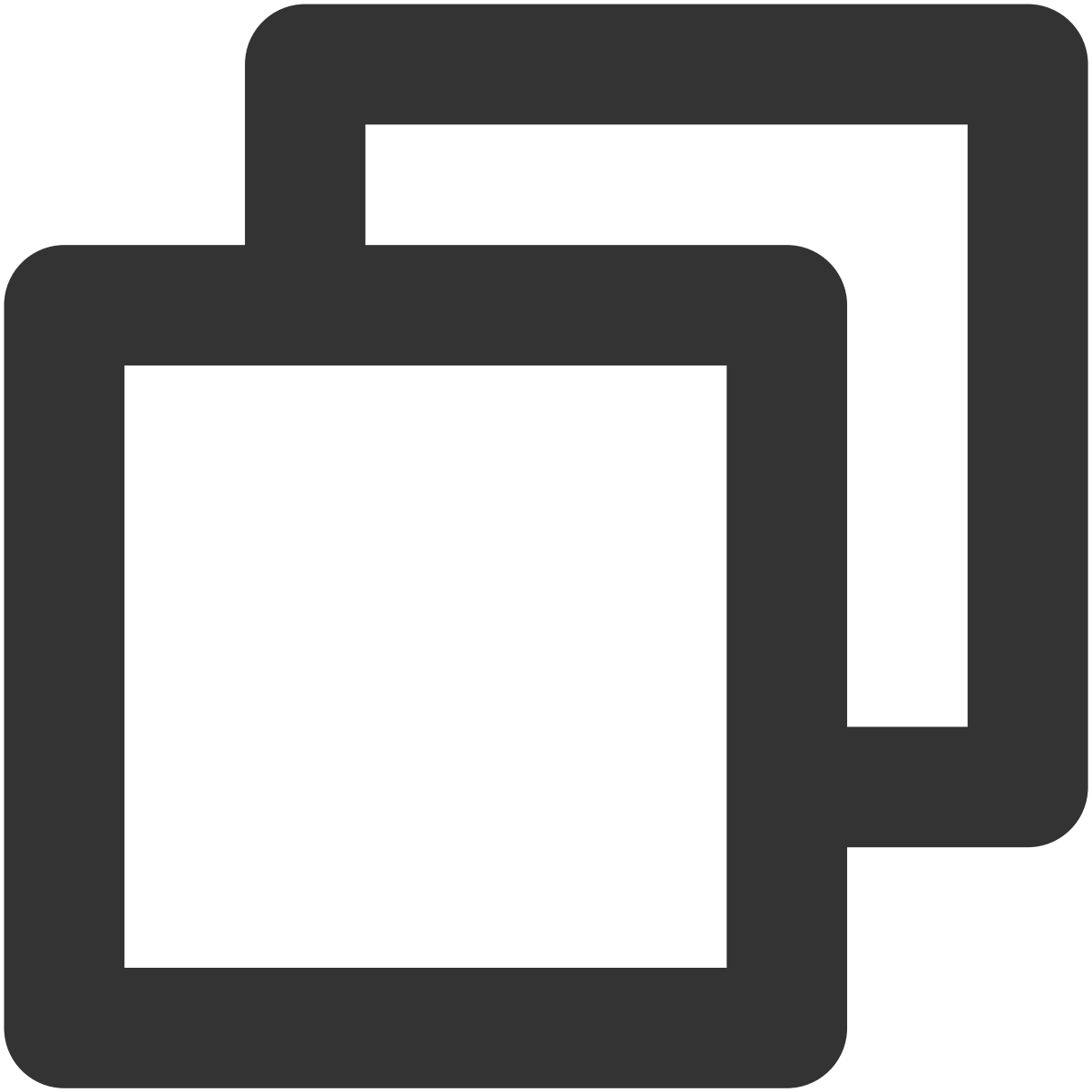
```
HASH(&lt;expr1&gt; any[, &lt;expr2&gt; any, ...])
```

支持引擎：SPARKSQL, PRESTO

使用说明：返回所有参数的哈希值。 `SPARKSQL` 与 `PRESTO` 计算方式不一致，可能会得到不同的结果

返回类型： `integer`

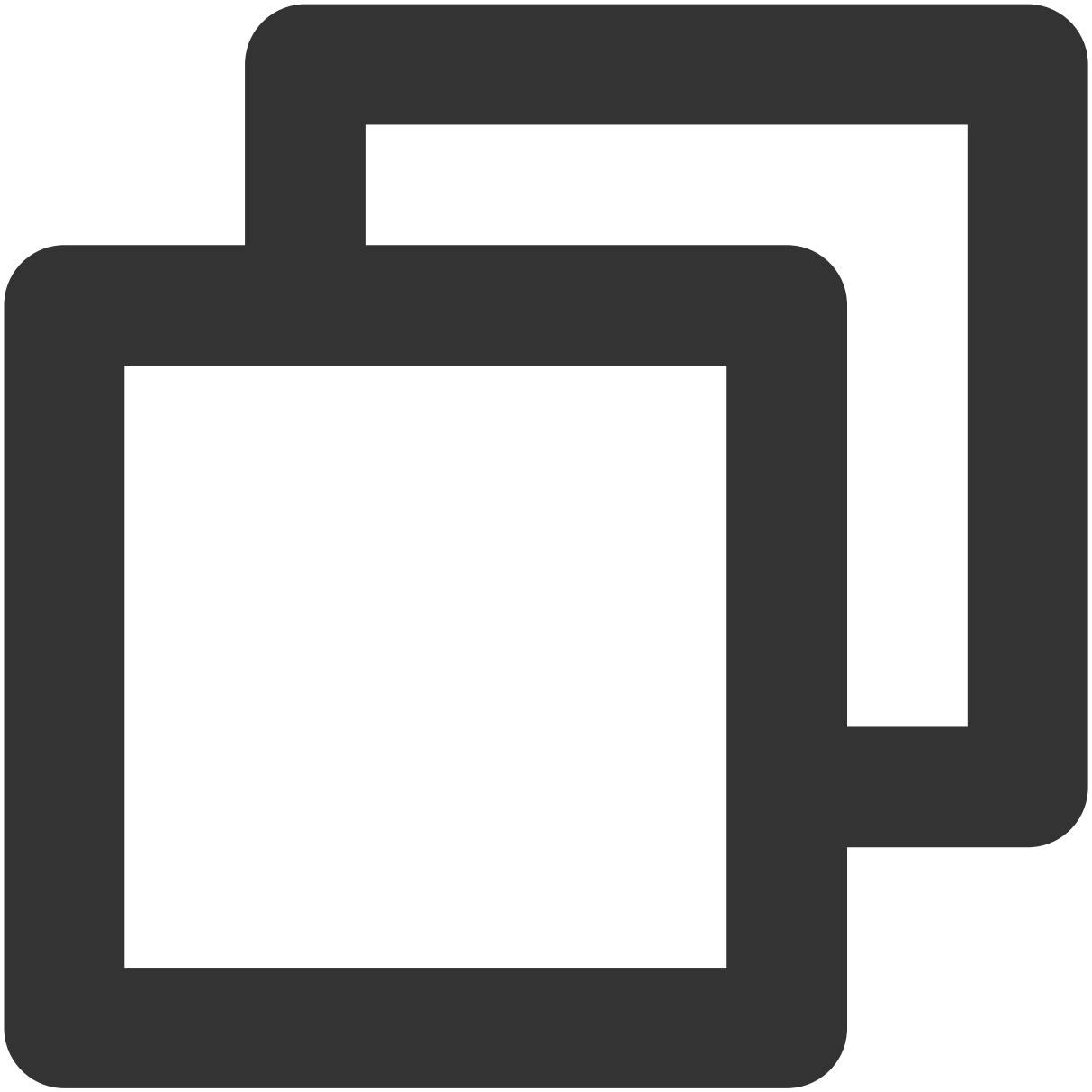
示例：



```
> SELECT hash('tencent', array(123), 2);
-412995102
```

XXHASH64

函数语法：



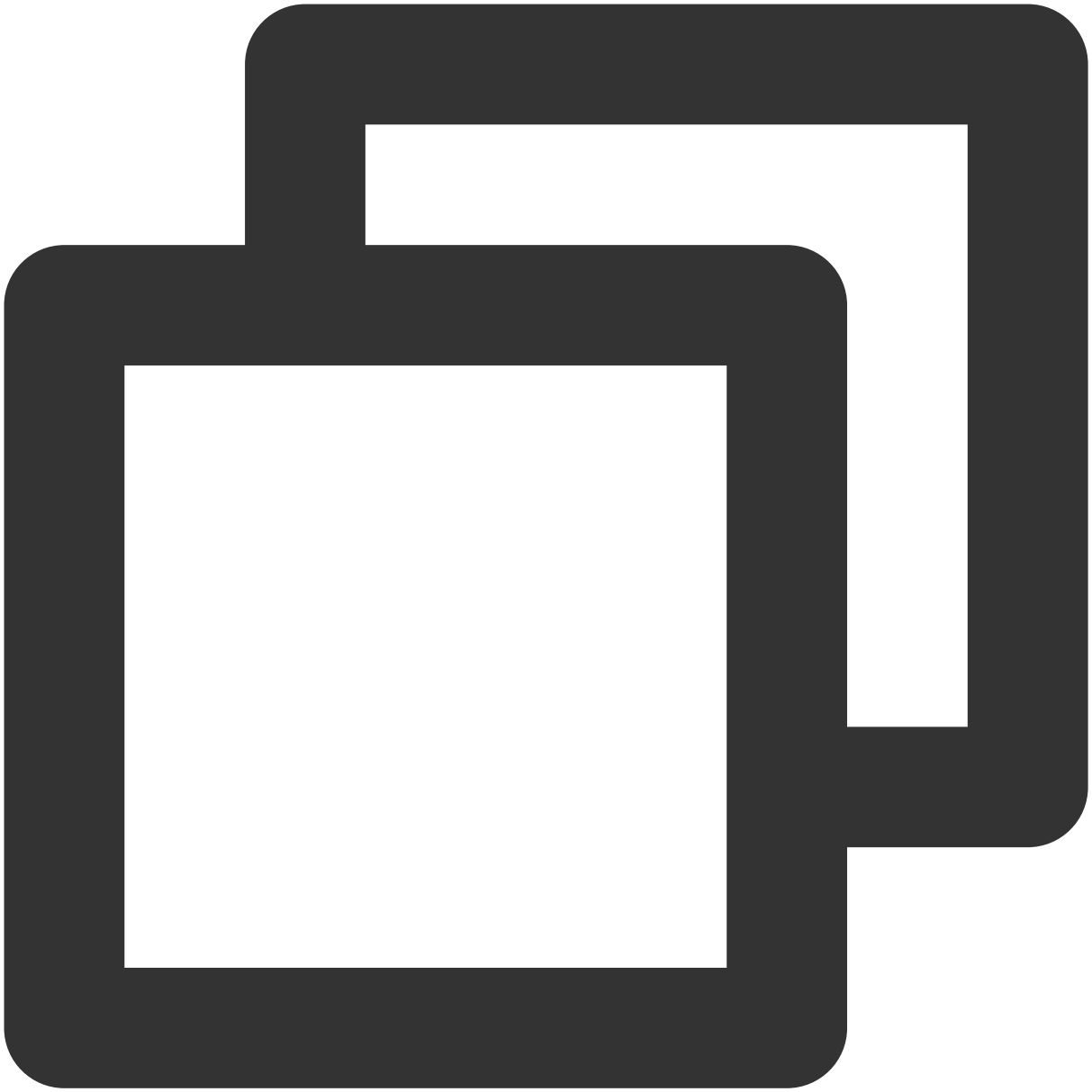
```
XXHASH64(&lt;expr1&gt; any[, &lt;expr2&gt; any, ...])
```

支持引擎：SPARKSQL, PRESTO

使用说明：返回参数的64位哈希值。 `SPARKSQL` 与 `PRESTO` 计算方式不一致，可能会得到不同的结果

返回类型：bigint

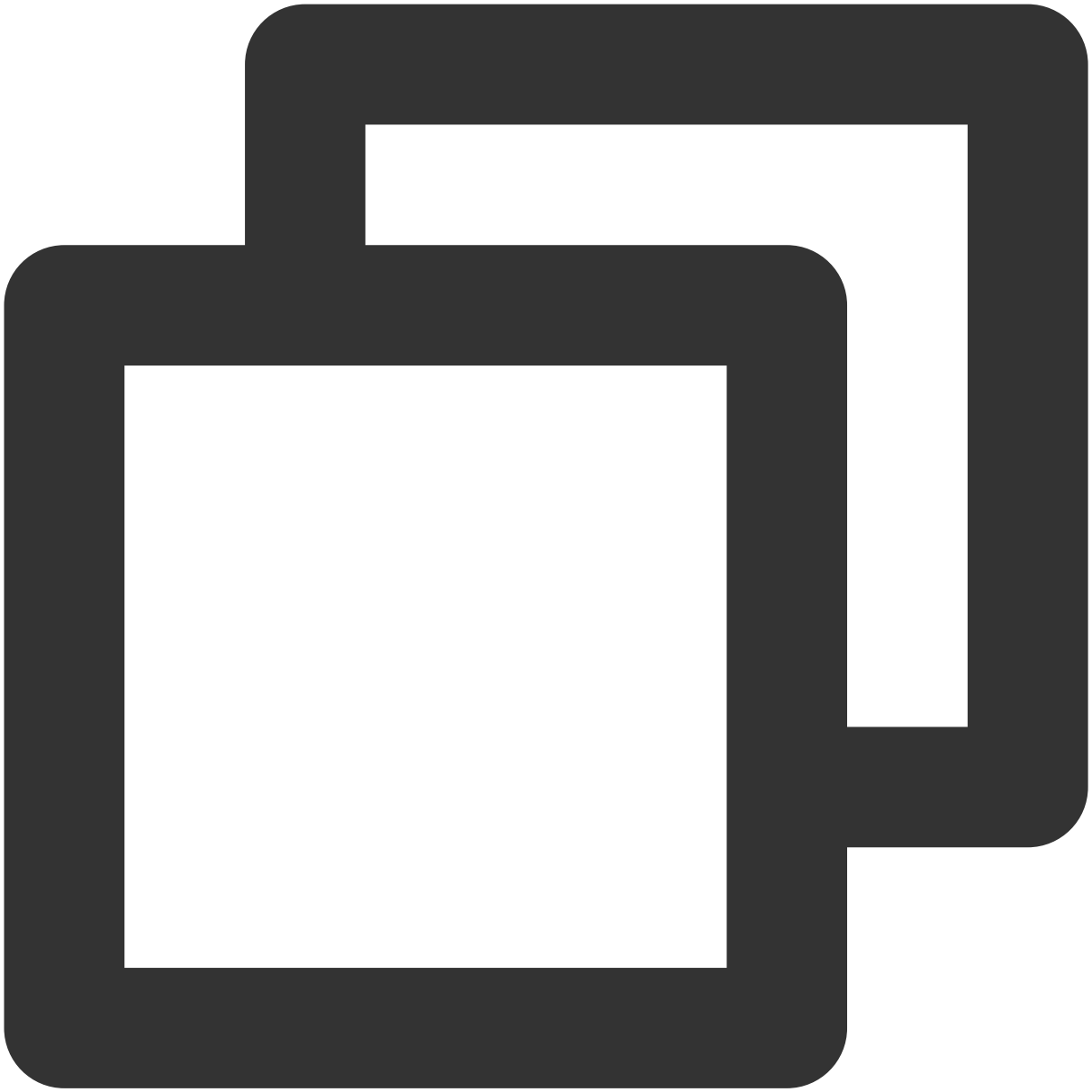
示例：



```
> SELECT xxhash64('tencent', array(123), 2);
-1900074178543885261
```

SHA

函数语法：



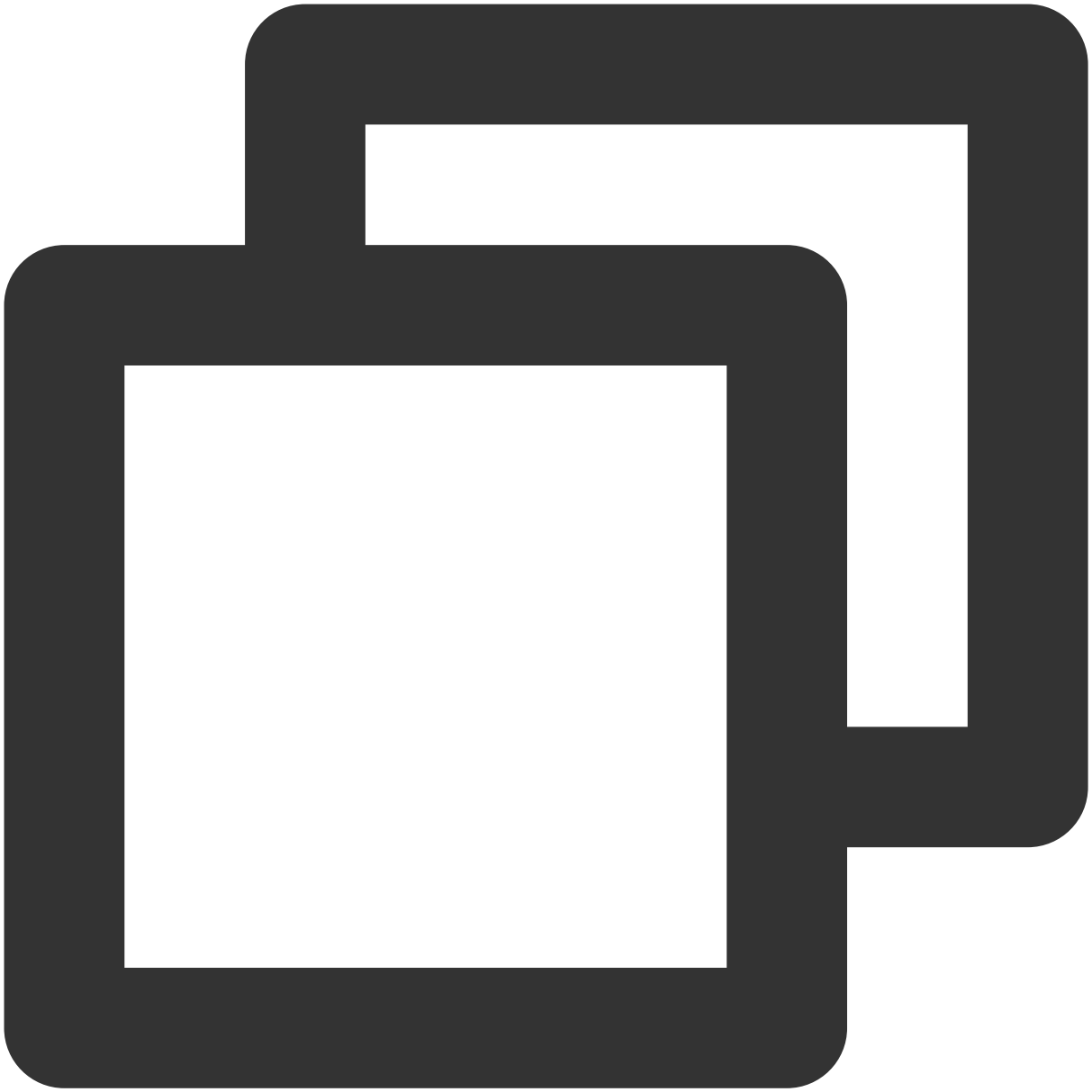
```
SHA(<expr> string|binary)
```

支持引擎：SPARKSQL, PRESTO

使用说明：以十六进制字符串形式返回 `expr` 的sha1哈希值。

返回类型： `string`

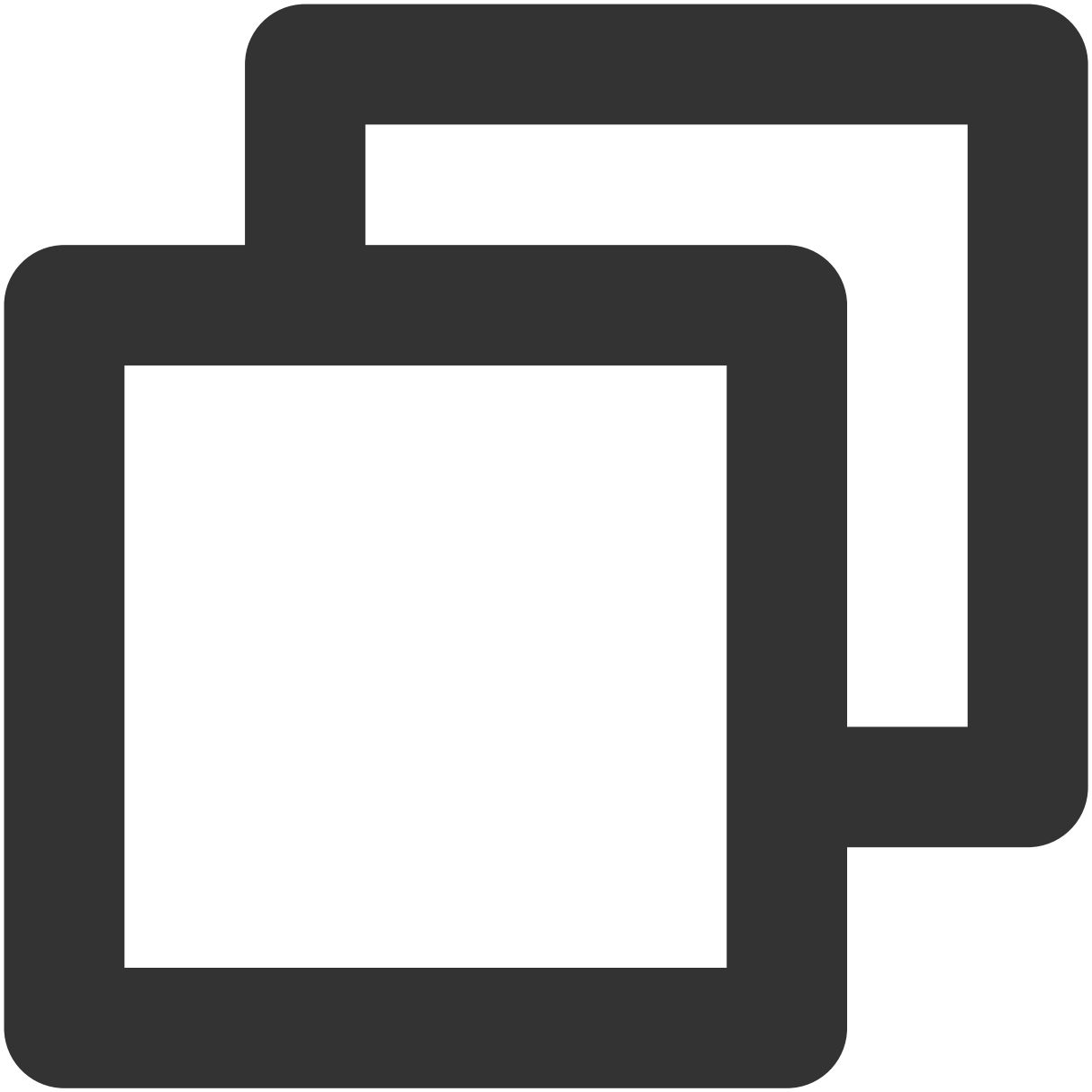
示例：



```
> select sha('tencent');
f94b2c96e2f127726ef4bcec6bc779f0f2e7888f
```

SHA1

函数语法：



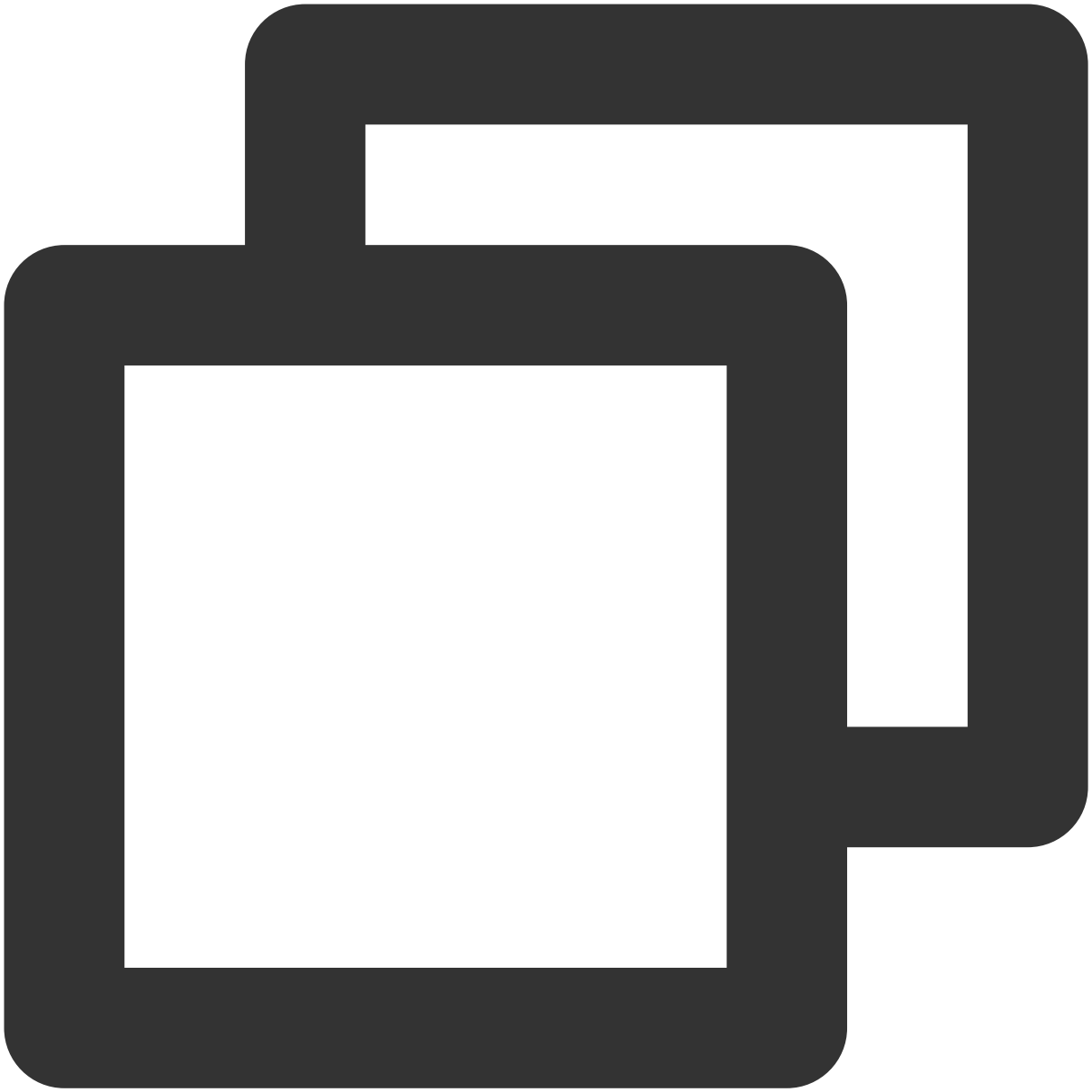
```
SHA1(<expr> string|binary)
```

支持引擎：SPARKSQL, PRESTO

使用说明：以十六进制字符串形式返回 `expr` 的sha1哈希值。

返回类型： `string`

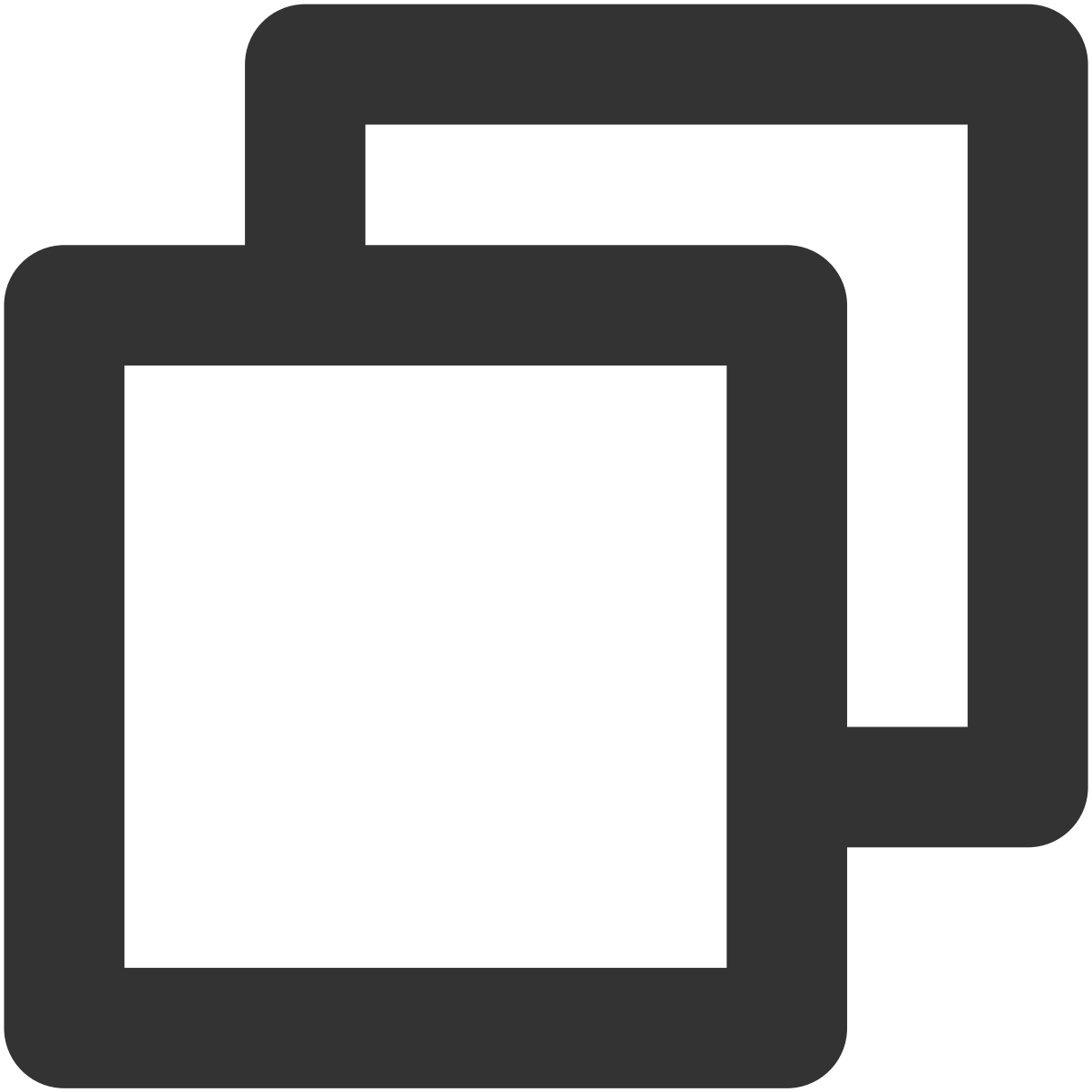
示例：



```
> select sha1('tencent');
f94b2c96e2f127726ef4bcec6bc779f0f2e7888f
```

SHA2

函数语法：



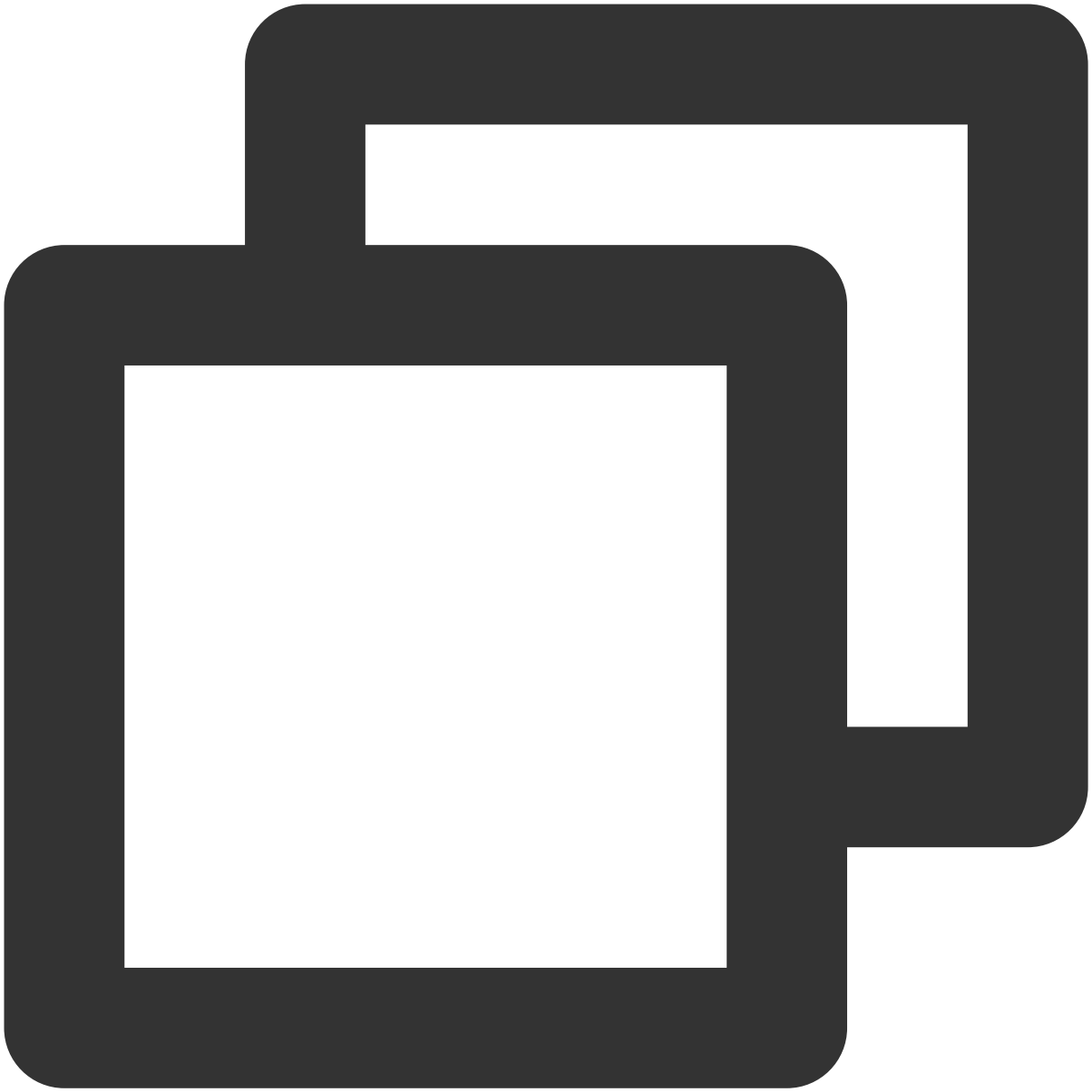
```
SHA2(expr string|binary, bitLength int)
```

支持引擎：SPARKSQL, PRESTO

使用说明：以十六进制字符串形式返回 `expr` 的SHA-2族的校验和。支持SHA-224、SHA-256、SHA-384和SHA-512。位长度0等于256。

返回类型： `string`

示例：



```
> select sha2('tencent', 256);
9c8ae69b84f21a2e46df9edf0063a697afec050188ff2884ddc8ab32b5e58c43
```

AES_ENCRYPT

函数语法：

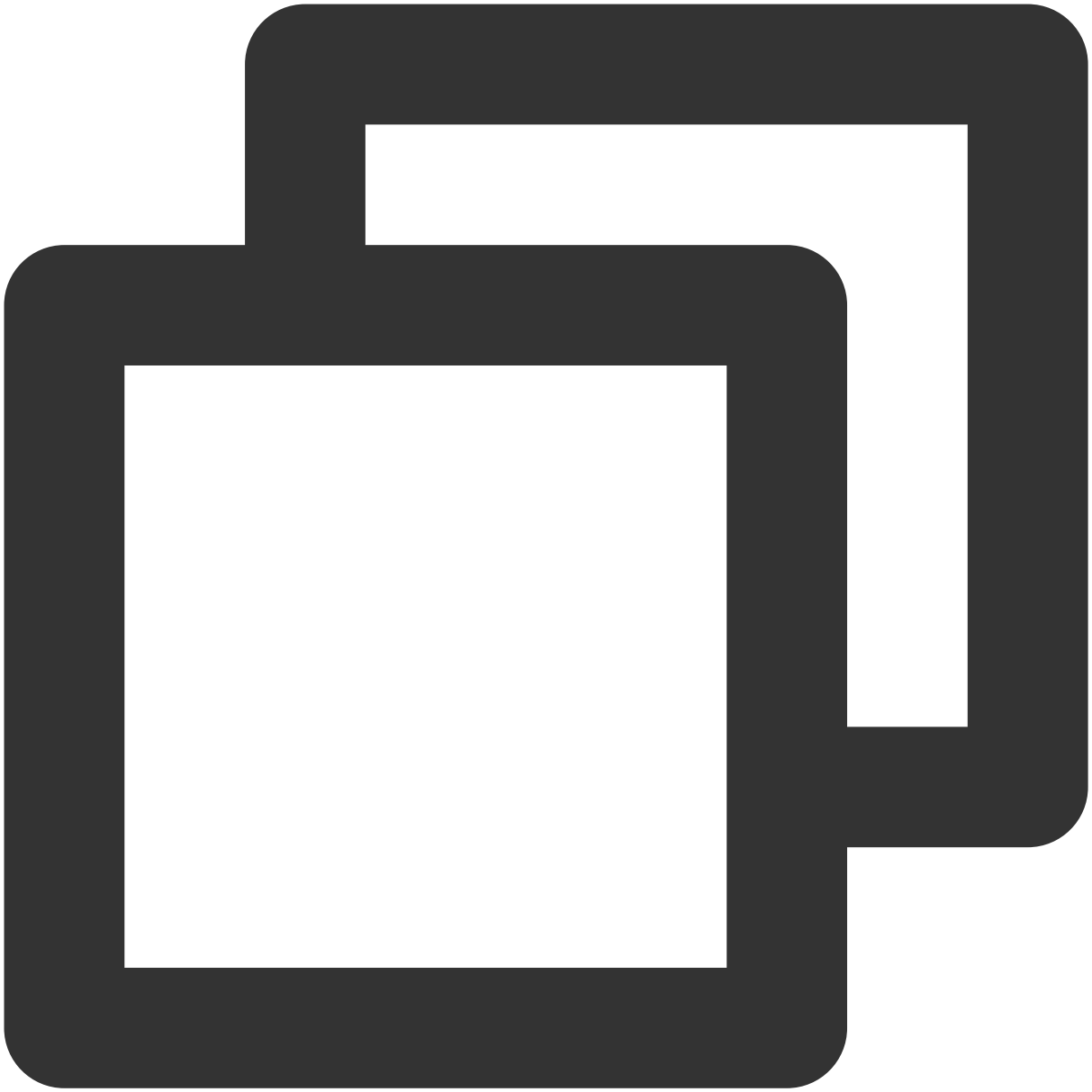
```
AES_ENCRYPT(<expr> string|binary, <key> string|binary)
```

支持引擎： PRESTO

使用说明：使用AES算法加密 `expr`

返回类型： `binary`

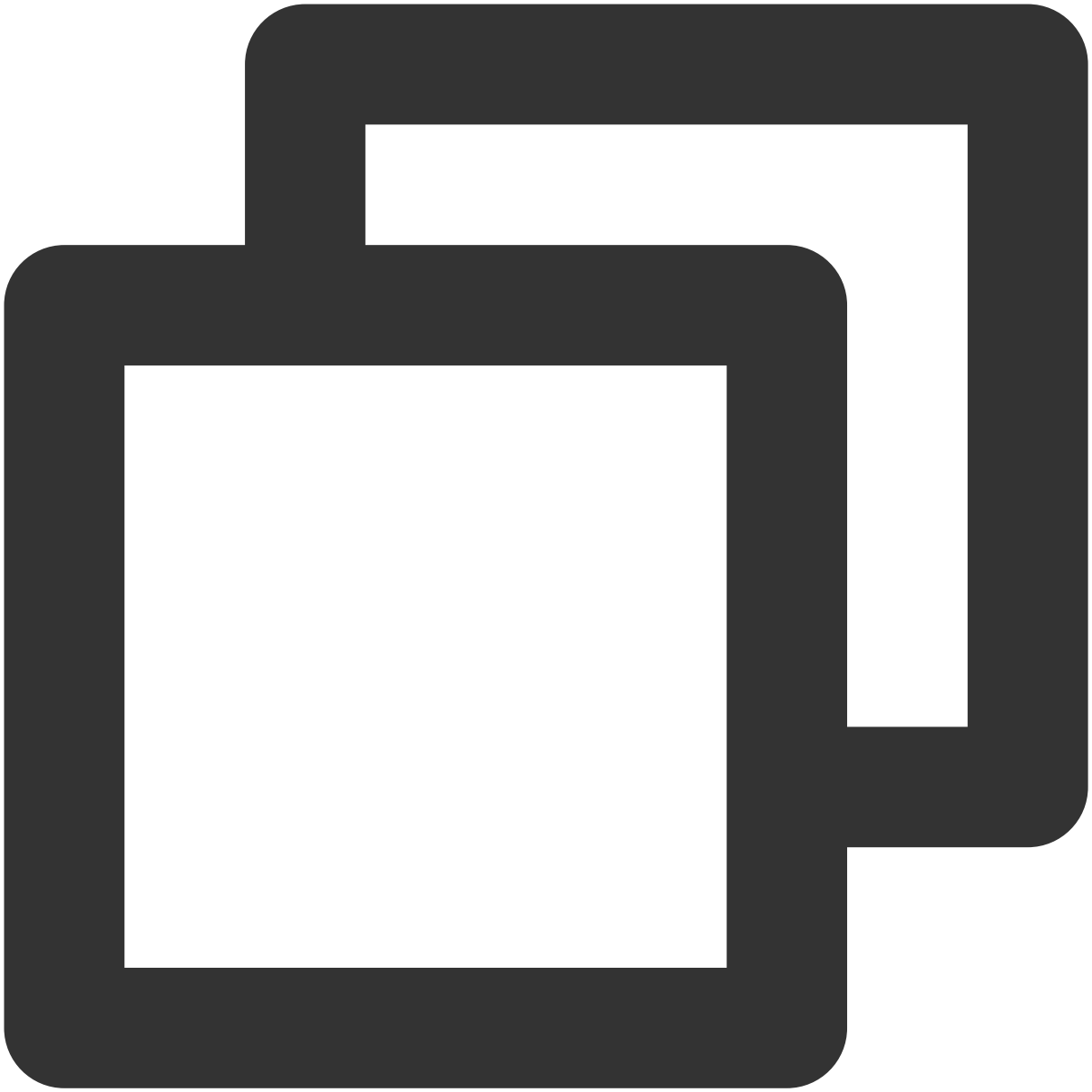
示例：



```
> SELECT hex(aes_encrypt('tencent', '0000111122223333'));
B99B99CE3359A736DBB9811ED8815C01
```

AES_DECRYPT

函数语法：



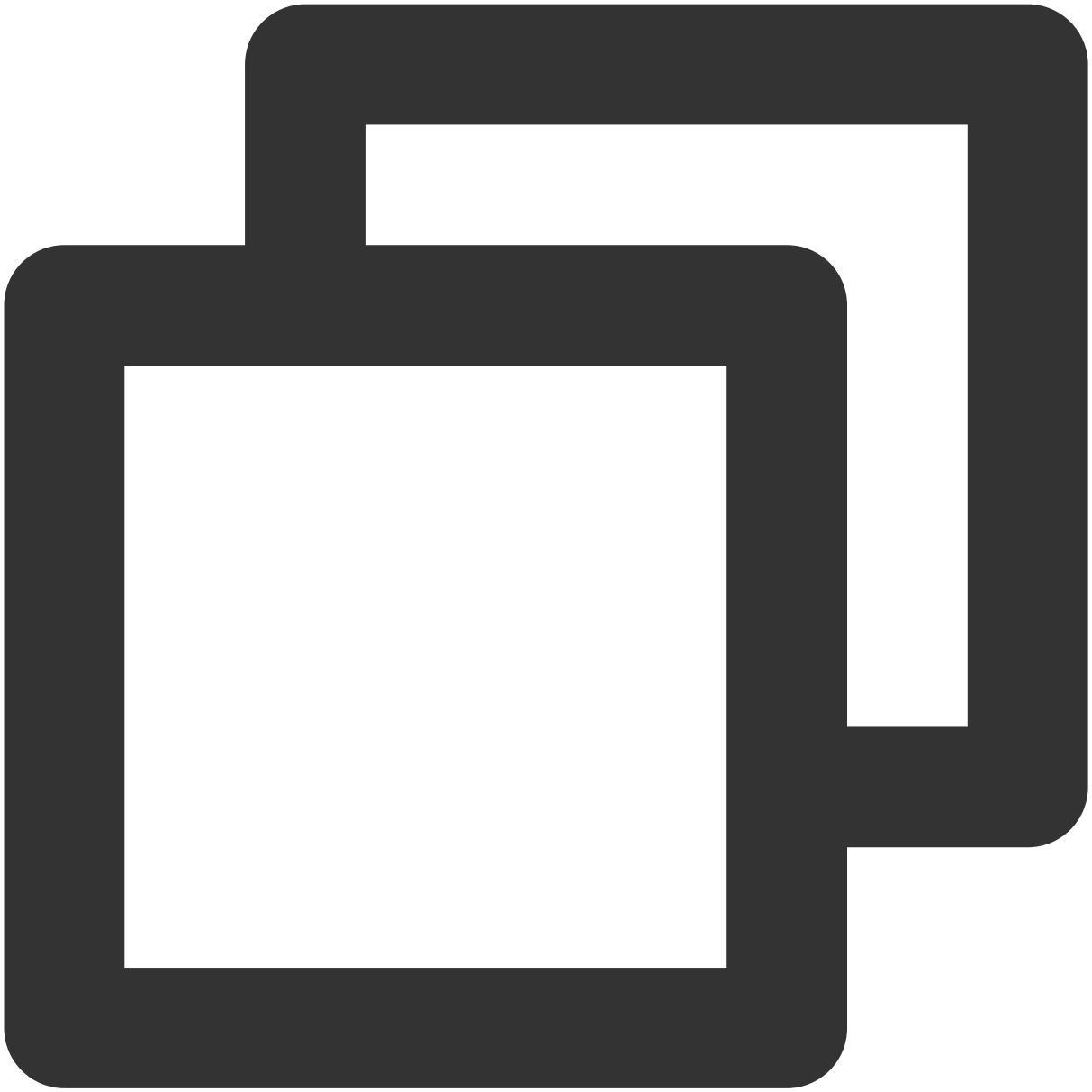
```
AES_DECRYPT(<expr> string|binary, <key> string|binary)
```

支持引擎： `PRESTO`

使用说明：使用AES算法解密 `expr`

返回类型： `binary`

示例：



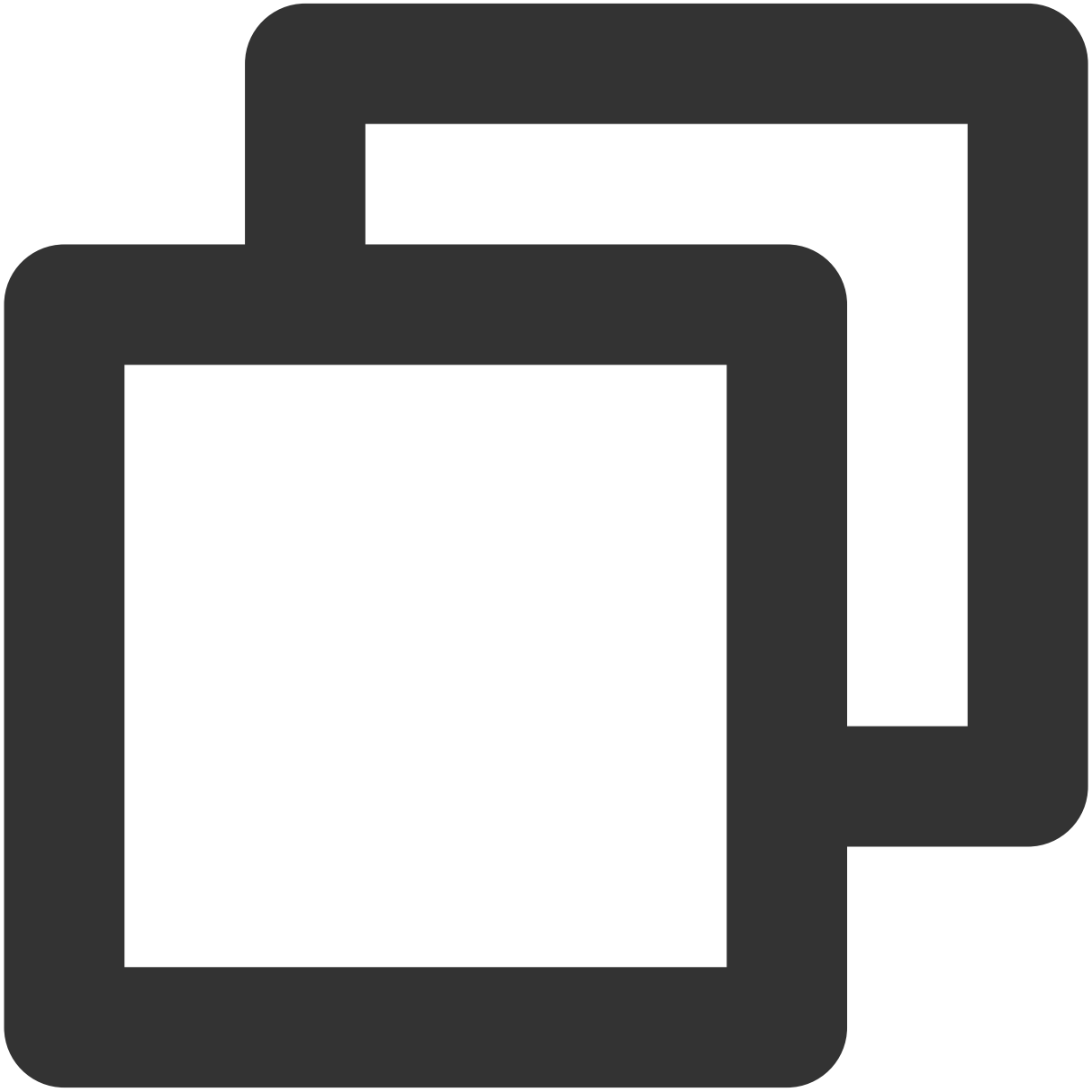
```
> SELECT aes_decrypt(unhex('B99B99CE3359A736DBB9811ED8815C01'), '0000111122223333')
tencent
```

位运算函数

最近更新时间：2024-08-07 17:31:40

BIT_COUNT

函数语法：



```
BIT_COUNT(<expr> bigint|boolean)
```

支持引擎：SparkSQL 、Presto

使用说明：将 `expr` 设置为无符号64位整数，返回其位为1的个数，如果参数为 NULL，则返回 NULL。

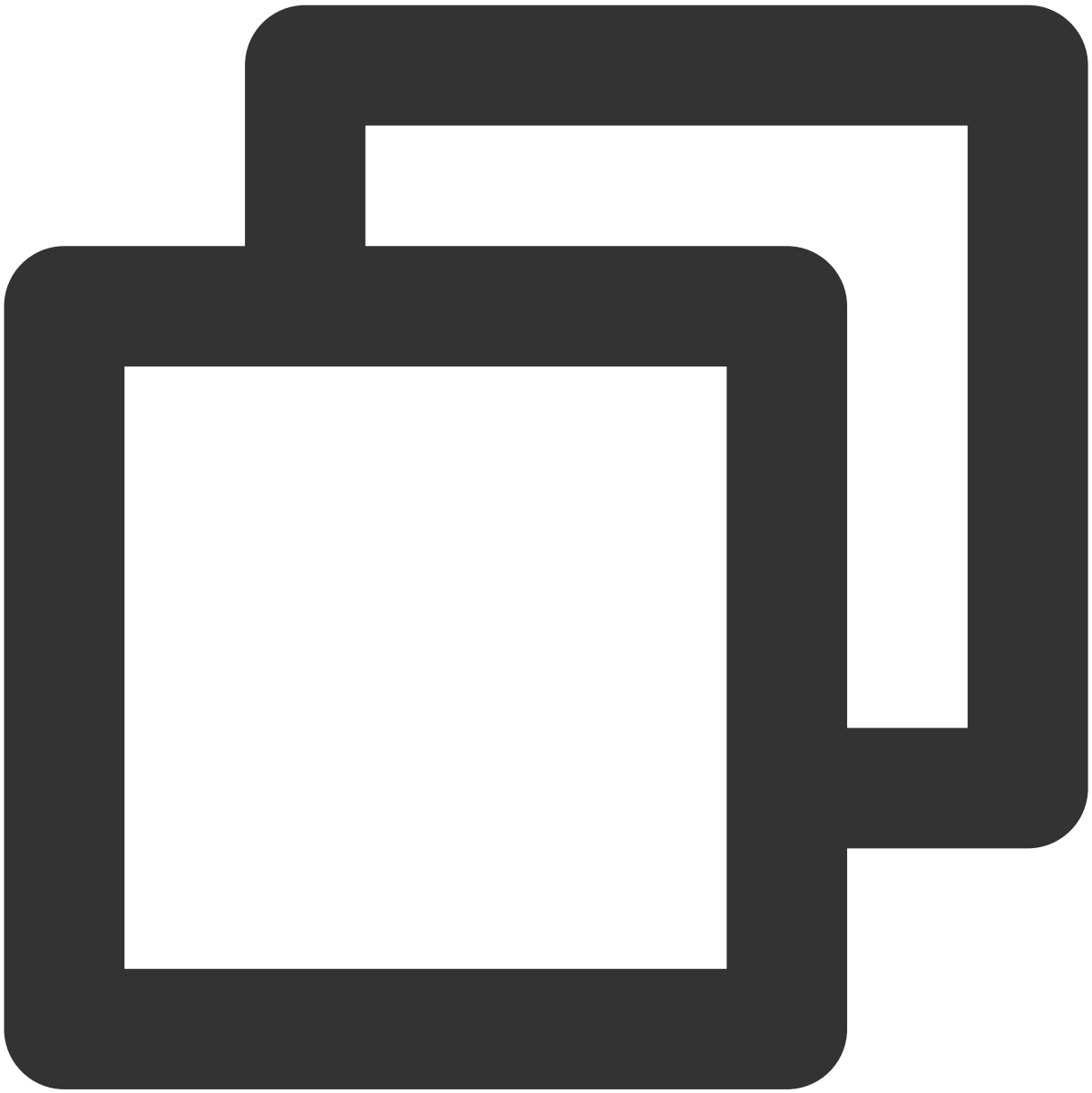
返回类型： `integer`

示例：

```
> SELECT bit_count(5);
 2
```

BIT_GET

函数语法：



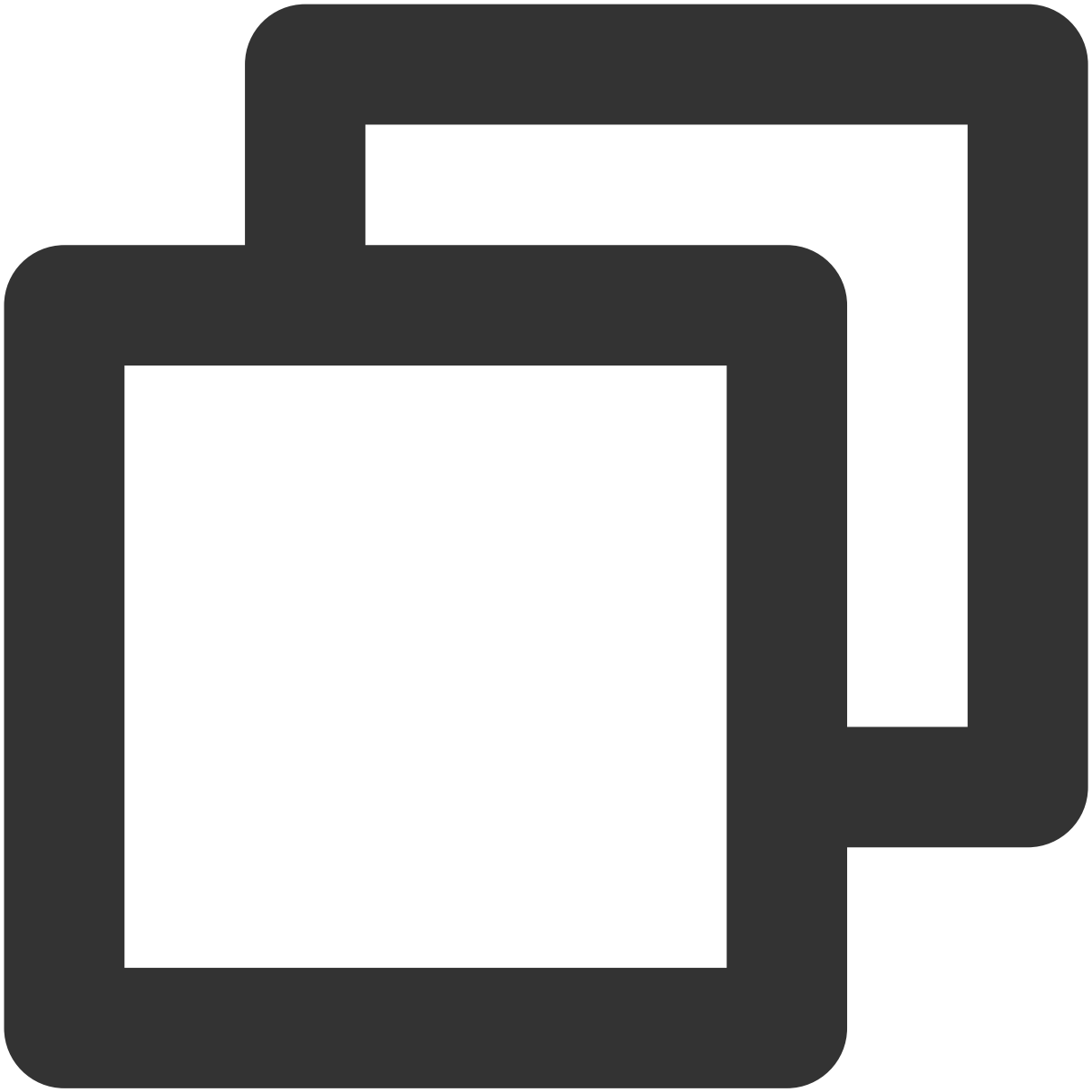
```
BIT_GET(<expr> bigint|boolean, <pos> integer)
```

支持引擎：SparkSQL 、Presto

使用说明：返回指定位置位（0或1）的值。位置从右到左编号，从零开始。位置参数不能为负。

返回类型： `integer`

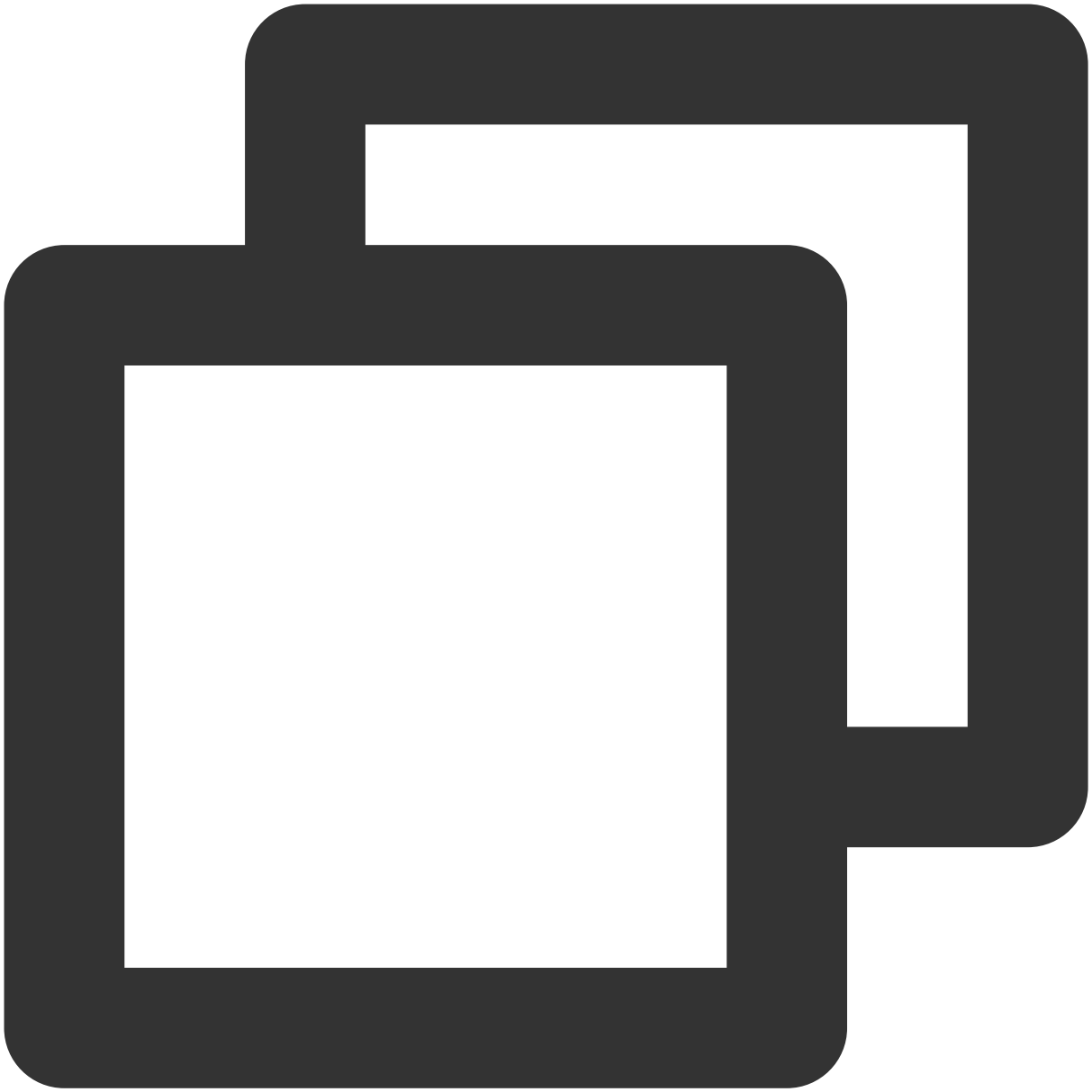
示例：



```
> SELECT bit_get(11, 0);
 1
> SELECT bit_get(11, 2);
 0
```

GETBIT

函数语法：



```
GETBIT(<expr> bigint|boolean, <pos> integer)
```

支持引擎：SparkSQL 、Presto

使用说明：返回指定位置位（0或1）的值。位置从右到左编号，从零开始。位置参数不能为负。

返回类型： `integer`

示例：

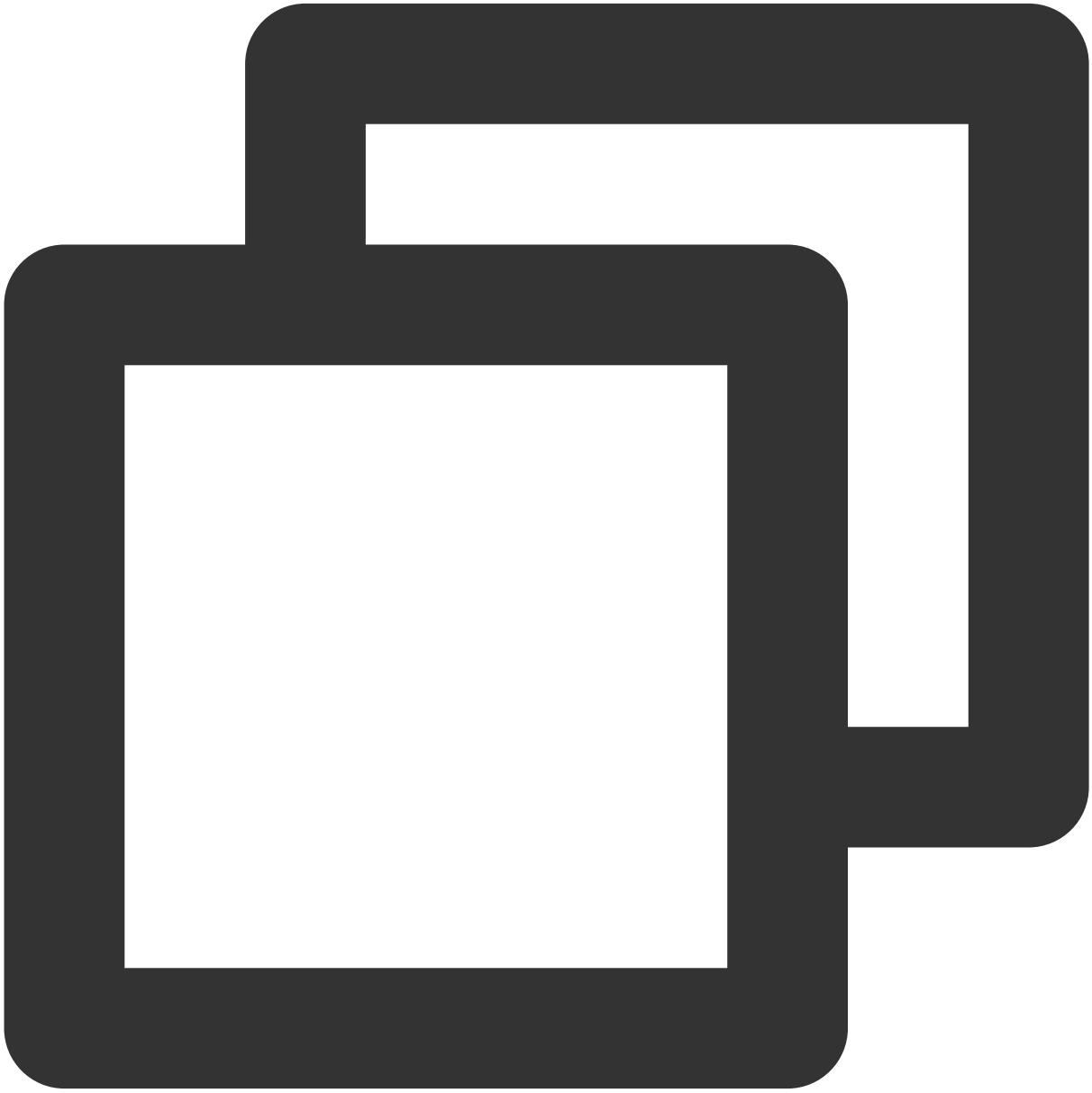
```
> SELECT getbit(11, 0);
 1
> SELECT getbit(11, 2);
 0
```

集合函数

最近更新时间：2024-08-07 17:31:54

ARRAY

函数语法：



```
ARRAY(<e1> T, ..., <en> T)
```

支持引擎：SparkSQL、Presto。

使用说明：根据给定元素生成数组

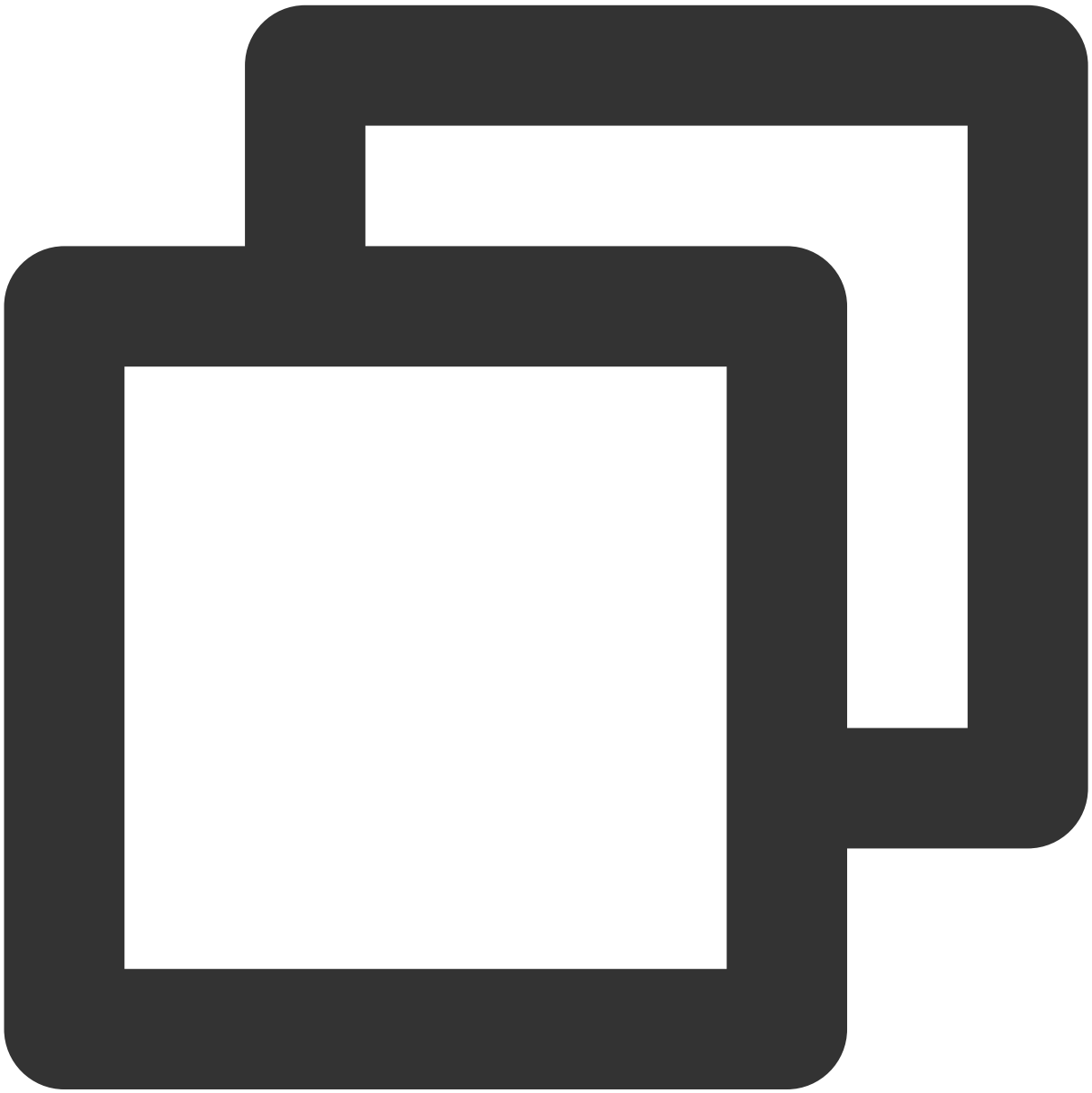
返回类型：array<T>。

示例：

```
> SELECT array(1, 2, 3);
 [1,2,3]
```

FILTER

函数语法：



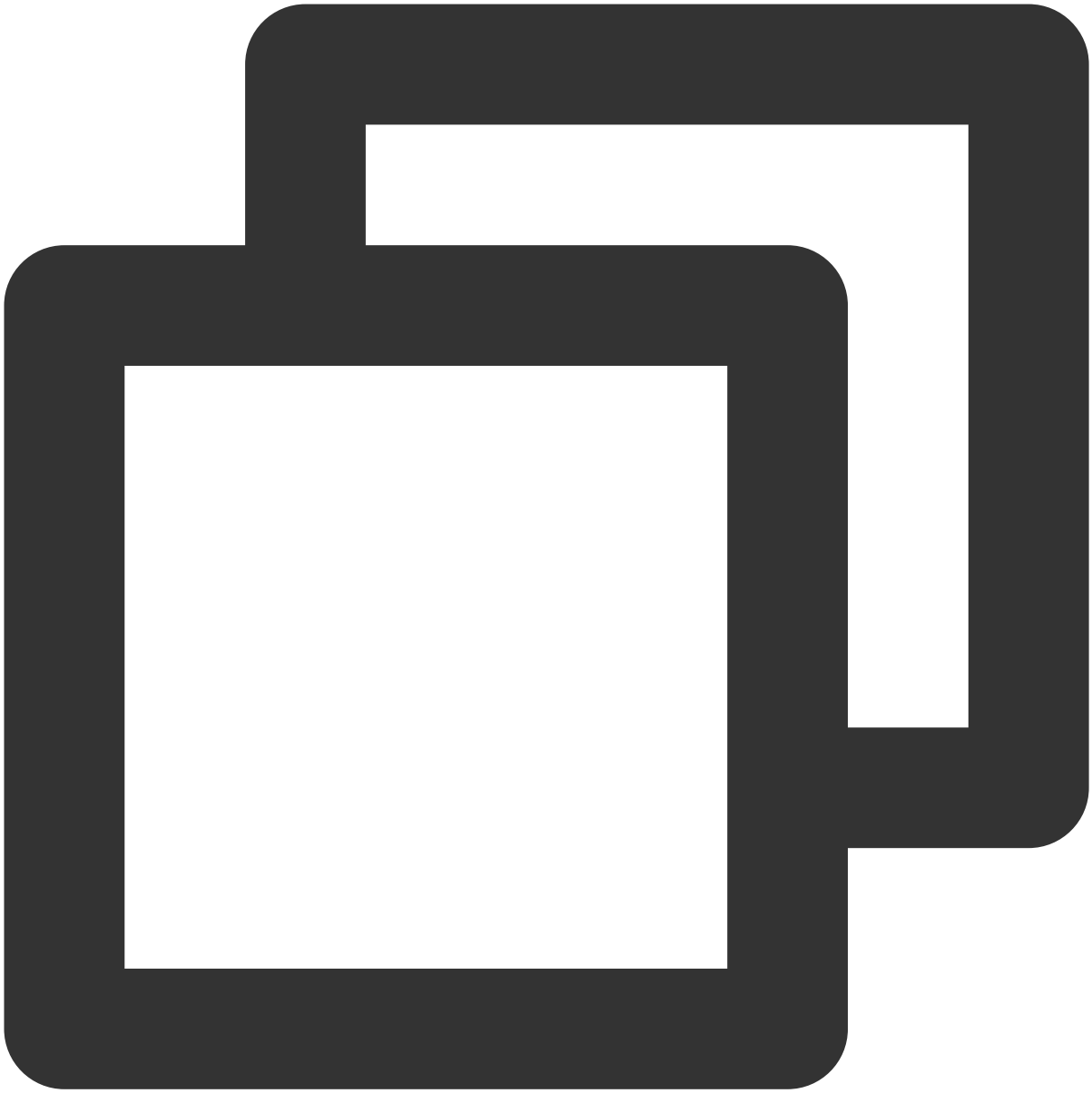
```
FILTER(<expr> array<T>, <predicate> function(T[, integer])->boolean)
```

支持引擎：SparkSQL。

使用说明：使用给定谓词过滤输入数组。

返回类型：array<T>。

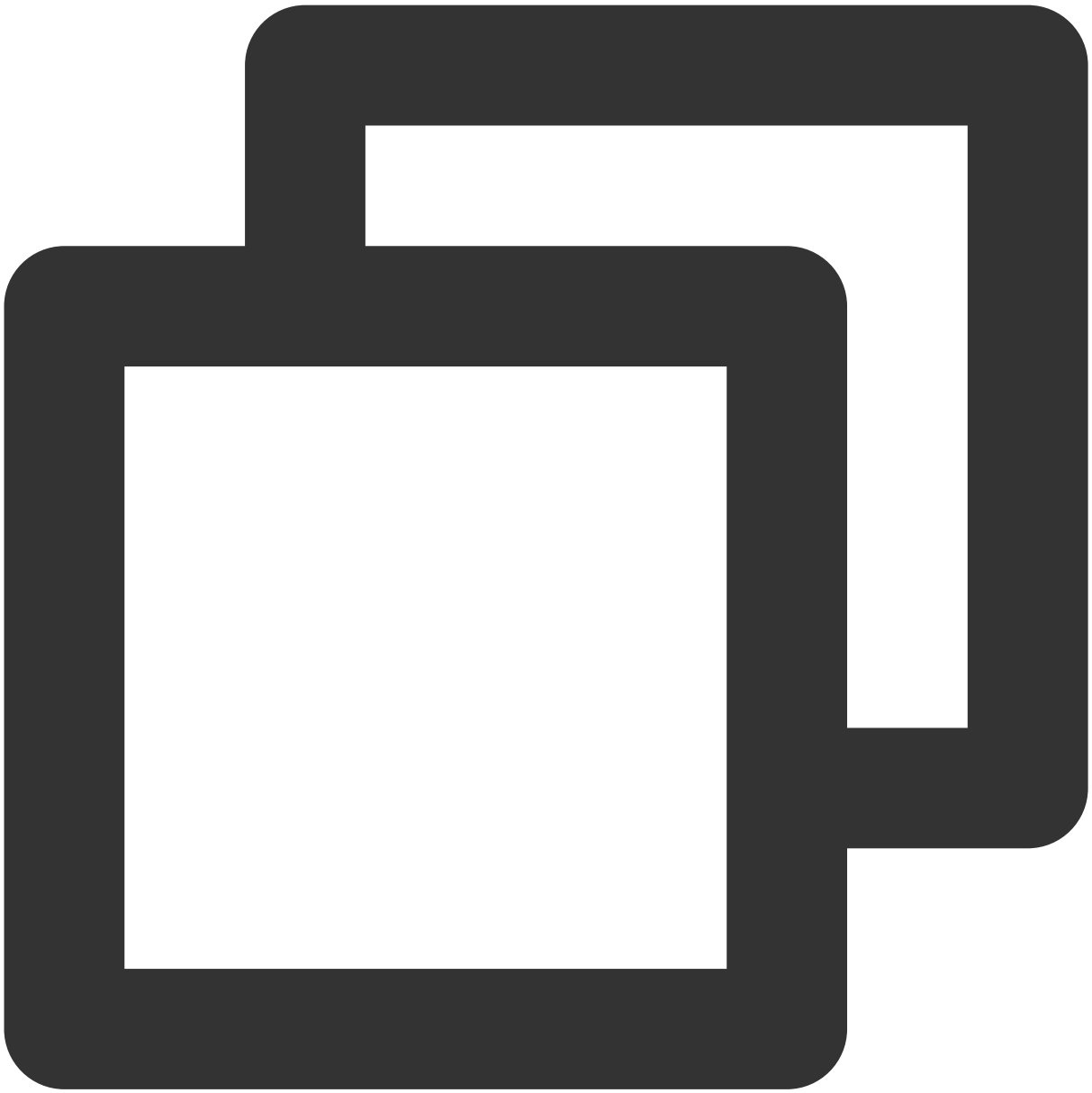
示例：



```
> SELECT `filter`(array(1, 2, 3), x -> x % 2 == 1);
 [1,3]
> SELECT `filter`(array(0, 2, 3), (x, i) -> x > i);
 [2,3]
> SELECT `filter`(array(0, null, 2, 3, null), x -> x IS NOT NULL);
 [0,2,3]
```

TRANSFORM

函数语法：



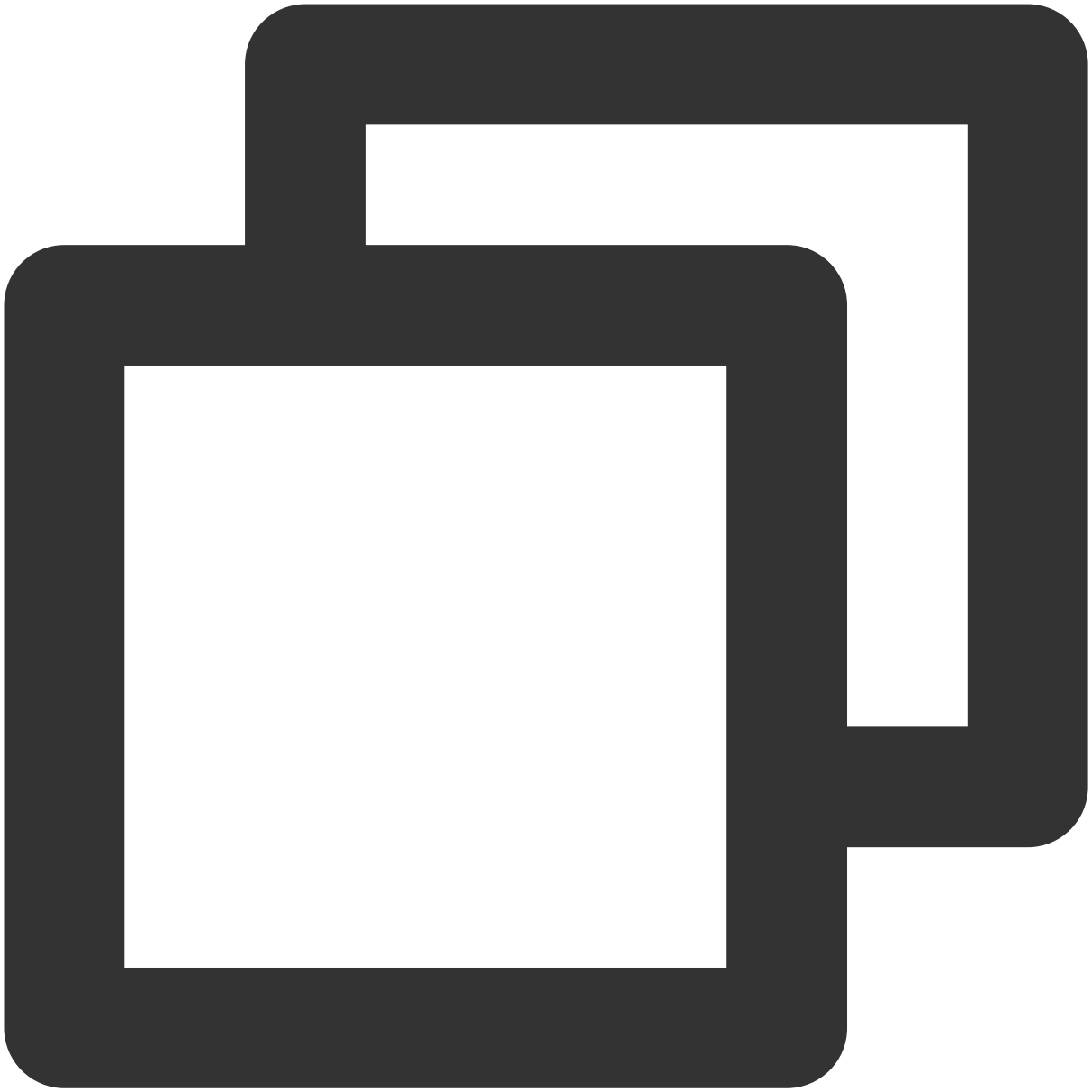
```
TRANSFORM(<expr> array<T>, <func> function(T[, integer])->U)
```

支持引擎：SparkSQL

使用说明：使用func变换数组中的元素。

返回类型：array<U>

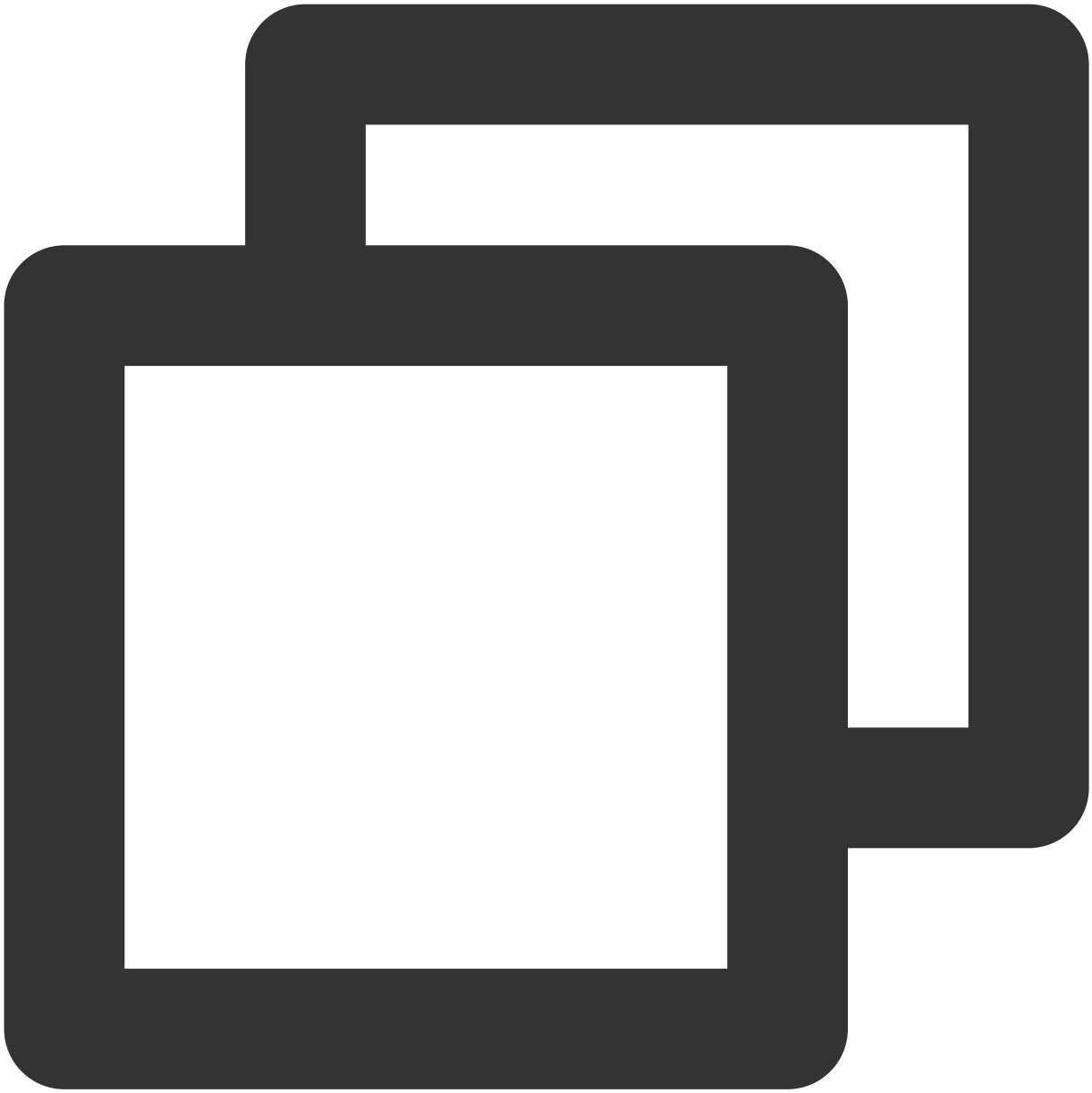
示例：



```
> SELECT transform(array(1, 2, 3), x -> x + 1);
 [2,3,4]
> SELECT transform(array(1, 2, 3), (x, i) -> x + i);
 [1,3,5]
```

ZIP_WITH

函数语法：



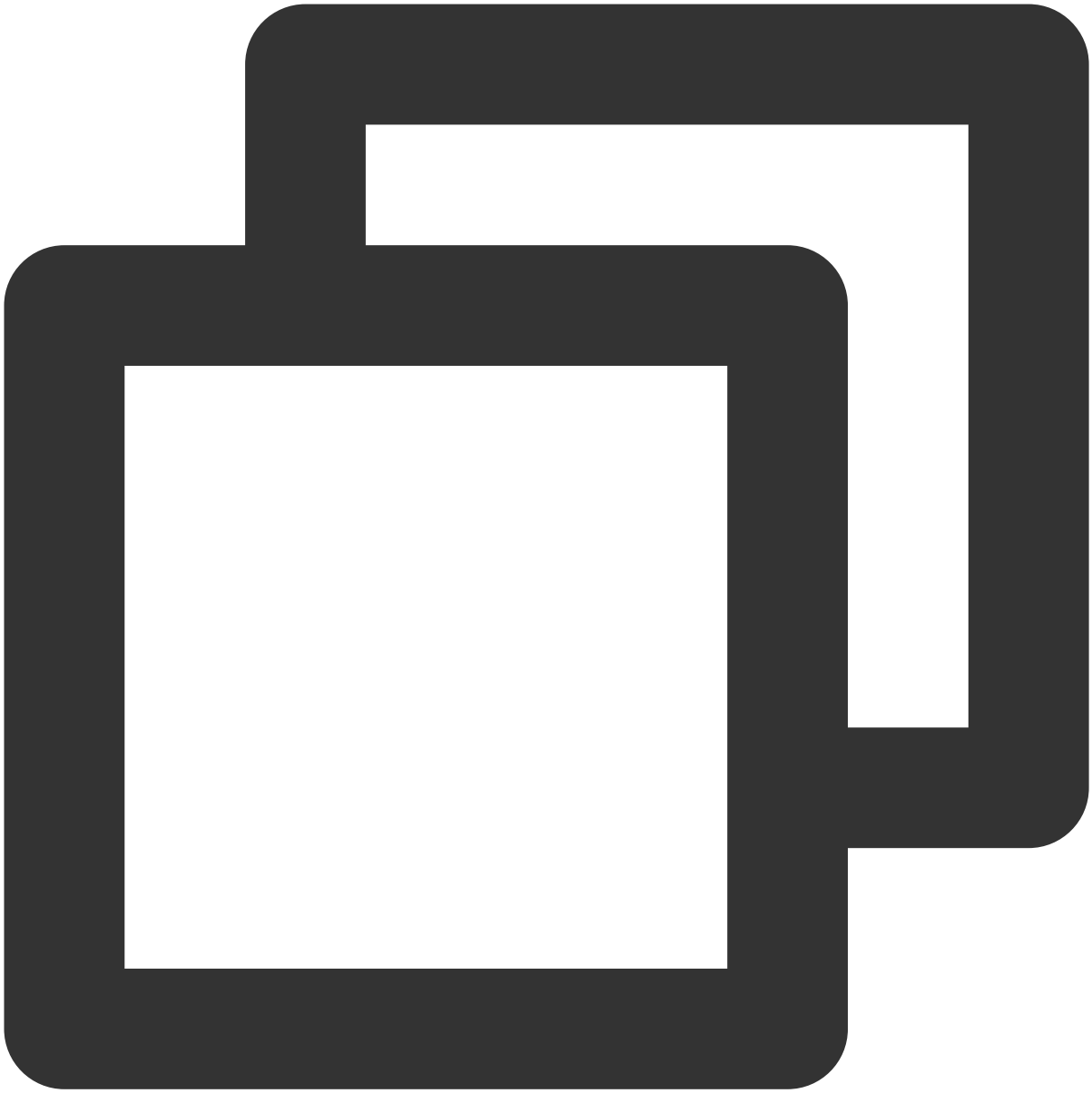
```
ZIP_WITH(<left> array<T>, <right> array<U>, <func> function(T, U)->R)
```

支持引擎：SparkSQL。

使用说明：使用函数将两个给定数组按元素合并为单个数组。如果一个数组较短，则在应用函数之前，在末尾追加null以匹配较长数组的长度。

返回类型：array<R>。

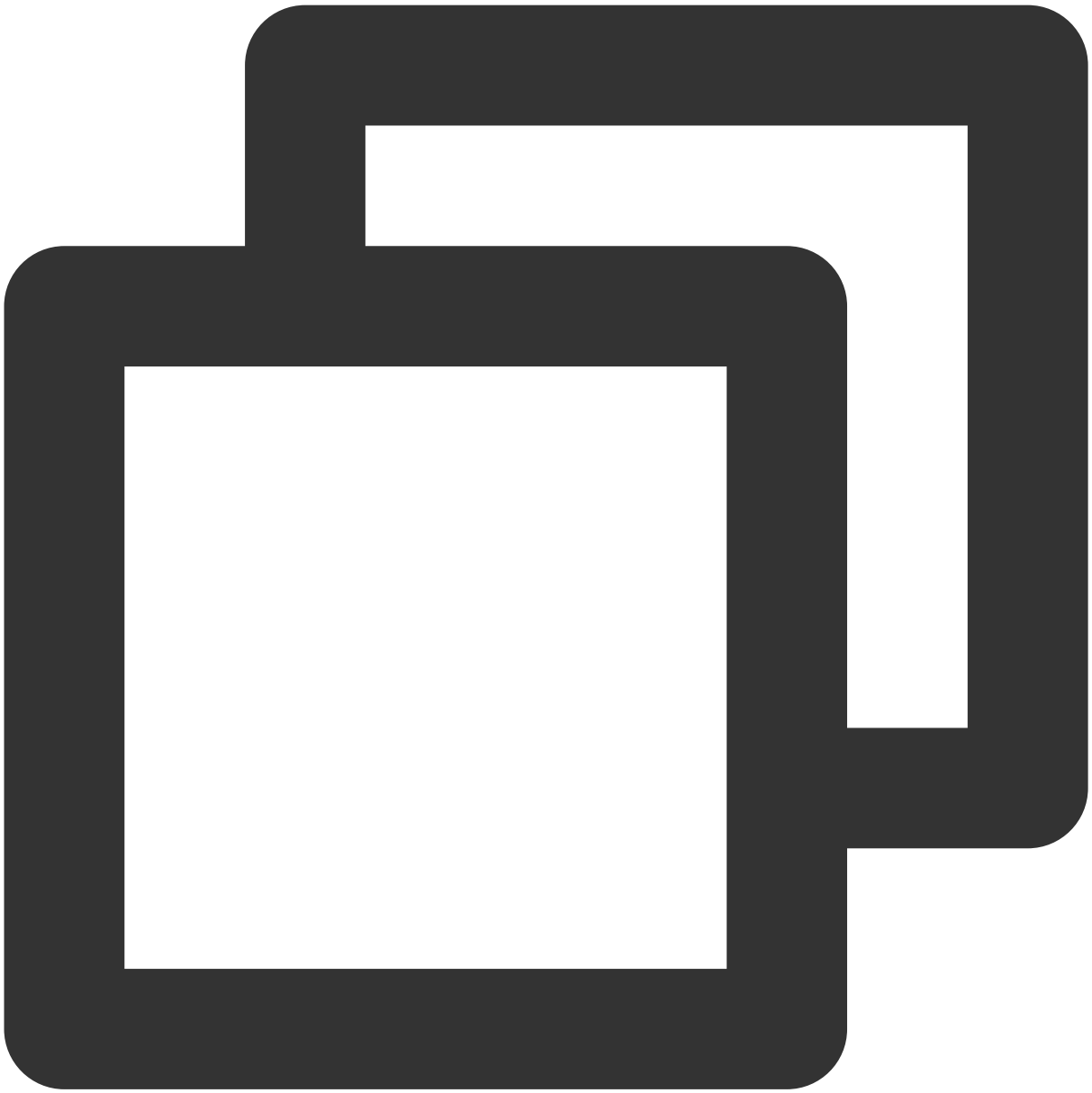
示例：



```
> SELECT zip_with(array(1, 2, 3), array('a', 'b', 'c'), (x, y) -> (y, x));
 [{"y":"a","x":1},{"y":"b","x":2},{"y":"c","x":3}]
> SELECT zip_with(array(1, 2), array(3, 4), (x, y) -> x + y);
 [4,6]
> SELECT zip_with(array('a', 'b', 'c'), array('d', 'e', 'f'), (x, y) -> concat(x, y
 ["ad","be","cf"]
```

FORALL

函数语法：



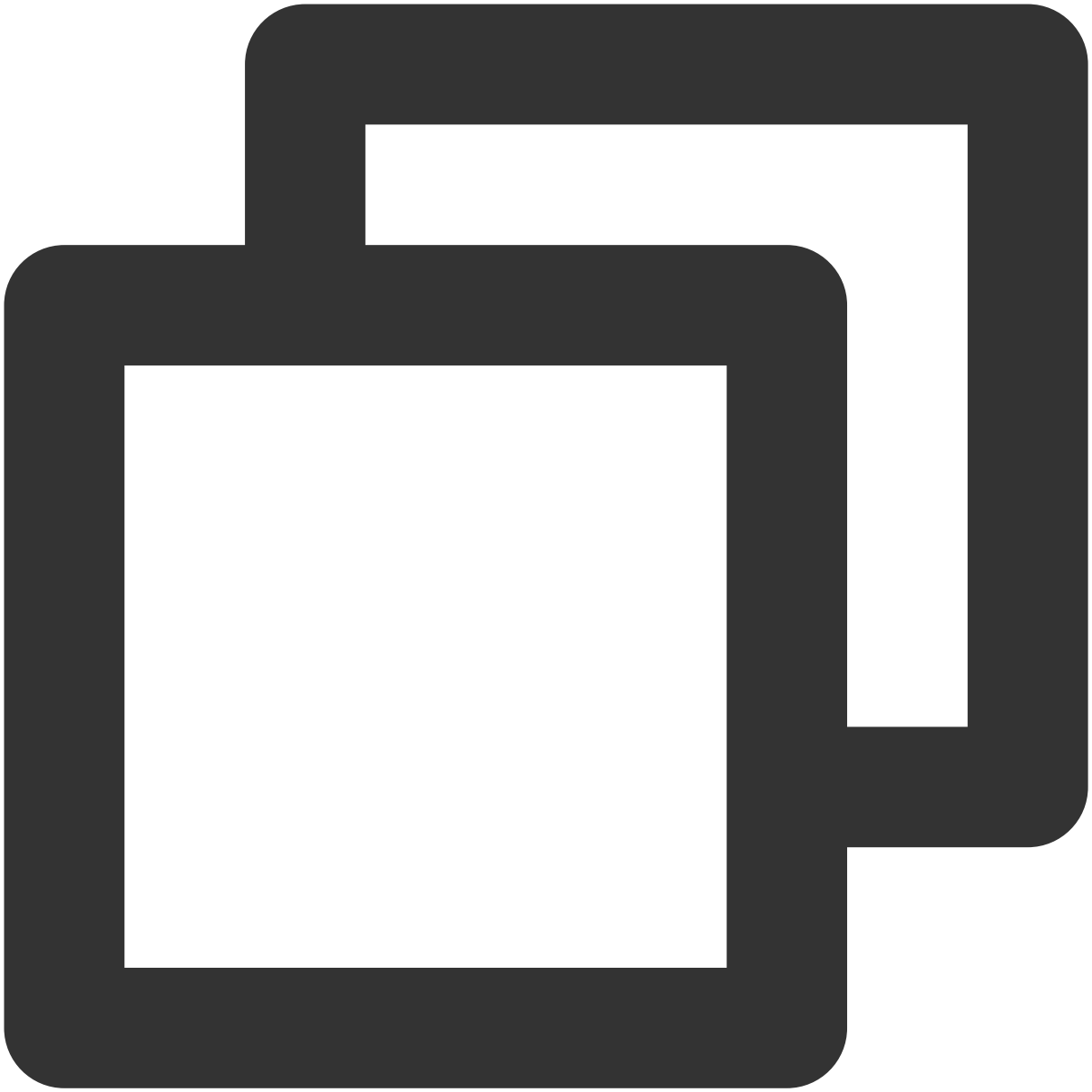
```
FORALL(<expr> array<T>, <pred> function(T)->boolean)
```

支持引擎：SparkSQL。

使用说明：测试谓词是否适用于数组中的所有元素。

返回类型：boolean。

示例：

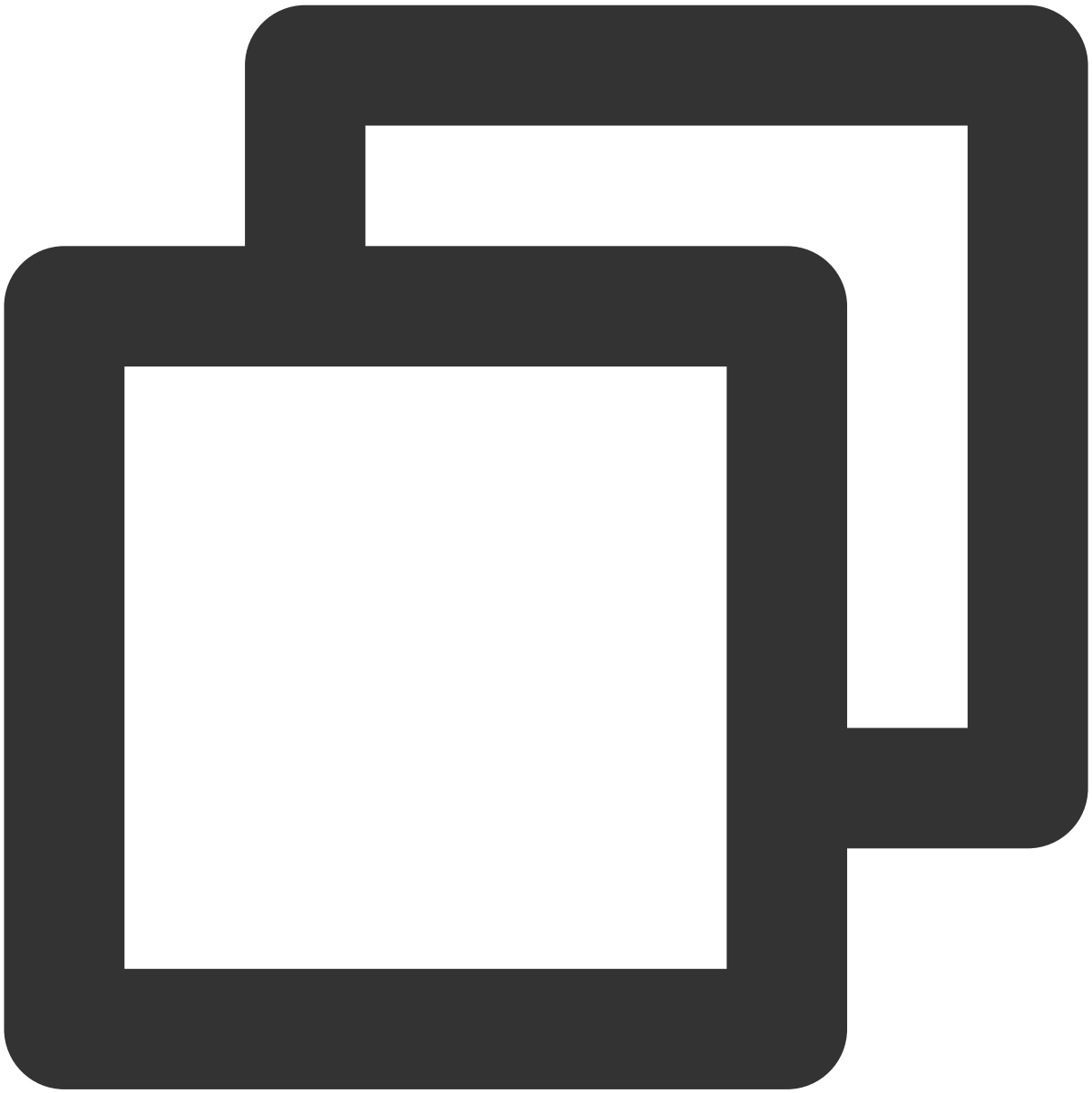


```
> SELECT forall(array(1, 2, 3), x -> x % 2 == 0);
 false
> SELECT forall(array(2, 4, 8), x -> x % 2 == 0);
 true
> SELECT forall(array(1, null, 3), x -> x % 2 == 0);
 false
> SELECT forall(array(2, null, 8), x -> x % 2 == 0);
 NULL
```

版权所有：腾讯云计算（北京）有限责任公司

AGGREGATE

函数语法：



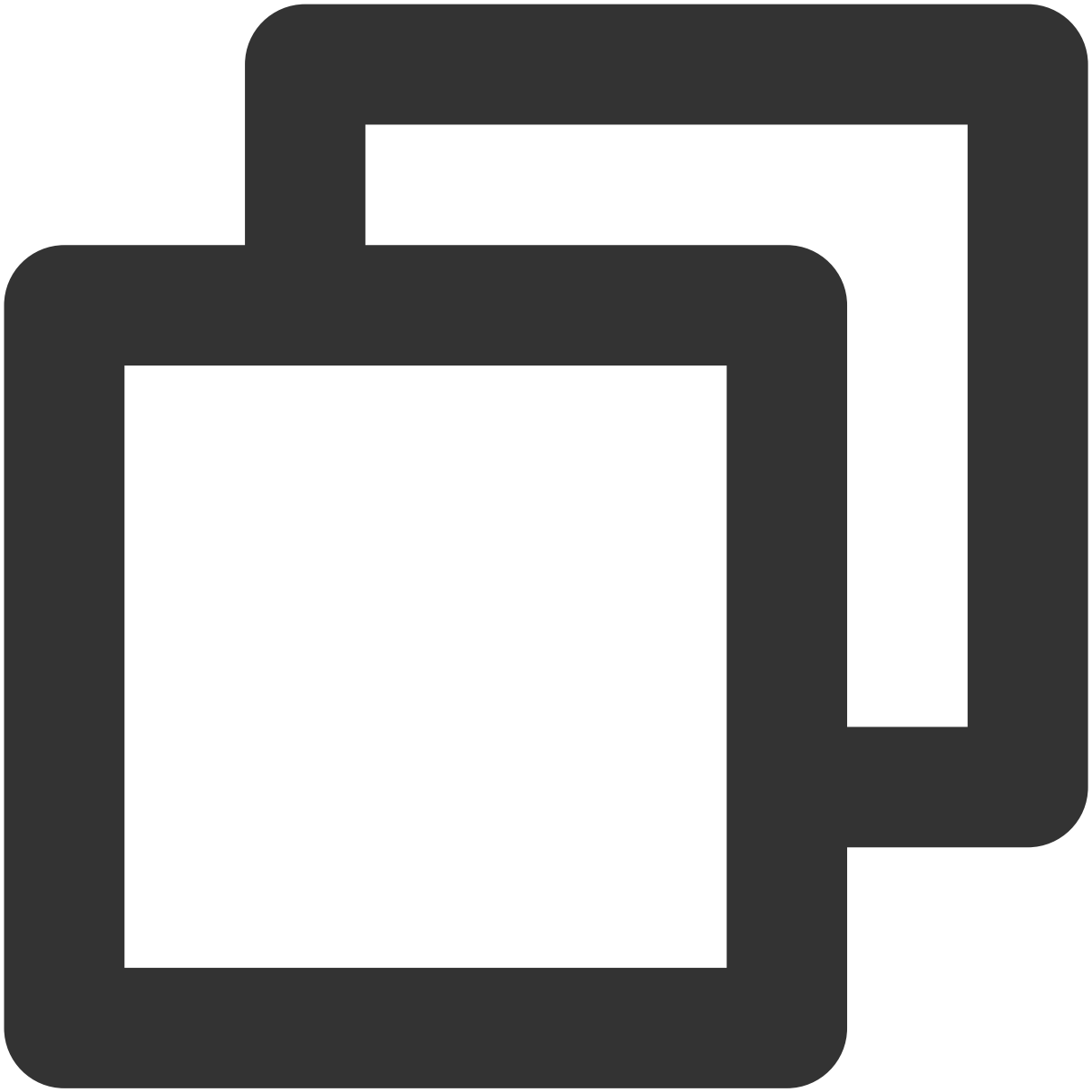
```
AGGREGATE(<expr> array<T>, <start> U, <merge> function(U, T)->U, <finish> function(
```

支持引擎：SparkSQL。

使用说明：聚合数组 expr 中的元素。

返回类型：R。

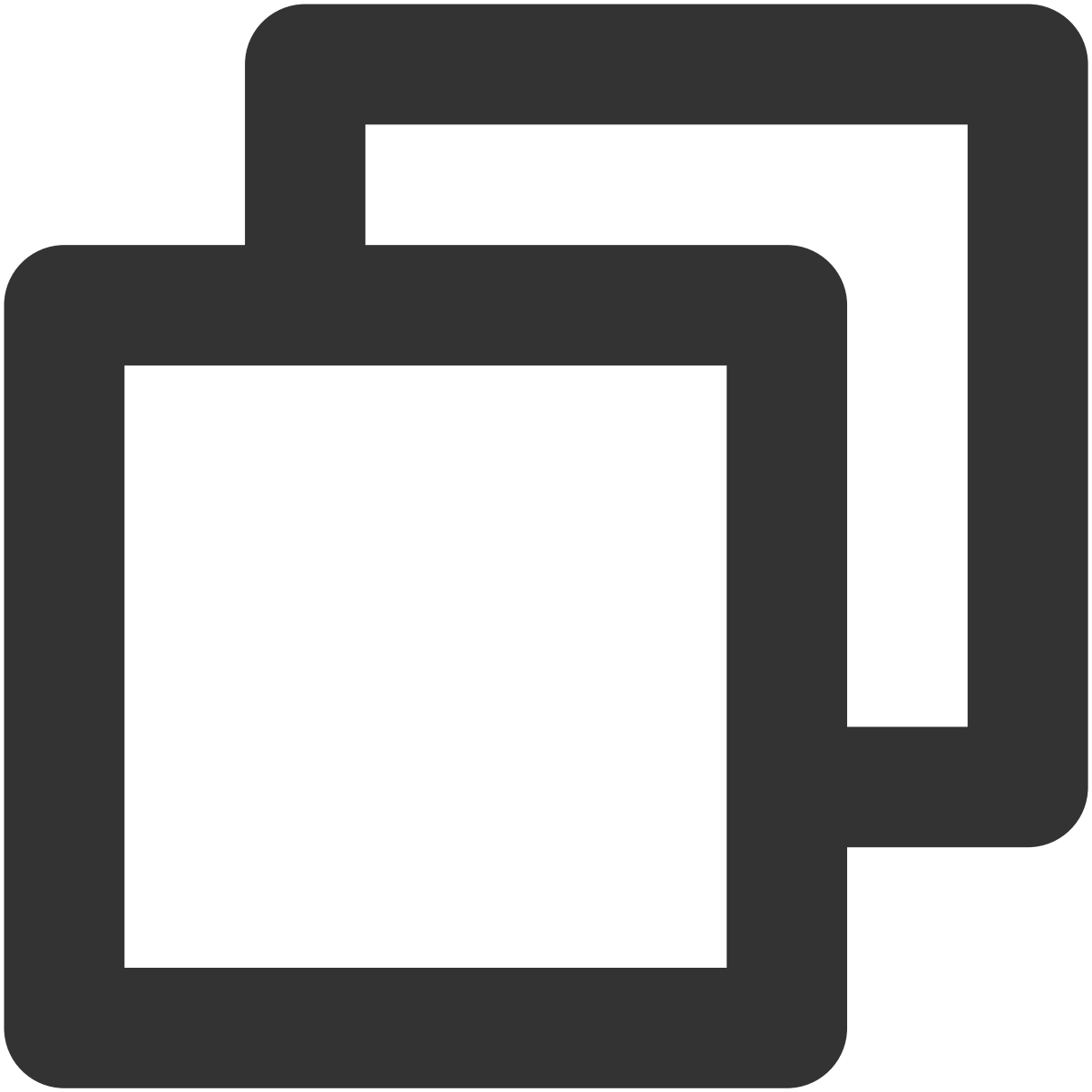
示例：



```
> SELECT aggregate(array(1, 2, 3), 0, (acc, x) -> acc + x);
 6
> SELECT aggregate(array(1, 2, 3), 0, (acc, x) -> acc + x, acc -> acc * 10);
 60
```

EXISTS

函数语法：



```
EXISTS(<expr> array<T>, <pred> function(T)->boolean)
```

支持引擎：SparkSQL。

使用说明：测试谓词是否适用于数组中的一个或多个元素。

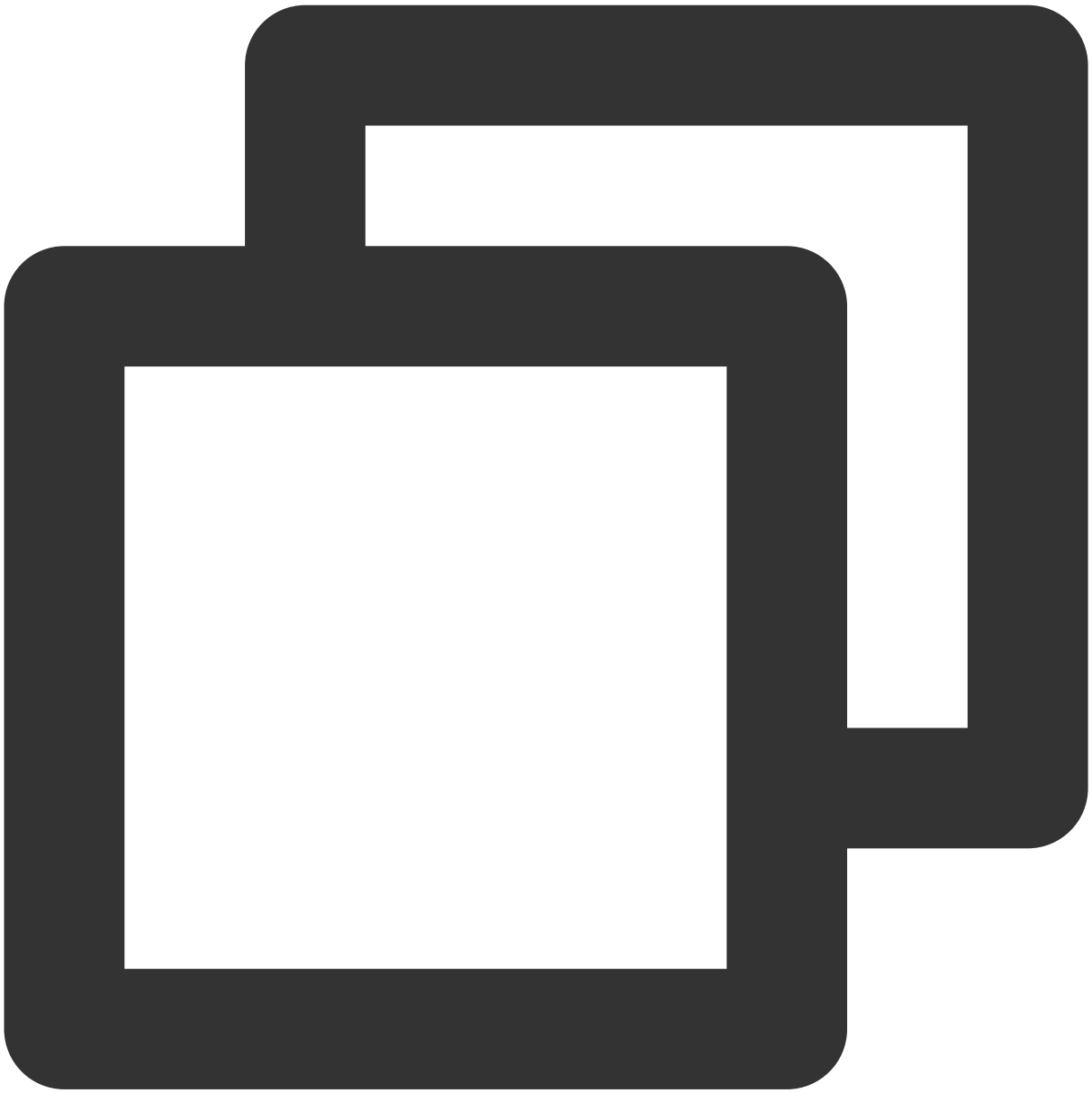
返回类型：boolean。

示例：

```
> SELECT exists(array(1, 2, 3), x -> x % 2 == 0);
 true> SELECT exists(array(1, 2, 3), x -> x % 2 == 10);
 false
> SELECT exists(array(1, null, 3), x -> x % 2 == 0);
 NULL
> SELECT exists(array(0, null, 2, 3, null), x -> x IS NULL);
 true
> SELECT exists(array(1, 2, 3), x -> x IS NULL);
 false
```

ARRAY_CONTAINS

函数语法：



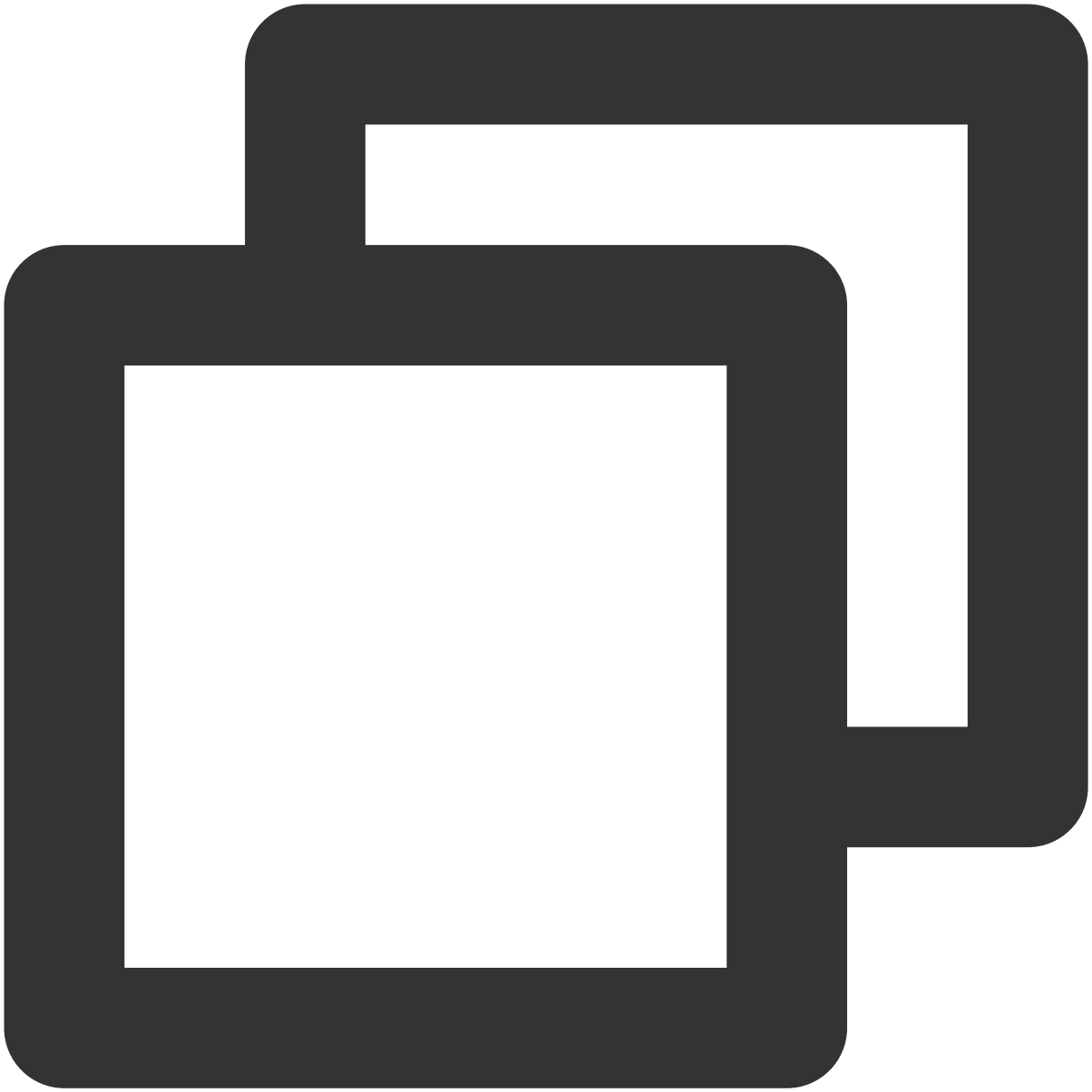
```
ARRAY_CONTAINS(<expr> array<T>, <value> T)
```

支持引擎：SparkSQL、Presto。

使用说明：如果数组包含 value，则返回 true。

返回类型：boolean。

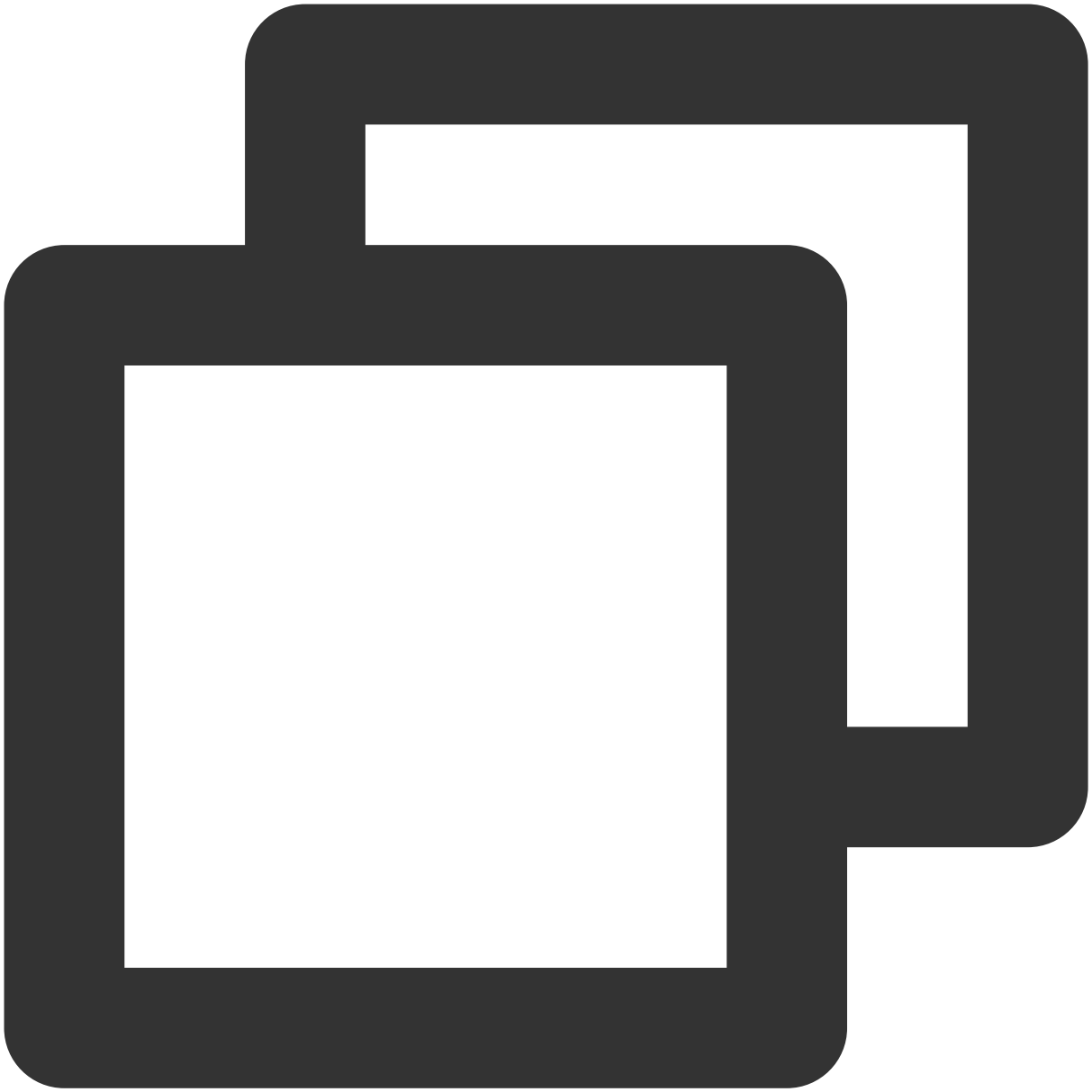
示例：



```
> SELECT array_contains(array(1, 2, 3), 2);
 true
```

ARRAYS_OVERLAP

函数语法：



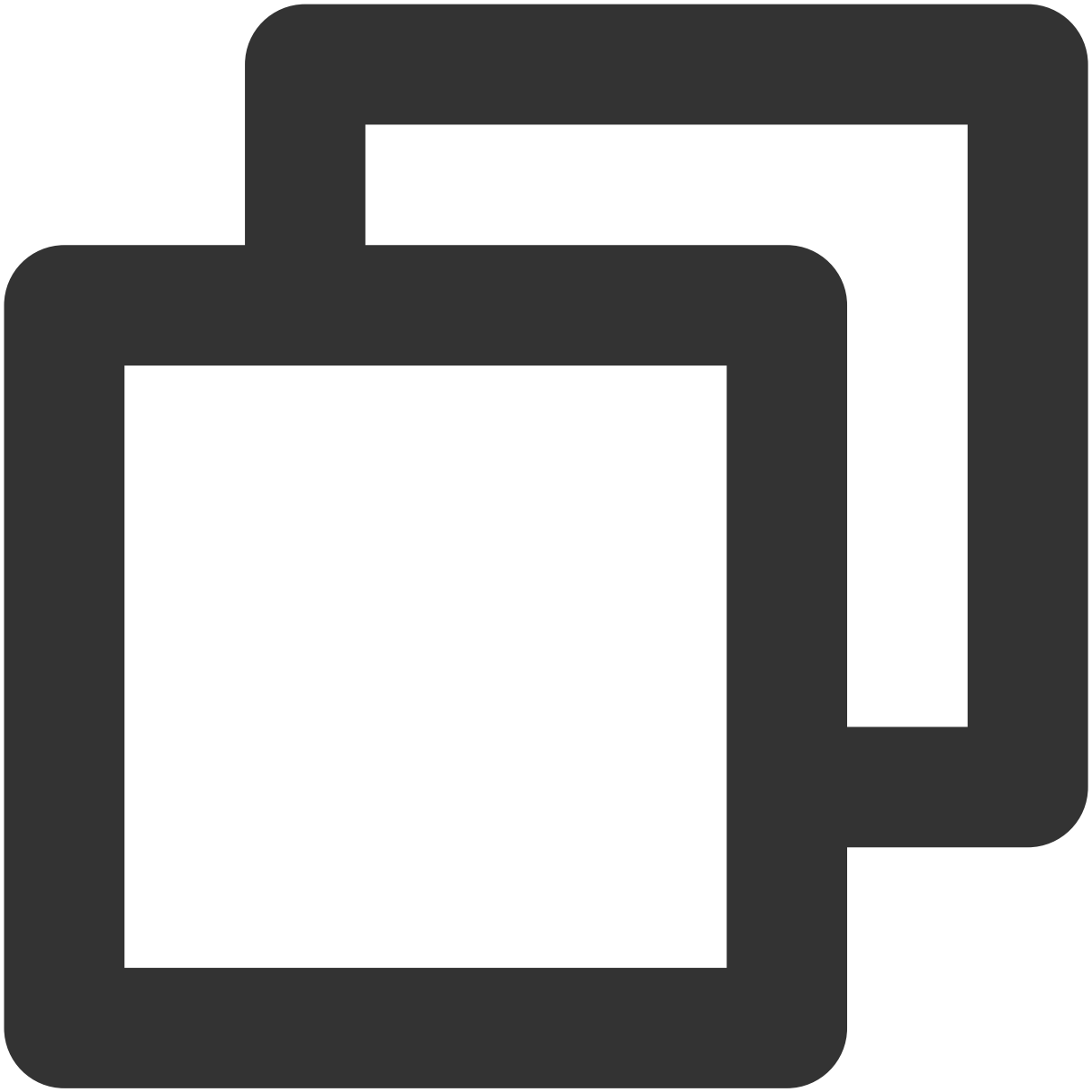
```
ARRAYS_OVERLAP(<a> array<T>, <b> array<U>)
```

支持引擎：SparkSQL、Presto。

使用说明：如果 a 至少包含 b 中也存在的非 null 元素，则返回 true。如果数组没有公共元素，并且它们都是非空的，并且其中任何一个包含 null 元素，则返回 null，否则返回 false。

返回类型：boolean。

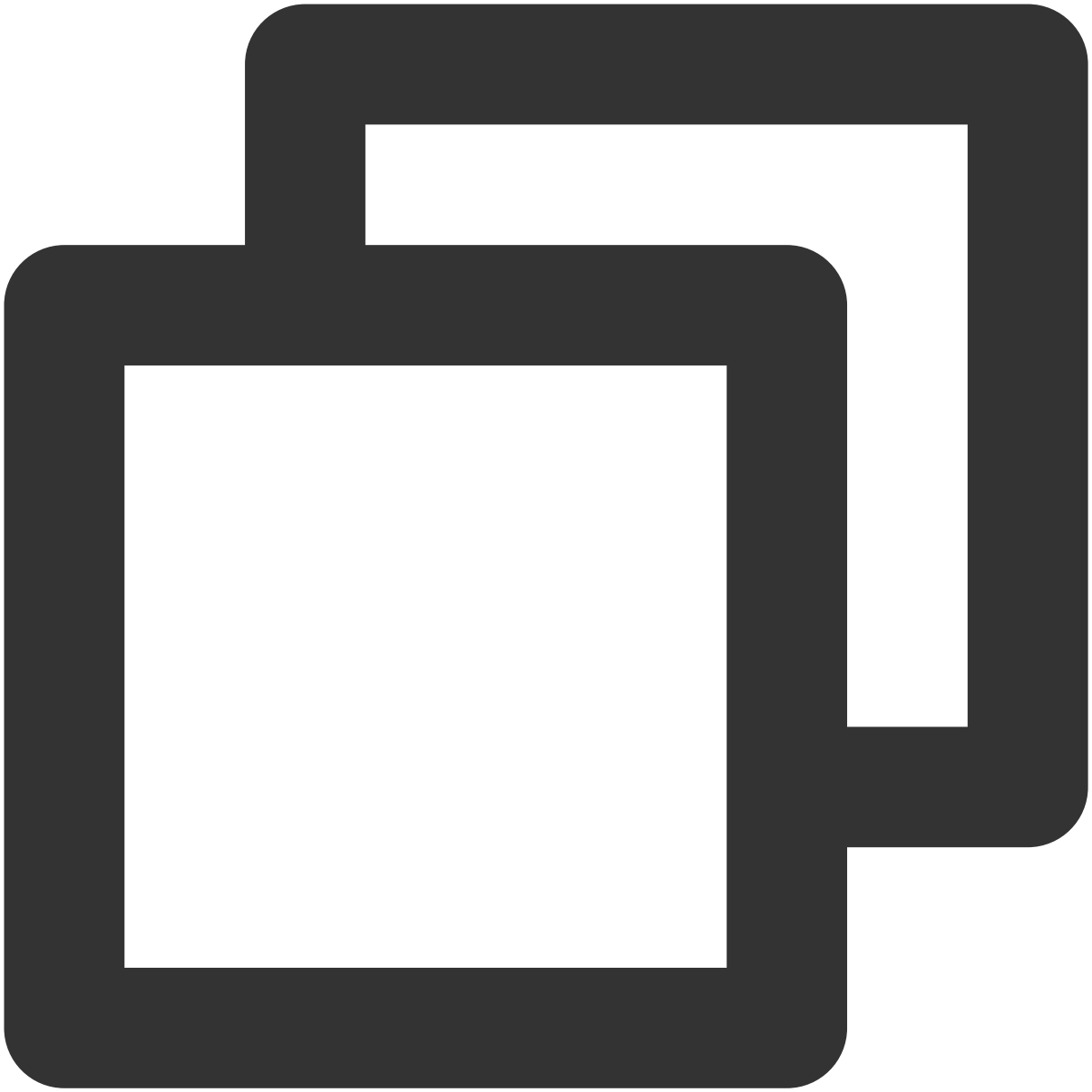
示例：



```
> SELECT arrays_overlap(array(1, 2, 3), array(3, 4, 5));
 true
```

ARRAY_INTERSECT

函数语法：



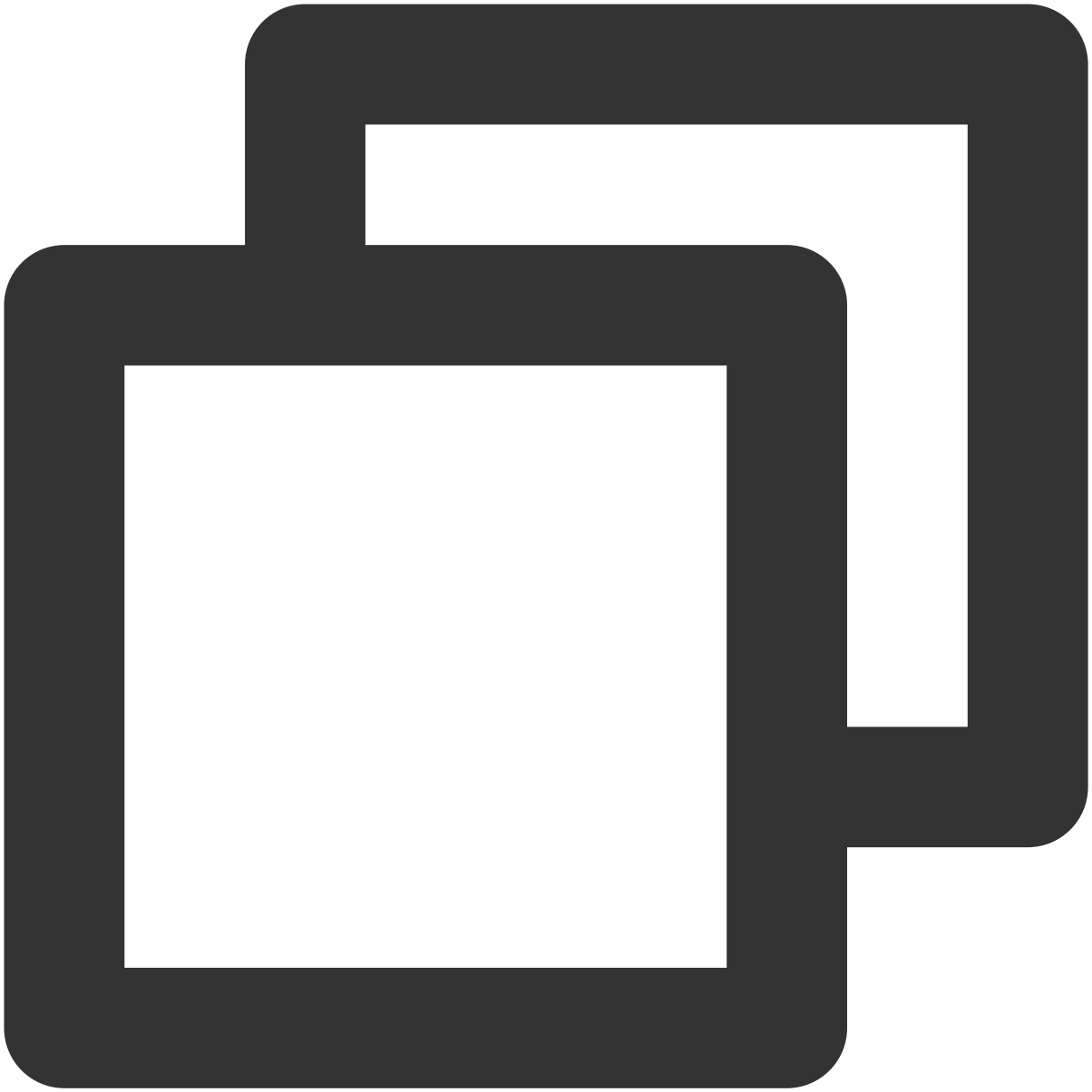
```
ARRAY_INTERSECT(<a> array<T>, <b> array<T>)
```

支持引：SparkSQL、Presto。

使用说：返回 a 和 b 相交处的元素数组，无重复项。

返回类：array<T>。

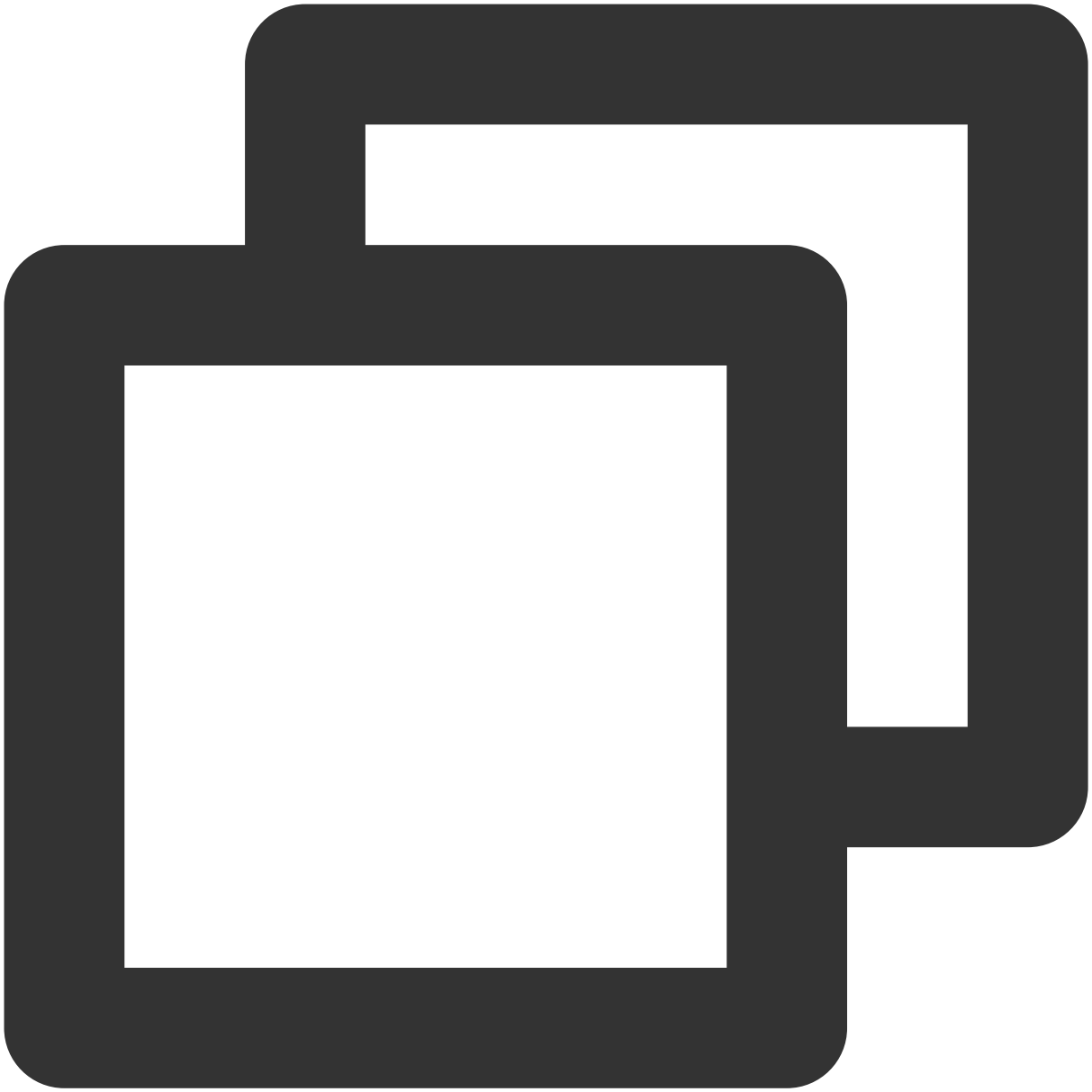
示例：



```
> SELECT array_intersect(array(1, 2, 3), array(1, 3, 5));
 [1,3]
```

ARRAY_JOIN

函数语法：



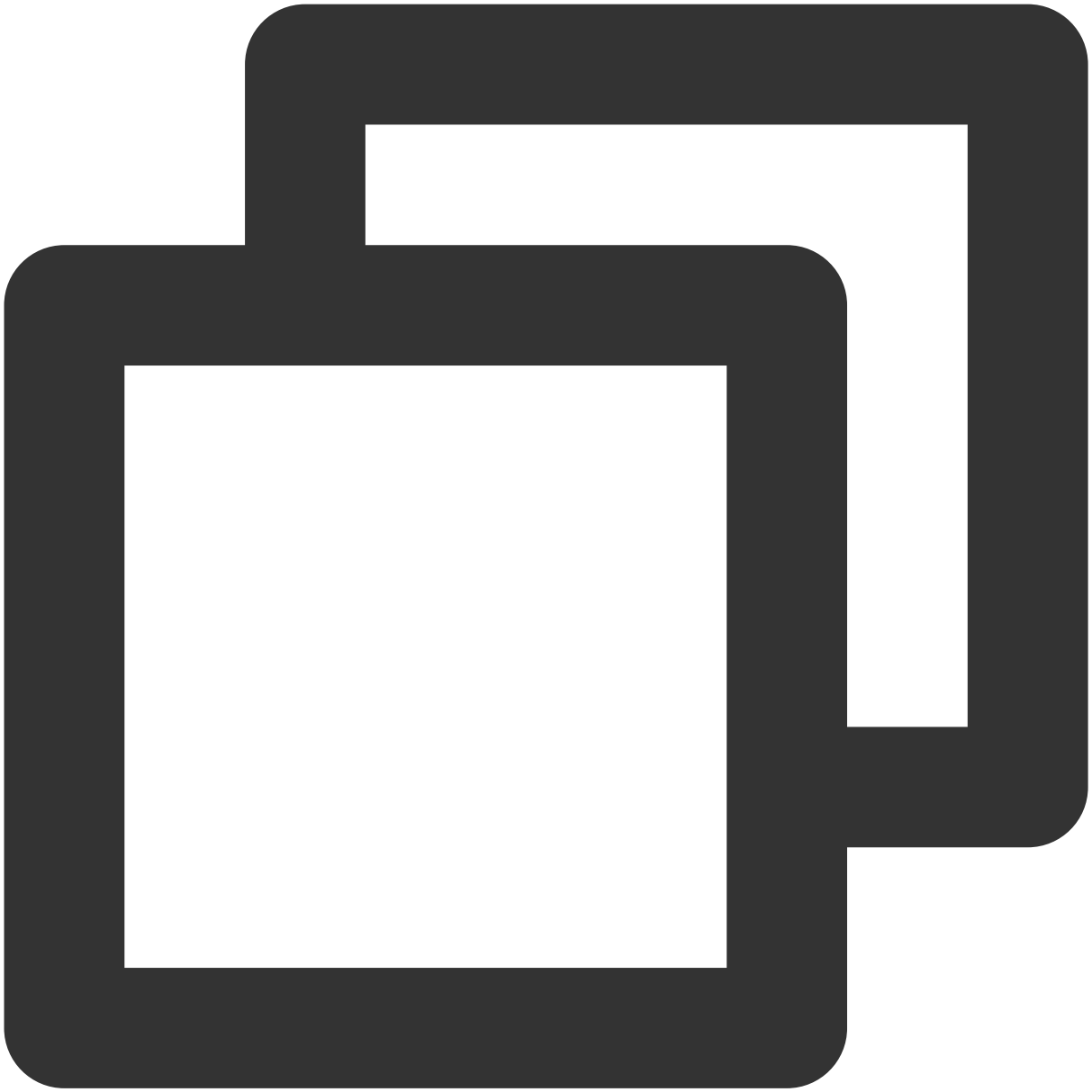
```
ARRAY_JOIN(<a> ARRAY<T>, <delimiter> string[, <nullReplacement> string])
```

支持引擎：SparkSQL、Presto。

使用说明：使用分隔符和可选字符串连接给定数组的元素以替换 null。如果没有为 nullReplacement 设置值，则会过滤所有 null 值。

返回类型：string。

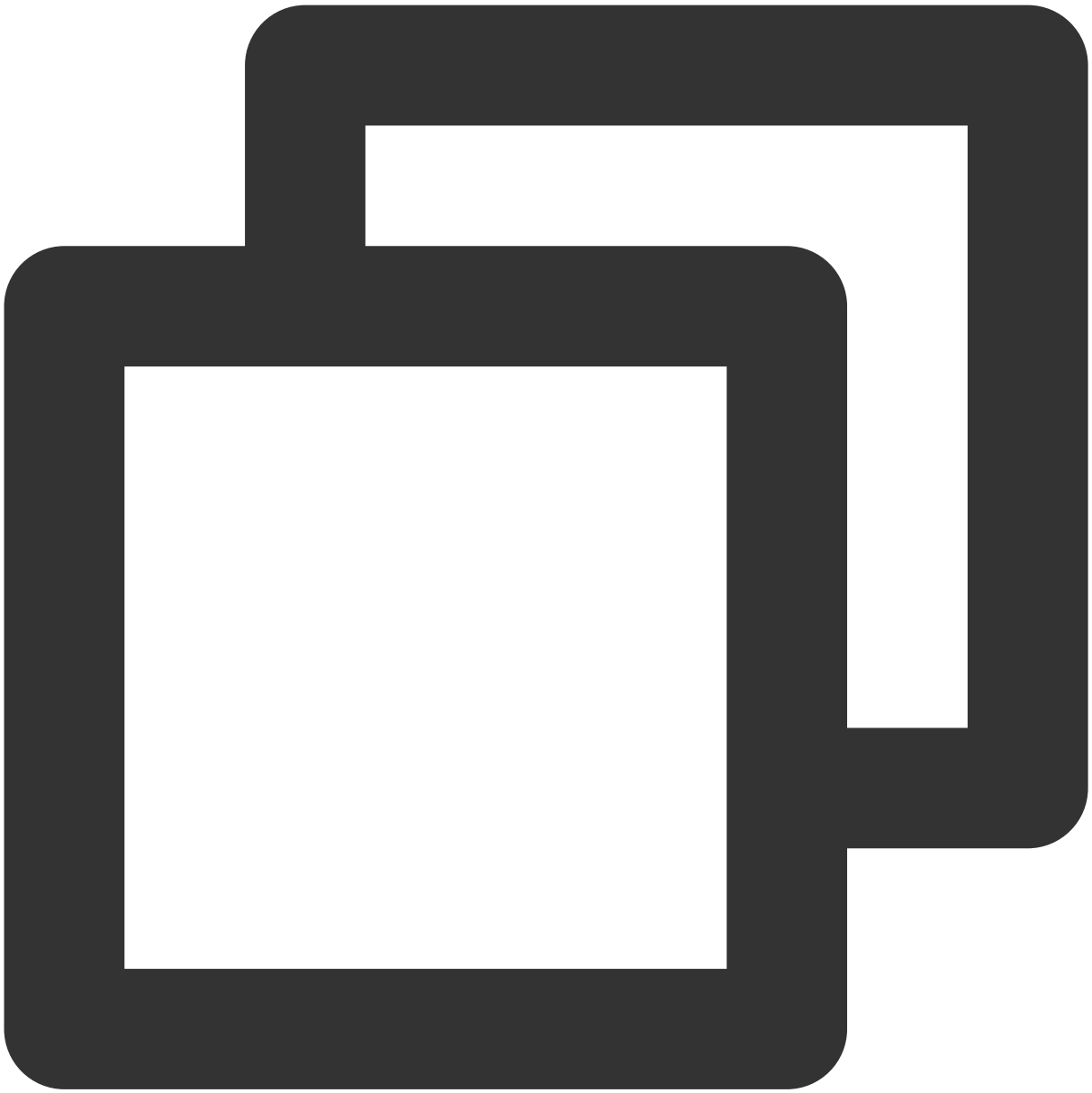
示例：



```
> SELECT array_join(array('hello', 'world'), ' ');
 hello world
> SELECT array_join(array('hello', null ,'world'), ' ');
 hello world
> SELECT array_join(array('hello', null ,'world'), ' ', ',');
 hello , world
```

ARRAY_POSITION

函数语法：



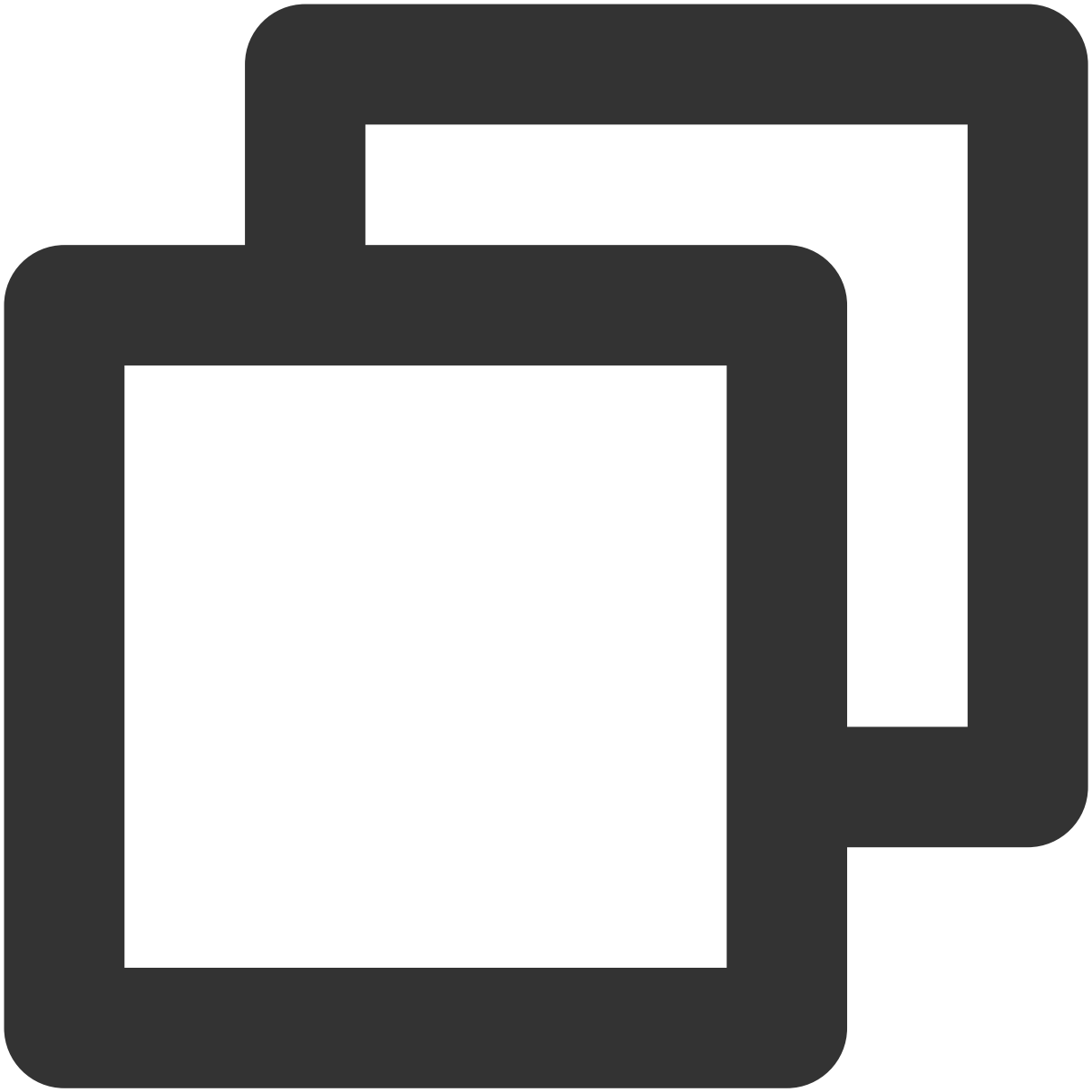
```
ARRAY_POSITION(<a> array<T>, <element> T)
```

支持引擎：SparkSQL、Presto。

使用说明：返回数组第一个元素的（从1开始计数）索引。

返回类型：integer。

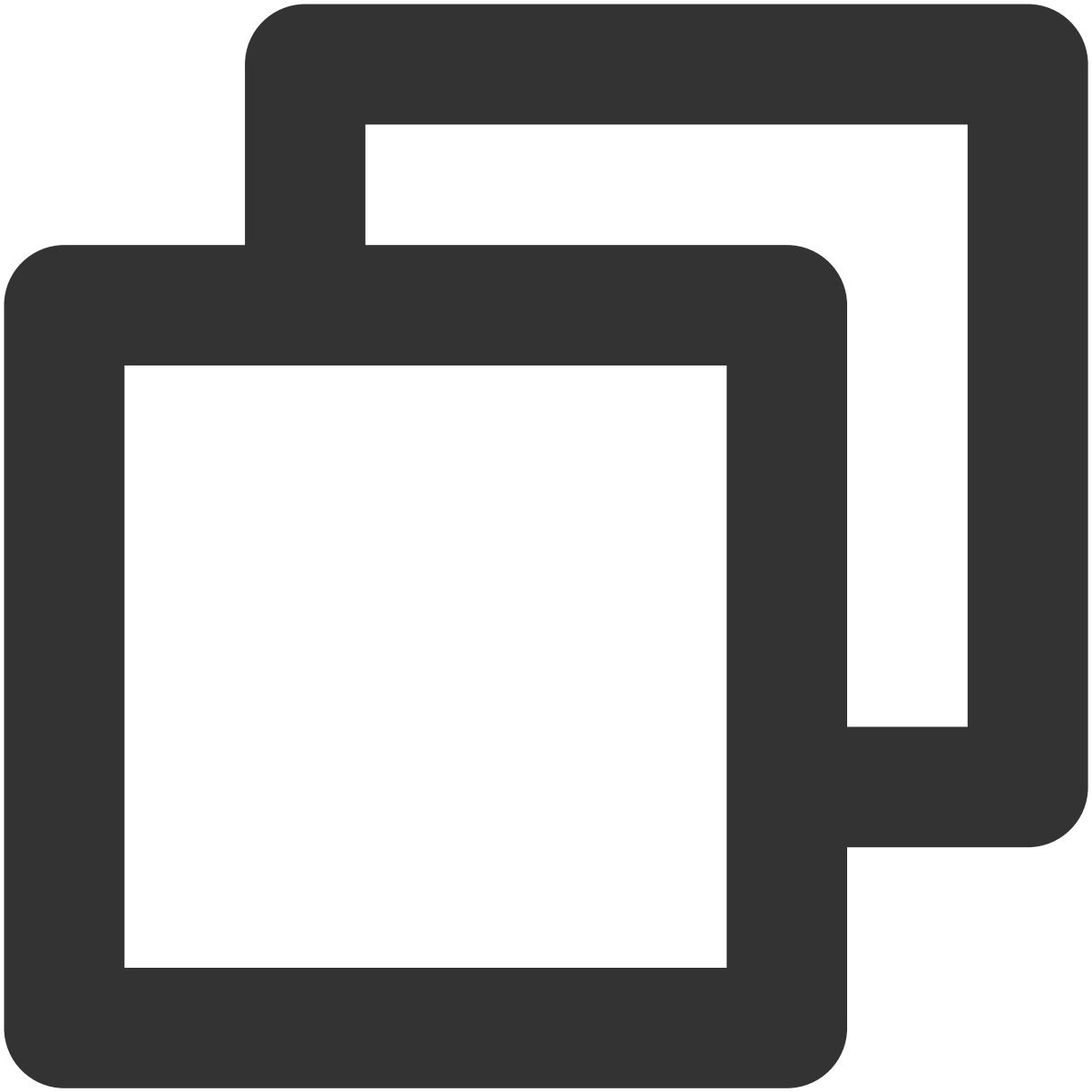
示例：



```
> SELECT array_position(array(3, 2, 1), 1);
 3
```

ARRAY_SORT

函数语法：



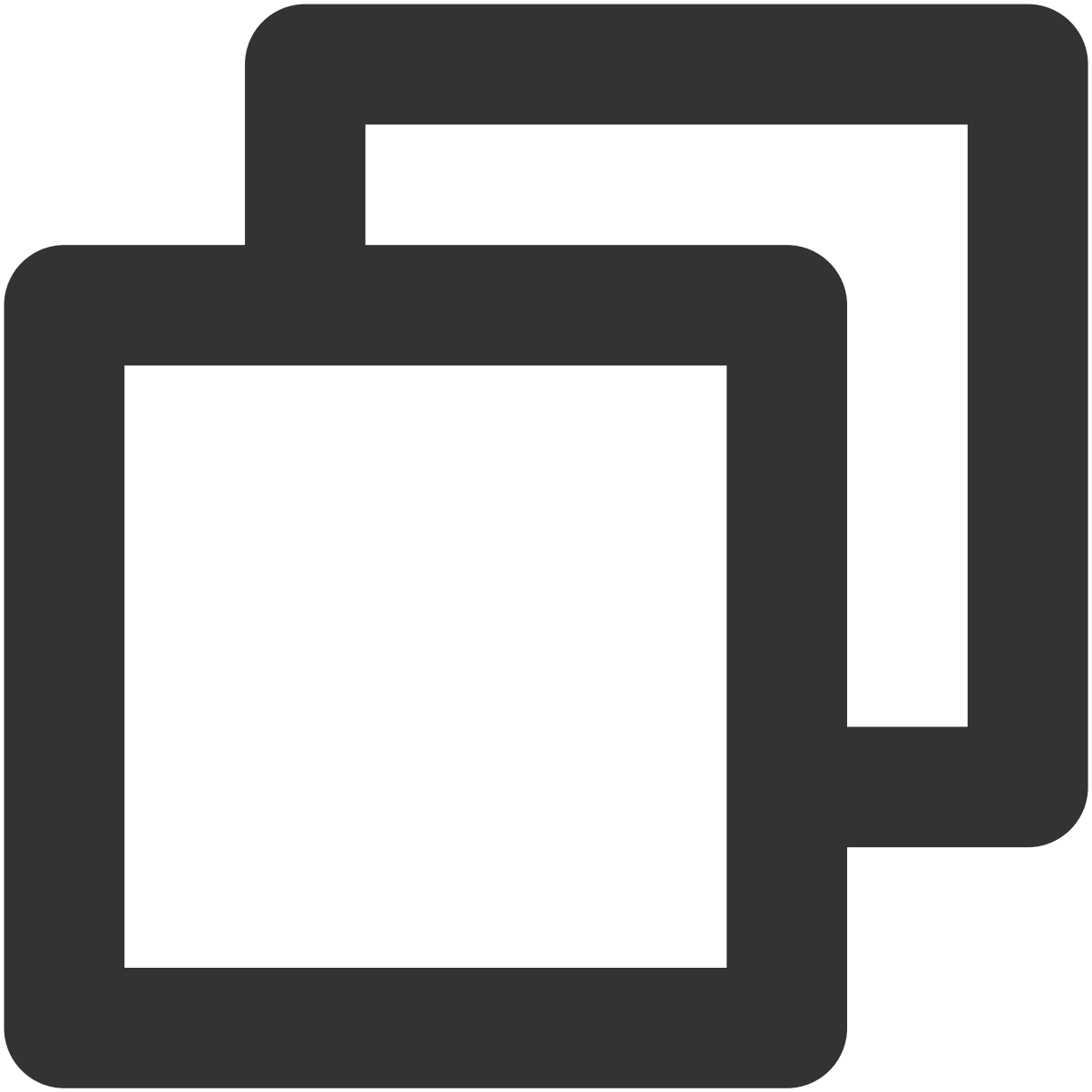
```
ARRAY_SORT(<a> array<T>[, <func> function(T, T)->integer])
```

支持引擎：SparkSQL、Presto。

使用说明：对输入数组排序。如果省略 func，则按升序排序。

返回类型：array<T>。

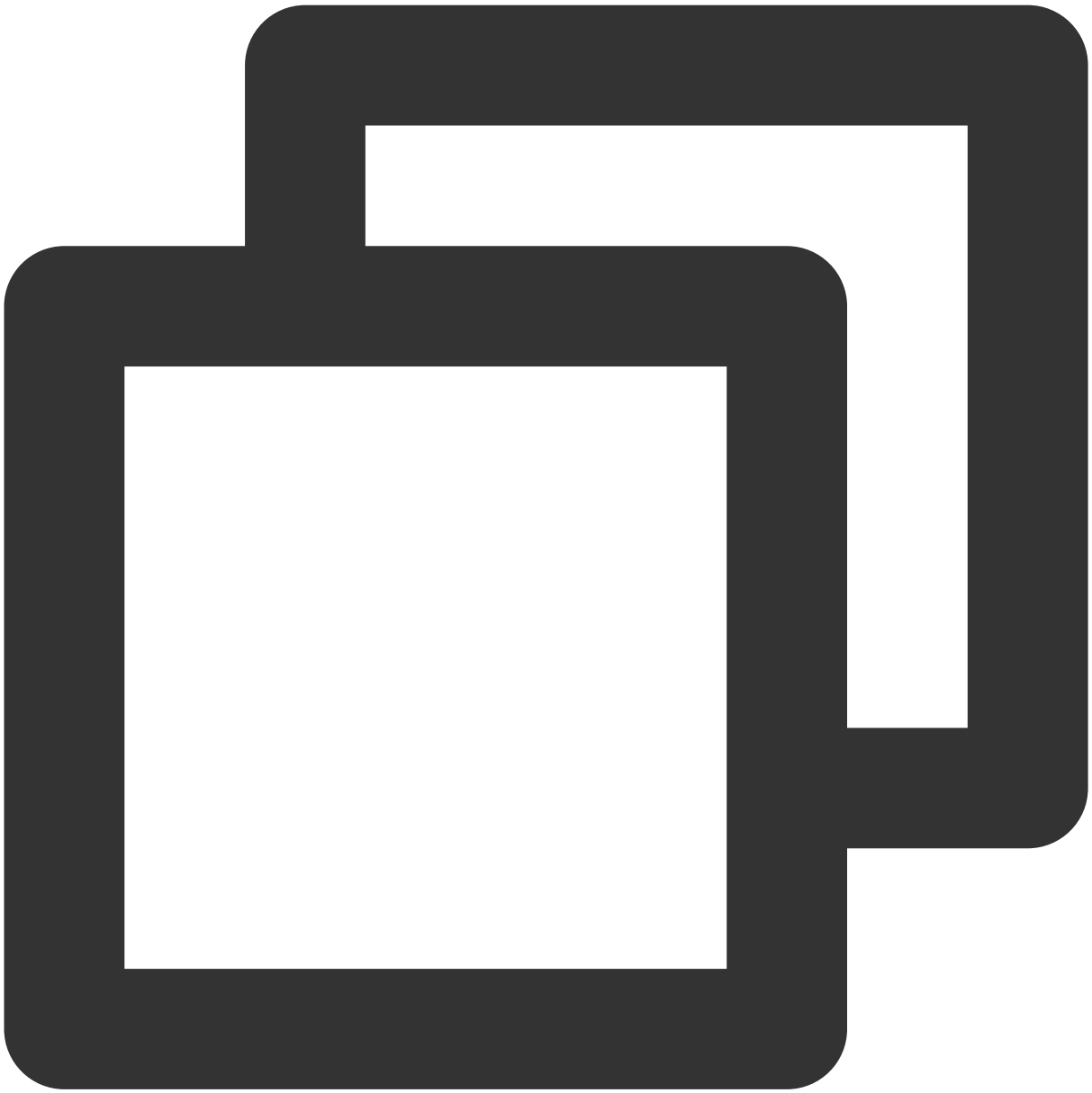
示例：



```
> SELECT array_sort(array(5, 6, 1), (left, right) -> case when left < right then -1
 [1,5,6]
> SELECT array_sort(array('bc', 'ab', 'dc'), (left, right) -> case when left is nul
 ["dc","bc","ab"]
> SELECT array_sort(array('b', 'd', null, 'c', 'a'));
 ["a","b","c","d",null]
```

ARRAY_EXCEPT

函数语法：



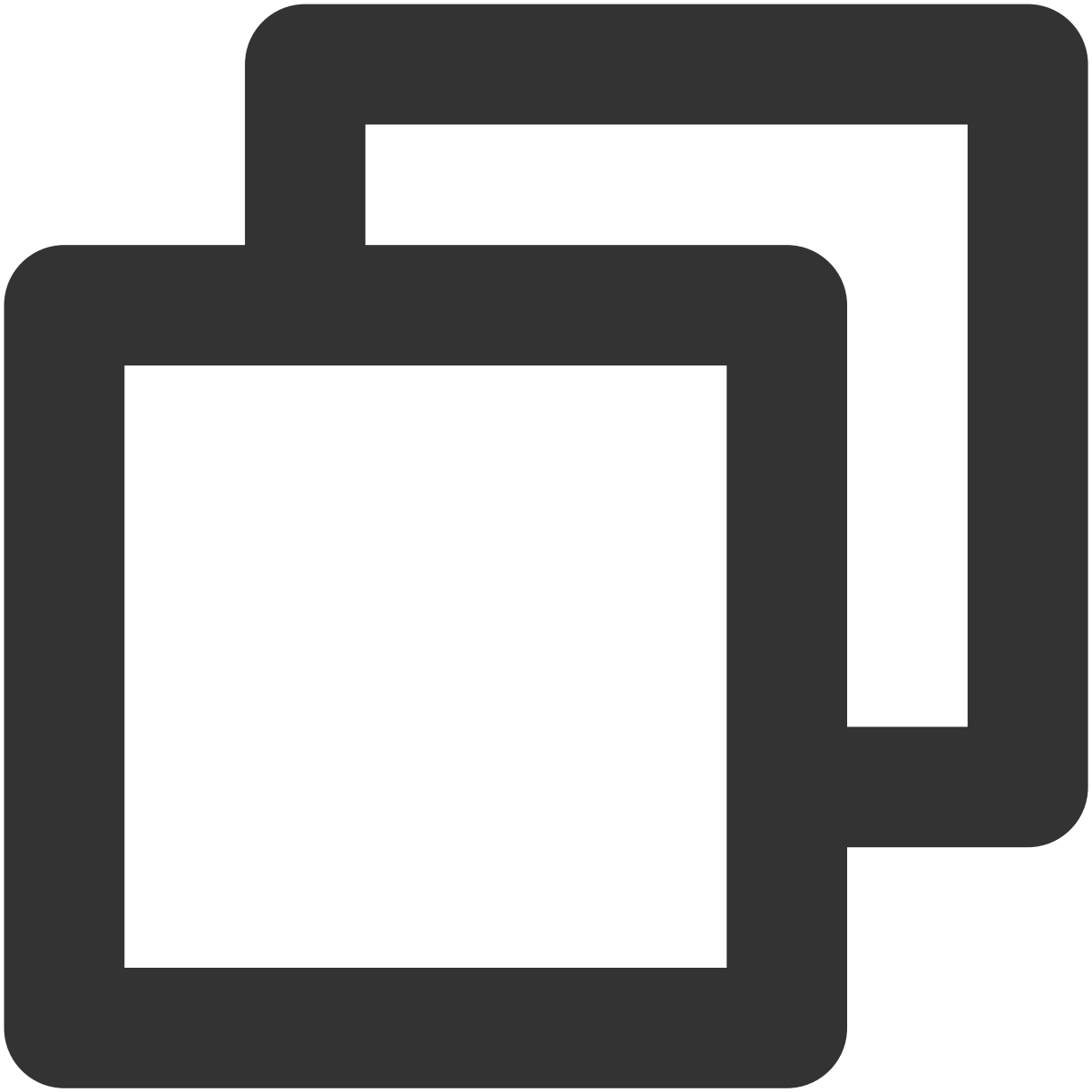
```
ARRAY_EXCEPT(<a> array<T>, <b> array<T>)
```

支持引擎：SparkSQL、Presto。

使用说明：返回 a 中但不在 b 中的元素数组，不包含重复项。

返回类型：array<T>。

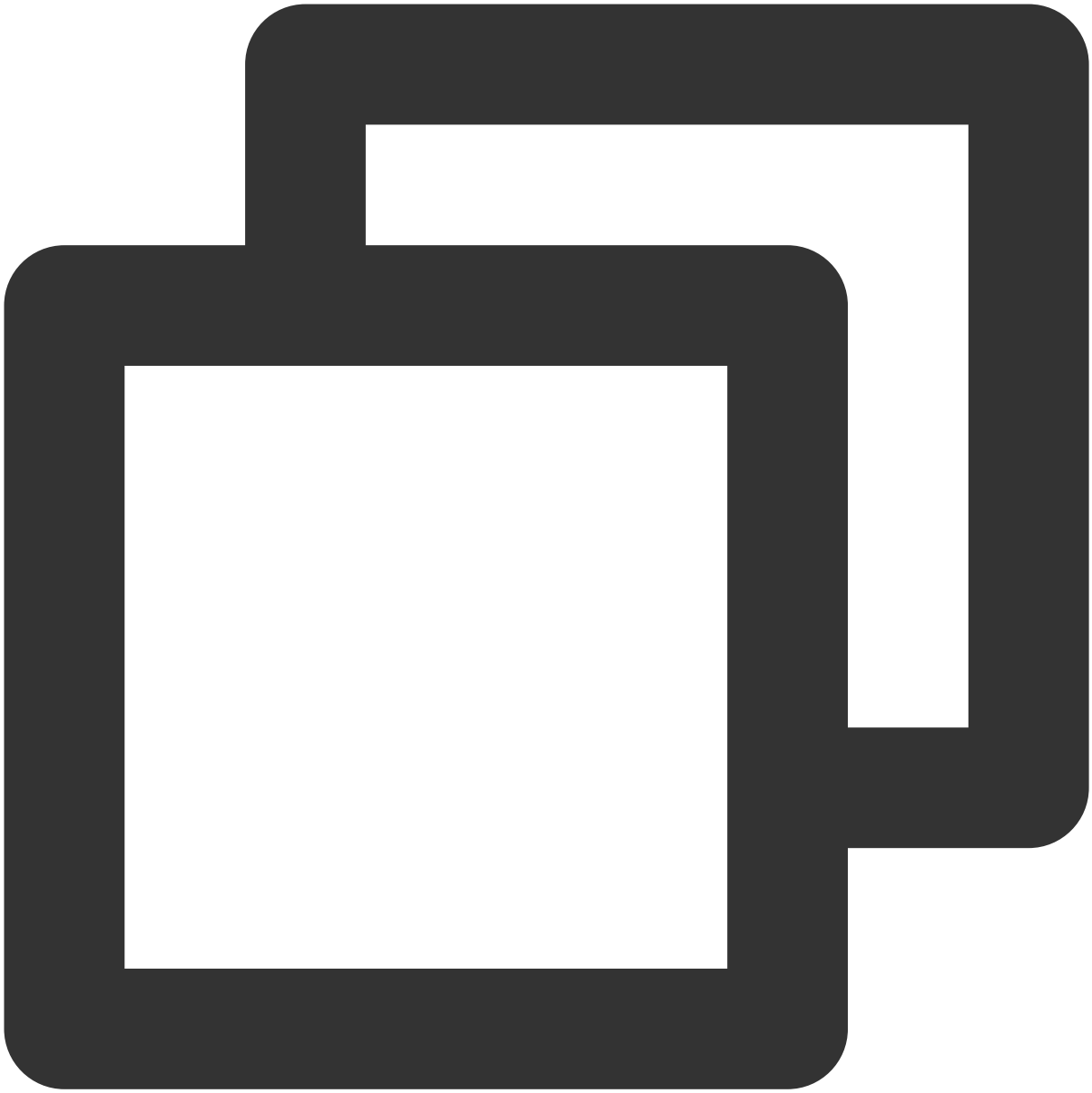
示例：



```
> SELECT array_except(array(1, 2, 3), array(1, 3, 5));
 [2]
```

ARRAY_UNION

函数语法：



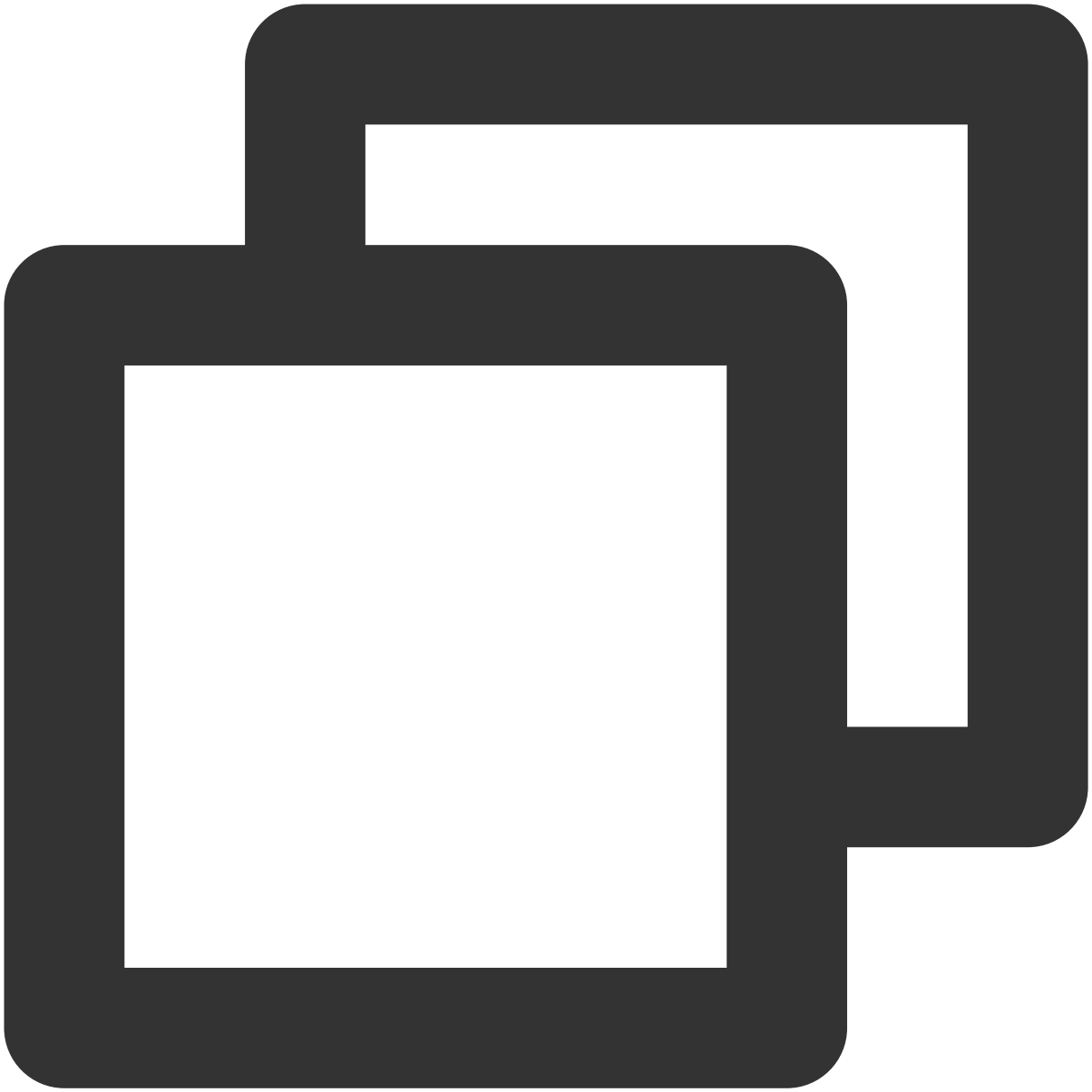
```
ARRAY_UNION(<a> array<T>, <b> array<T>)
```

支持引擎：SparkSQL、Presto。

使用说明：返回 a 和 b 并集中的元素数组，不包含重复项。

返回类型：array<T>。

示例：



```
> SELECT array_union(array(1, 2, 3), array(1, 3, 5));
 [1,2,3,5]
```

NAMED_STRUCT

函数语法：

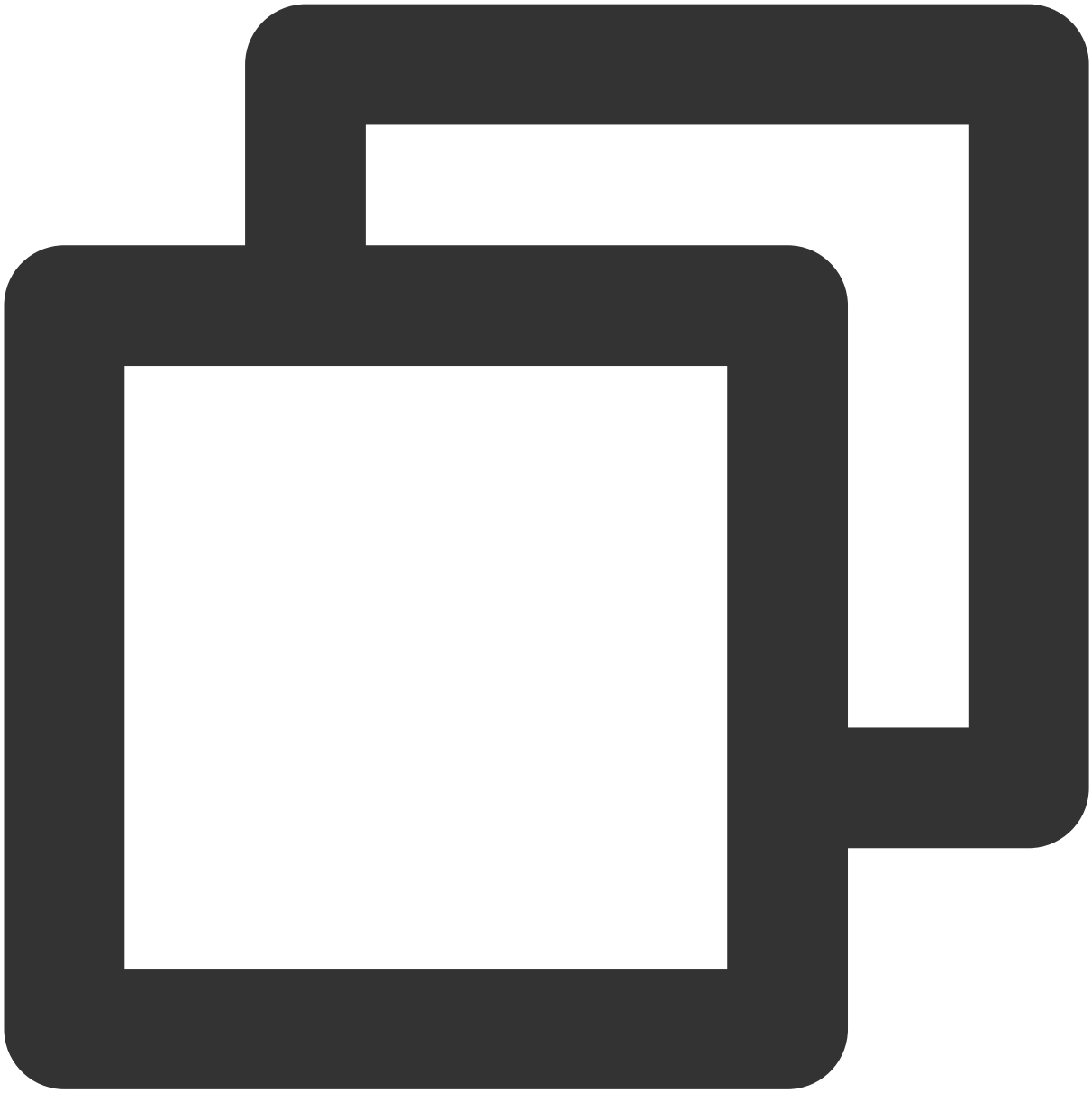
```
NAMED_STRUCT(name1 K, val1 V, ...)
```

支持引擎：SparkSQL。

使用说明：使用给定的字段名和值创建 struct。

返回类型：struct。

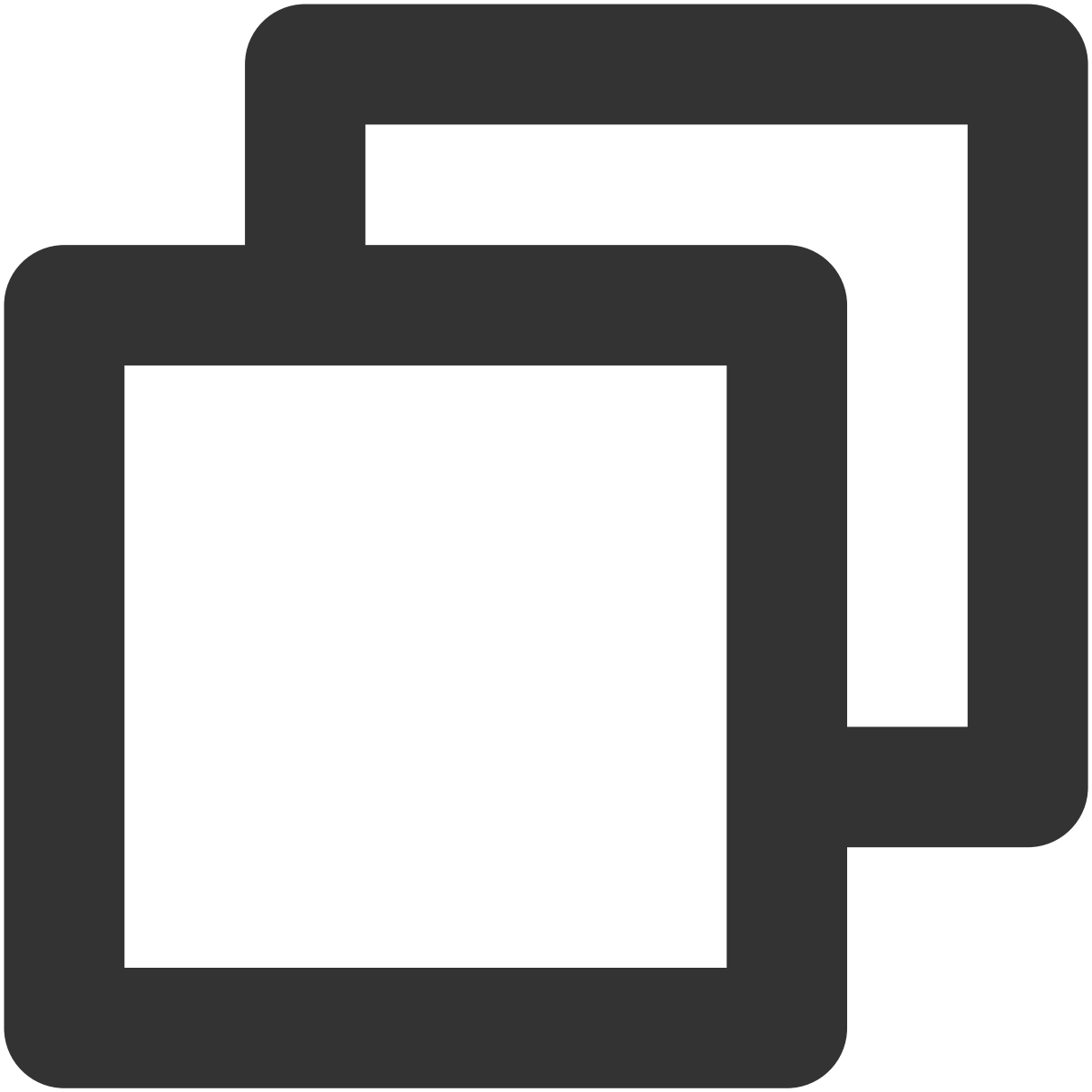
示例：



```
> SELECT named_struct("a", 1, "b", 2, "c", 3);
 {"a":1,"b":2,"c":3}
```

STRUCT

函数语法：



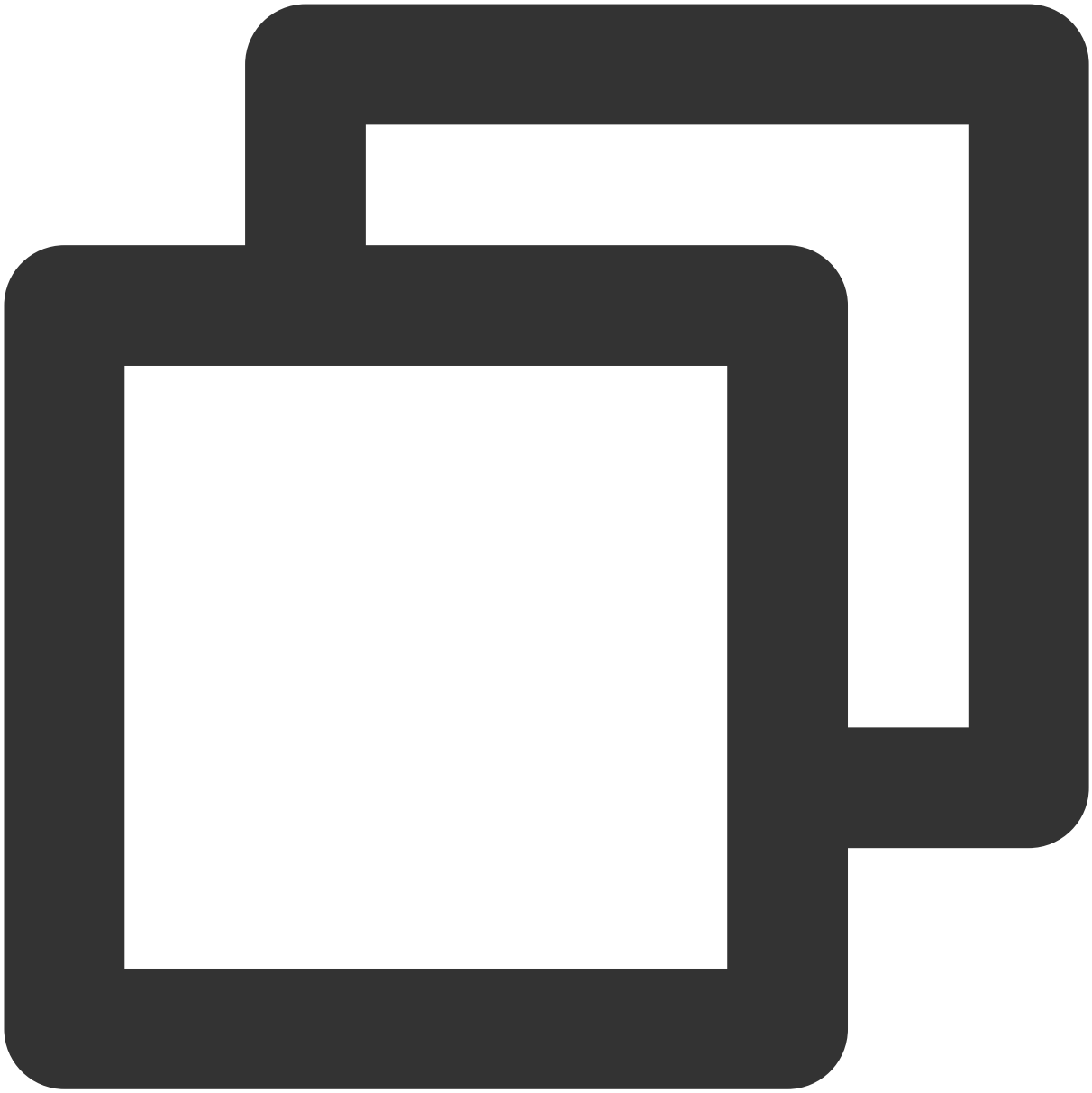
```
STRUCT(<col1> T1, <col2> T2, ...)
```

支持引擎：SparkSQL、Presto。

使用说明：使用给定的字段名和值创建 struct。

返回类型：struct。

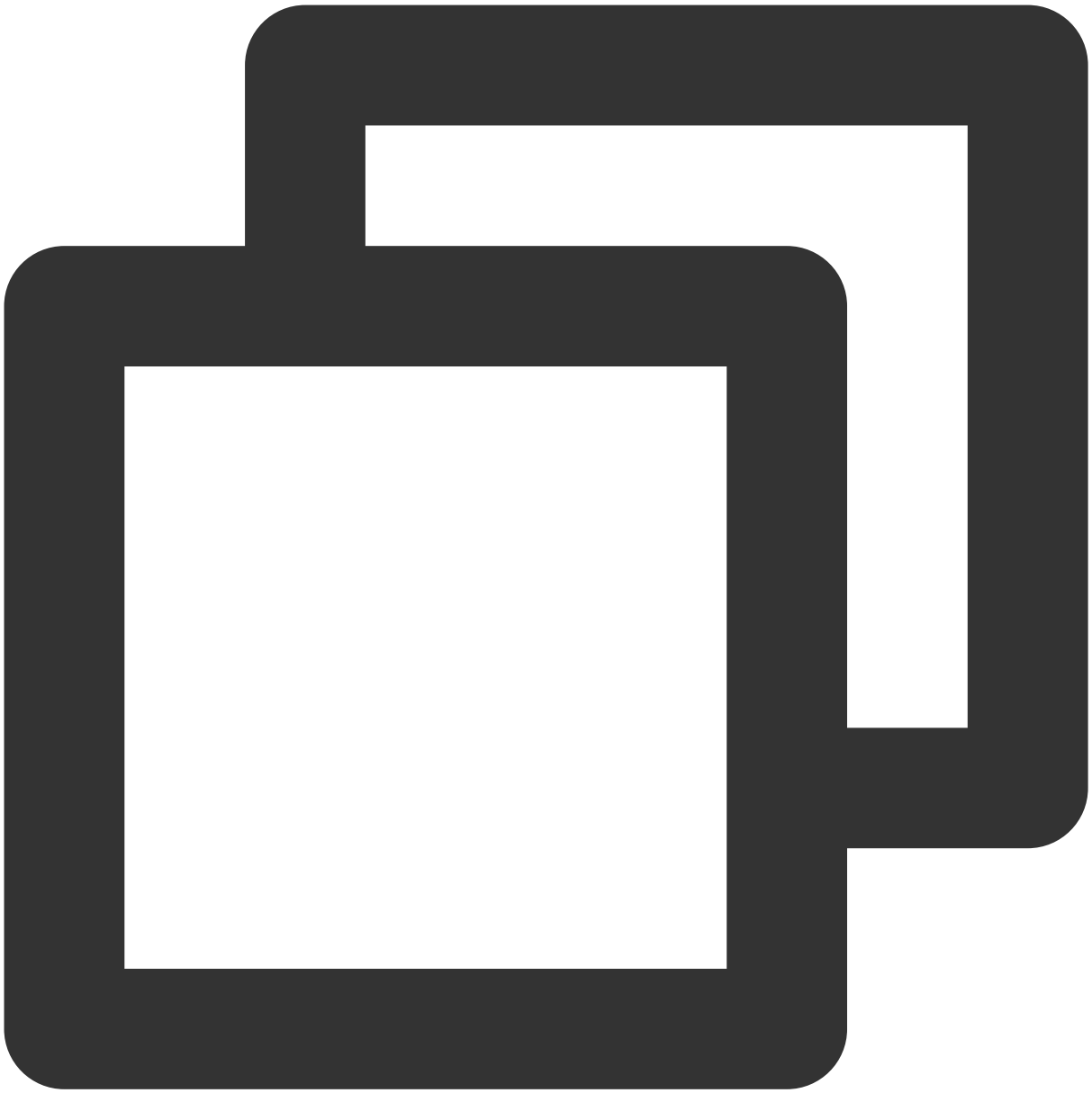
示例：



```
> SELECT struct('a', 'b', 'c');
{"col1":"a","col2":"b","col3":"c"}

> SELECT struct('a', 'b', 'c', 1, 2);
{"col1":"a","col2":"b","col3":"c","col4":1,"col5":2}
```

SLICE

函数语法：



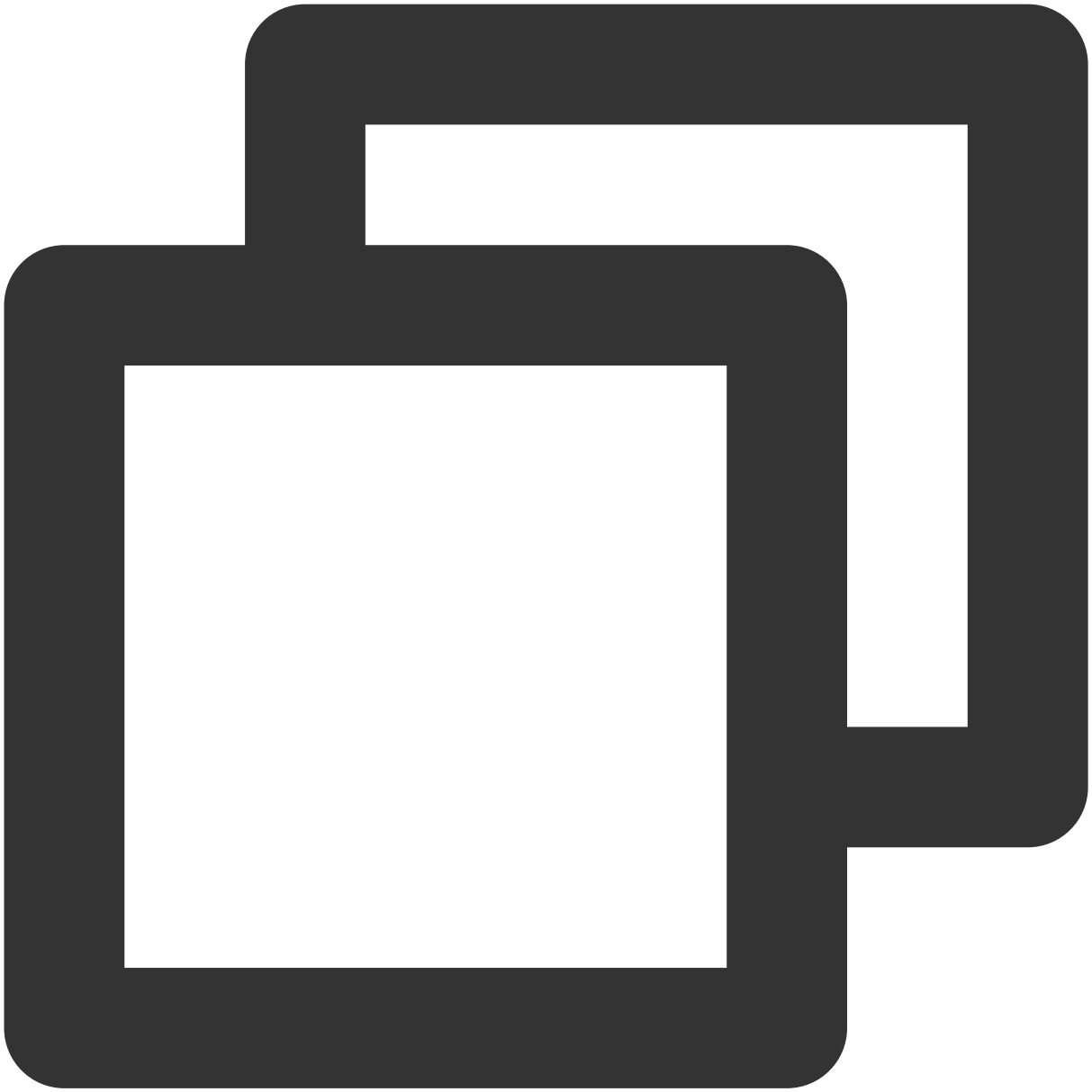
```
SLICE(<a> array<T>, <start> integer, <length> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：返回数组 a 从索引 start 开始（数组索引从1开始，如果开始为负，则从结尾开始），长度为的具有指定长度 length 的子集。

返回类型：array<T>。

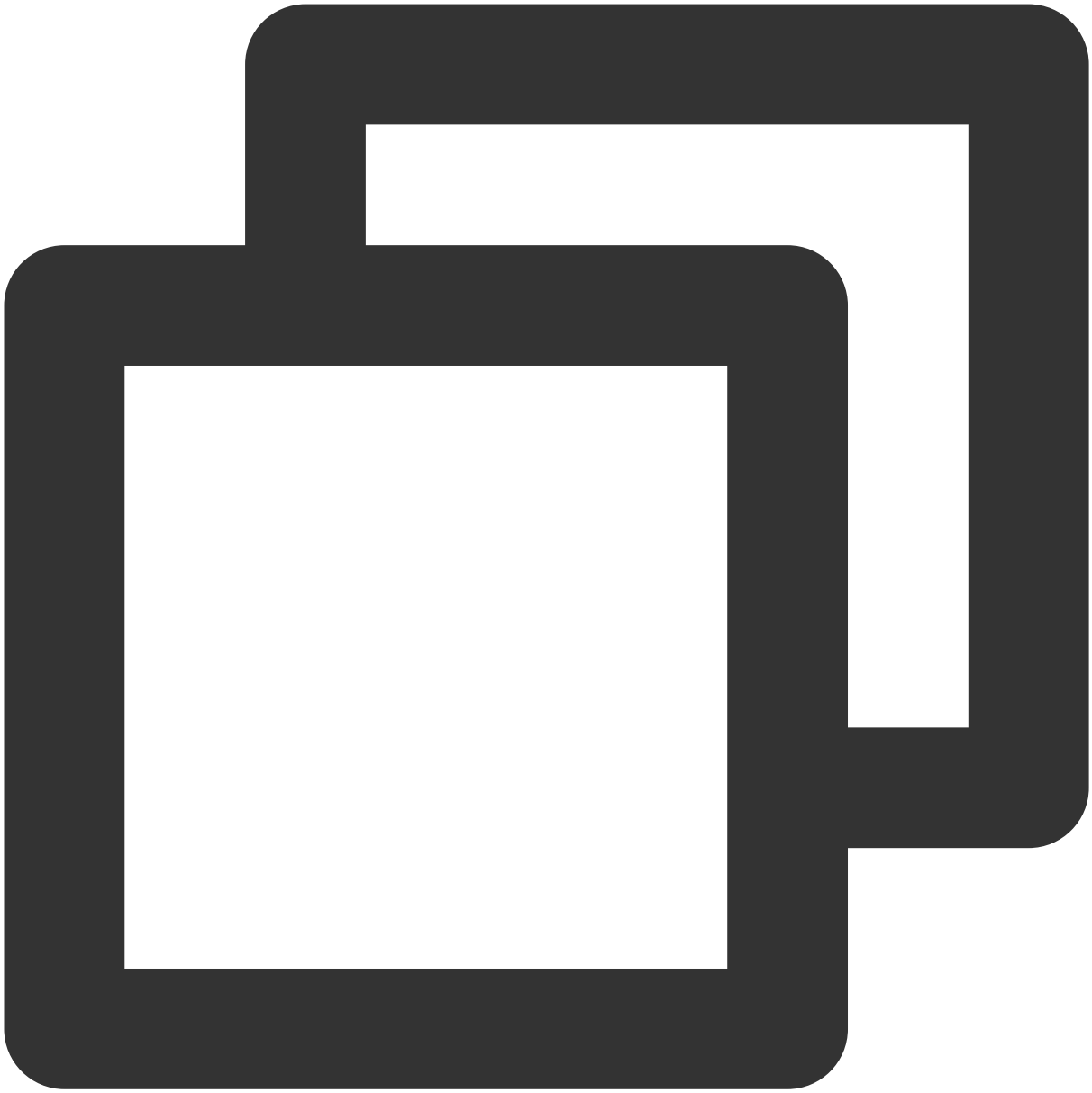
示例：



```
> SELECT slice(array(1, 2, 3, 4), 2, 2);
 [2,3]
> SELECT slice(array(1, 2, 3, 4), -2, 2);
 [3,4]
```

ARRAYS_ZIP

函数语法：



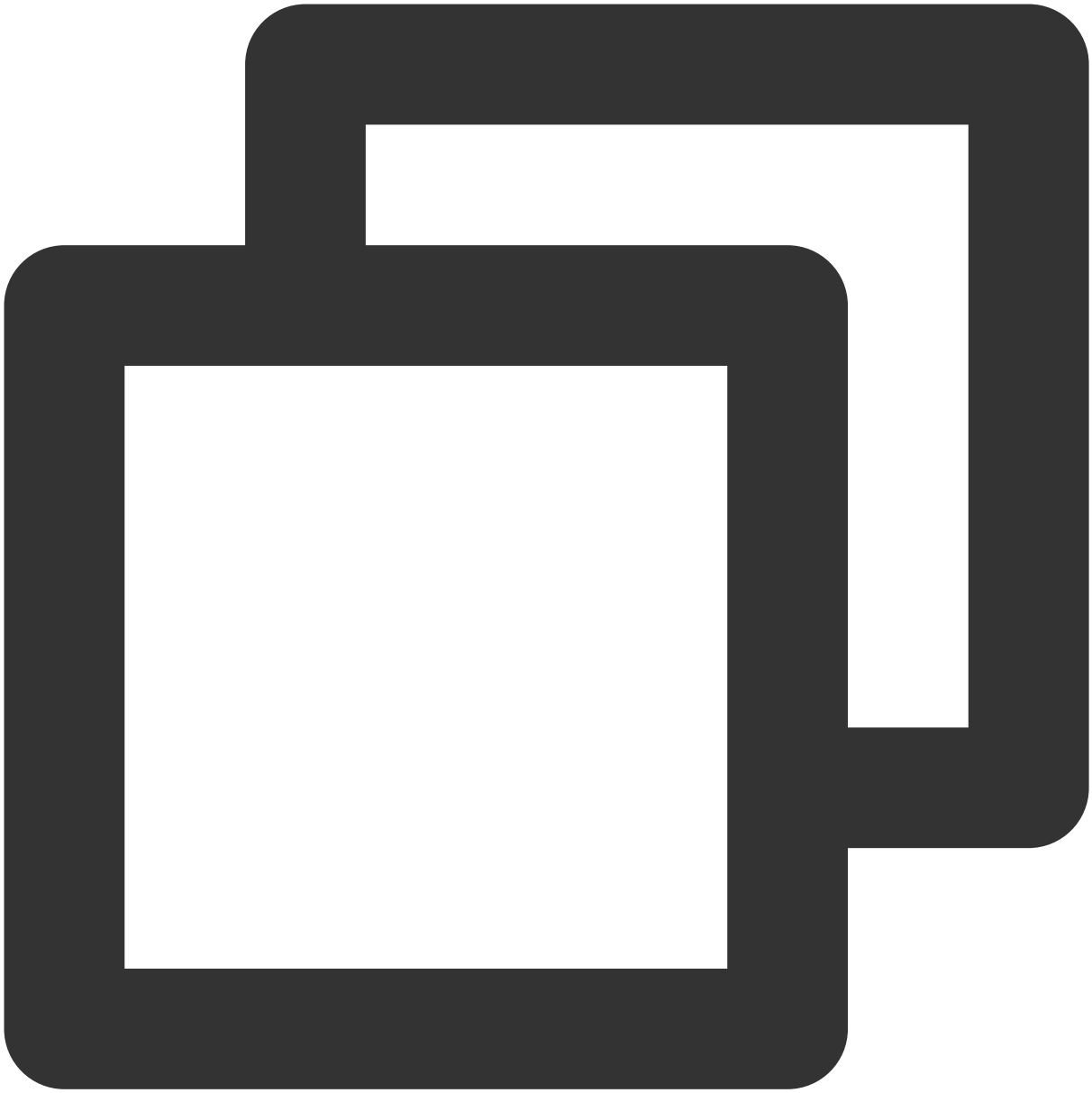
```
ARRAYS_ZIP(<a1> array<T>, ...)
```

支持引擎：SparkSQL。

使用说明：返回合并后的数组，元素为 struct 类型，其中第 N 个 struct 包含每个输入数组的第 N 个值。

返回类型：array<struct<string, T>>。

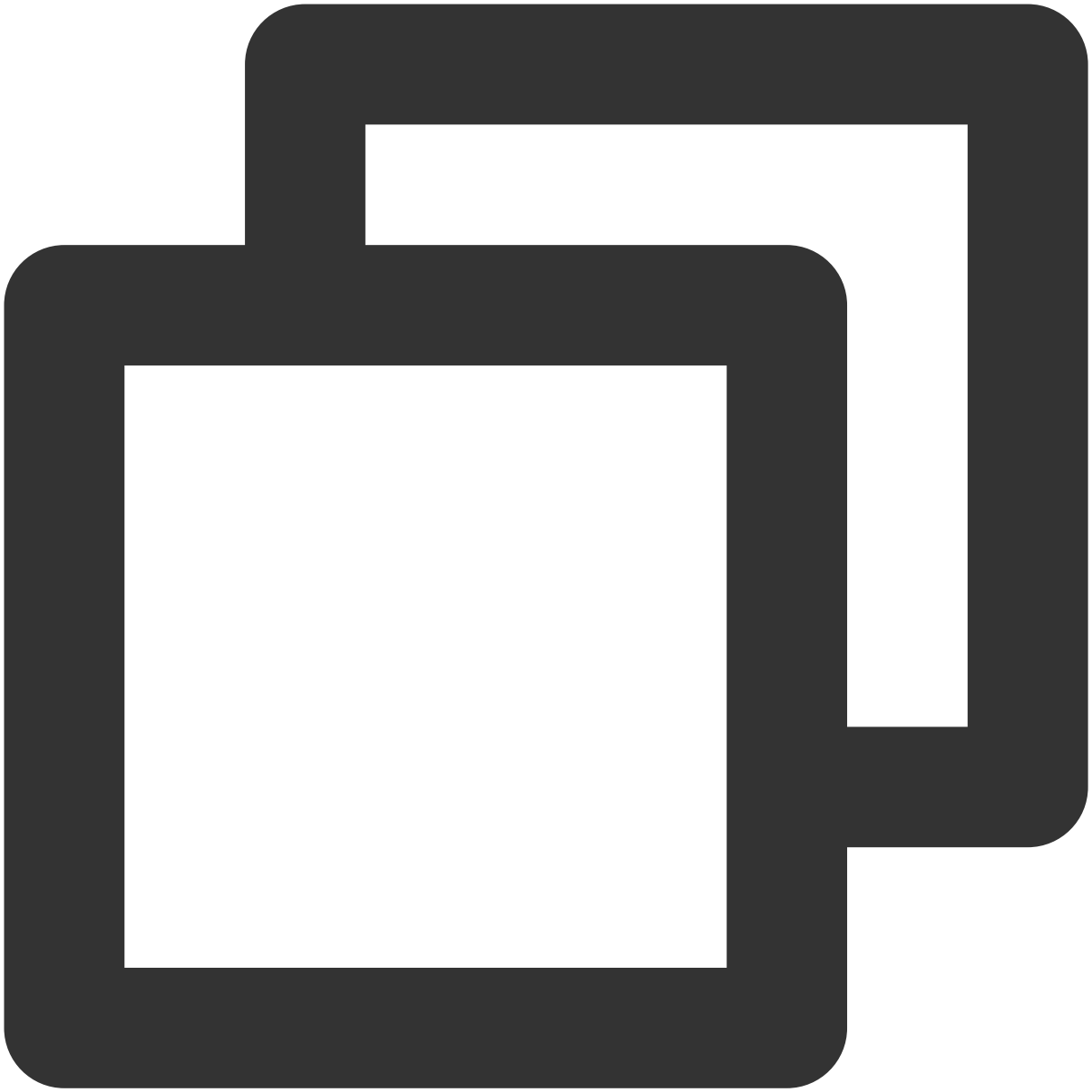
示例：



```
> SELECT arrays_zip(array(1, 2, 3), array(2, 3, 4));
 [{"0":1,"1":2},{"0":2,"1":3},{"0":3,"1":4}]
> SELECT arrays_zip(array(1, 2), array(2, 3), array(3, 4));
 [{"0":1,"1":2,"2":3},{"0":2,"1":3,"2":4}]
```

SORT_ARRAY

函数语法：



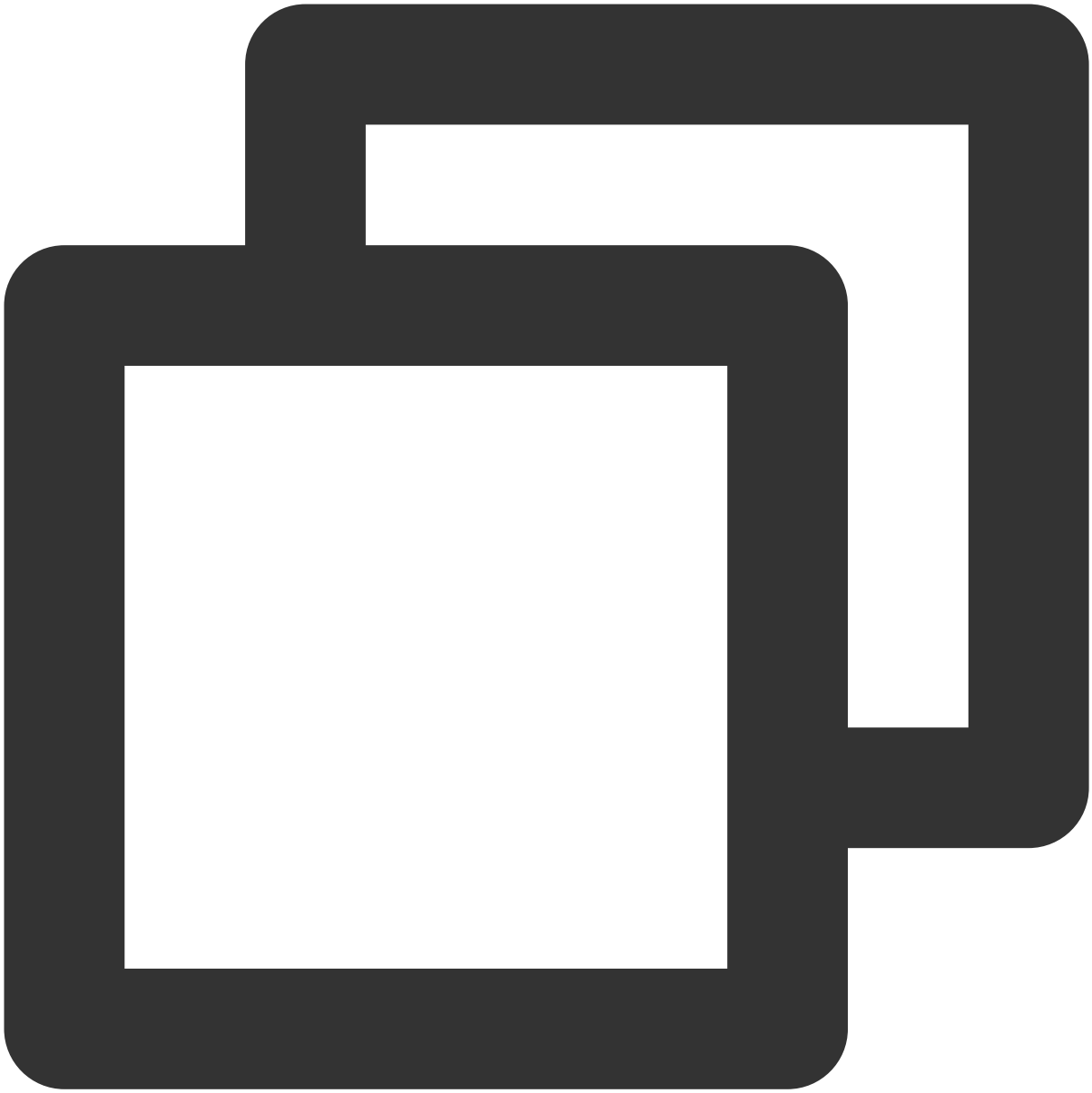
```
SORT_ARRAY(<a> array<T>[, ascendingOrder boolean])
```

支持引擎：SparkSQL、Presto。

使用说明：按升序或降序对输入数组排序。对于 double/float 类型，NaN 大于任何非 NaN 元素。Null 元素将按升序放置在返回数组的开头，或按降序放置在返回数组的末尾。

返回类型：array<T>。

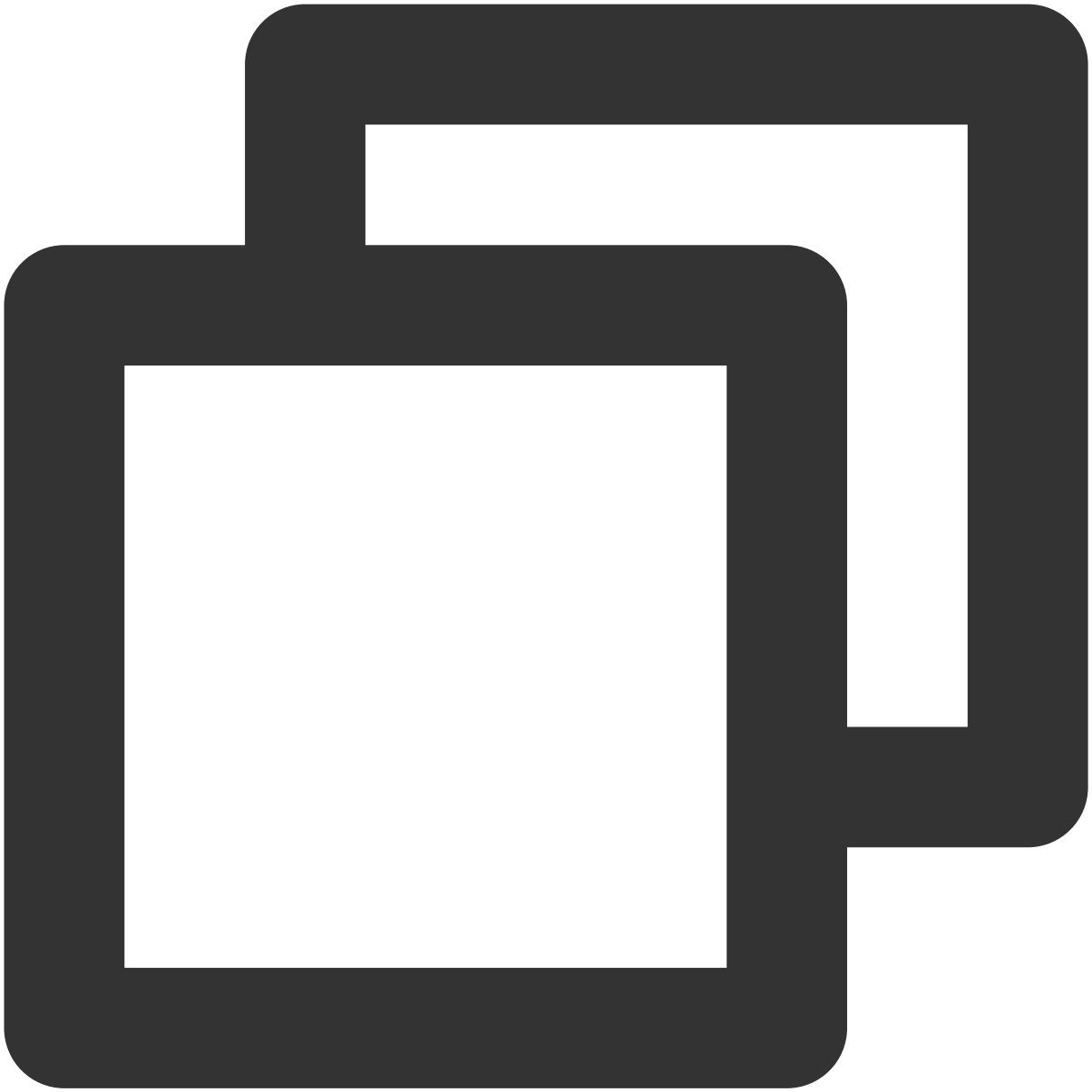
示例：



```
> SELECT sort_array(array('b', 'd', null, 'c', 'a'), true);
 [null,"a","b","c","d"]
```

SHUFFLE

函数语法：



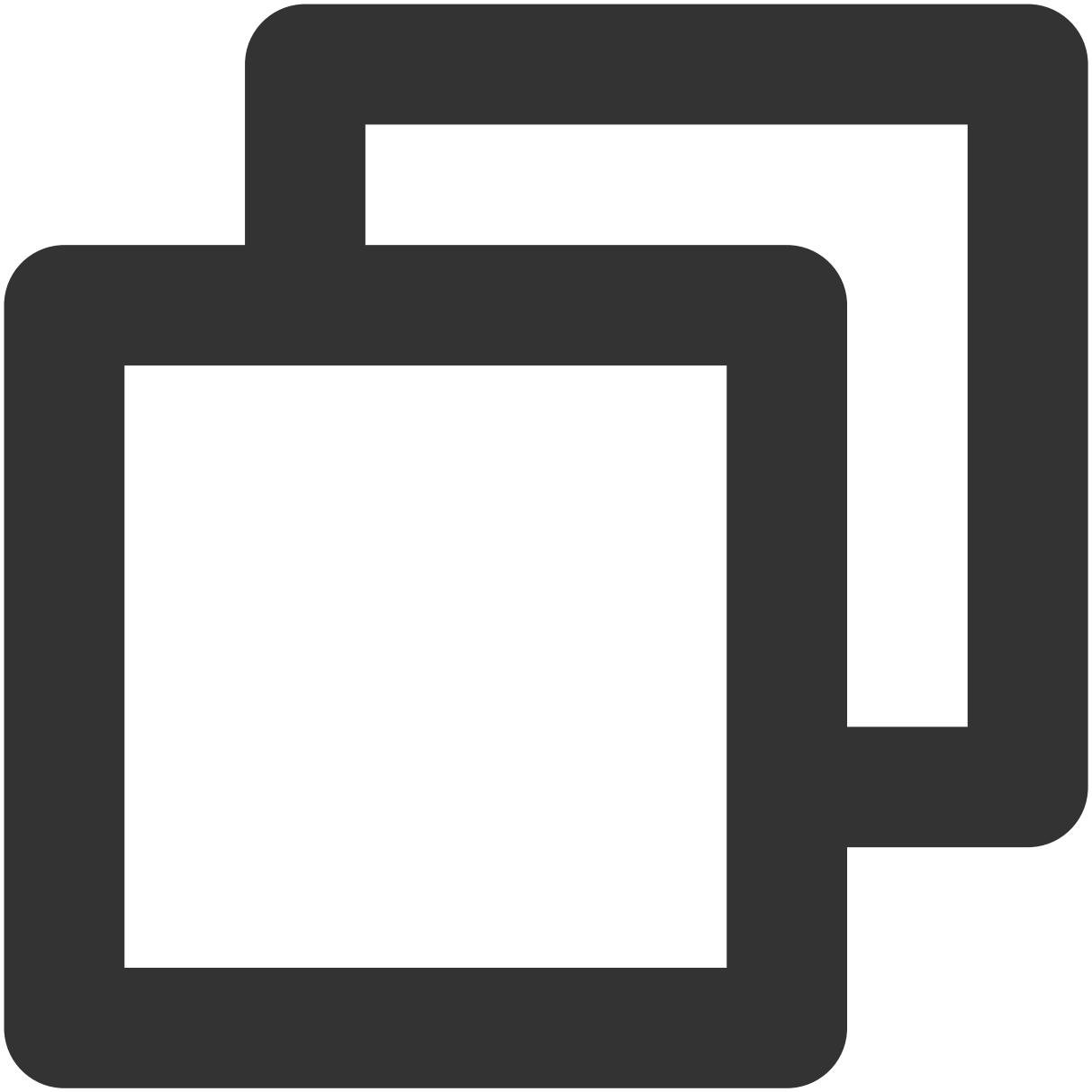
```
SHUFFLE(<a> array<T>)
```

支持引擎：SparkSQL、Presto。

使用说明：返回给定数组的随机排列。

返回类型：array<T>。

示例：



```
> SELECT shuffle(array(1, 20, 3, 5));
 [3,1,5,20]
> SELECT shuffle(array(1, 20, null, 3));
 [20,null,3,1]
```

ARRAY_MAX

函数语法：

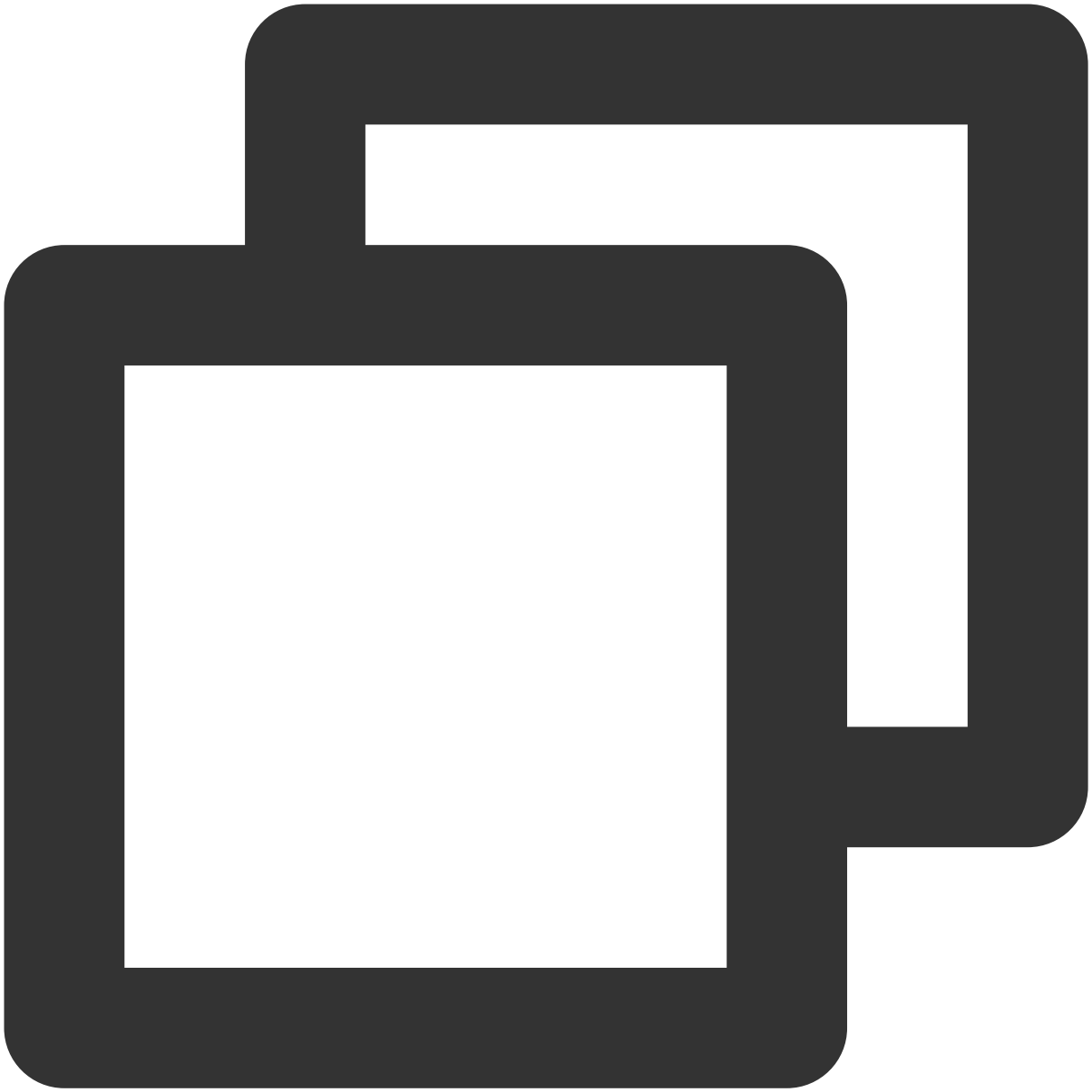
```
ARRAY_MAX(<a> array<T>)
```

支持引擎：SparkSQL、Presto。

使用说明：返回数组中的最大值。对于 double/float 类型，NaN 大于任何非 NaN 元素。跳过空元素。

返回类型：array<T>。

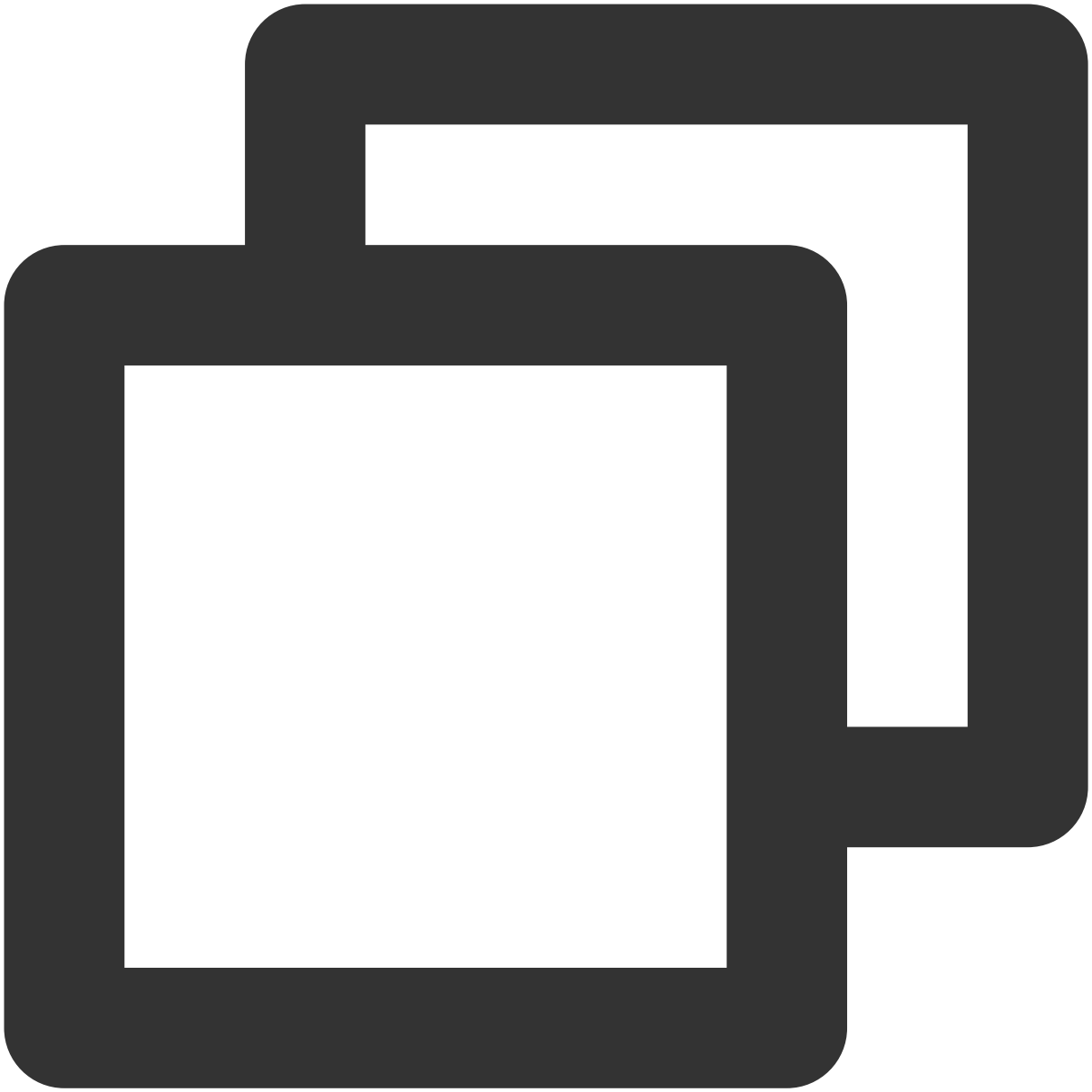
示例：



```
> SELECT array_max(array(1, 20, null, 3));
 20
```

ARRAY_MIN

函数语法：



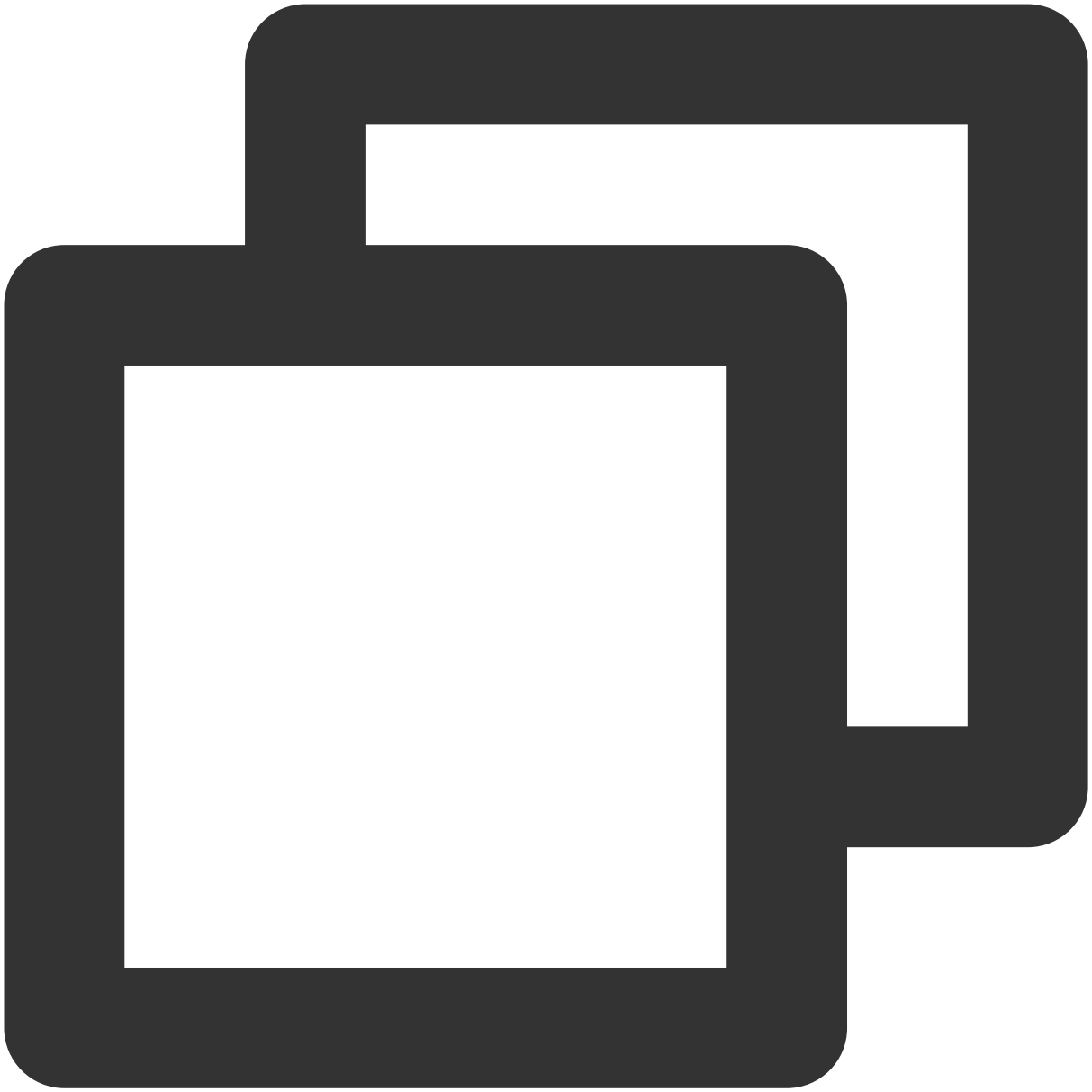
```
ARRAY_MIN(<a> array<T>)
```

支持引擎：SparkSQL、Presto。

使用说明：返回数组中的最小值。对于 double/float 类型，NaN 大于任何非 NaN 元素。跳过空元素。

返回类型：array<T>。

示例：



```
> SELECT array_min(array(1, 20, null, 3));
 1
```

FLATTEN

函数语法：

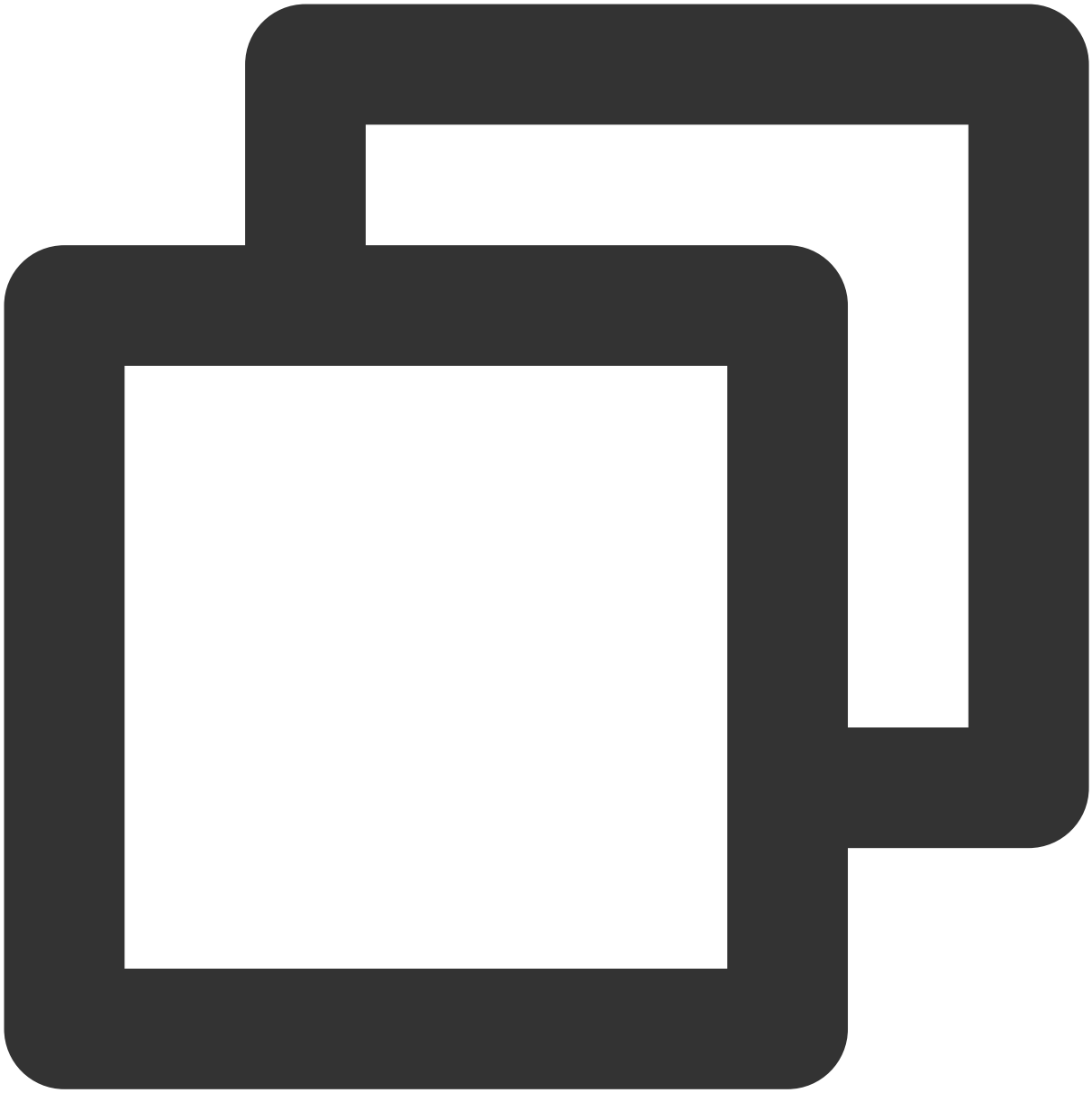
```
FLATTEN(<aa> array<array<T>>
```

支持引擎：SparkSQL、Presto。

使用说明：将二位数组转换为一维数组。

返回类型：array<T>。

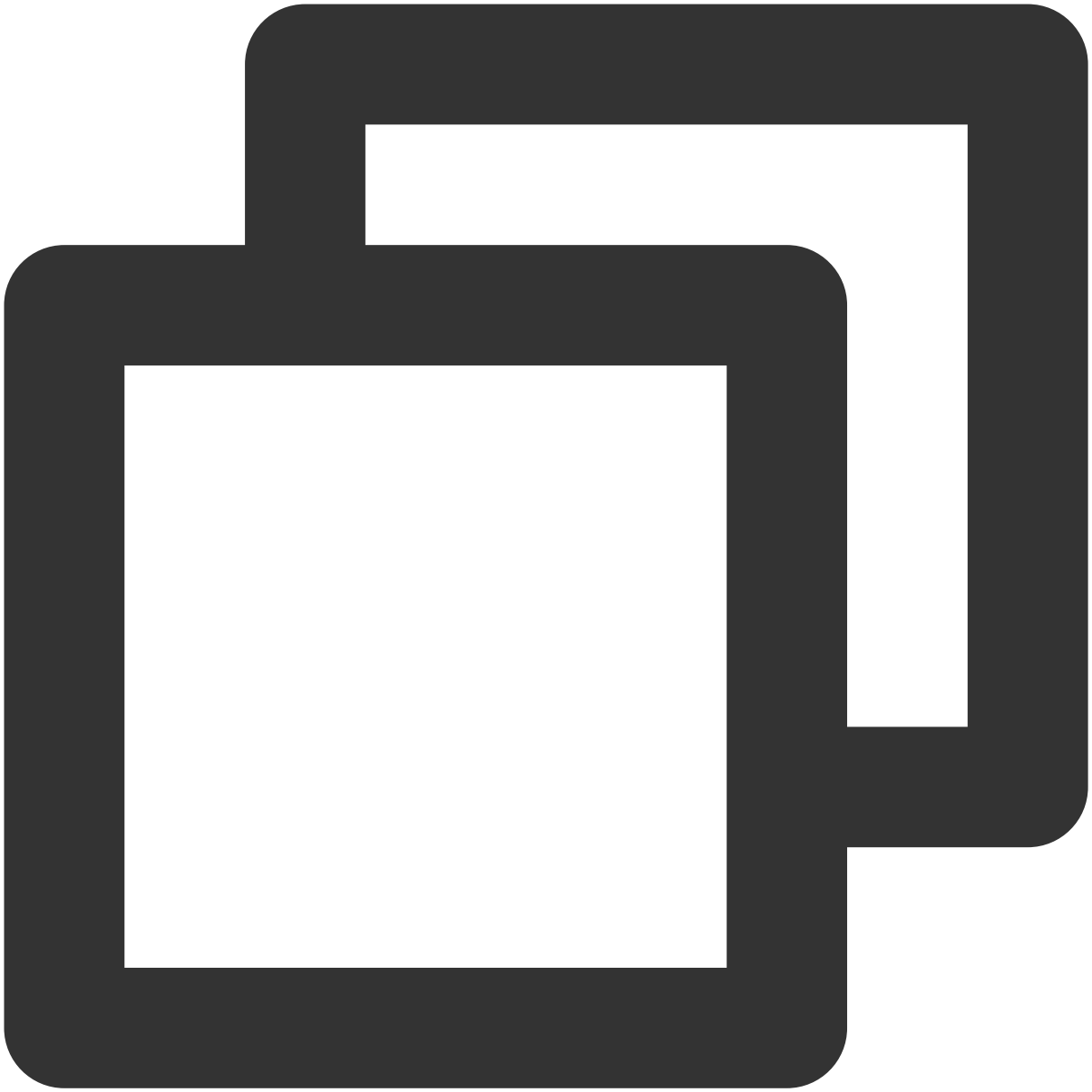
示例：



```
> SELECT flatten(array(array(1, 2), array(3, 4)));
 [1,2,3,4]
```

SEQUENCE

函数语法：



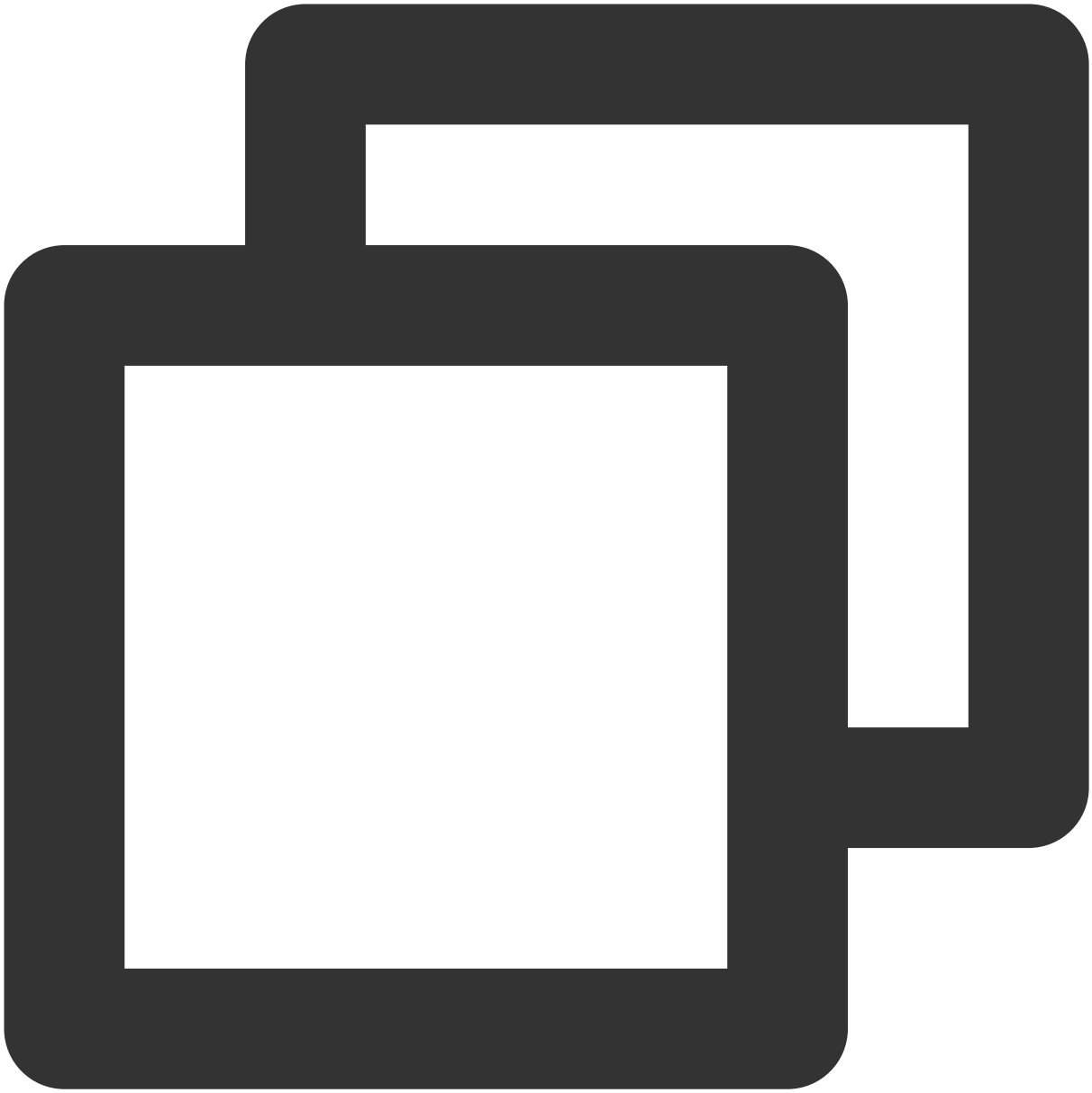
```
SEQUENCE(<start> integer|date|timestamp, end integer|date|timestamp[, step integer|
```

支持引擎：SparkSQL、Presto

使用说明：生成从 start 到 end（包含 end）的数组，并逐步递增。返回元素的类型与 start 与 end 的类型相同。start 和 stop 表达式必须解析为同一类型。如果开始和停止表达式解析为 date 或 timestamp 类型，则 step 必须解析为 interval 或 year-month interval 或 day-time interval 类型，否则解析为与 start 与 end 相同的类型。

返回类型： `与start` 相同

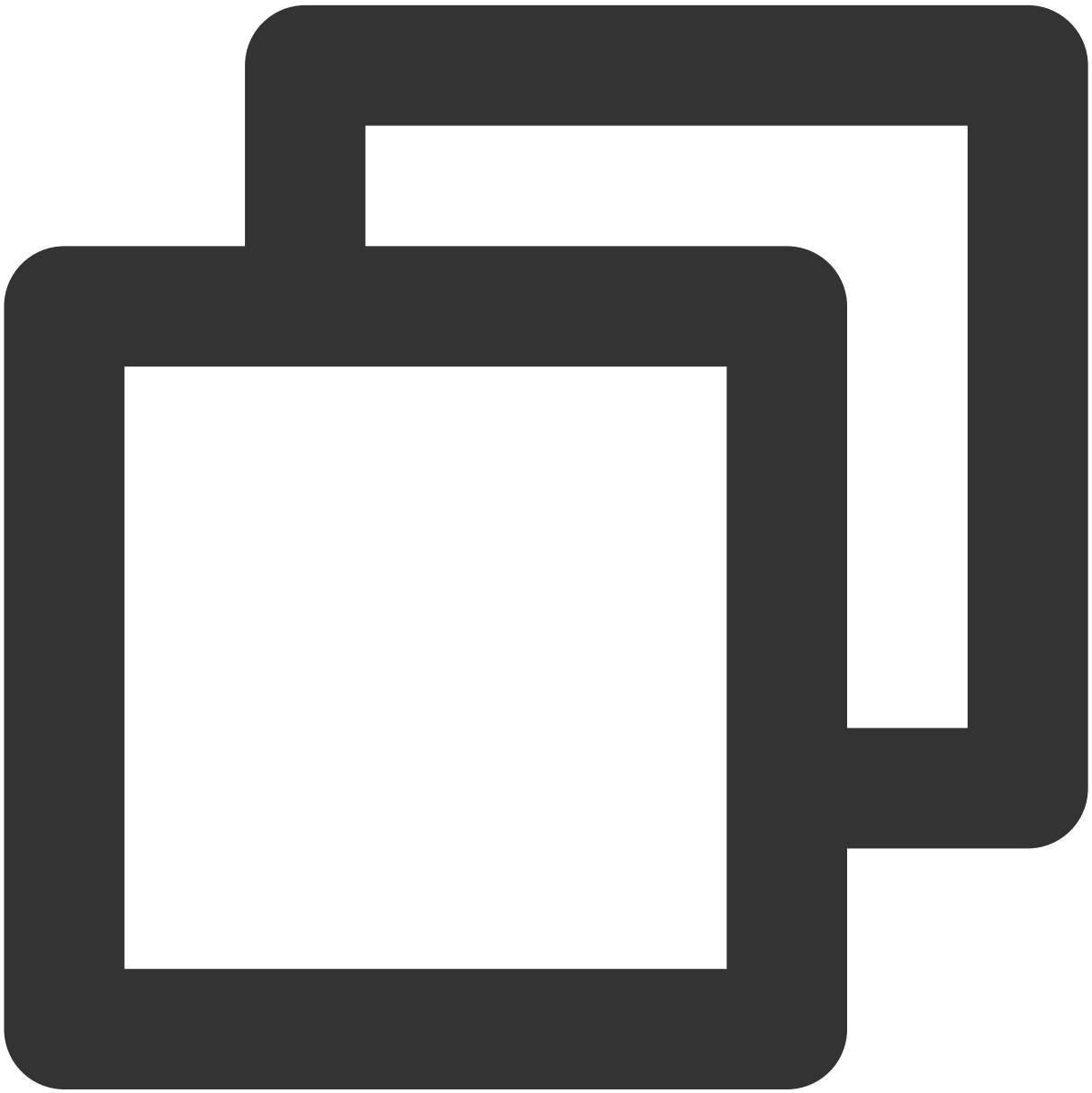
示例：



```
> SELECT sequence(1, 5);
 [1,2,3,4,5]
> SELECT sequence(5, 1);
 [5,4,3,2,1]
> SELECT sequence(to_date('2018-01-01'), to_date('2018-03-01'), interval 1 month);
 [2018-01-01,2018-02-01,2018-03-01]
> SELECT sequence(to_date('2018-01-01'), to_date('2018-03-01'), interval '0-1' year
 [2018-01-01,2018-02-01,2018-03-01]
```

ARRAY_REPEAT

函数语法：



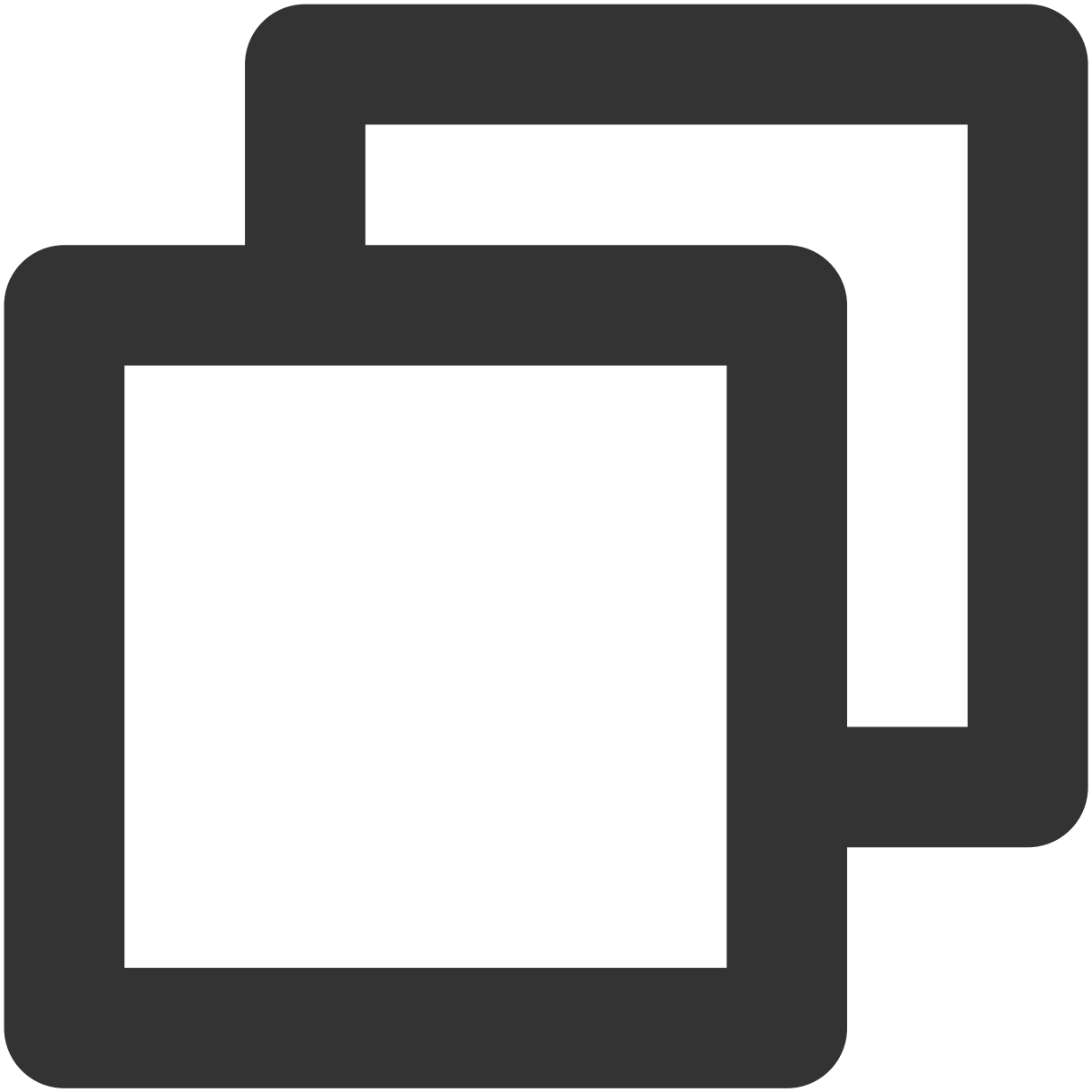
```
ARRAY_REPEAT(<element> T, <count> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：返回包含 count 个 element 的数组。

返回类型：array<T>。

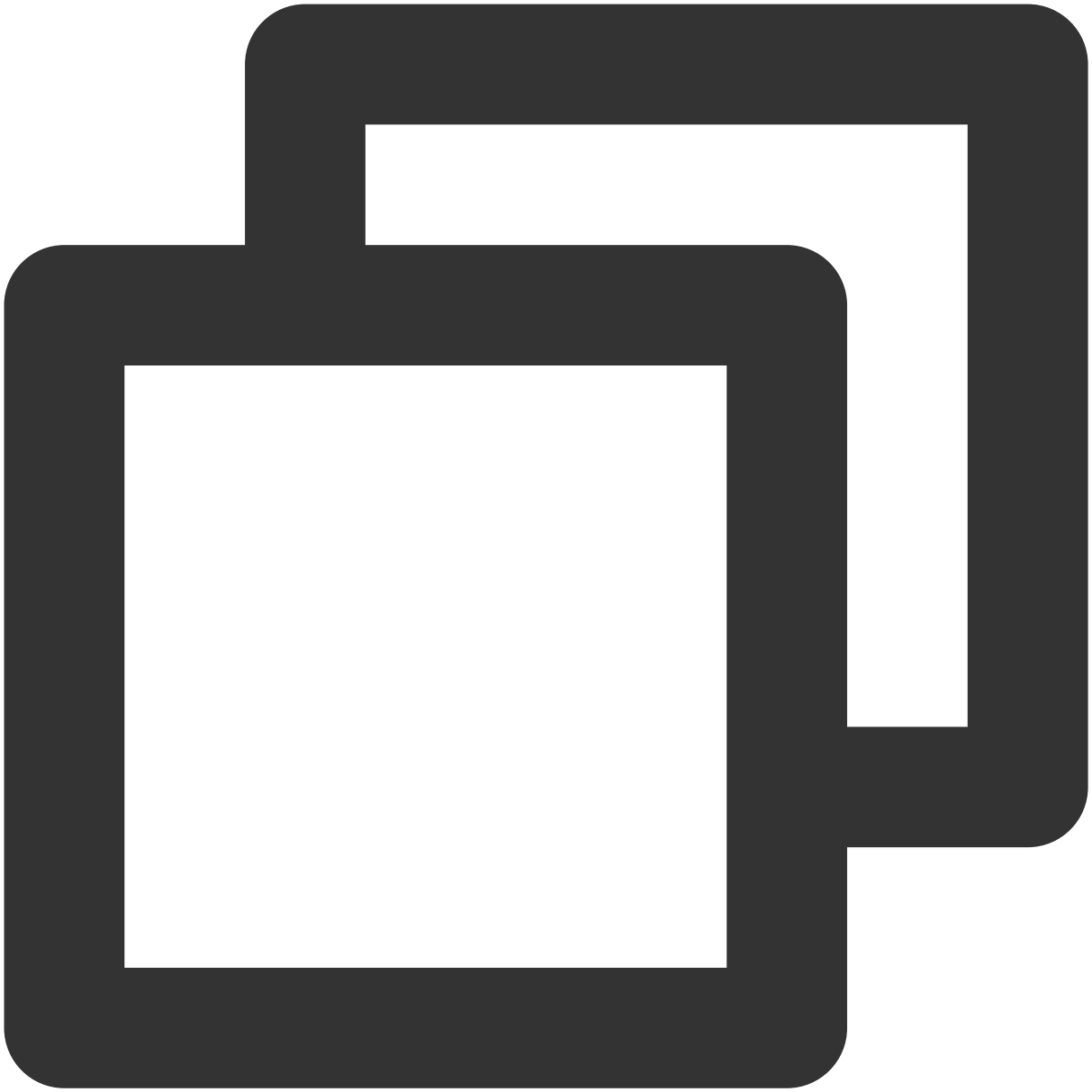
示例：



```
> SELECT array_repeat('123', 2);
 ["123","123"]
```

ARRAY_REMOVE

函数语法：



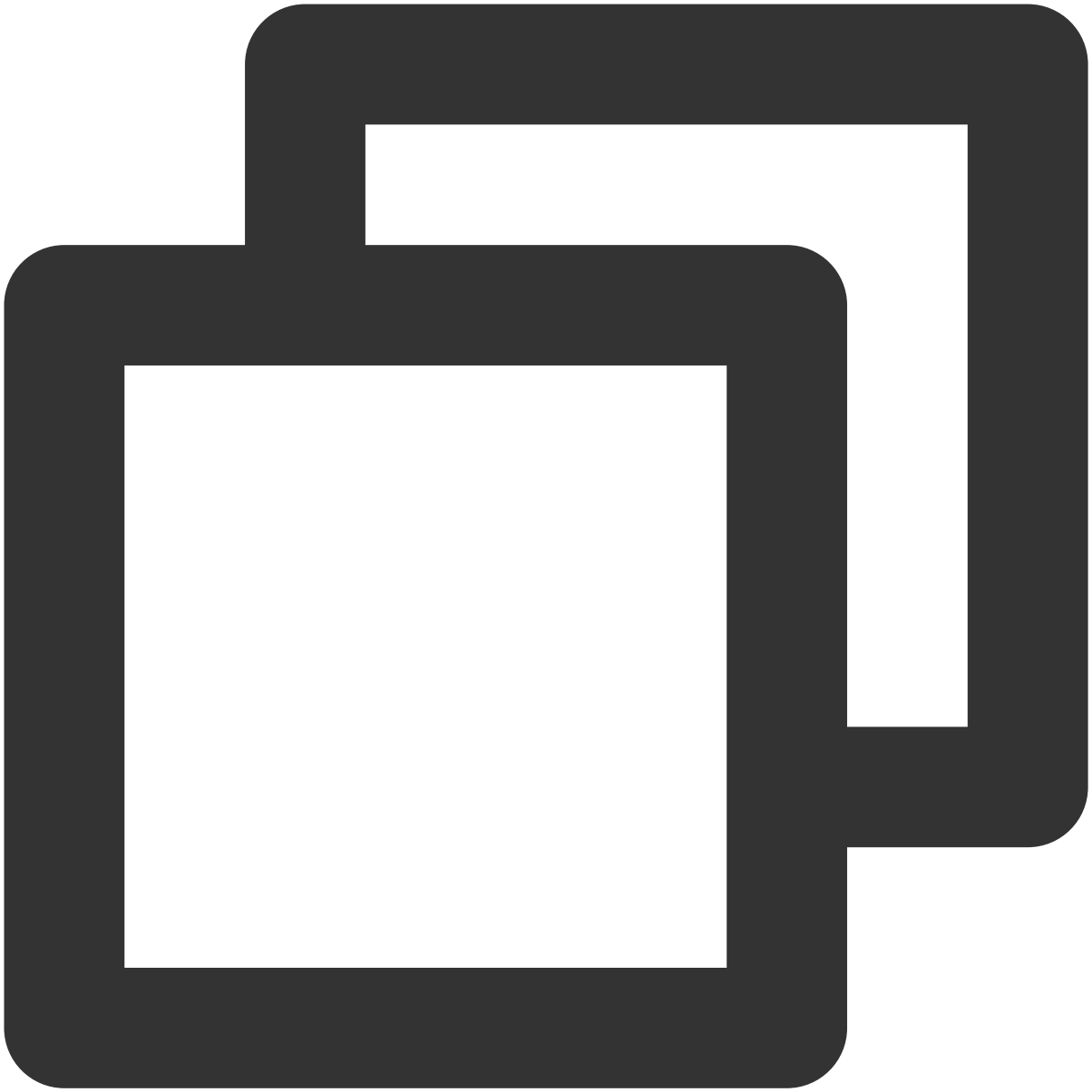
```
ARRAY_REMOVE(<a> array<T>, <element> T)
```

支持引擎：SparkSQL、Presto。

使用说明：从数组中删除所有等于元素的元素。

返回类型：array<T>。

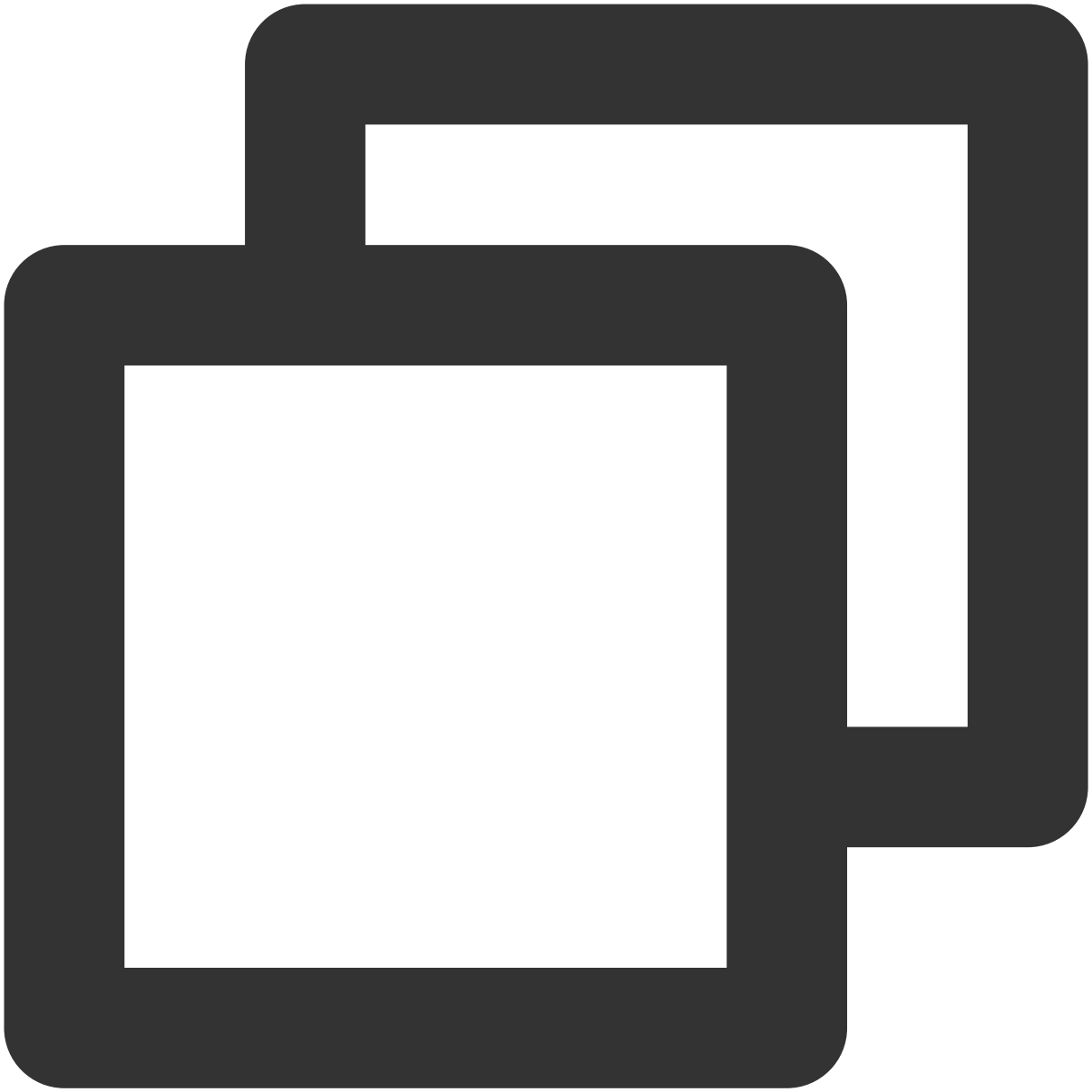
示例：



```
> SELECT array_remove(array(1, 2, 3, null, 3), 3);
 [1,2,null]
```

ARRAY_DISTINCT

函数语法：



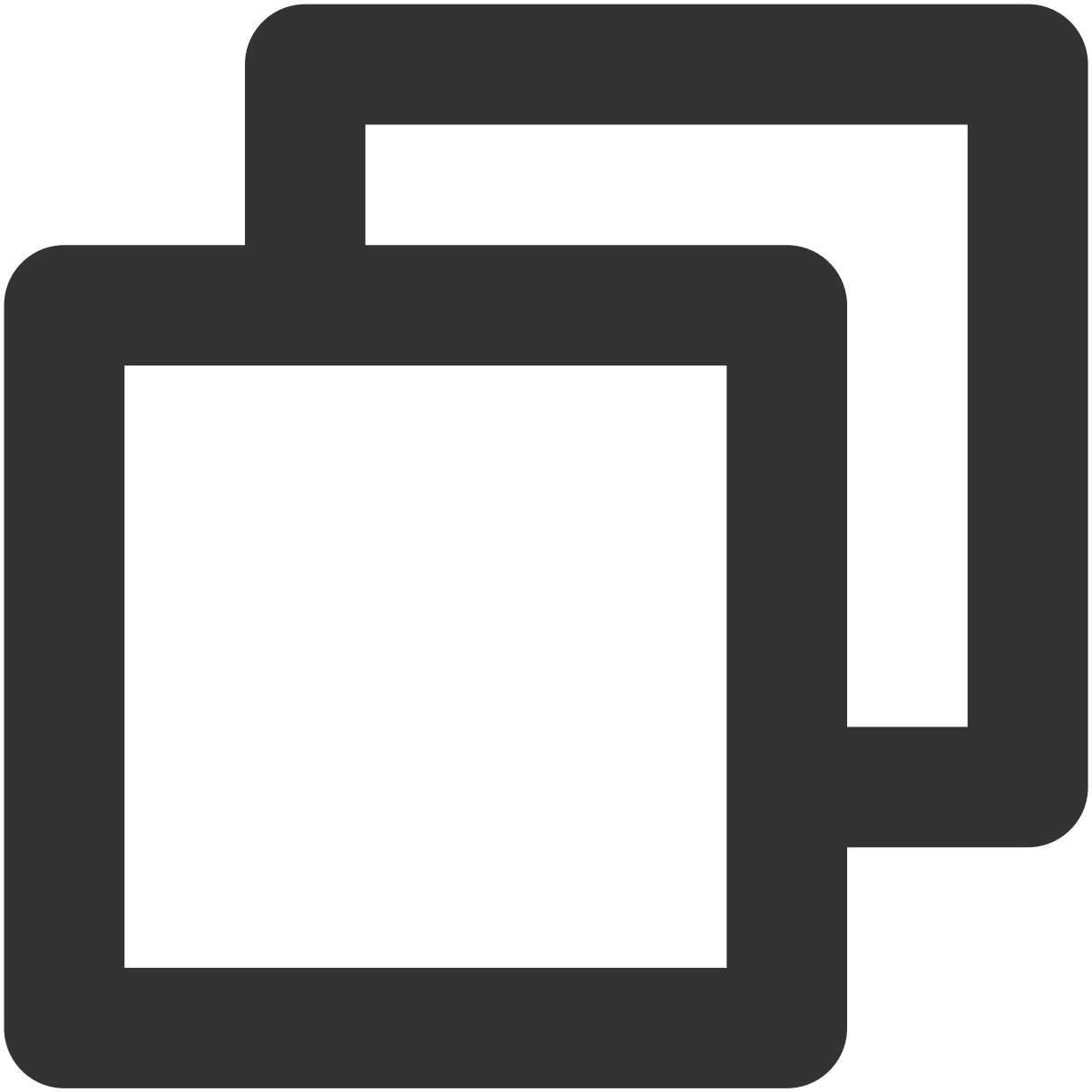
```
ARRAY_DISTINCT(<a> array<T>)
```

支持引擎：SparkSQL、Presto。

使用说明：从数组中删除重复值。

返回类型：array<T>。

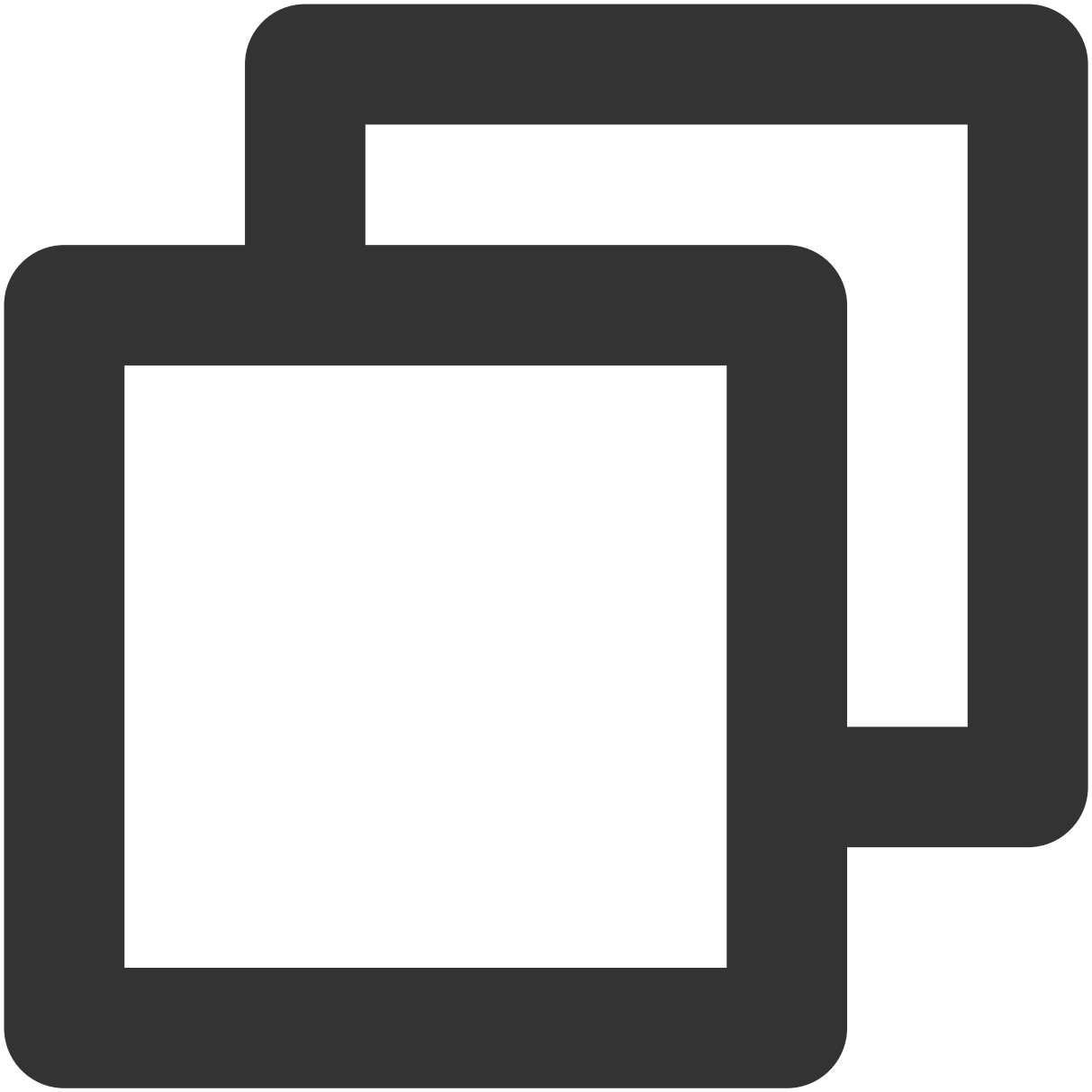
示例：



```
> SELECT array_distinct(array(1, 2, 3, null, 3));
 [1,2,3,null]
```

ELEMENT_AT

函数语法：



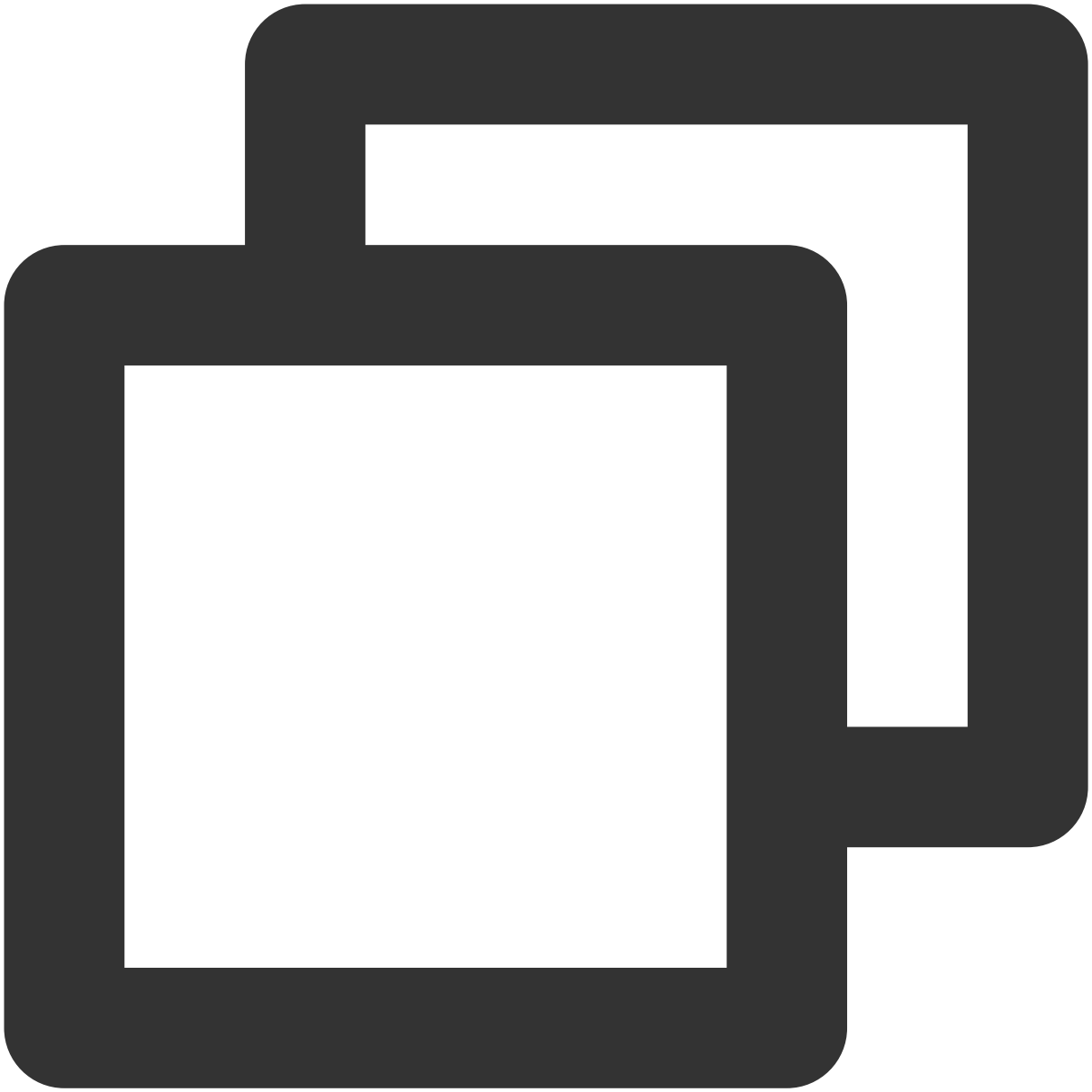
```
ELEMENT_AT(<a> array<T>, <index> integer)
ELEMENT_AT(<m> map<K, V>, <key> K)
```

支持引擎：SparkSQL、Presto

使用说明：返回给定（从1开始计数）索引处的数组元素或返回给定键的值。

返回类型：T，V

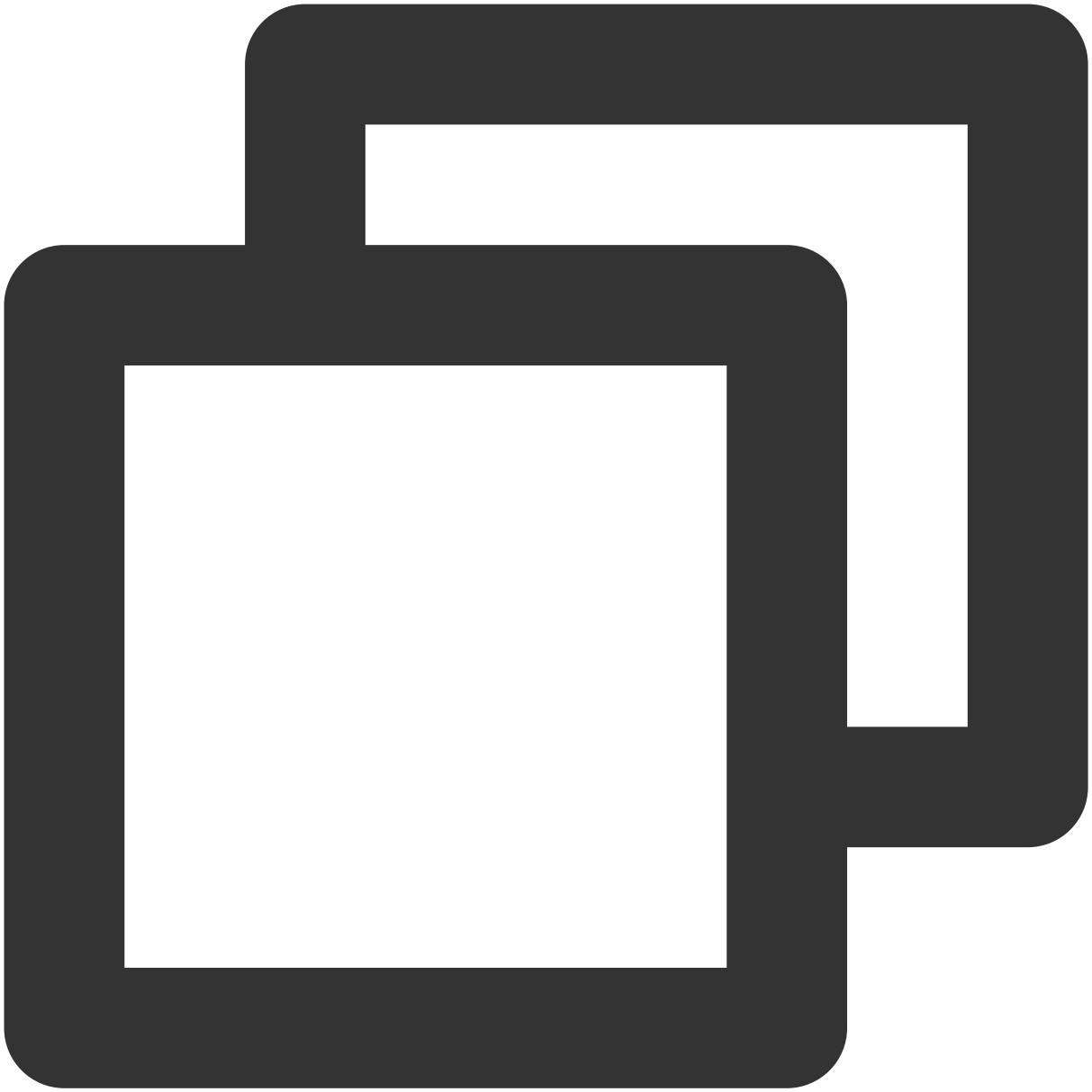
示例：



```
> SELECT element_at(array(1, 2, 3), 2);
 2
> SELECT element_at(map(1, 'a', 2, 'b'), 2);
 b
```

MAP

函数语法：



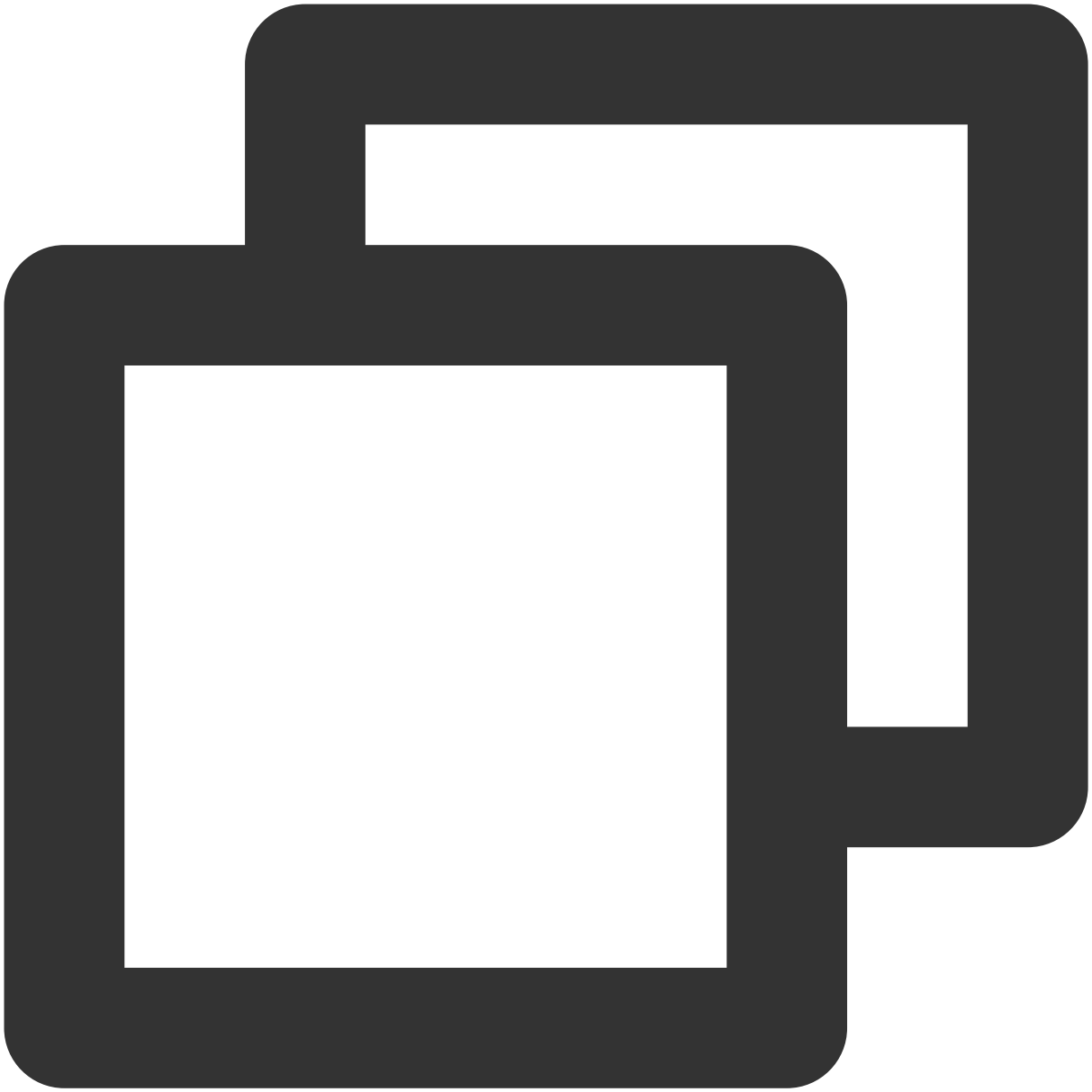
```
MAP(<k1> K, <v1> V, ...)
```

支持引擎：SparkSQL、Presto。

使用说明：根据给定元素生成 map。

返回类型：MAP<K, V>。

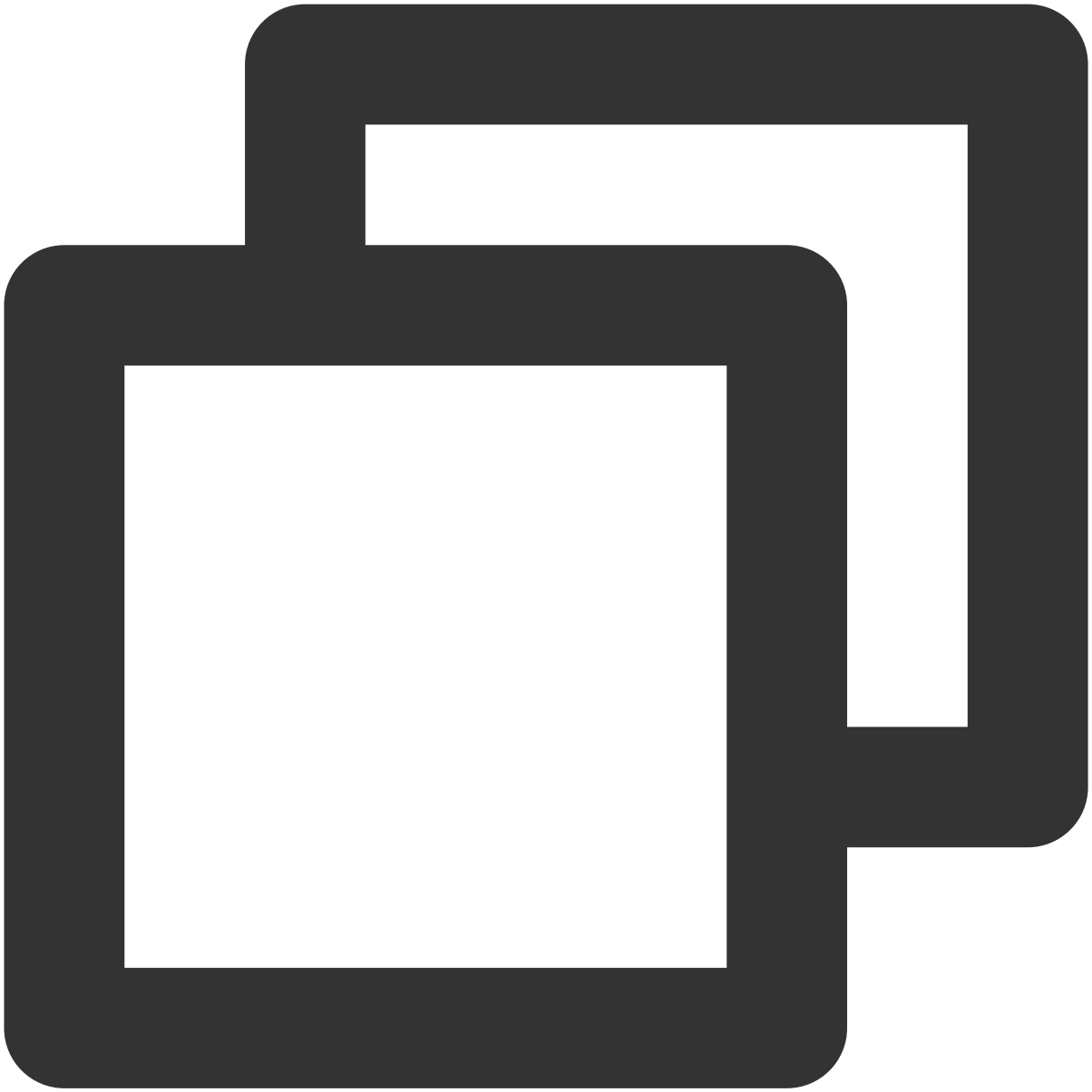
示例：



```
> SELECT map(1.0, '2', 3.0, '4');
 {1.0:"2",3.0:"4"}
```

MAP_FROM_ARRAYS

函数语法：



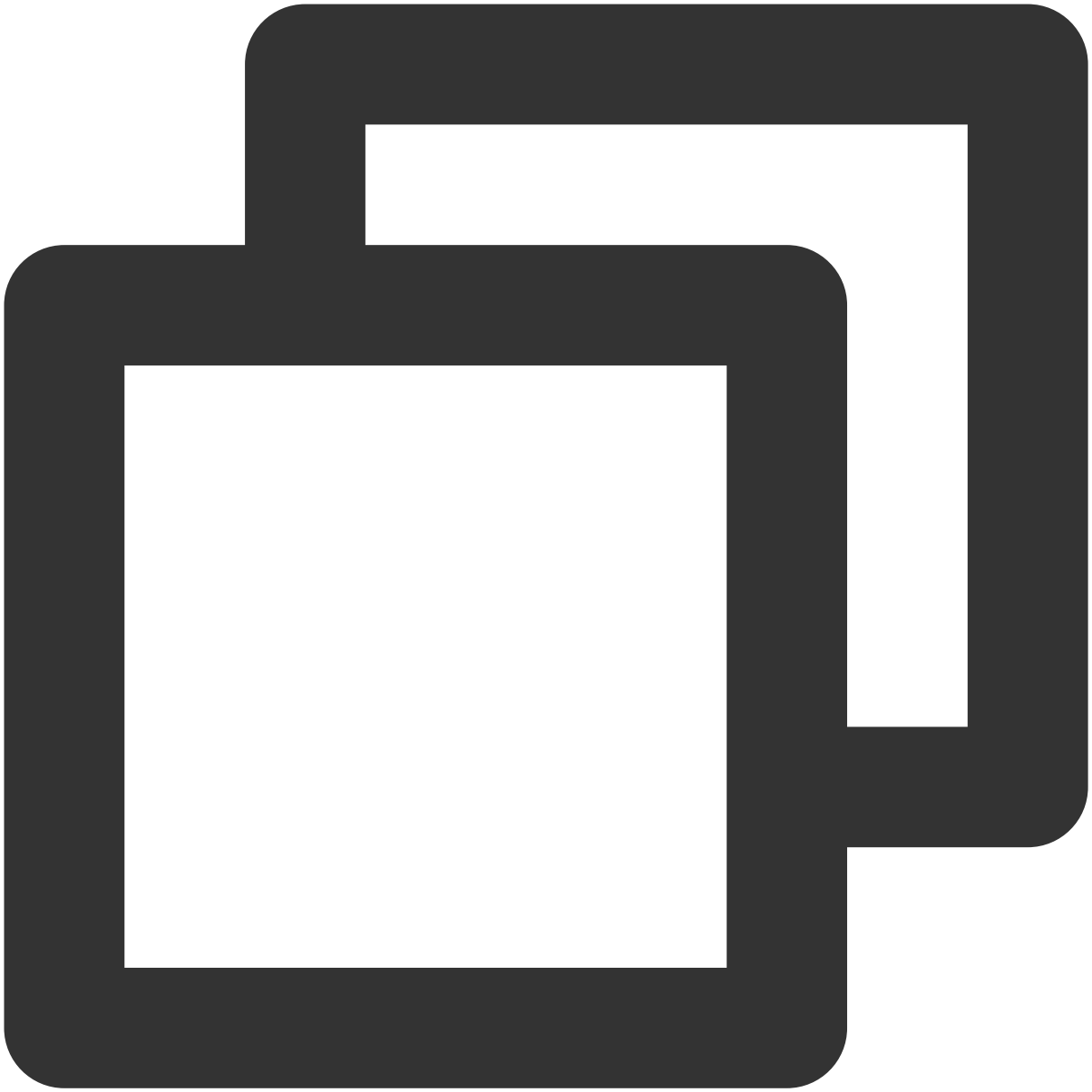
```
MAP_FROM_ARRAYS(<keys> array<K>, <values> array<V>)
```

支持引擎：SparkSQL、Presto。

使用说明：使用一对给定的 key/value 数组创建 map。keys 中的所有元素都不应为 null。

返回类型：map<K, V>。

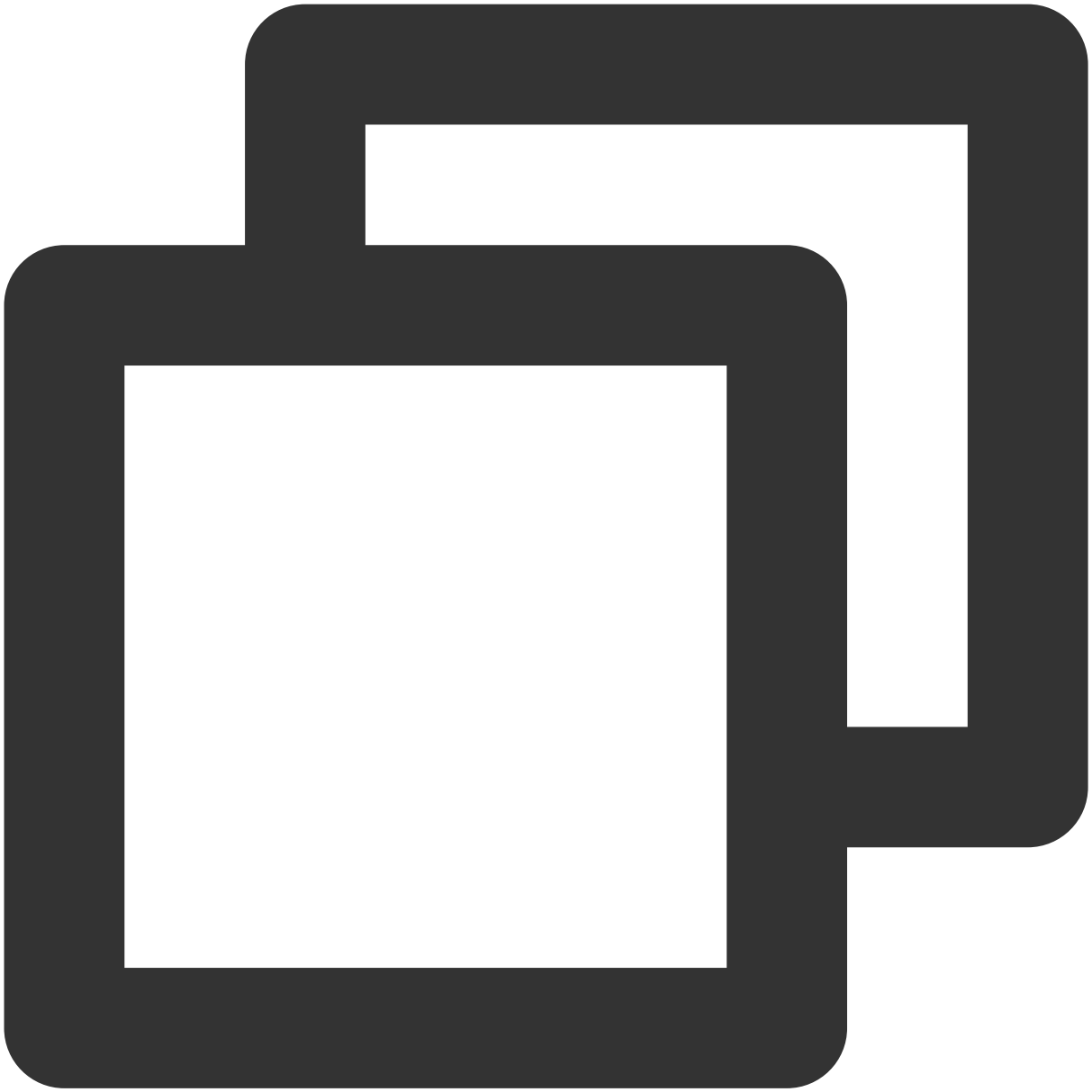
示例：



```
> SELECT map_from_arrays(array(1.0, 3.0), array('2', '4'));
 {1.0:"2",3.0:"4"}
```

MAP_KEYS

函数语法：



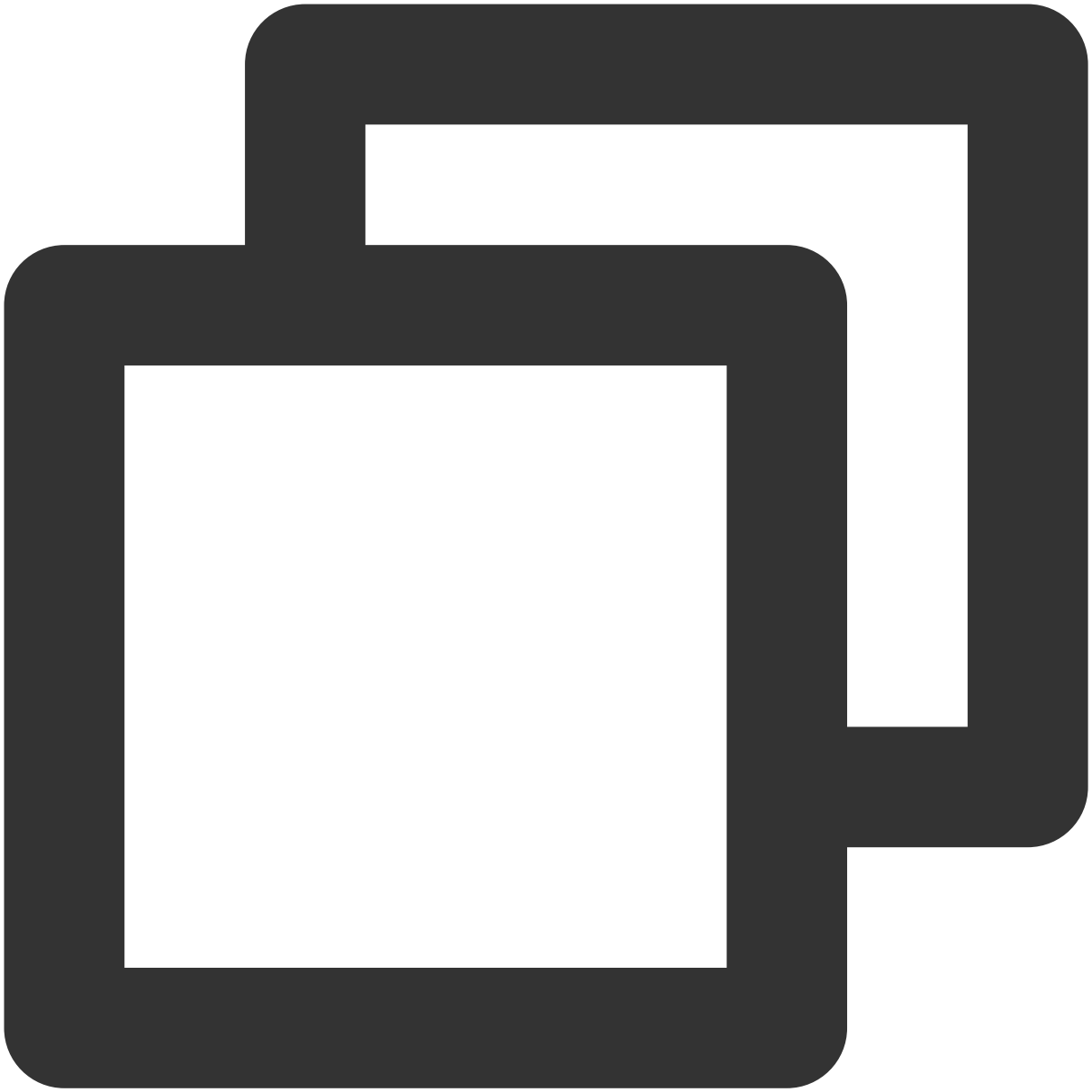
```
MAP_KEYS(<m> map<K, V>)
```

支持引擎：SparkSQL、Presto。

使用说明：返回包含 map 键的无序数组。

返回类型：array<K>。

示例：



```
> SELECT map_keys(map(1, 'a', 2, 'b'));
 [1,2]
```

MAP_VALUES

函数语法：

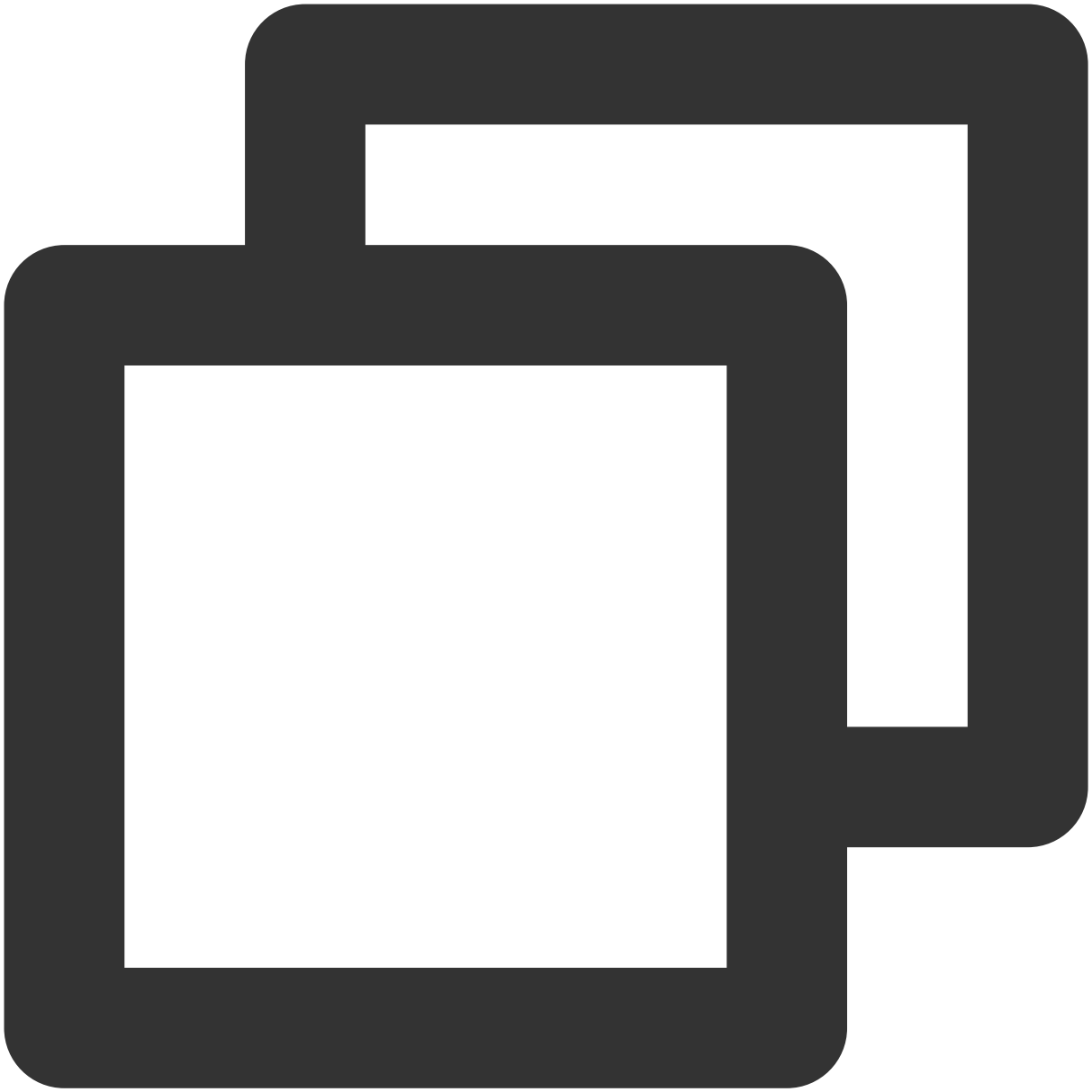
```
MAP_VALUES(<m> map<K, V>)
```

支持引擎：SparkSQL、Presto。

使用说明：返回包含 map 值的无序数组。

返回类型：array<V>。

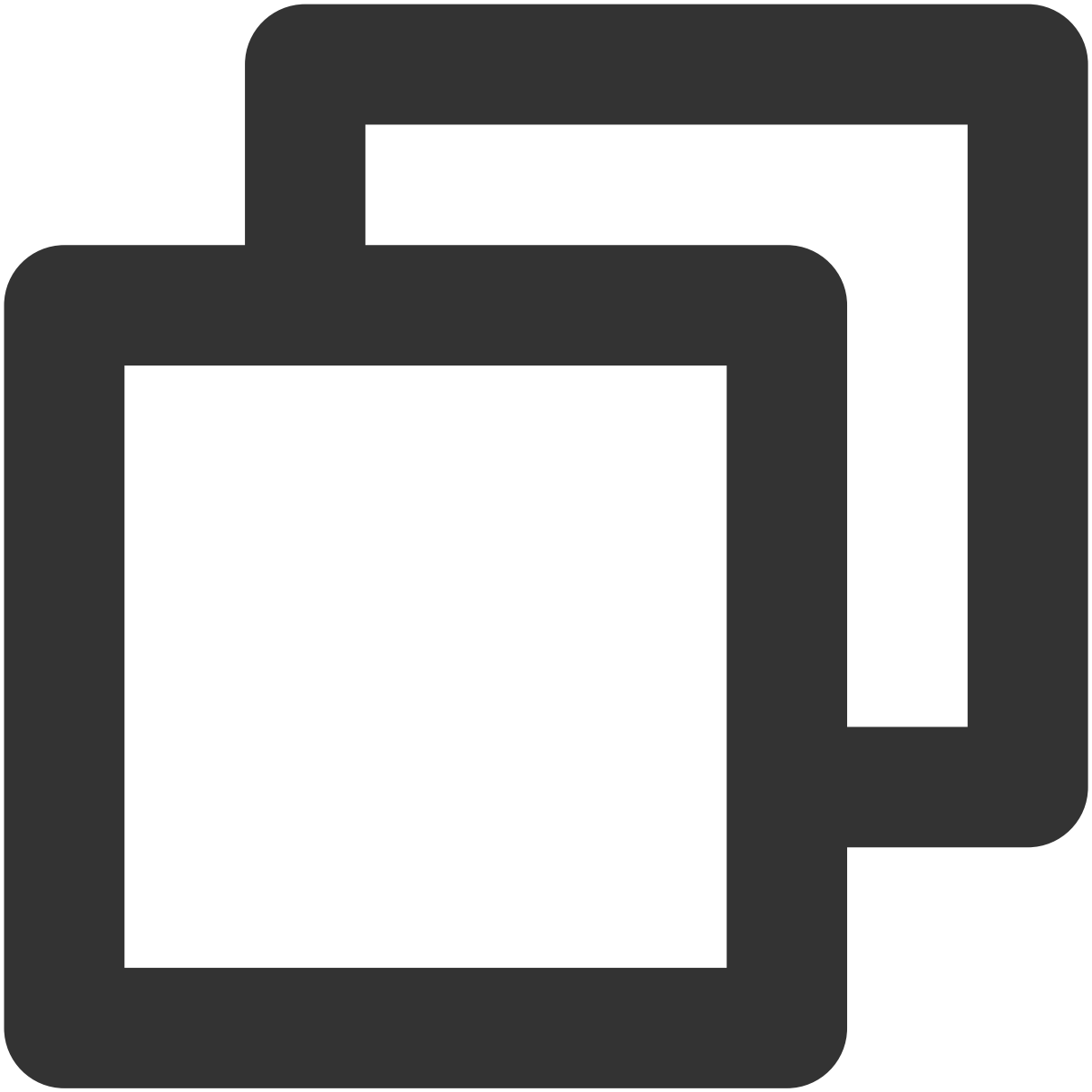
示例：



```
> SELECT map_values(map(1, 'a', 2, 'b'));
 ["a","b"]
```

MAP_ENTRIES

函数语法：



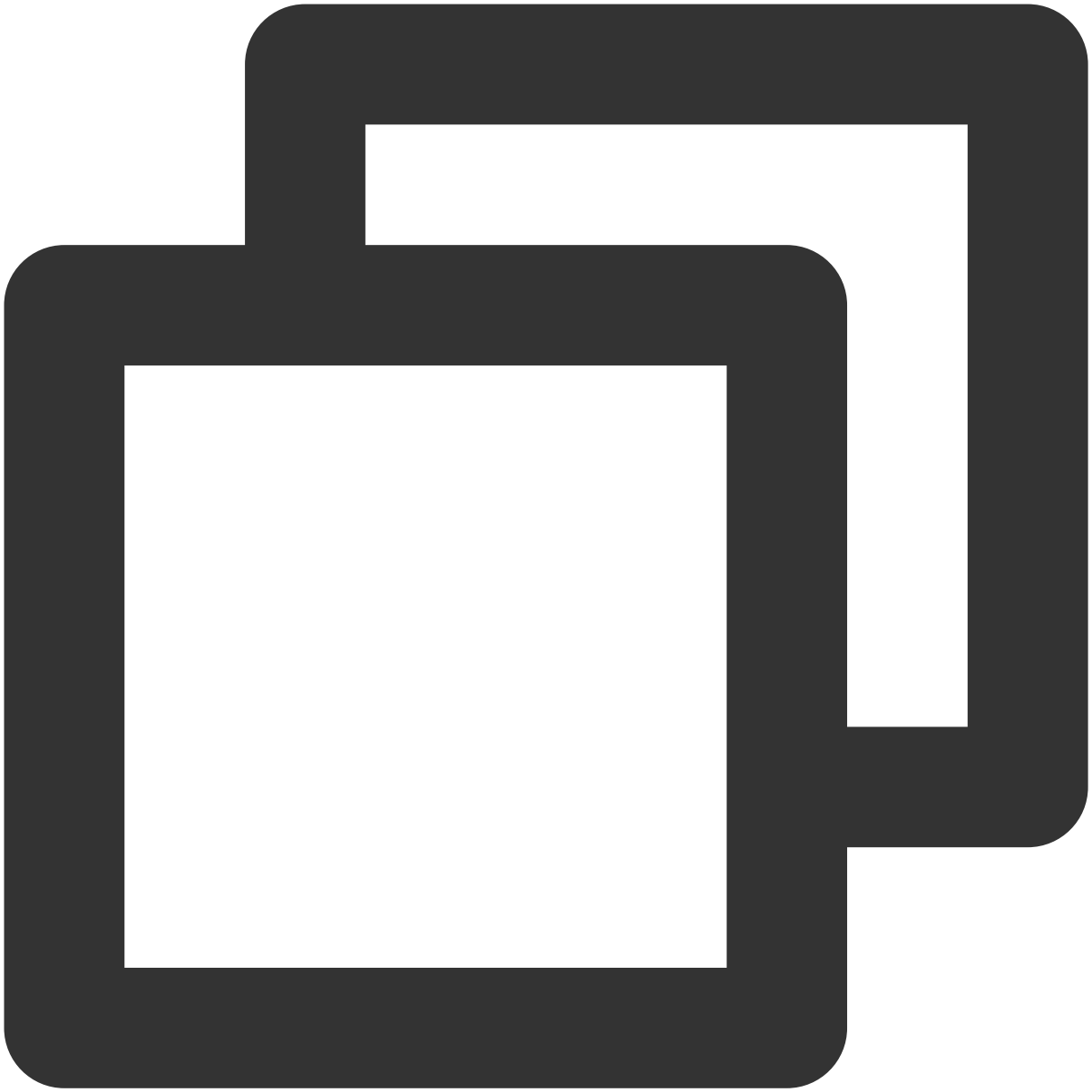
```
MAP_ENTRIES(<m> map<K, V>)
```

支持引擎：SparkSQL、Presto。

使用说明：返回给定 map 中所有项的无序数组。

返回类型：array<struct<K, V>>。

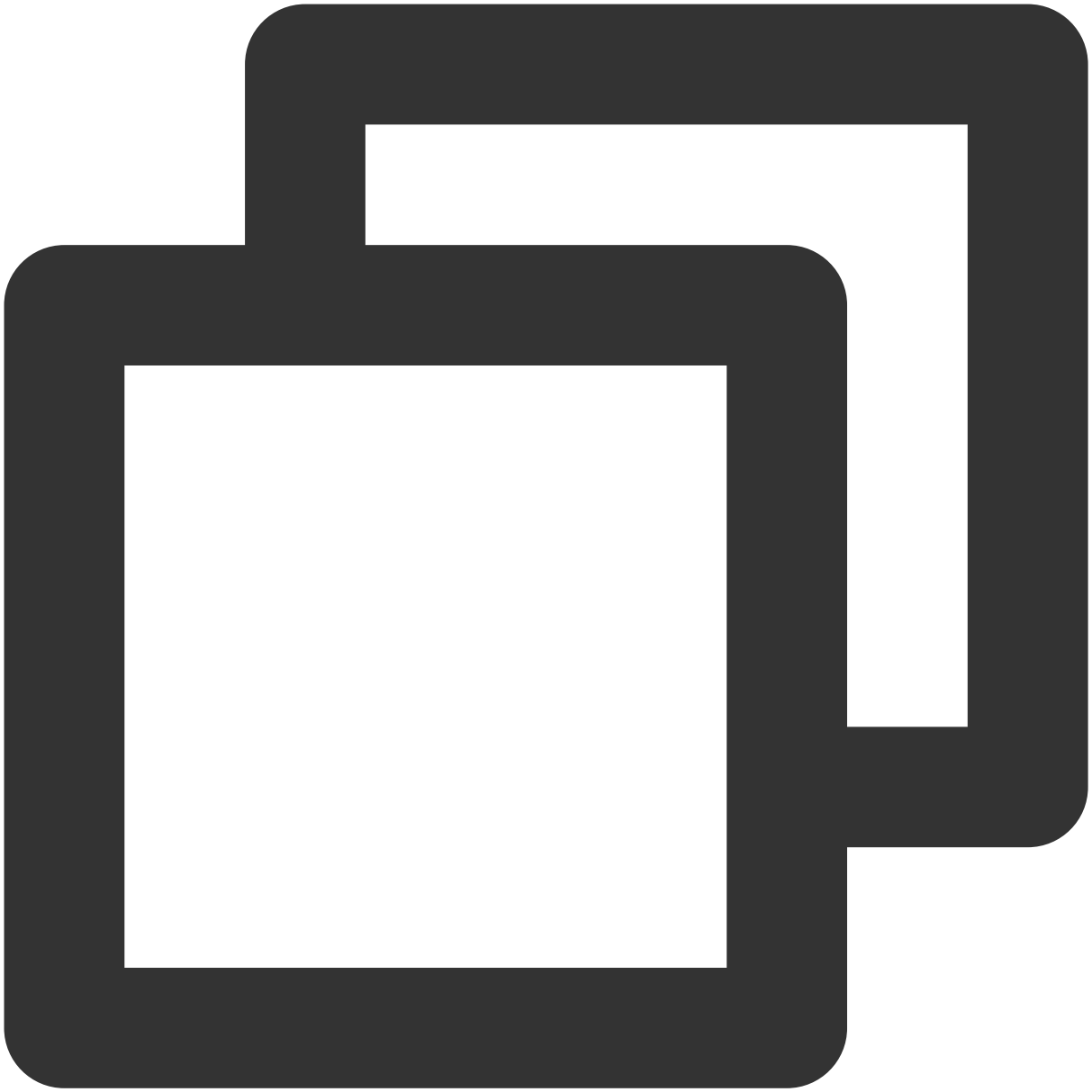
示例：



```
> SELECT map_entries(map(1, 'a', 2, 'b'));
 [{"key":1,"value":"a"},{"key":2,"value":"b"}]
```

MAP_FROM_ENTRIES

函数语法：



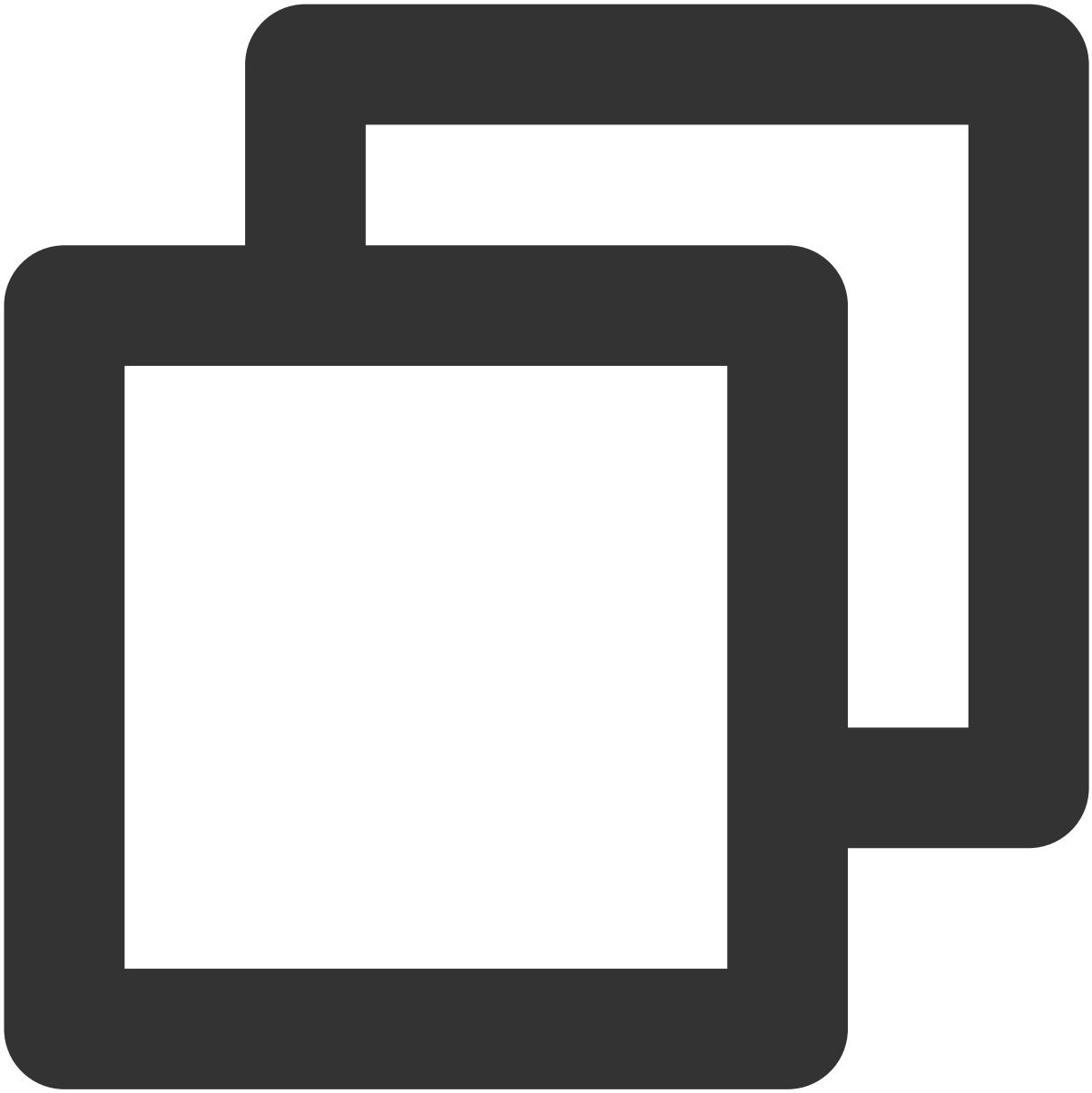
```
MAP_FROM_ENTRIES(<entries> array<struct<K, V>>)
```

支持引擎：SparkSQL、Presto。

使用说明：返回从给定的条目数组创建的映射。

返回类型：map<K, V>。

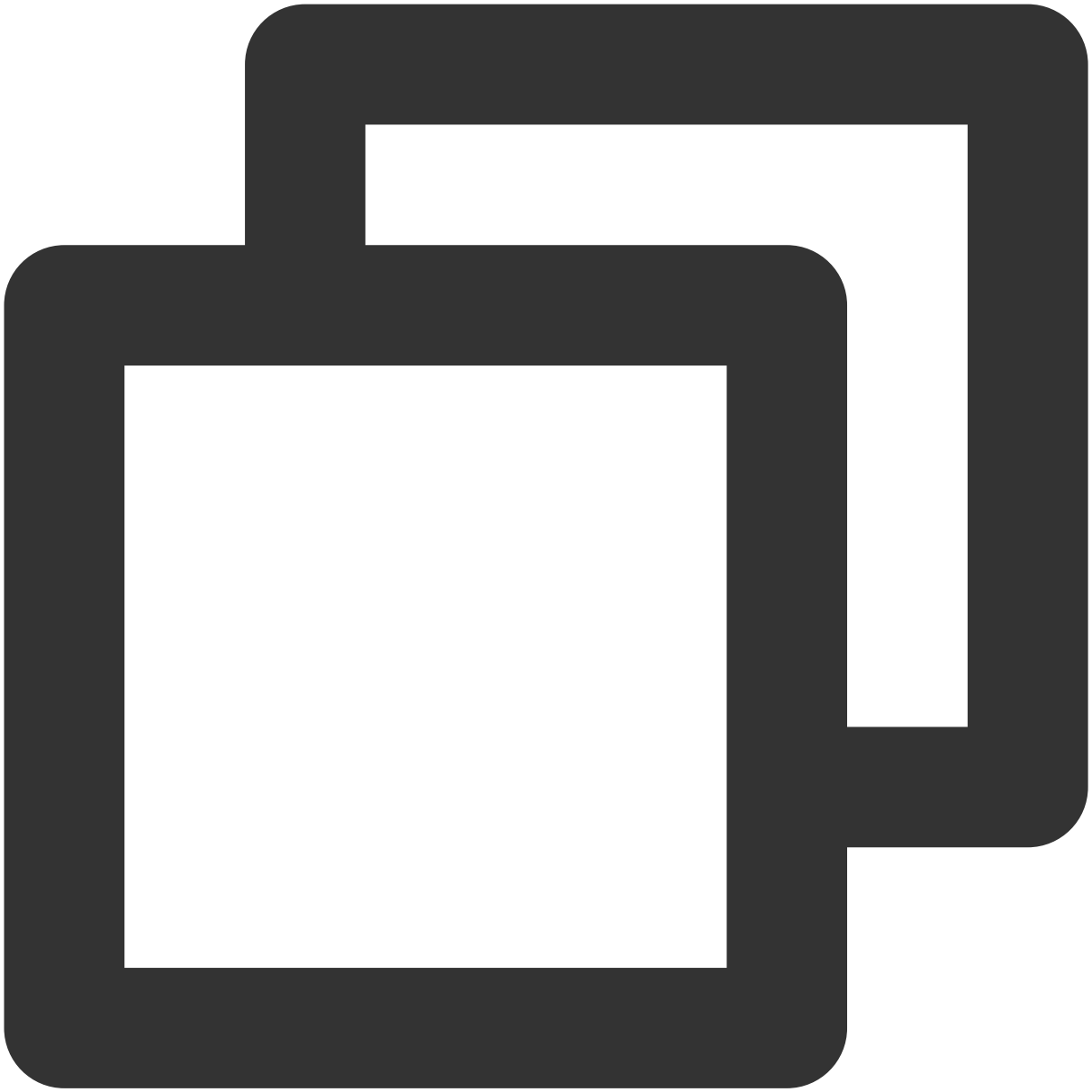
示例：



```
> SELECT map_from_entries(array(struct(1, 'a'), struct(2, 'b')));
 {1:"a",2:"b"}
```

MAP_CONCAT

函数语法：



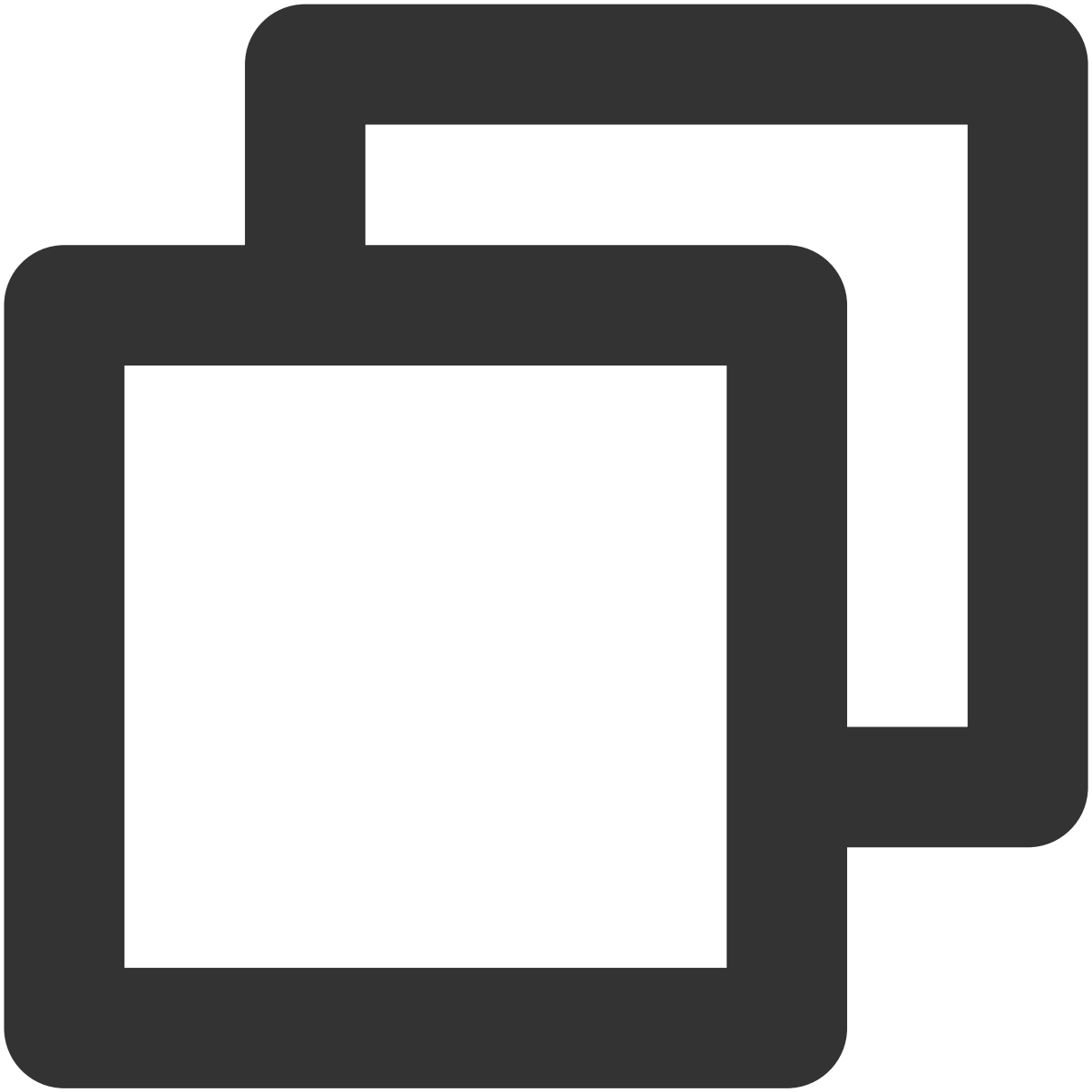
```
MAP_CONCAT(map1 map<K, V>, ...)
```

支持引擎：SparkSQL、Presto。

使用说明：返回所有给定映射的并集。

返回类型：map<K, V>。

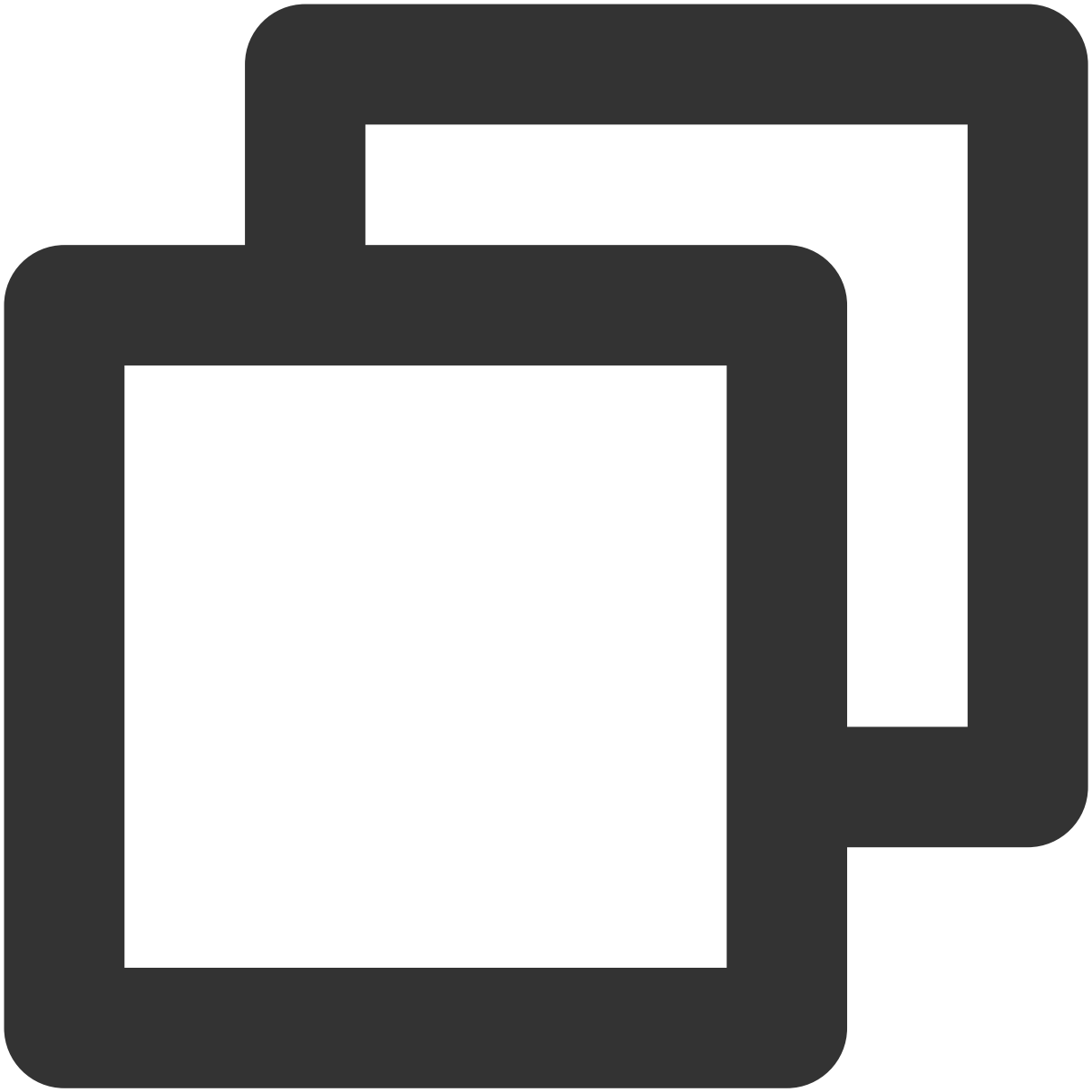
示例：



```
> SELECT map_concat(map(1, 'a', 2, 'b'), map(3, 'c'));
 {1:"a",2:"b",3:"c"}
```

MAP_FILTER

函数语法：



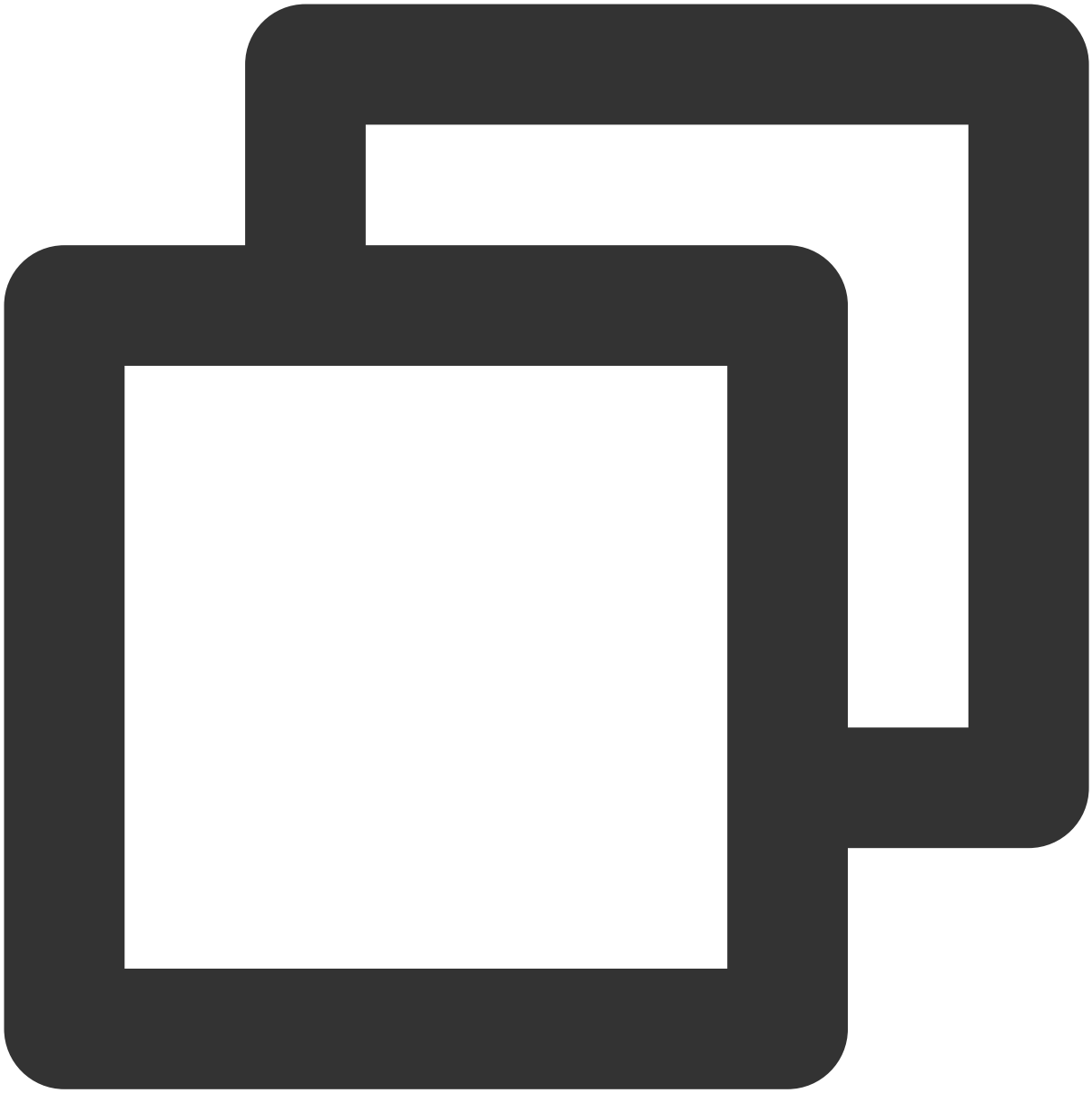
```
MAP_FILTER(<m> map<K, V>, <func> function(K, V)->boolean)
```

支持引擎：SparkSQL。

使用说明：使用 func 过滤 m 中的条目。

返回类型：map<K, V>。

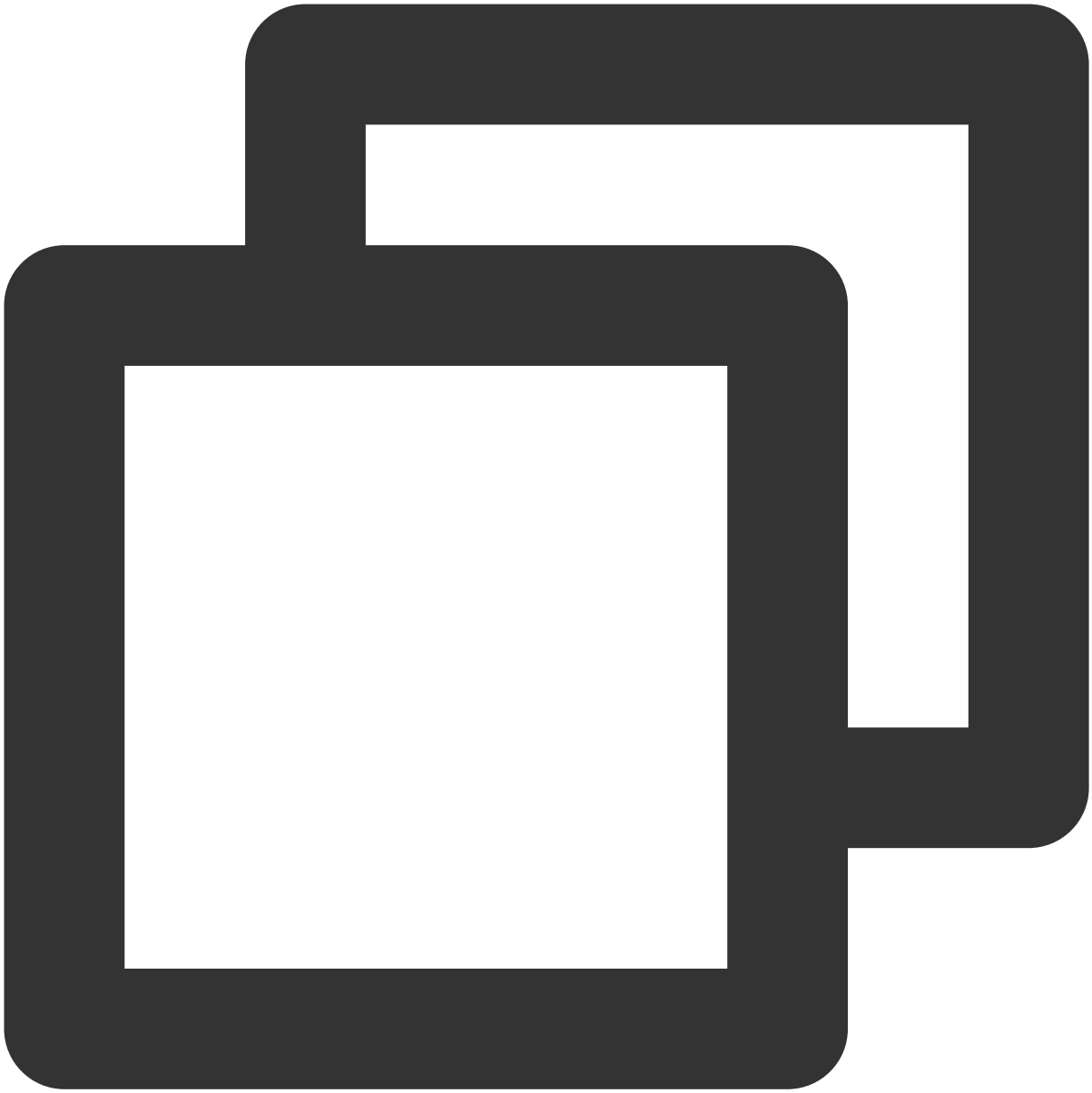
示例：



```
> SELECT map_filter(map(1, 0, 2, 2, 3, -1), (k, v) -> k > v);
 {1:0,3:-1}
```

MAP_ZIP_WITH

函数语法：



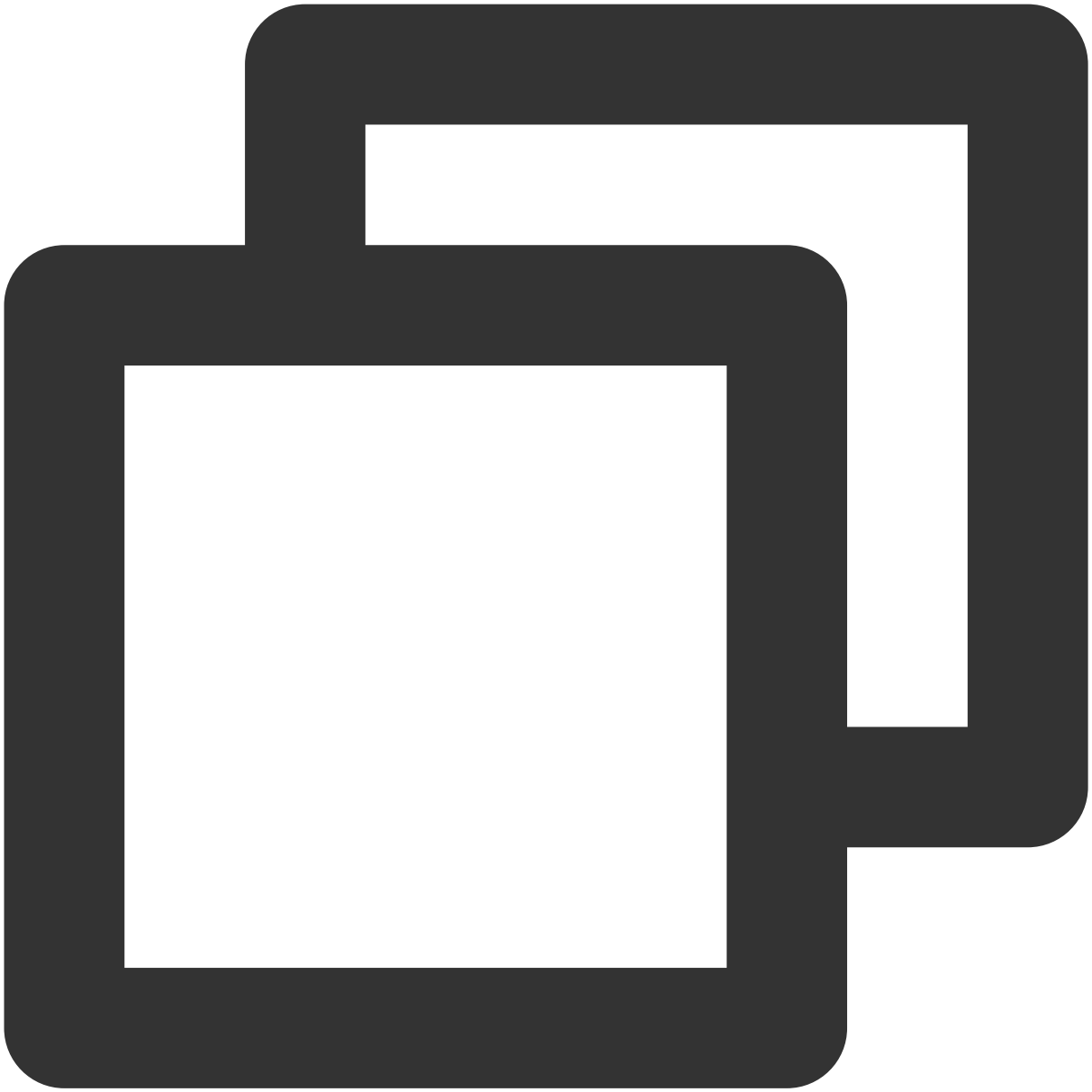
```
MAP_ZIP_WITH(<map1> map<K, V1>, <map2> map<K, V2>, <func> function(K, V1, V2)->R)
```

支持引擎：SparkSQL。

使用说明：通过 func 将两个给定映射合并为单个映射。对于仅在一个映射中显示的键，值将被置为 NULL。如果输入映射包含重复键，则仅将重复键的第一个条目传递给 func。

返回类型：MAP<K, R>。

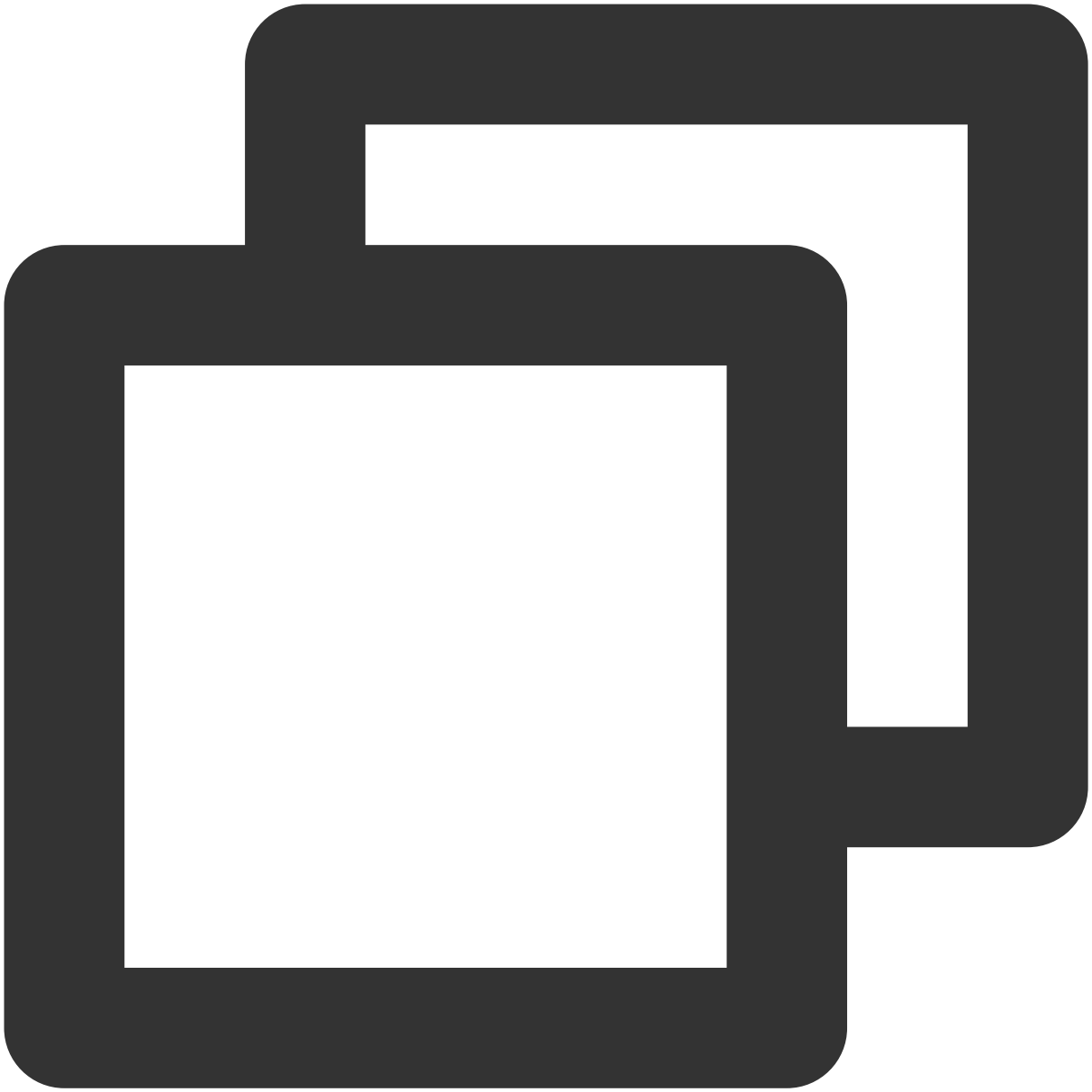
示例：



```
> SELECT map_zip_with(map(1, 'a', 2, 'b'), map(1, 'x', 2, 'y'), (k, v1, v2) -> conc
 {1:"ax",2:"by"}
```

TRANSFORM_KEYS

函数语法：



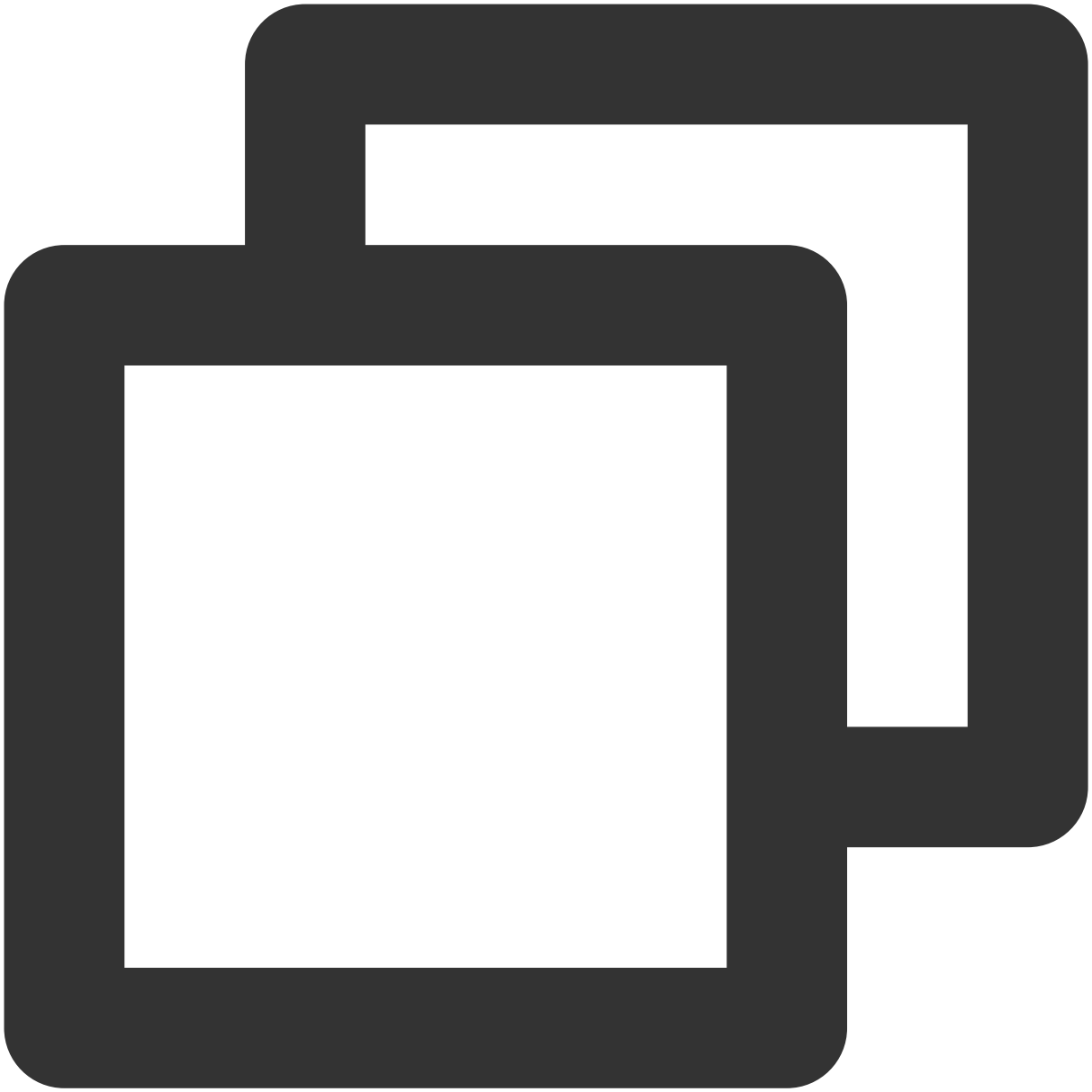
```
TRANSFORM_KEYS(<m> map<K, V>, <func> function(K, V)->R)
```

支持引擎：SparkSQL。

使用说明：使用 func 变换 map 中的 keys。

返回类型：map<R, V>。

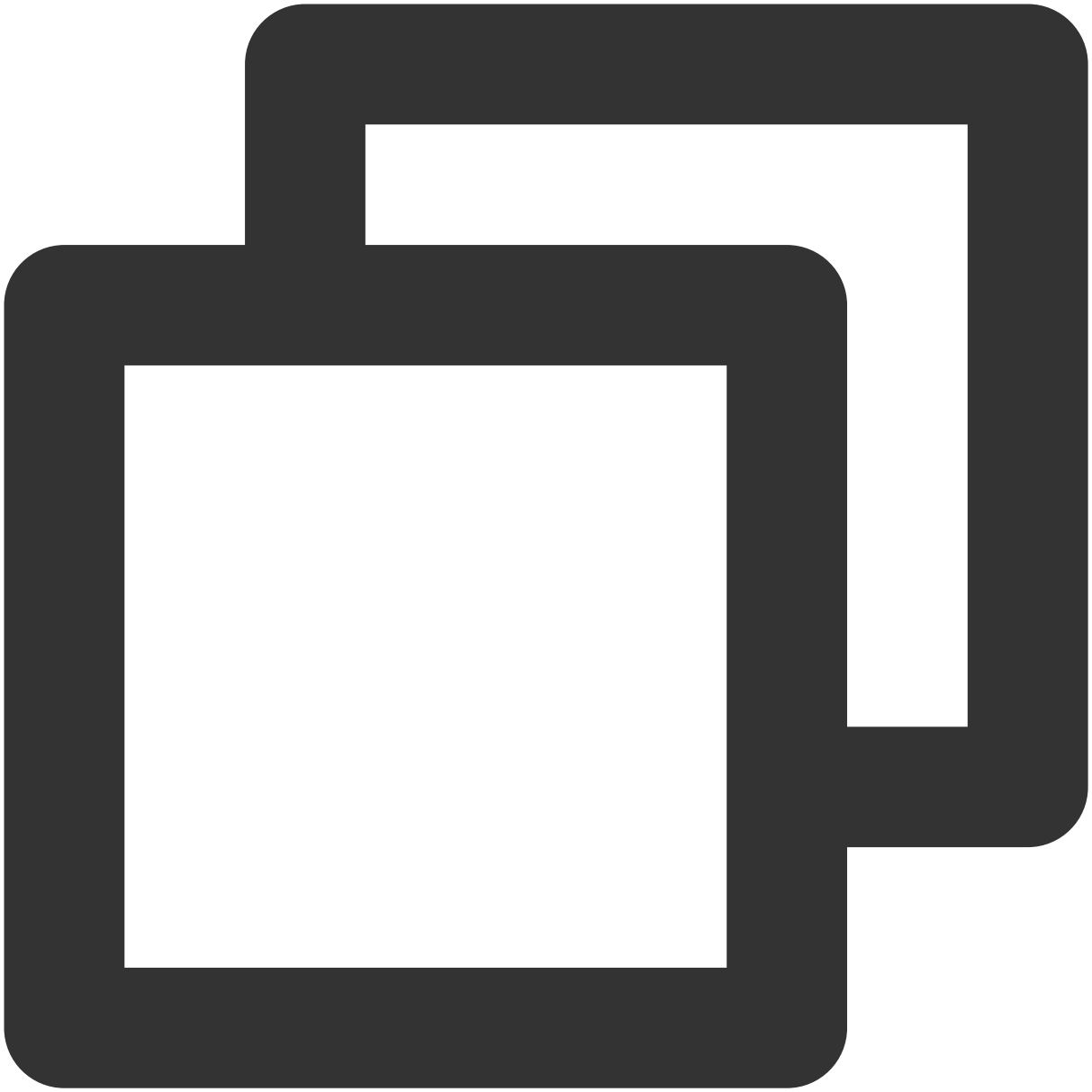
示例：



```
> SELECT transform_keys(map_from_arrays(array(1, 2, 3), array(1, 2, 3)), (k, v) ->
 {2:1,3:2,4:3}
> SELECT transform_keys(map_from_arrays(array(1, 2, 3), array(1, 2, 3)), (k, v) ->
 {2:1,4:2,6:3}
```

TRANSFORM_VALUES

函数语法：



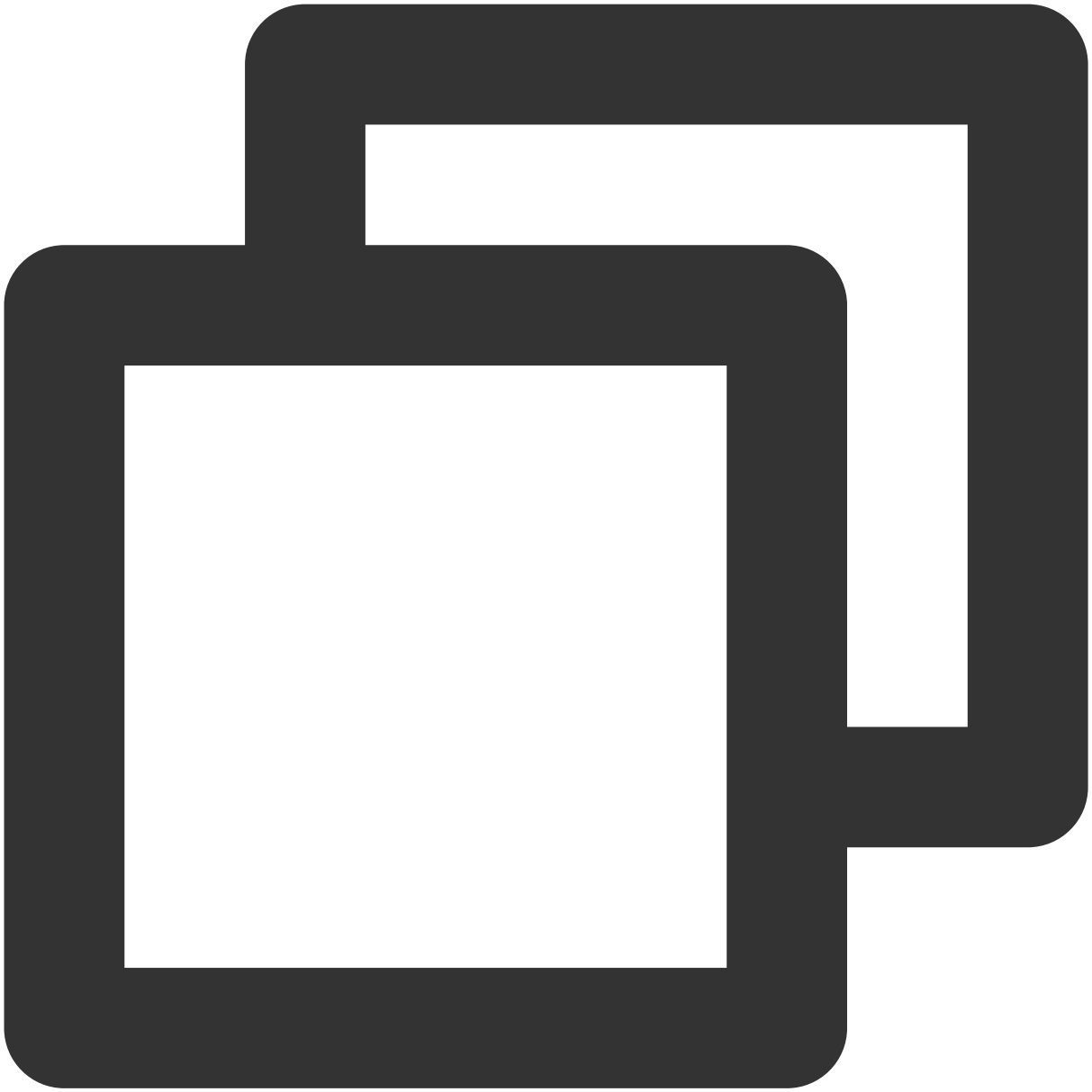
```
TRANSFORM_VALUES(<m> map<K, V>, <func> function(K, V)->R)
```

支持引擎：SparkSQL。

使用说明：使用 func 变换 map 中的 values。

返回类型：map<K, R>。

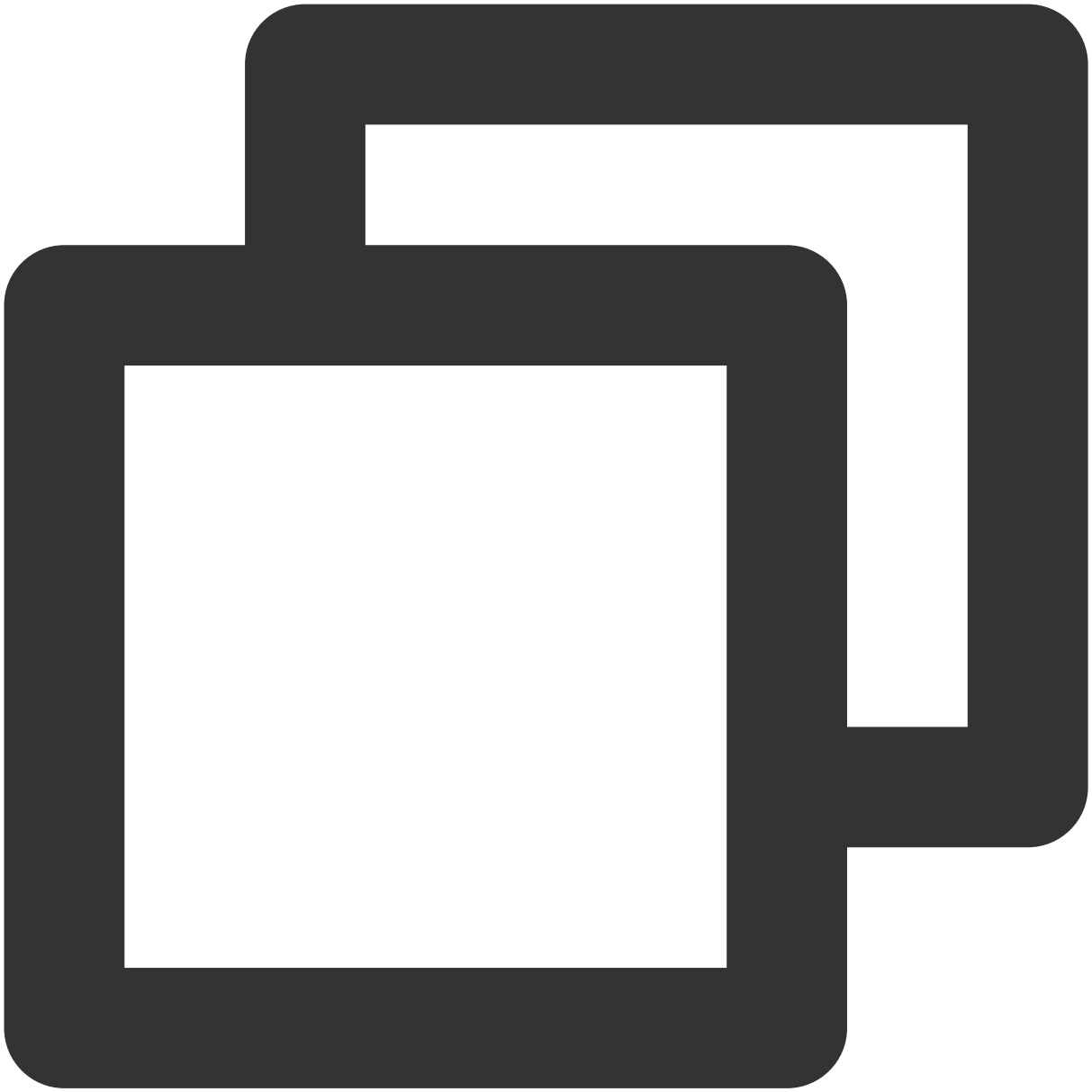
示例：



```
> SELECT transform_values(map_from_arrays(array(1, 2, 3), array(1, 2, 3)), (k, v) -
 {1:2,2:3,3:4}
> SELECT transform_values(map_from_arrays(array(1, 2, 3), array(1, 2, 3)), (k, v) -
 {1:2,2:4,3:6}
```

SIZE

函数语法：



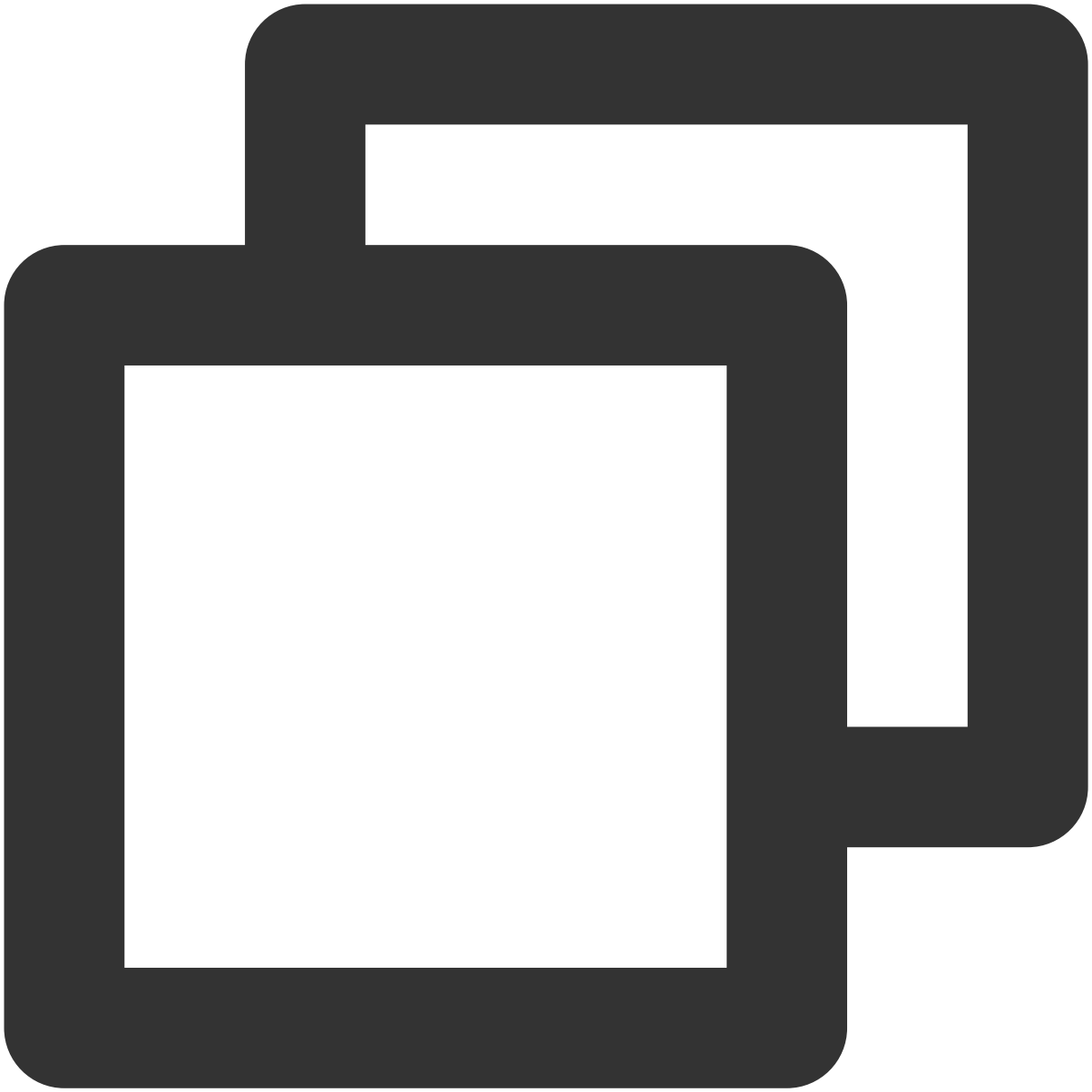
```
SIZE(<expr> array<T>|map<K, V>)
```

支持引擎：SparkSQL、Presto。

使用说明：返回数组或映射的大小。

返回类型：integer。

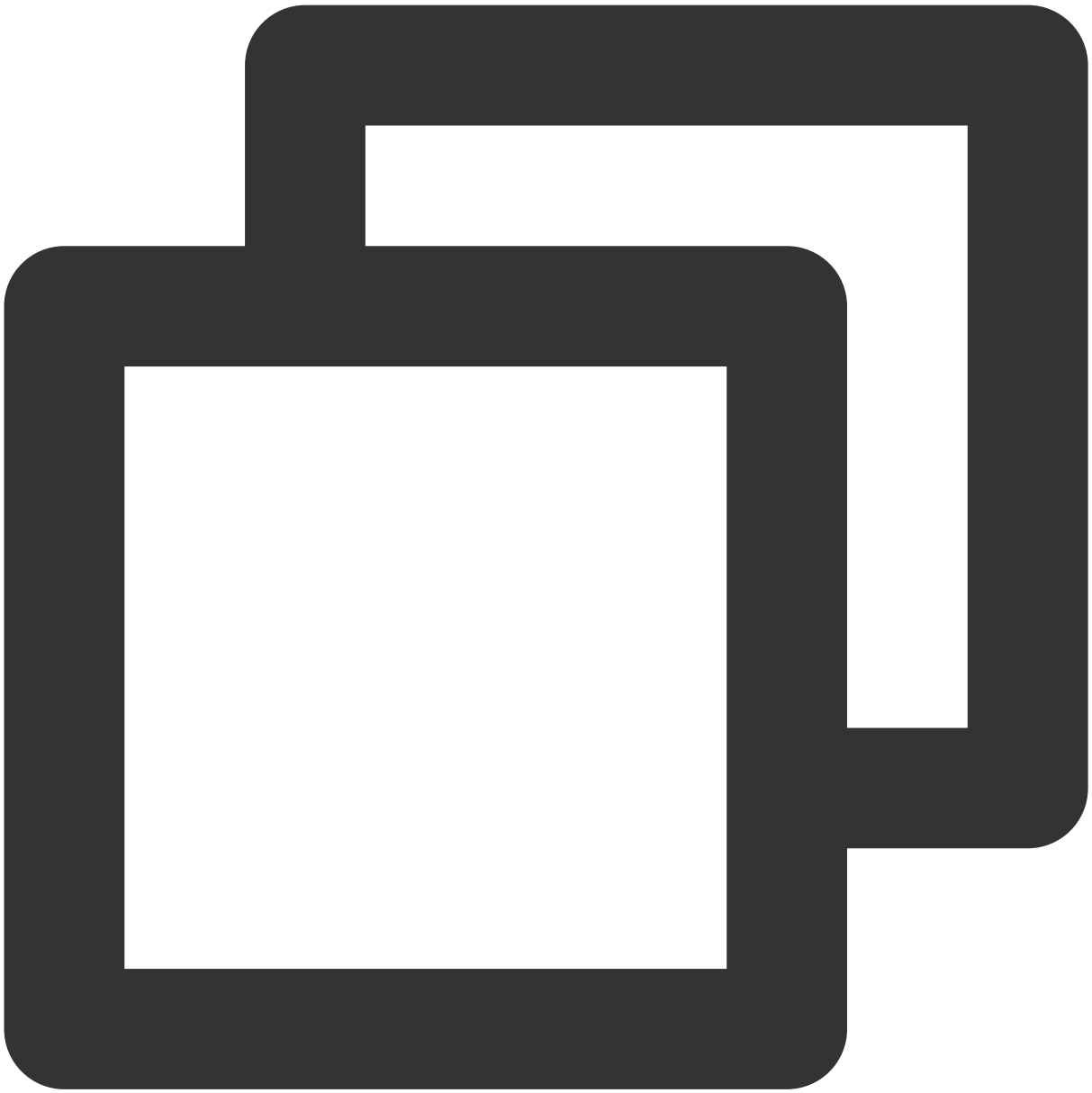
示例：



```
> SELECT size(array('b', 'd', 'c', 'a'));
 4
> SELECT size(map('a', 1, 'b', 2));
 2
> SELECT size(NULL);
 -1
```

CARDINALITY

函数语法：



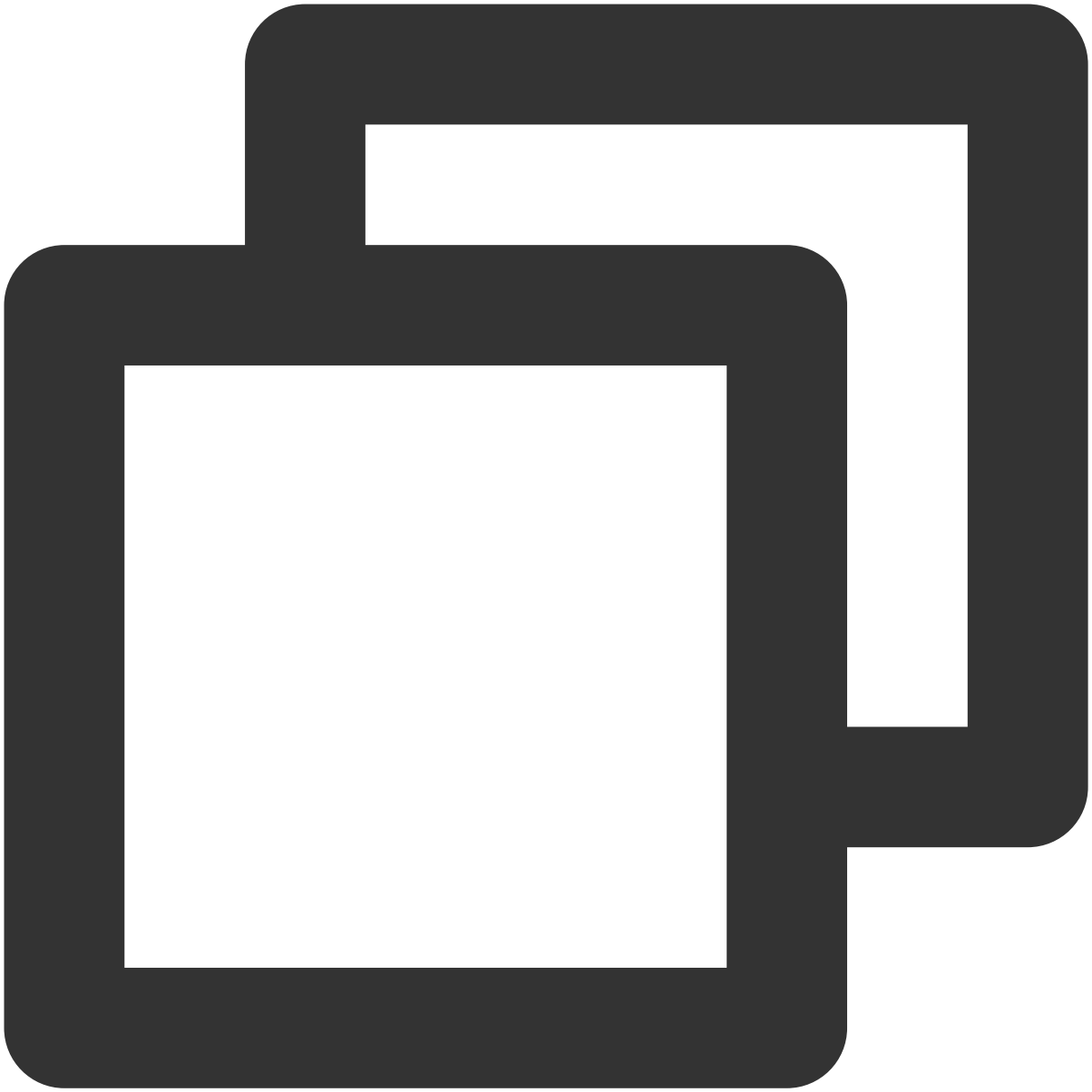
```
CARDINALITY(<expr> array<T>|map<K, V>)
```

支持引擎：SparkSQL、Presto。

使用说明：返回数组或映射的大小。

返回类型：integer。

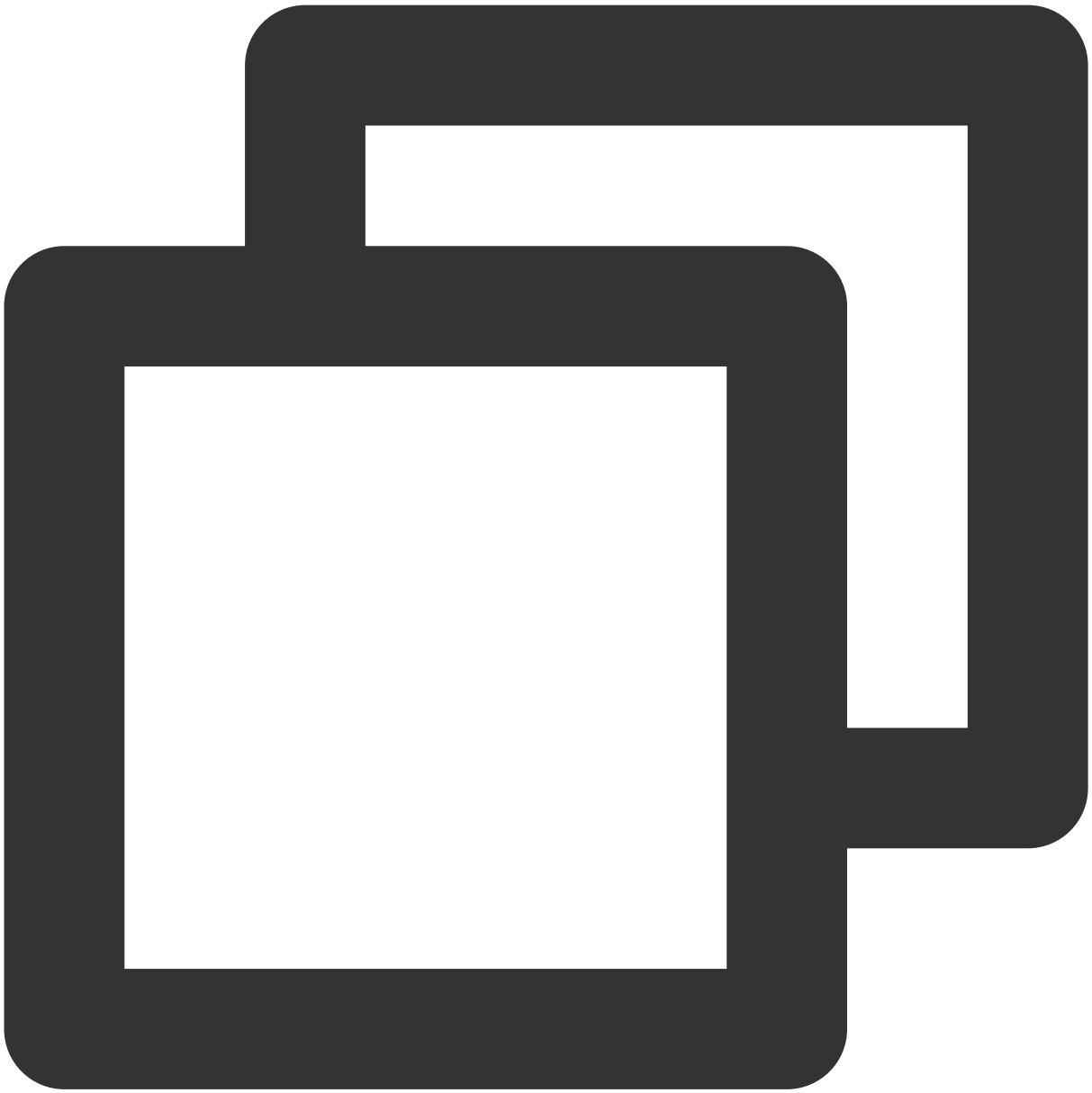
示例：



```
> SELECT cardinality(array('b', 'd', 'c', 'a'));
 4
> SELECT cardinality(map('a', 1, 'b', 2));
 2
> SELECT cardinality(NULL);
 -1
```

ANY_MATCH / FORALL

函数语法：



```
ANY_MATCH(<expr> array<T>, x -> lambda(x))
FORALL(<expr> array<T>, x -> lambda(x))
```

支持引擎：SparkSQL。

使用说明：对数组的每个元素，依次执行lambda表达式，只有1个返回true则函数返回true，否则返回false。

如果数组中有元素为NULL，则返回NULL。

返回类型：boolean

示例：

```
> SELECT any_match(array(1, 2, 3), x -> x % 2 == 0);
false
> SELECT any_match(array(2, 4, 8), x -> x % 2 == 0);
true
> SELECT any_match(array(1, null, 3), x -> x % 2 == 0);
NULL
```

日期时间函数

最近更新时间：2024-08-07 17:32:11

DATE

函数语法：

```
DATE(<expr> date|timestamp|string)
```

支持引擎：SparkSQL、Presto 。

使用说明：强制类型转换为 date 。

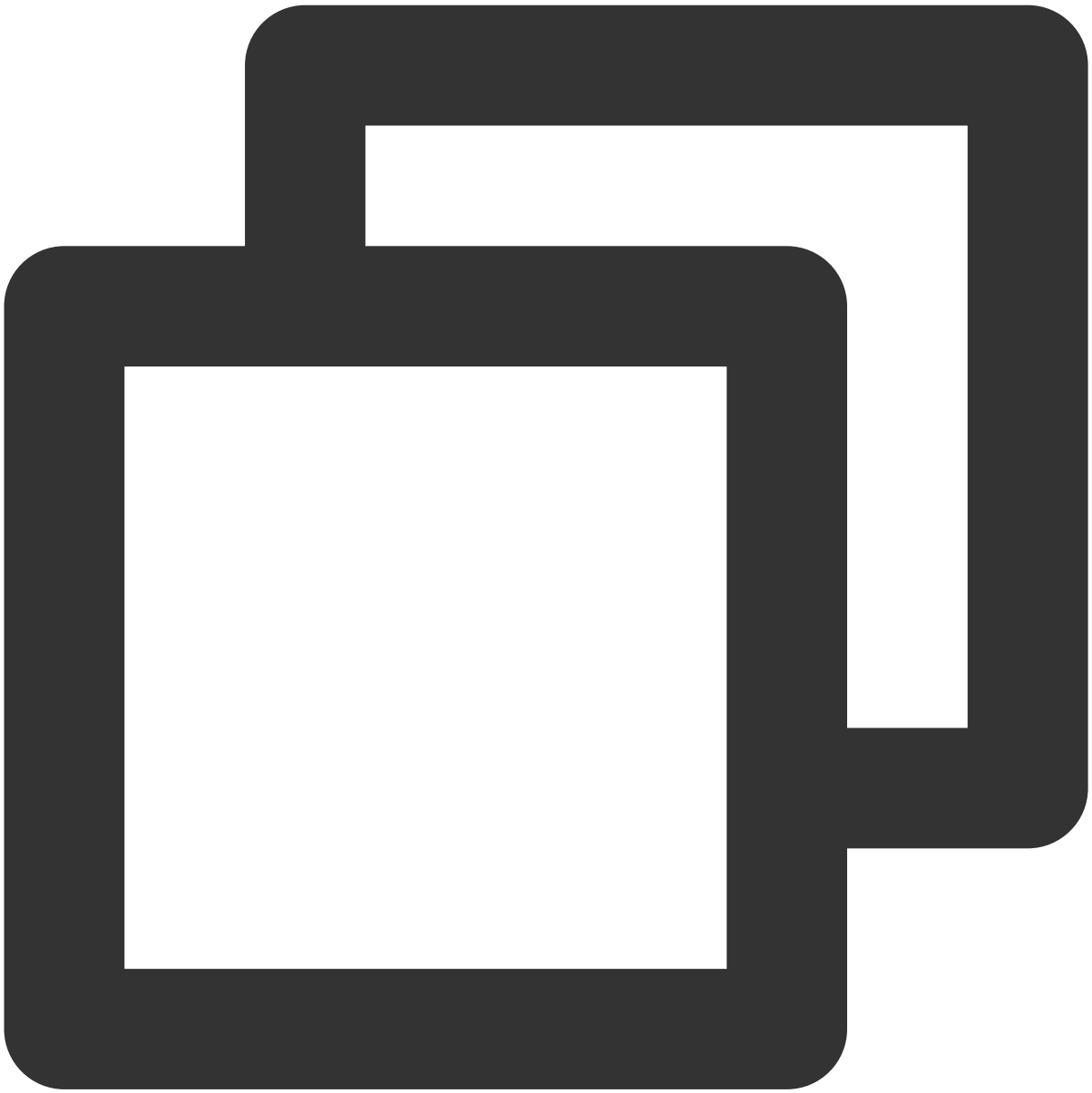
返回类型：date 。

示例：

```
> select date('2022-02-02');
 2022-02-02
```

TIMESTAMP

函数语法：



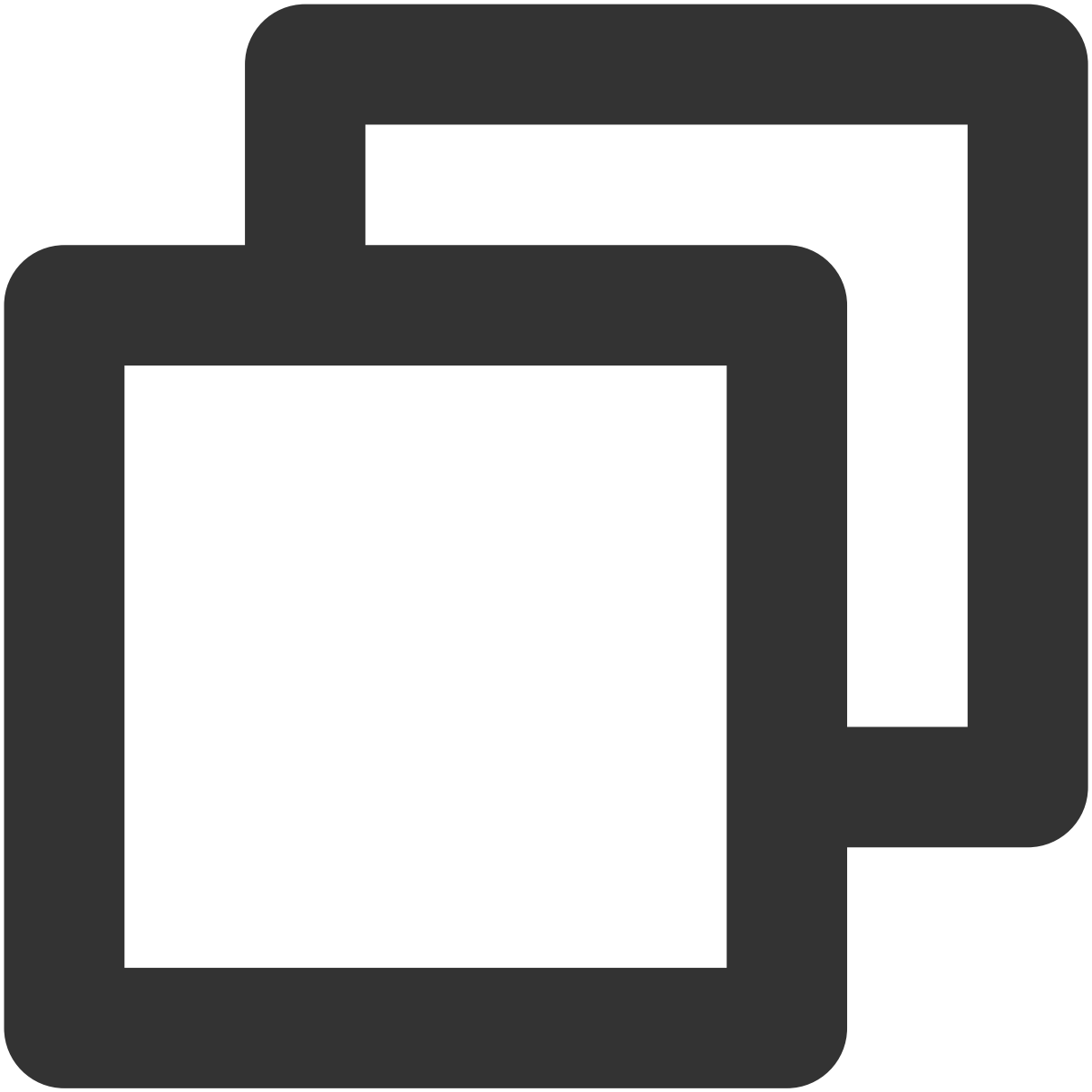
```
TIMESTAMP(<expr> date|timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：强制类型转换为 timestamp。

返回类型：timestamp。

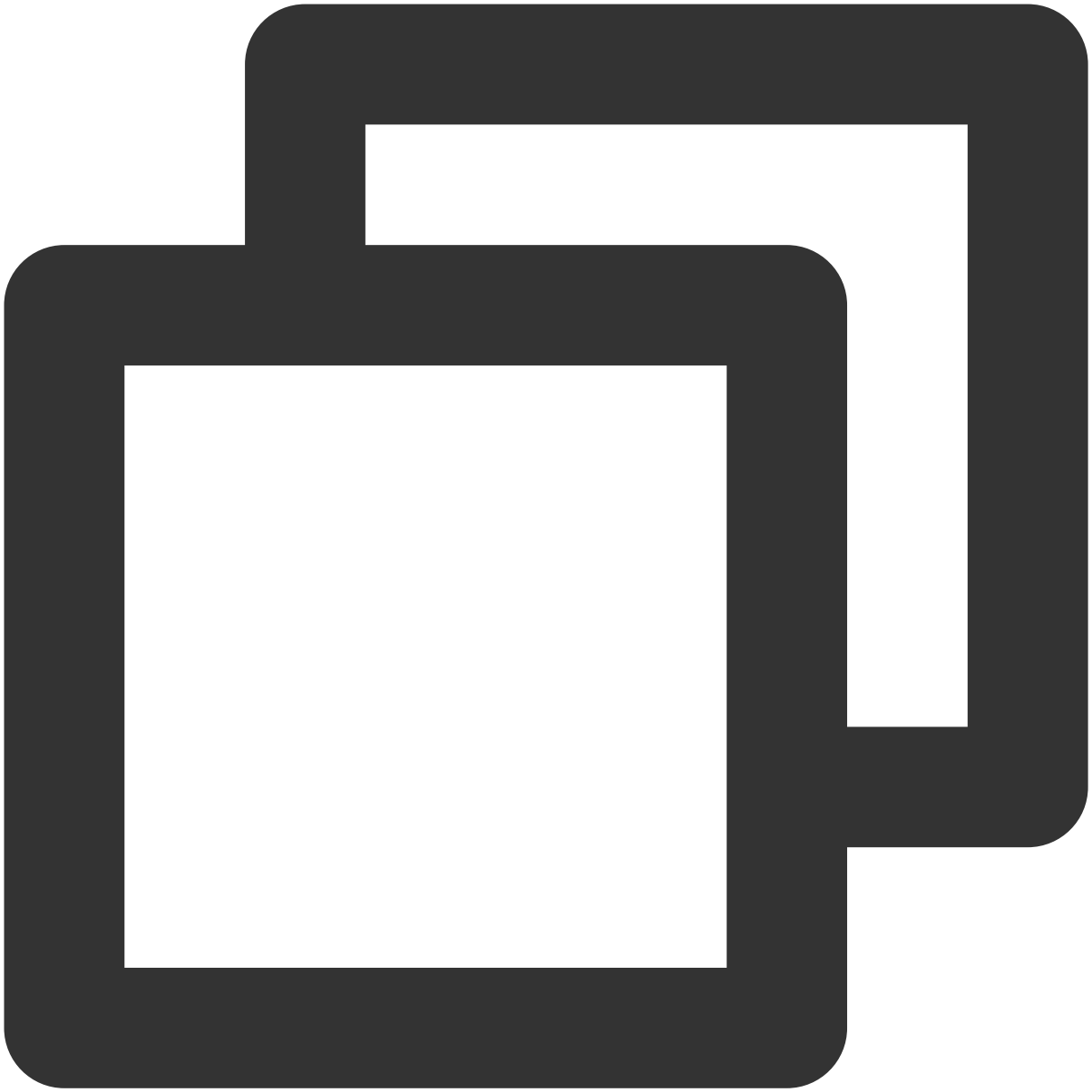
示例：



```
> select timestamp('2022-02-02 11:11:11');
 2022-02-02 11:11:11
```

ADD_MONTHS

函数语法：



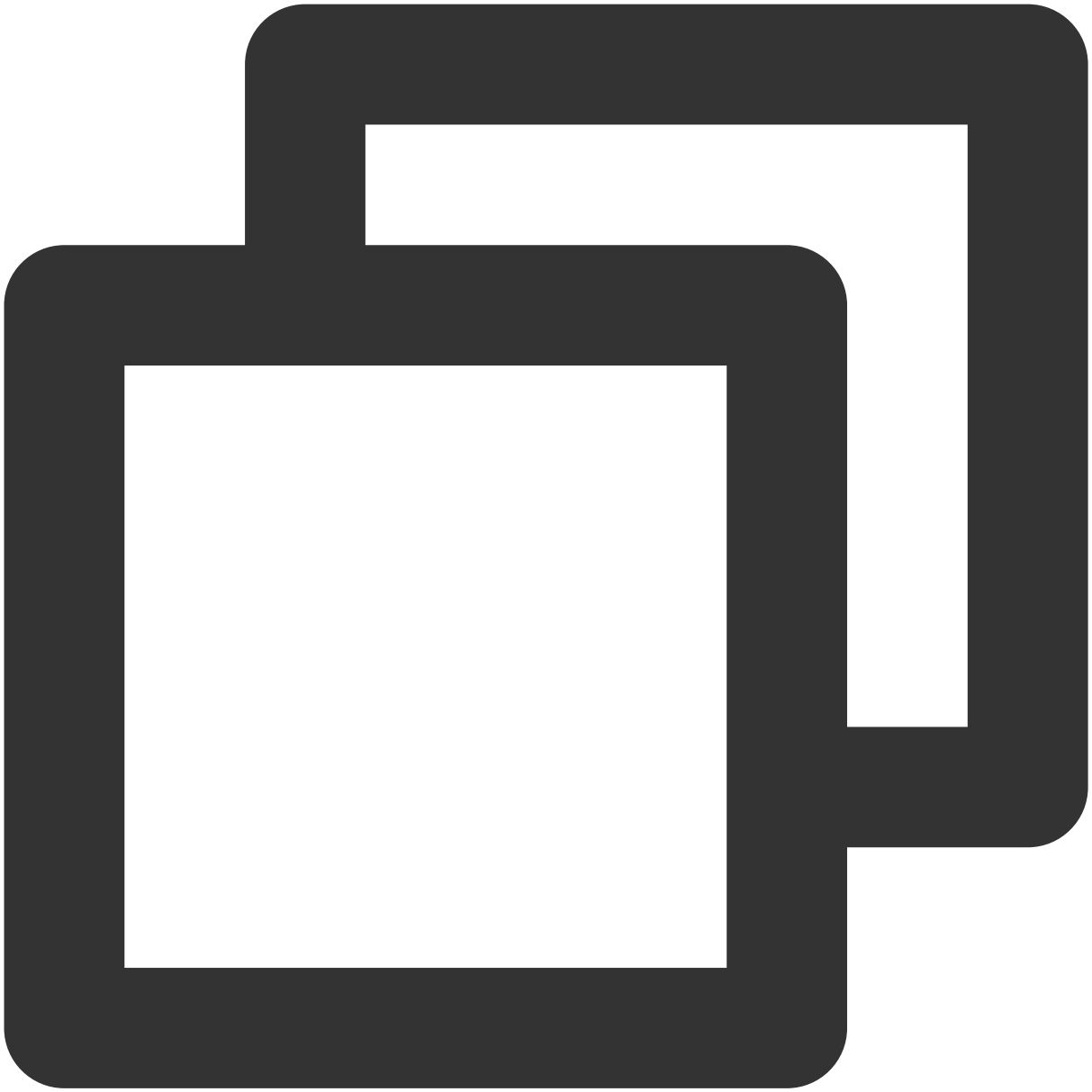
```
ADD_MONTHS(<start_date> date|timestamp|string, <num> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：返回 start_date 之后的 num 个月的日期。

返回类型：date。

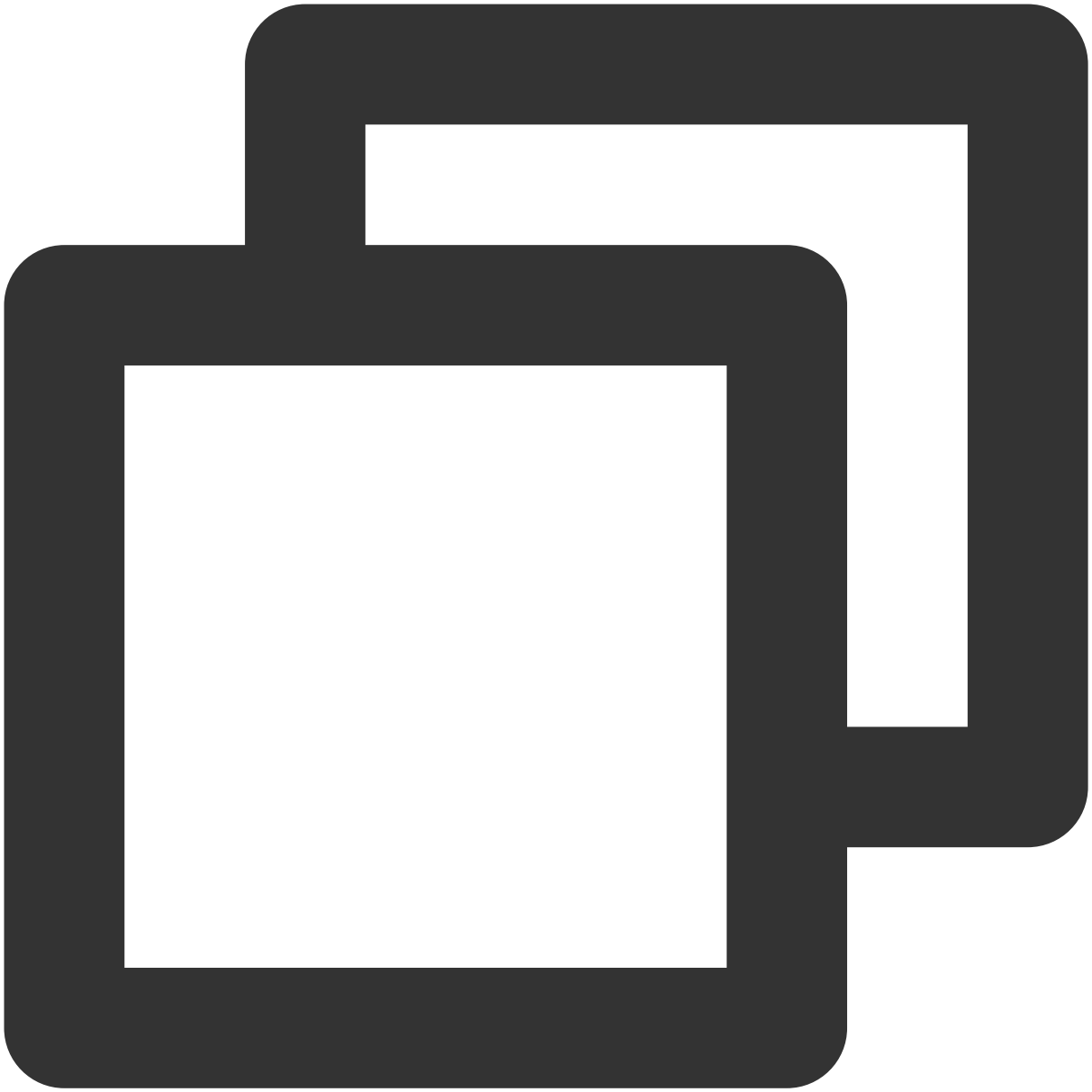
示例：



```
> SELECT add_months('2016-08-31', 1);
 2016-09-30
```

CURRENT_DATE

函数语法：



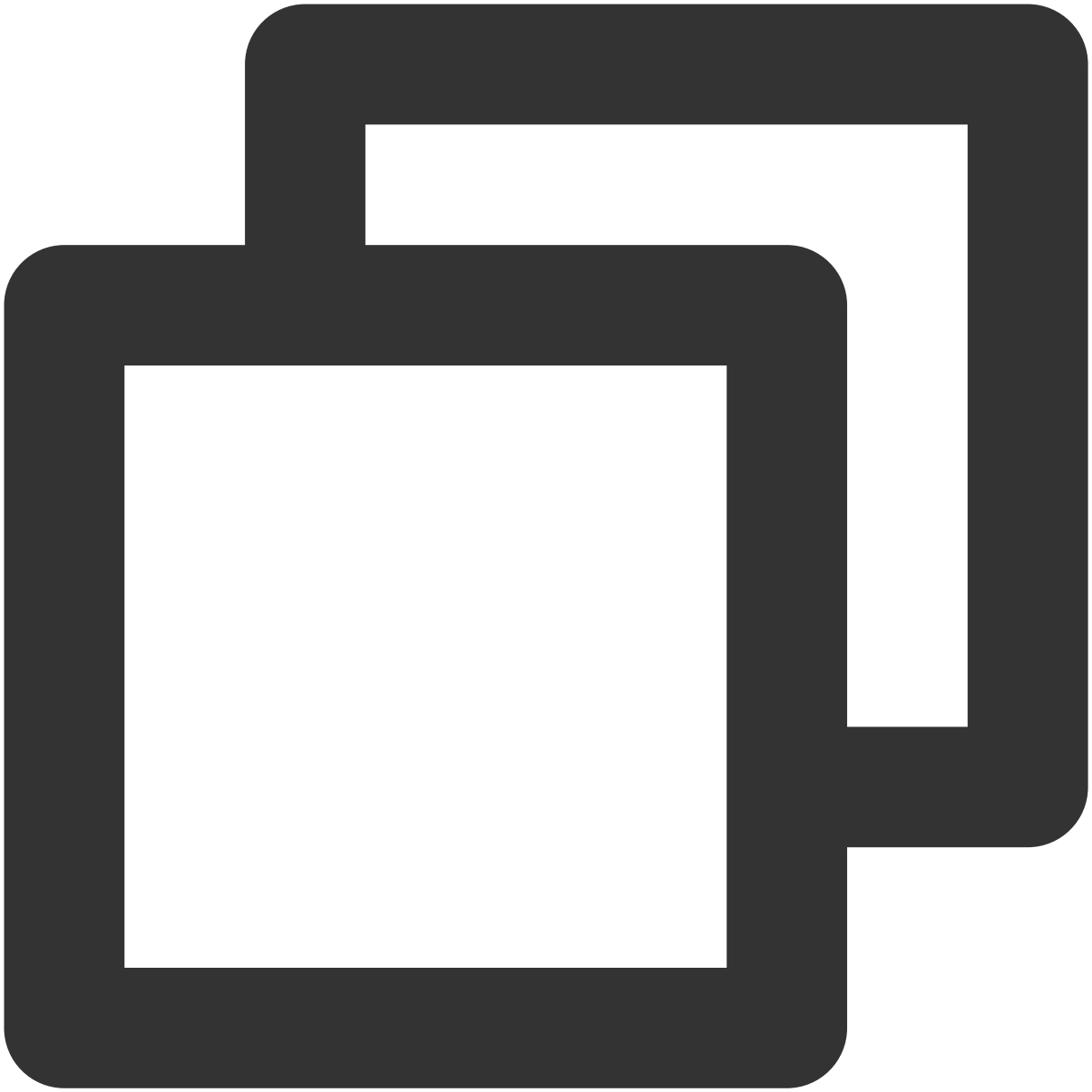
```
CURRENT_DATE
```

支持引擎：SparkSQL、Presto。

使用说明：返回查询计算开始时的当前日期。

返回类型：date。

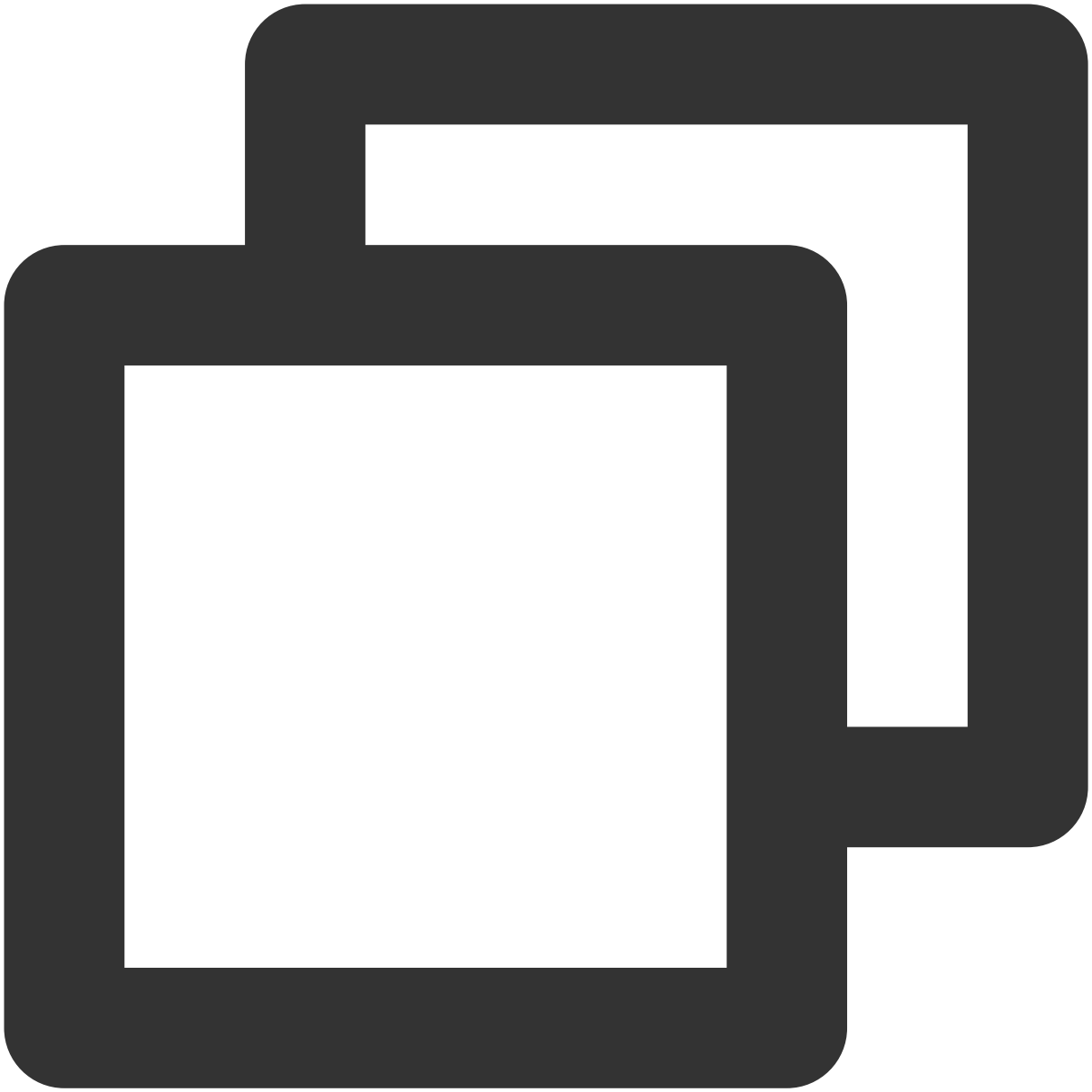
示例：



```
> SELECT CURRENT_DATE;
 2022-07-27
```

CURRENT_TIMESTAMP

函数语法：



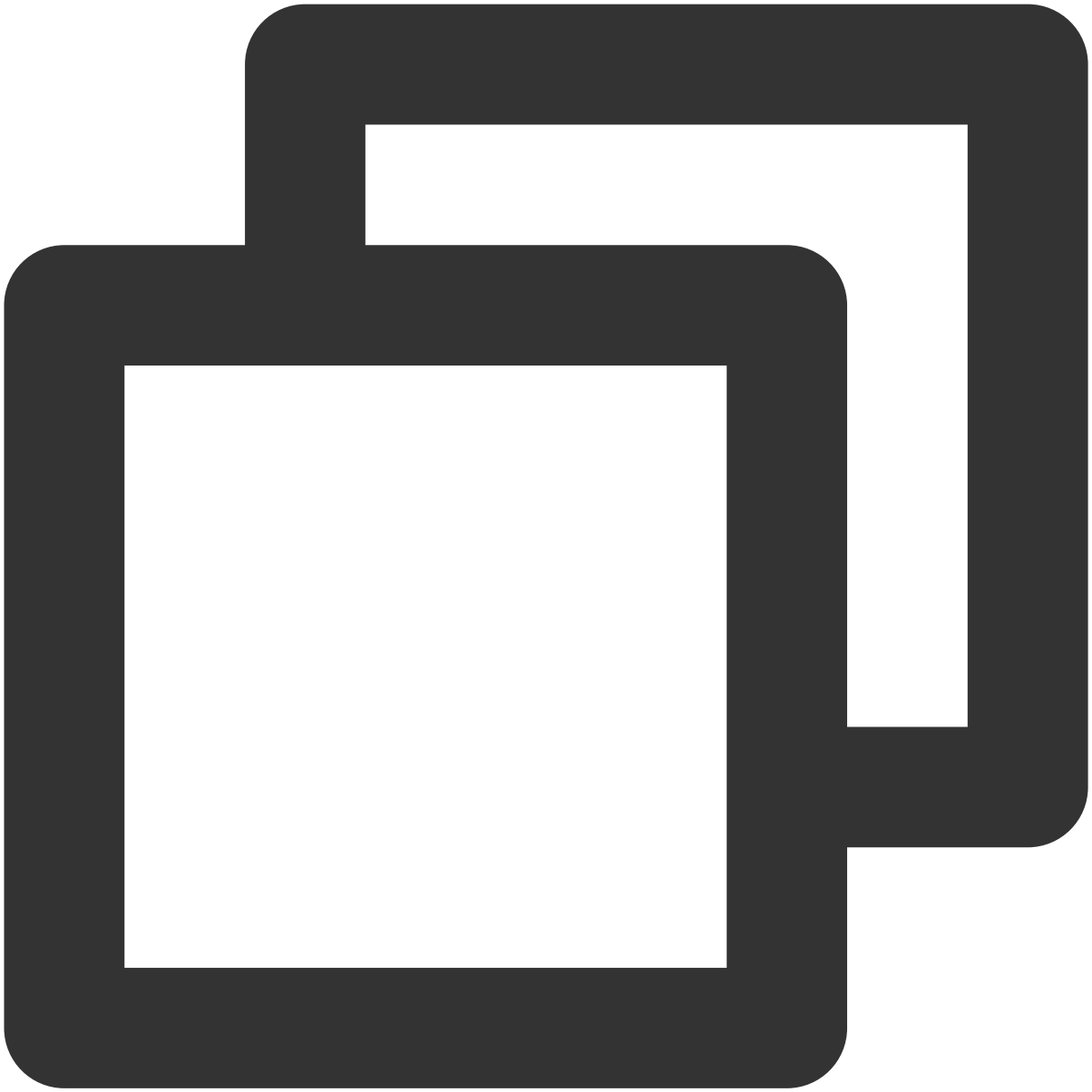
```
CURRENT_TIMESTAMP
```

支持引擎：SparkSQL、Presto。

使用说明：返回查询计算开始时的当前时间戳。

返回类型：timestamp。

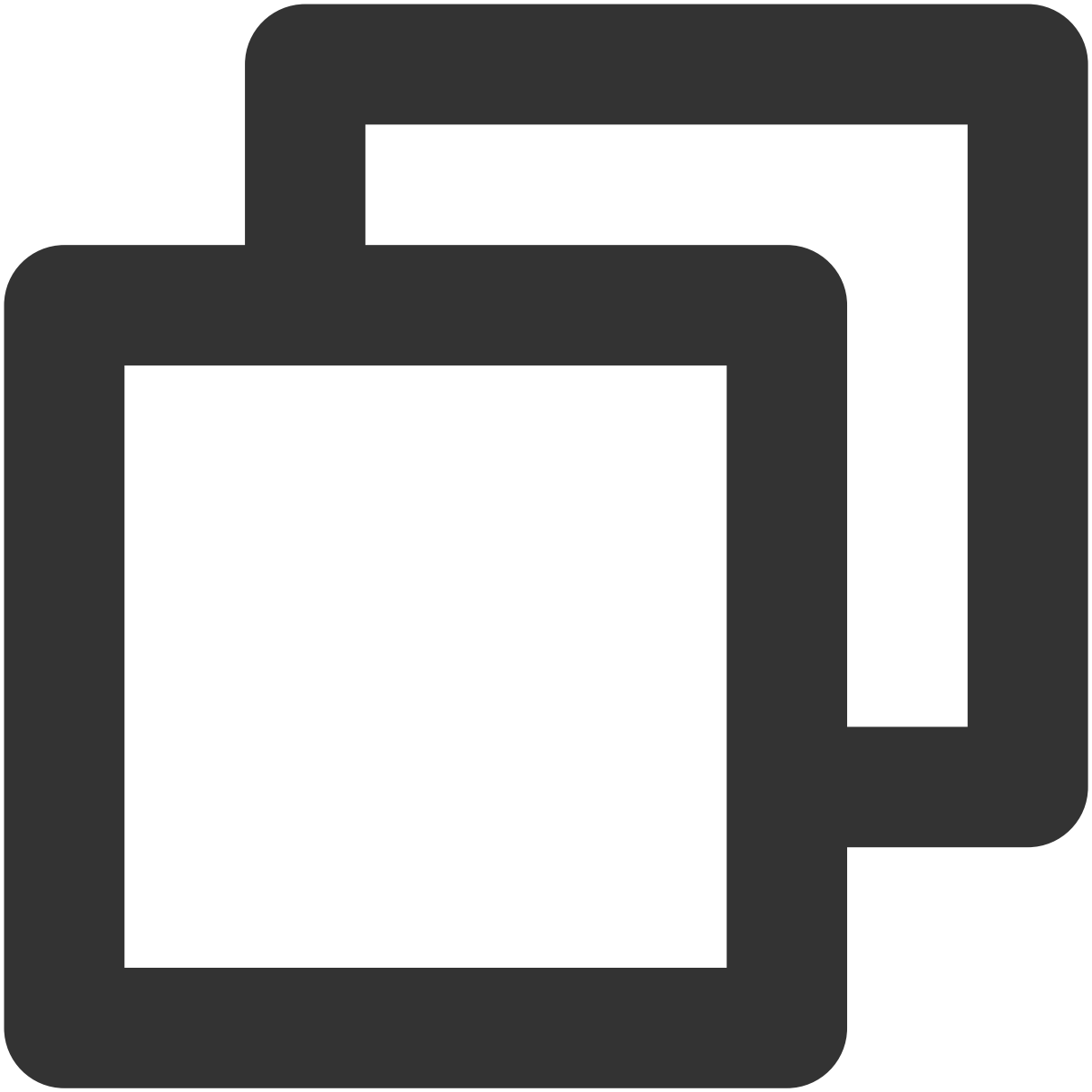
示例：



```
> SELECT CURRENT_TIMESTAMP;
 2022-07-27 18:06:00.632
```

CURRENT_TIMEZONE

函数语法：



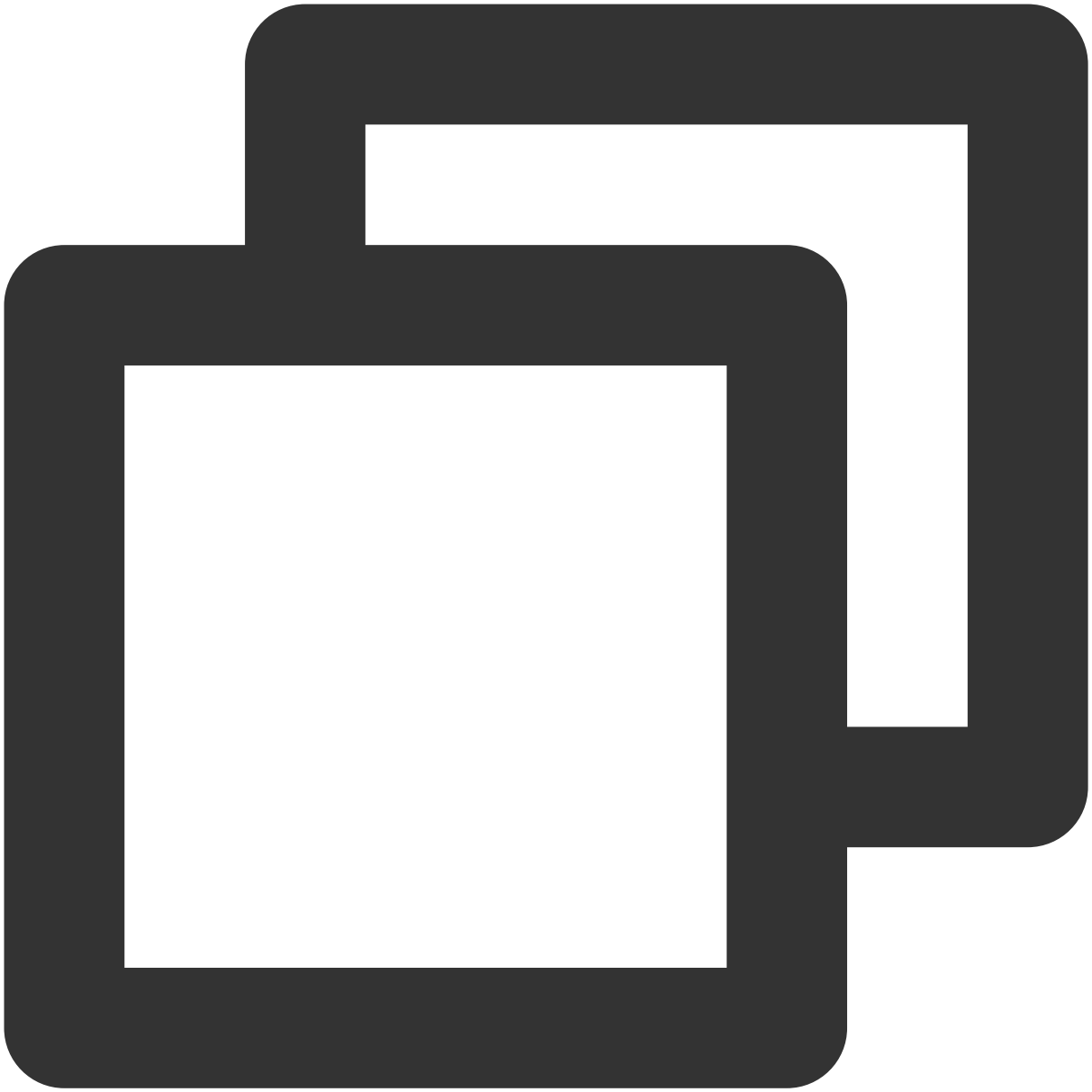
```
CURRENT_TIMEZONE(
```

支持引擎：SparkSQL、Presto。

使用说明：返回当前会话本地时区。

返回类型：string。

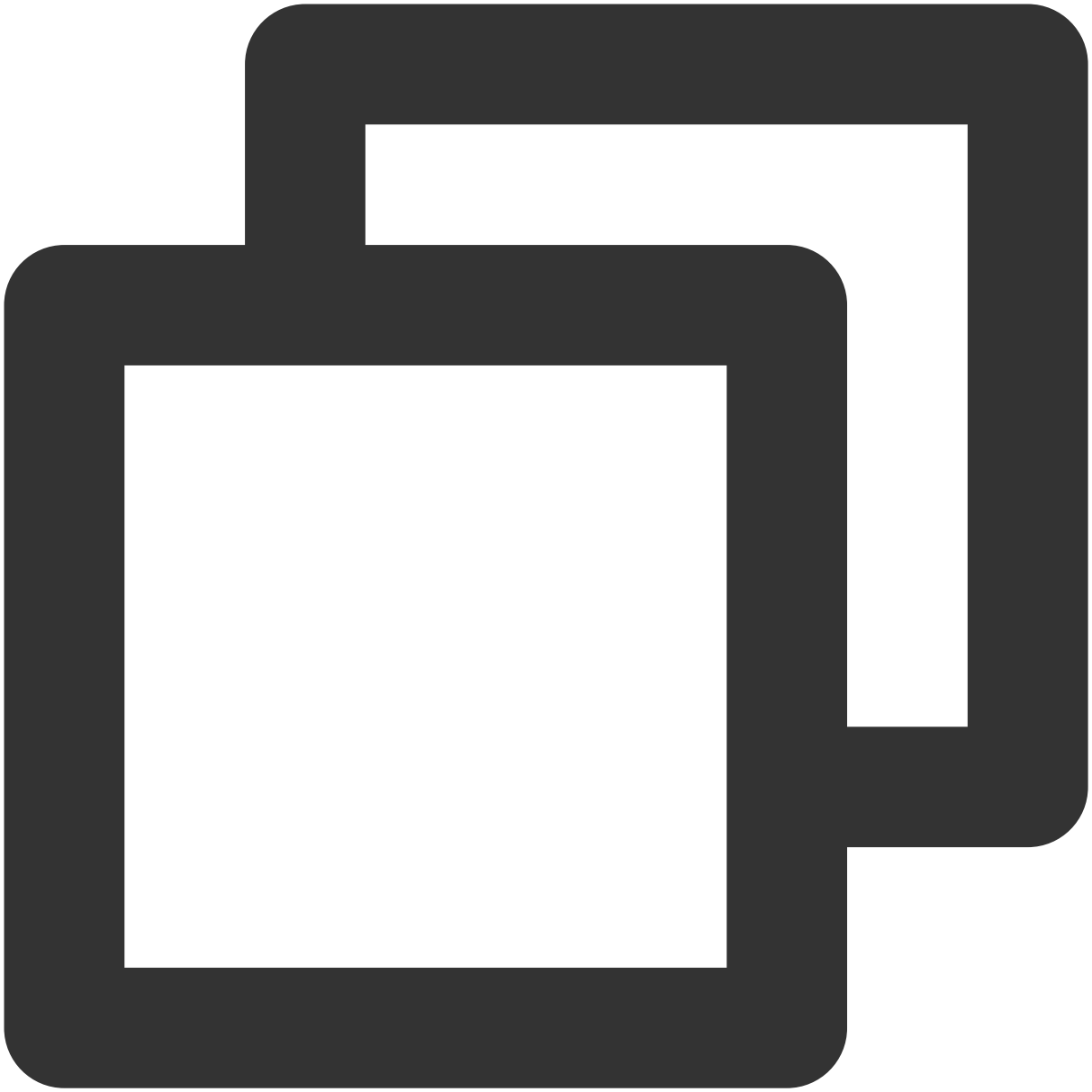
示例：



```
> select CURRENT_TIMEZONE();
Asia/Shanghai
```

DATEDIFF

函数语法：



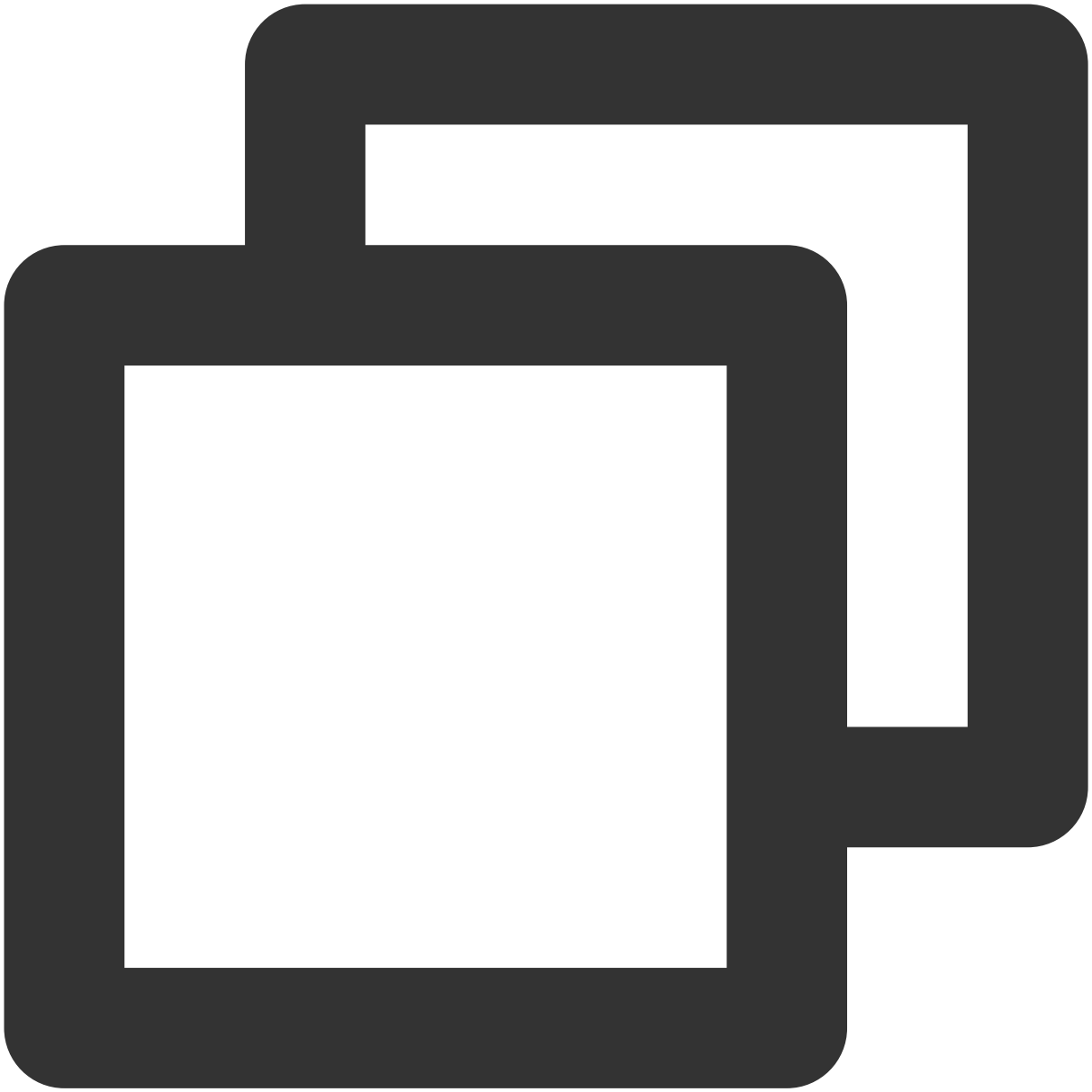
```
DATEDIFF(<end> date|timestamp|string, <start> date|timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回从 start 到 end 的天数。

返回类型：integer。

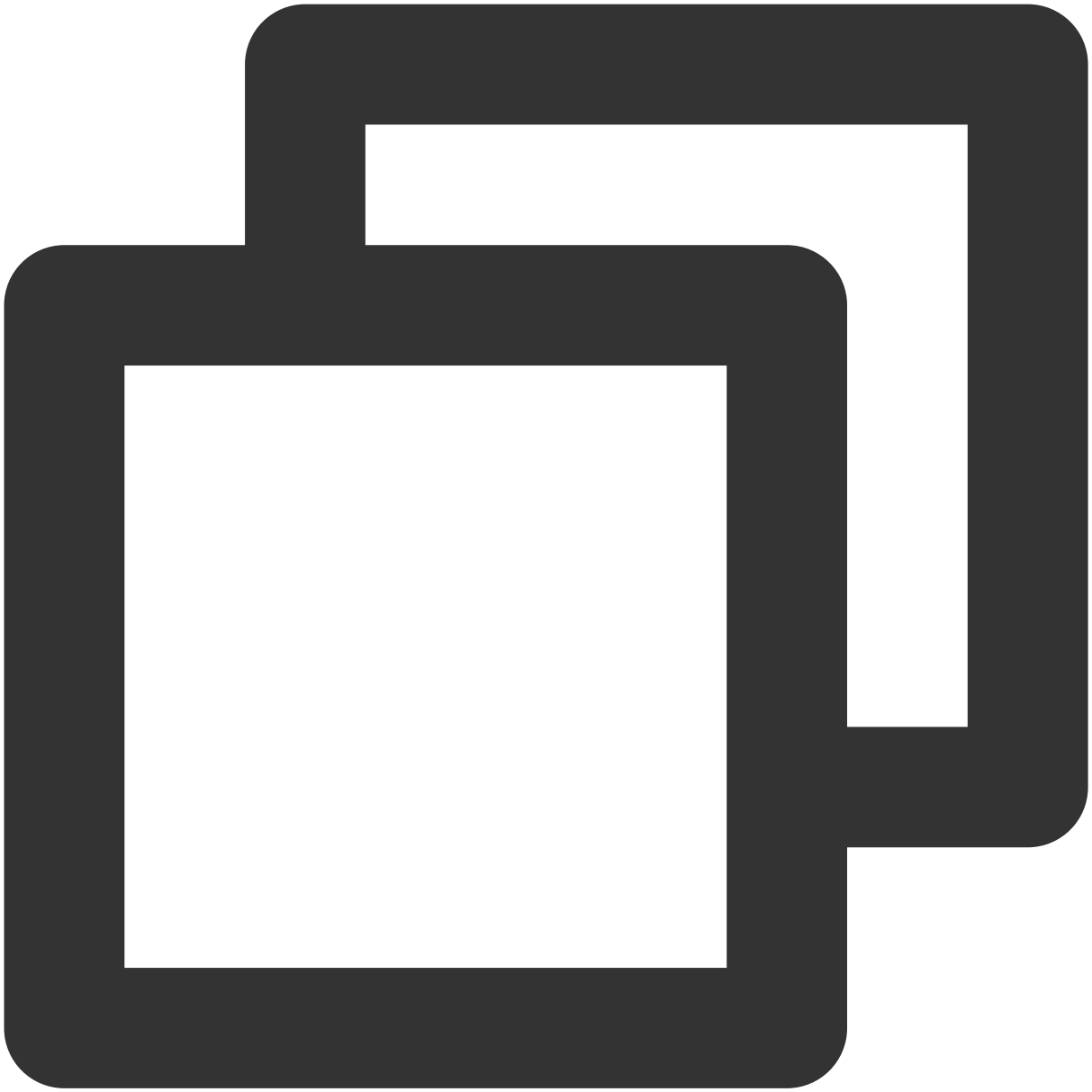
示例：



```
> SELECT datediff('2009-07-31', '2009-07-30');
 1
> SELECT datediff('2009-07-30', '2009-07-31');
 -1
```

DATE_ADD

函数语法：



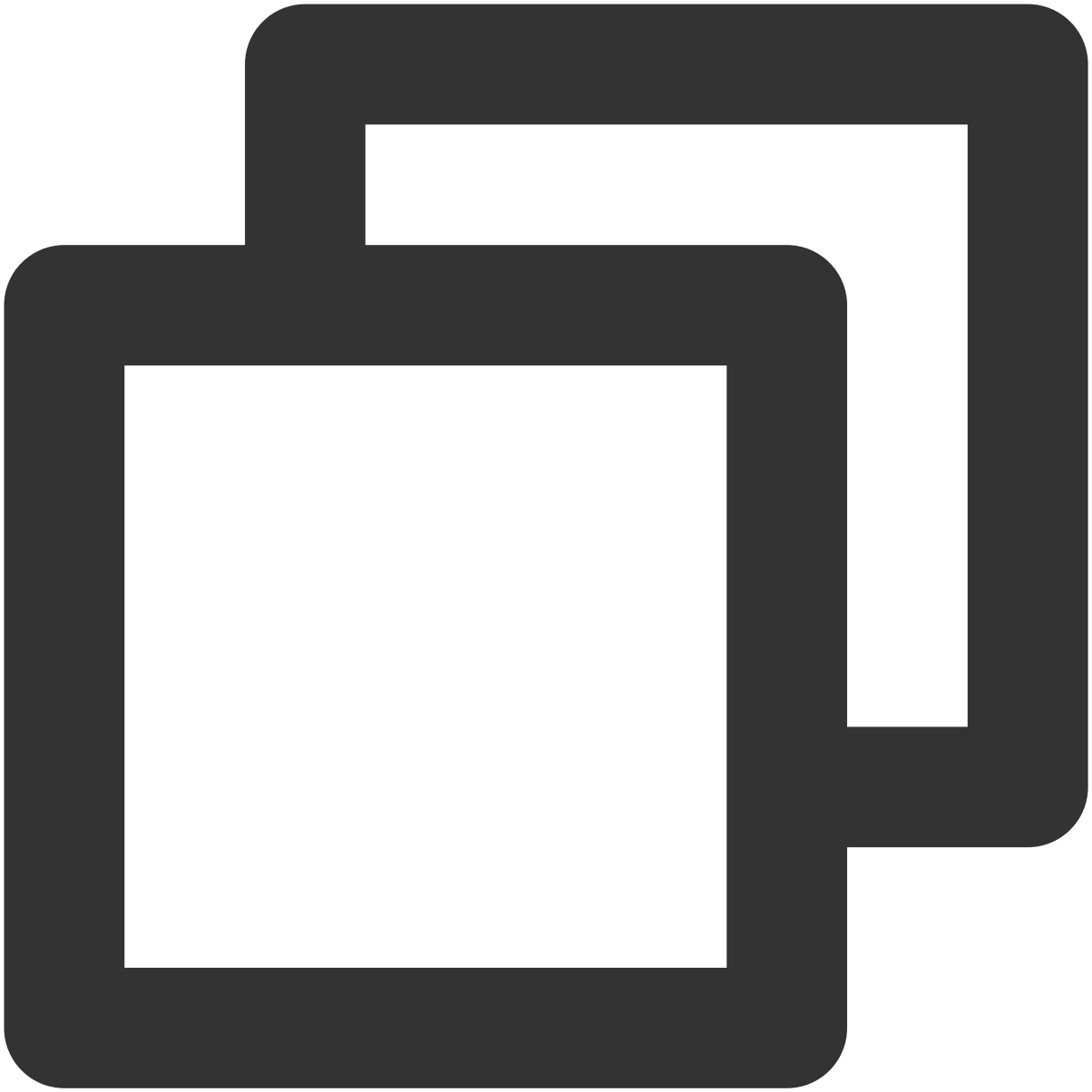
```
DATE_ADD(<start_dates> date|timestamp|string, <num> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：返回 start_date 后 num 天的日期。

返回类型：date。

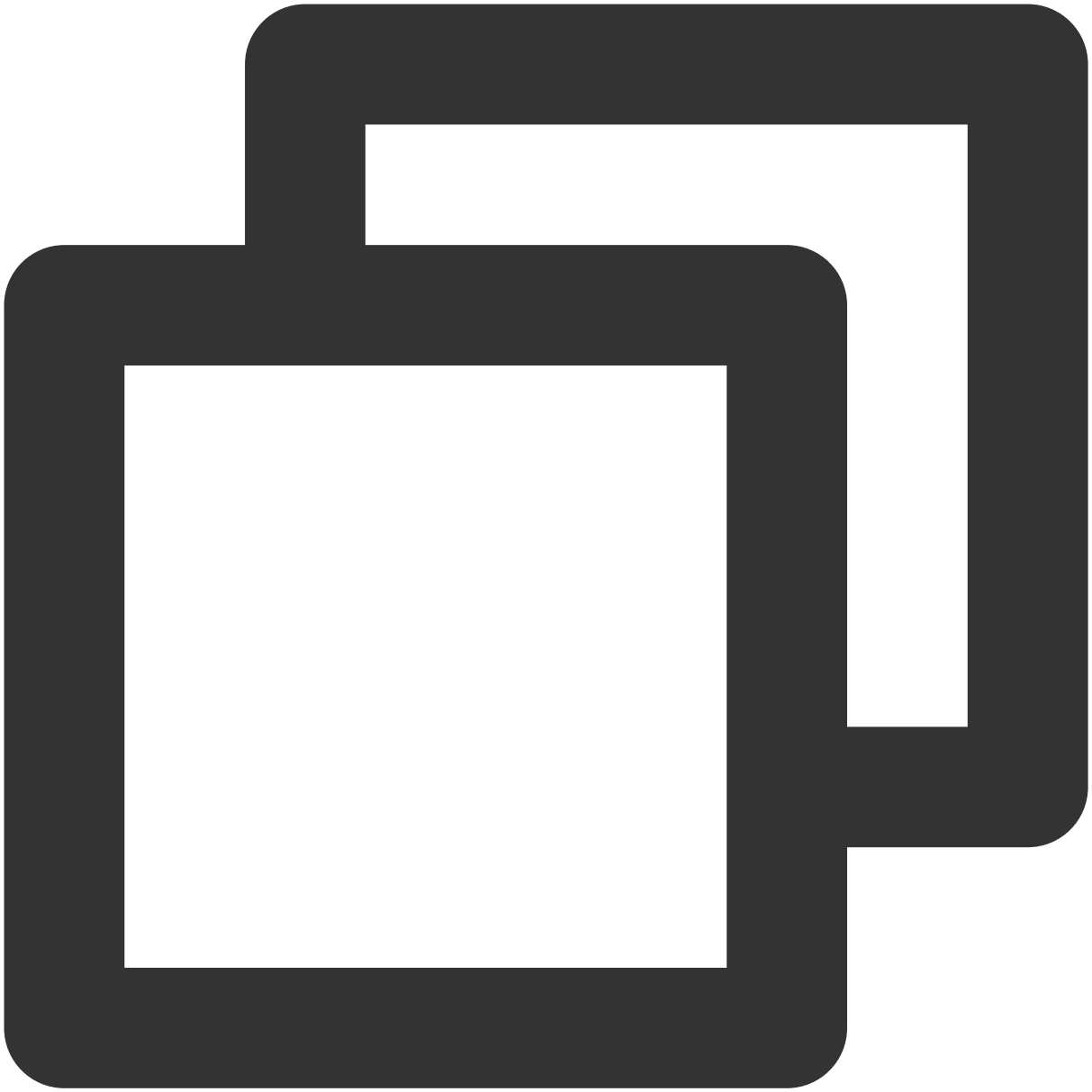
示例：



```
> SELECT date_add('2016-07-30', 1);
 2016-07-31
```

DATE_FORMAT

函数语法：



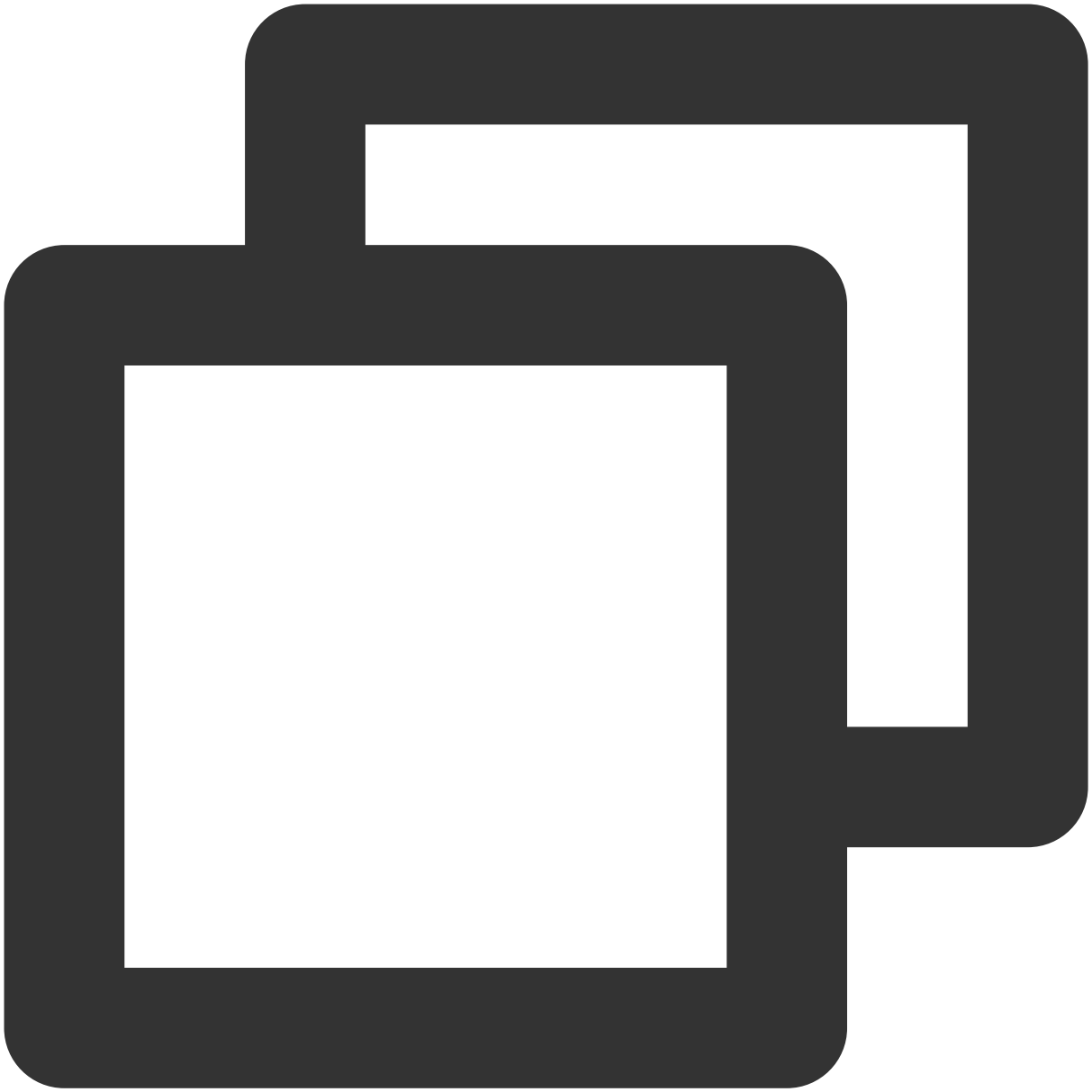
```
DATE_FORMAT(<ts> date|timestamp|string, <format> string)
```

支持引擎：SparkSQL、Presto。

使用说明：将时间戳转换指定日期格式 format 的字符串值。

返回类型：string。

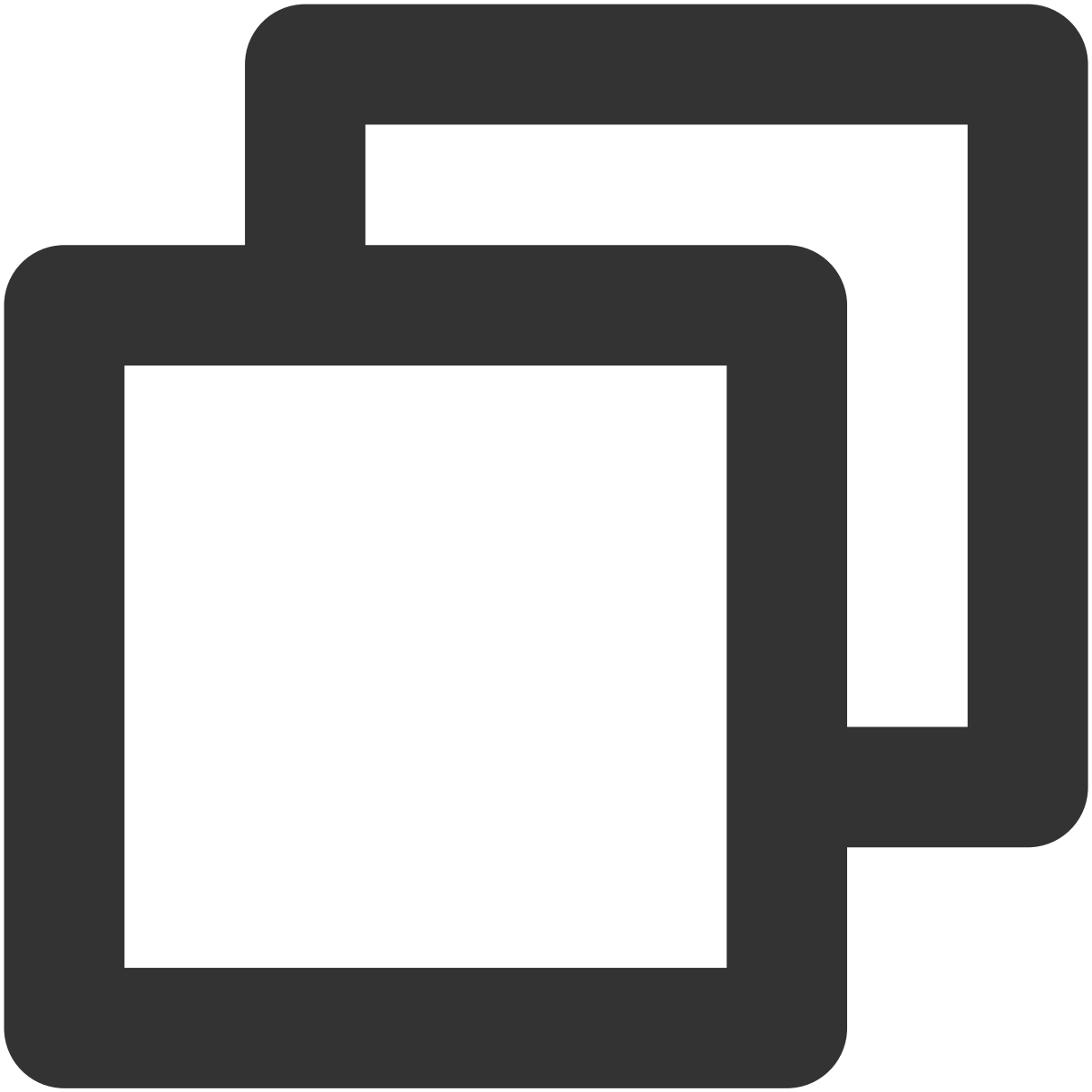
示例：



```
> SELECT date_format('2016-04-08', 'y');
 2016
```

DATE_SUB

函数语法：



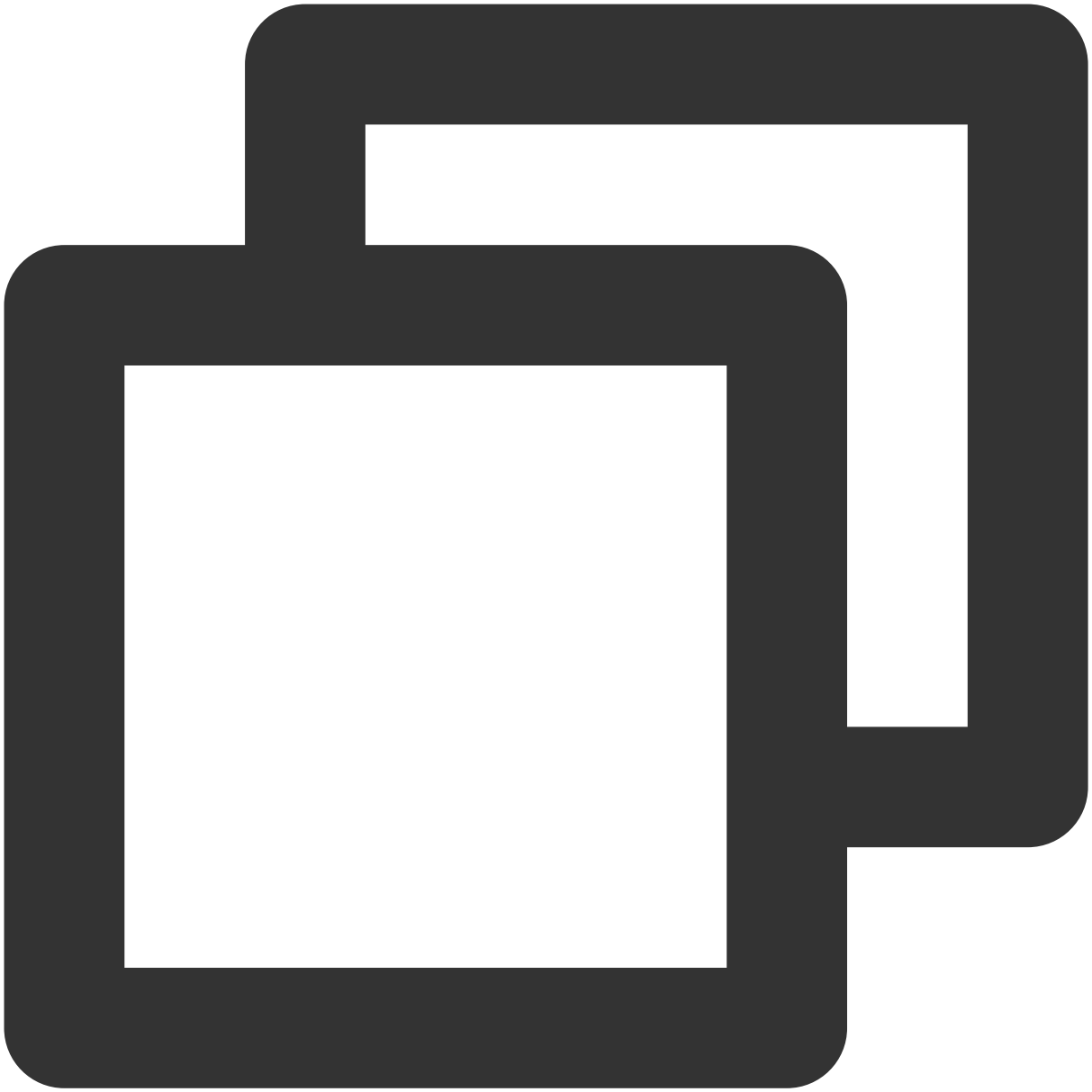
```
DATE_SUB(<start_date> date|timestamp|string, <num> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：返回 start_date 前 num 天的日期。

返回类型：date。

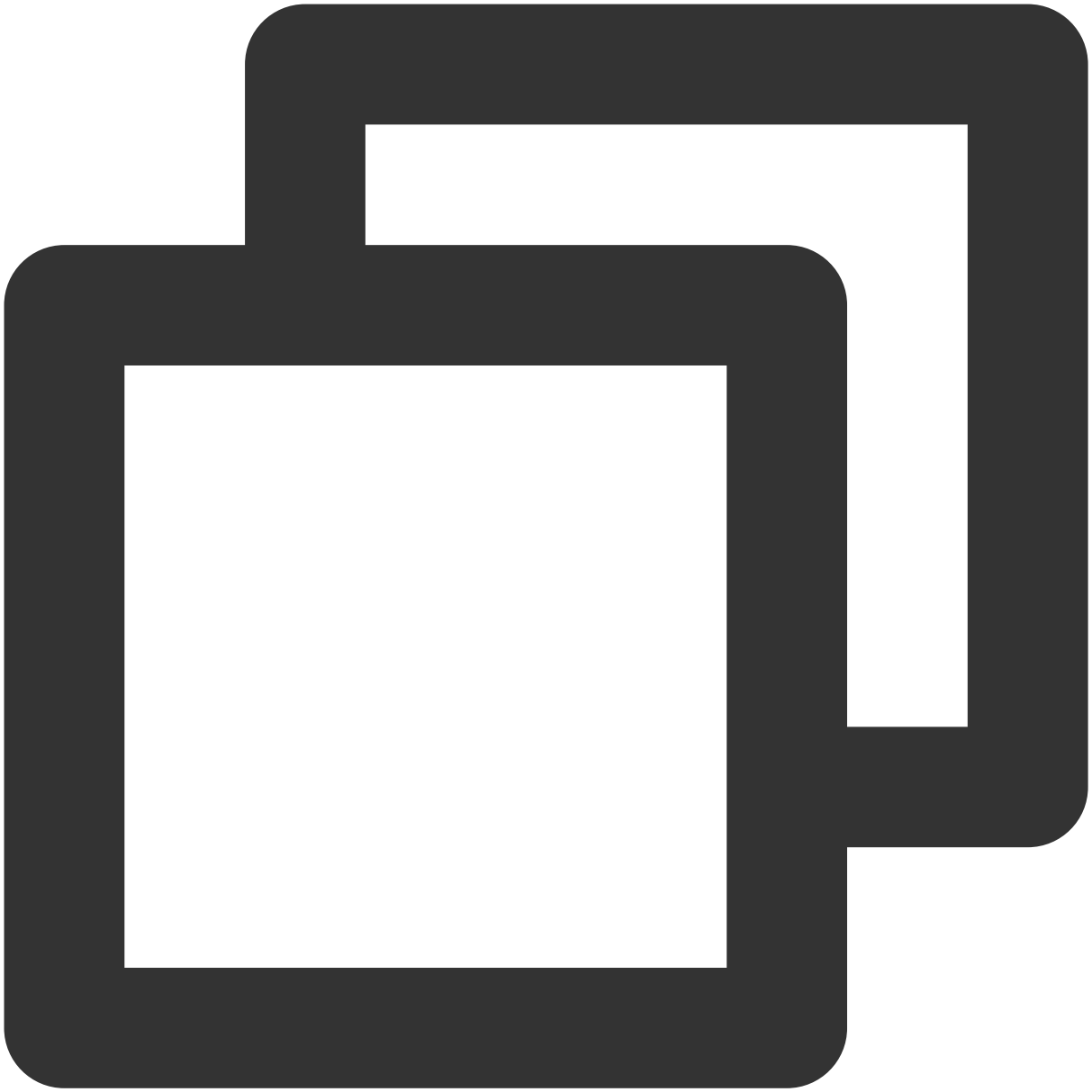
示例：



```
> SELECT date_sub('2016-07-30', 1);
 2016-07-29
```

DAY

函数语法：



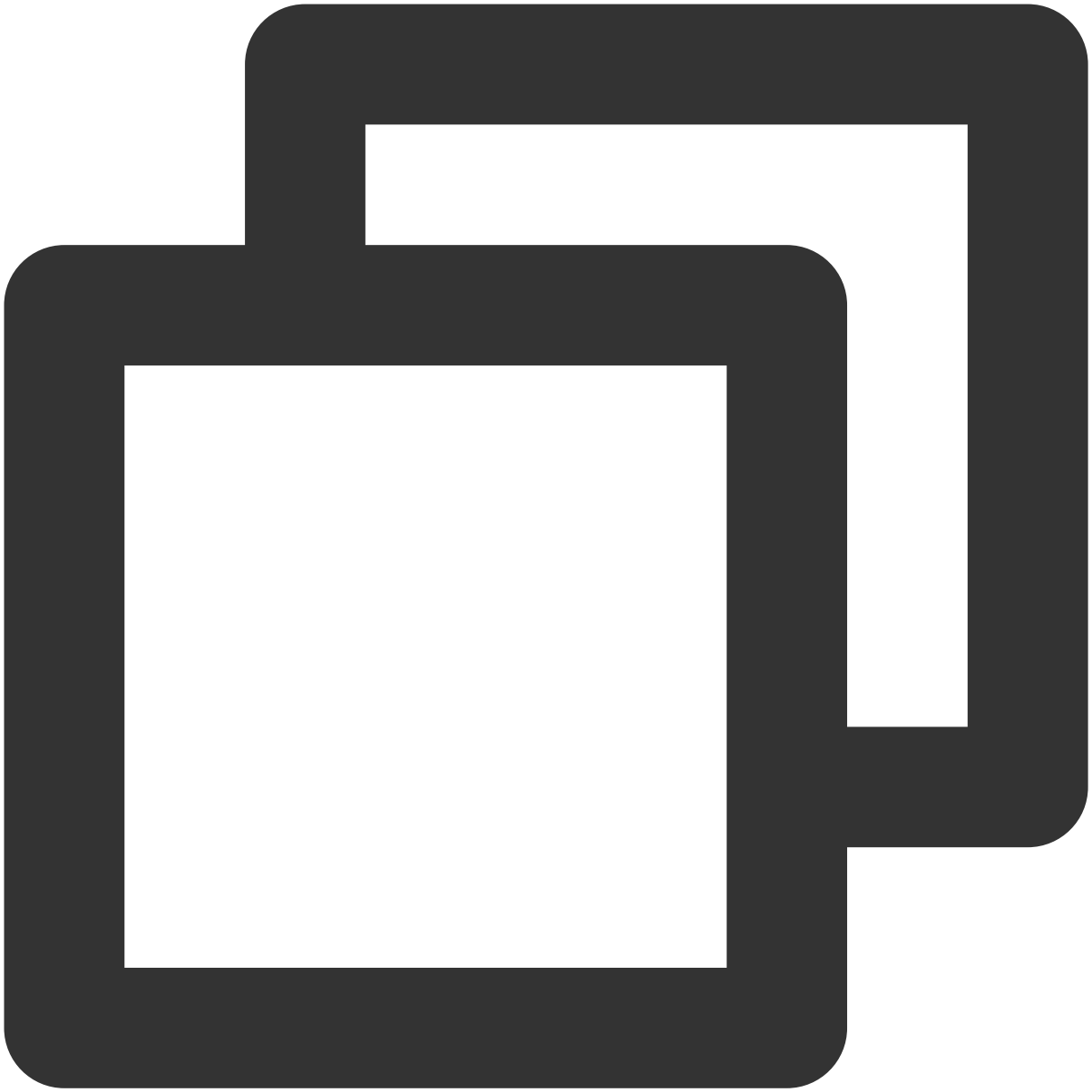
```
DAY(<d> date|timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：计算日期/时间戳 d 是这个月的第几天。

返回类型：integer。

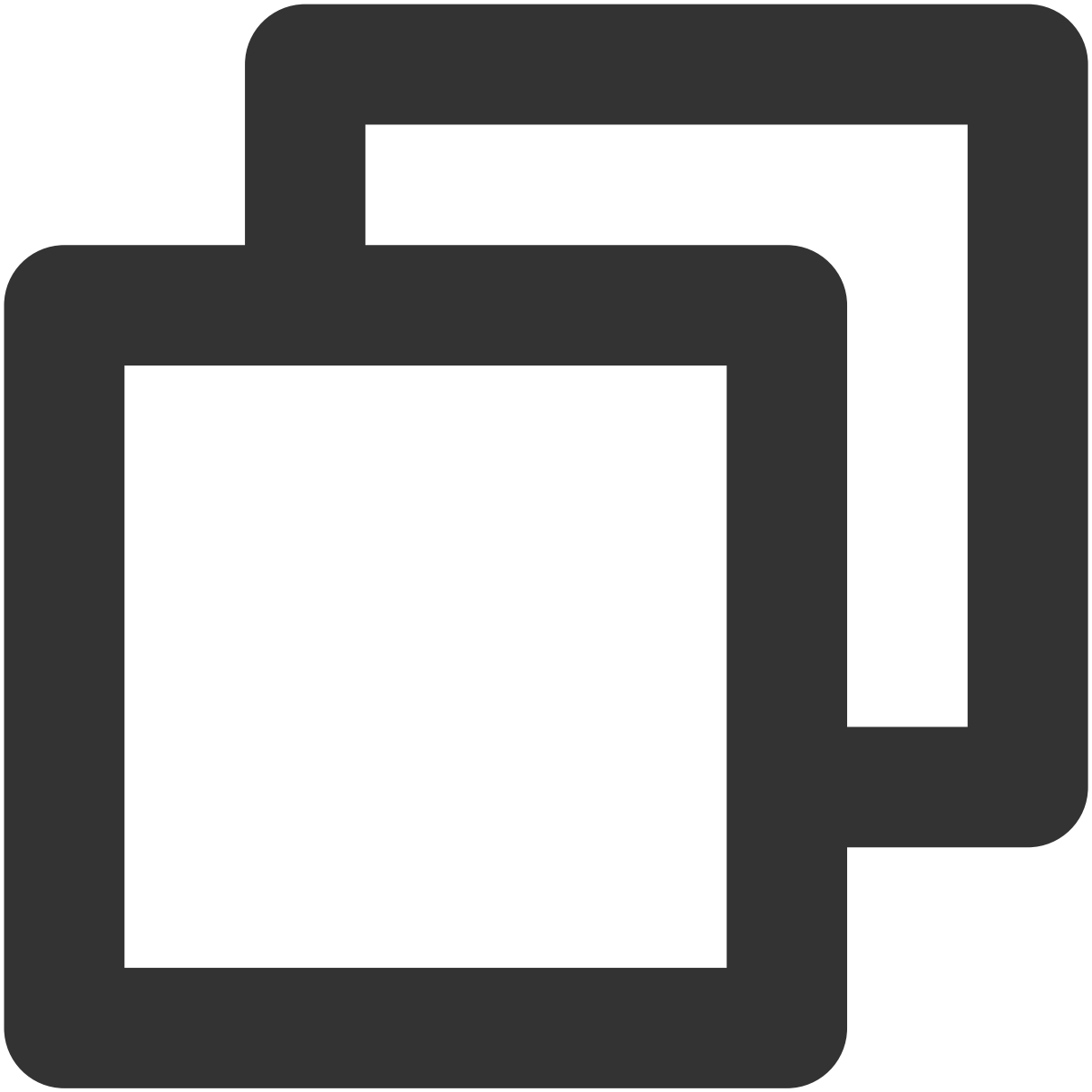
示例：



```
> SELECT day('2009-07-30');
 30
```

DAYOFYEAR

函数语法：



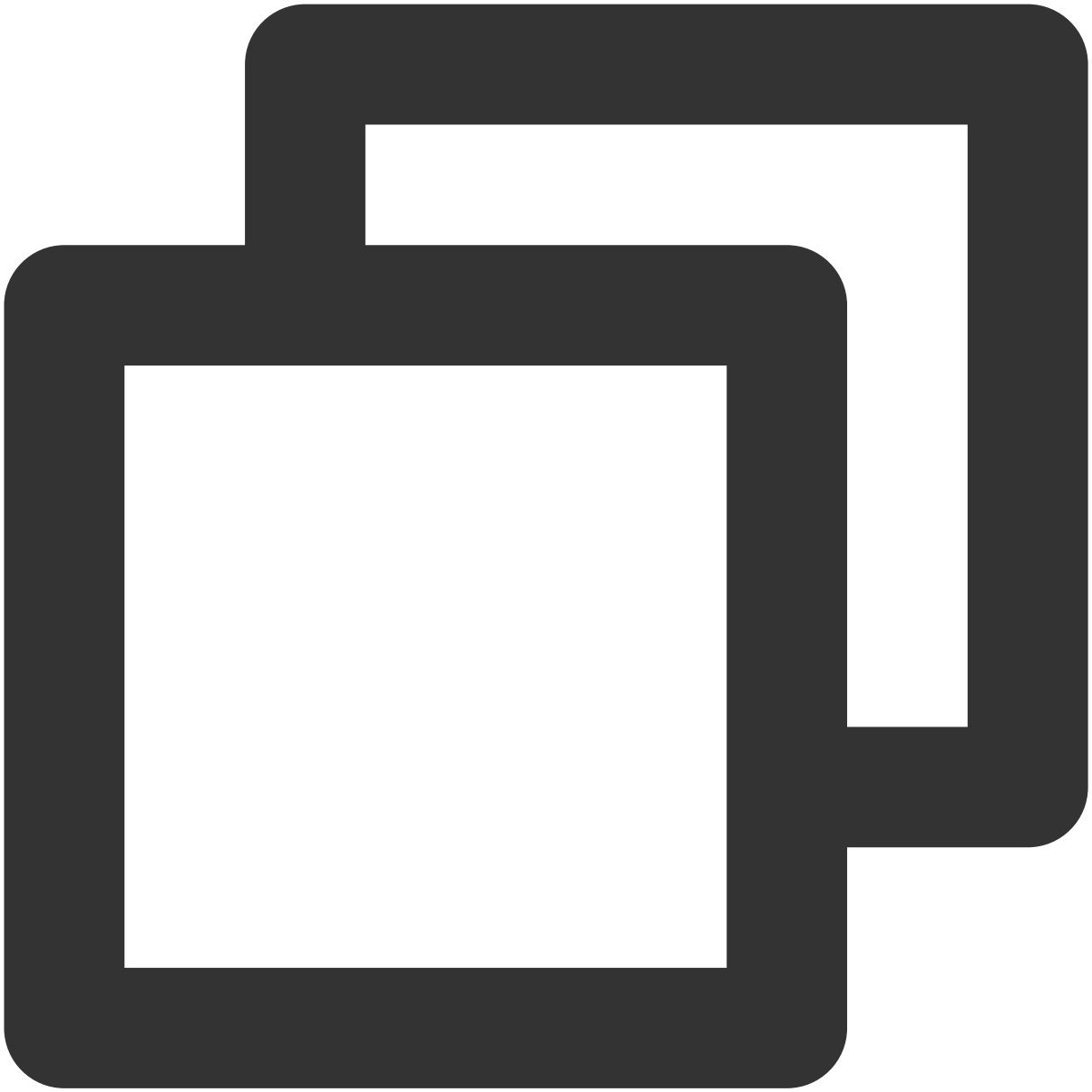
```
DAYOFYEAR(<d> date|timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：计算 d 是这年的第几天。

返回类型：integer。

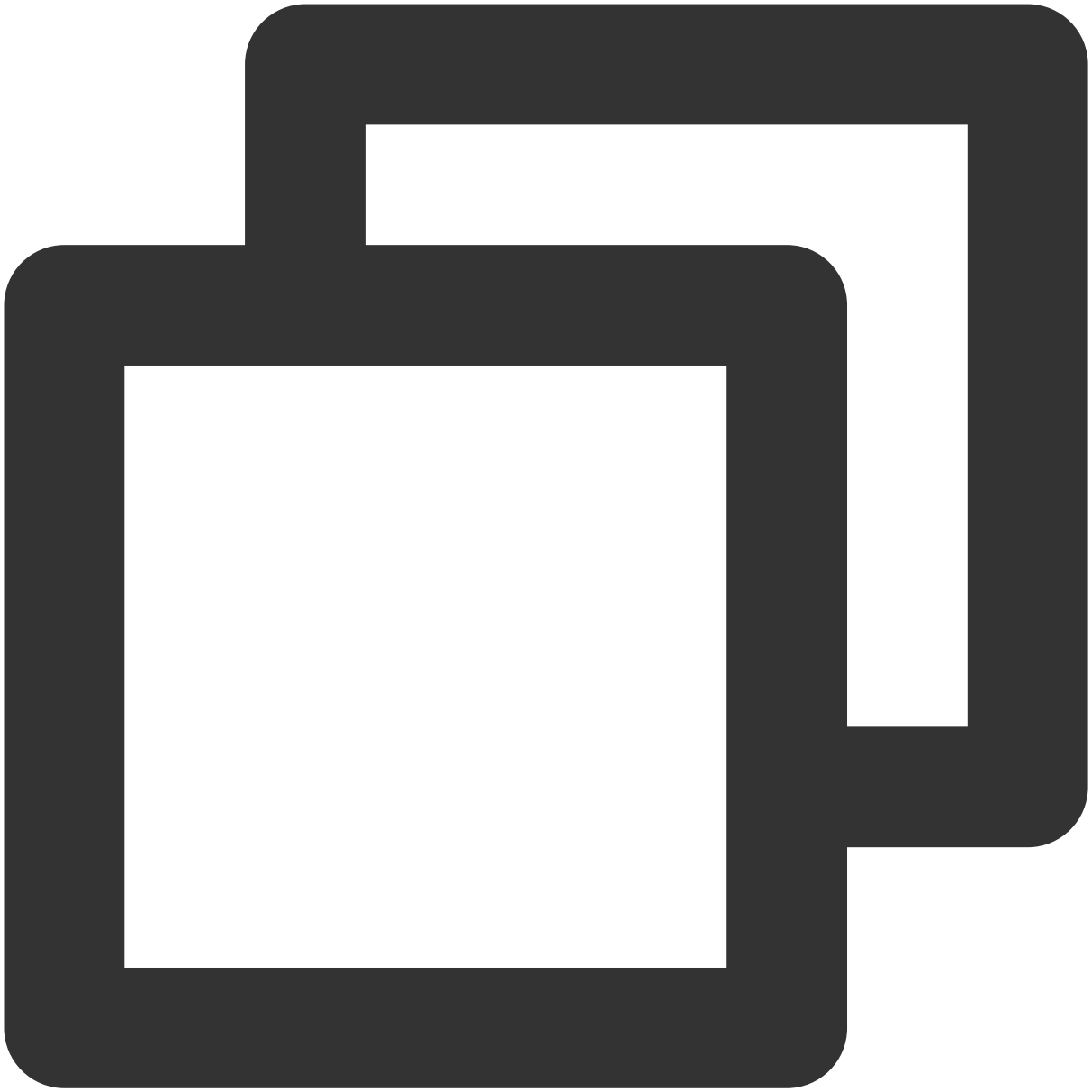
示例：



```
> SELECT dayofyear('2016-04-09');
 100
```

DAYOFMONTH

函数语法：



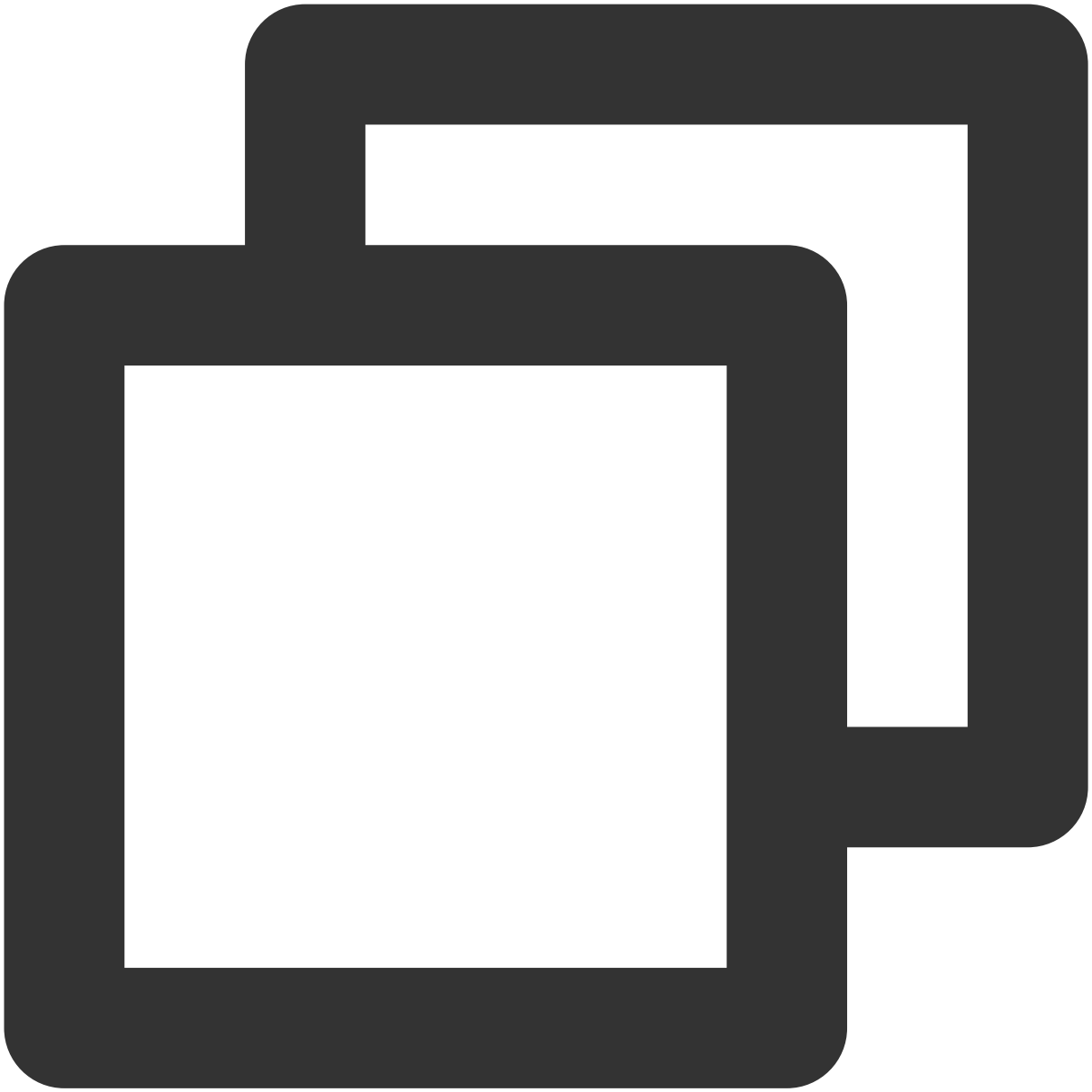
```
DAYOFMONTH
```

支持引擎：SparkSQL、Presto。

使用说明：计算日期/时间戳 d 是这个月的第几天。

返回类型：integer。

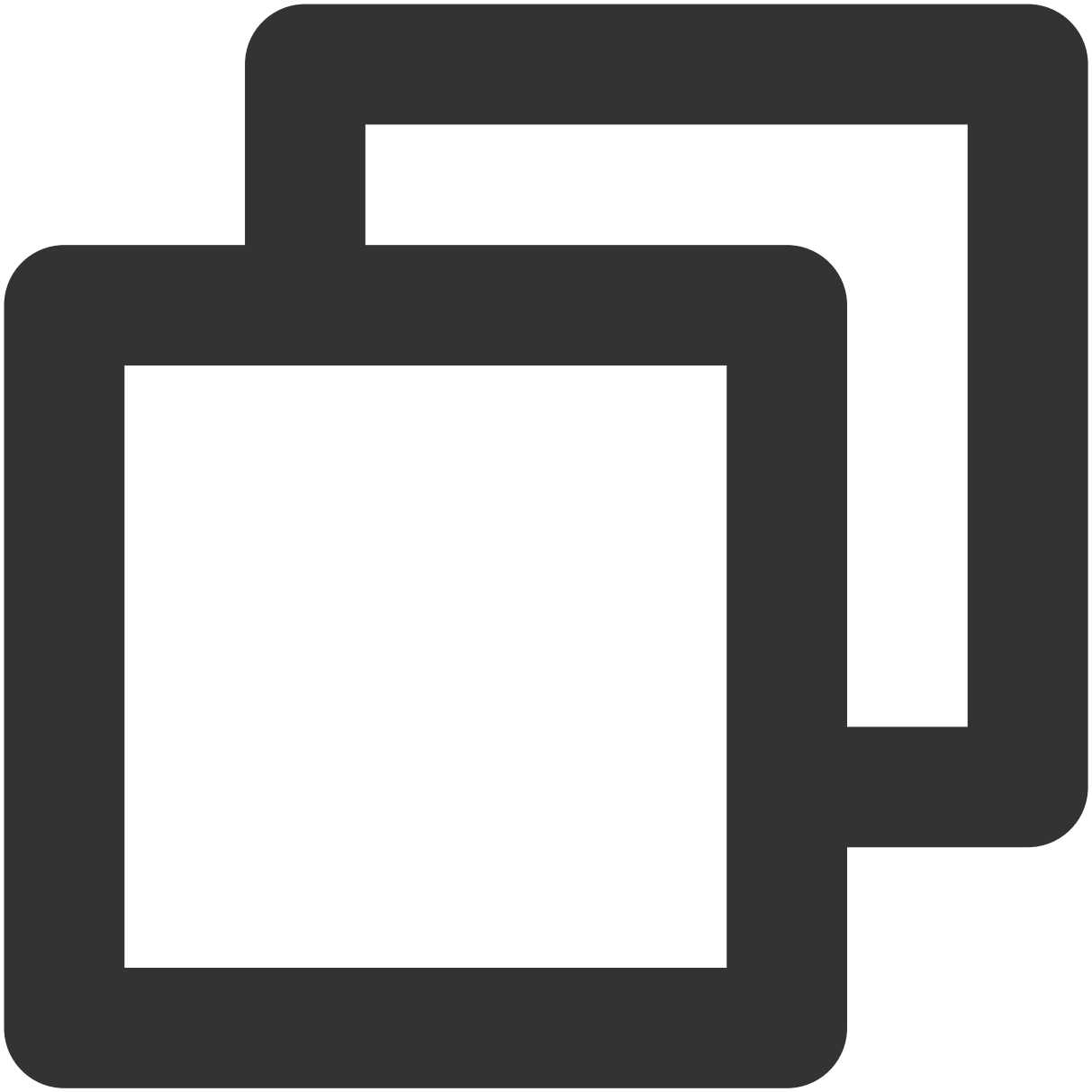
示例：



```
> SELECT dayofmonth('2009-07-30');
 30
```

FROM_UNIXTIME

函数语法：



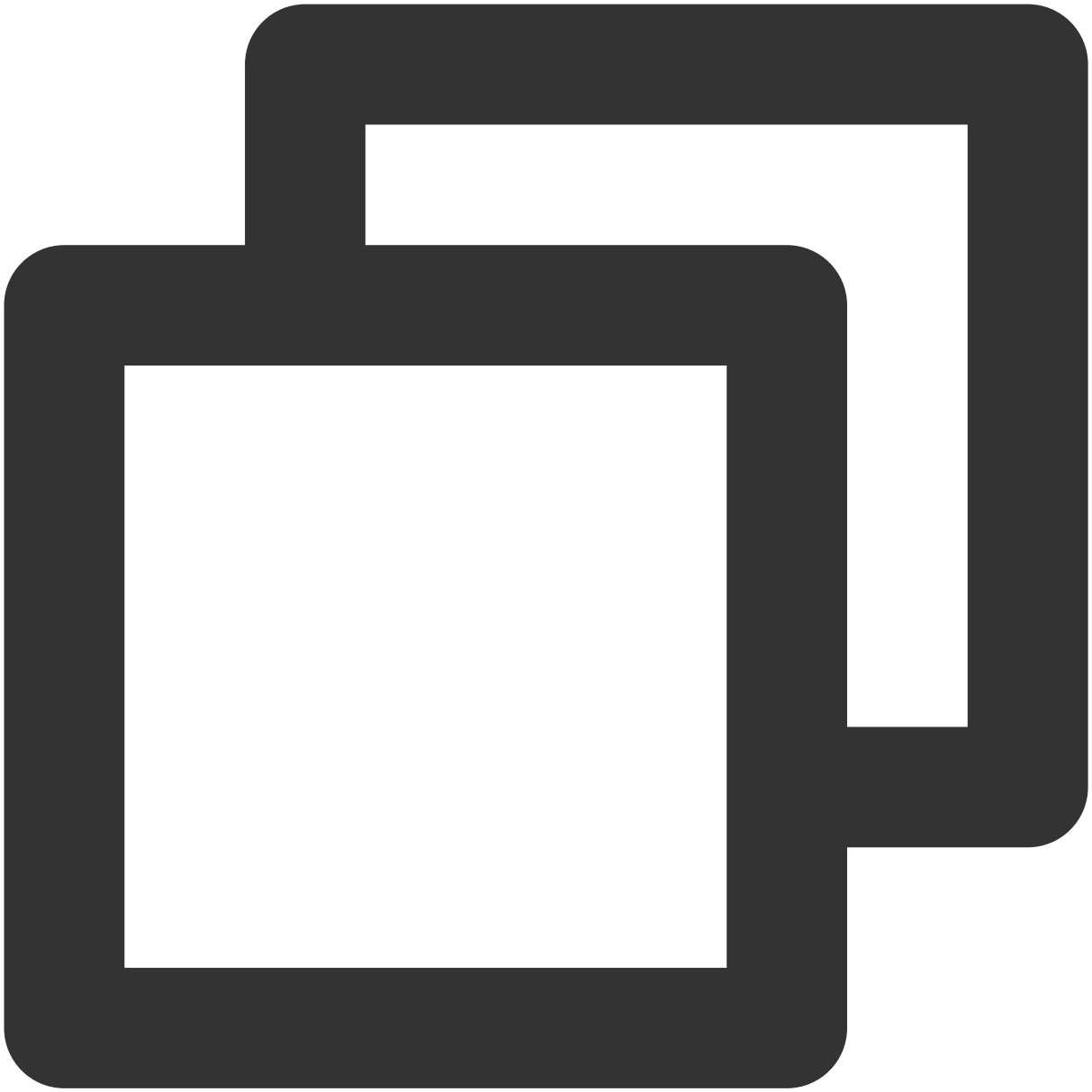
```
FROM_UNIXTIME(<unix_time> bigint[, <fmt> string])
```

支持引擎：SparkSQL、Presto。

使用说明：以格式 fmt 返回 unix_time 所代表的日期/时间。如果省略 fmt，则使用 'yyyy-MM-dd HH:mm:ss'。

返回类型：string。

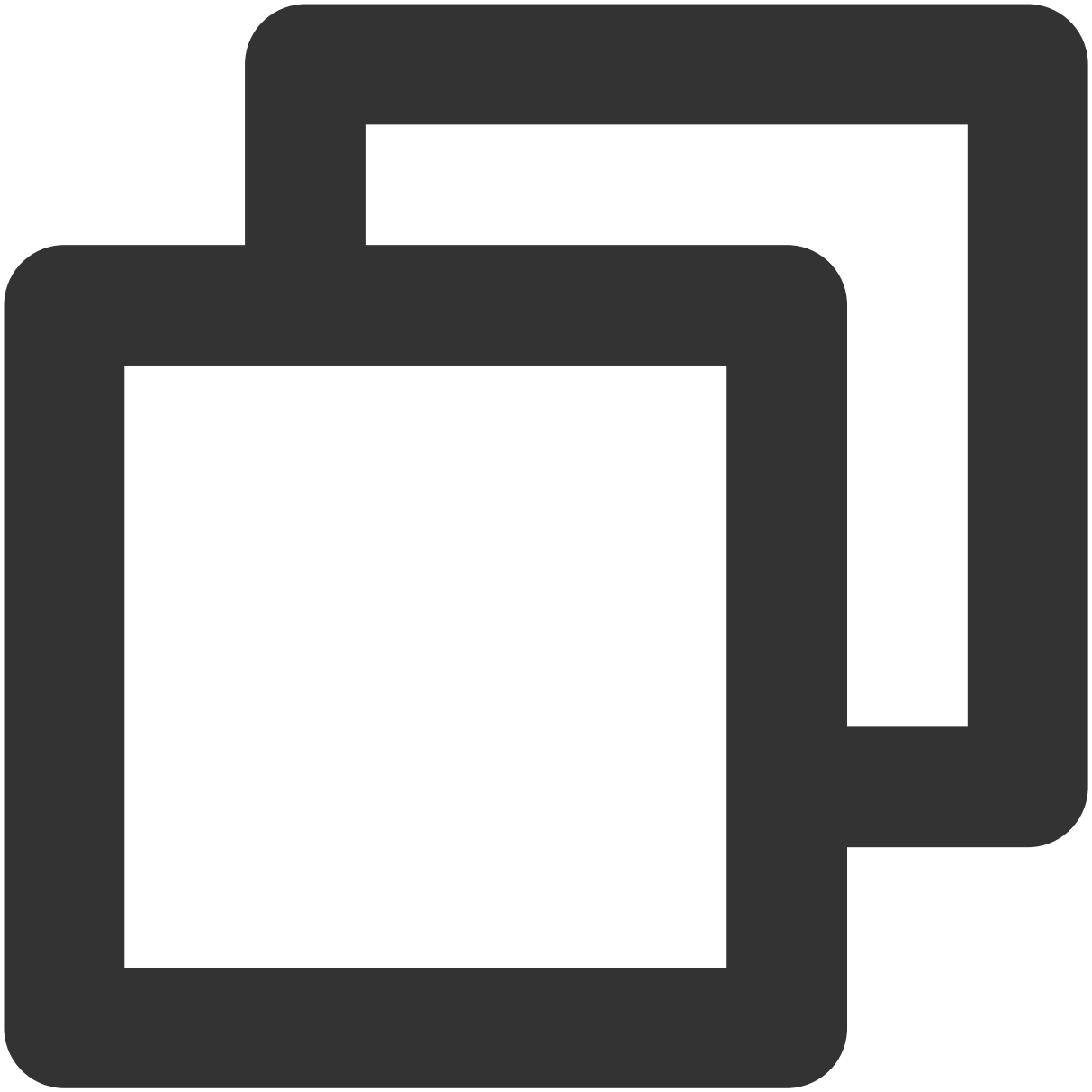
示例：



```
> SELECT from_unixtime(0, 'yyyy-MM-dd HH:mm:ss');
 1969-12-31 16:00:00
> SELECT from_unixtime(0);
 1969-12-31 16:00:00
```

FROM_UTC_TIMESTAMP

函数语法：



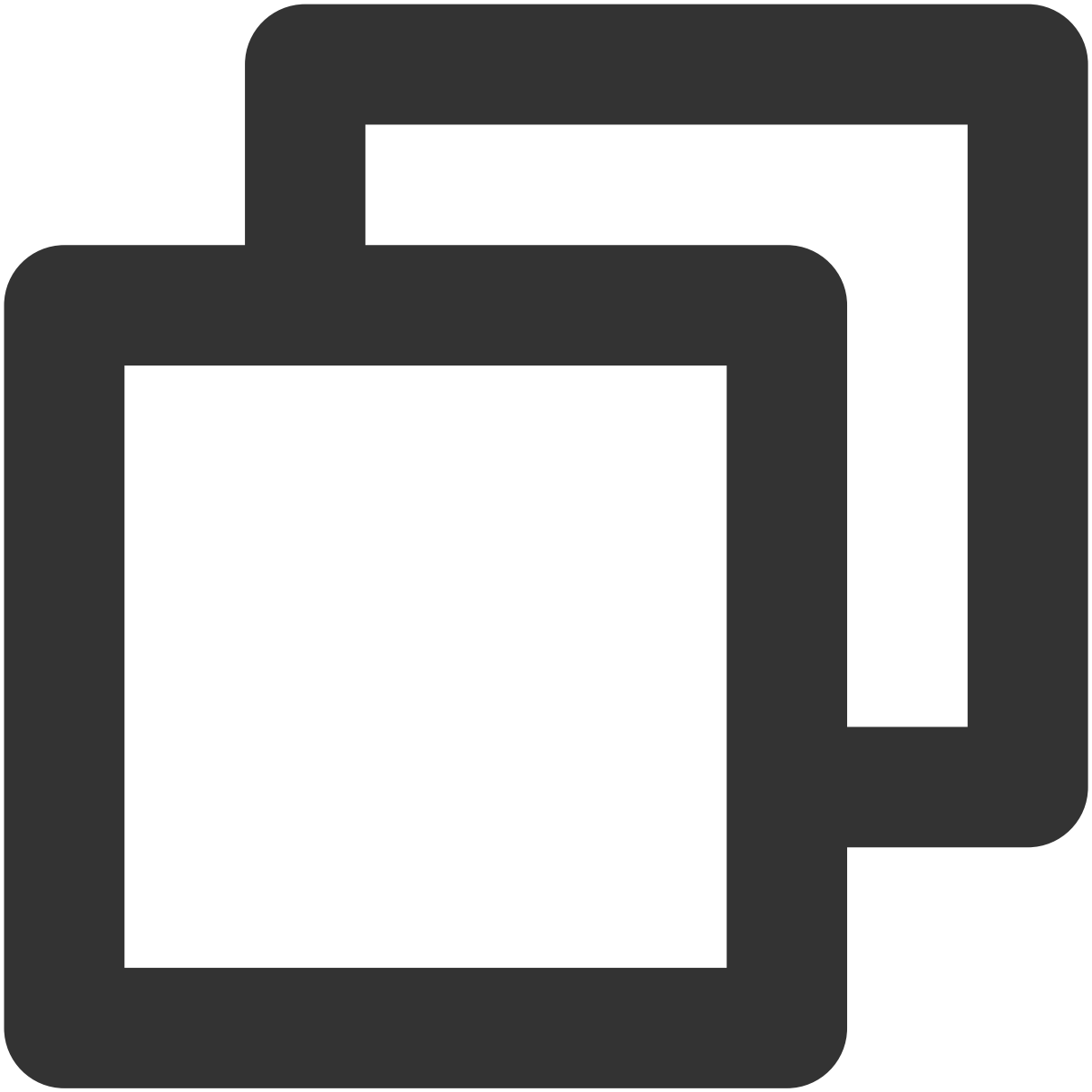
```
FROM_UTC_TIMESTAMP(<ts> timestamp, <timezone> string)
```

支持引擎：SparkSQL、Presto。

使用说明：给定 utc 时间戳并将该时间呈现为给定时区的时间戳。

返回类型：timestamp。

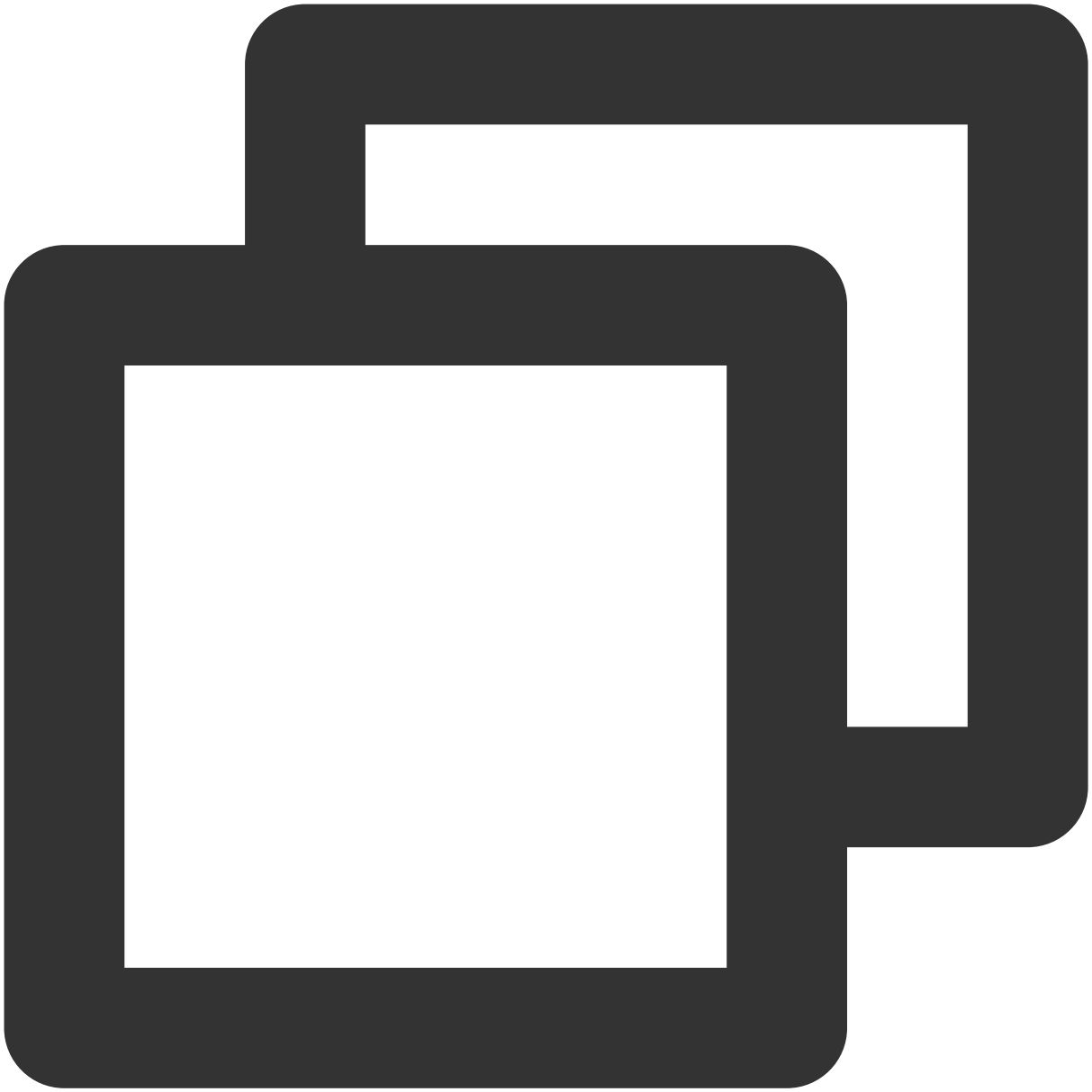
示例：



```
> SELECT from_utc_timestamp('2016-08-31', 'Asia/Seoul');
 2016-08-31 09:00:00
```

HOUR

函数语法：



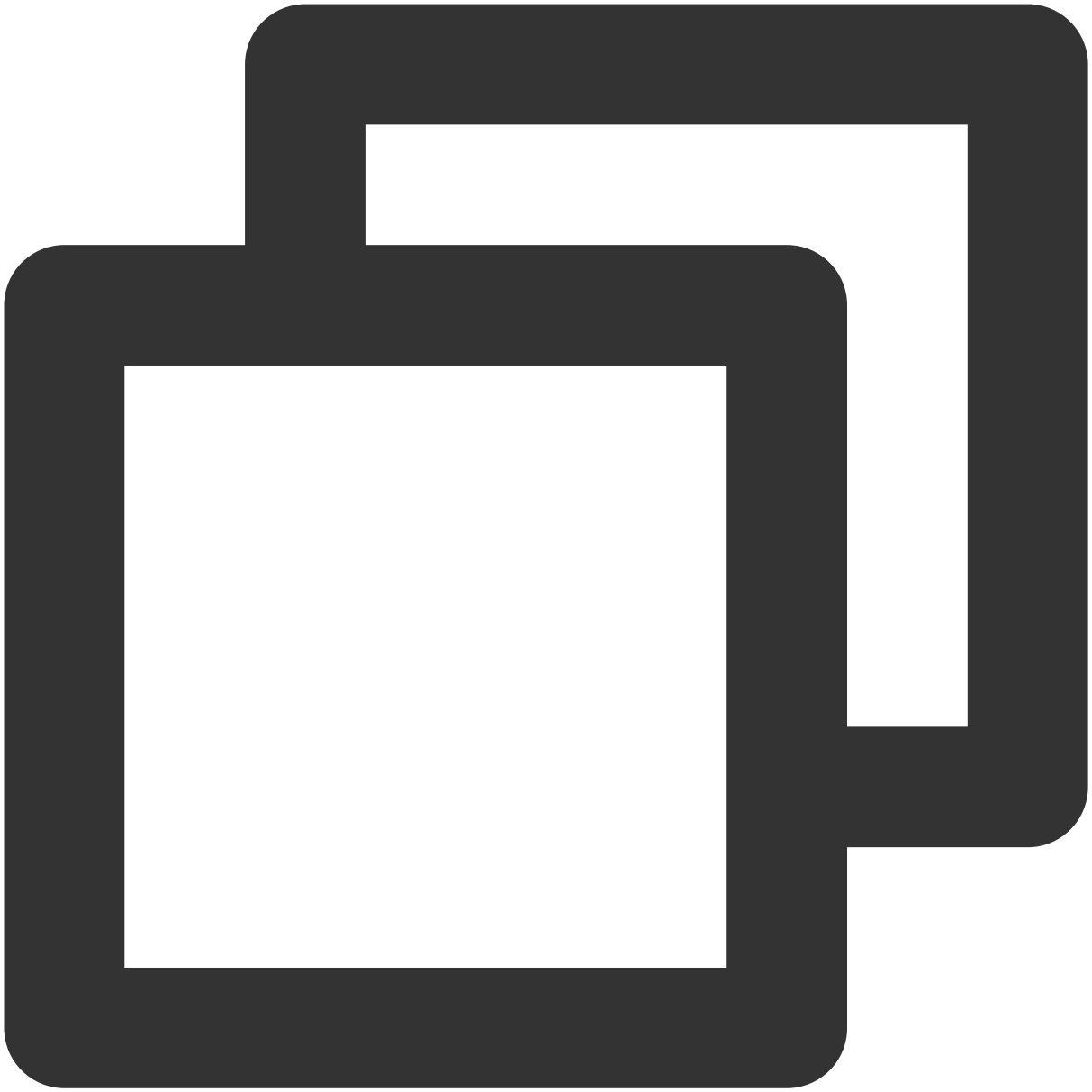
```
HOUR(<ts> string|timestamp)
```

支持引擎：SparkSQL、Presto。

使用说明：返回指定时间戳 ts 的小时部分。

返回类型：integer。

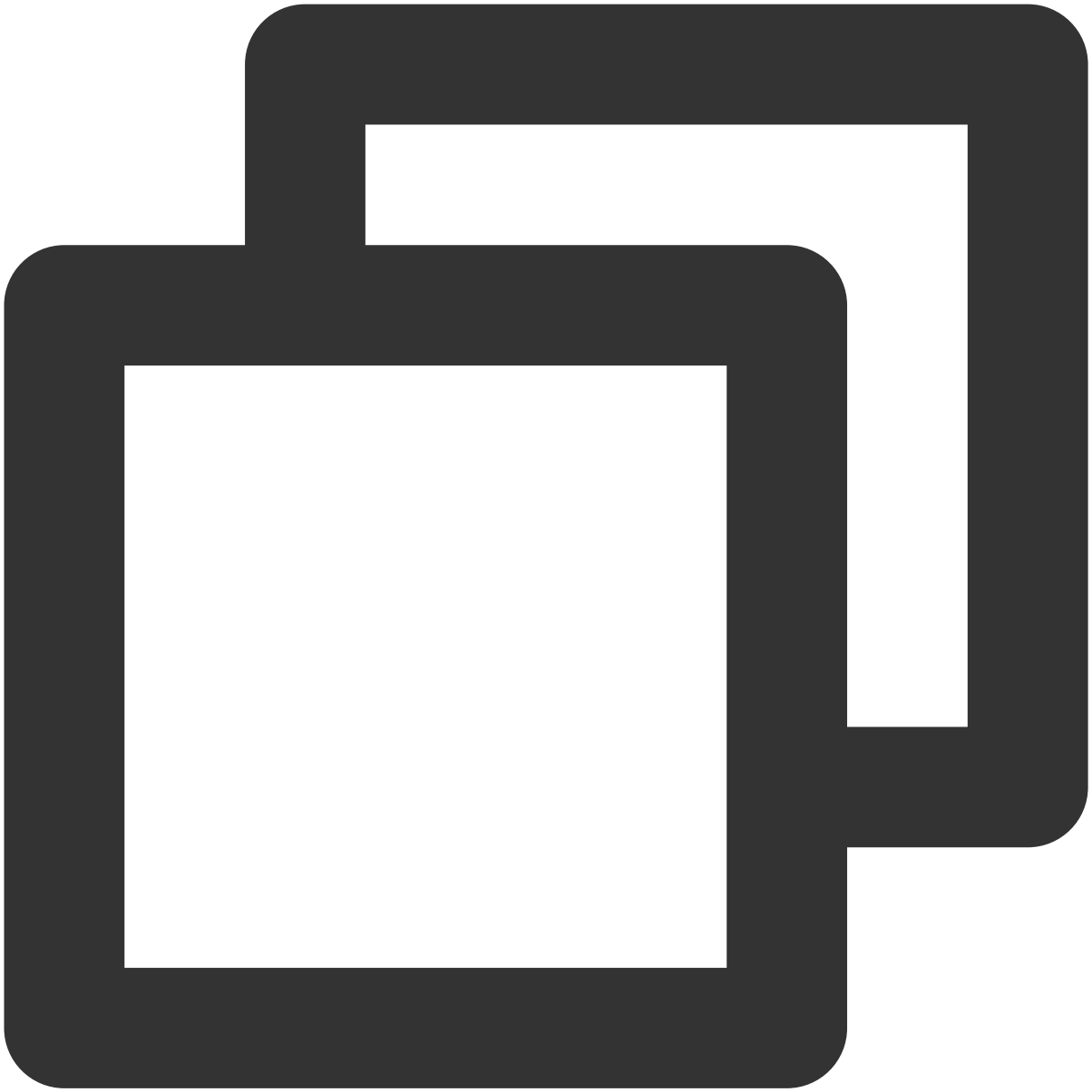
示例：



```
> SELECT hour('2009-07-30 12:58:59');
 12
```

LAST_DAY

函数语法：



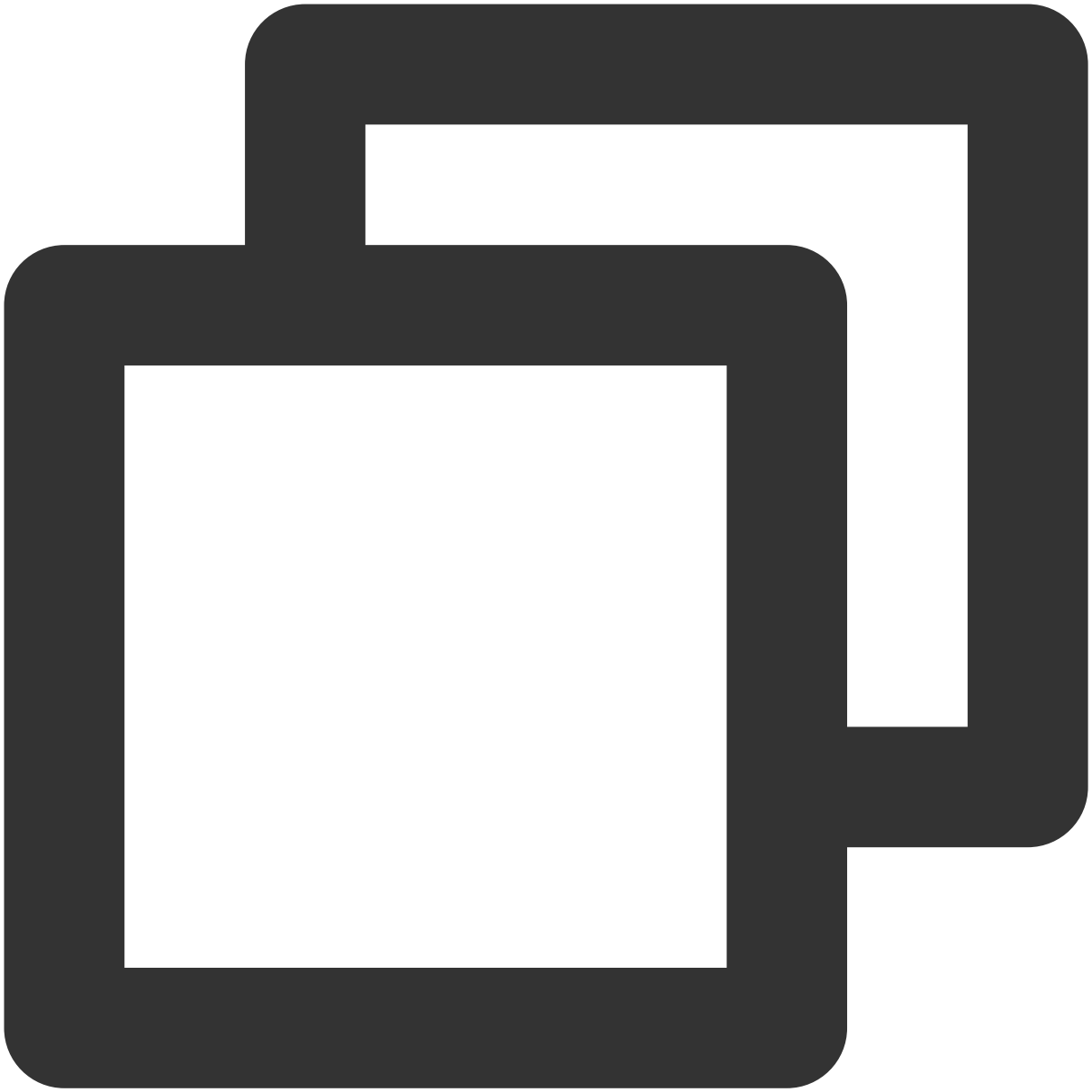
```
LAST_DAY(<d> date|string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回日期 d 当前这个月的最后一天。

返回类型：date。

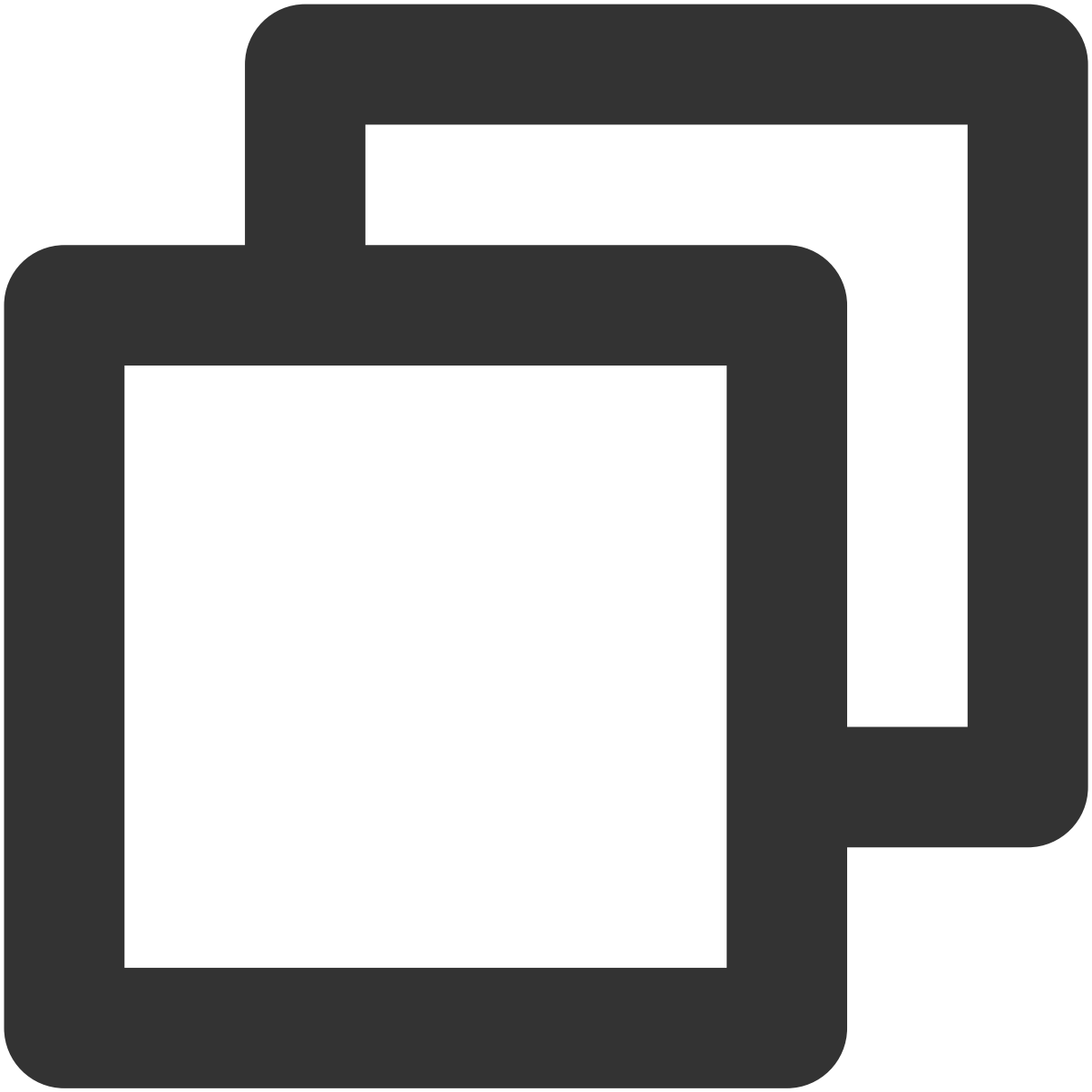
示例：



```
> SELECT last_day('2009-01-12');
 2009-01-31
```

MINUTE

函数语法：



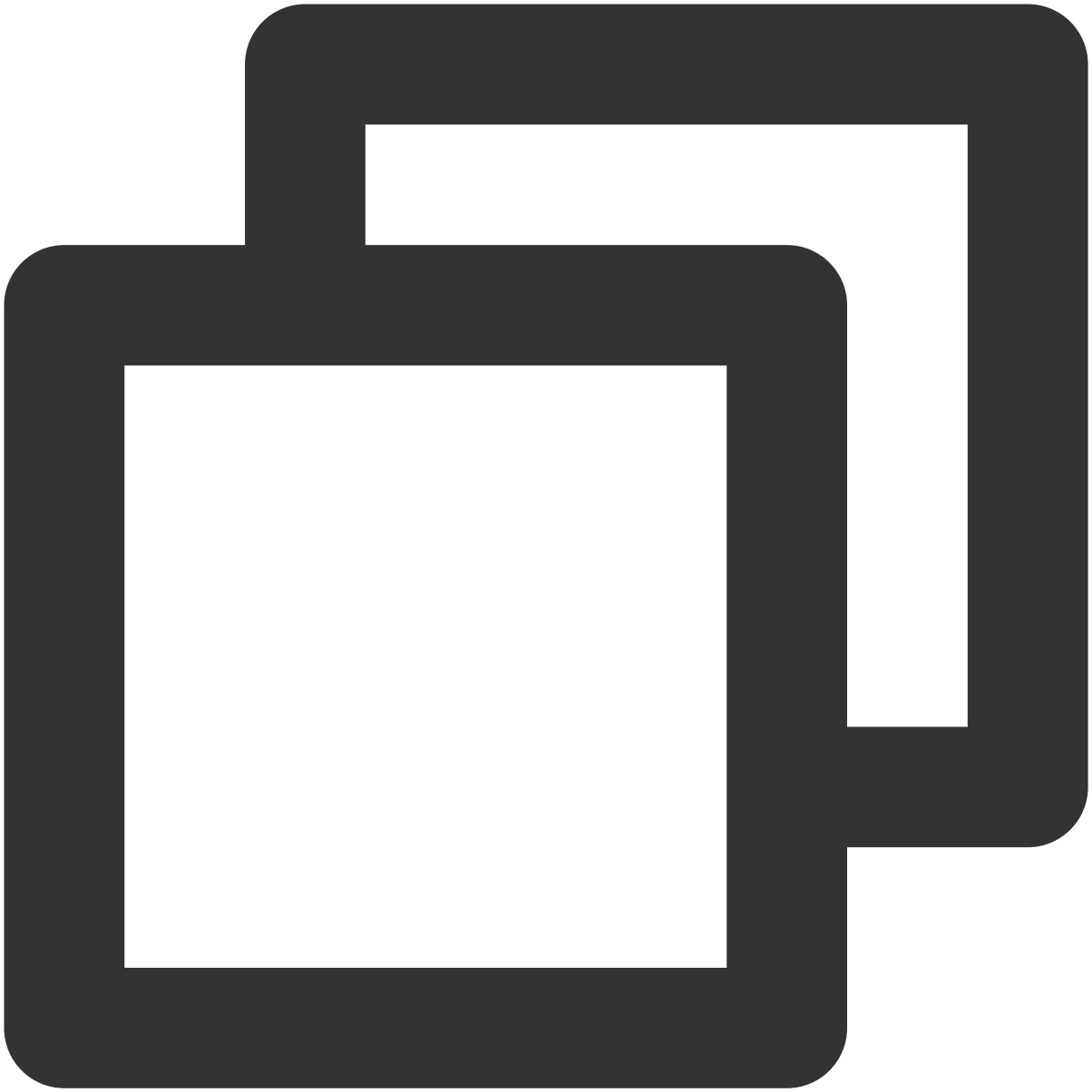
```
MINUTE(<ts> timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回时间戳 ts 的分钟。

返回类型：integer。

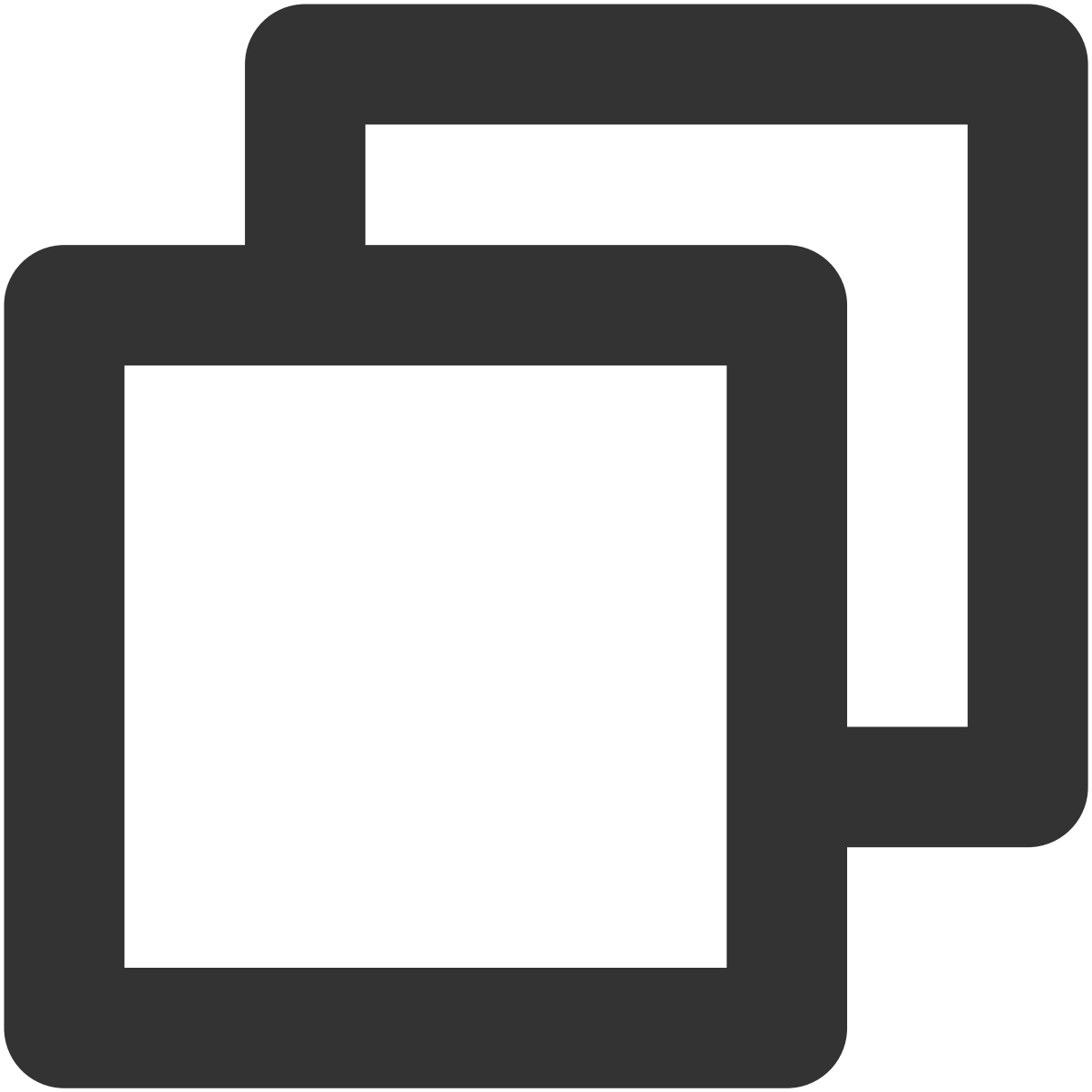
示例：



```
> SELECT minute('2009-07-30 12:58:59');
 58
```

MONTH

函数语法：



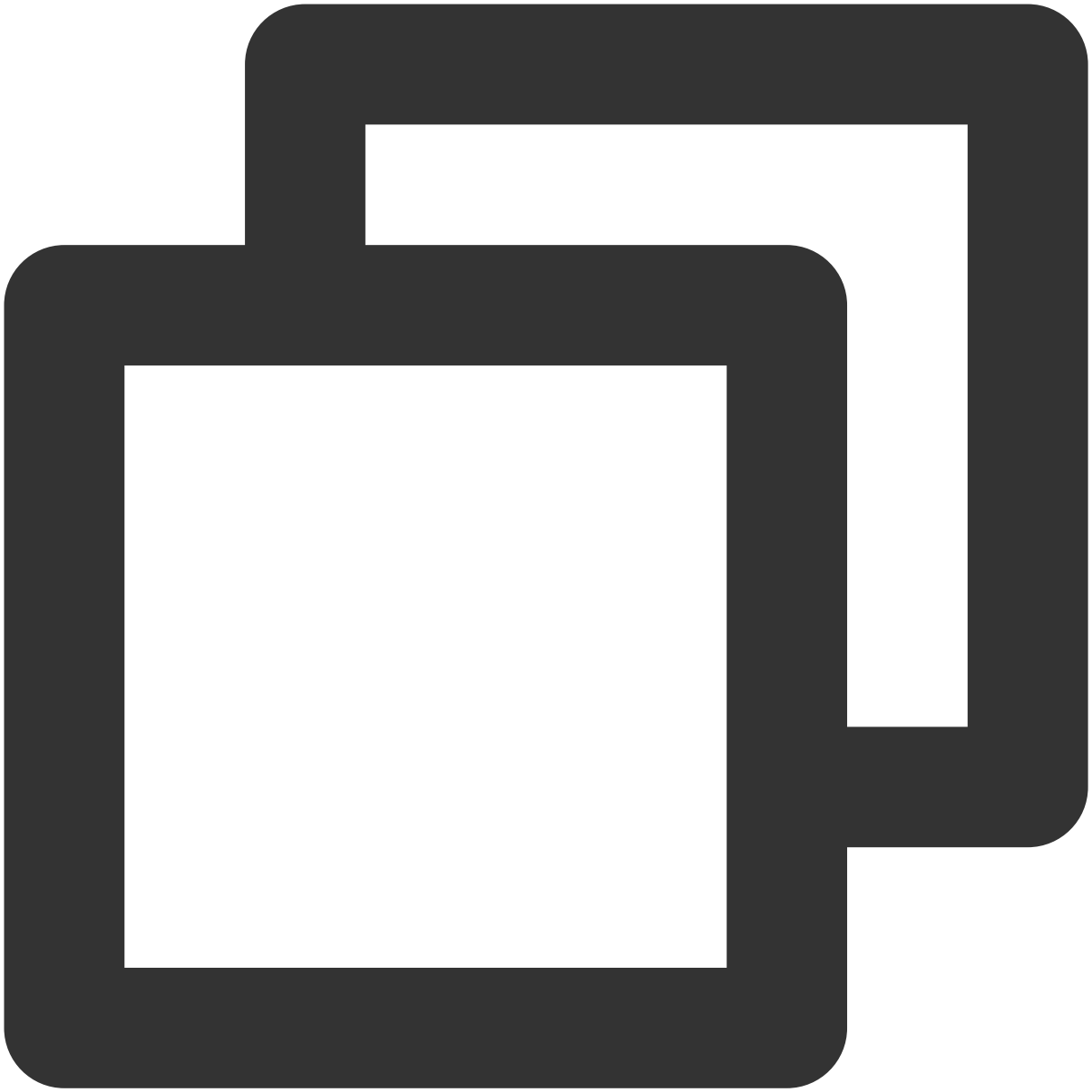
```
MONTH(<d> date|timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回日期 d 的月份。

返回类型：integer。

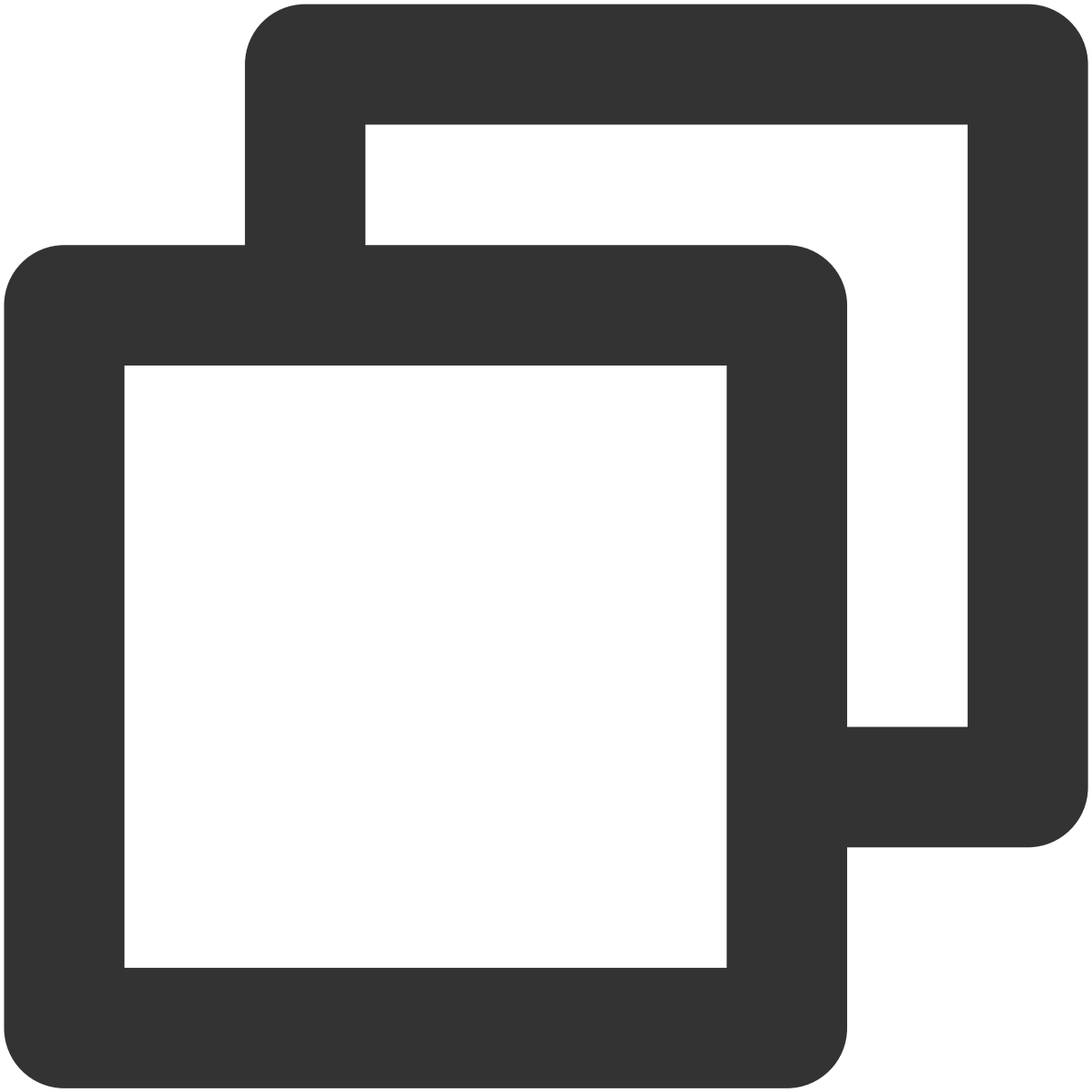
示例：



```
> SELECT month('2016-07-30');
 7
```

MONTHS_BETWEEN

函数语法：



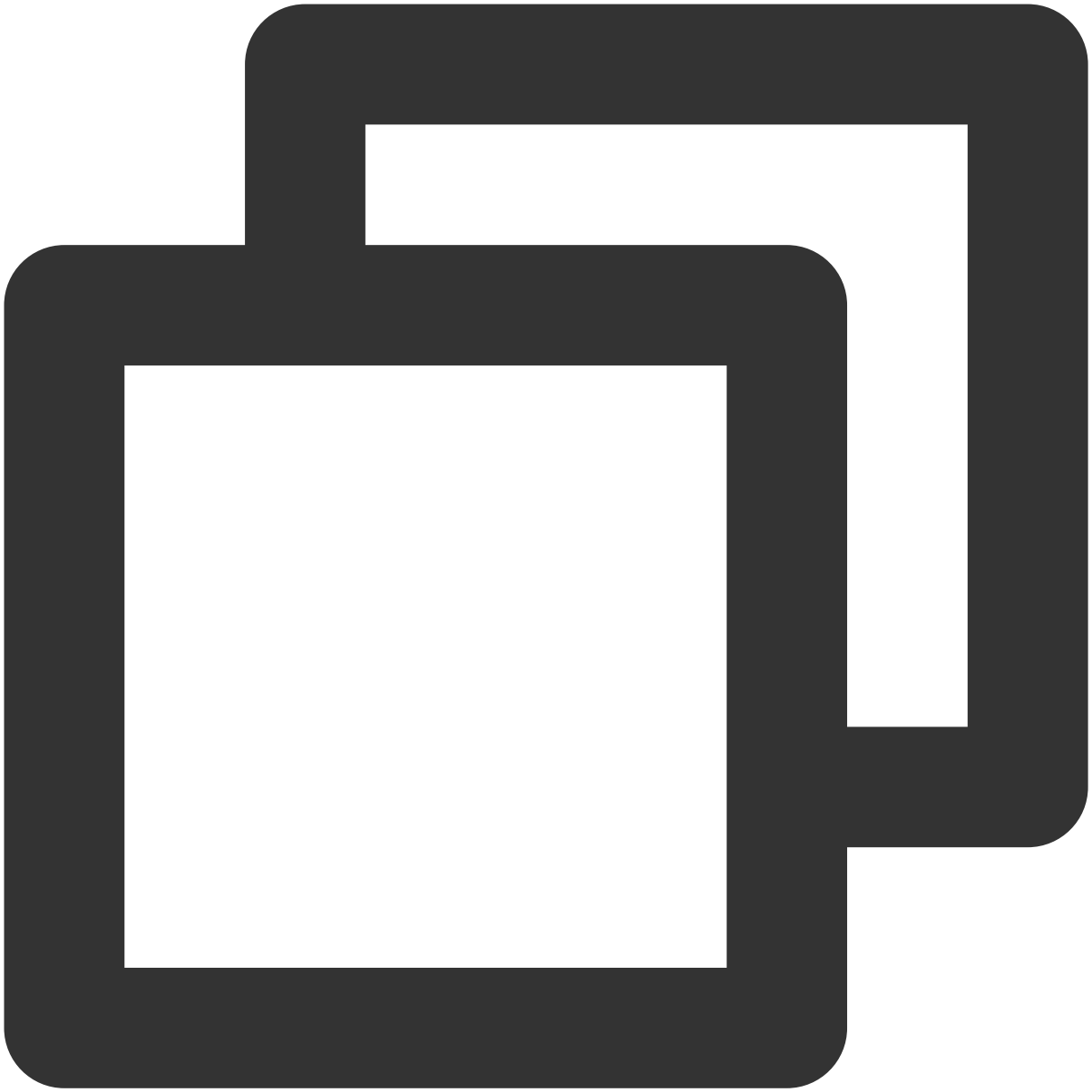
```
MONTHS_BETWEEN(<ts1> date|timestamp|string, <ts2> date|timestamp|string, <roundOff>
```

支持引擎：SparkSQL、Presto。

使用说明：如果 ts1 晚于 ts2，则结果为正。如果 ts1 和 ts2 在当月的同一天，或两者都是当月的最后一天，则将忽略当天的时间。否则，差值根据每月31天计算，并四舍五入到8位，除非 roundOff 为false。

返回类型：double

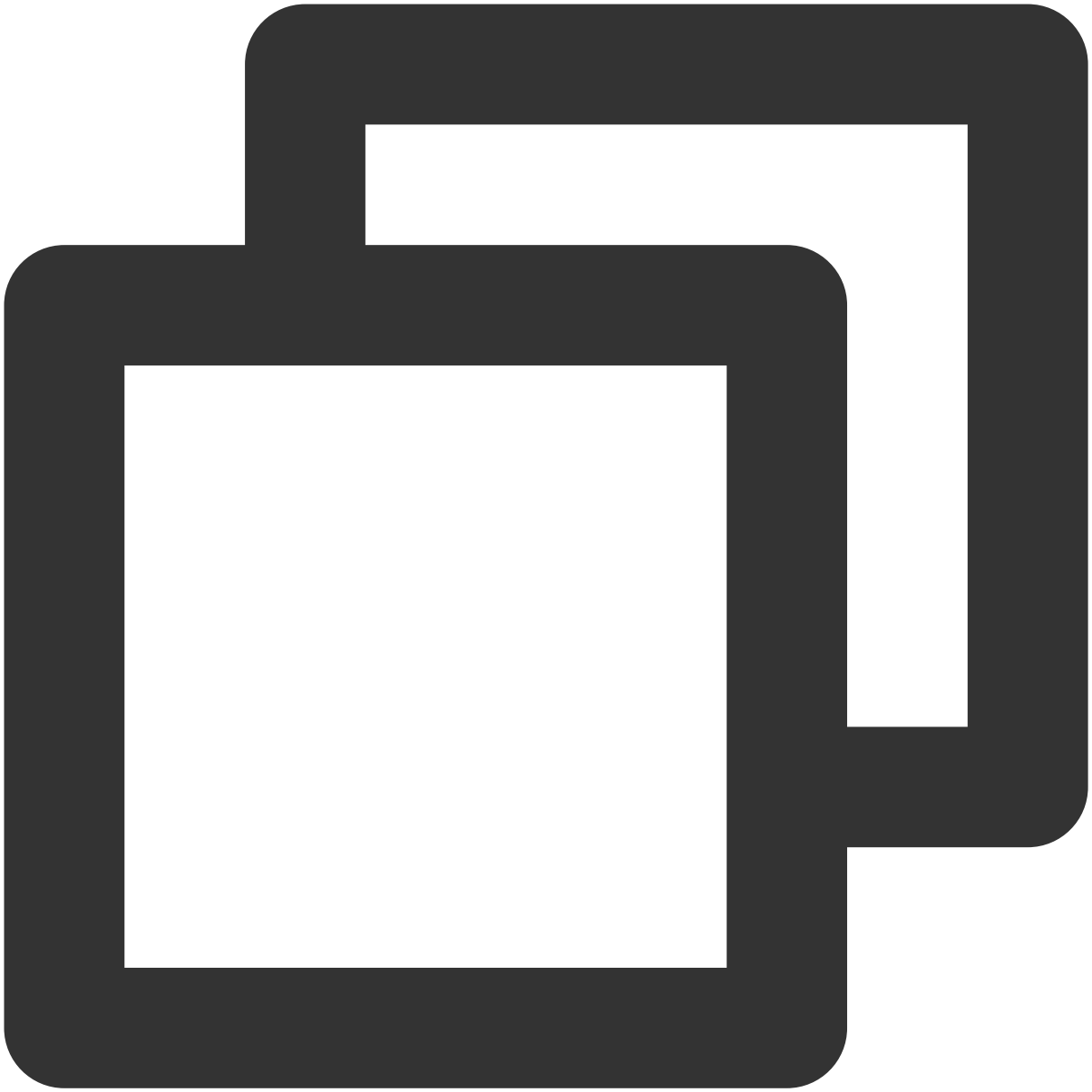
示例：



```
> SELECT months_between('1997-02-28 10:30:00', '1996-10-30');
 3.94959677
> SELECT months_between('1997-02-28 10:30:00', '1996-10-30', false);
 3.9495967741935485
```

NEXT_DAY

函数语法：



```
NEXT_DAY(<start_date> date|timestamp|string, <day_of_week> string
```

支持引擎：SparkSQL、Presto。

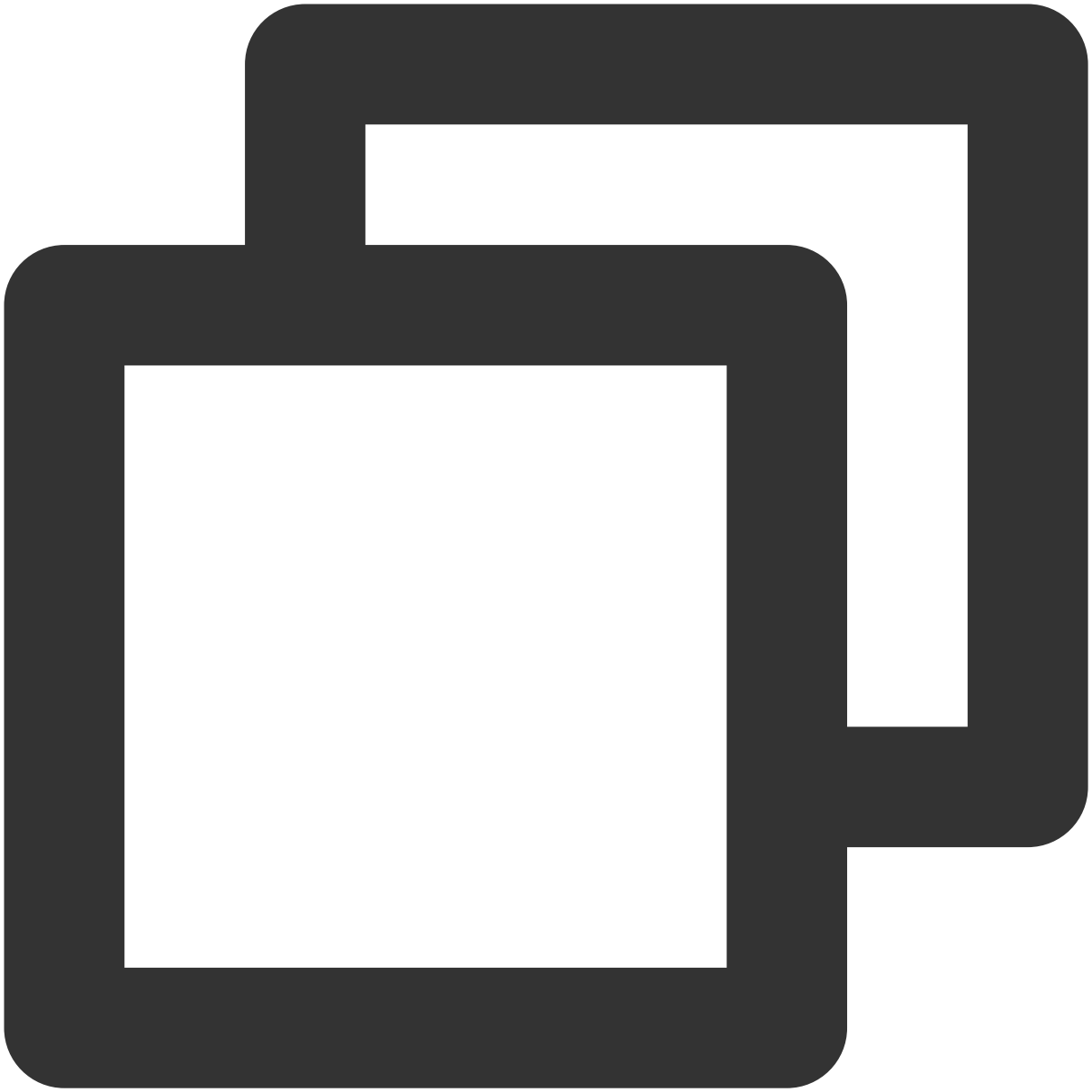
使用说明：返回start_date后的第一个指定星期。

返回类型：

SparkSQL：date

Presto：string

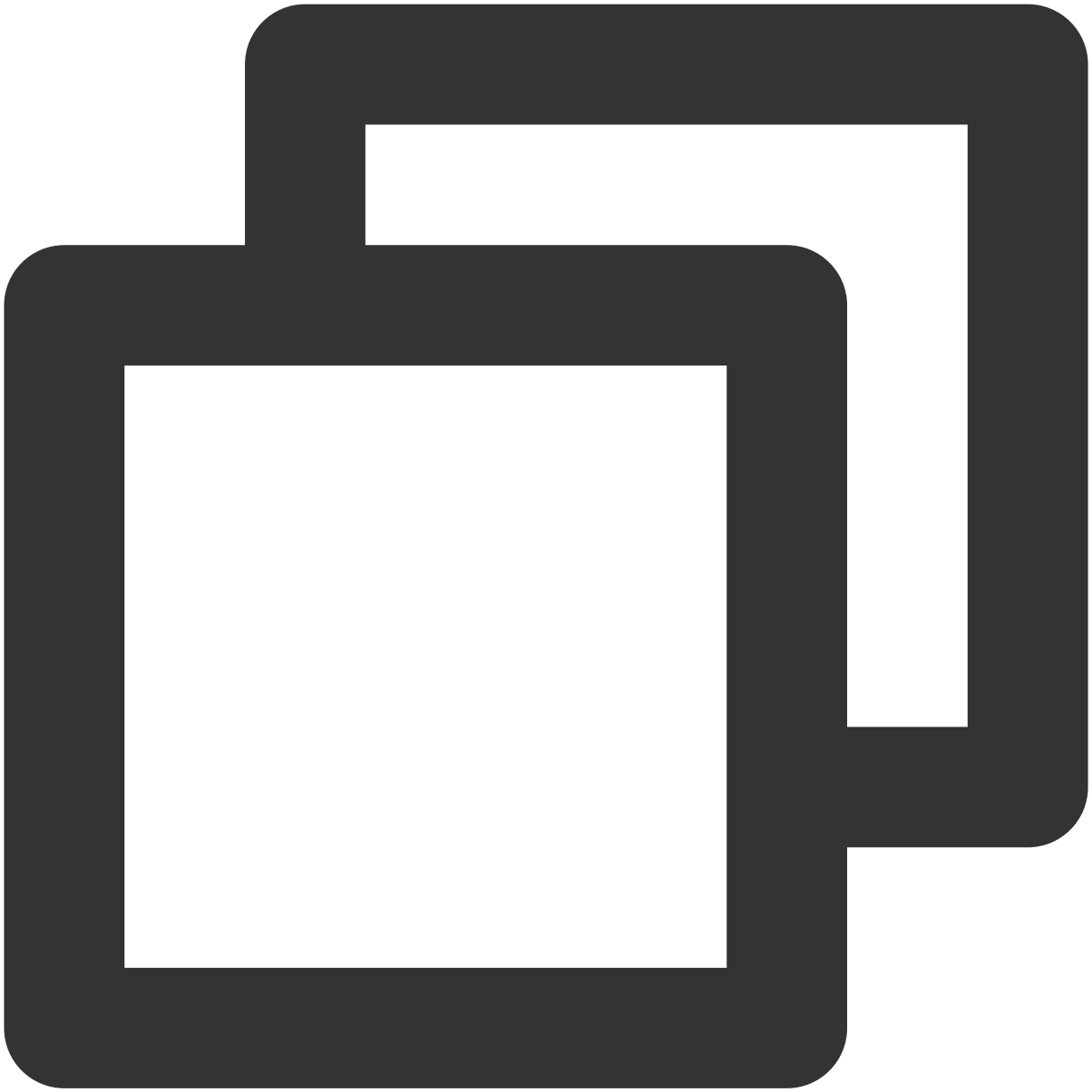
示例：



```
> SELECT next_day('2015-01-14', 'TU');
 2015-01-20
```

NOW

函数语法：



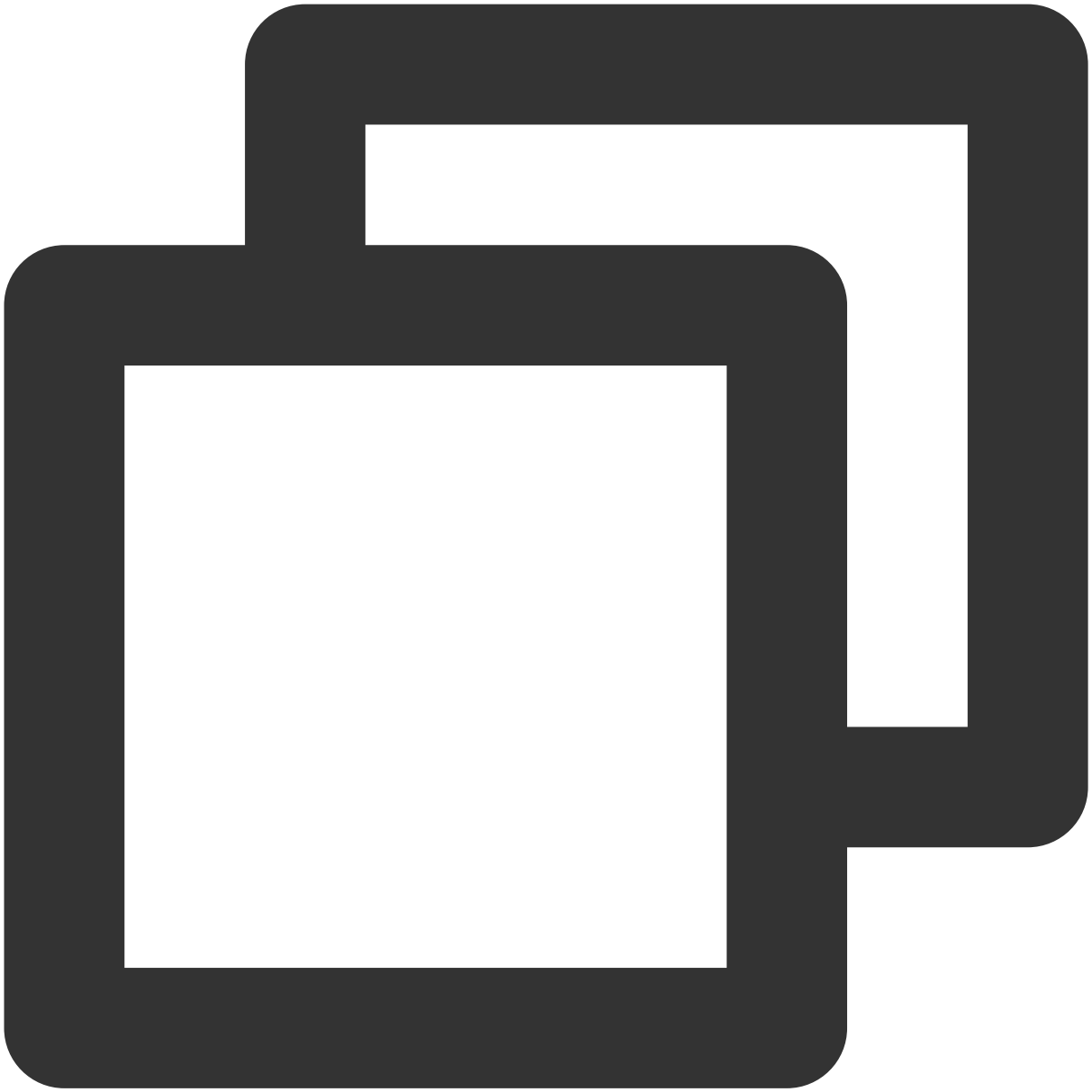
```
NOW()
```

支持引擎：SparkSQL、Presto。

使用说明：返回当前时间戳。

返回类型：timestamp。

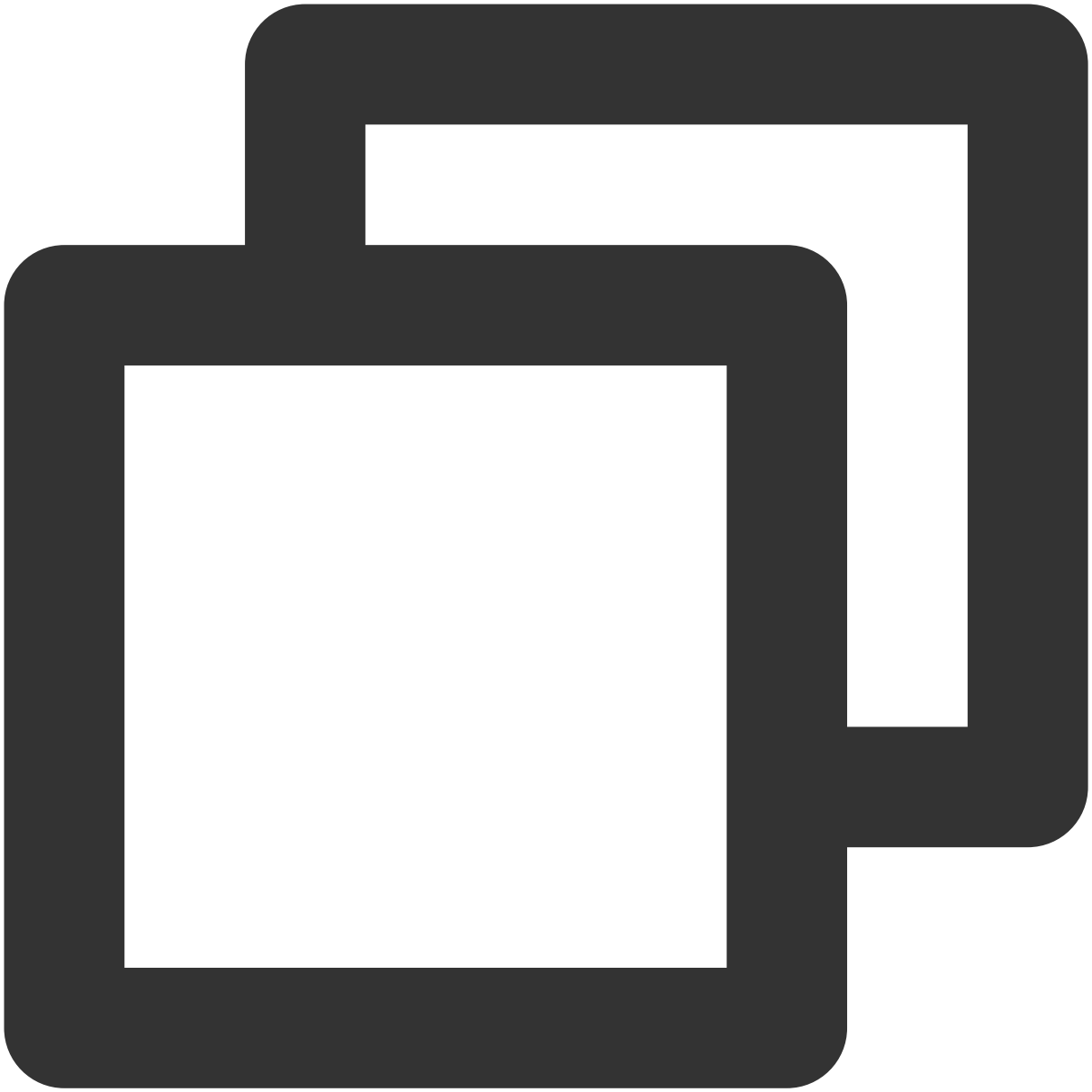
示例：



```
> SELECT now();
 2020-04-25 15:49:11.914
```

QUARTER

函数语法：



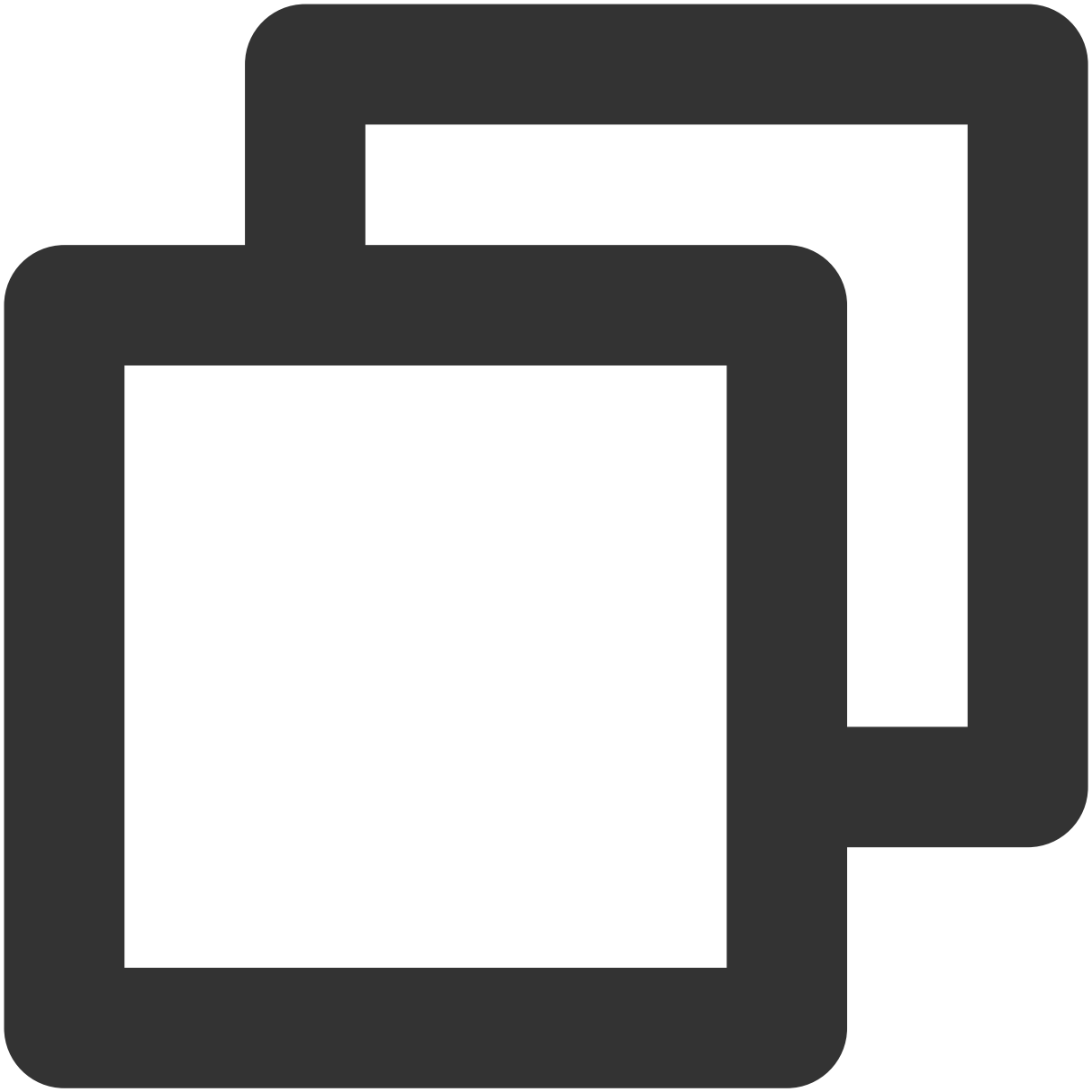
```
QUARTER(<d> date|timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回 d 所在季度。

返回类型：integer。

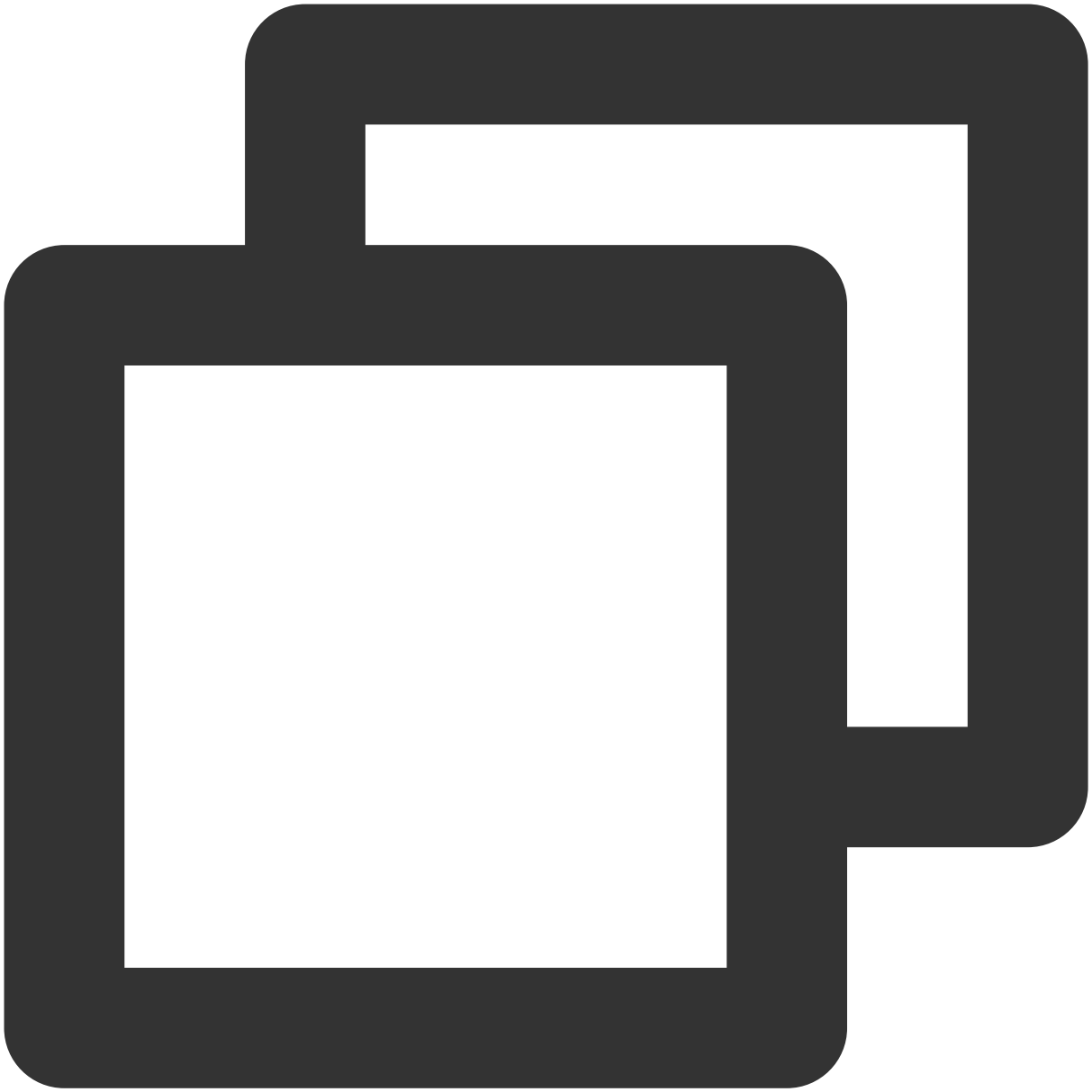
示例：



```
> SELECT quarter('2016-08-31');
 3
```

SECOND

函数语法：



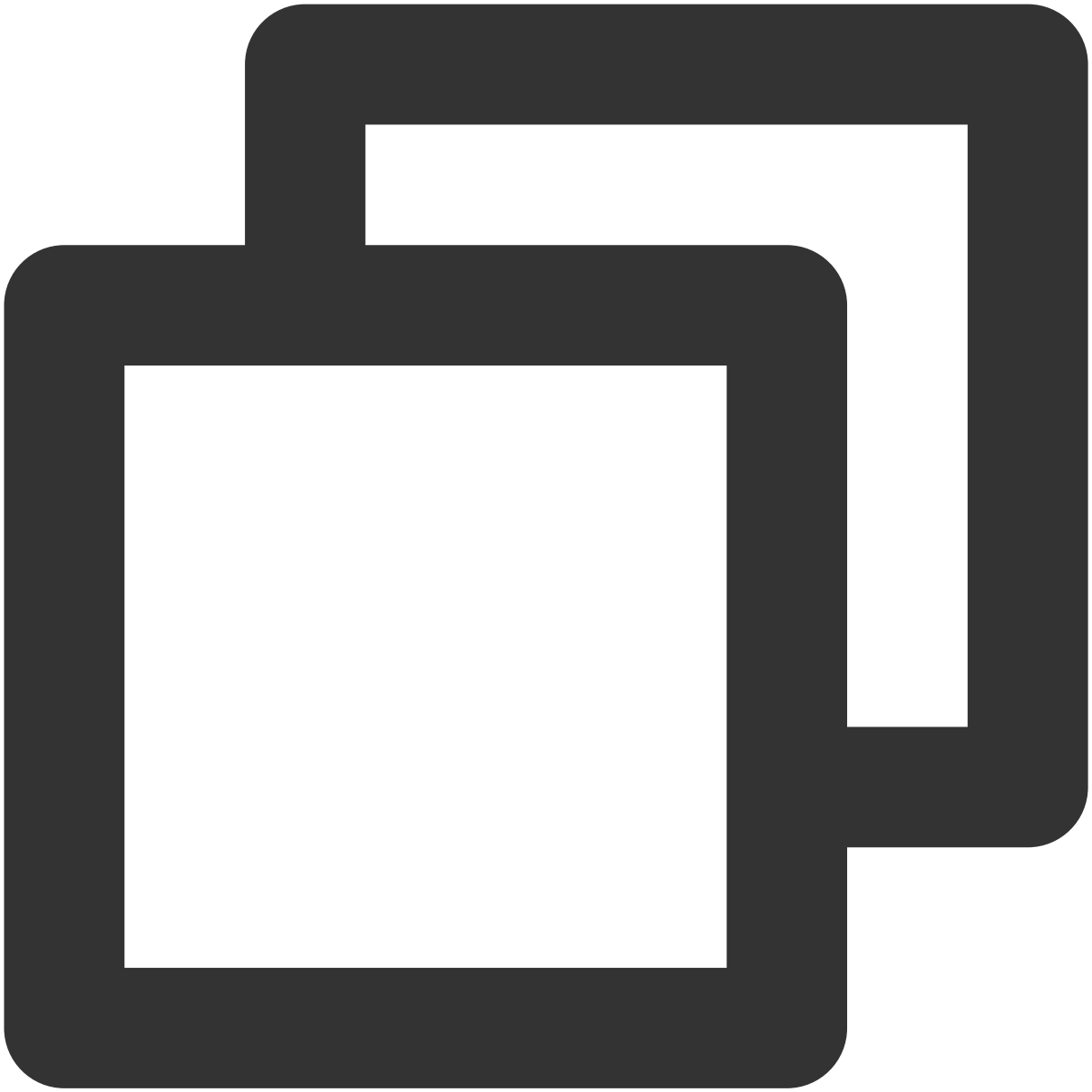
```
SECOND(<ts> timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回当前时间戳的秒数。

返回类型：integer。

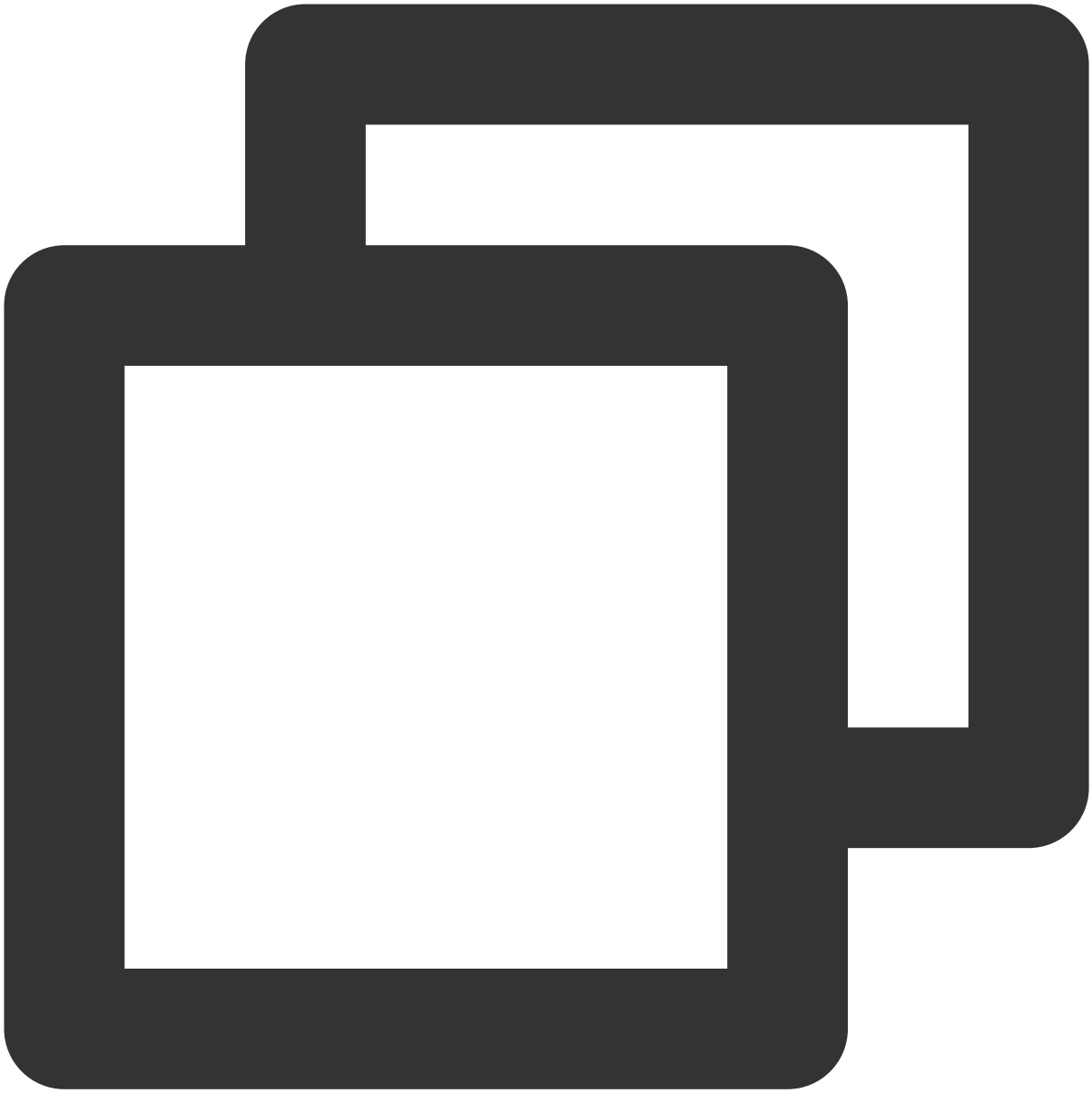
示例：



```
> SELECT second('2009-07-30 12:58:59');
 59
```

TO_TIMESTAMP

函数语法：



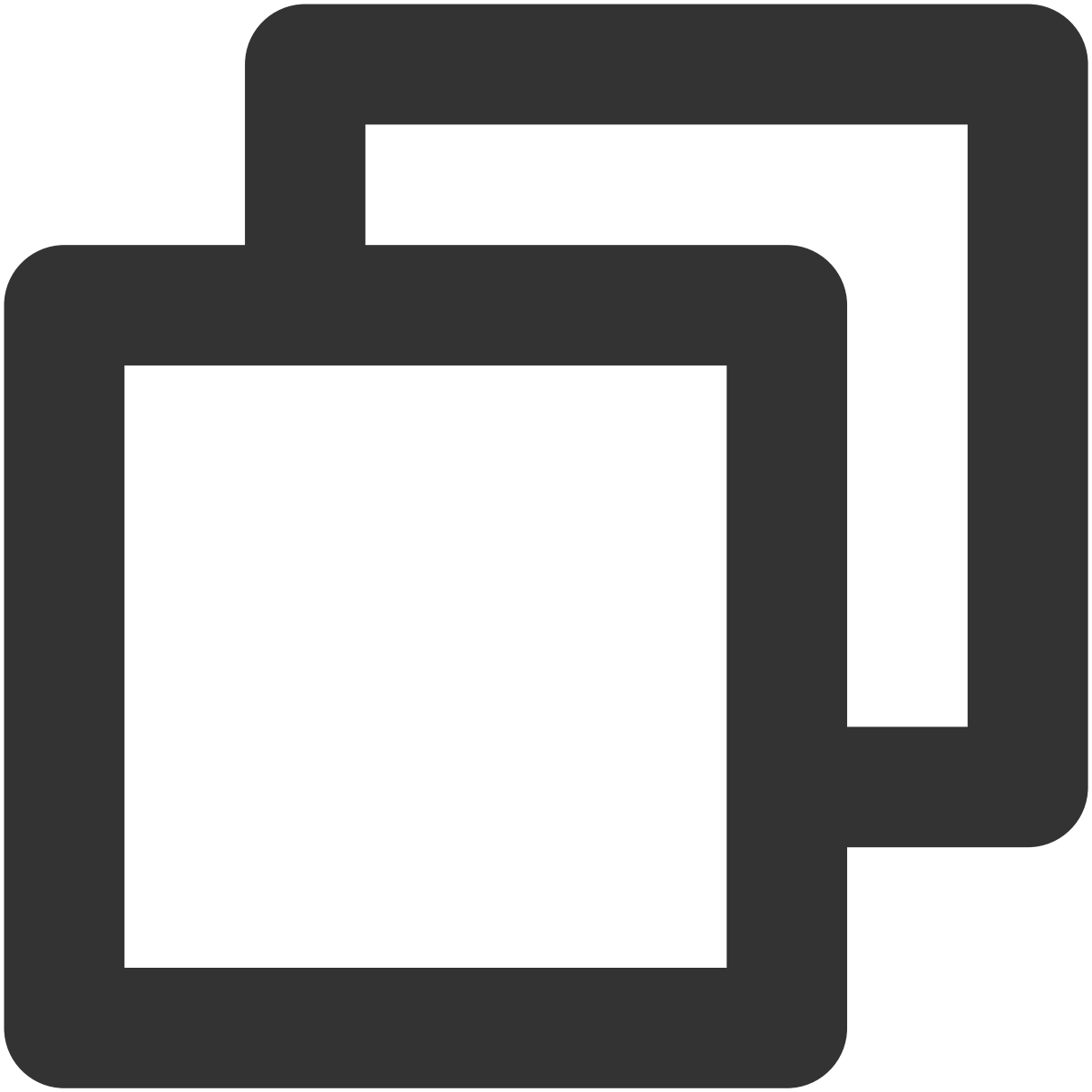
```
TO_TIMESTAMP(<ts_str> string[, <fmt> string]
```

支持引擎：SparkSQL、Presto。

使用说明：将 fmt 格式的 ts_str 表达式解析为时间戳。输入无效则返回 NULL。默认情况下，如果省略 fmt，它将遵循时间戳的强制转换规则。结果数据类型与配置值一致。

返回类型：timestamp。

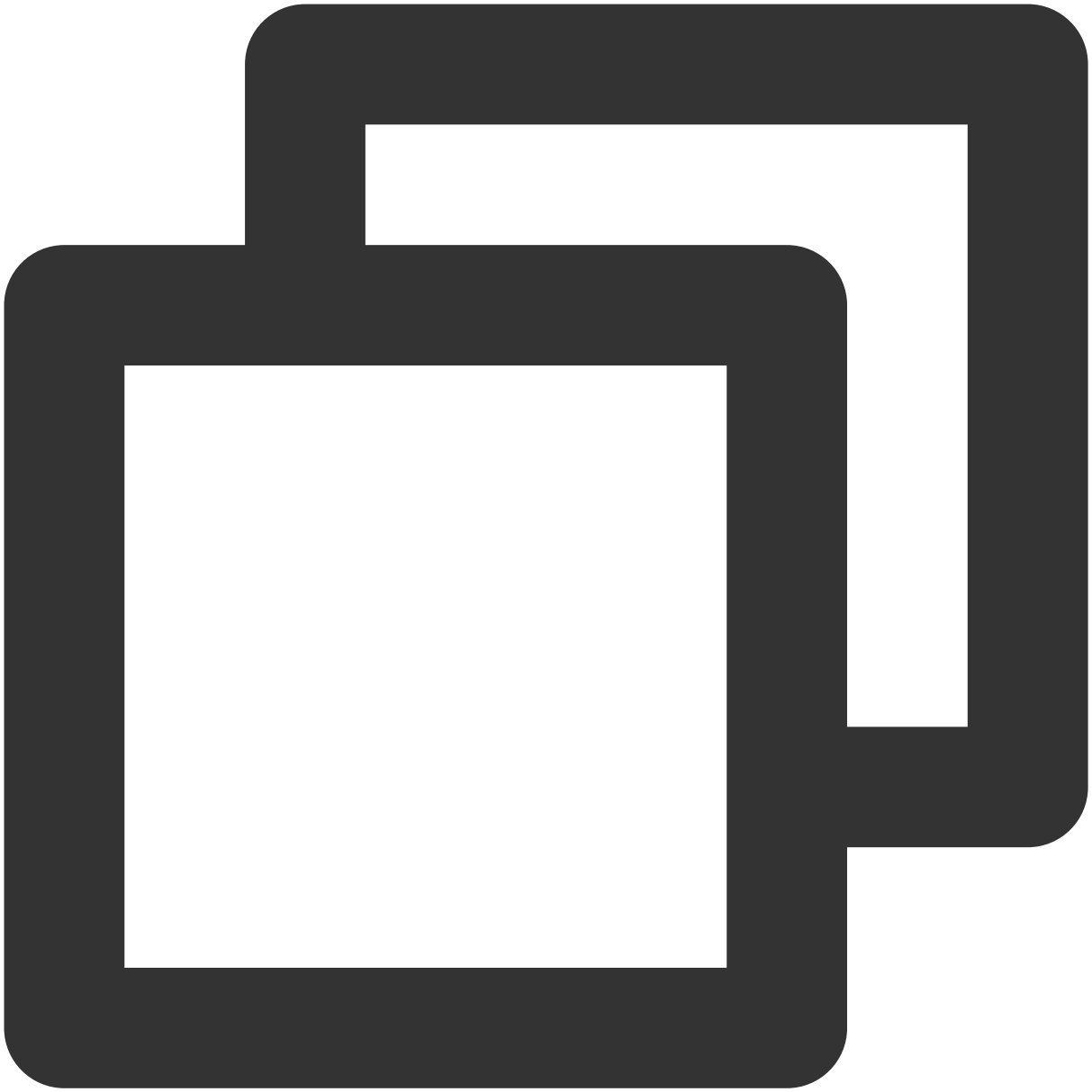
示例：



```
> SELECT to_timestamp('2016-12-31 00:12:00');
 2016-12-31 00:12:00
> SELECT to_timestamp('2016-12-31', 'yyyy-MM-dd');
 2016-12-31 00:00:00
```

TO_DATE

函数语法：



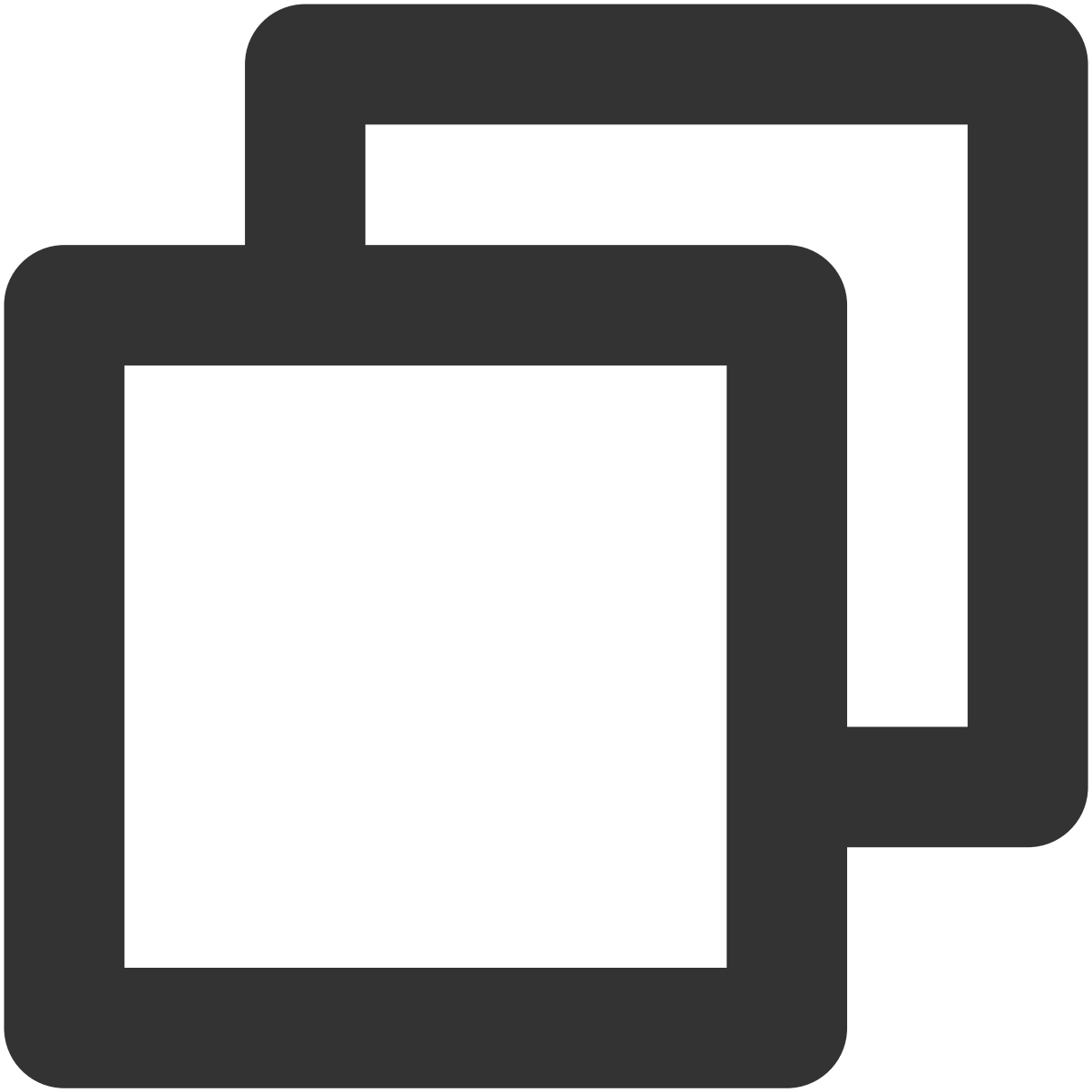
```
TO_DATE(<date_str> string[, <fmt> string])
```

支持引擎：SparkSQL、Presto。

使用说明：将 fmt 格式的 date_str 表达式解析为日期。输入无效则返回 NULL。默认情况下，如果省略 fmt，它将遵循日期的强制转换规则。结果数据类型与配置值一致。

返回类型：date。

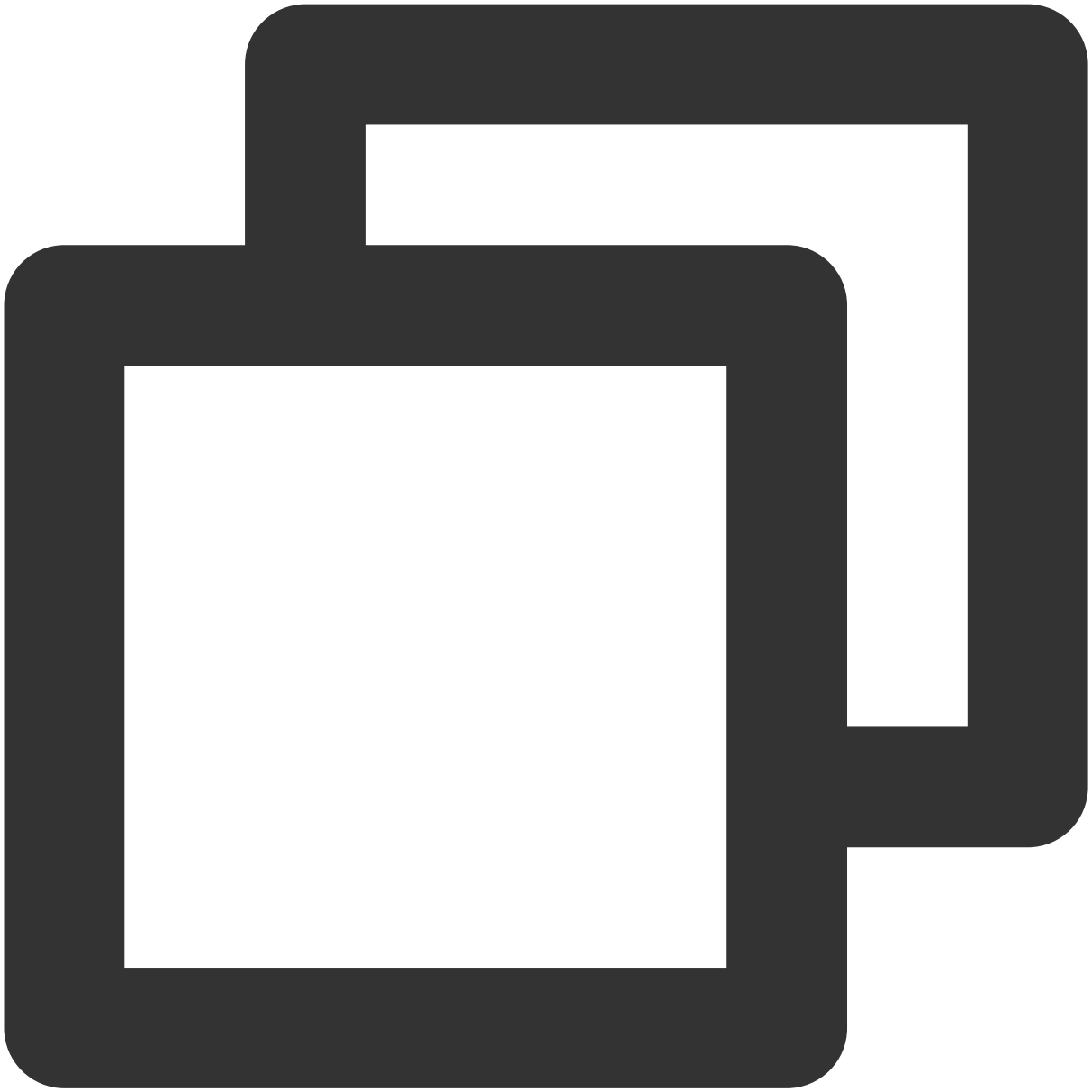
示例：



```
> SELECT to_date('2009-07-30 04:17:52');
 2009-07-30
> SELECT to_date('2016-12-31', 'yyyy-MM-dd');
 2016-12-31
```

TO_UNIX_TIMESTAMP

函数语法：



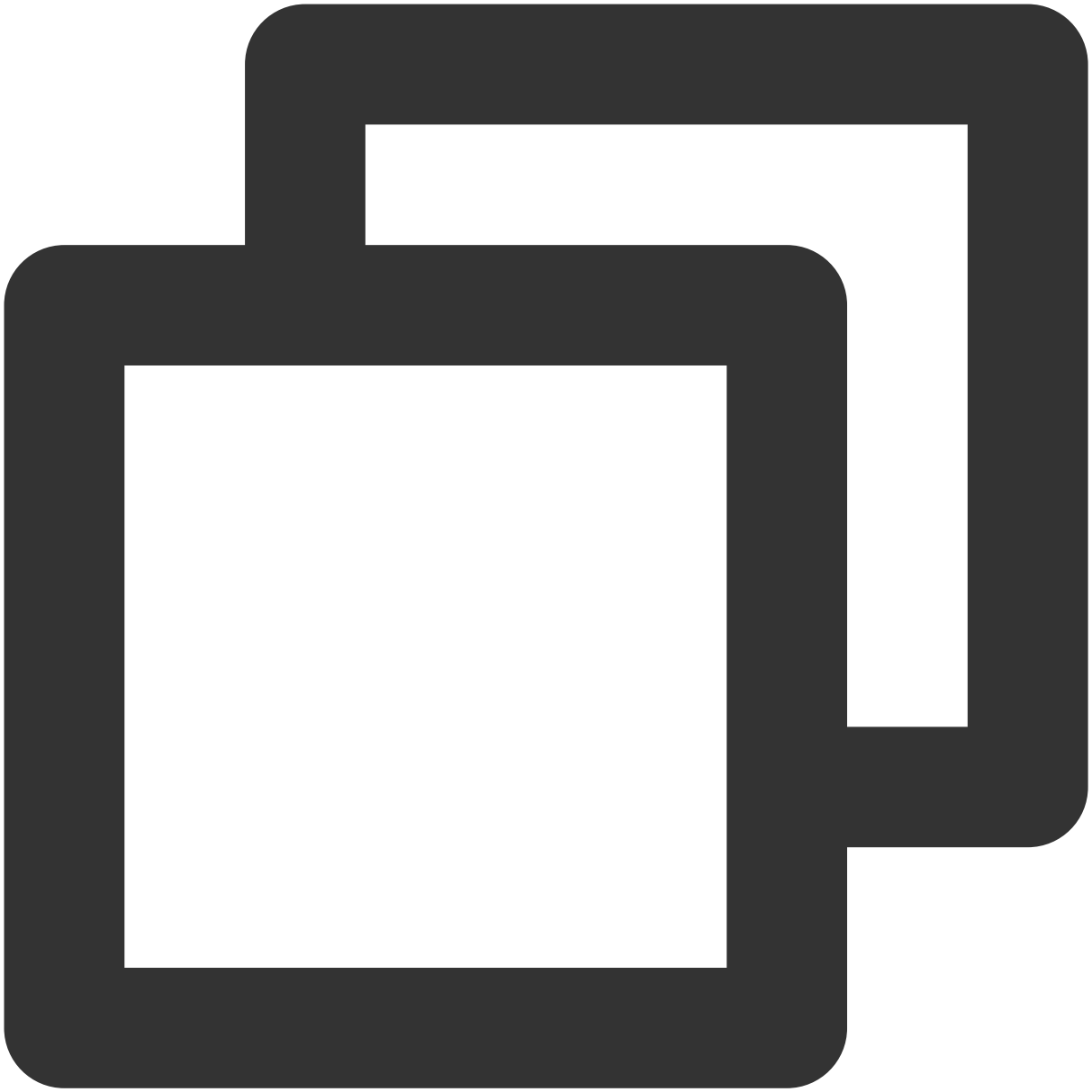
```
TO_UNIX_TIMESTAMP(<ts> date|timestamp|string[, <fmt> string])
```

支持引擎：SparkSQL、Presto。

使用说明：返回 ts 的 unix 时间戳。

返回类型：bigint。

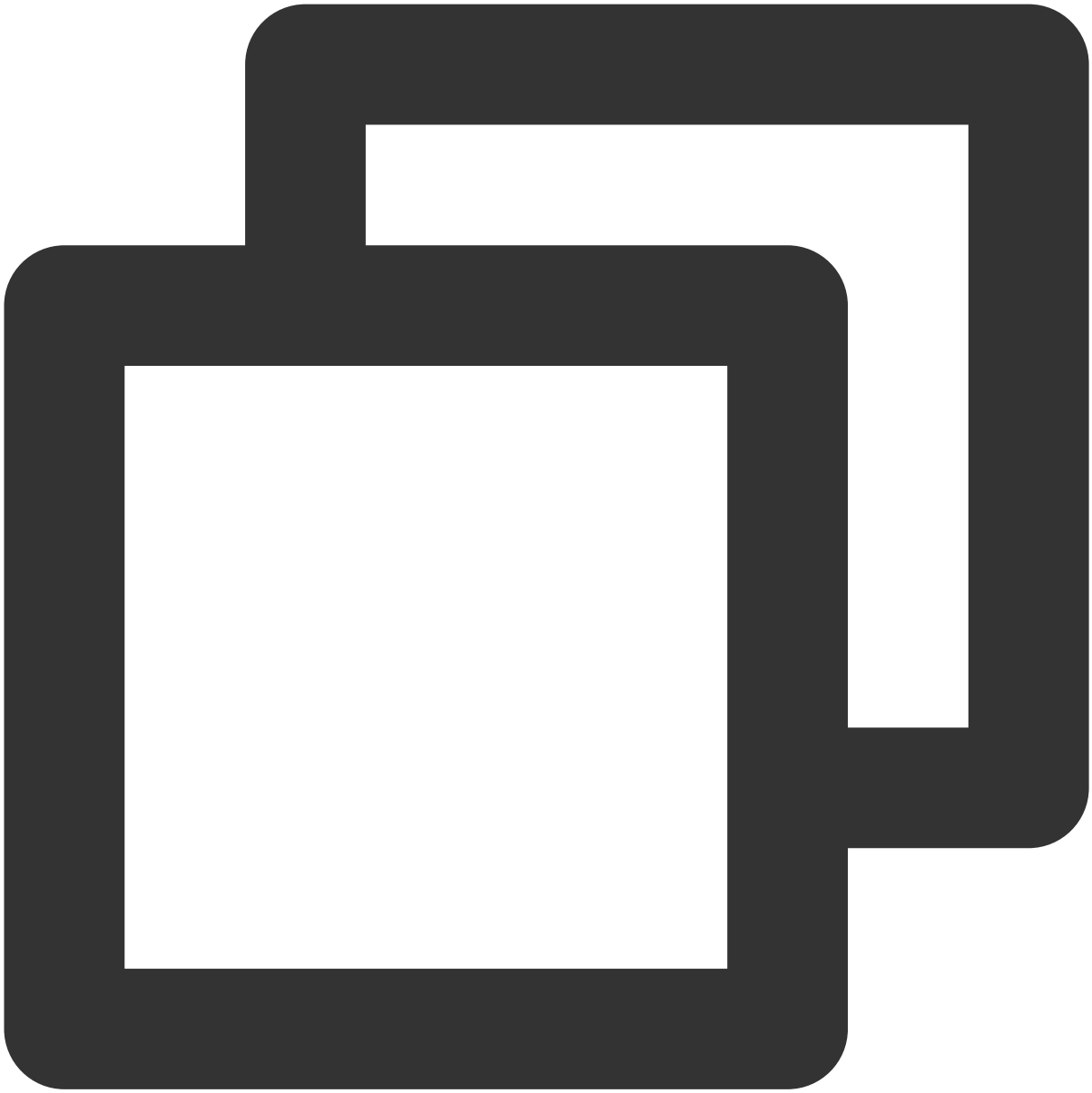
示例：



```
> SELECT to_unix_timestamp('2016-04-08', 'yyyy-MM-dd');
 1460098800
```

TO_UTC_TIMESTAMP

函数语法：



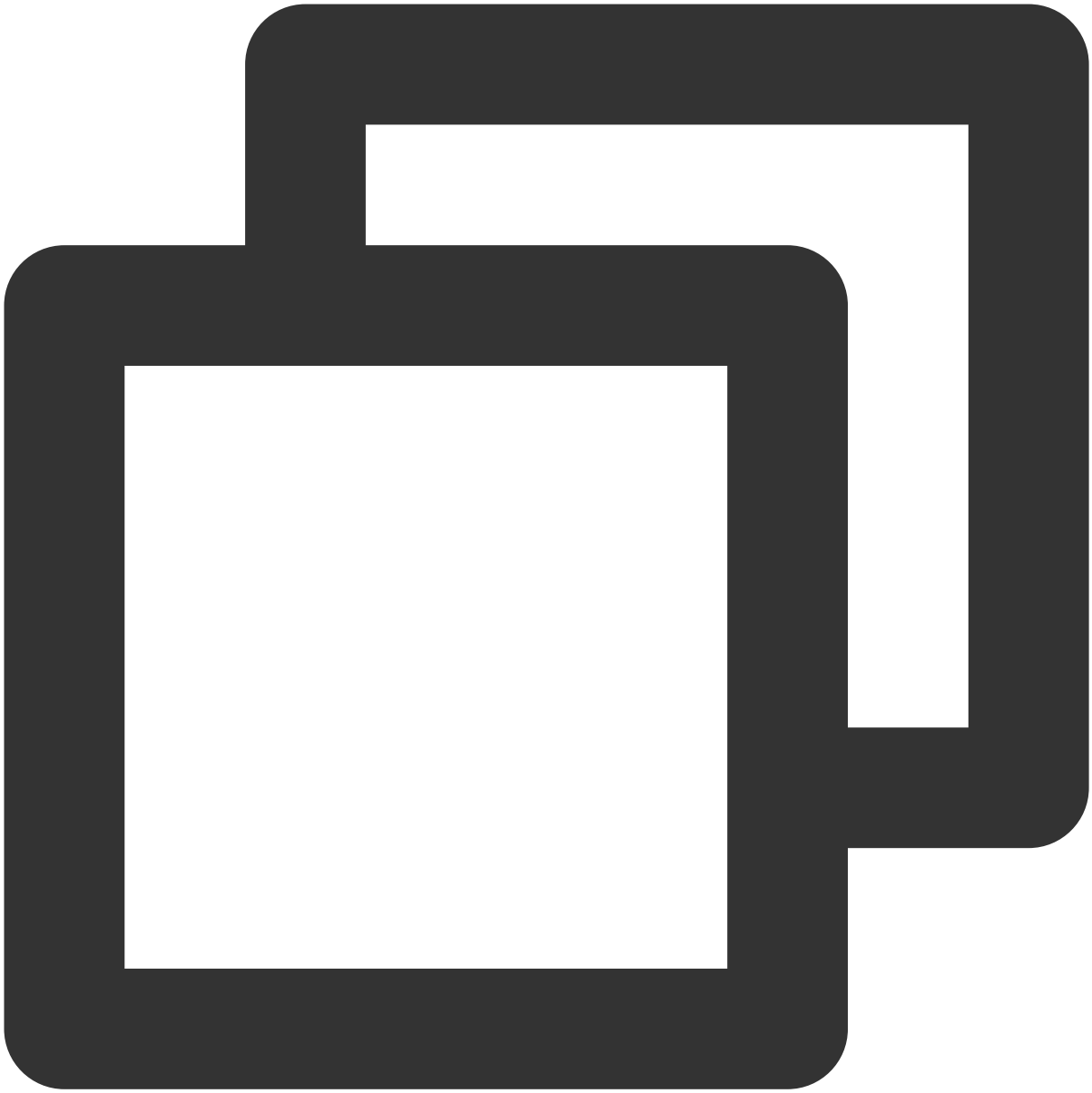
```
-- SparkSQL
TO_UTC_TIMESTAMP(<ts> date|timestamp|string, <timezone> string)

-- Presto
TO_UTC_TIMESTAMP(<ts> date|timestamp|string|interger|double|decimal, <timezone> str
```

支持引擎：SparkSQL、Presto。

使用说明：将给定时区中的时间戳转换为 UTC。

返回类型：timestamp。

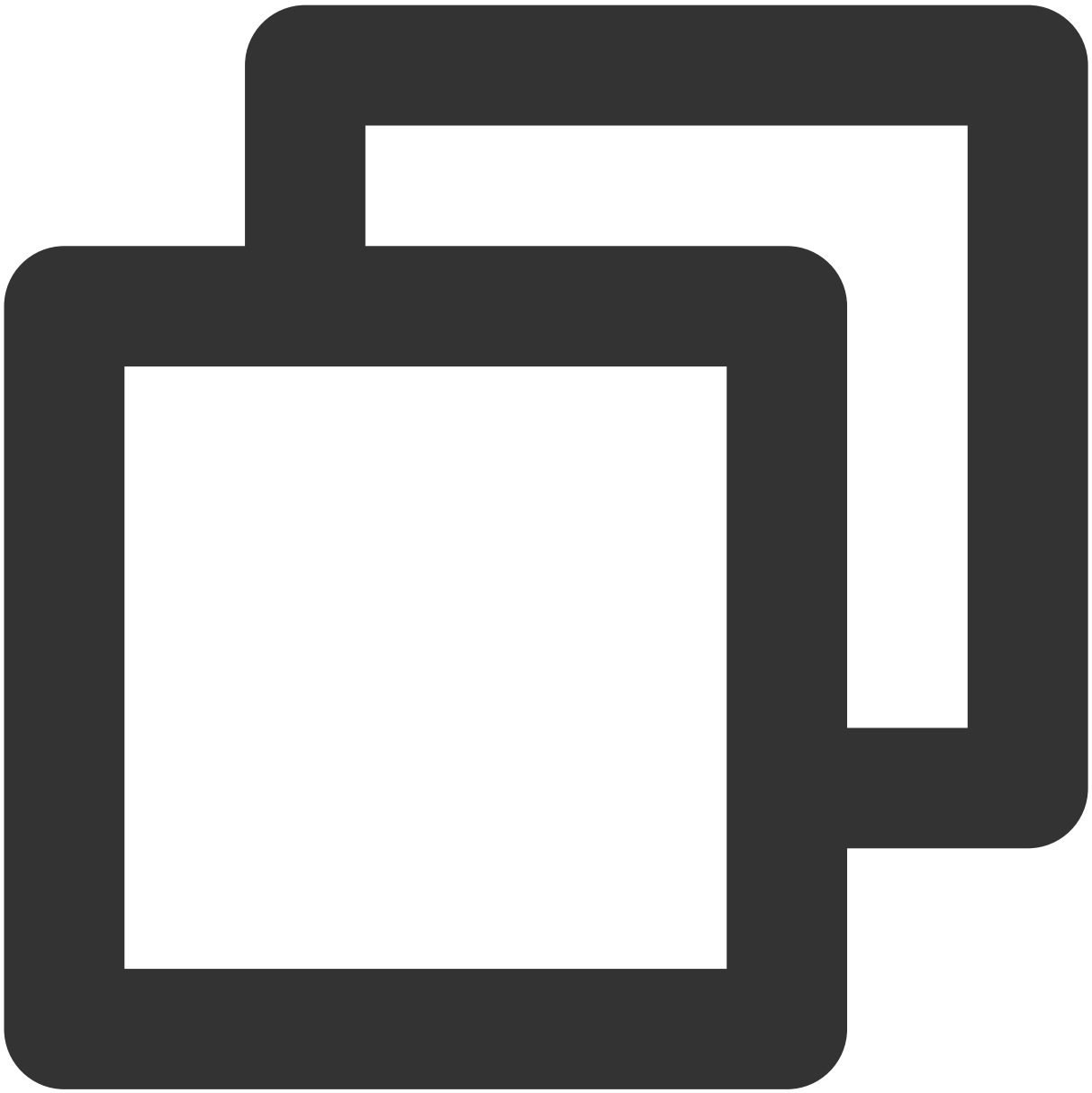
示例：



```
-- SparkSQL
> SELECT to_utc_timestamp('2016-08-31', 'Asia/Seoul');
 2016-08-30 15:00:00
-- Presto
> select to_utc_timestamp(10000, 'UTC');
 1970-01-01 08:00:10
```

TRUNC

函数语法：



```
TRUNC(<d> date|string, <fmt> string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回将日期 d 按照 fmt 指定的时间单位进行截取后的日期值。

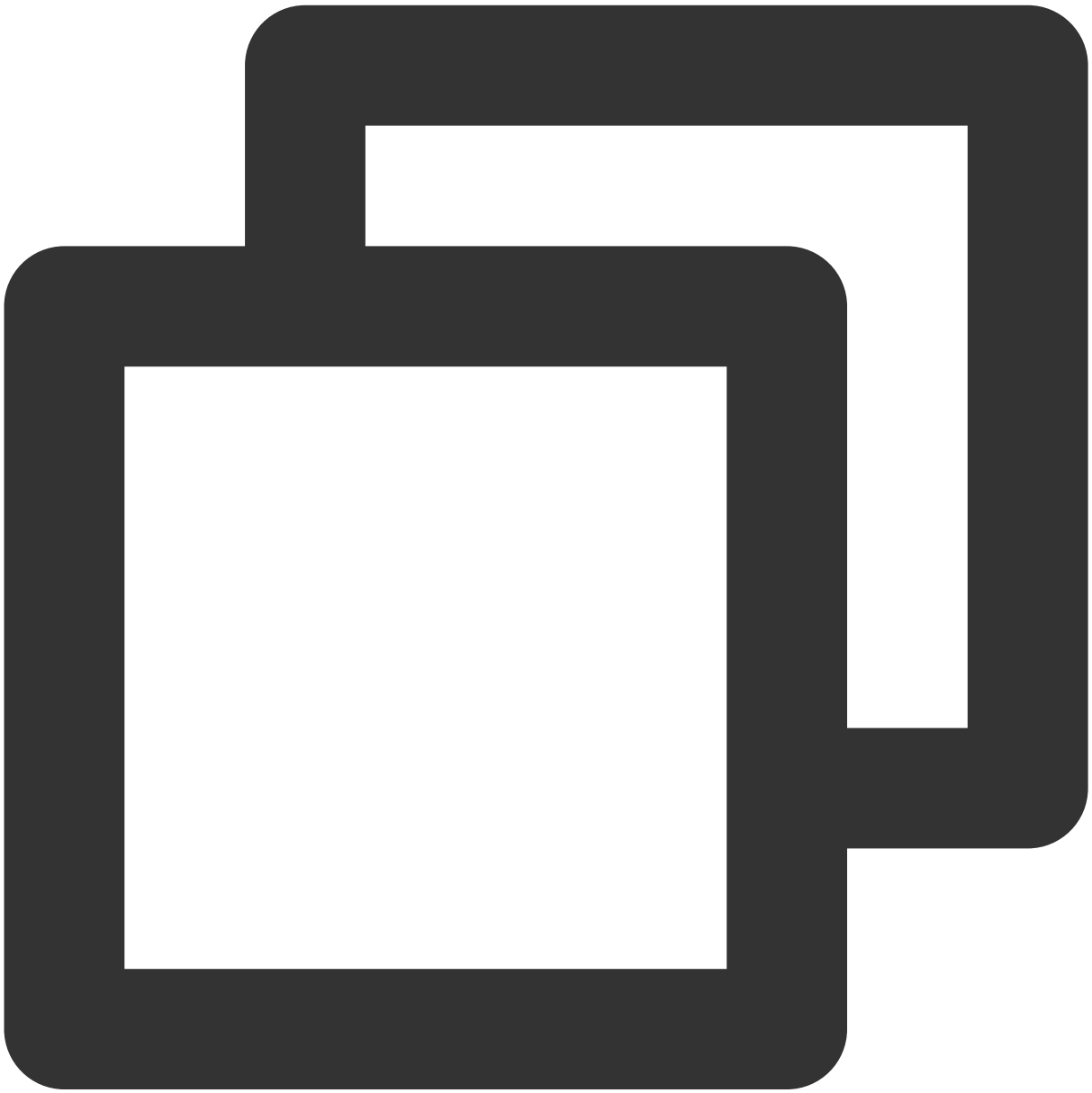
返回类型：date。

示例：

```
> SELECT trunc('2019-08-04', 'week');
 2019-07-29
> SELECT trunc('2019-08-04', 'quarter');
 2019-07-01
> SELECT trunc('2009-02-12', 'MM');
 2009-02-01
> SELECT trunc('2015-10-27', 'YEAR');
 2015-01-01
```

DATE_TRUNC

函数语法：



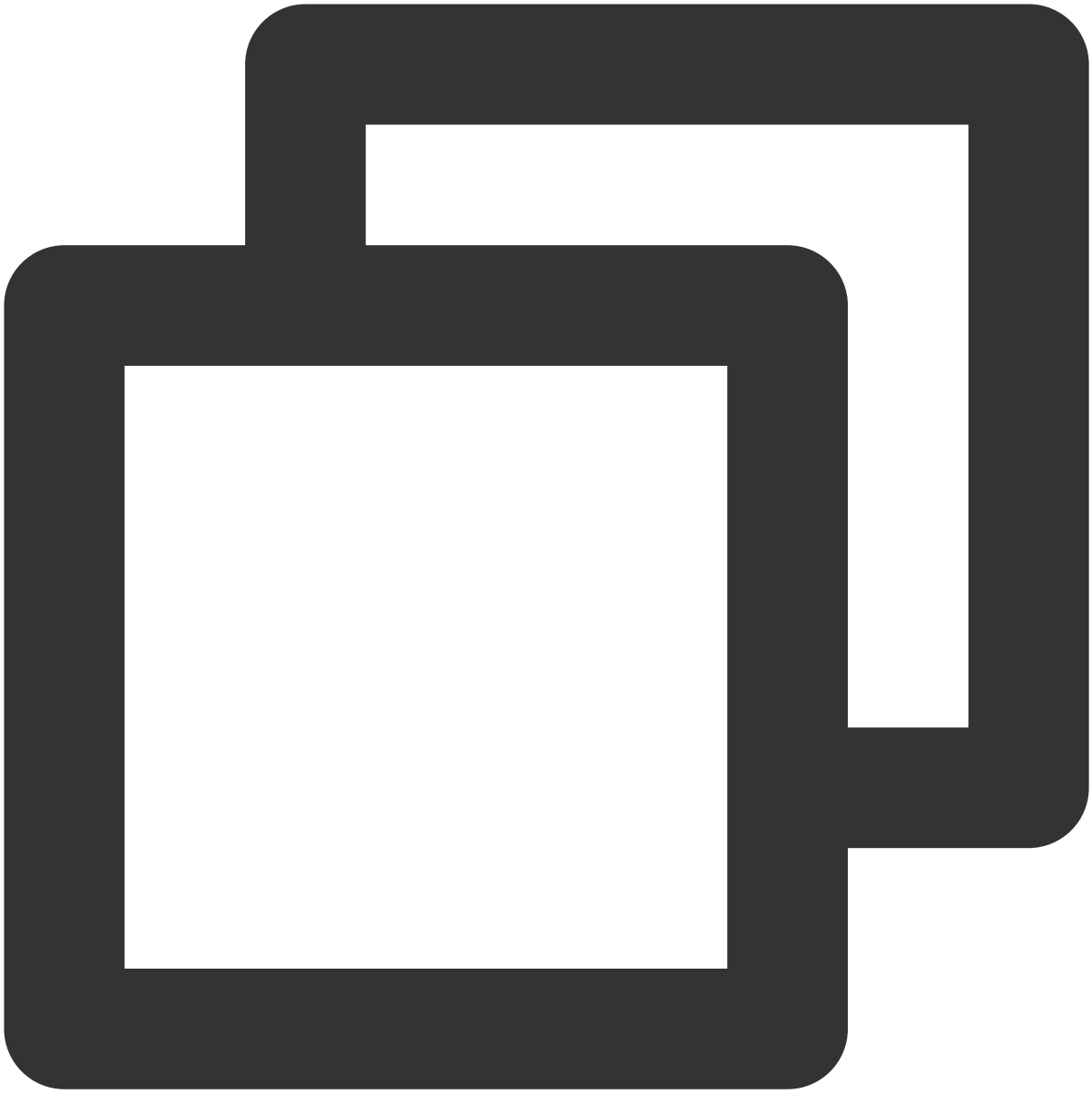
```
DATE_TRUNC(<fmt> string, <ts> date|timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回将时间戳 ts 按照 fmt 截断后的时间戳 。

返回类型：timestamp。

示例：

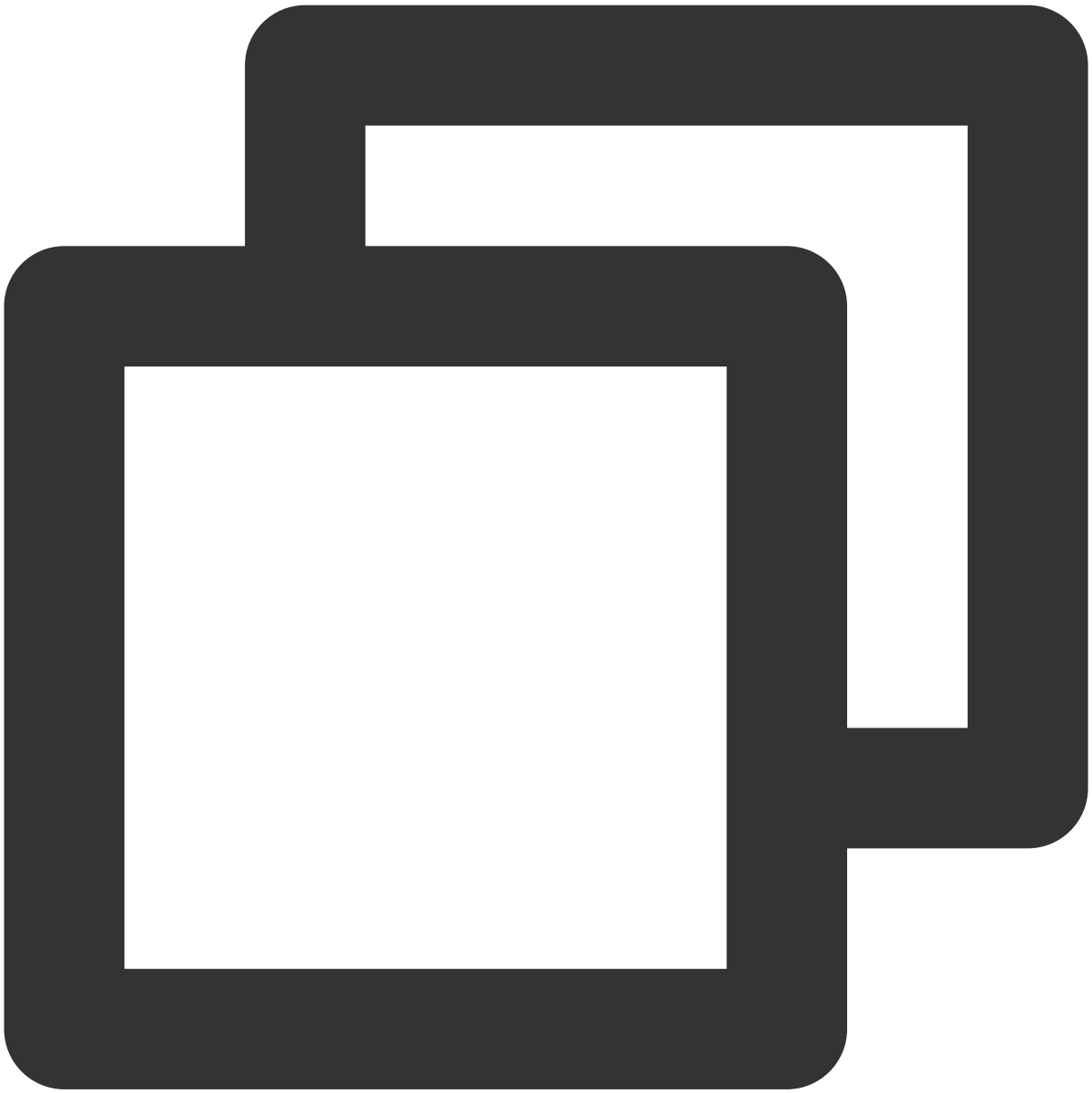


```
> SELECT date_trunc('YEAR', '2015-03-05T09:32:05.359');
 2015-01-01 00:00:00
> SELECT date_trunc('MM', '2015-03-05T09:32:05.359');
 2015-03-01 00:00:00
> SELECT date_trunc('DD', '2015-03-05T09:32:05.359');
 2015-03-05 00:00:00
> SELECT date_trunc('HOUR', '2015-03-05T09:32:05.359');
 2015-03-05 09:00:00
> SELECT date_trunc('MILLISECOND', '2015-03-05T09:32:05.123456');
 2015-03-05 09:32:05.123
```

版权所有：腾讯云计算（北京）有限责任公司

UNIX_TIMESTAMP

函数语法：



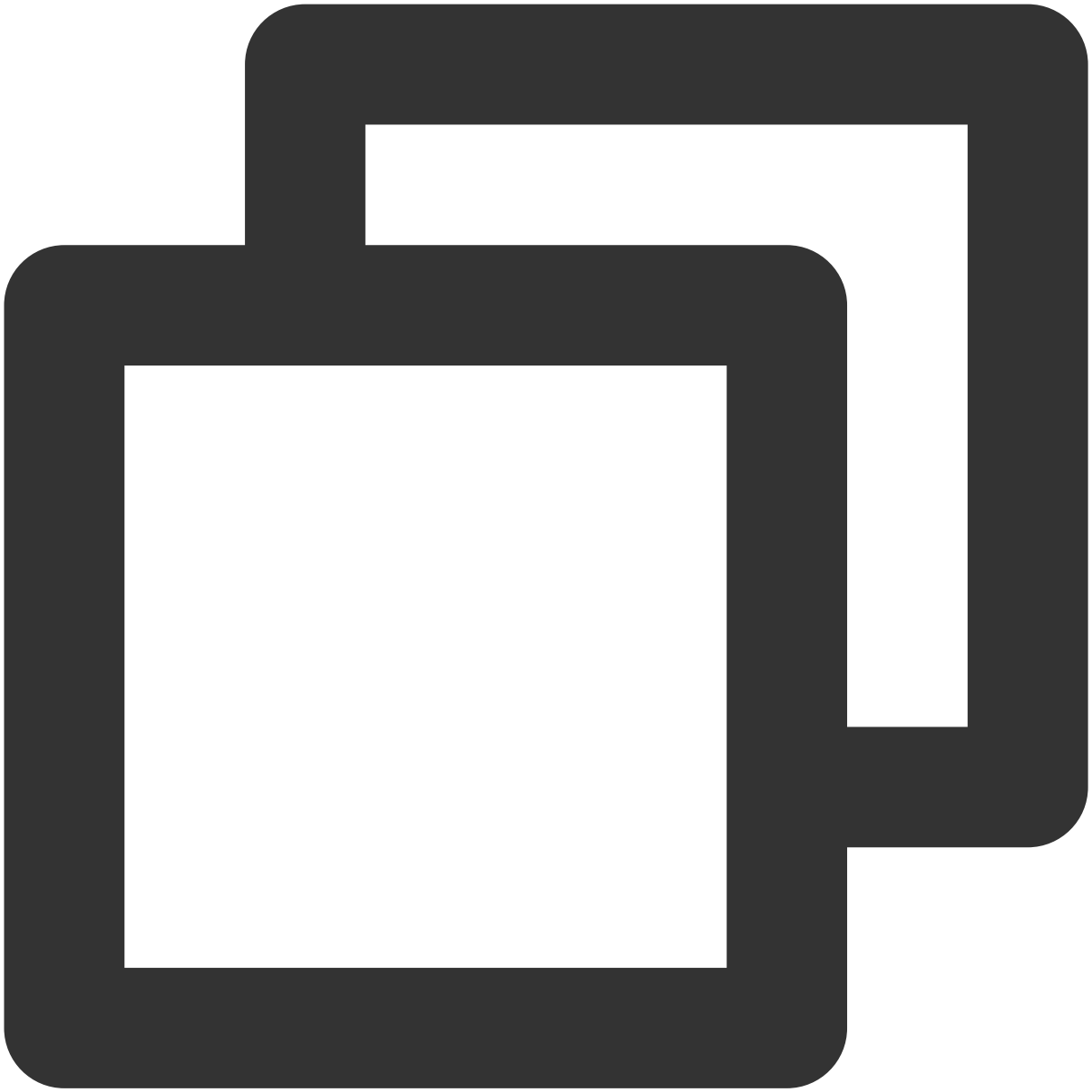
```
UNIX_TIMESTAMP([<ts> date|timestamp|string[, fmt]])
```

支持引擎：SparkSQL、Presto。

使用说明：返回当前或指定时间的 UNIX 时间戳。

返回类型：bigint。

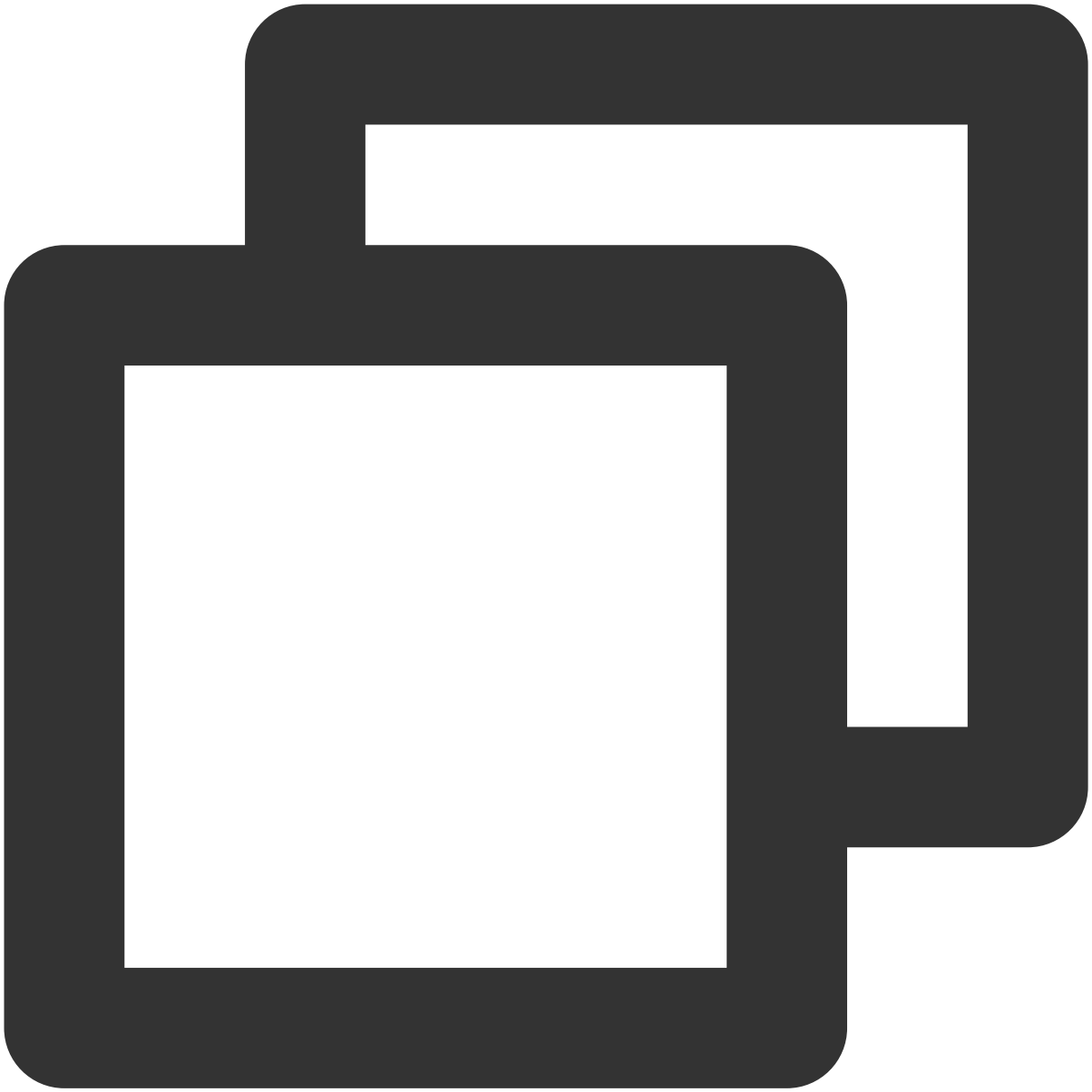
示例：



```
> SELECT unix_timestamp();
 1476884637
> SELECT unix_timestamp('2016-04-08', 'yyyy-MM-dd');
 1460041200
```

DAYOFWEEK

函数语法：



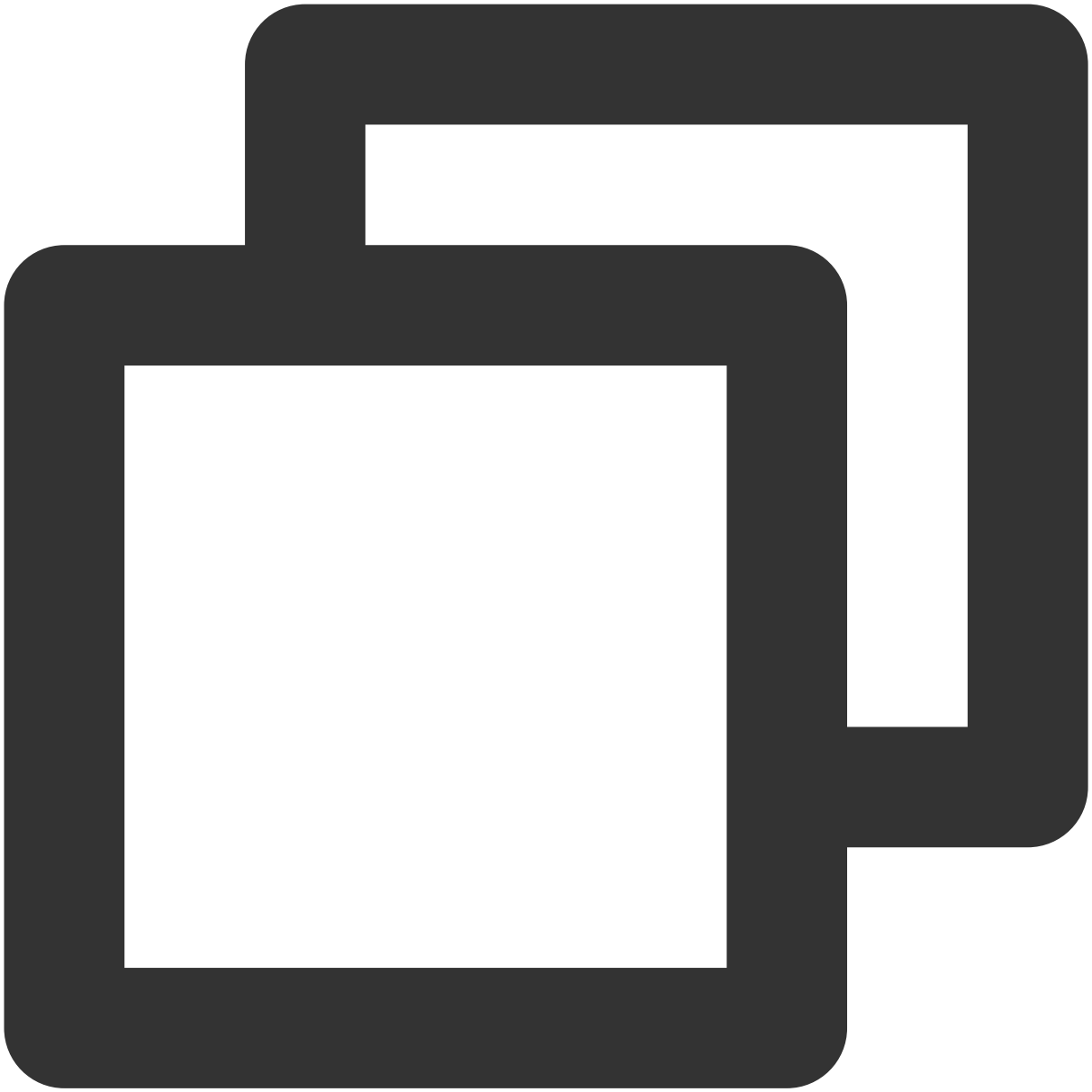
```
DAYOFWEEK(<d> date|timestamp|string)
```

支持引擎：parkSQL、Presto。

使用说明：回日期/时间戳“d”是星期几。

返回类型：nteger。

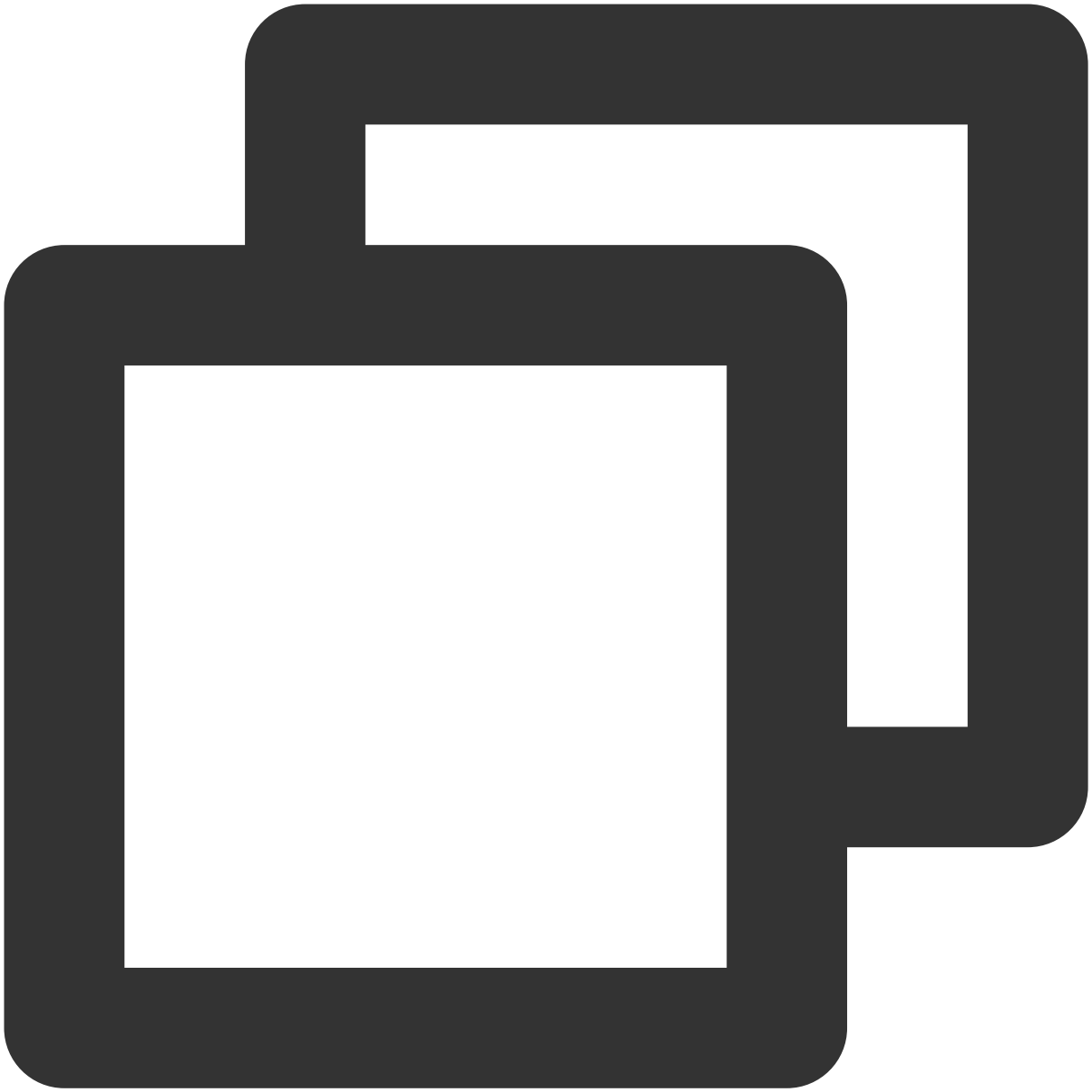
示例：



```
> SELECT dayofweek('2009-07-30');
 5
```

WEEKDAY

函数语法：



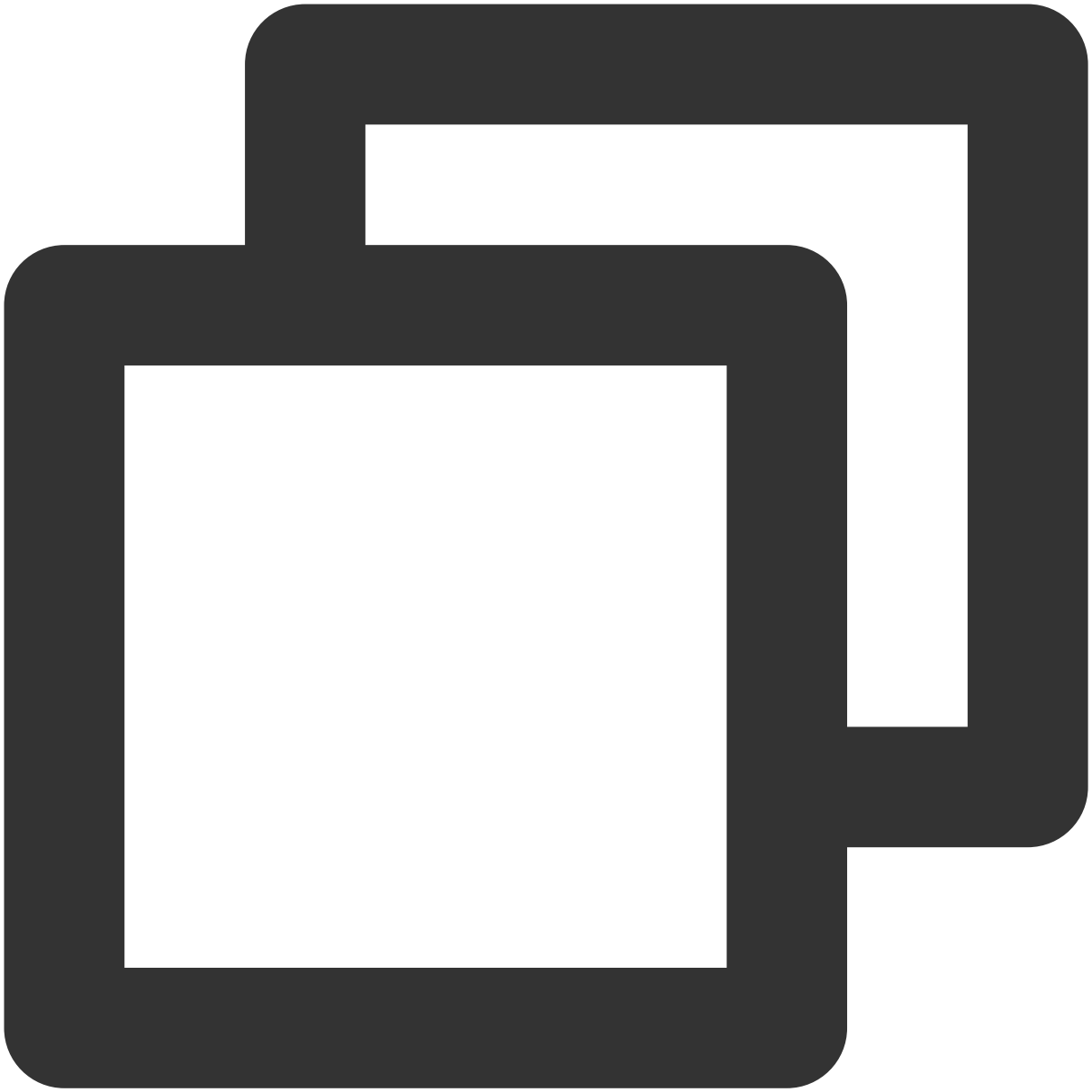
```
WEEKDAY(<d> date|timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回日期/时间戳“d”是星期几。

返回类型：integer。

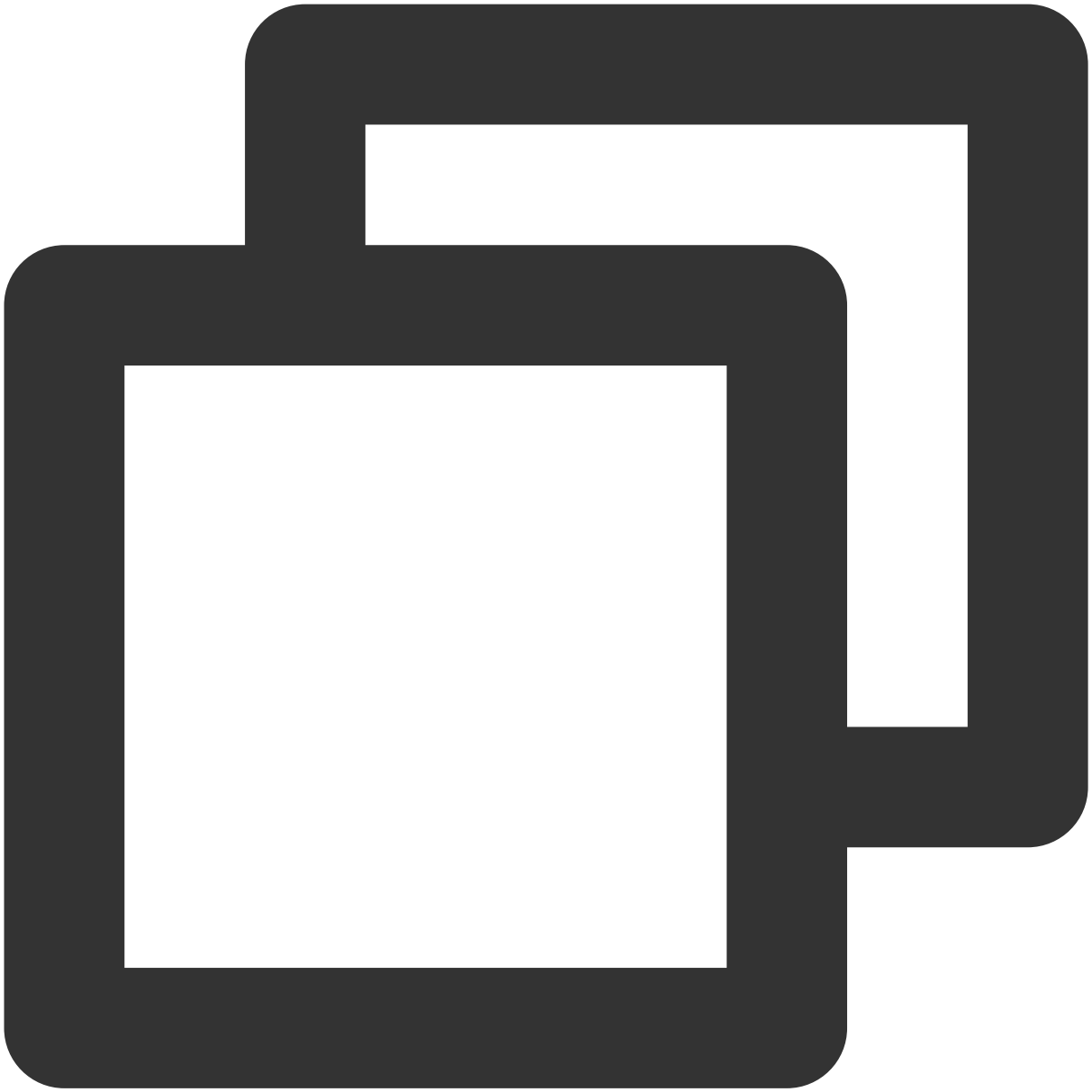
示例：



```
> SELECT weekday('2009-07-30');
 3
```

WEEKOFYEAR

函数语法：



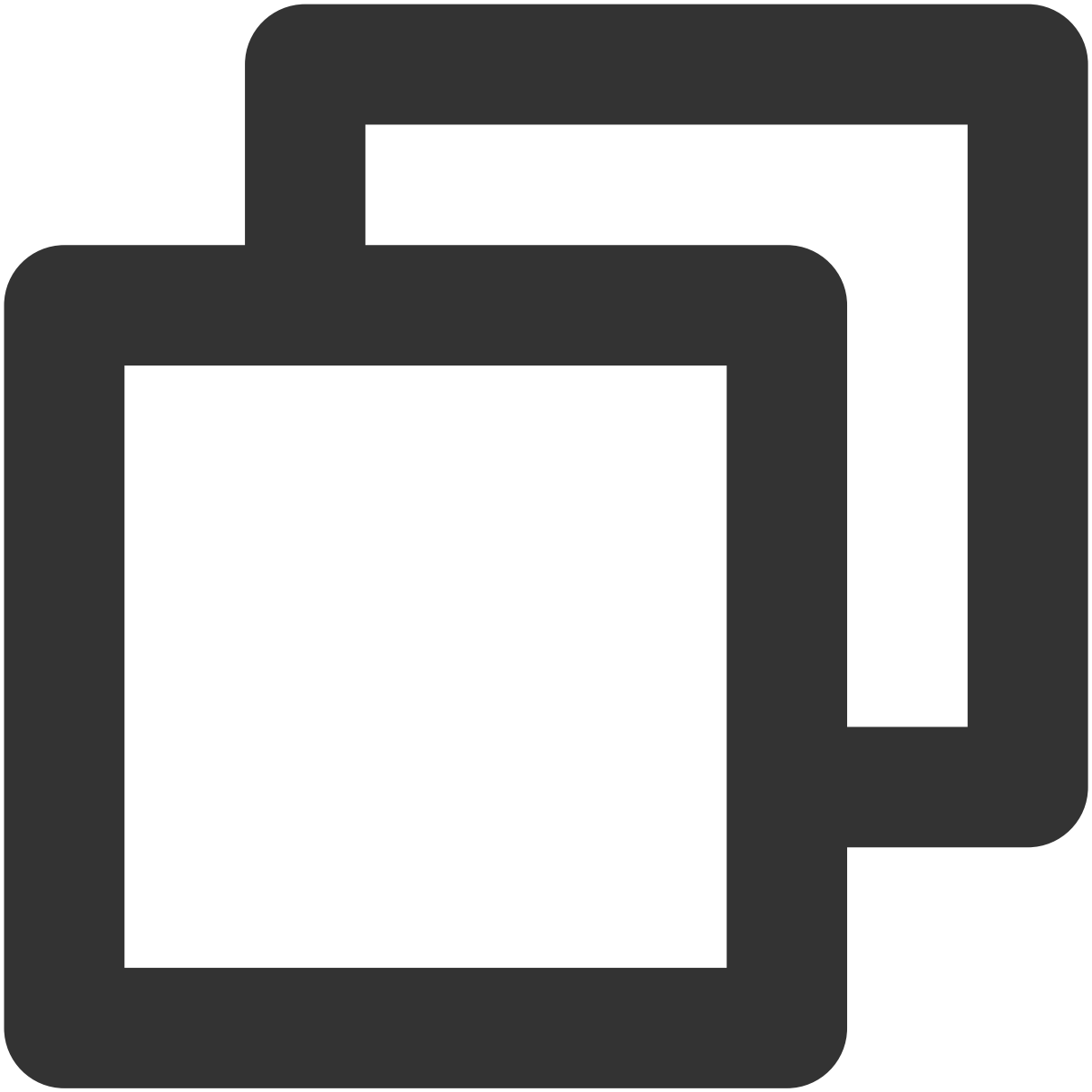
```
WEEKOFYEAR(<d> date|timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回给定日期“d”是一年中的第几周。

返回类型：integer。

示例：



```
> SELECT weekofyear('2008-02-20');
 8
```

YEAR

函数语法：

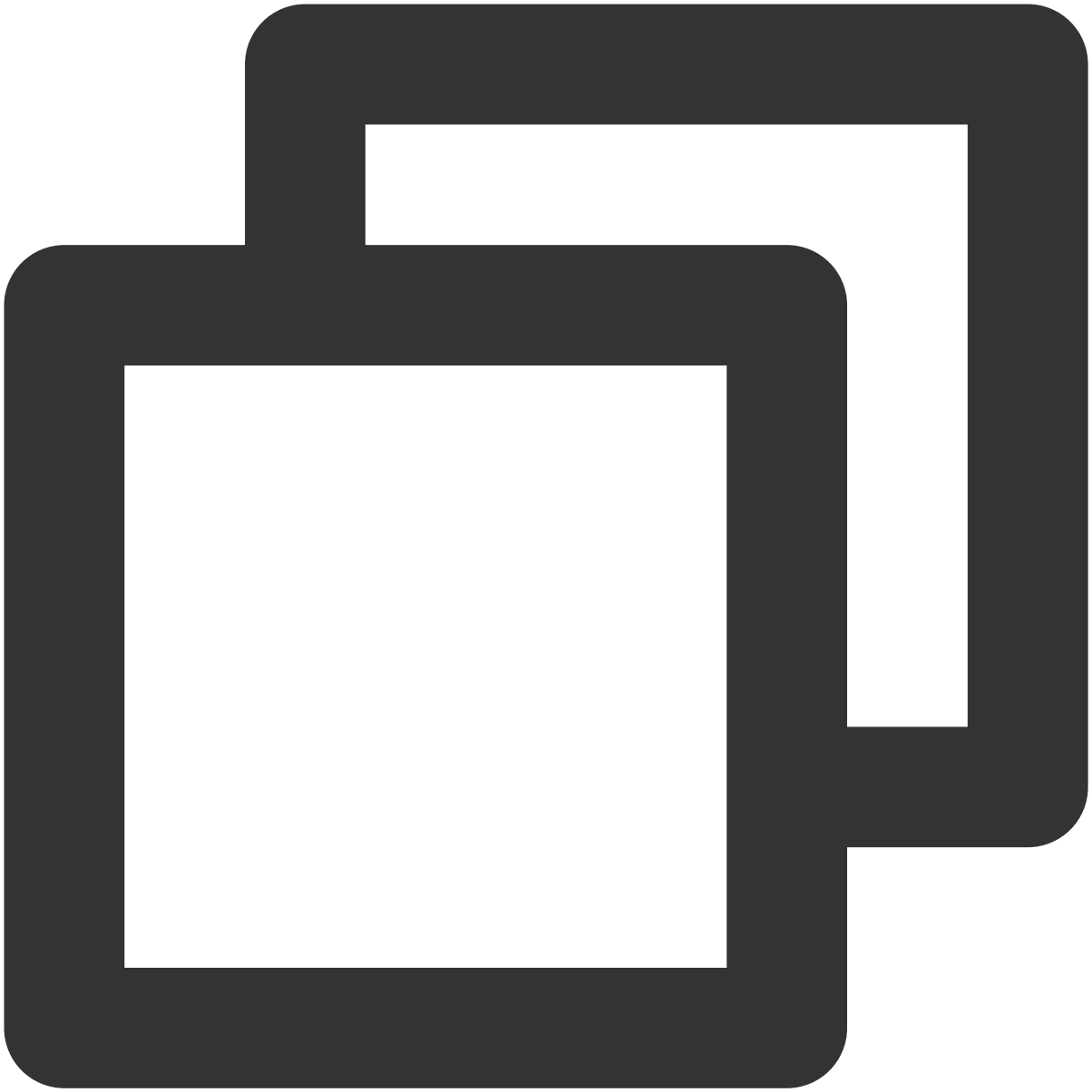
```
YEAR(<d> date|timestamp|string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回日期/时间戳“d”中的年份。

返回类型：integer。

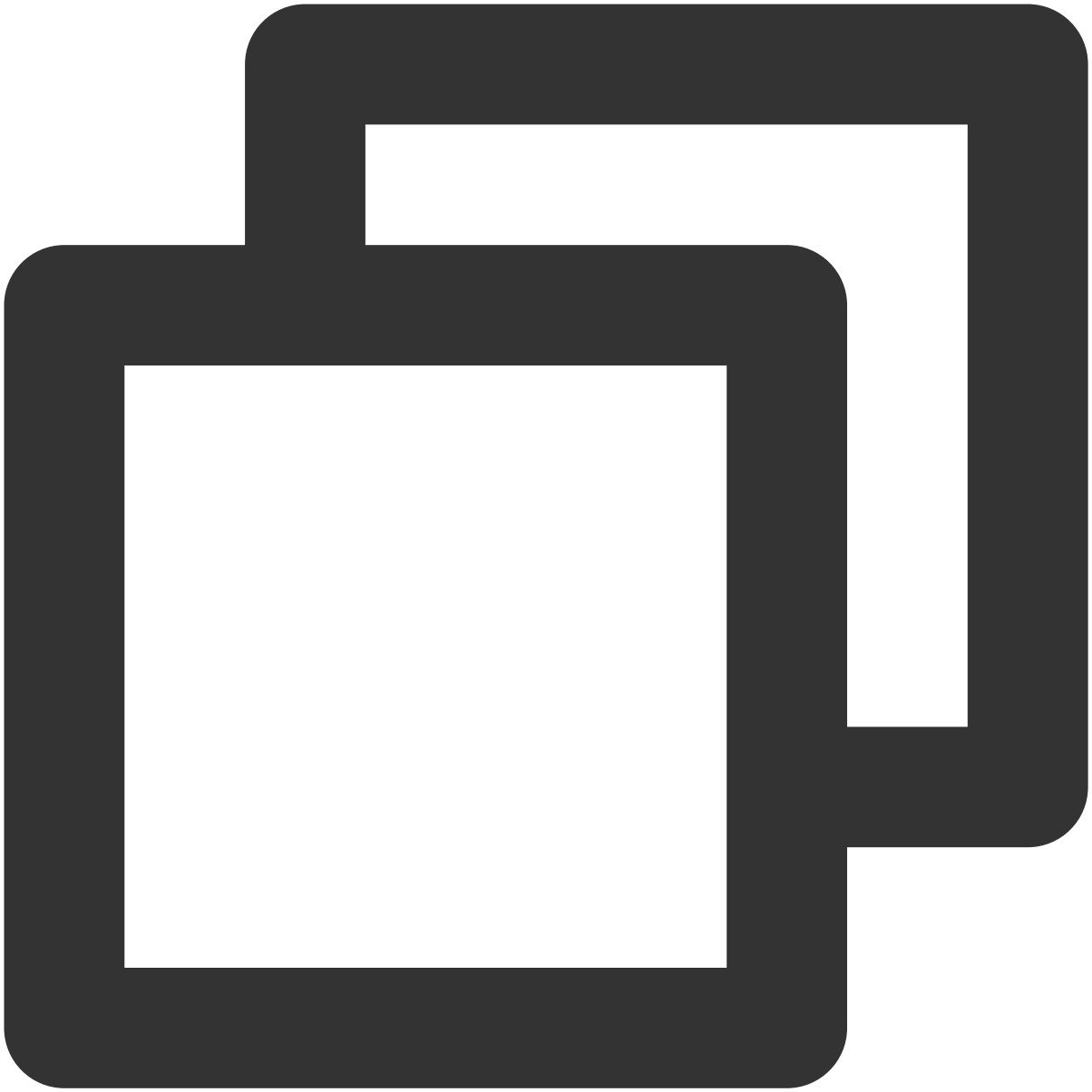
示例：



```
> SELECT year('2016-07-30');
 2016
```

MAKE_DATE

函数语法：



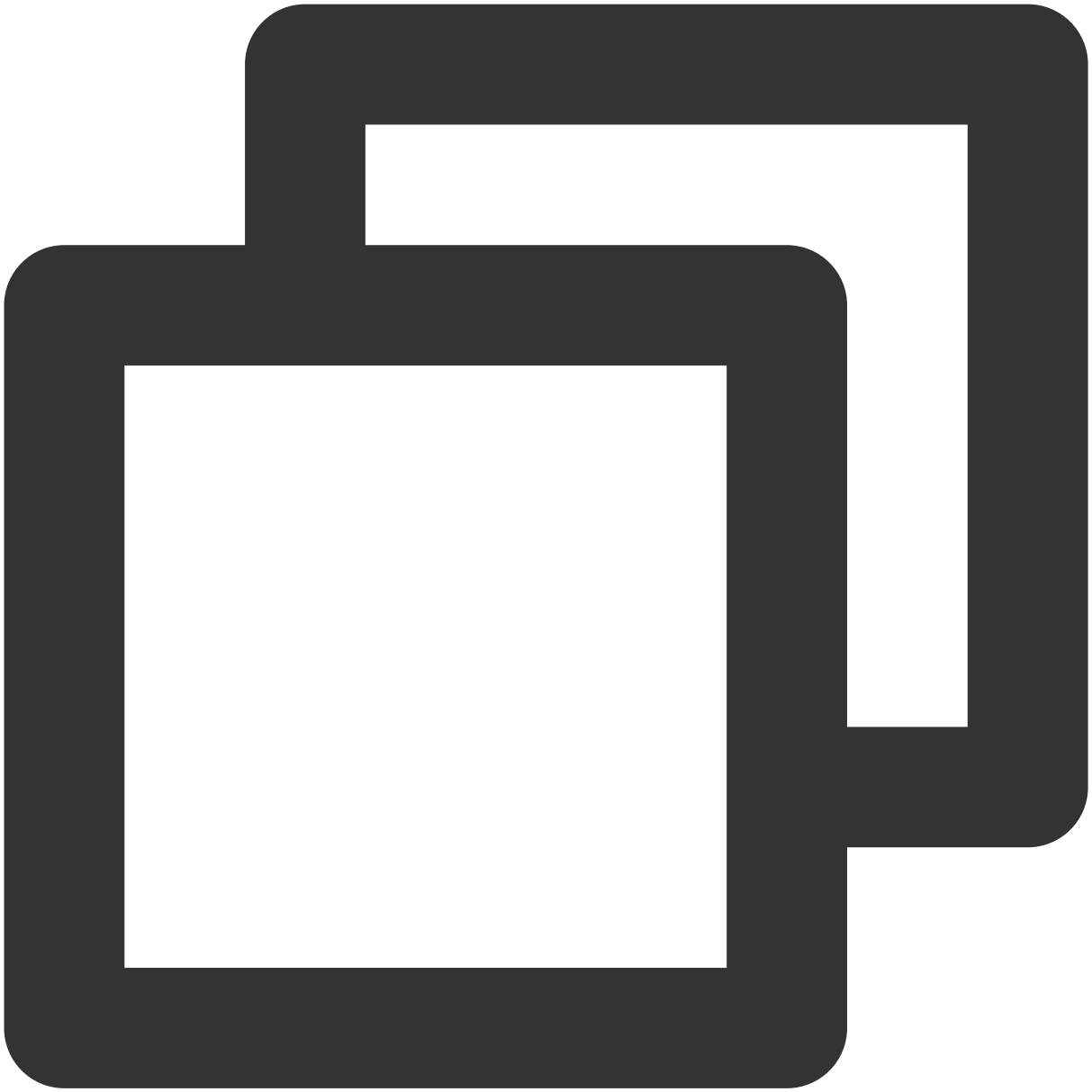
```
MAKE_DATE(<year> integer, <month> integer, <day> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：从 year、month 和 day 字段创建日期。

返回类型：date。

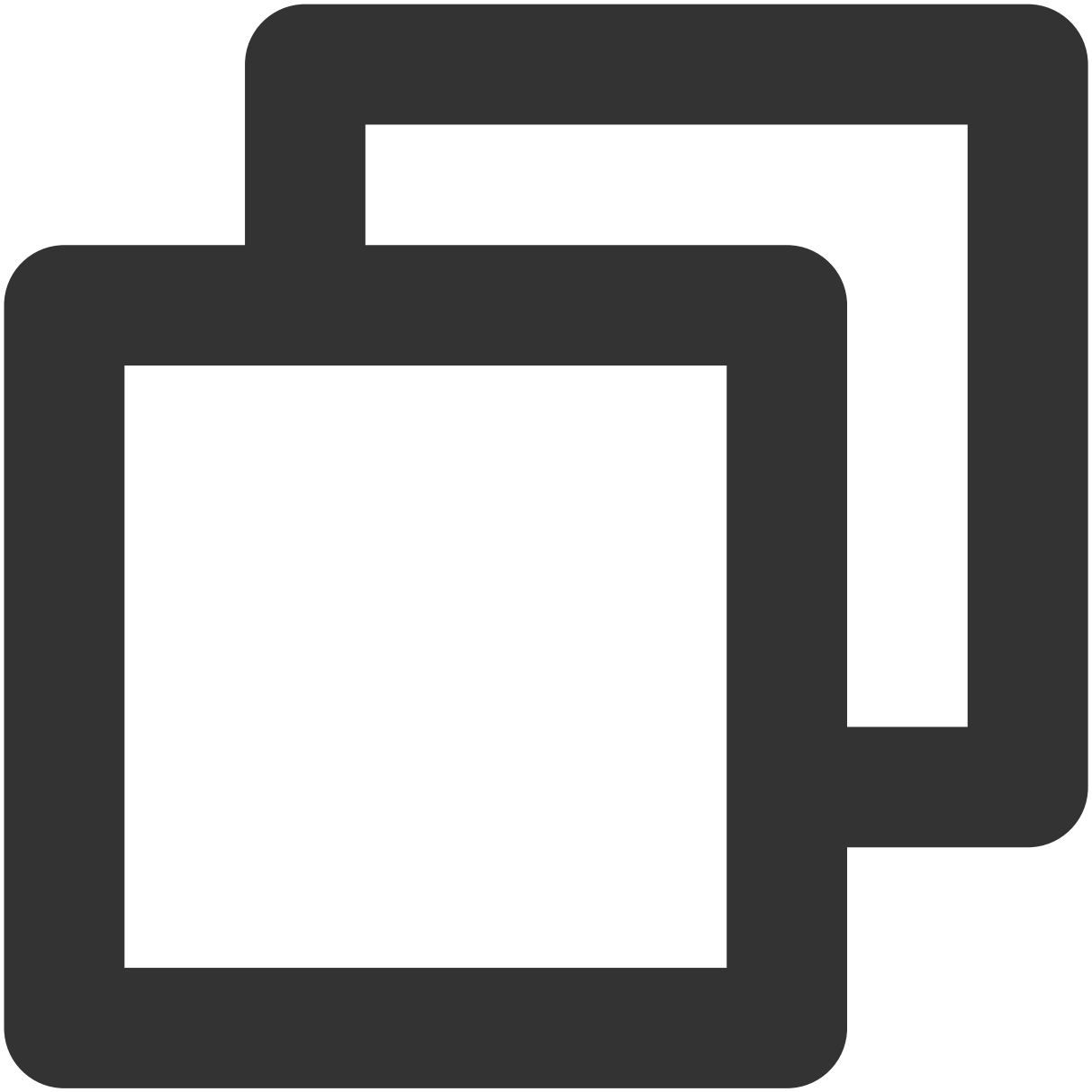
示例：



```
> SELECT make_date(2013, 7, 15);
 2013-07-15
> SELECT make_date(2019, 7, NULL);
 NULL
```

MAKE_TIMESTAMP

函数语法：



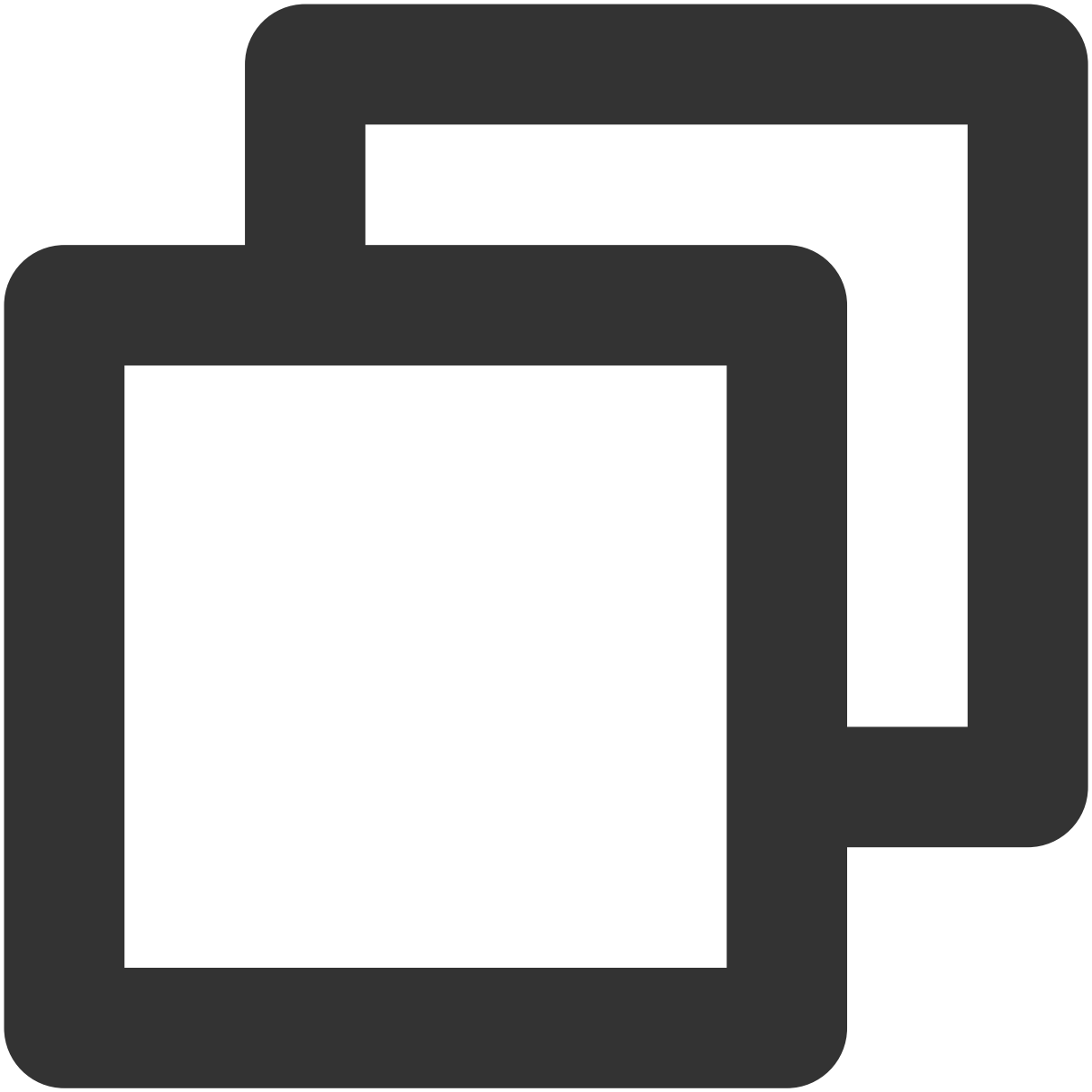
```
MAKE_TIMESTAMP(<year> integer, <month> integer, <day> integer, <hour> integer, <min
```

支持引擎：SparkSQL、Presto。

使用说明：根据给定字段创建时间戳。

返回类型：timestamp。

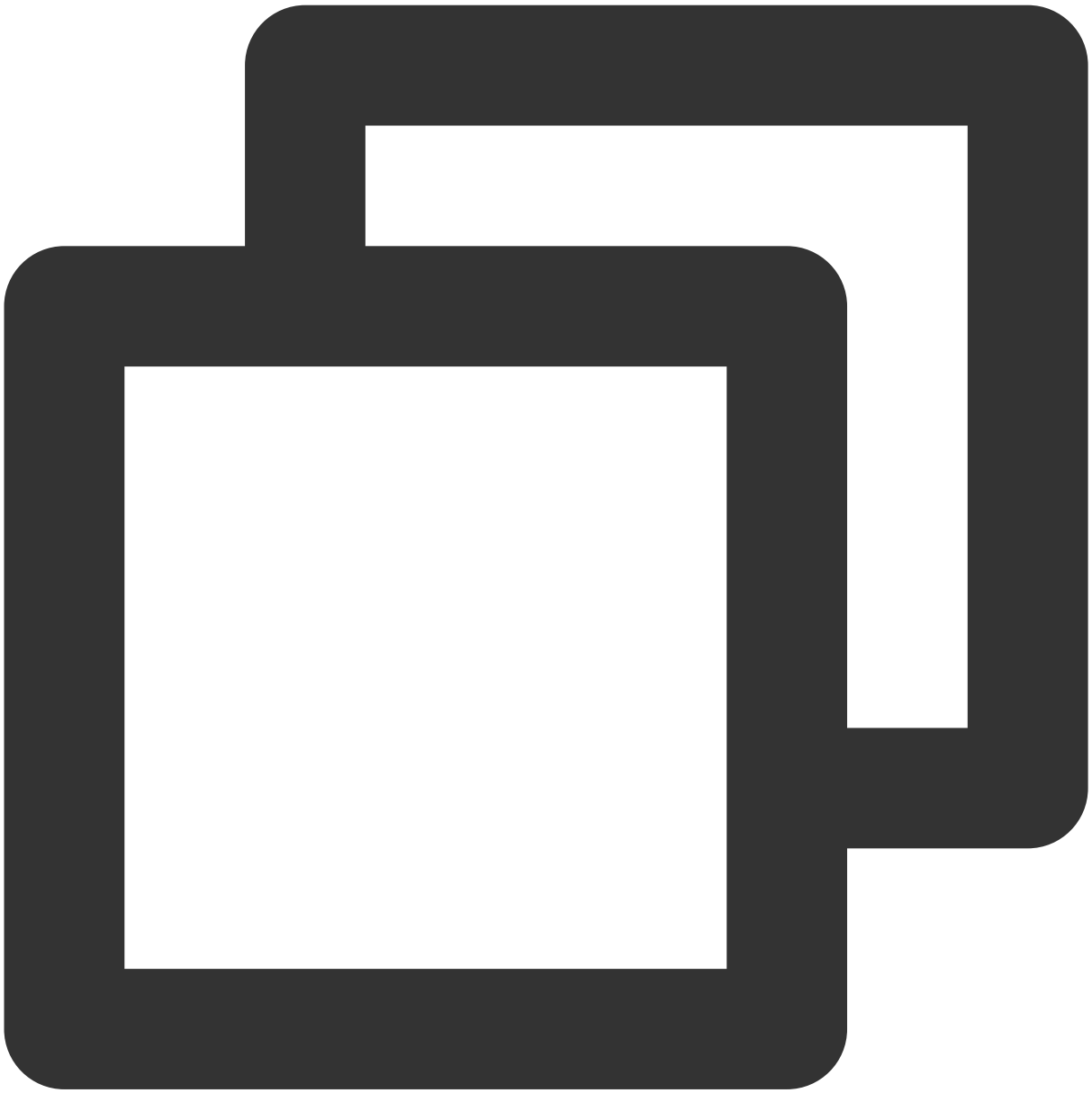
示例：



```
> SELECT make_timestamp(2014, 12, 28, 6, 30, 45.887);
 2014-12-28 06:30:45.887
> SELECT make_timestamp(2014, 12, 28, 6, 30, 45.887, 'CET');
 2014-12-27 21:30:45.887
> SELECT make_timestamp(2019, 6, 30, 23, 59, 60);
 2019-07-01 00:00:00
> SELECT make_timestamp(2019, 6, 30, 23, 59, 1);
 2019-06-30 23:59:01
> SELECT make_timestamp(null, 7, 22, 15, 30, 0);
 NULL
```

DATE_PART

函数语法：



```
DATE_PART(<field> string, <source> date|timestamp)
```

支持引擎：SparkSQL、Presto。

使用说明：提取日期/时间戳的一部分。

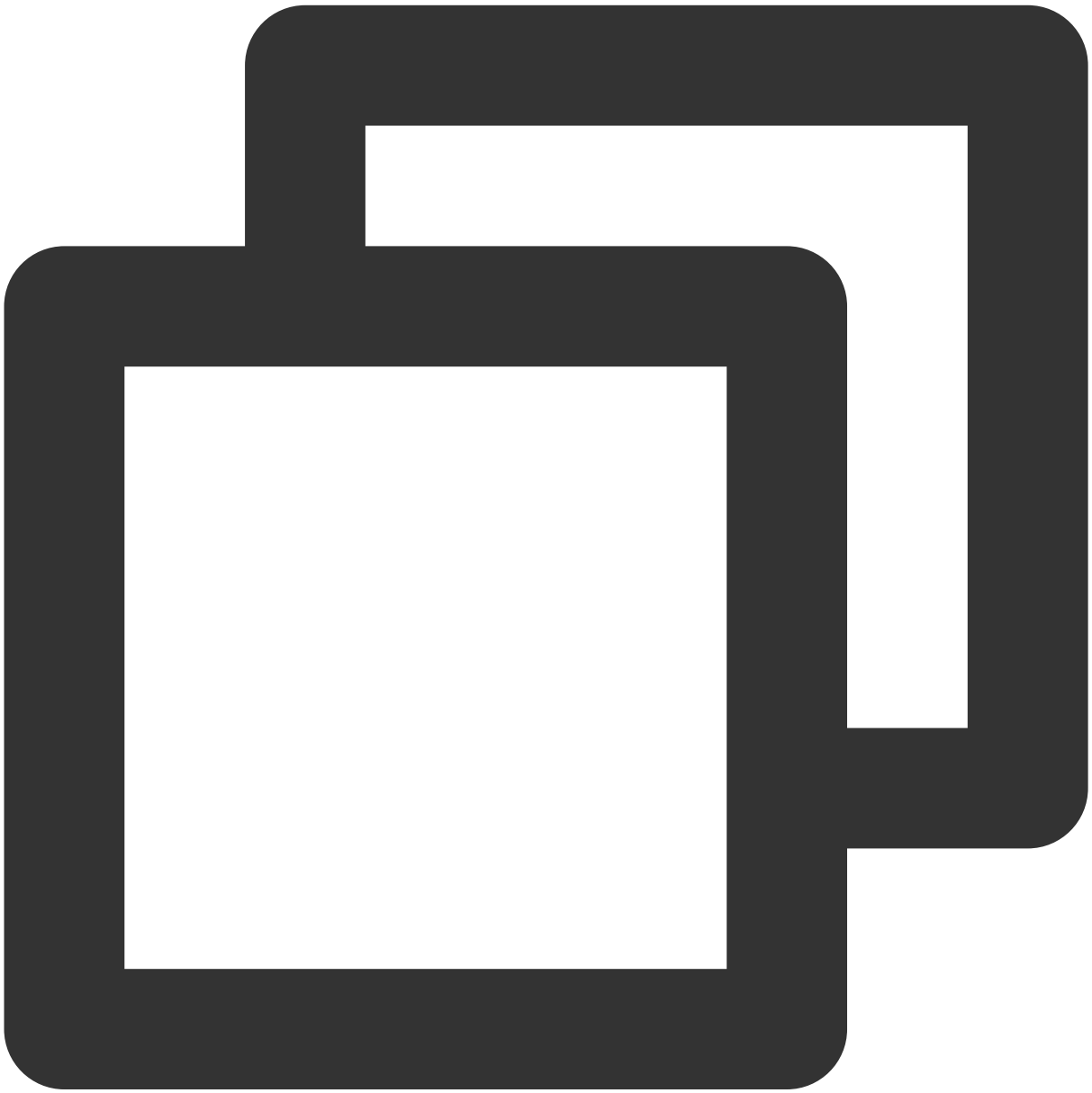
返回类型：integer|double。

示例：

```
> SELECT date_part('YEAR', TIMESTAMP '2019-08-12 01:00:00.123456');
 2019
> SELECT date_part('week', timestamp '2019-08-12 01:00:00.123456');
 33
> SELECT date_part('doy', DATE'2019-08-12');
 224
> SELECT date_part('SECONDS', timestamp'2019-10-01 00:00:01.000001');
 1.000001
```

DATE_FROM_UNIX_DATE

函数语法：



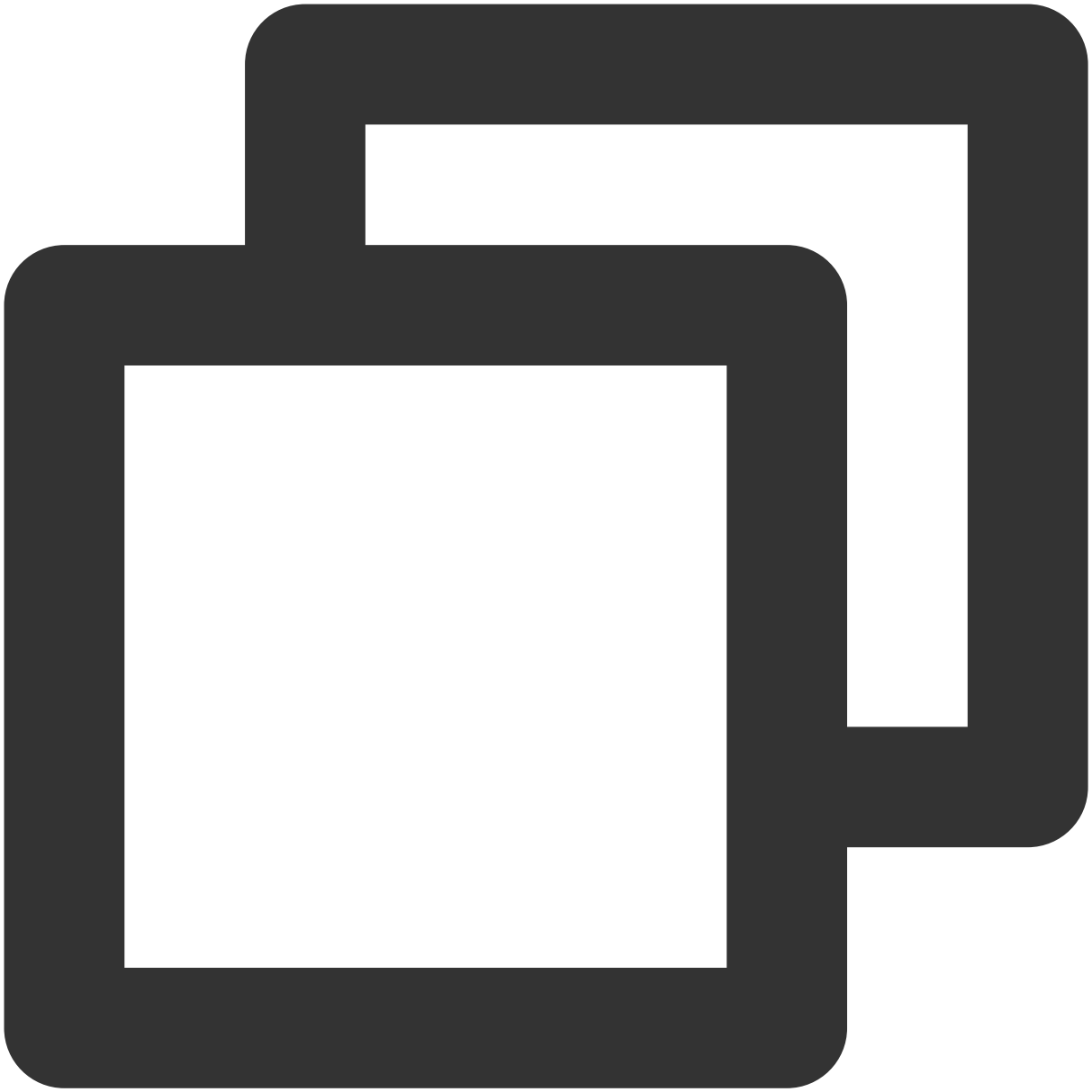
```
DATE_FROM_UNIX_DATE(<unix_timestamp> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：根据自1970-01-01以来的天数创建日期。

返回类型：date。

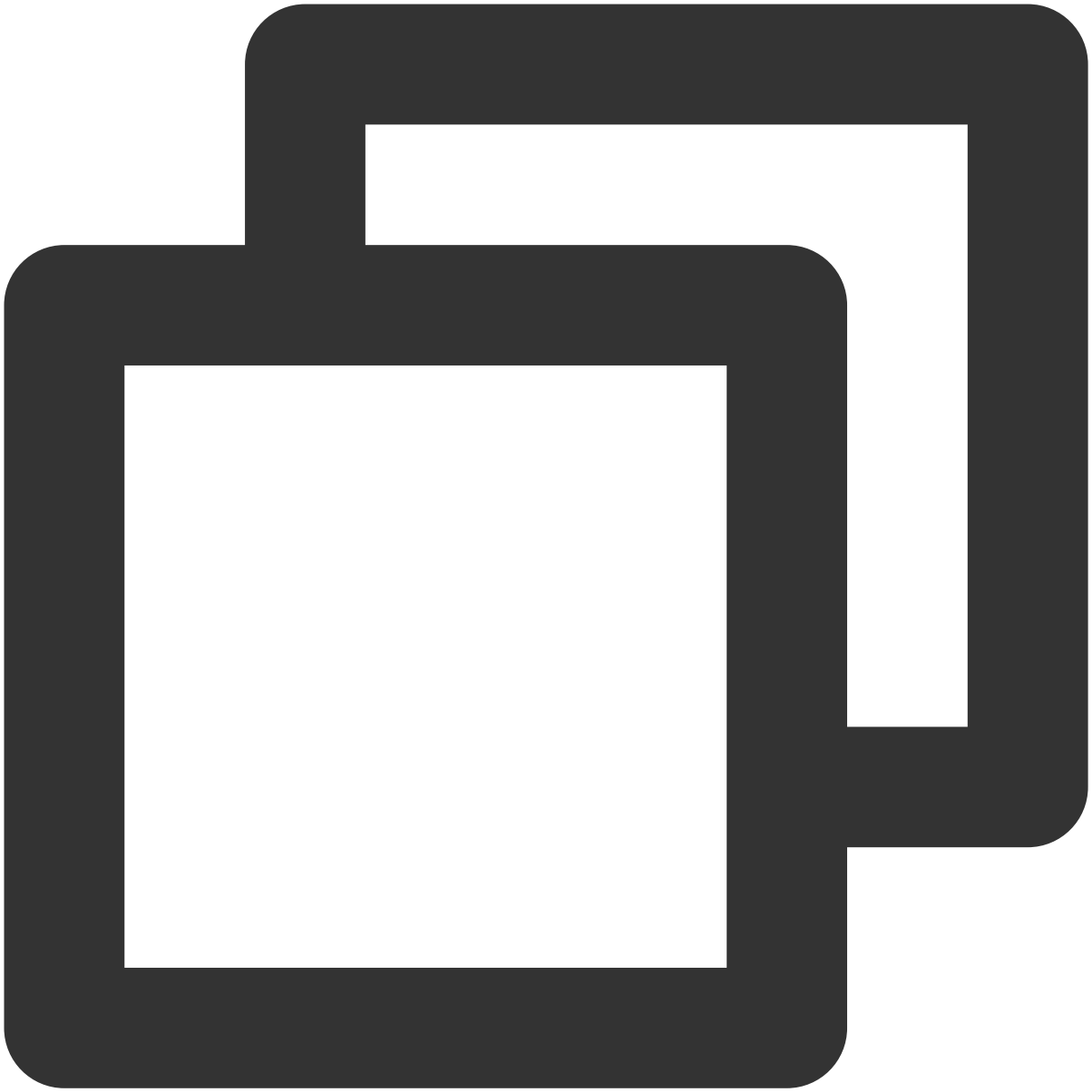
示例：



```
> SELECT date_from_unix_date(1);
 1970-01-02
```

UNIX_DATE

函数语法：



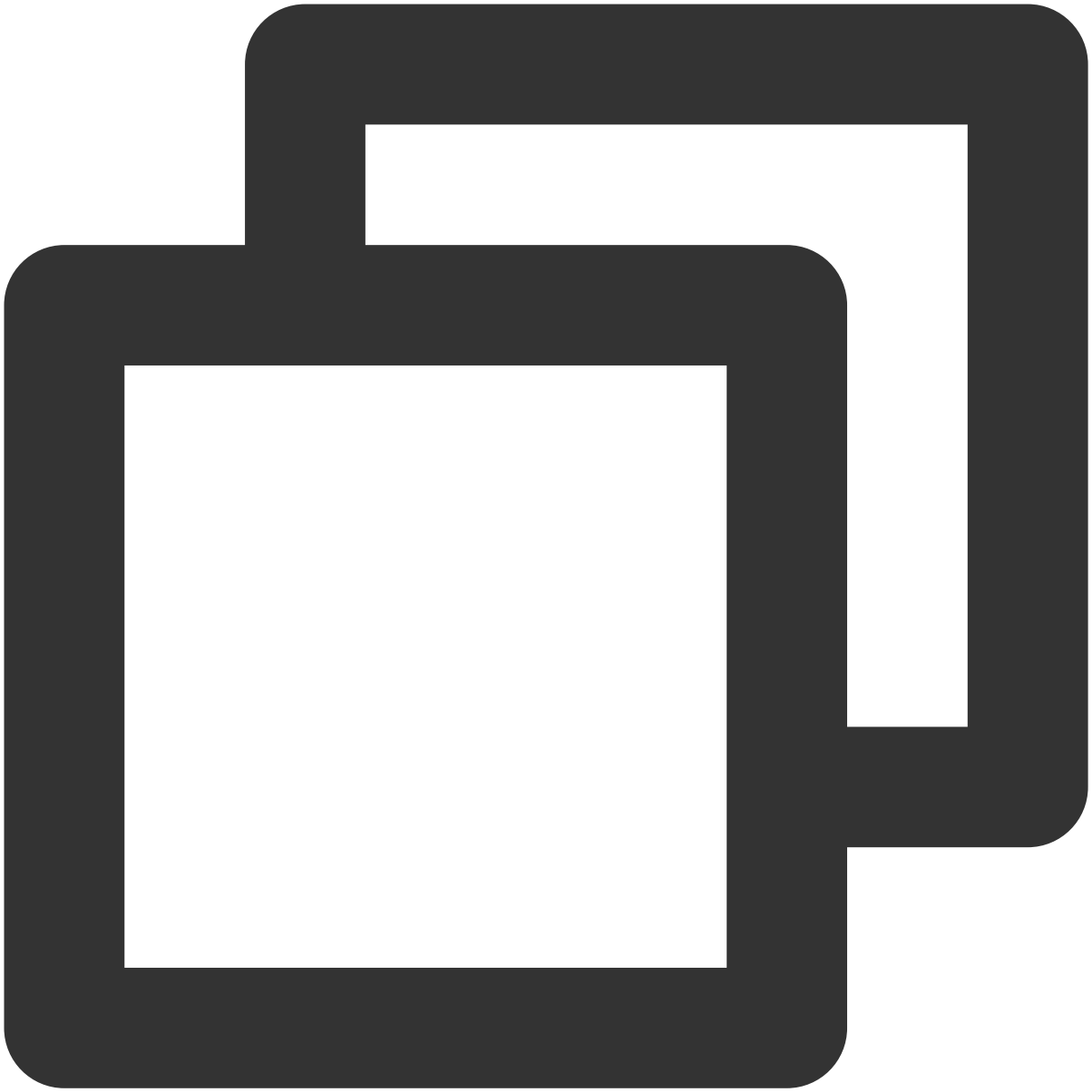
```
UNIX_DATE(<d> date)
```

支持引擎：SparkSQL、Presto。

使用说明：返回自1970-01-01以来的天数。

返回类型：integer。

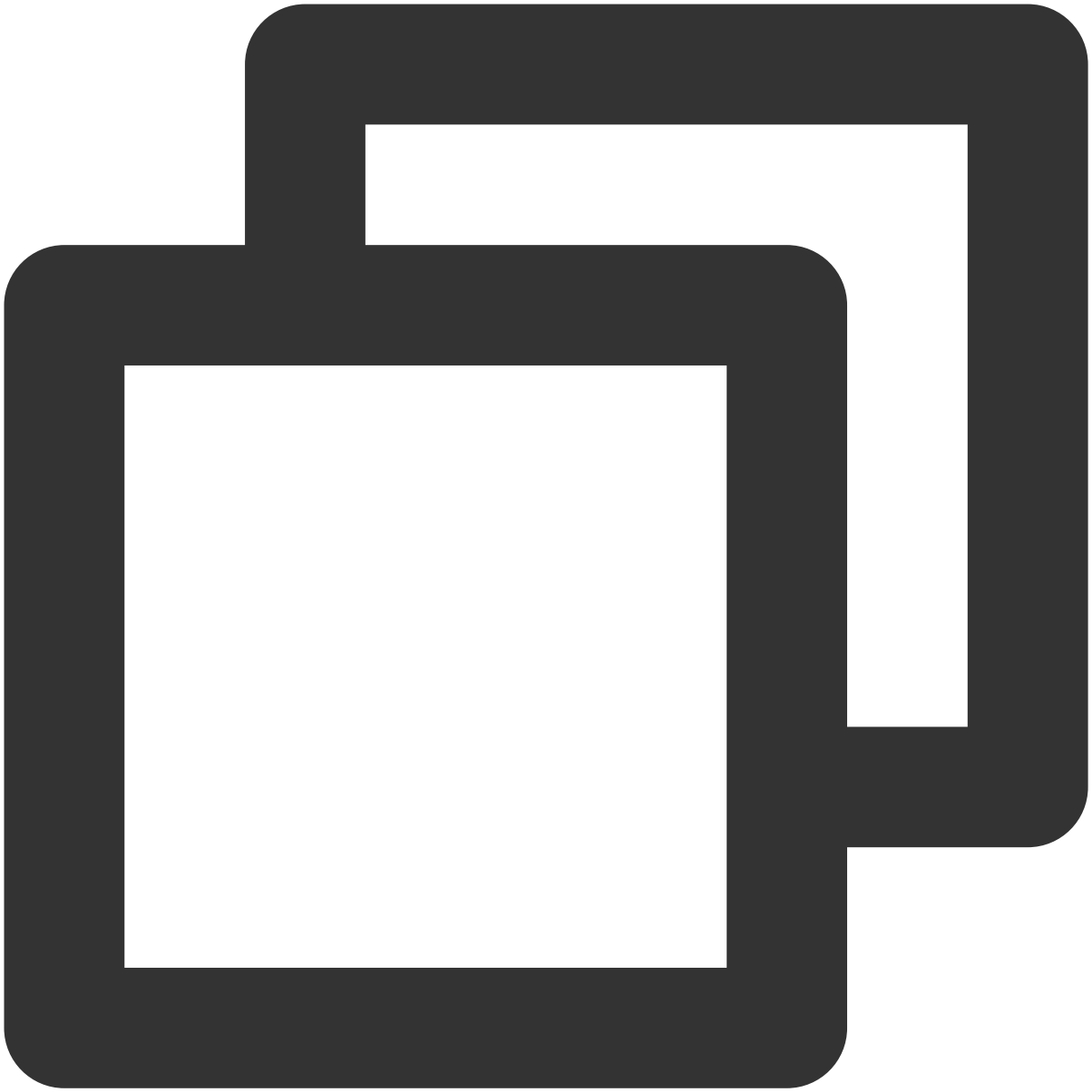
示例：



```
> SELECT unix_date(DATE("1970-01-02"));
 1
```

TIMESTAMP_SECONDS

函数语法：



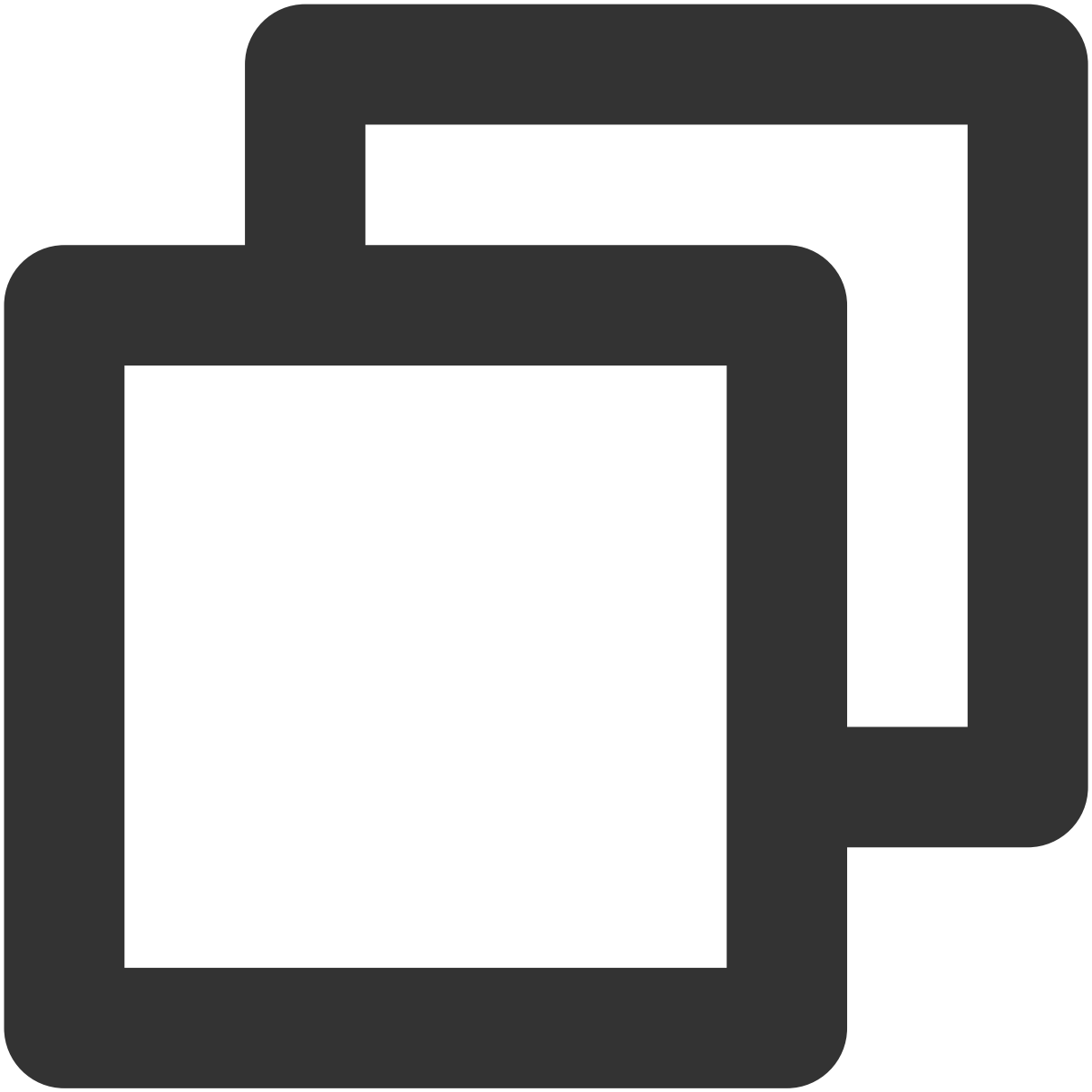
```
TIMESTAMP_SECONDS(<sec> bigint|double|decimal)
```

支持引擎：SparkSQL、Presto。

使用说明：从 UTC 纪元以来的秒数创建时间戳。

返回类型：timestamp。

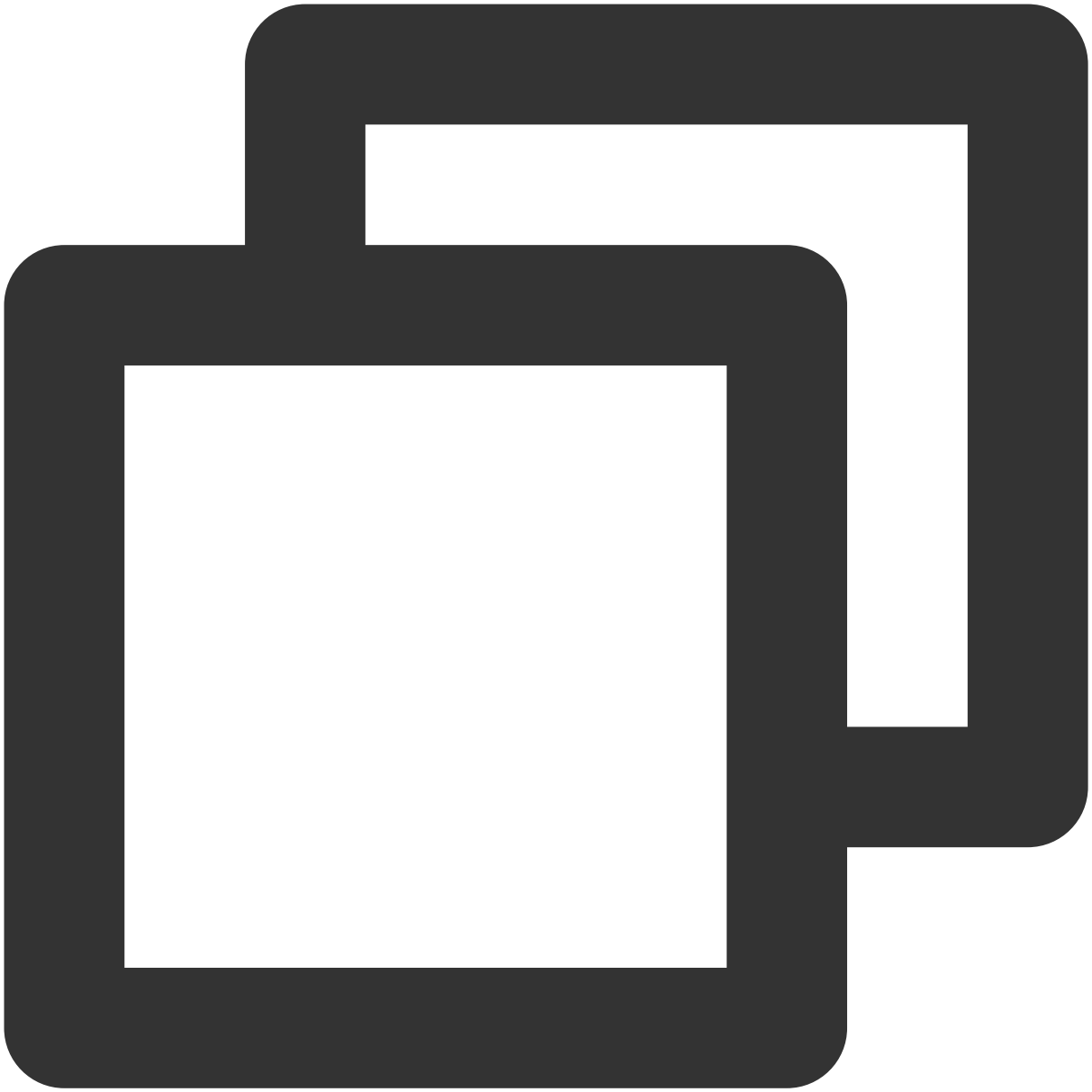
示例：



```
> SELECT timestamp_seconds(1230219000);
 2008-12-25 07:30:00
> SELECT timestamp_seconds(1230219000.123);
 2008-12-25 07:30:00.123
```

TIMESTAMP_MILLIS

函数语法：



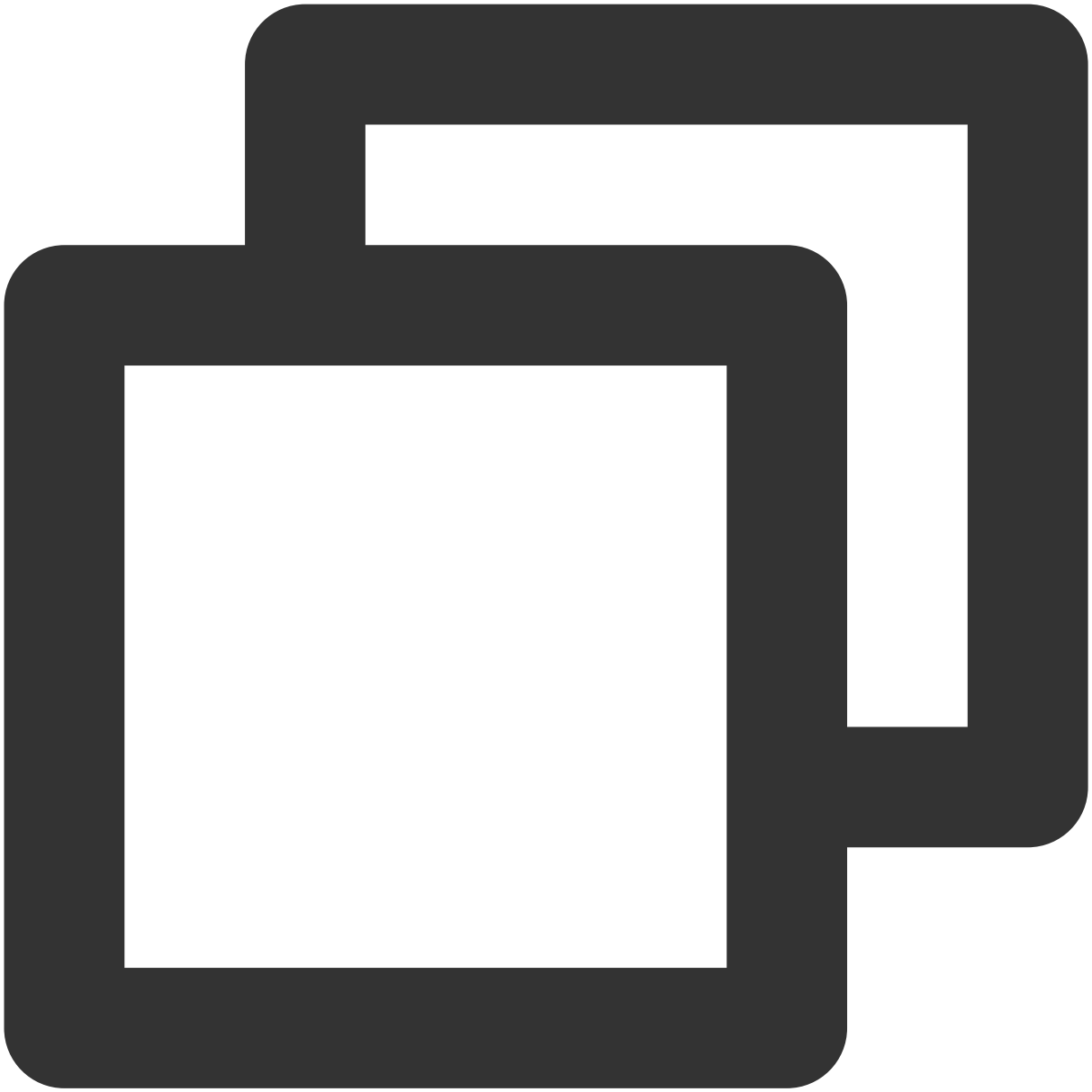
```
TIMESTAMP_MILLIS(<milli> bigint|double|decimal)
```

支持引擎：SparkSQL、Presto。

使用说明：从 UTC 纪元以来的毫秒数创建时间戳。

返回类型：timestamp。

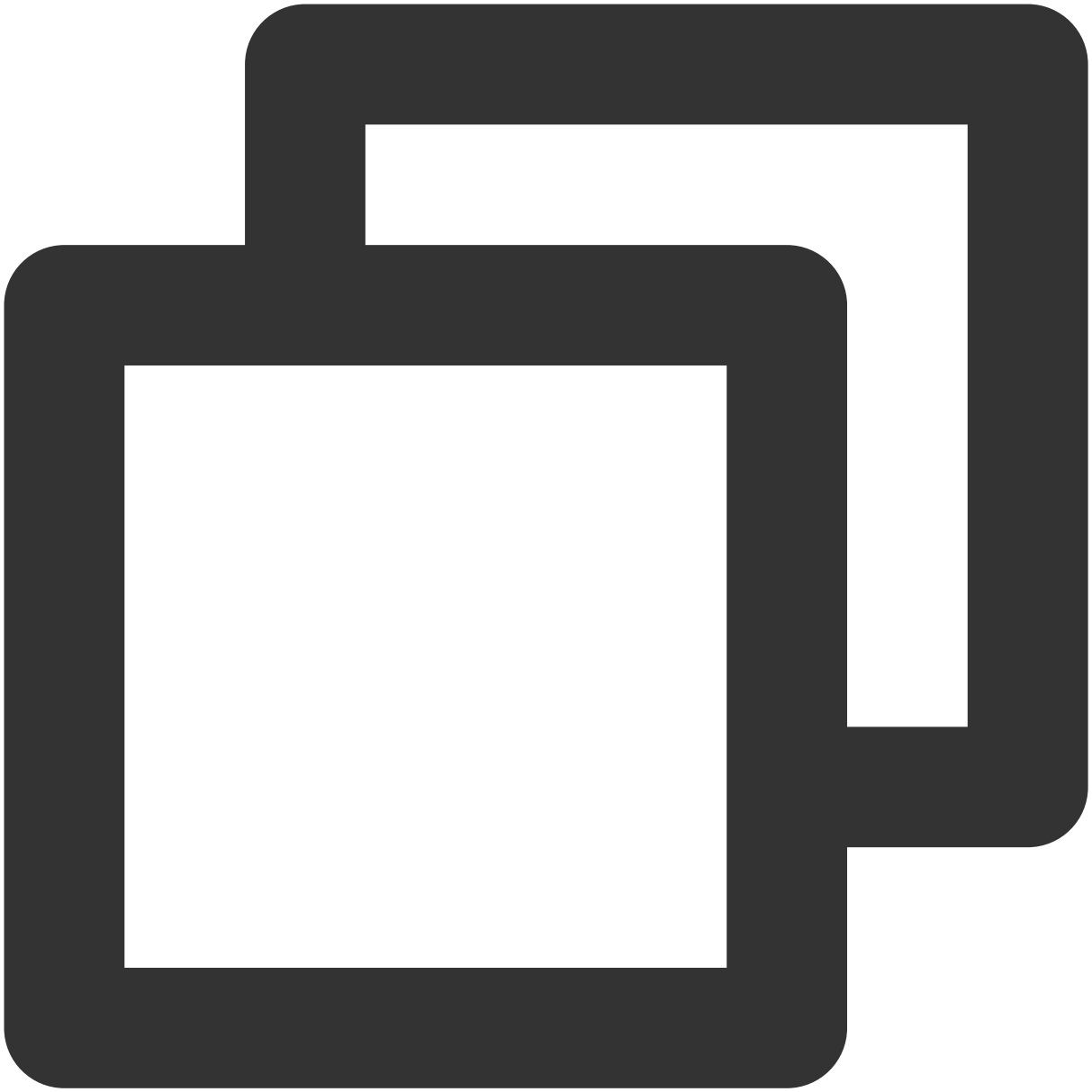
示例：



```
> SELECT timestamp_millis(1230219000123);
 2008-12-25 07:30:00.123
```

TIMESTAMP_MICROS

函数语法：



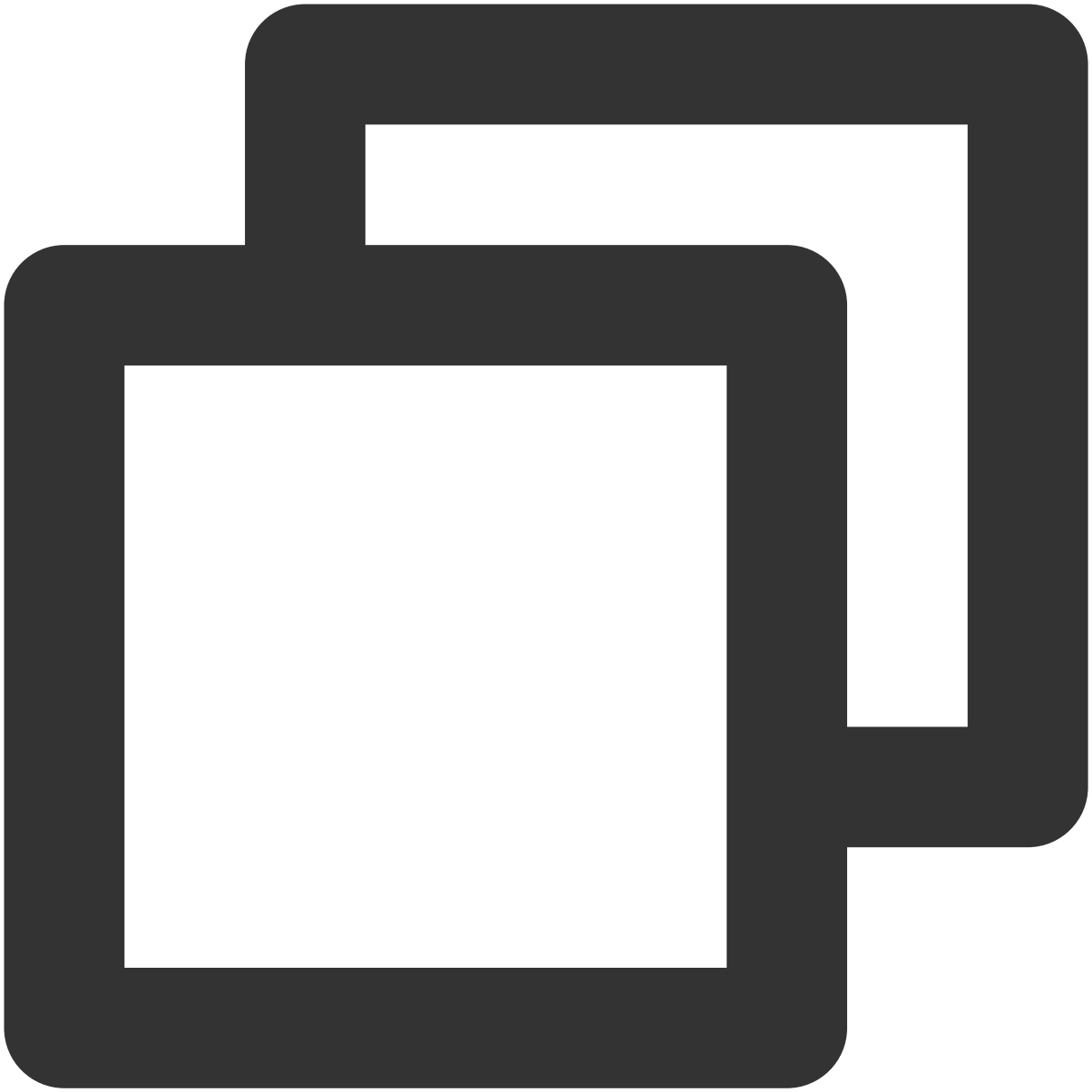
```
TIMESTAMP_MICROS(<micro> bigint)
```

支持引擎：SparkSQL、Presto。

使用说明：从 UTC 纪元以来的毫秒数创建时间戳。

返回类型：timestamp。

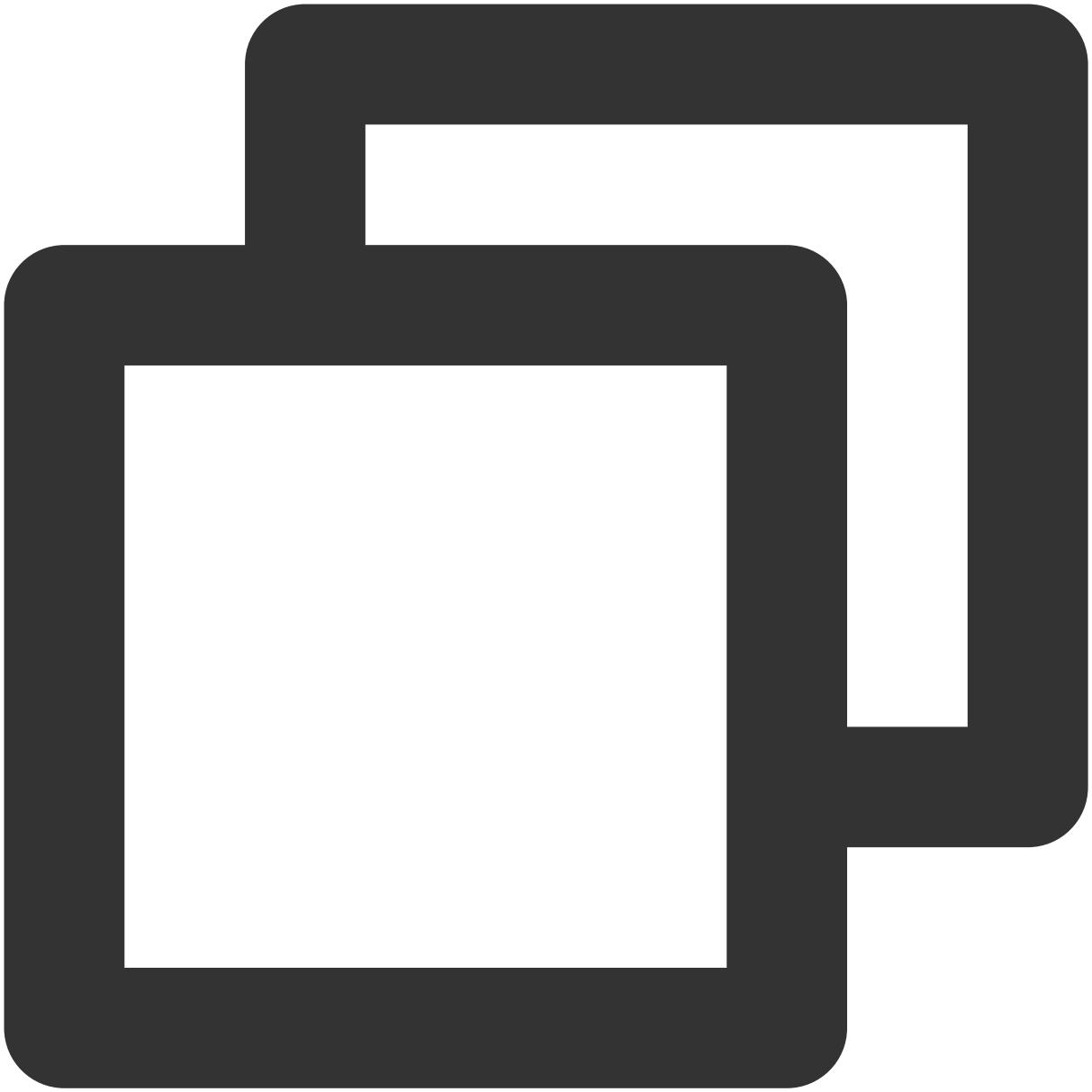
示例：



```
> SELECT timestamp_micros(1230219000123123);
 2008-12-25 07:30:00.123123
```

UNIX_SECONDS

函数语法：



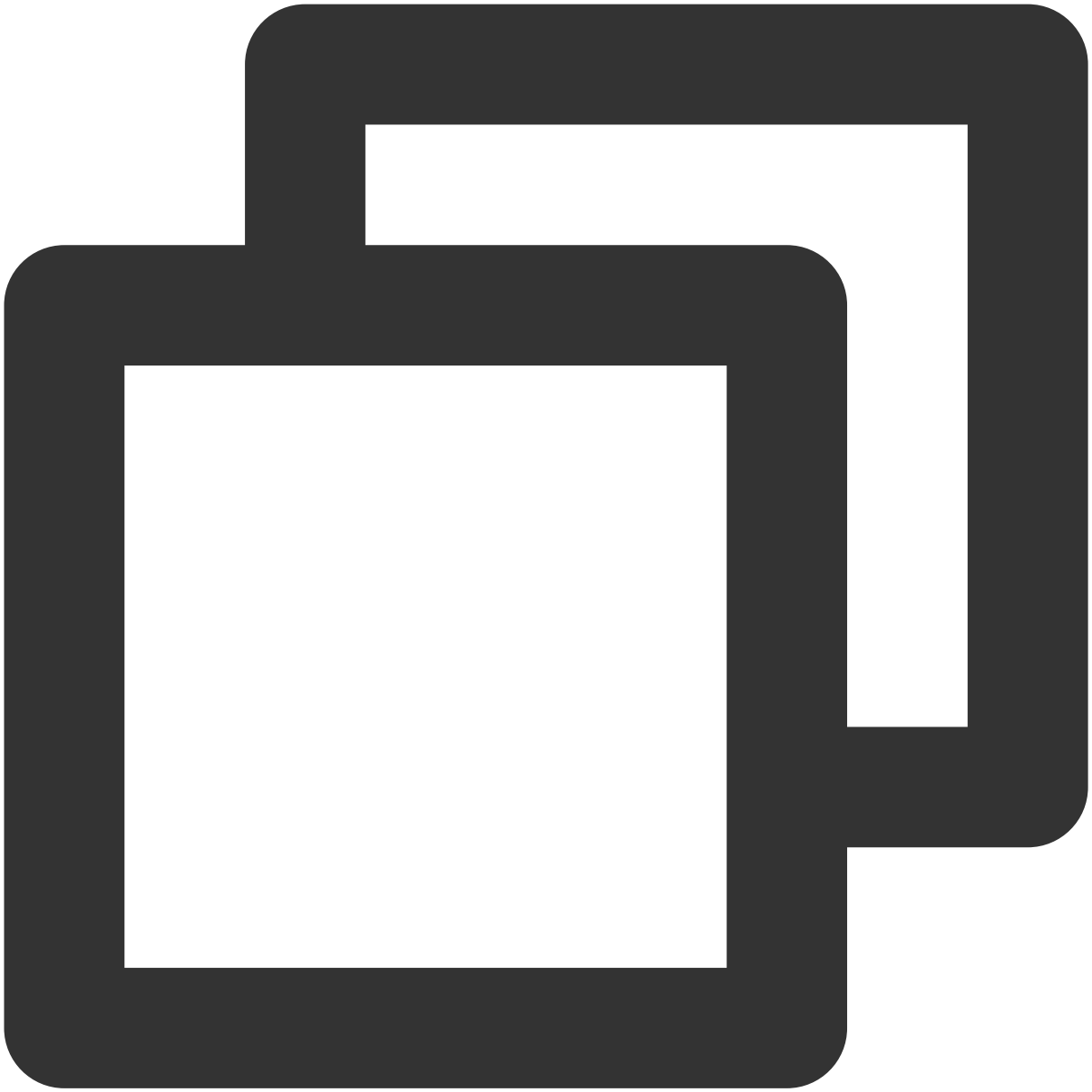
```
UNIX_SECONDS(<ts> timestamp)
```

支持引擎：SparkSQL、Presto。

使用说明：返回自1970-01-01 00:00:00 UTC以来的秒数。

返回类型：bigint。

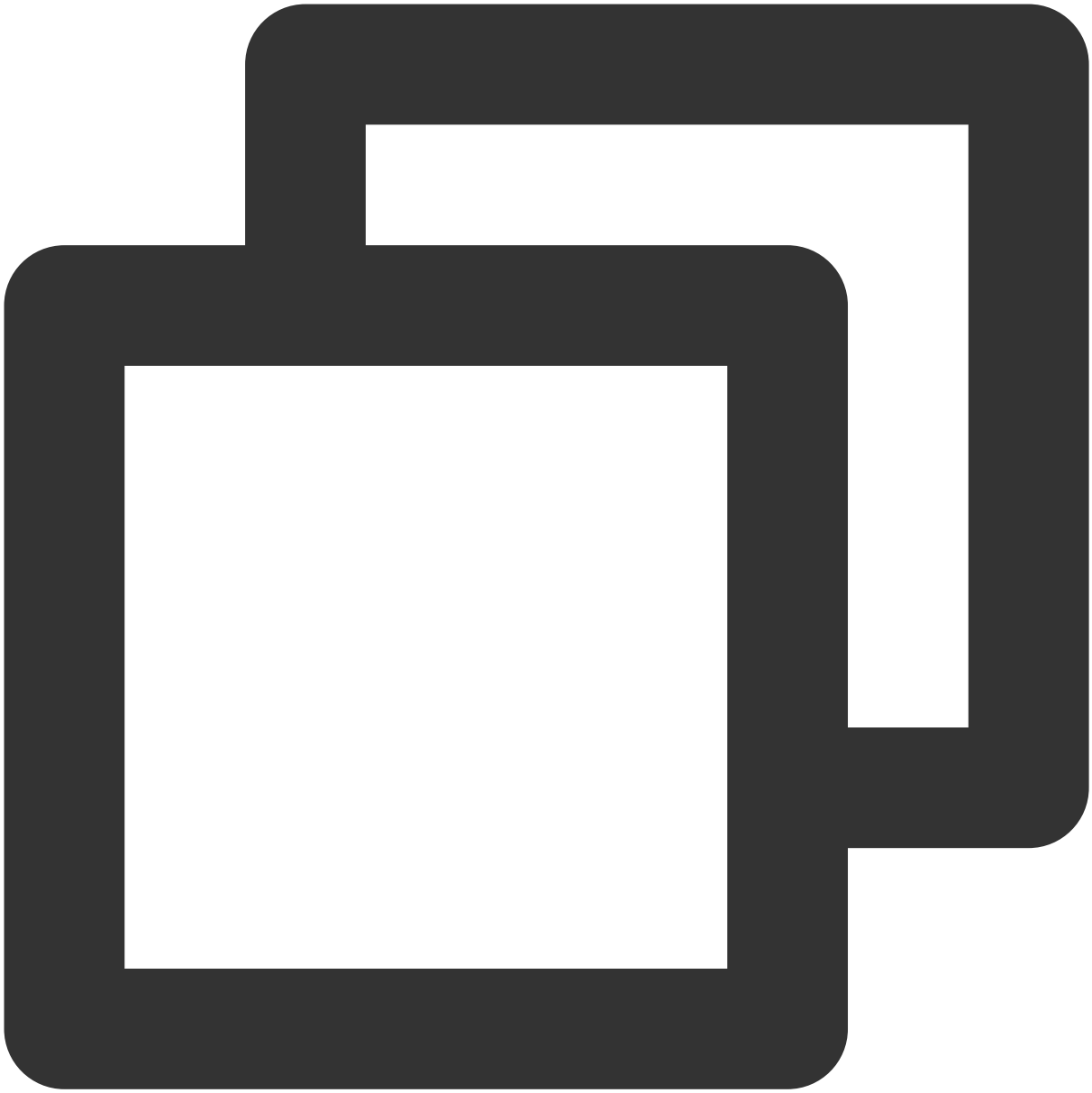
示例：



```
> SELECT unix_seconds(TIMESTAMP('1970-01-01 00:00:01Z'));
 1
```

UNIX_MILLIS

函数语法：



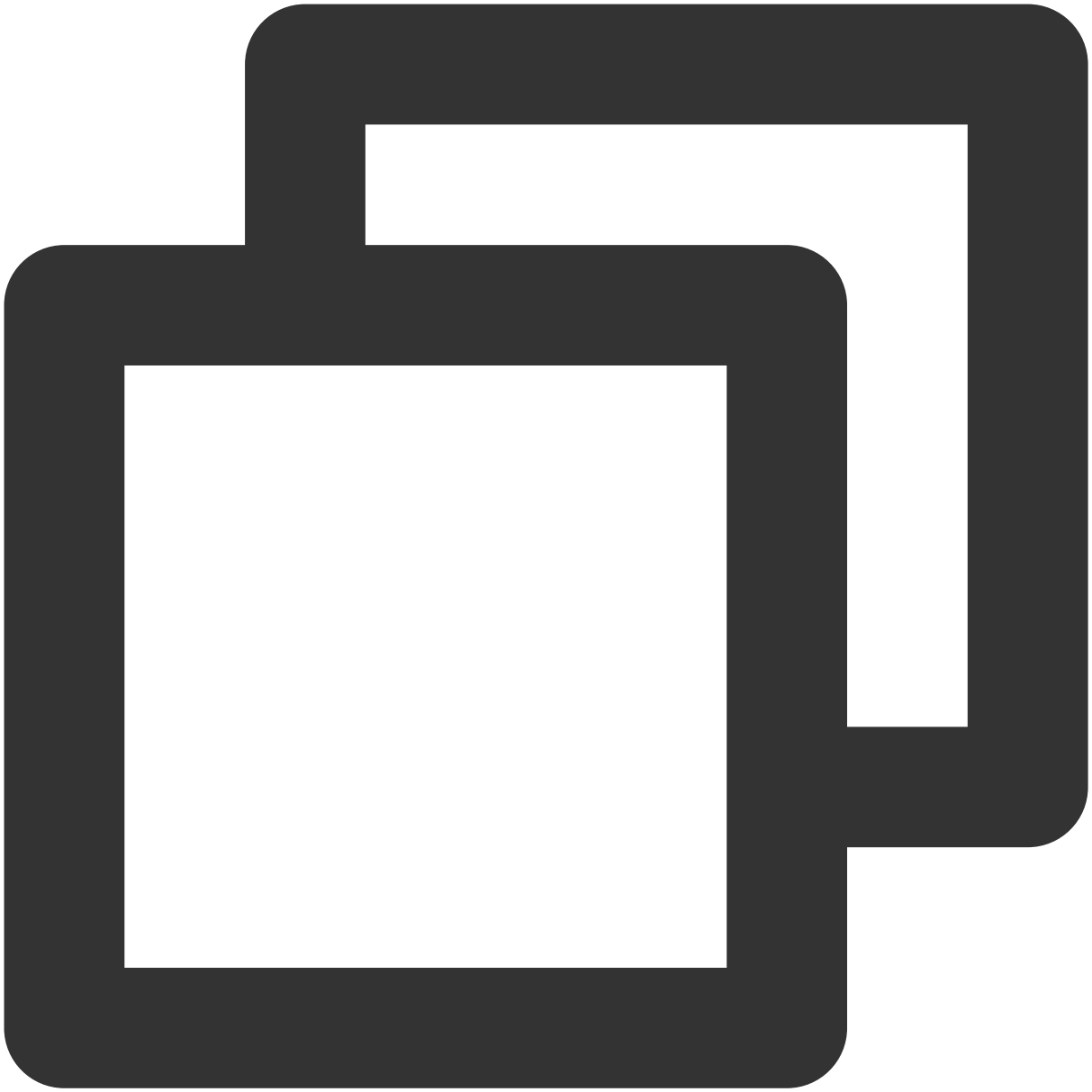
```
UNIX_MILLIS(<ts> timestamp)
```

支持引擎：SparkSQL、Presto。

使用说明：返回自1970-01-01 00:00:00 UTC 以来的毫秒数。

返回类型：bigint。

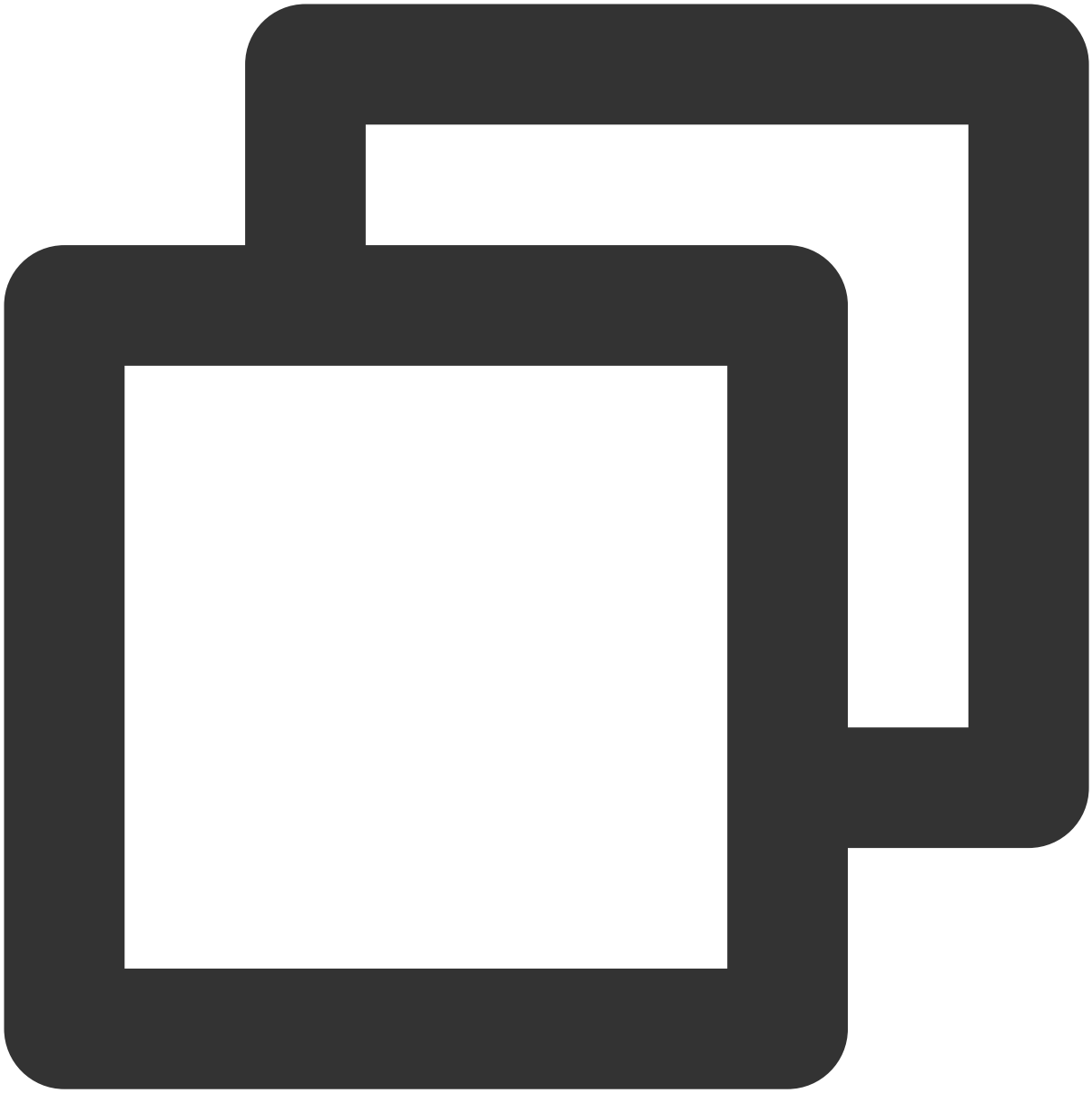
示例：



```
> SELECT unix_millis(TIMESTAMP('1970-01-01 00:00:01Z'));
 1000
```

UNIX_MICROS

函数语法：



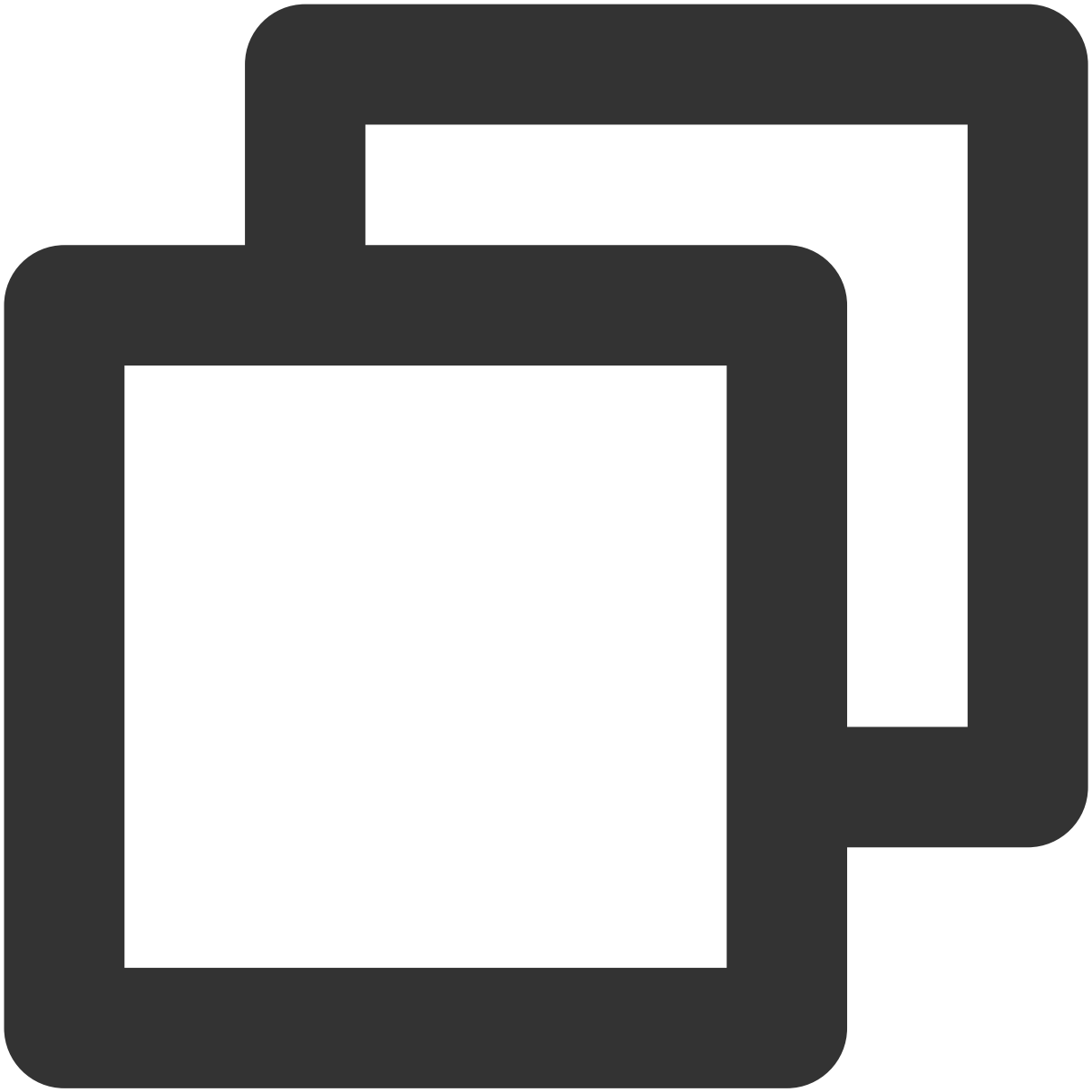
```
UNIX_MICROS(<ts> timestamp)
```

支持引擎：SparkSQL。

使用说明：返回自1970-01-01 00:00:00 UTC 以来的微秒数。

返回类型：bigint。

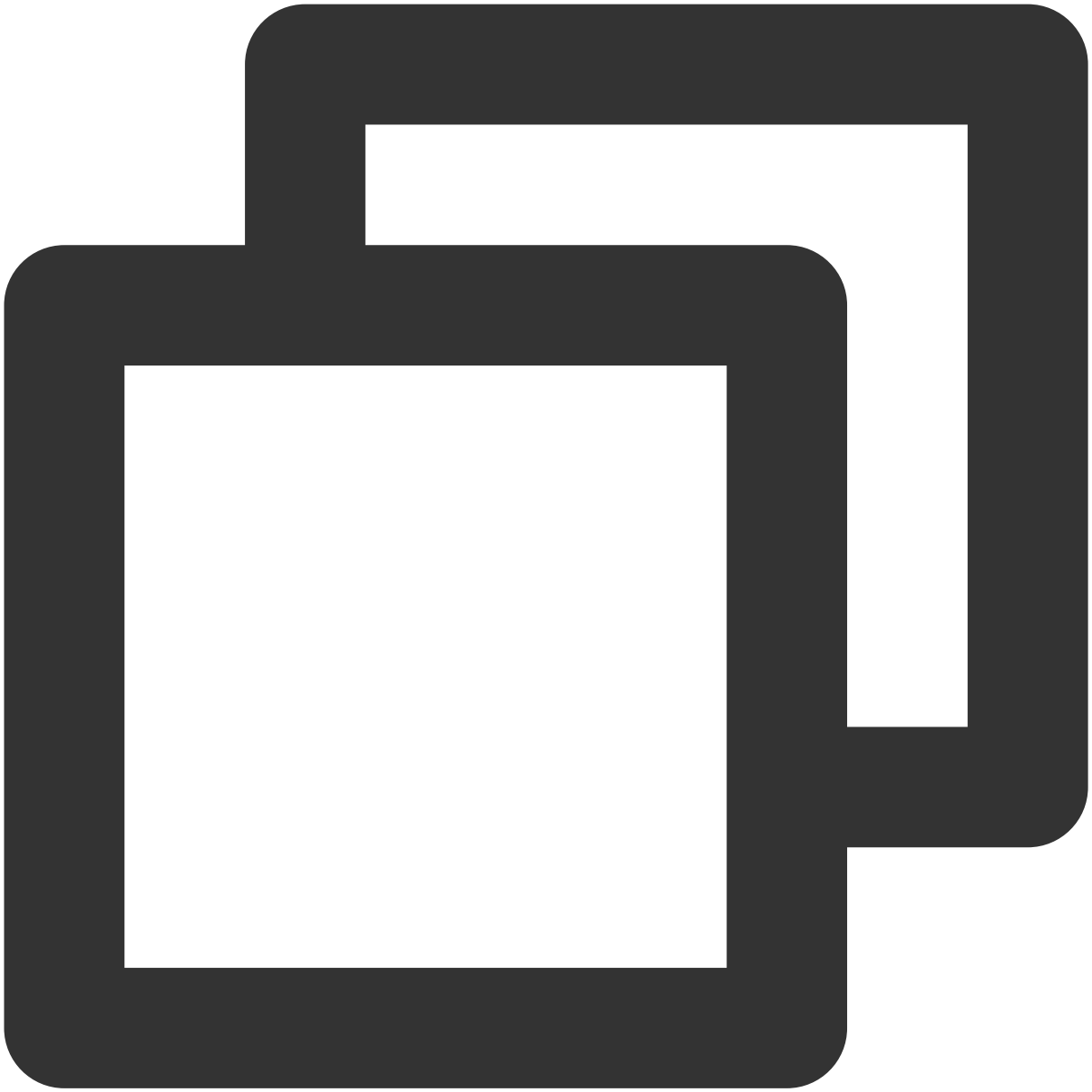
示例：



```
> SELECT unix_micros(timestamp '1970 00:00:00.001')
 1000
```

DATEADD/TIMESTAMP_ADD

函数语法：



```
TIMESTAMP_ADD(<date|timestamp> dt, <int> delta, <string> pattern)
TIMESTAMP_ADD(<date|timestamp> dt, <int> delta, <string> pattern)
```

支持引擎：SparkSQL。

引擎版本要求：需在2023年12月07日及以后购买的引擎、或升级到最新引擎。

使用说明：在指定的日期或时间上，根据指定的单位和偏移量计算最终的时间。

dt：输入的日期或时间格式。如果为null则返回为null。

delta：必填，int类型。如果为null则返回null。

pattern：必填，string类型，单位。pattern 可以支持：

年(标准"y"， 可兼容"-year", "yyyy")

月（标准"M", 可兼容"-month","-mon", "MM"）

日（标准"d"，可兼容"-day","dd"）

小时（标准"H"，可兼容"-hour", "hh", "HH", "h"）

分钟 (标准"m"，可兼容"mi", "mm")

秒（标准"s"，可兼容"ss"）

微秒（标准"S"，可兼容"SSS"）

不允许输入其他的类型，否则会报错。

注意：

当 delta 的单位是月时，如果 delta 的月部分在增加 delta 值之后不造成 Day 溢出，则保持 Day 值不变，否则将 Day 值设置为结果月份的最后1天。

注意输出的时间默认是时区的。

月M和分钟m的单位区别；兼容了0-12小时制h和0-24小时制H，但是推荐使用H。

返回类型：timestamp。

示例：

```
> SELECT timestamp_add(date '2016-12-31', -1, 'M');
2016-11-30T00:00:00.000+08:00
> SELECT timestamp_add(timestamp '2016-12-31 00:12:00', 1, 'm');
2016-12-31T00:13:00.000+08:00
> SELECT timestamp_add(timestamp '2016-12-31 00:00:00', -1, 'm');
2016-12-30T23:59:00.000+08:00
```

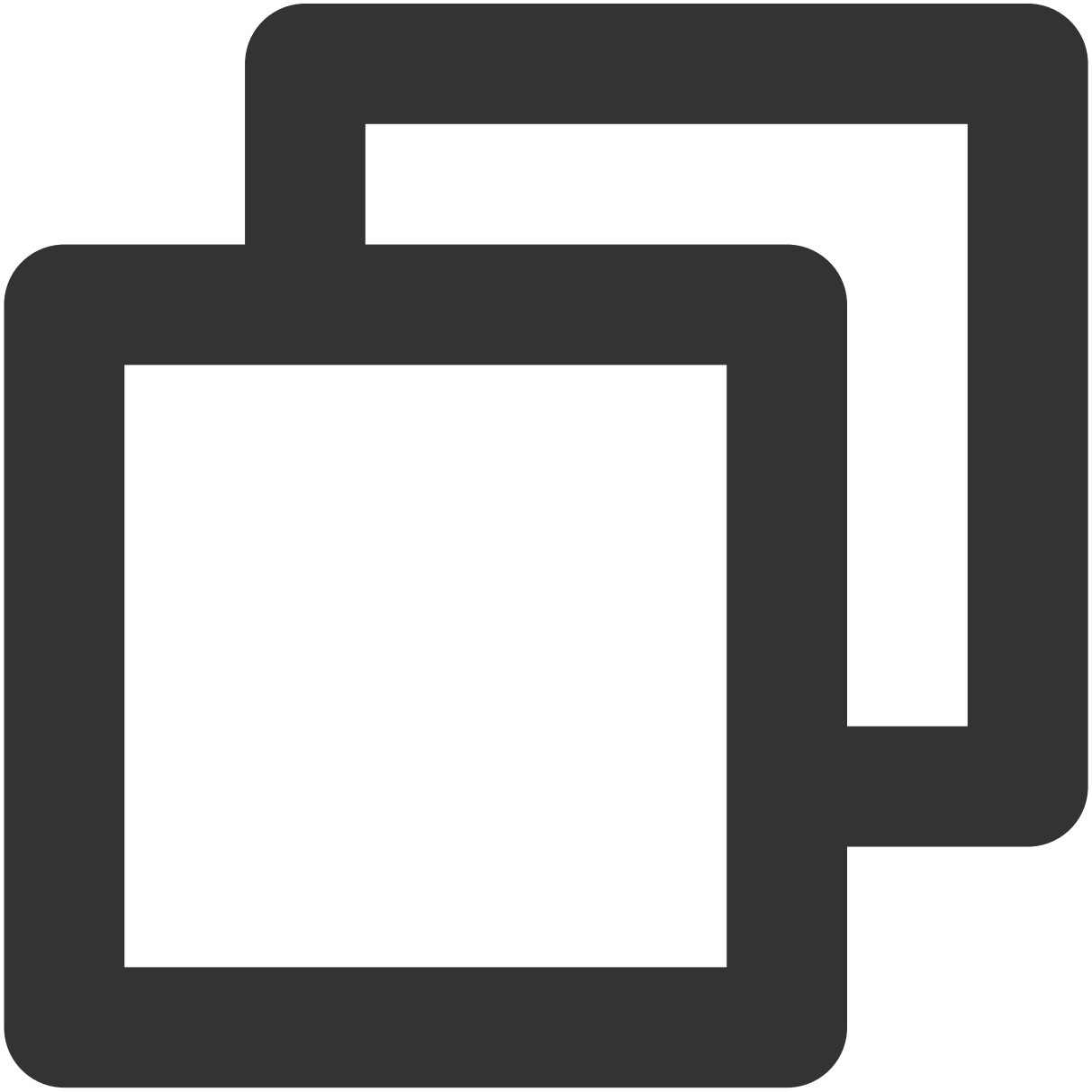
版权所有：腾讯云计算（北京）有限责任公司

JSON 函数

最近更新时间：2024-08-07 17:32:24

GET_JSON_OBJECT

函数语法：



```
GET_JSON_OBJECT(<json> string, <path> string)
```

支持引擎：SparkSQL、Presto。

使用说明：提取 json 对象。

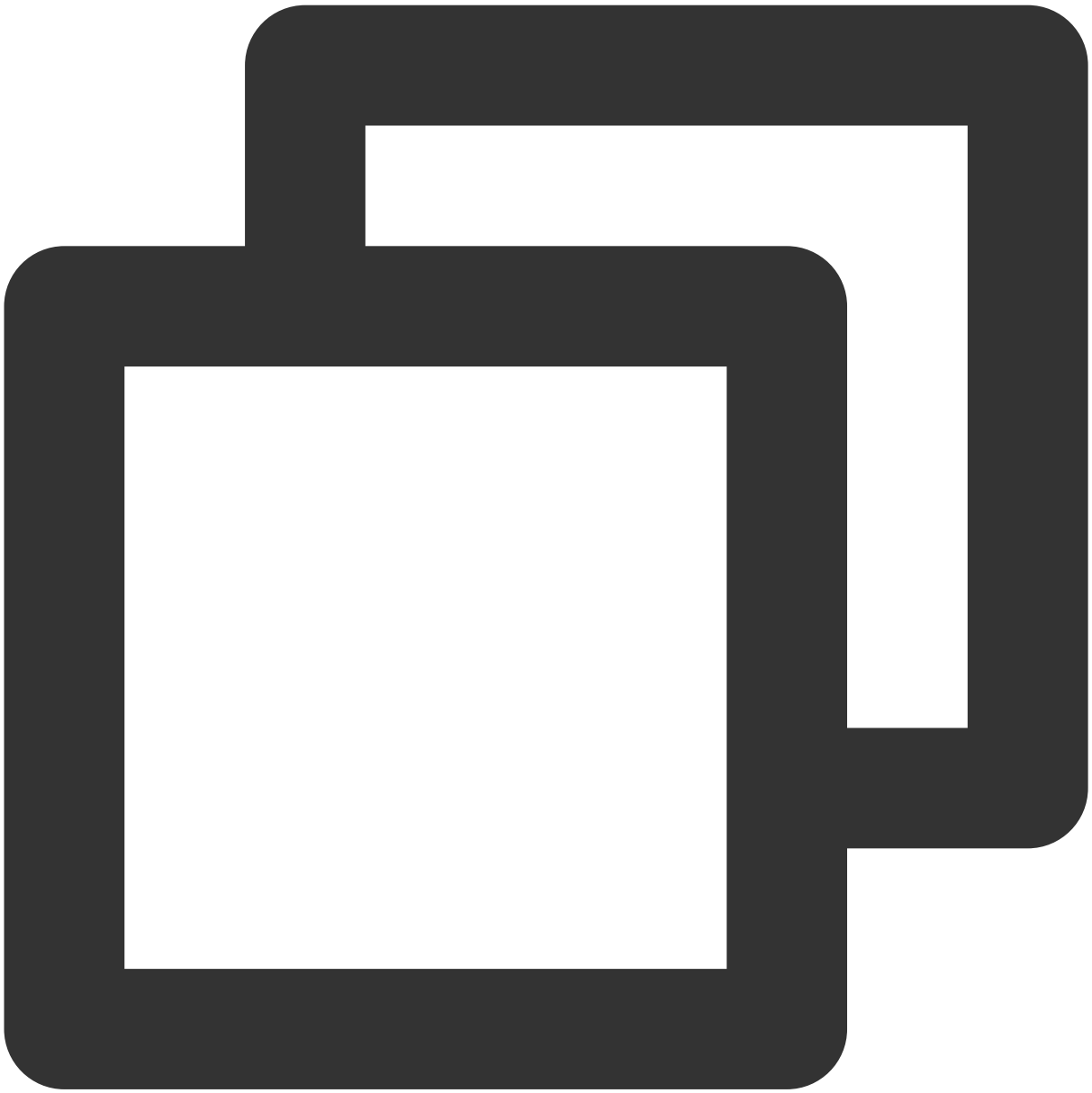
返回类型：string。

示例：

```
> SELECT get_json_object('{"a":"b"}', '$.a');
 b
```

JSON_TUPLE

函数语法：



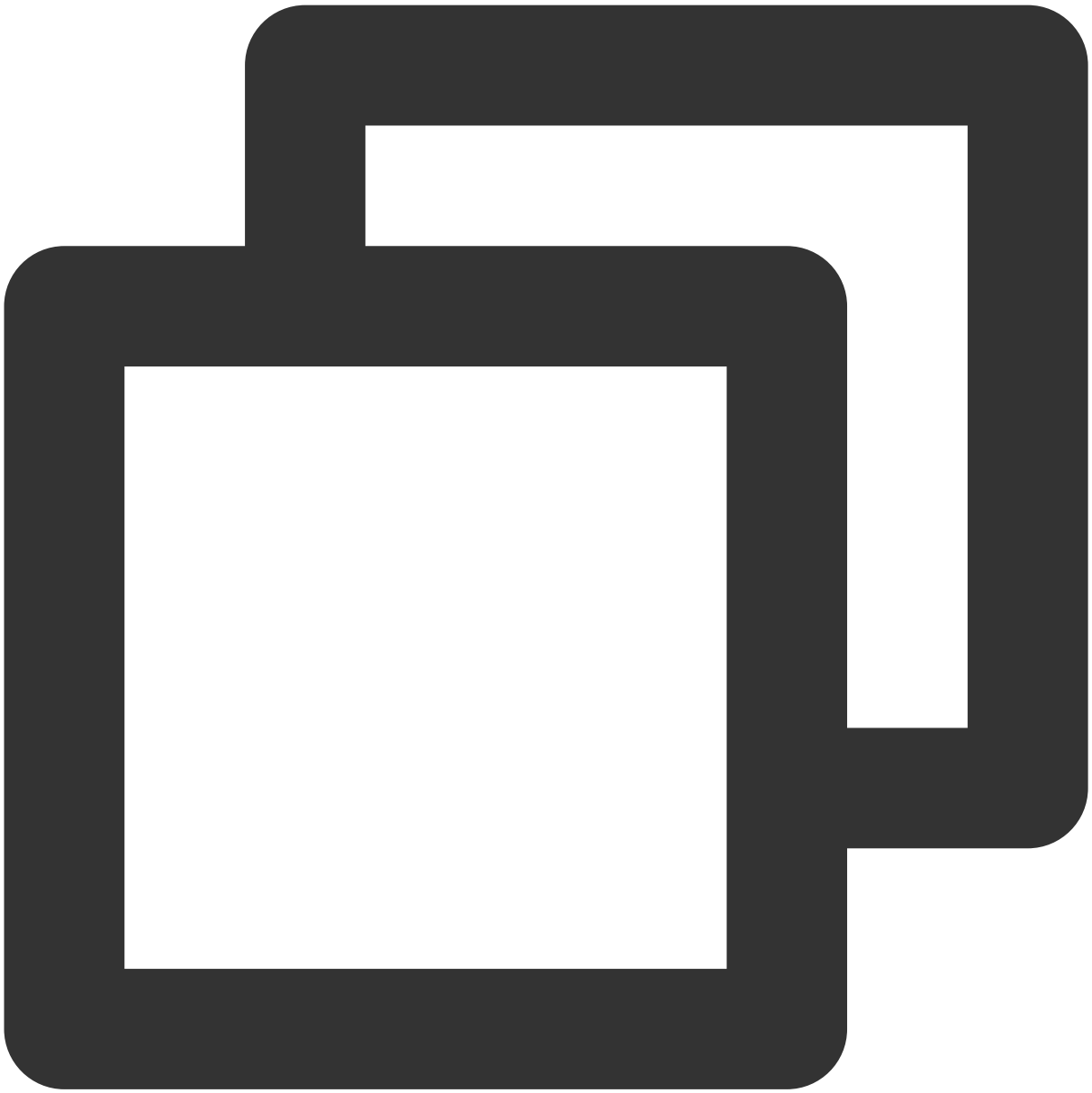
```
JSON_TUPLE(<json> string, <p1> string, ..., <pn> string)
```

支持引擎：SparkSQL。

使用说明：返回一个类似于函数 get_json_object 的元组，但它需要输入多个名称。所有输入参数和输出列类型都是字符串。

返回类型：struct<string, ..., string>。

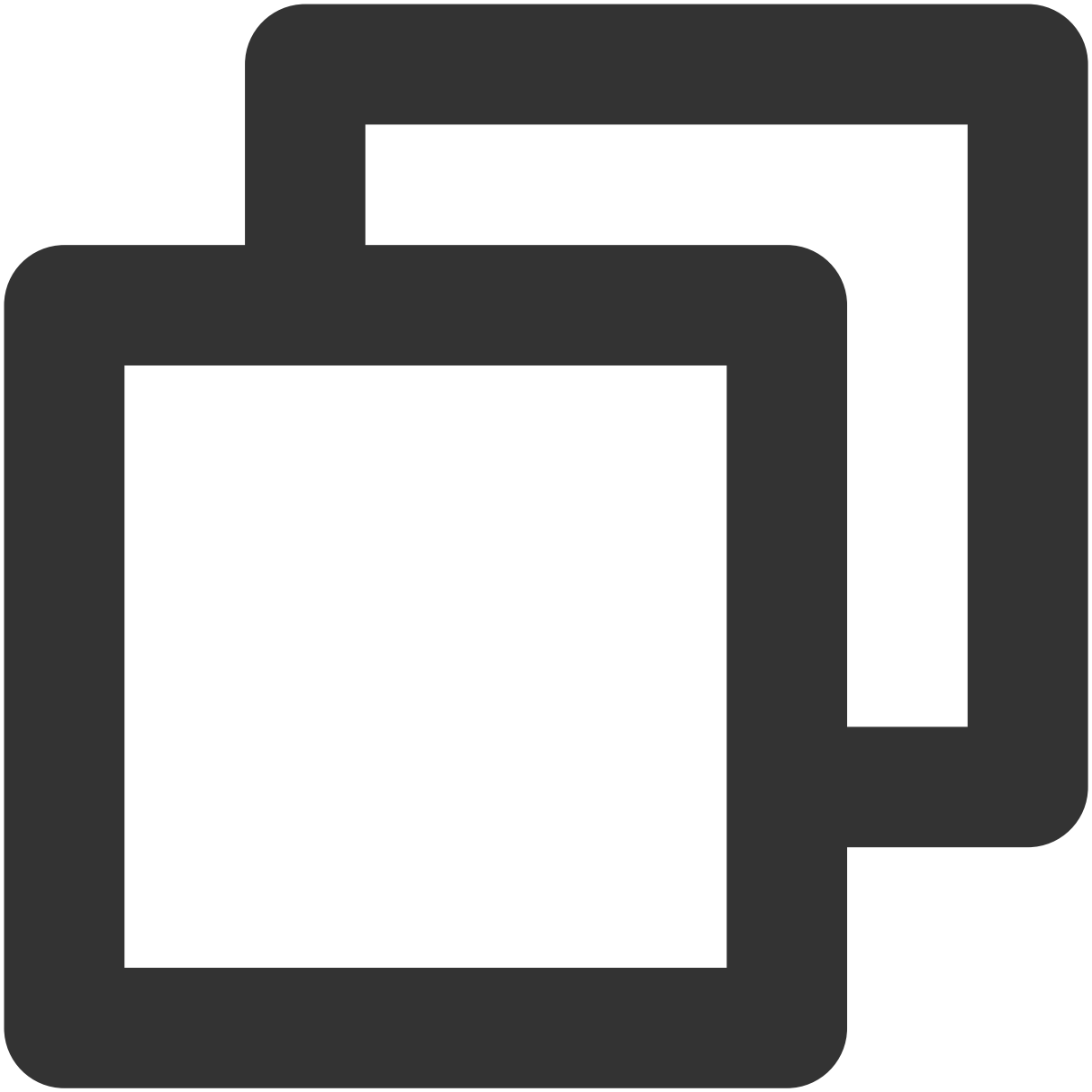
示例：



```
> SELECT json_tuple('{"a":1, "b":2}', 'a', 'b');
 1  2
```

TO_JSON

函数语法：



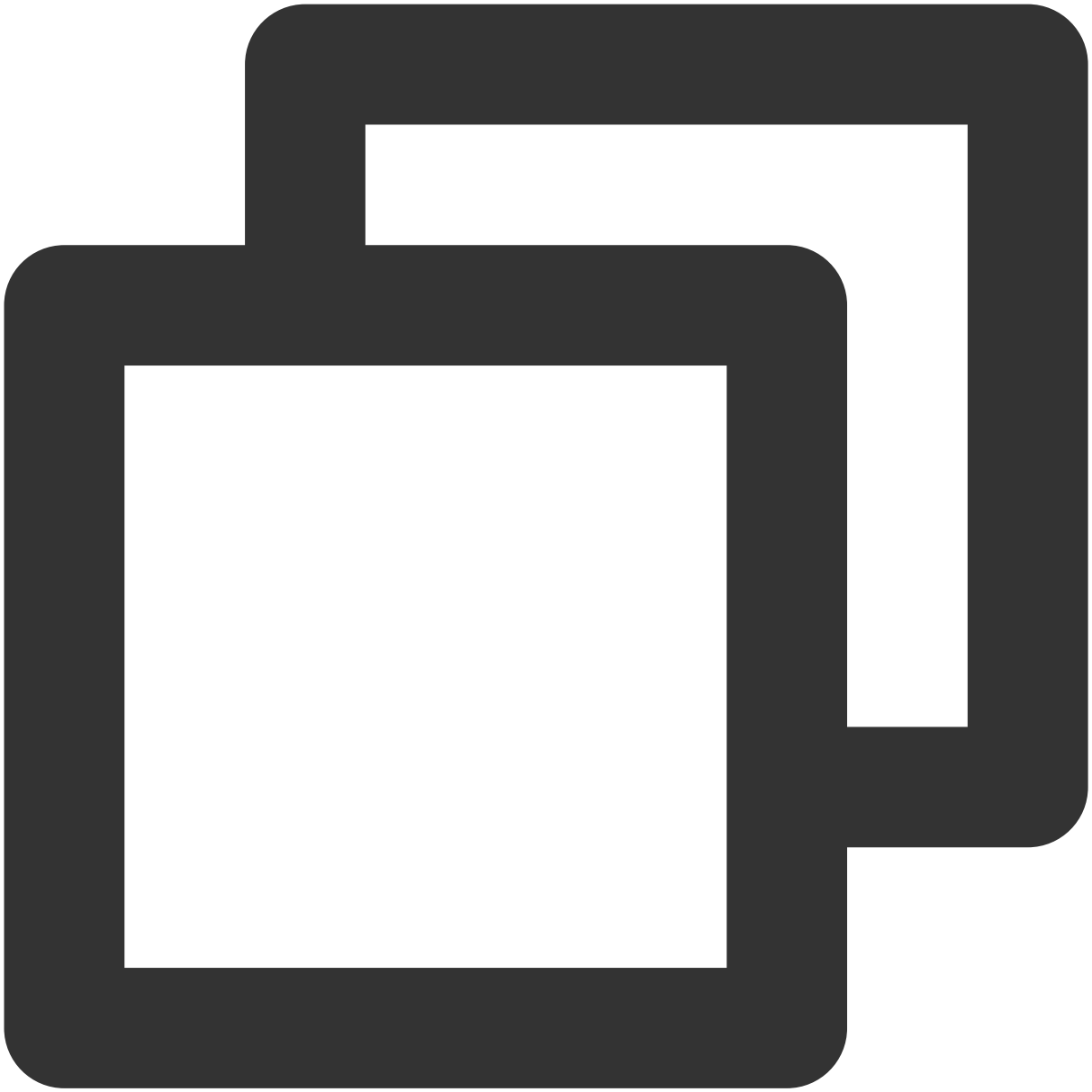
```
TO_JSON(<expr> struct|map|array[, <option> map])
```

支持引擎：SparkSQL、Presto。

使用说明：返回具有给定结构值的 JSON 字符串。

返回类型：string。

示例：

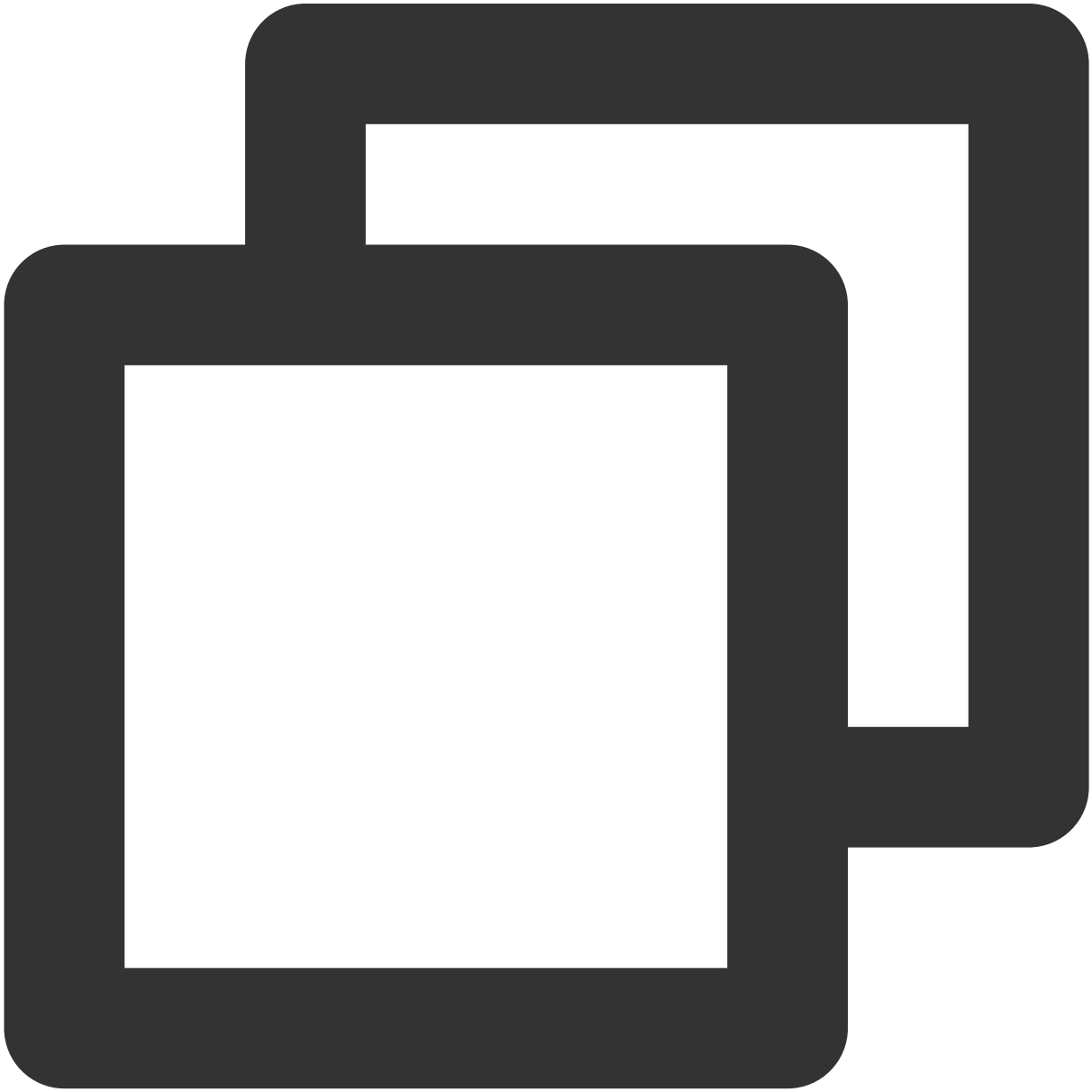


```
> SELECT to_json(named_struct('a', 1, 'b', 2));
 {"a":1,"b":2}
> SELECT to_json(named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd')), ma
 {"time":"26/08/2015"}
> SELECT to_json(array(named_struct('a', 1, 'b', 2)));
 [{"a":1,"b":2}]
> SELECT to_json(map('a', named_struct('b', 1)));
 {"a":{"b":1}}
> SELECT to_json(map(named_struct('a', 1),named_struct('b', 2)));
 {"[1]":{"b":2}}
> SELECT to_json(map('a', 1));
```

```
 {"a":1}
> SELECT to_json(array((map('a', 1))));
 [{"a":1}]
```

FROM_JSON

函数语法：



```
FROM_JSON(<json> string, <schema> string[, <options> map<string, string>])
```

支持引擎：SparkSQL、Presto。

使用说明：返回具有给定 jsonStr 和模式的结构值。

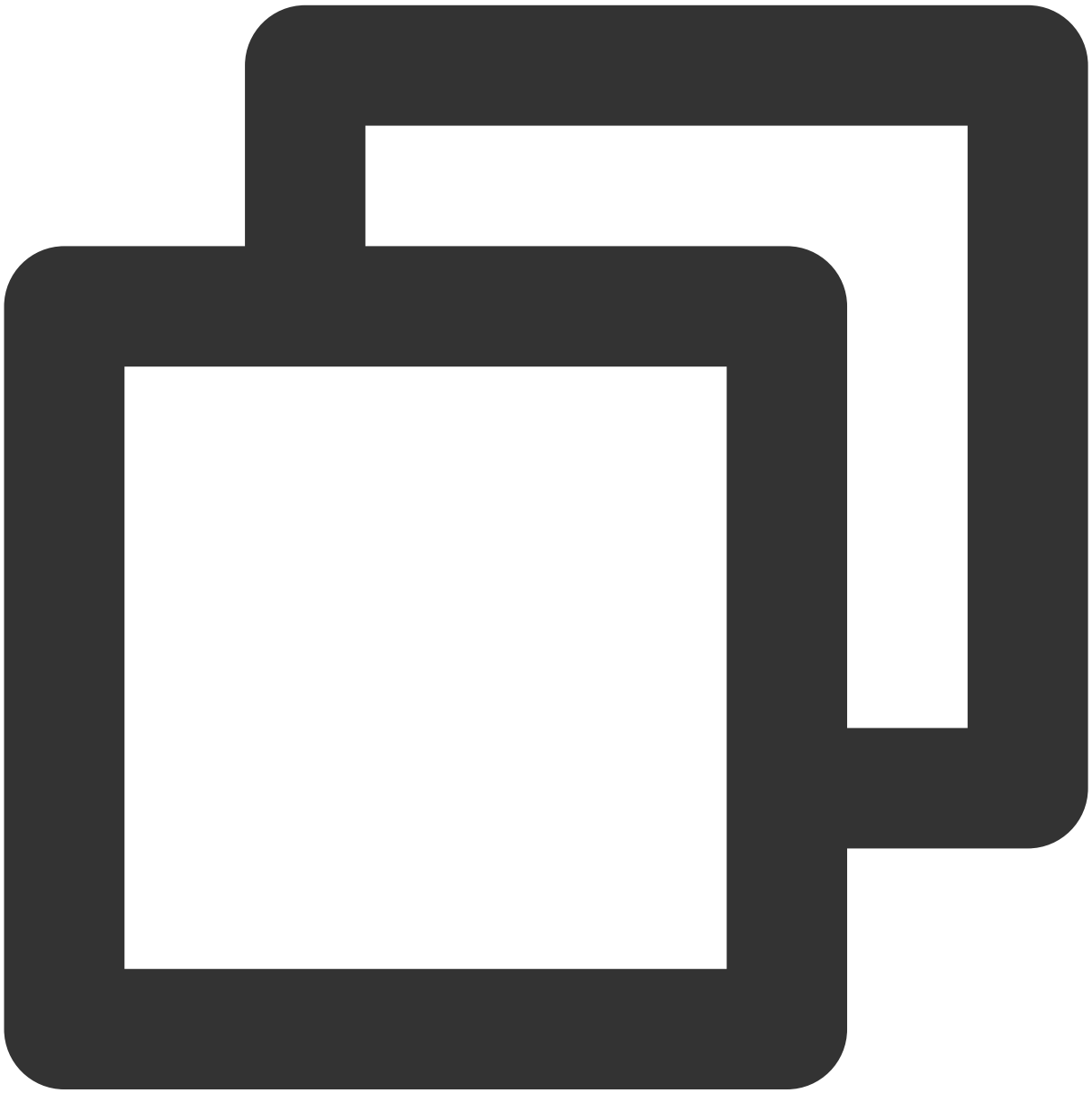
返回类型：struct。

示例：

```
> SELECT from_json('{"a":1, "b":0.8}', 'a INT, b DOUBLE');
 {"a":1,"b":0.8}
> SELECT from_json('{"time":"26/08/2015"}', 'time Timestamp', map('timestampFormat'
 {"time":2015-08-26 00:00:00}
```

SCHEMA_OF_JSON

函数语法：



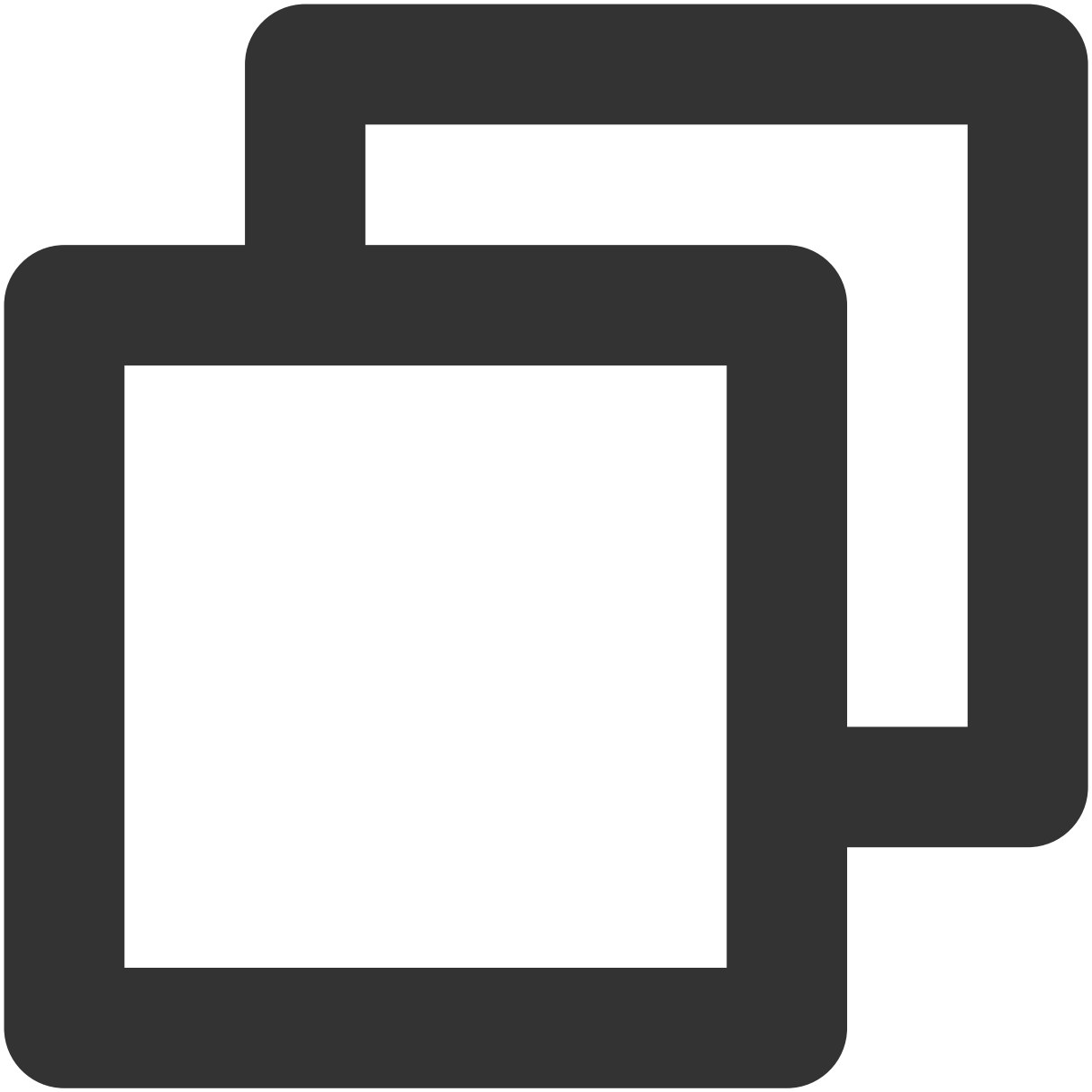
```
SCHEMA_OF_JSON(<json> string[, <options> map<string, string>])
```

支持引擎：SparkSQL。

使用说明：返回 JSON 字符串的 DDL 格式的结构。

返回类型：string。

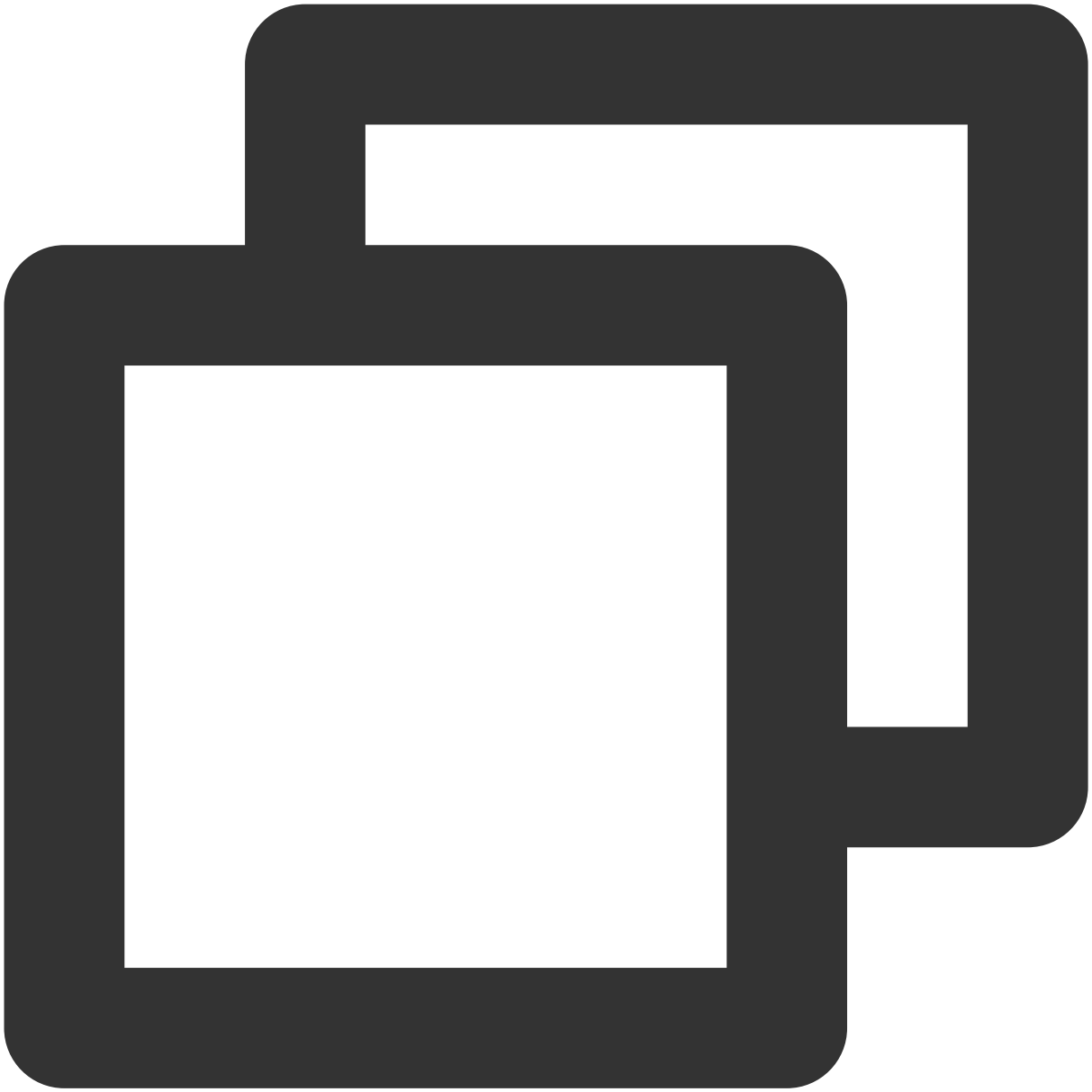
示例：



```
> SELECT schema_of_json('[{"col":0}]');
 ARRAY<STRUCT<`col`: BIGINT>>
> SELECT schema_of_json('[{"col":01}]', map('allowNumericLeadingZeros', 'true'));
 ARRAY<STRUCT<`col`: BIGINT>>
```

JSON_ARRAY_LENGTH

函数语法：



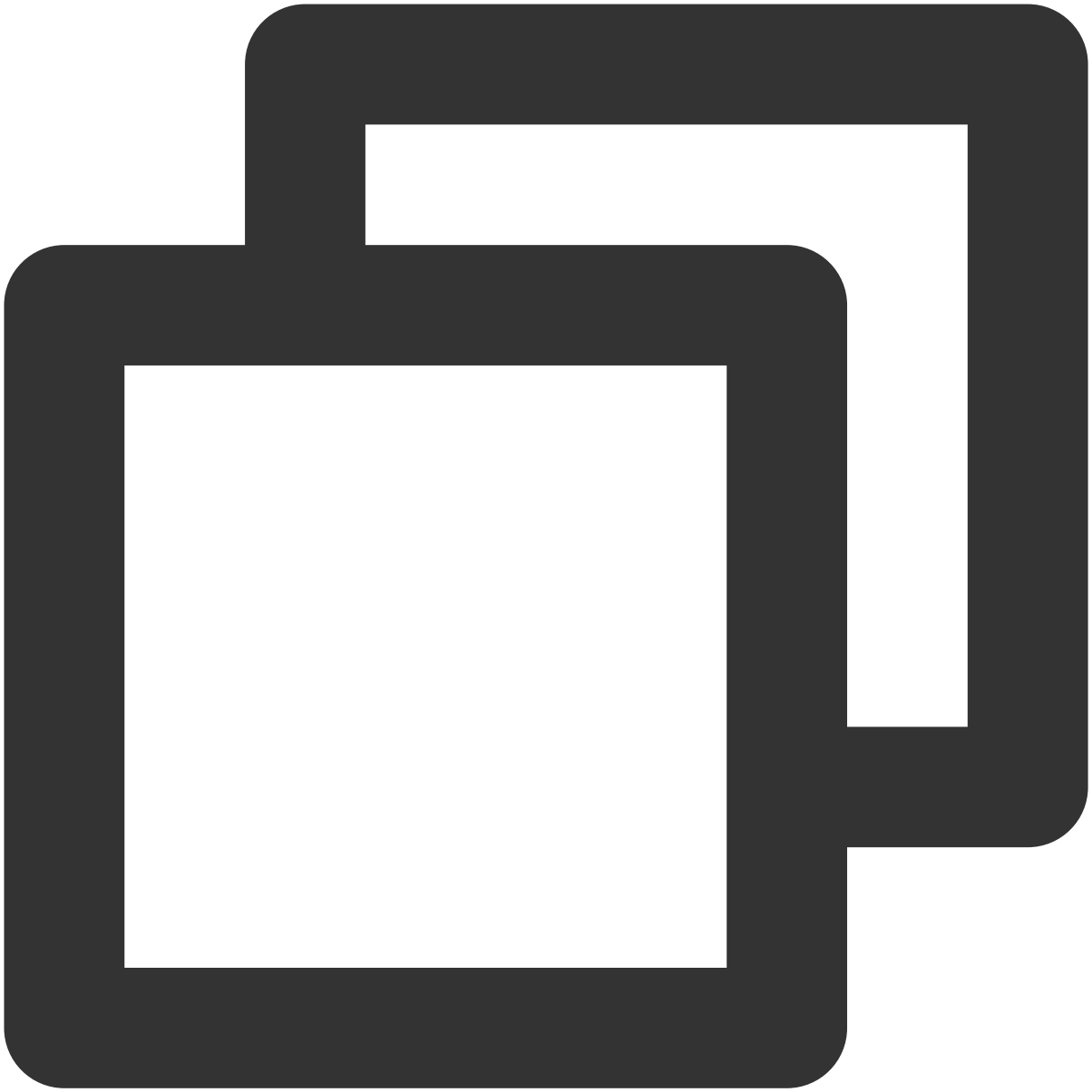
```
JSON_ARRAY_LENGTH(<jsonArray> string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回最外层 JSON 数组中的元素数。

返回类型：integer。

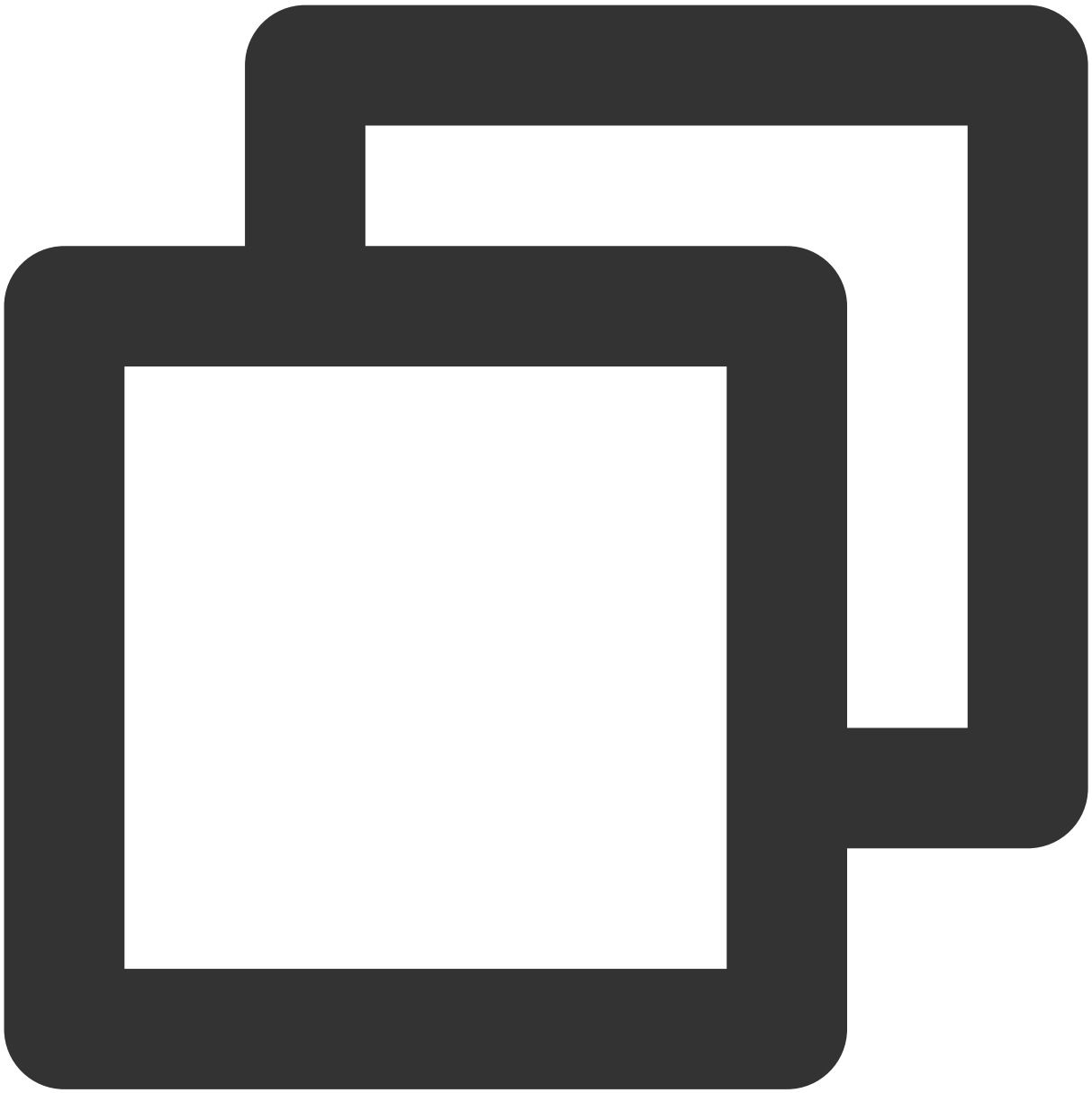
示例：



```
> SELECT json_array_length('[1,2,3,4]');
4
> SELECT json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4]');
5
> SELECT json_array_length('[1,2');
NULL
```

JSON_OBJECT_KEYS

函数语法：



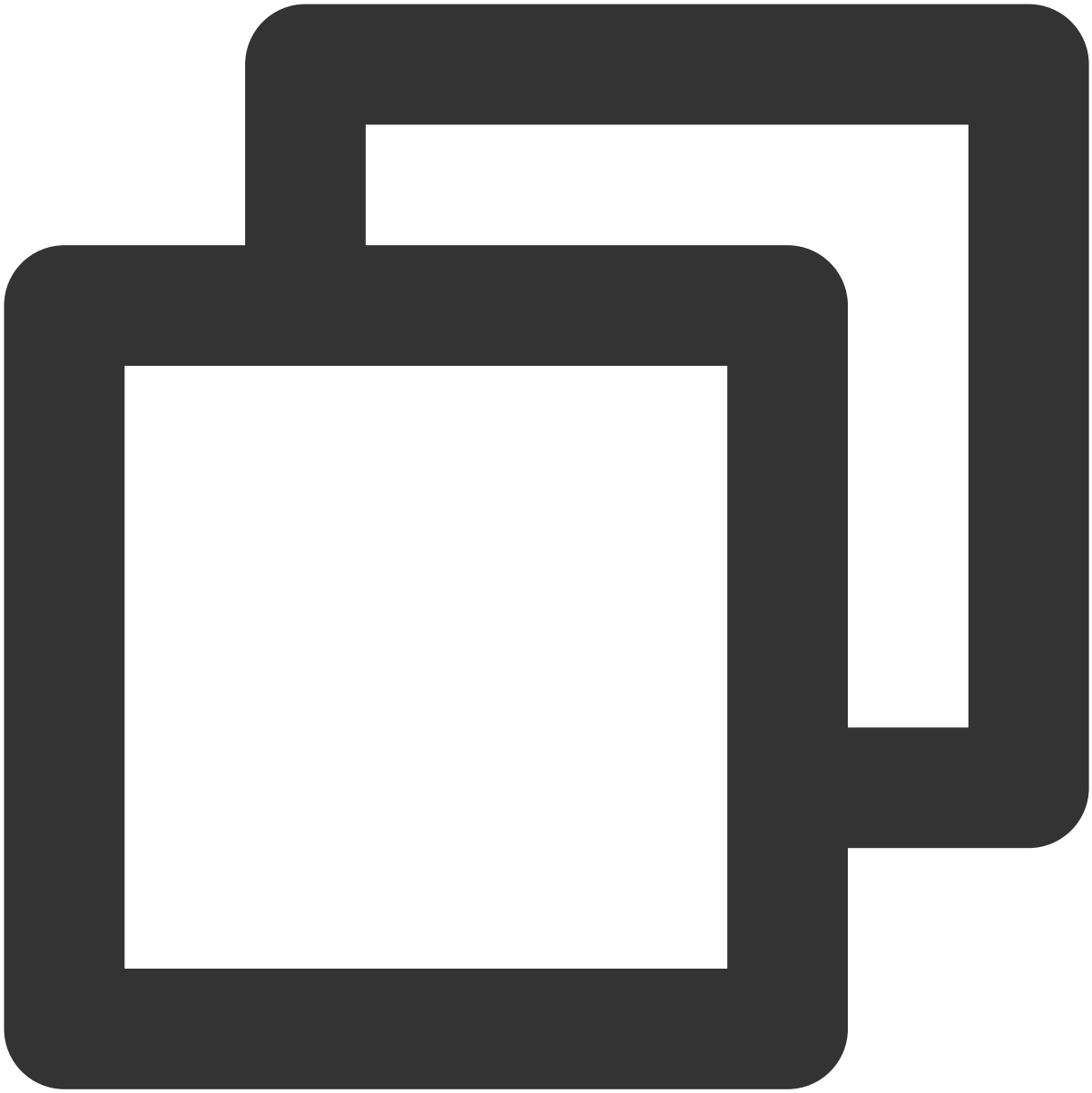
```
JSON_OBJECT_KEYS(<json> string)
```

支持引擎：SparkSQL、Presto。

使用说明：以数组形式返回最外层 JSON 对象的所有键。

返回类型：array<string>。

示例：



```
> SELECT json_object_keys('{}');
[]
> SELECT json_object_keys('{"key": "value"}');
["key"]
> SELECT json_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}}');
["f1","f2"]
```

数学函数

最近更新时间：2024-08-07 17:32:40

ABS

函数语法：

```
ABS(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的绝对值

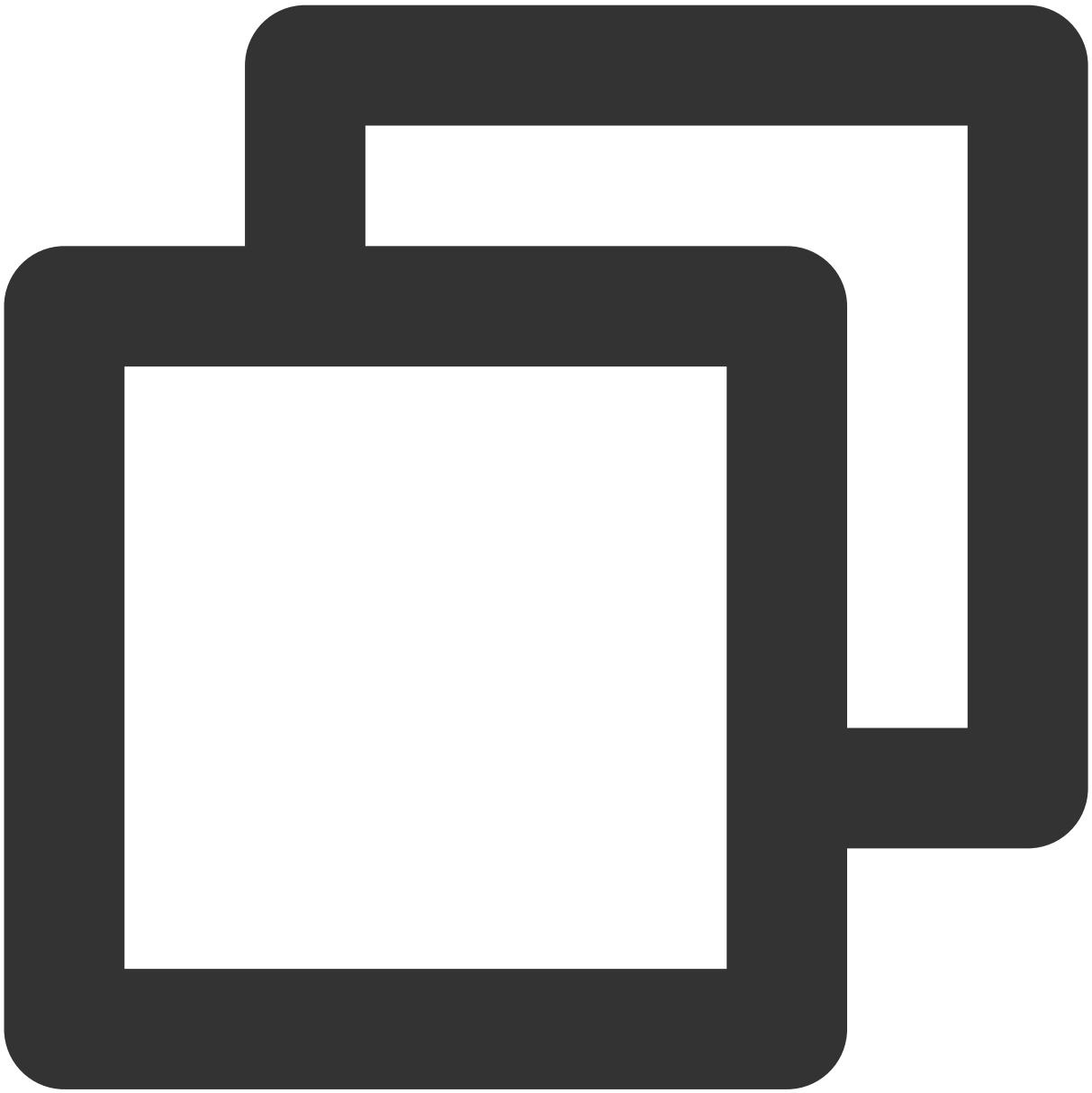
返回类型：与<expr>一致

示例：

```
> SELECT abs(-1);
 1
```

ACOS

函数语法：



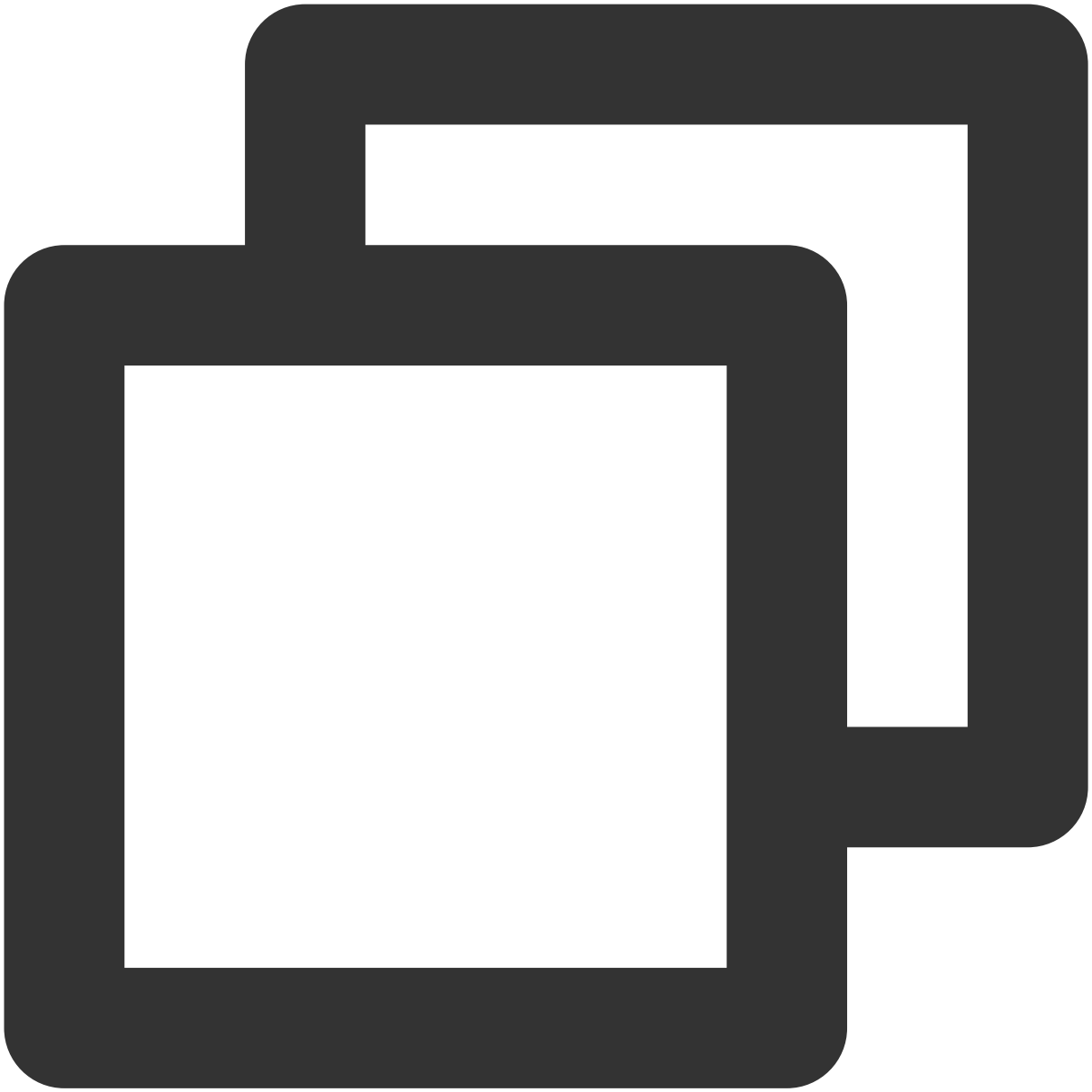
```
ACOS(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的反余弦

返回类型：double

示例：



```
> SELECT acos(1);
 0.0
> SELECT acos(2);
 NaN
```

ACOSH

函数语法：

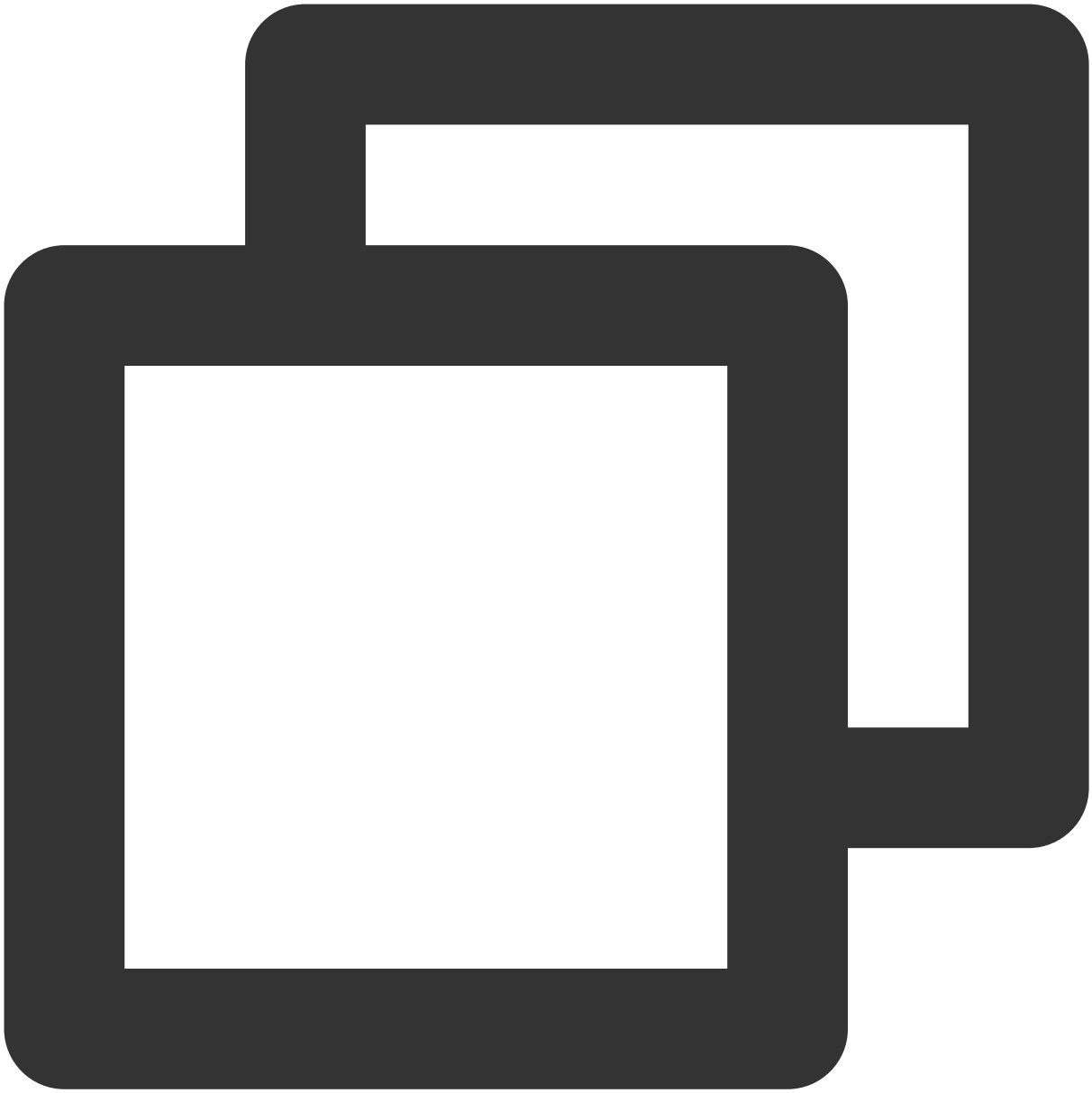
```
ACOSH(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的反双曲余弦值

返回类型：double

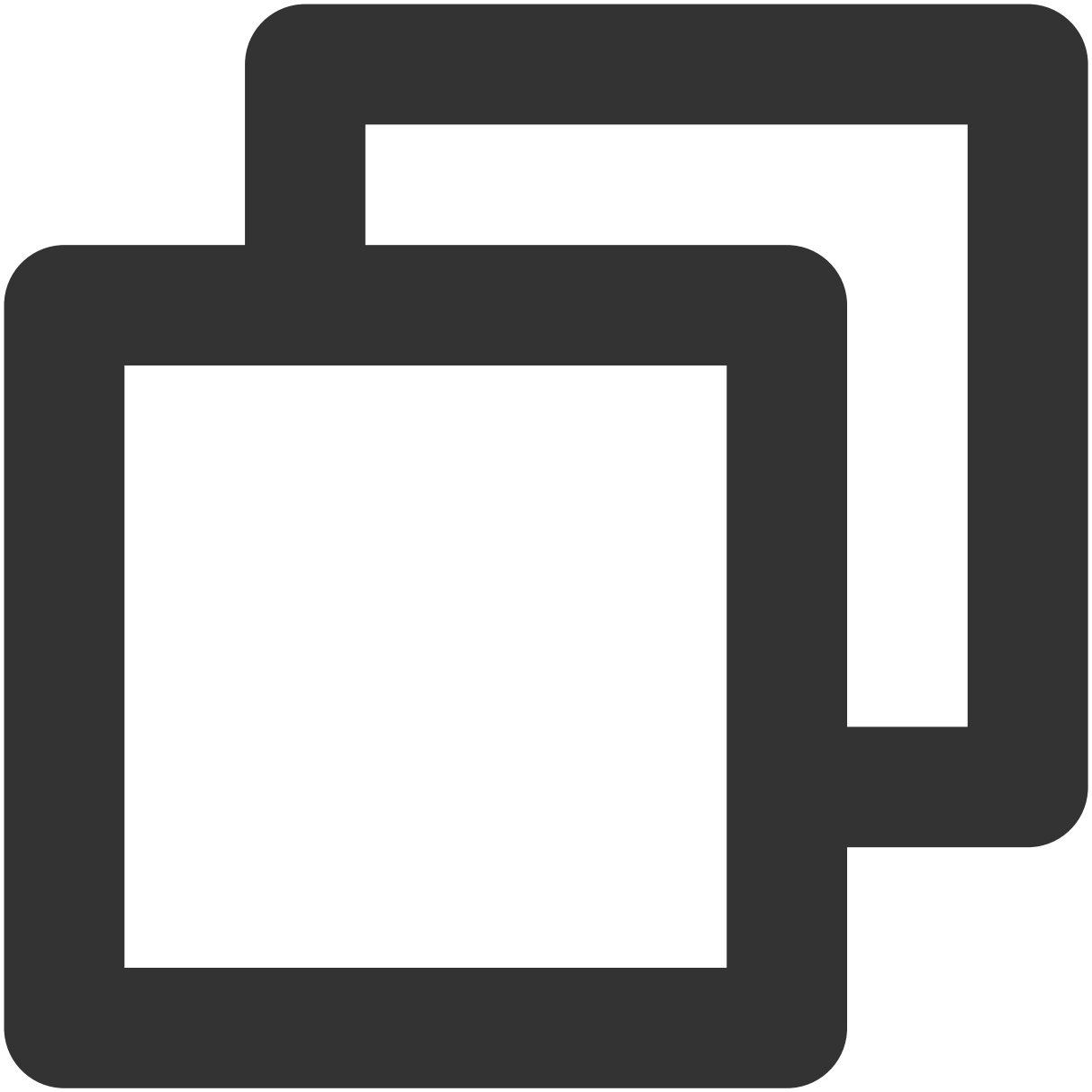
示例：



```
> SELECT acosh(1);
 0.0
> SELECT acosh(0);
 NaN
```

ASIN

函数语法：



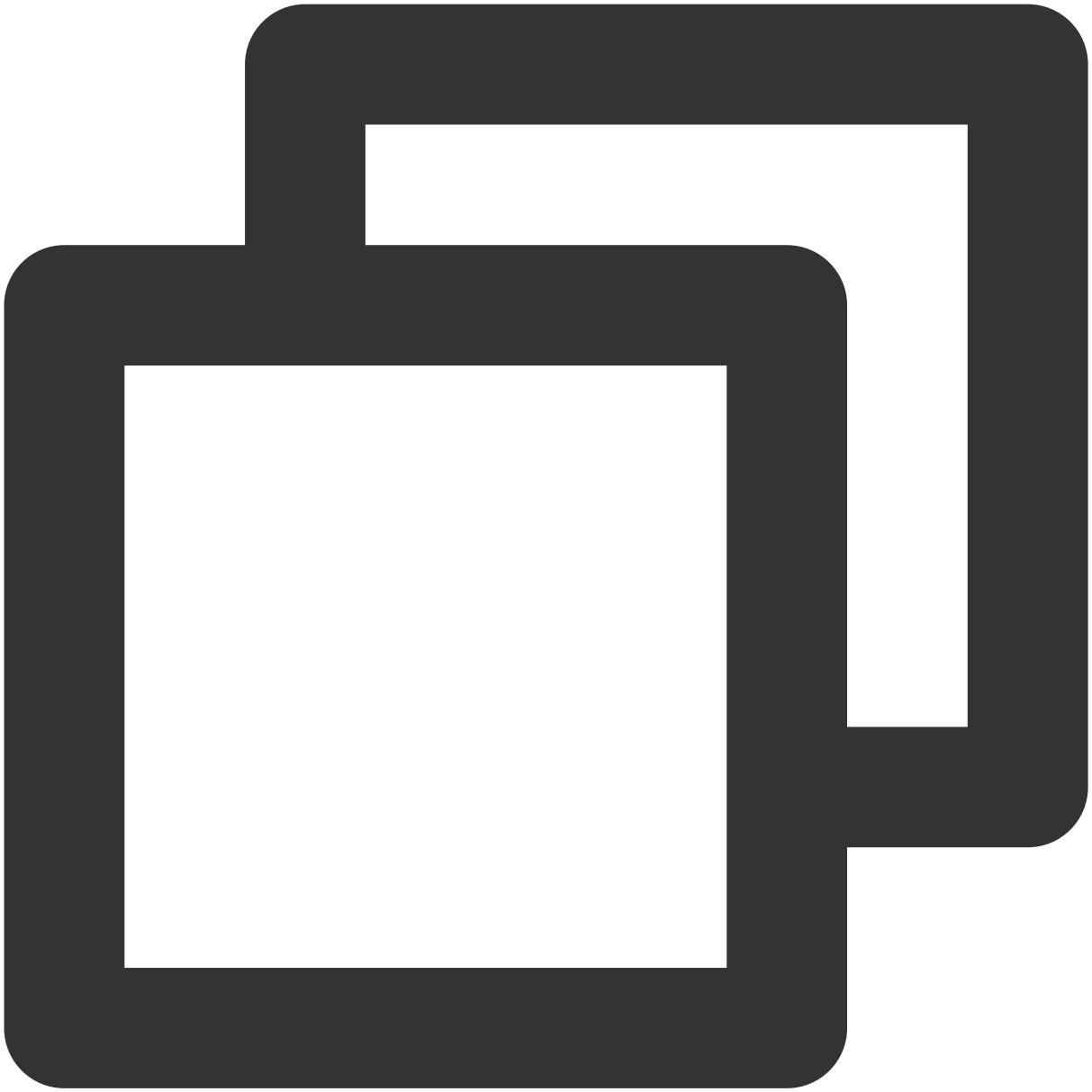
```
ASIN(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的反正弦

返回类型：double

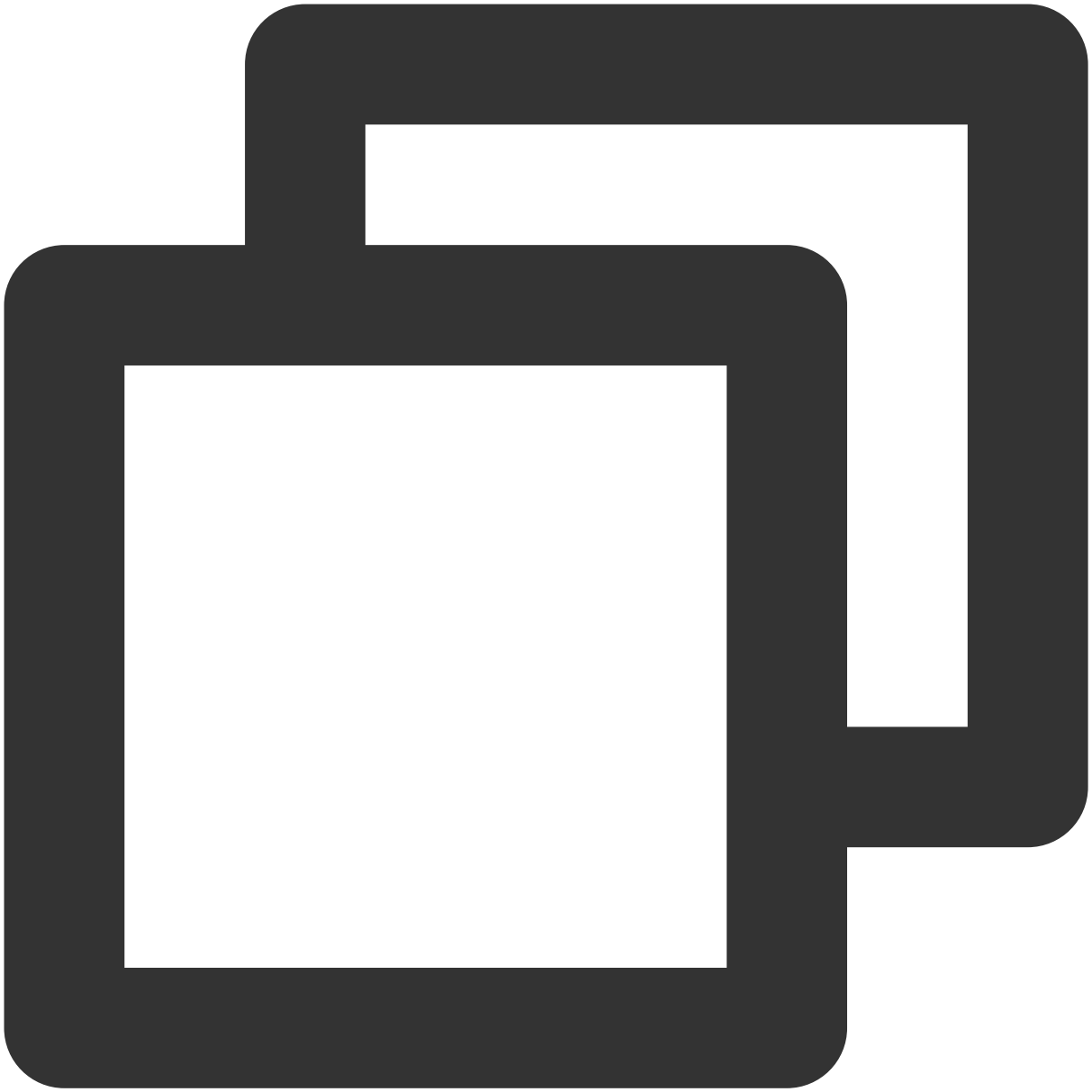
示例：



```
> SELECT asin(0);
 0.0
> SELECT asin(2);
 NaN
```

ASINH

函数语法：



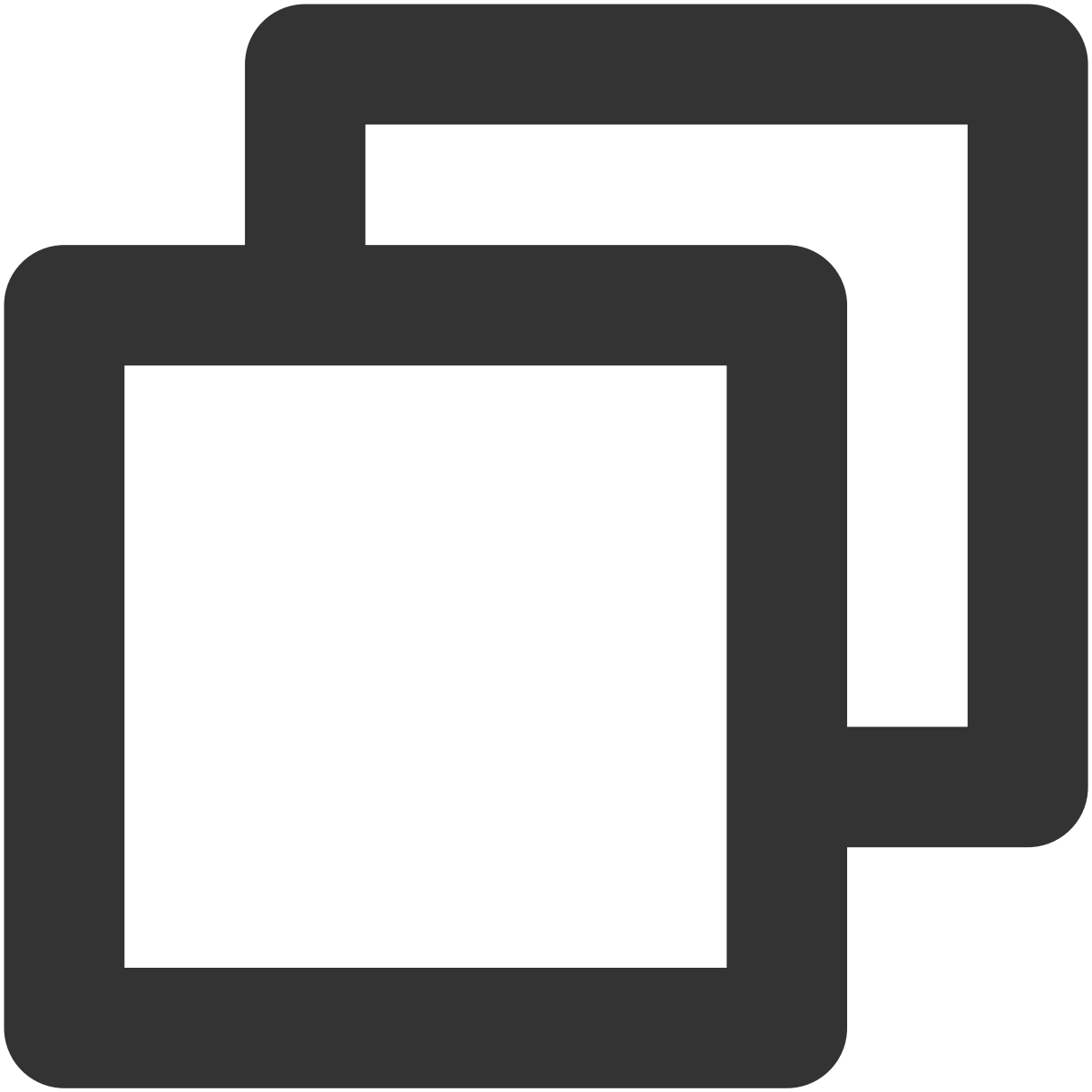
```
ASINH(<expr> integer|double|decimal)
```

支持引擎：SparkSQL

使用说明：返回expr的反双曲正弦值。

返回类型：double

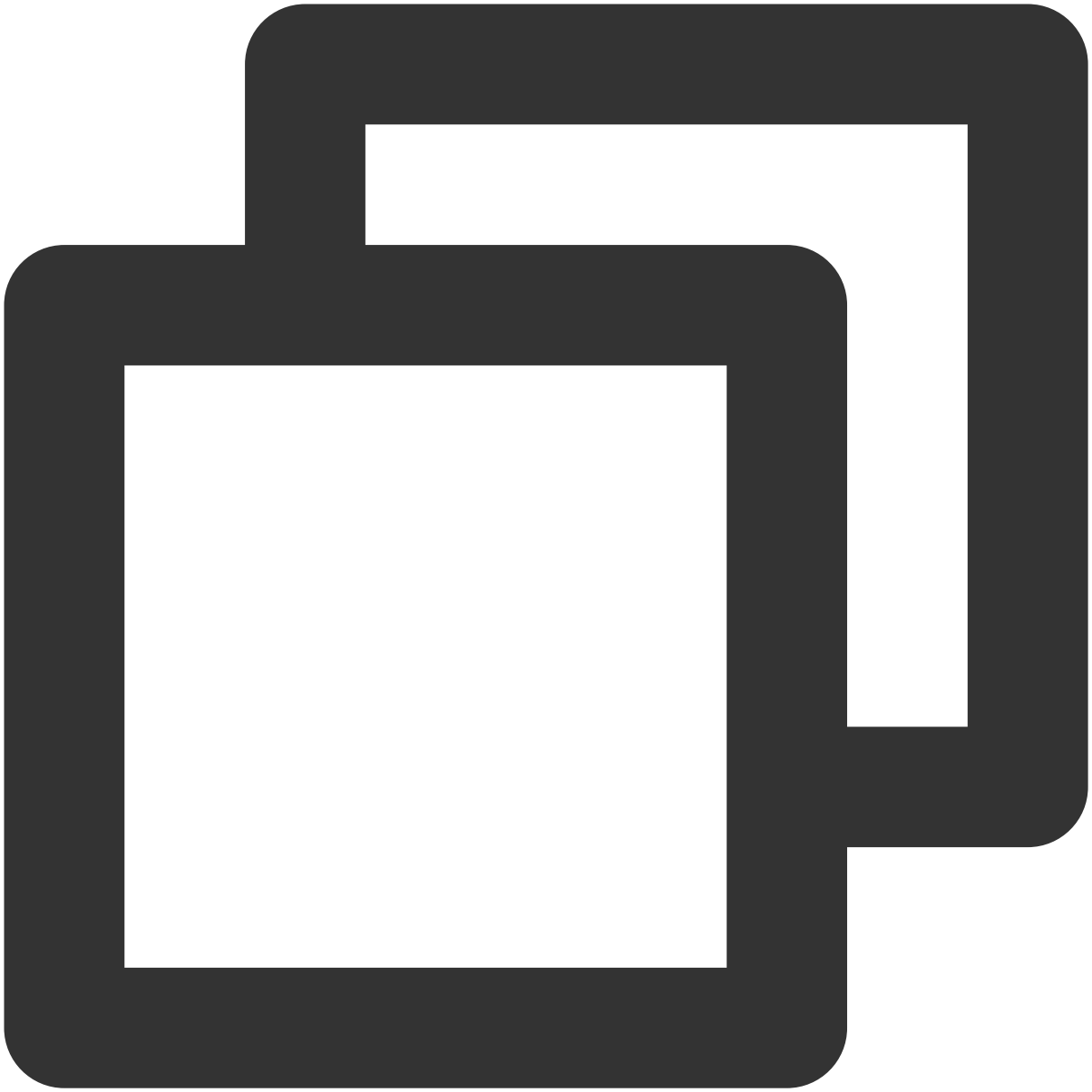
示例：



```
> SELECT asinh(0);
 0.0
```

ATAN

函数语法：



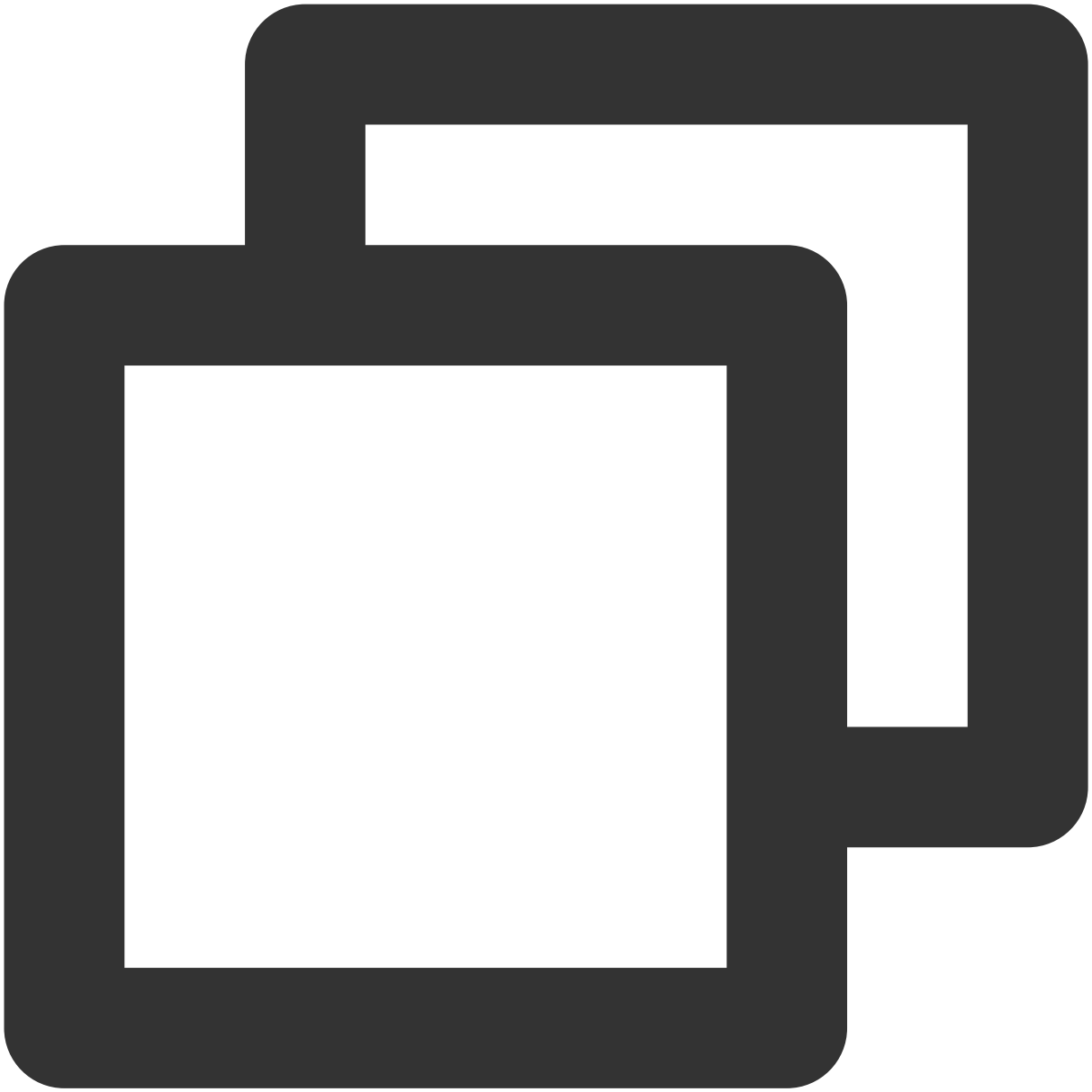
```
ATAN(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的反正切

返回类型：double

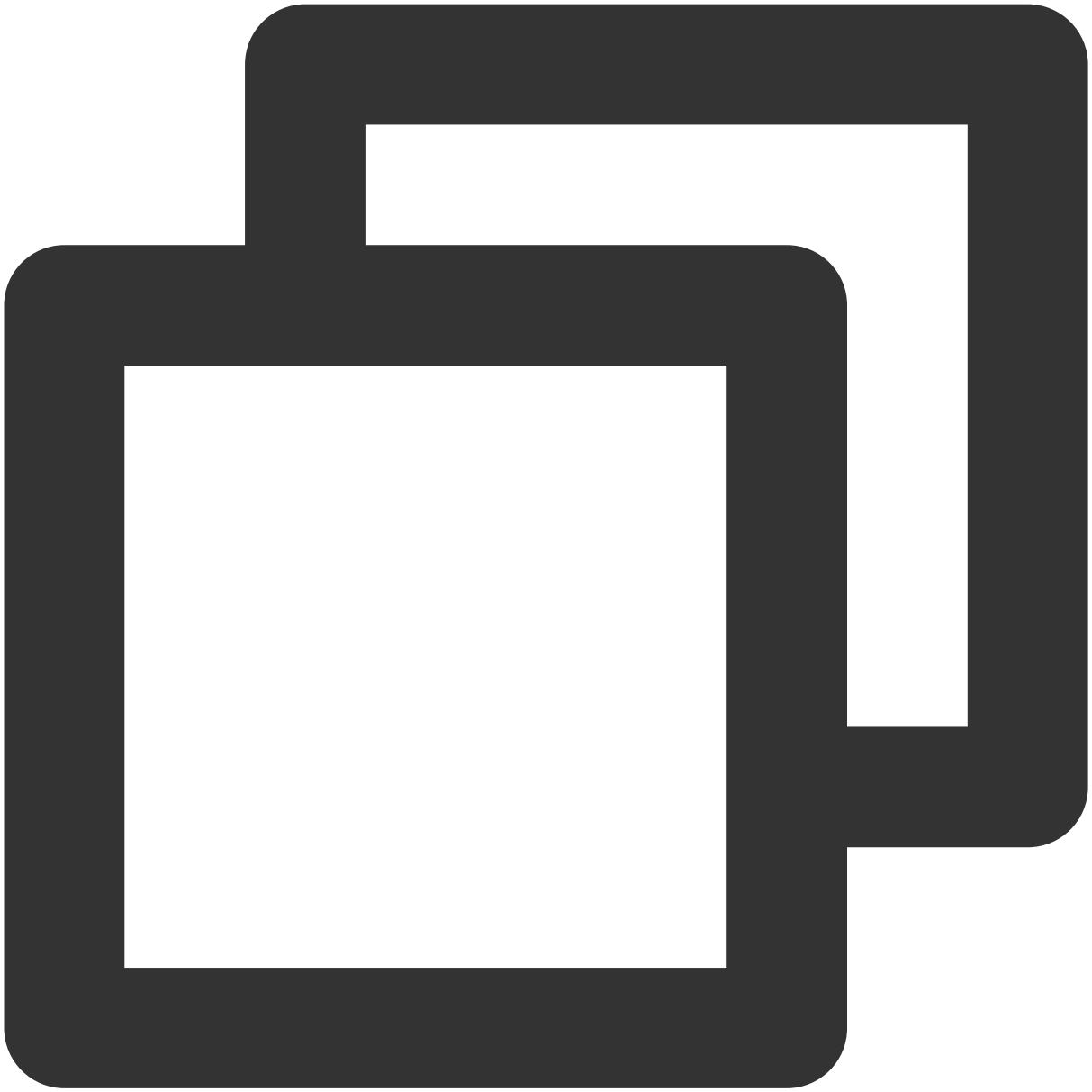
示例：



```
> SELECT atan(0);
 0.0
```

ATAN2

函数语法：



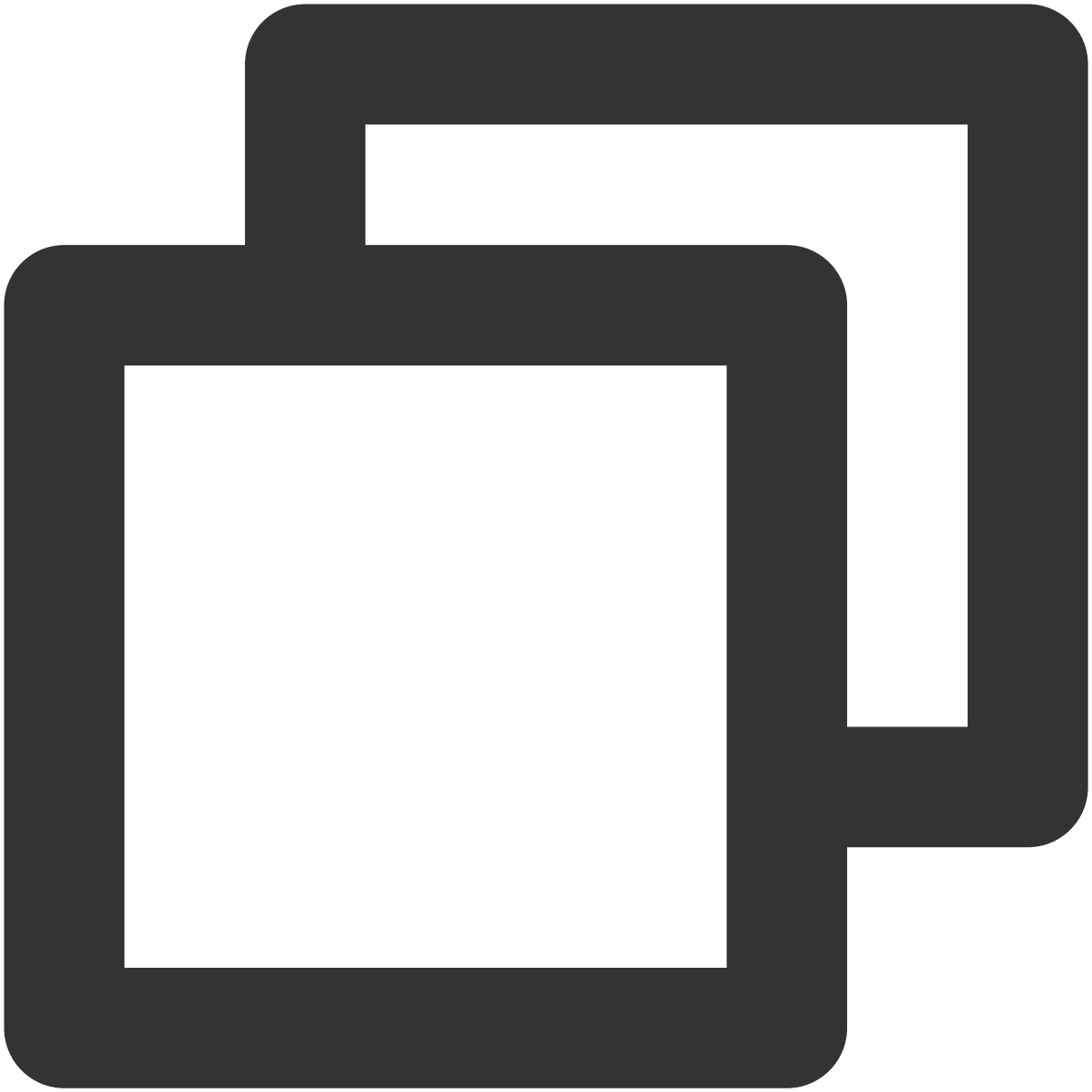
```
ATAN2(<x>, integer|double|decimal, <y> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回平面的正 x 轴与坐标给定的点之间的弧度角

返回类型：double

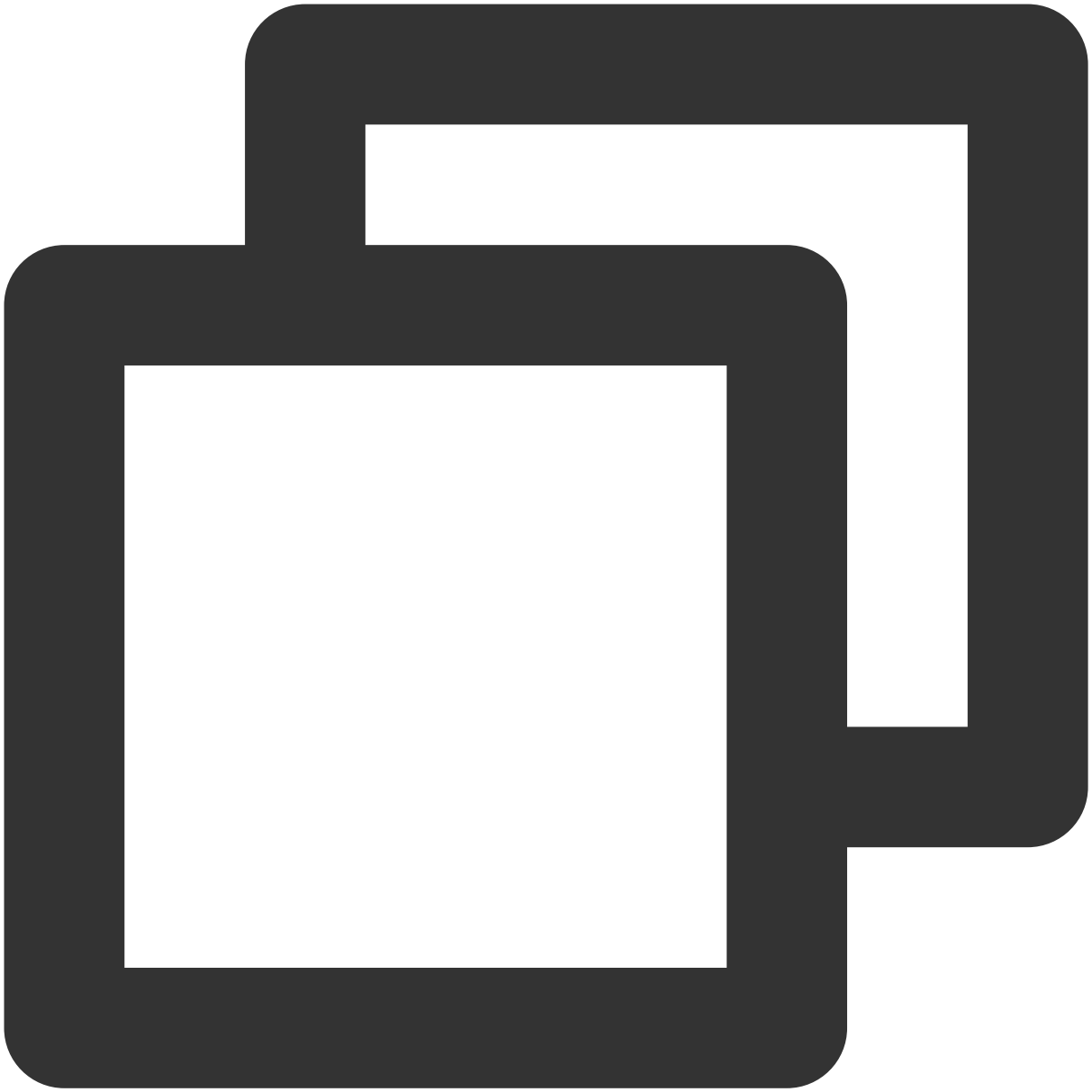
示例：



```
> SELECT atan2(0, 0);
 0.0
```

ATANH

函数语法：



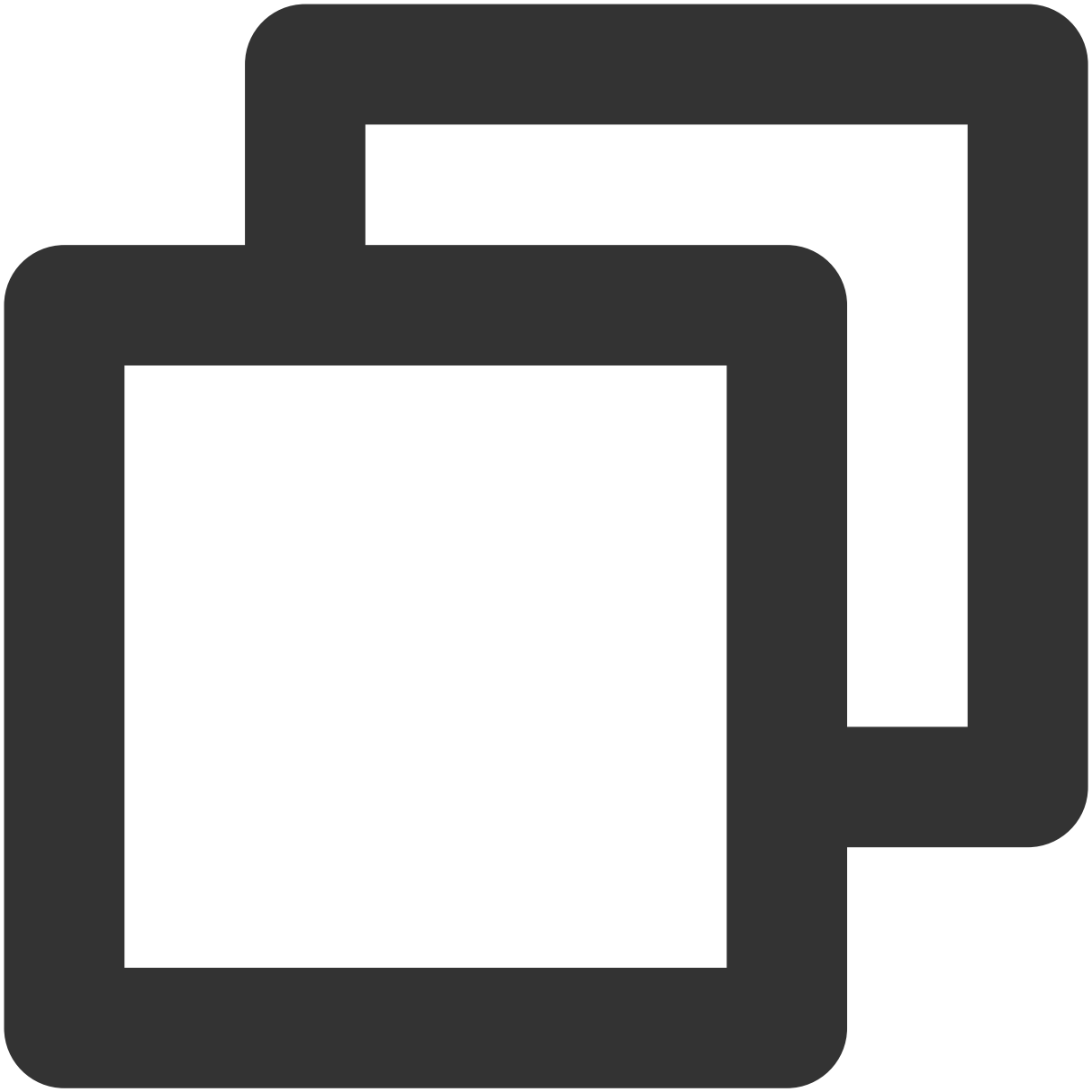
```
ATANH(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回 expr 的反双曲正切。

返回类型：double

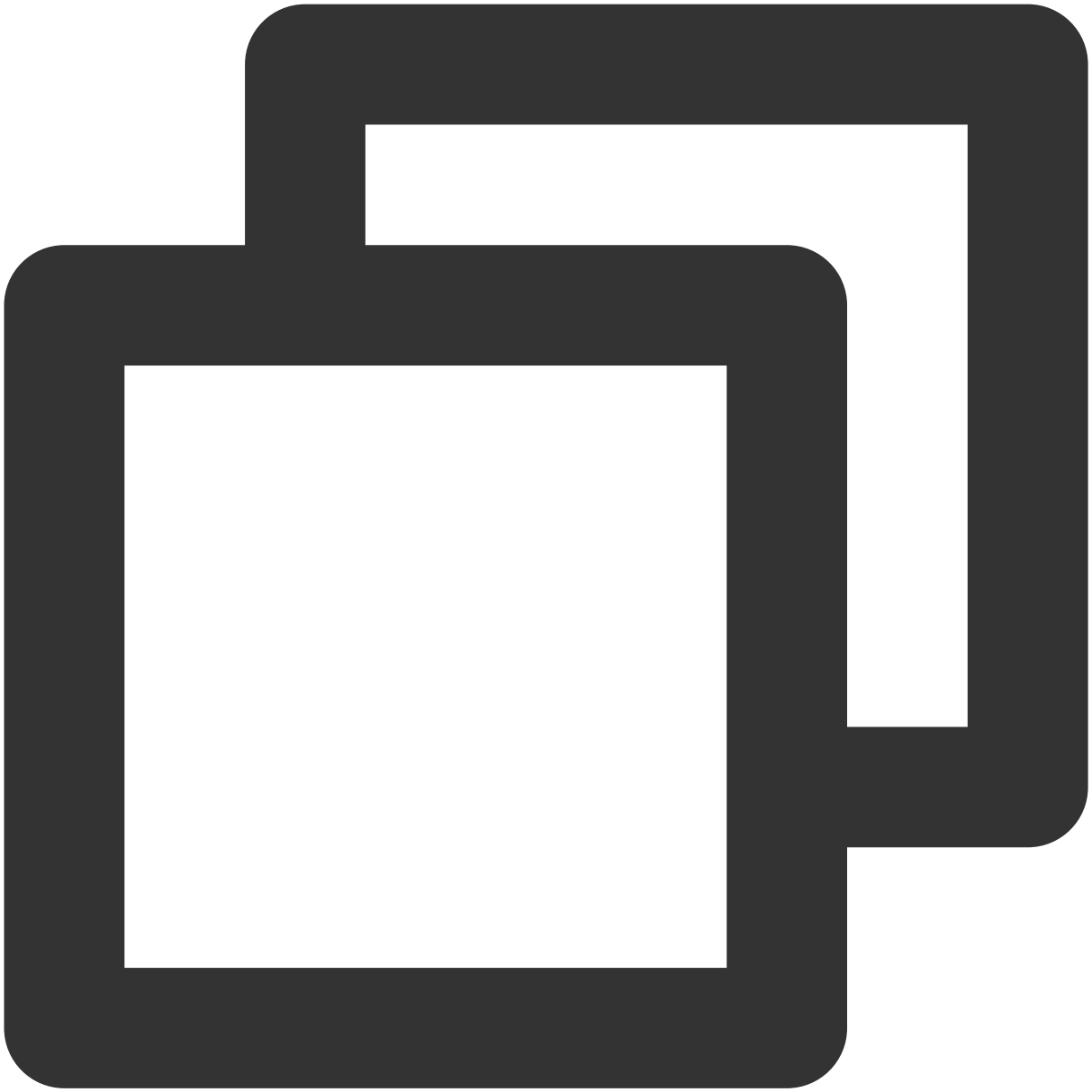
示例：



```
> SELECT atanh(0);
 0.0
> SELECT atanh(2);
 NaN
```

BIN

函数语法：



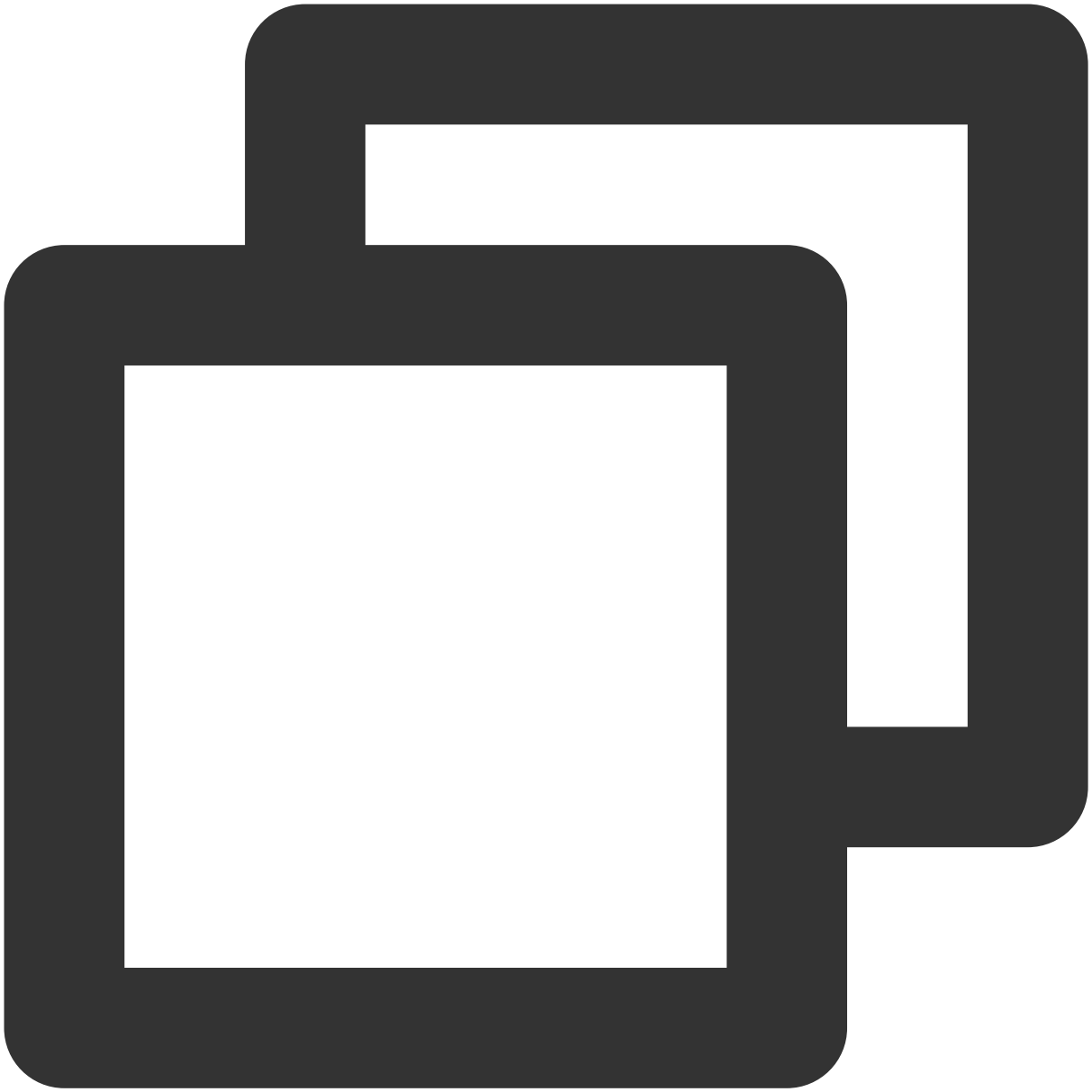
```
BIN(<expr> integer)
```

支持引擎：SparkSQL、Presto

使用说明：返回以二进制表示的长值expr的字符串表示形式。

返回类型：string

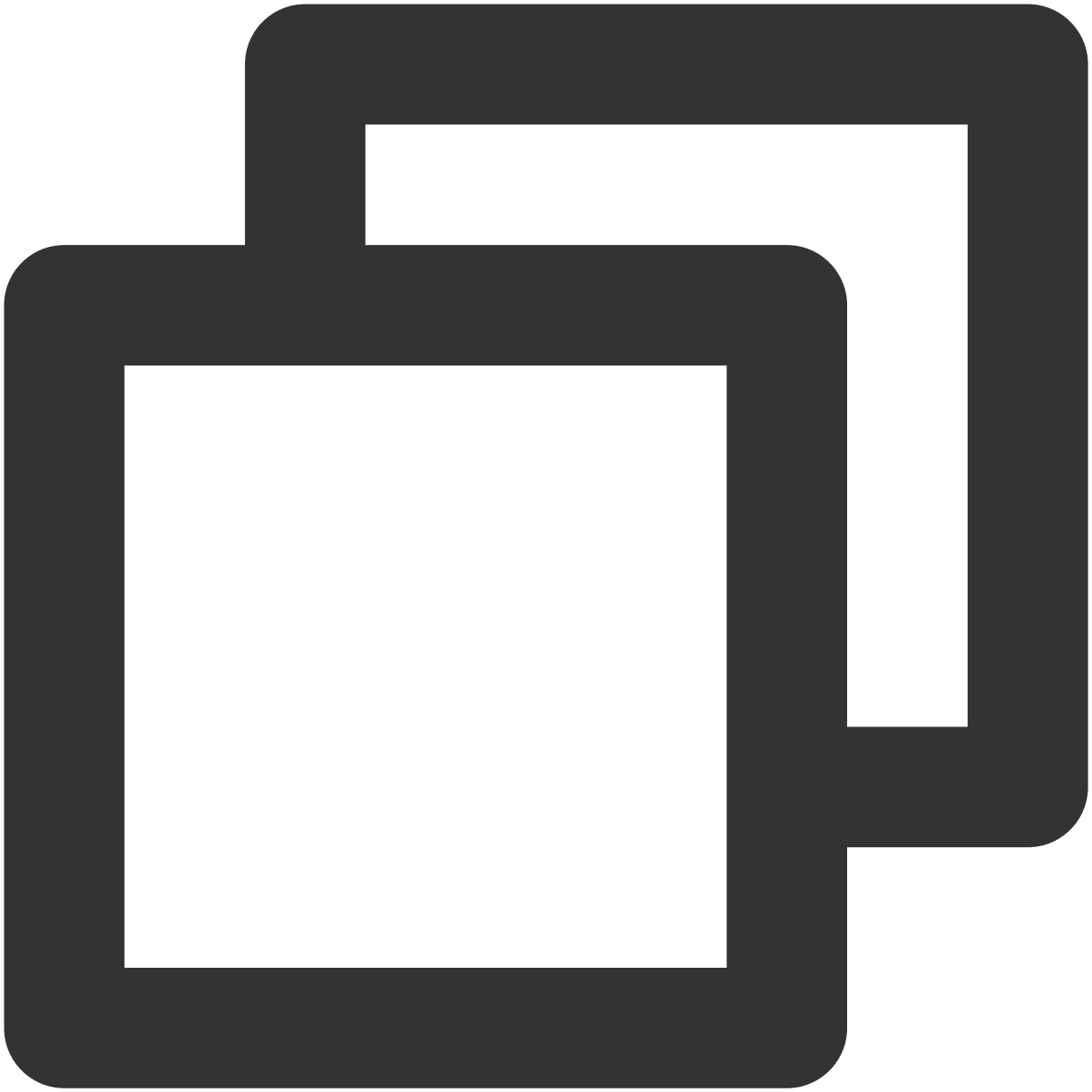
示例：



```
> SELECT bin(13);
 1101
> SELECT bin(-13);
 1111111111111111111111111111111111111111111111111111111111110011
```

BROUND

函数语法：



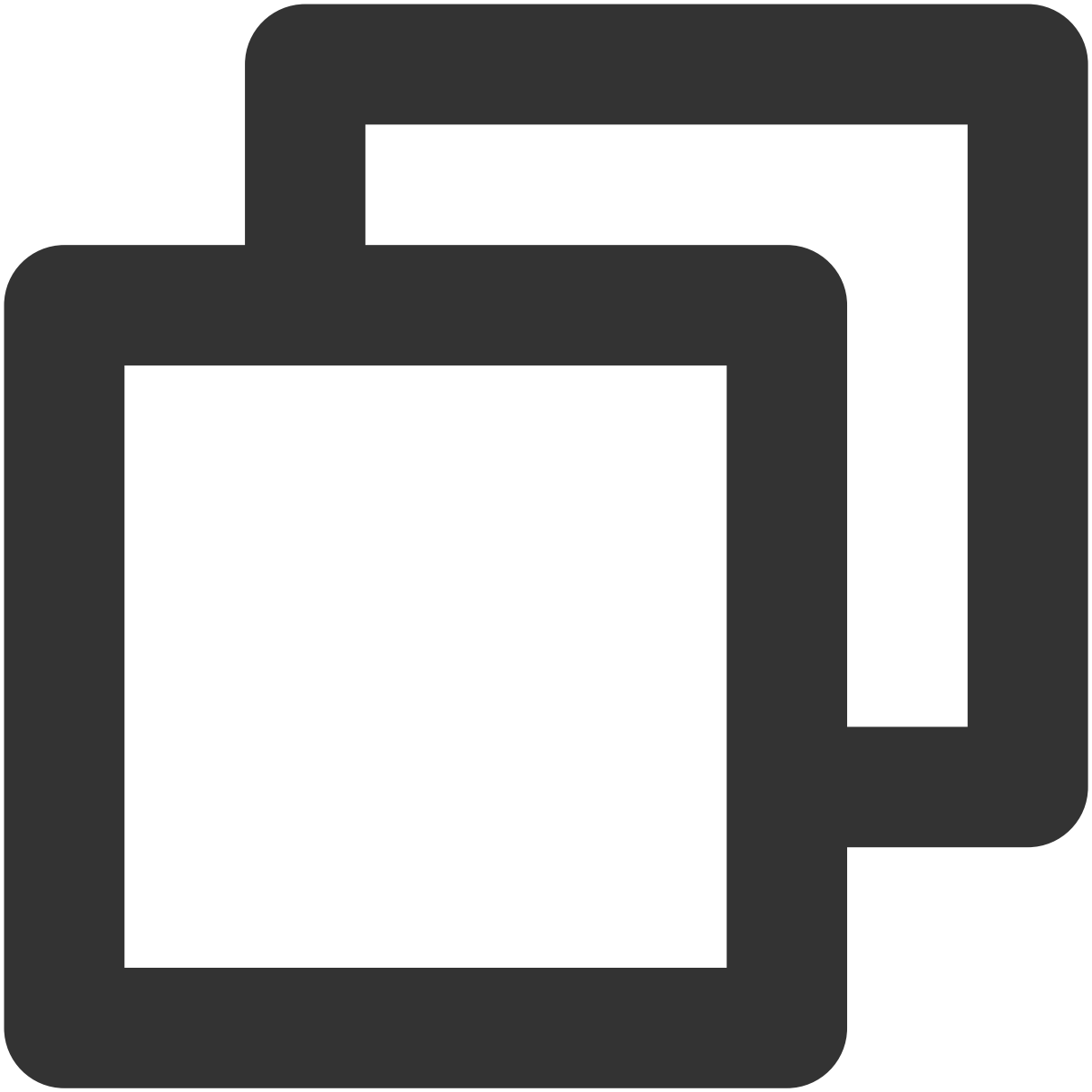
```
BROUND(<expr> integer|double|decimal, <d> integer)
```

支持引擎：SparkSQL、Presto

使用说明：返回 expr 使用 HALF_EVEN 舍入模式舍入到d位小数

返回类型：decimal

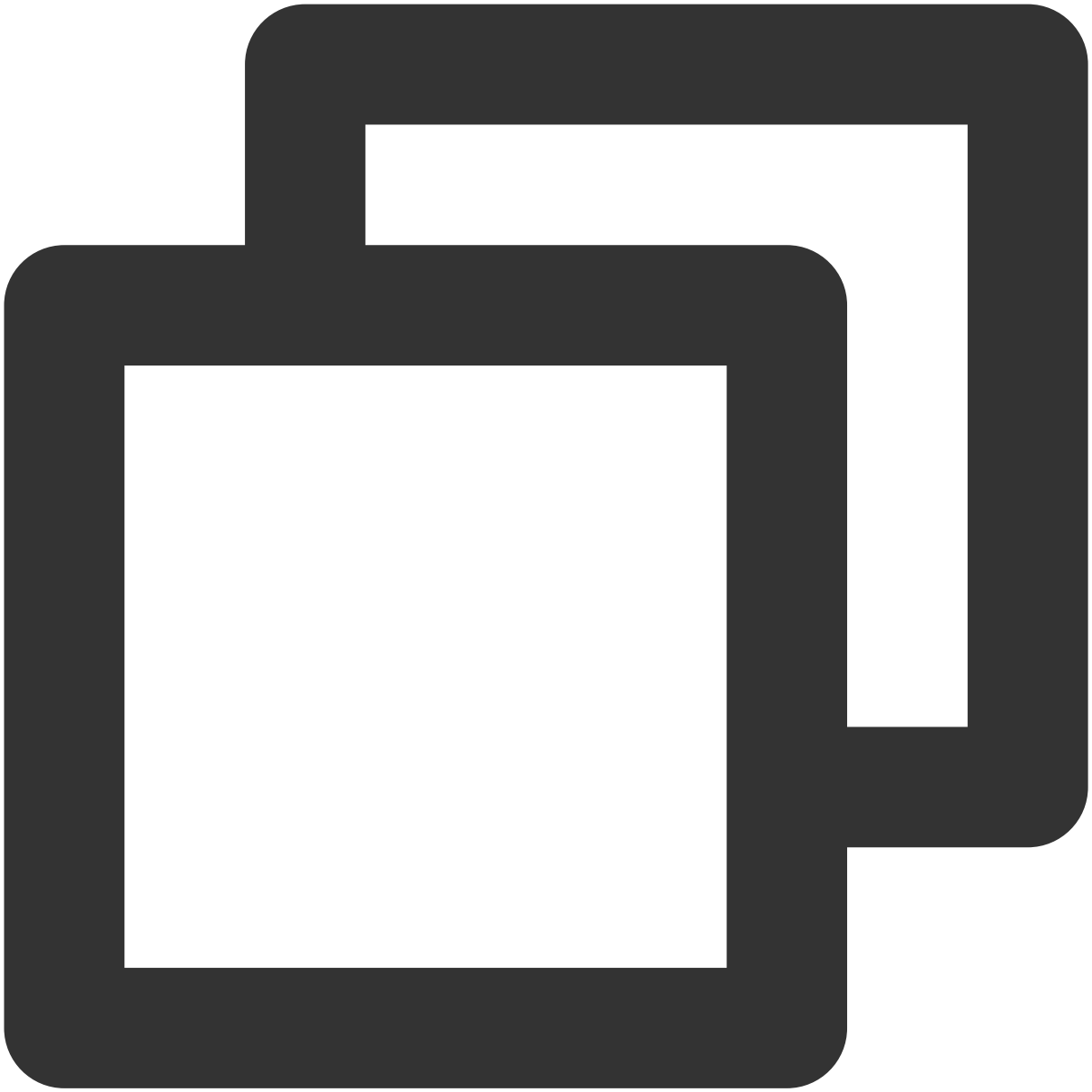
示例：



```
> SELECT bround(2.5, 0);
 2
```

CBRT

函数语法：



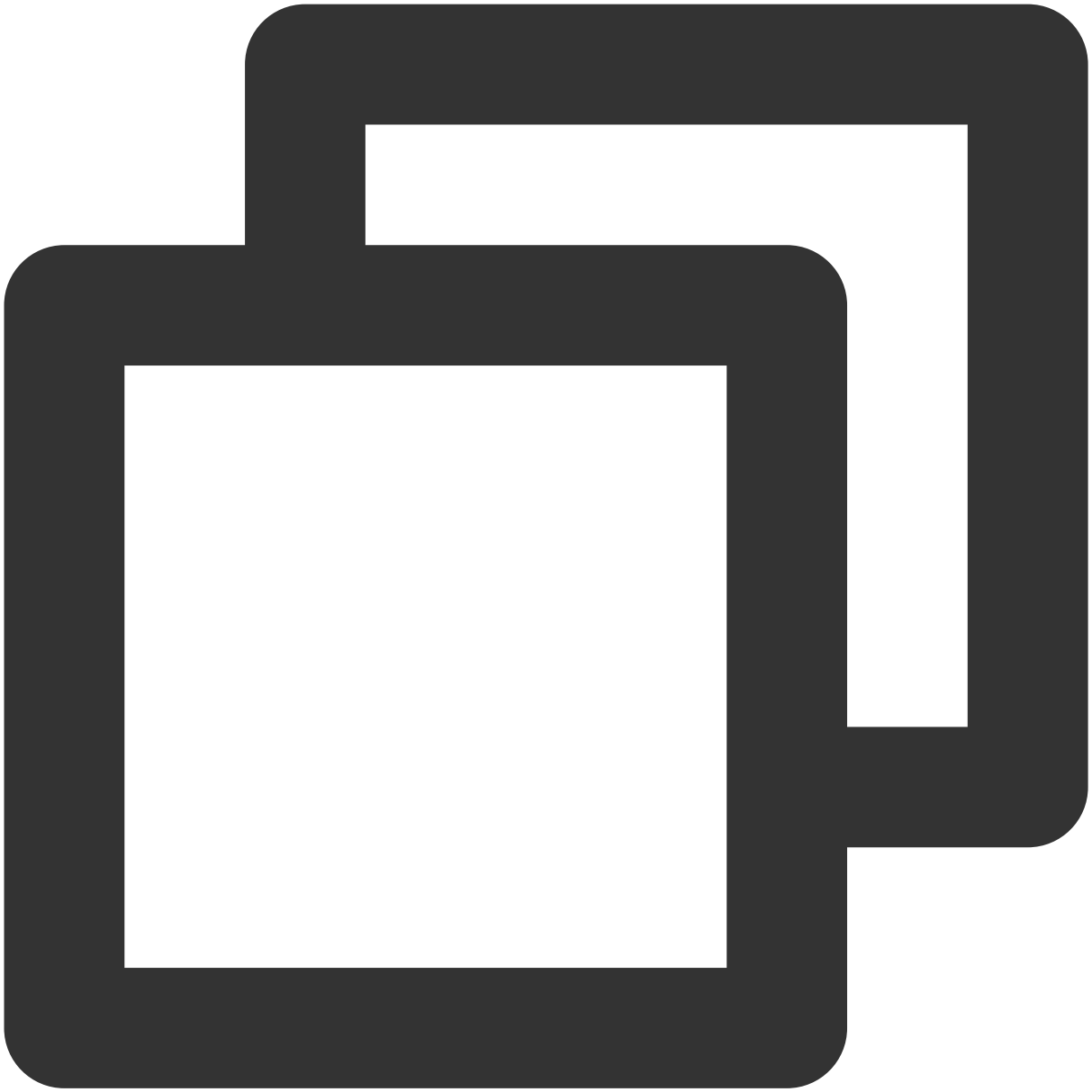
```
CBRT(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的立方根。

返回类型：double

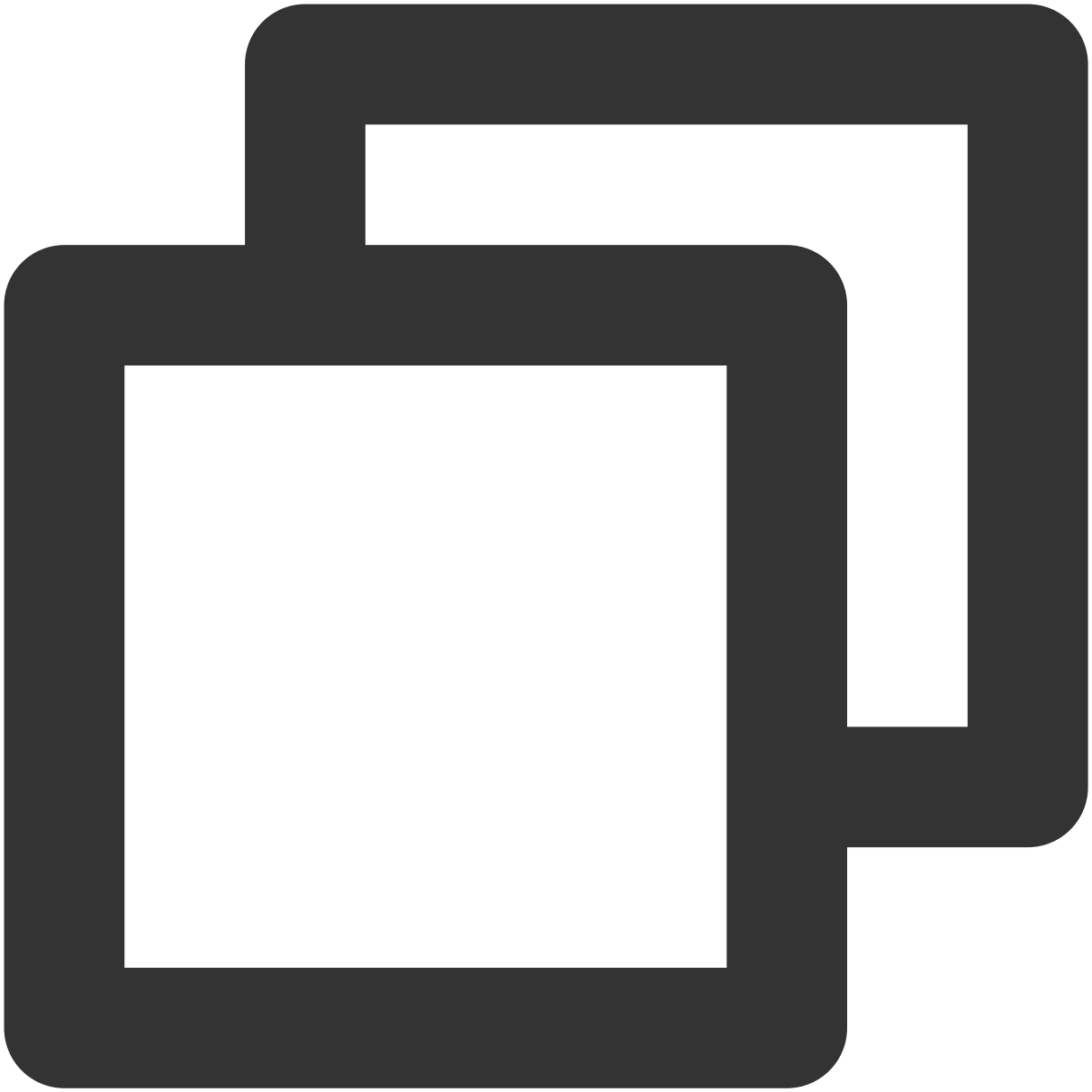
示例：



```
> SELECT cbrt(27.0);
 3.0
```

CEIL

函数语法：



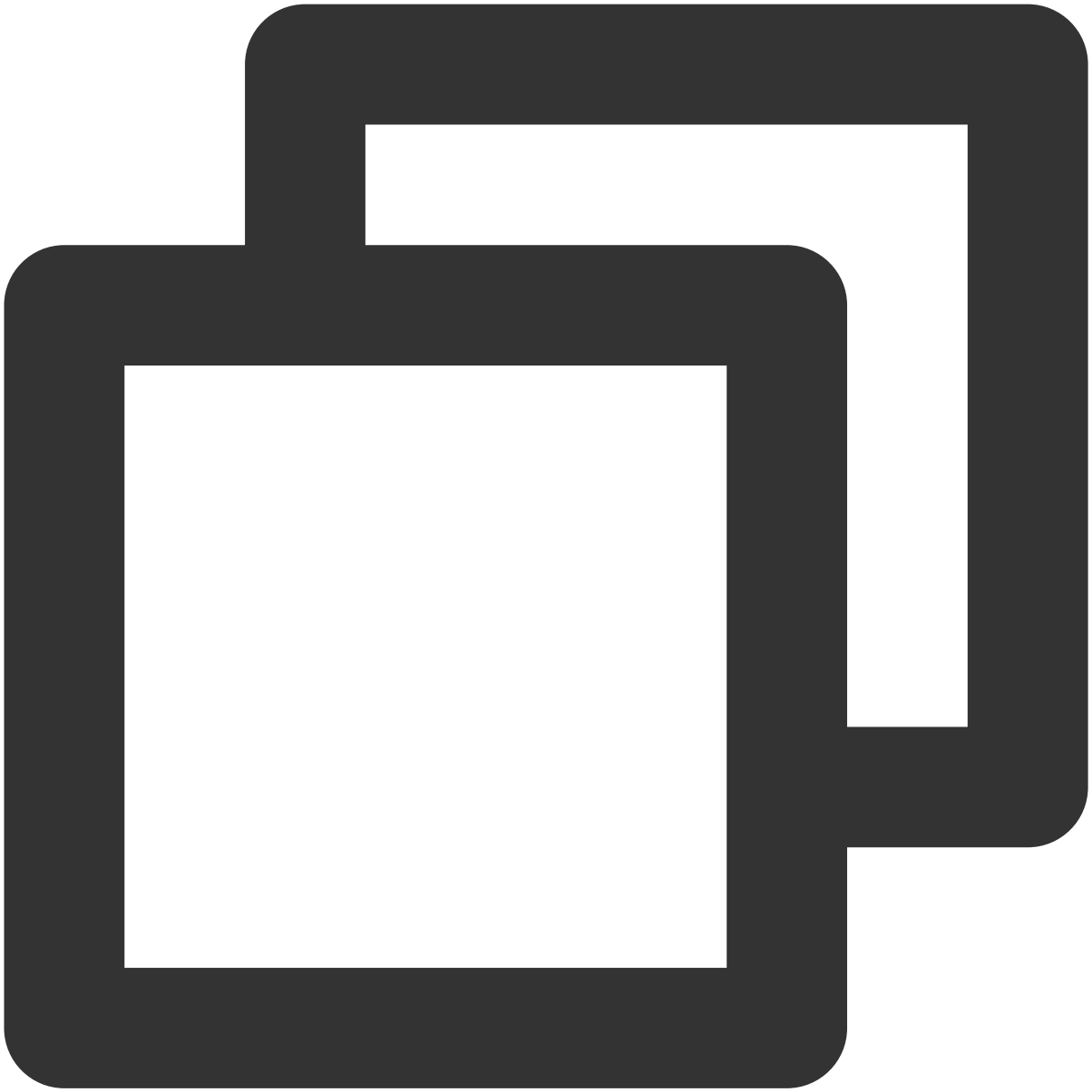
```
CEIL(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回不小于expr的最小整数

返回类型：integer

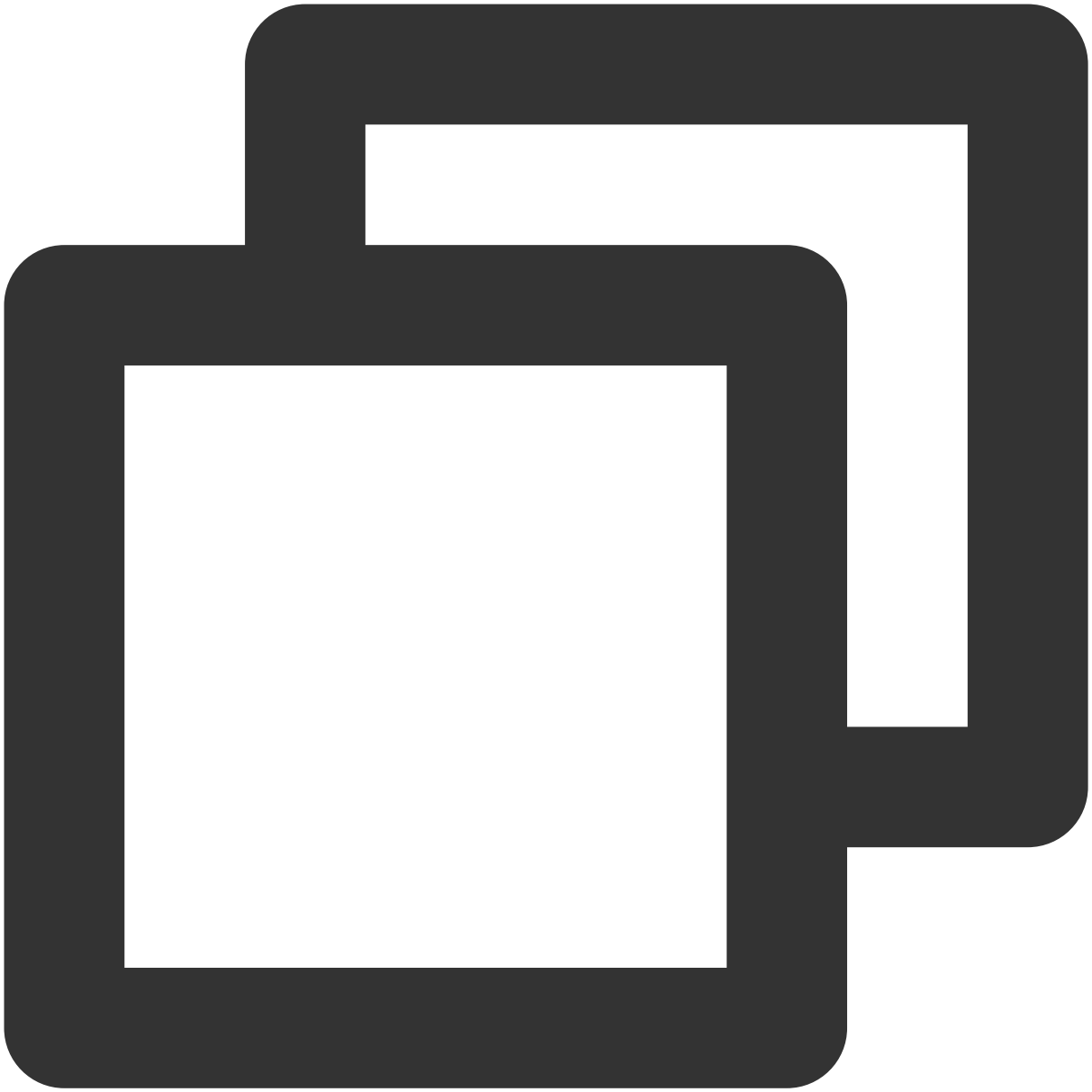
示例：



```
> SELECT ceil(-0.1);
 0
> SELECT ceil(5);
 5
```

COS

函数语法：



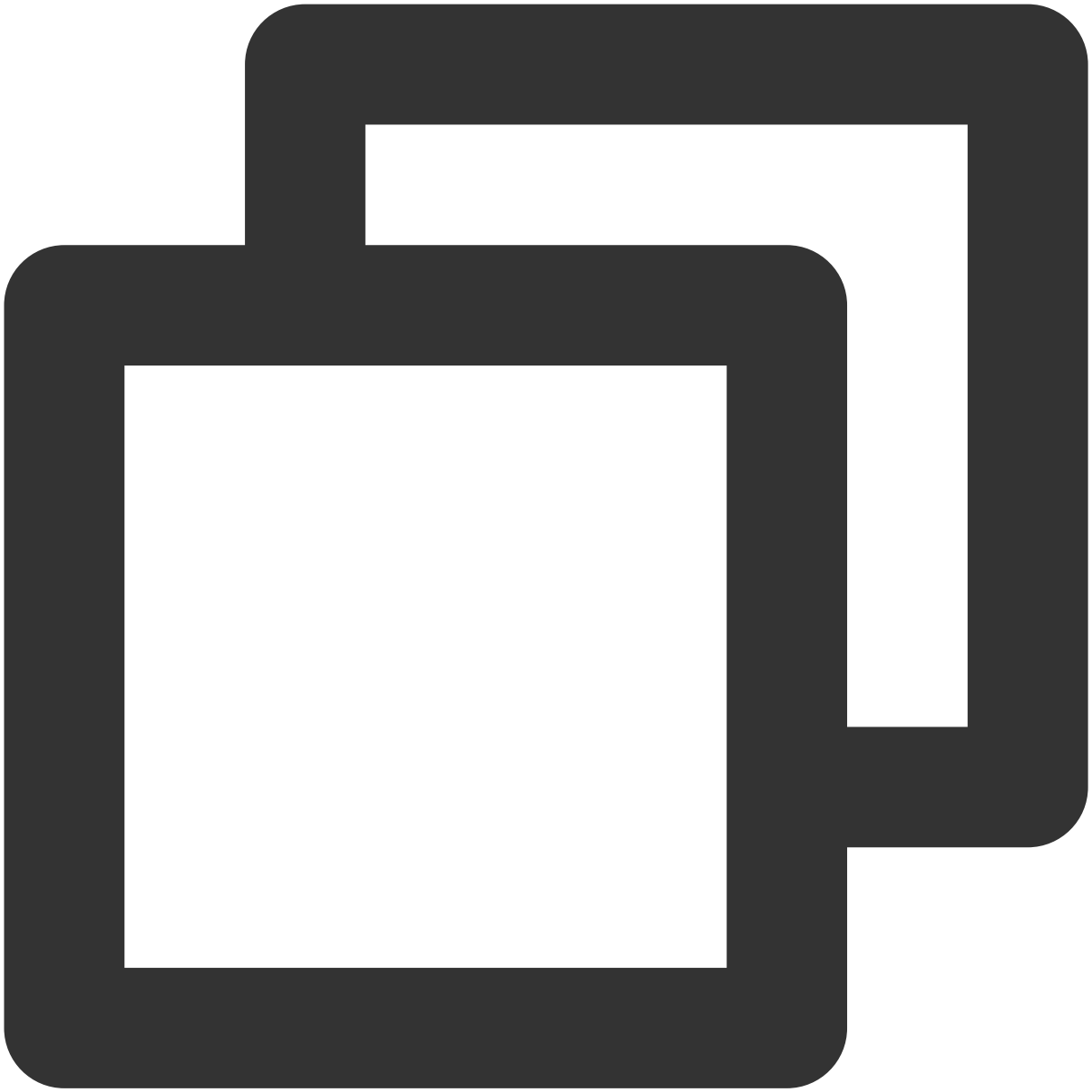
```
COS(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的余弦

返回类型：double

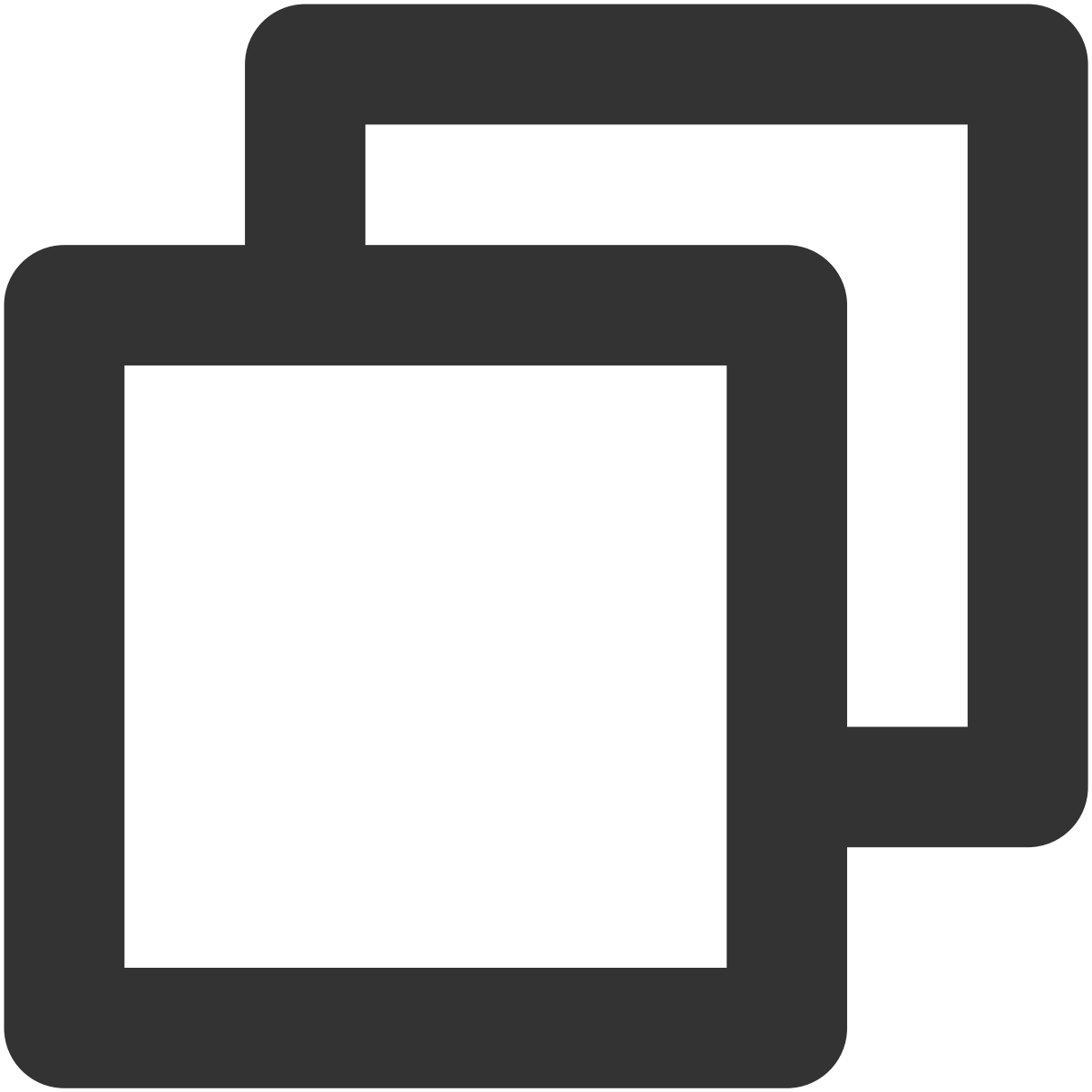
示例：



```
> SELECT cos(0);
 1.0
```

COSH

函数语法：



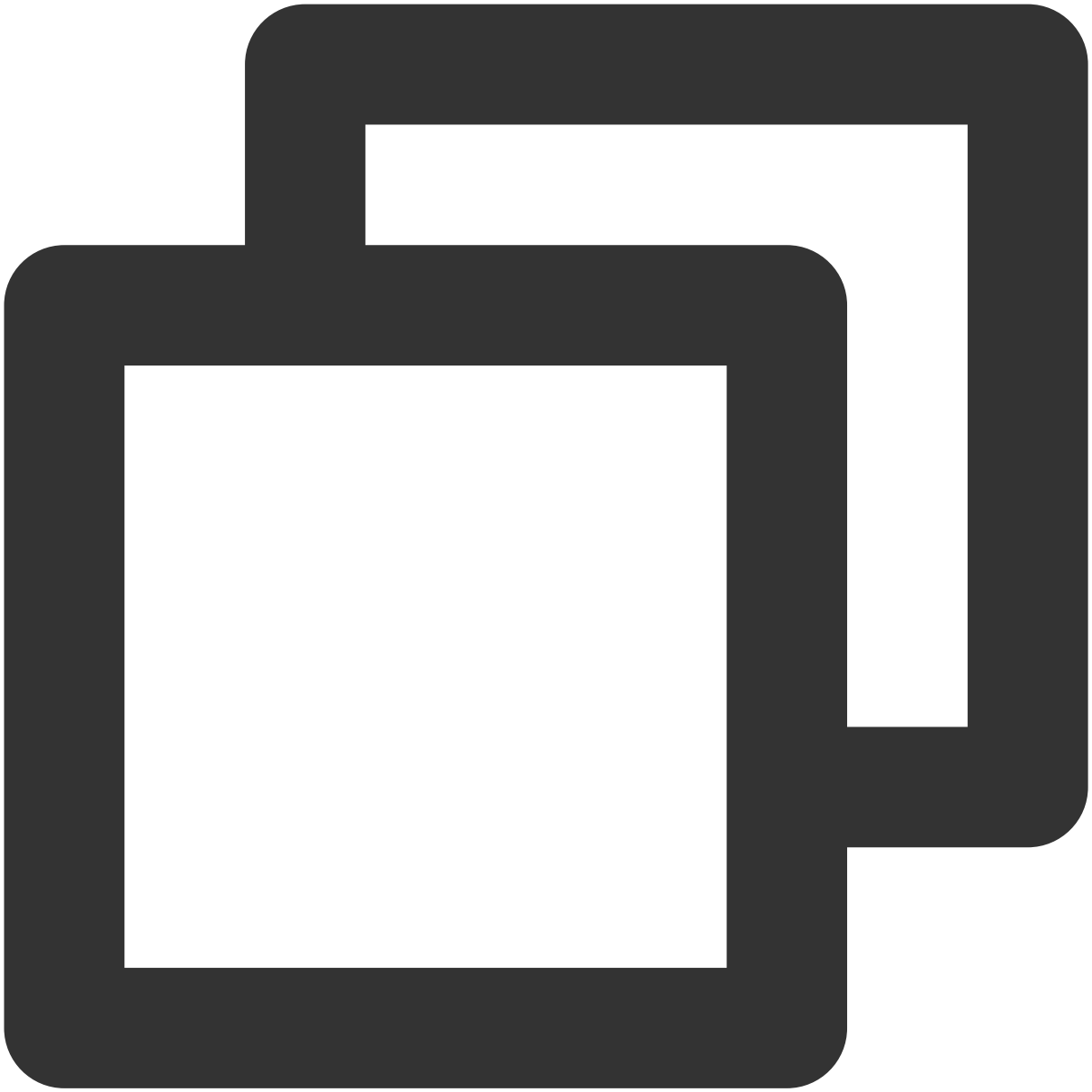
```
COSH(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的双曲余弦

返回类型：double

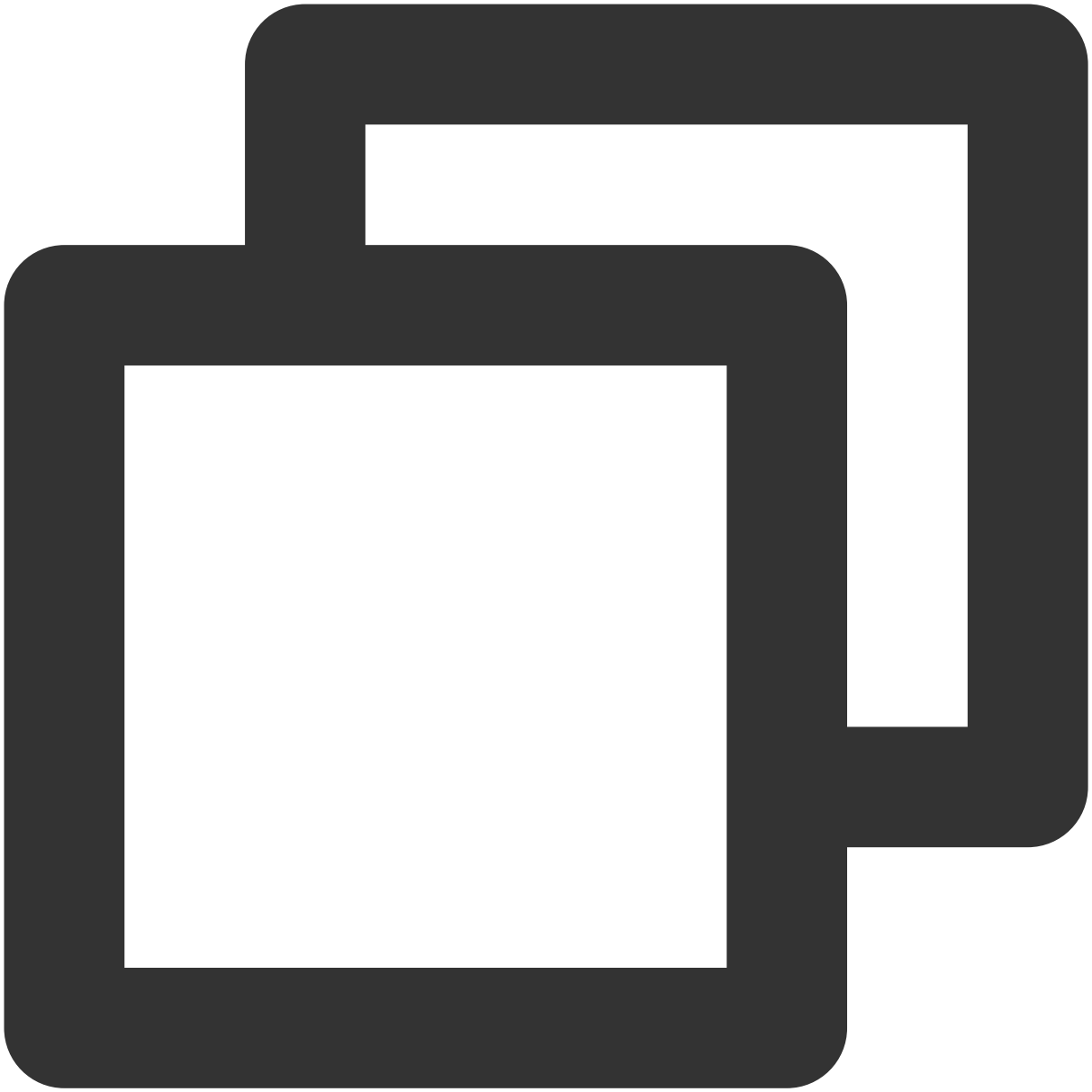
示例：



```
> SELECT cosh(0);
 1.0
```

CONV

函数语法：



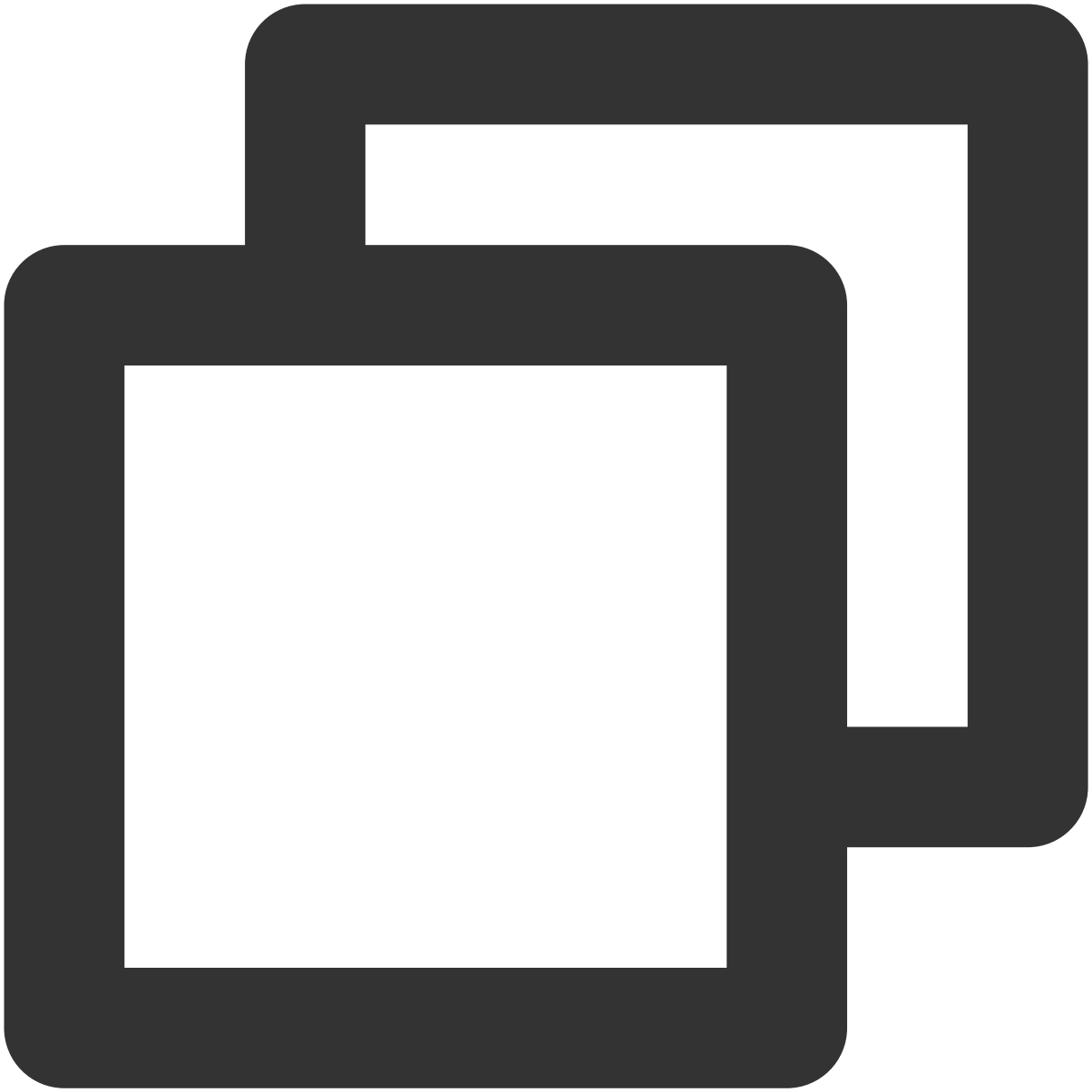
```
CONV(<num> bigint|string, <from_base> integer, <to_base> integer)
```

支持引擎：SparkSQL、Presto

使用说明：将num从from_base转换为to_base

返回类型：string

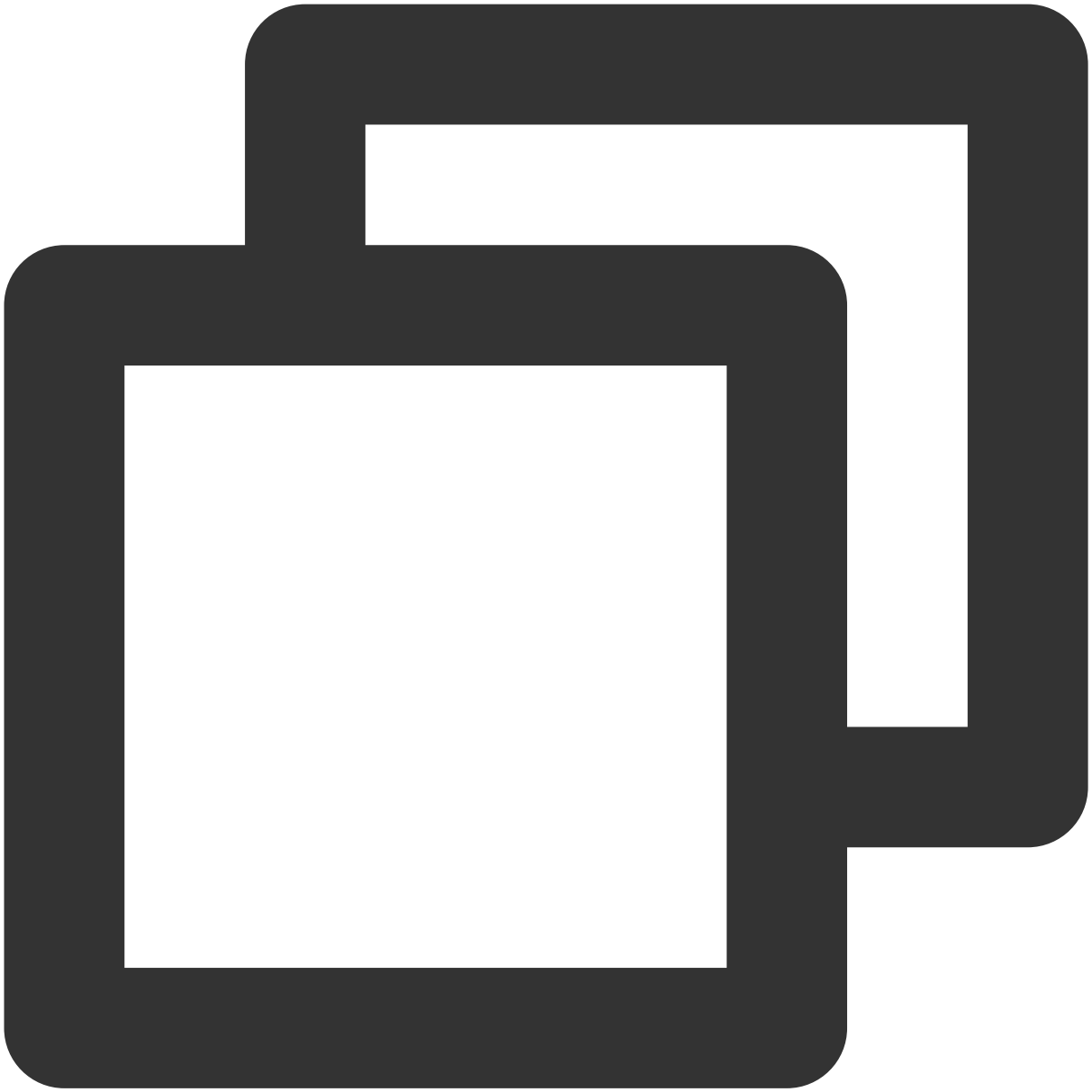
示例：



```
> SELECT conv('100', 2, 10);
 4
> SELECT conv(-10, 16, -10);
 -16
```

DEGREES

函数语法：



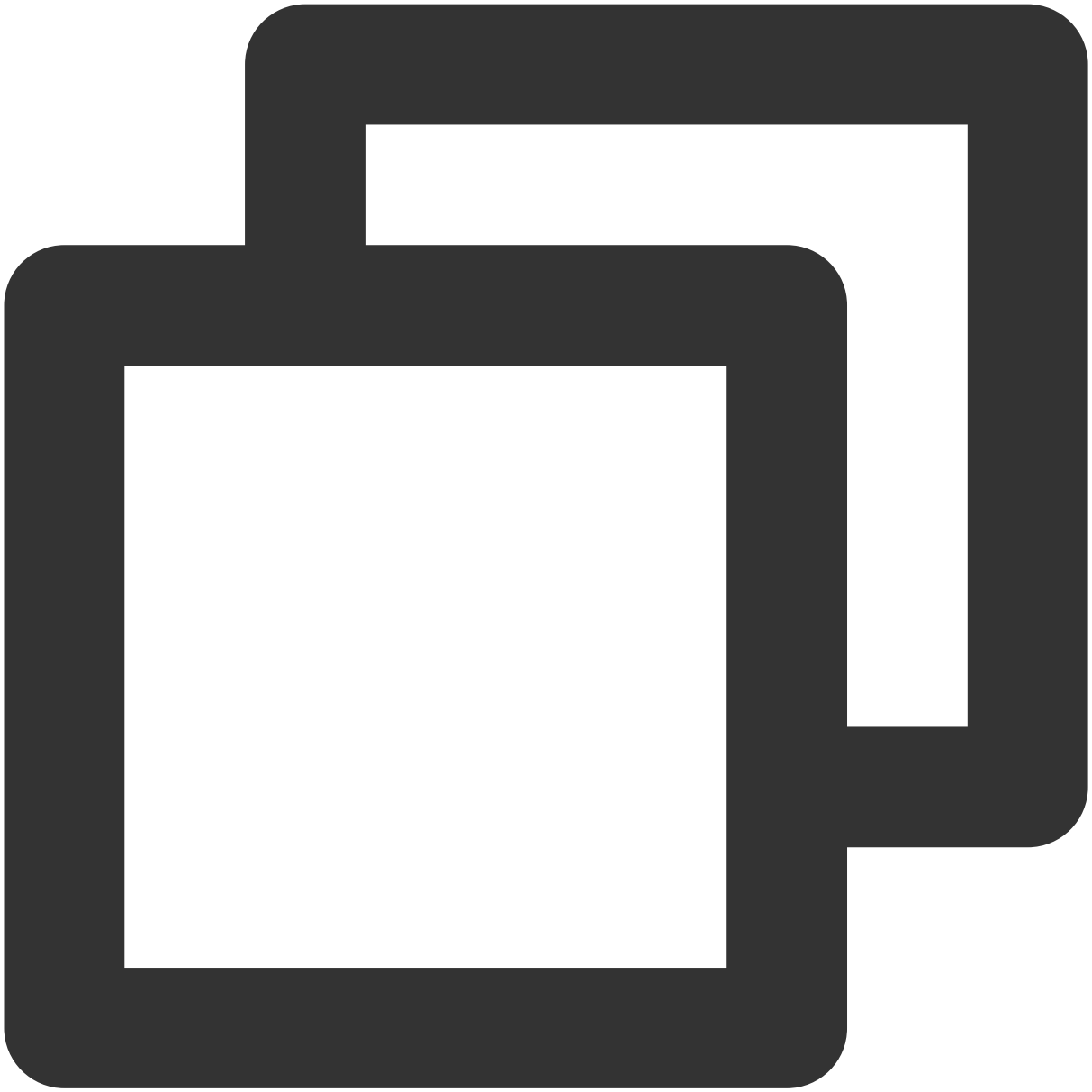
```
DEGREES(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：弧度转化为角度

返回类型：double

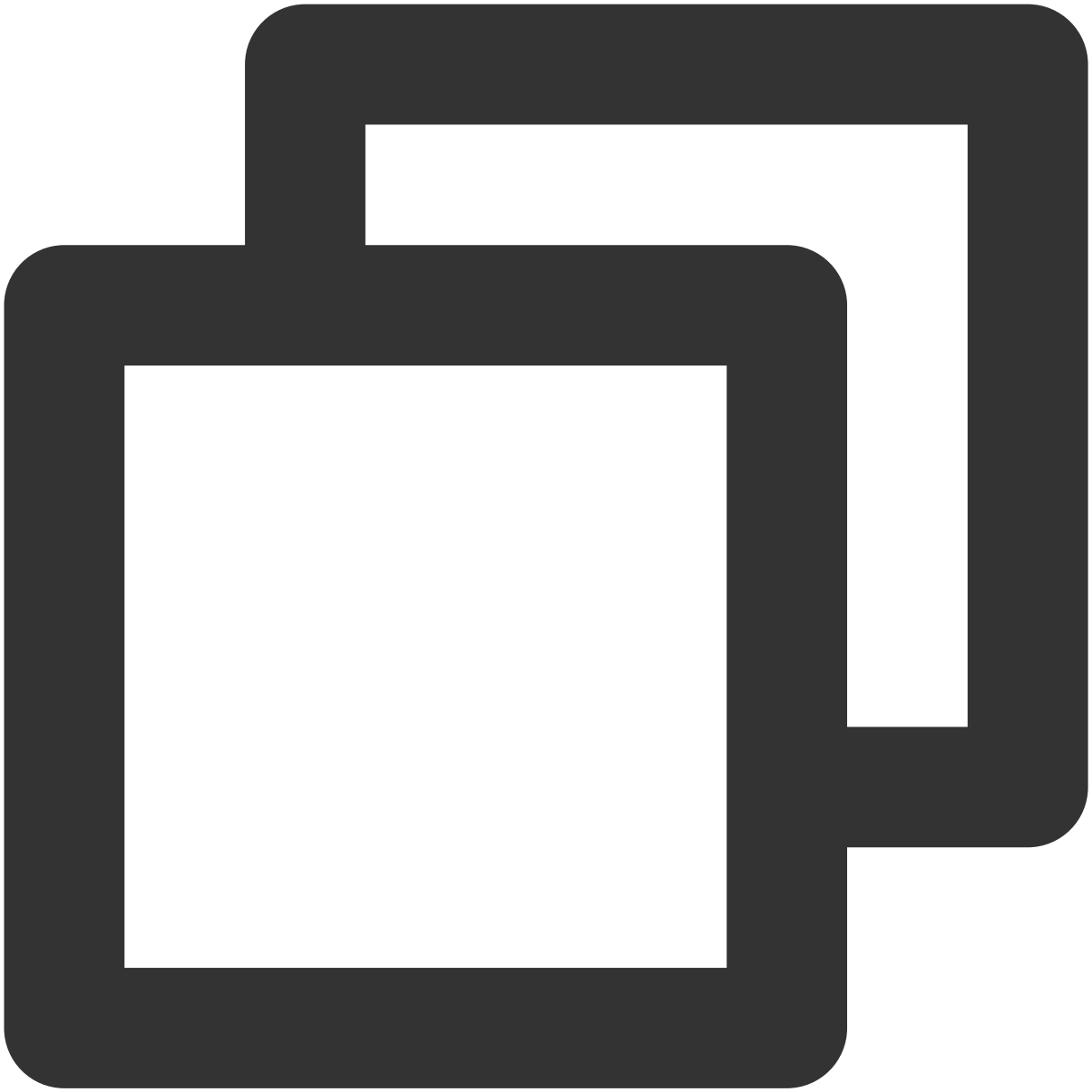
示例：



```
> SELECT degrees(3.141592653589793);
 180.0
```

E

函数语法：



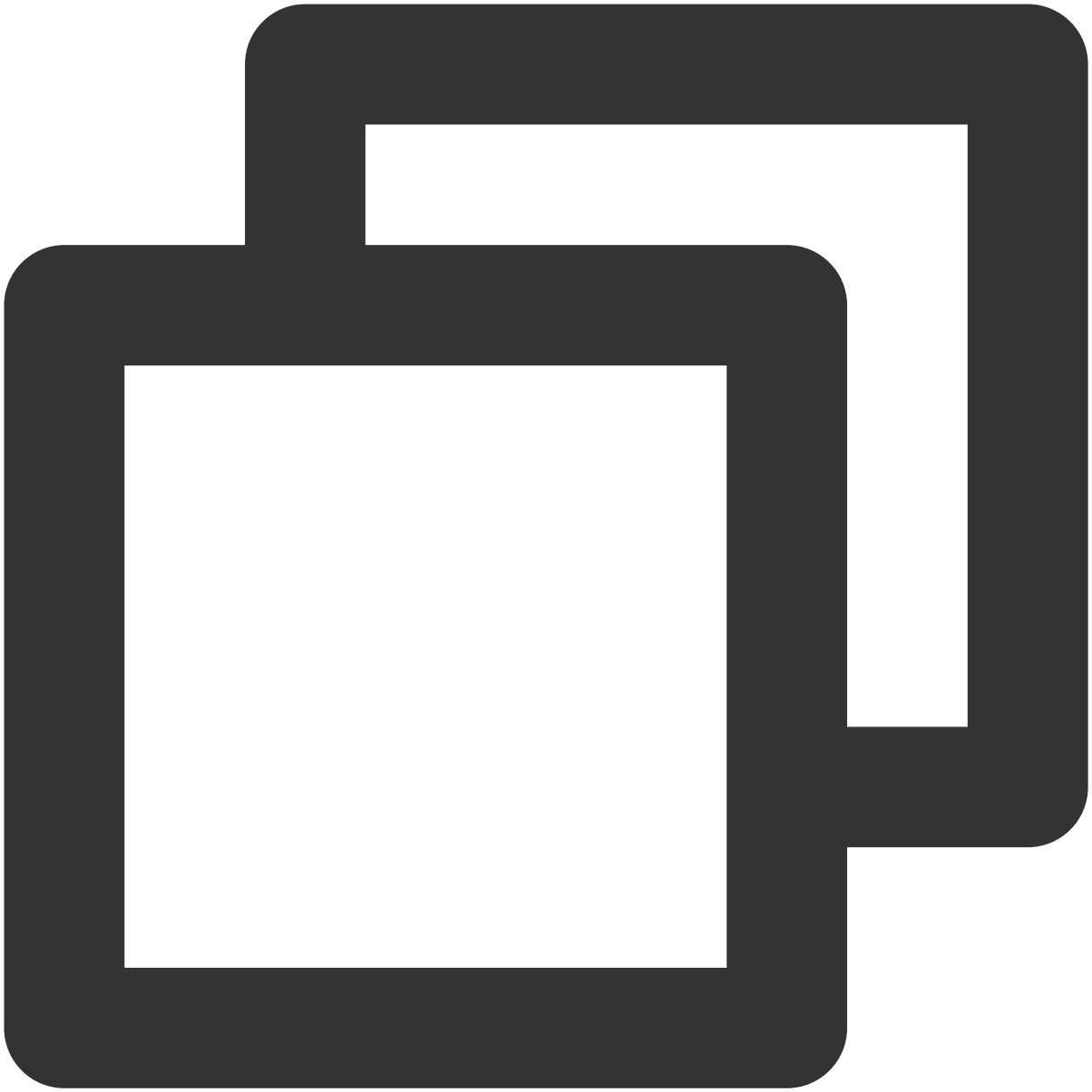
```
E()
```

支持引擎：SparkSQL、Presto

使用说明：返回欧拉常数

返回类型：double

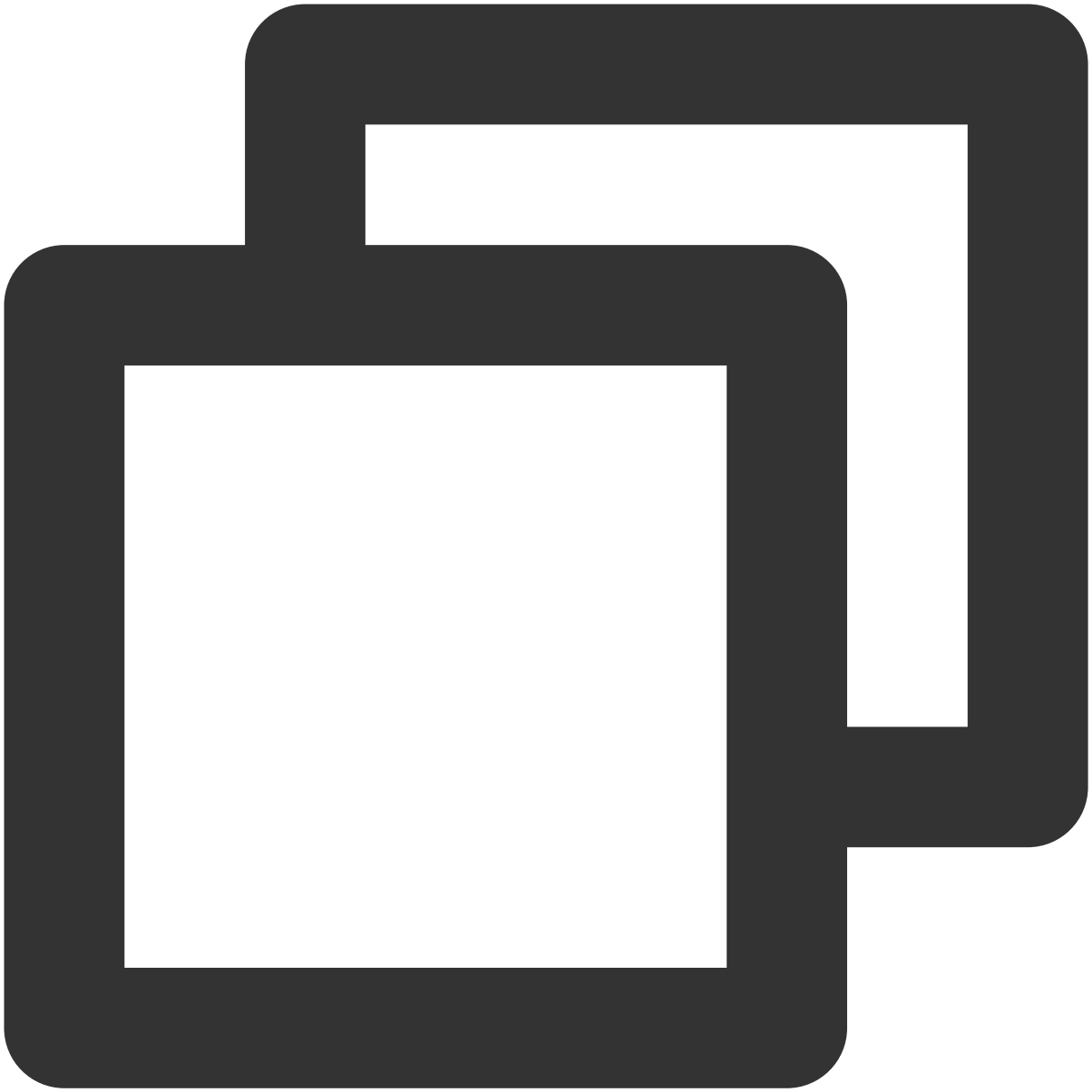
示例：



```
> SELECT e();
 2.718281828459045
```

EXP

函数语法：



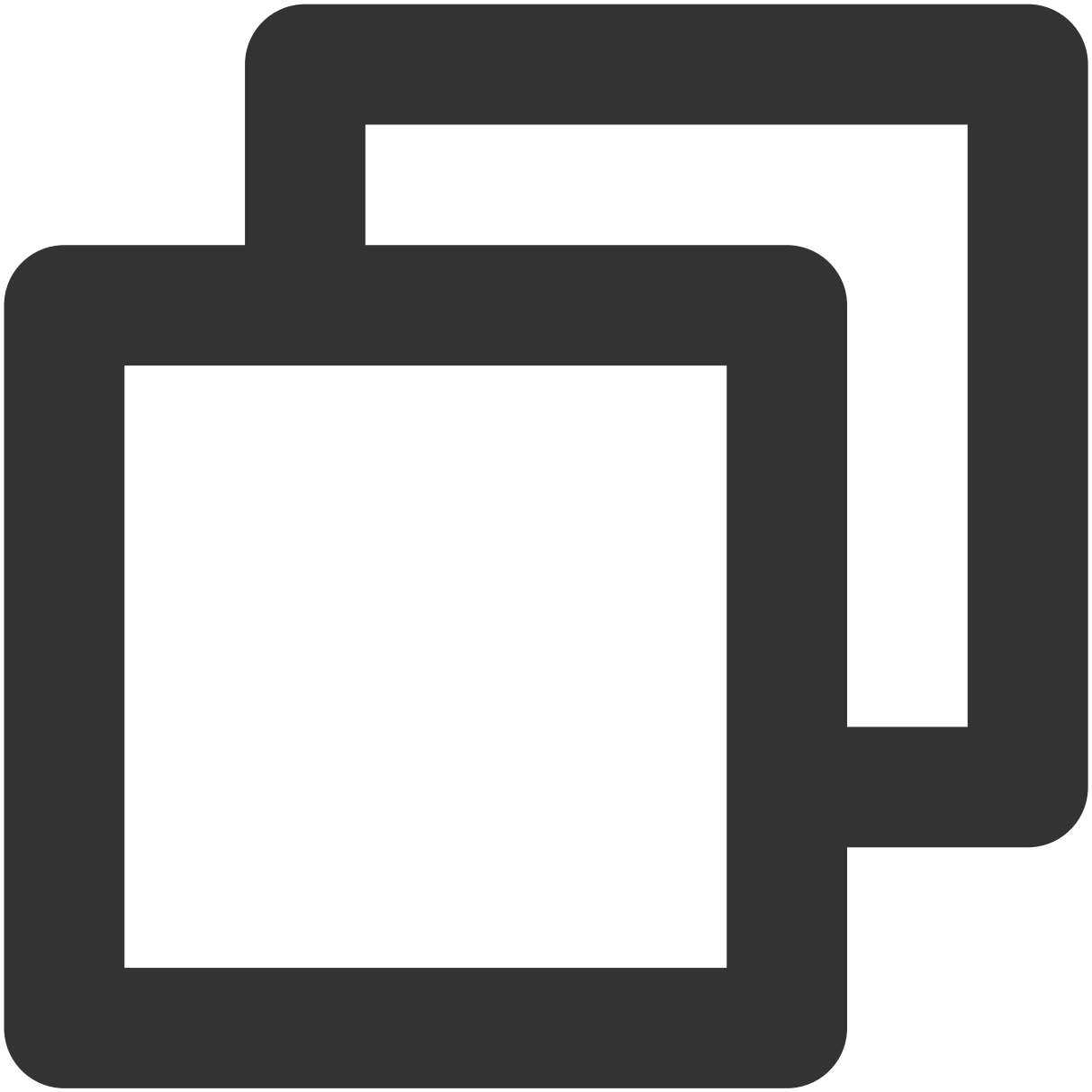
```
EXP(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回欧拉常数e的expr次方

返回类型：double

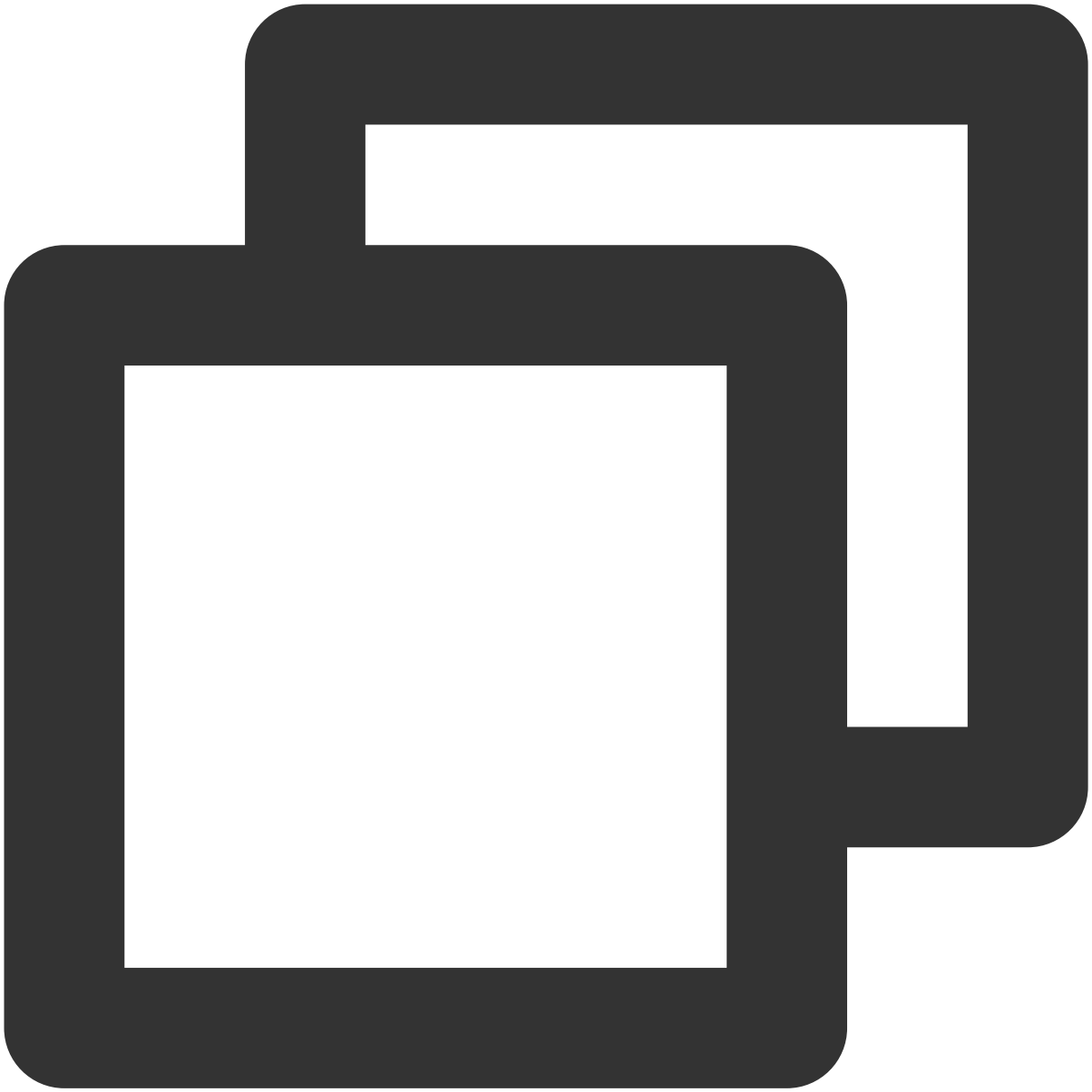
示例：



```
> SELECT exp(0);
 1.0
```

EXPM1

函数语法：



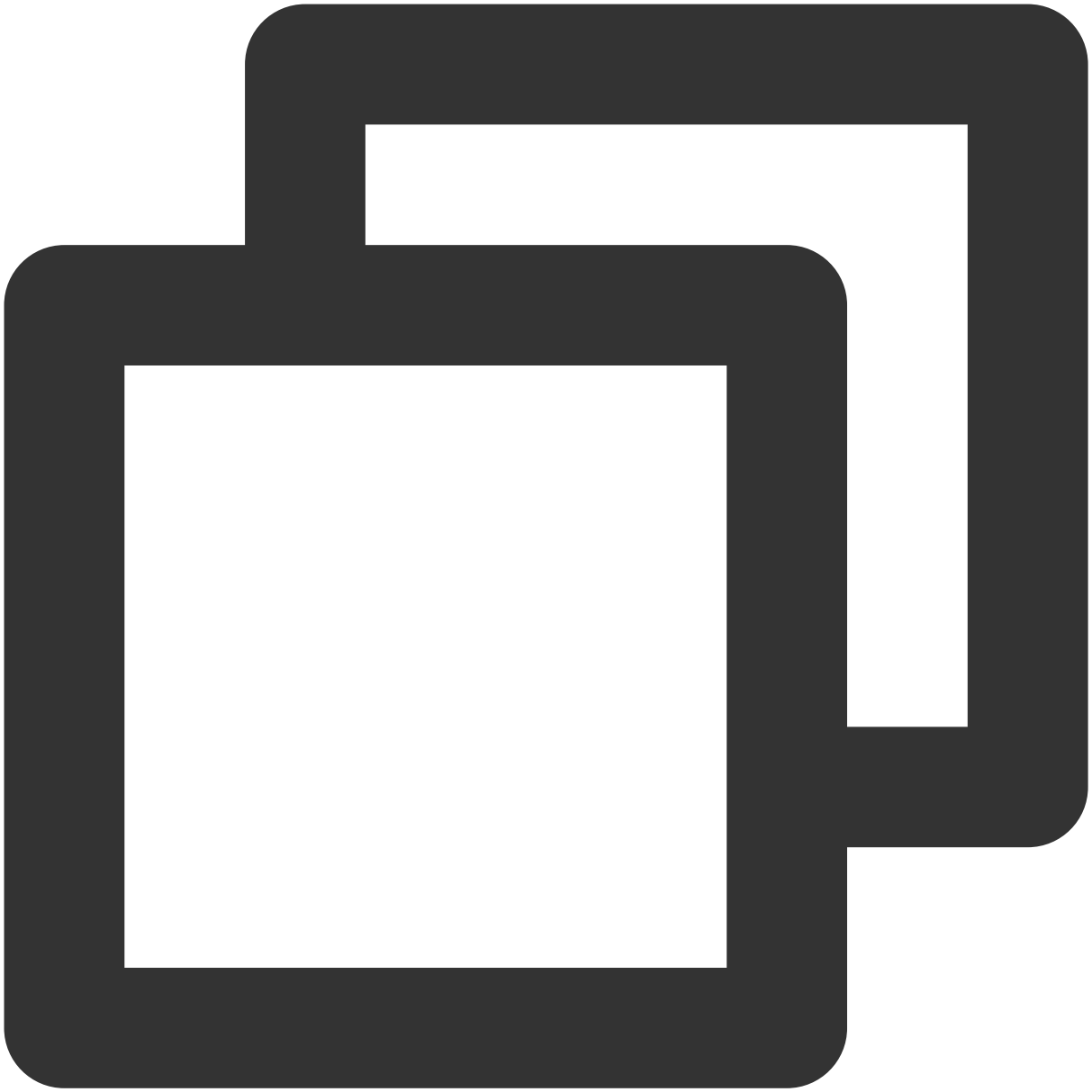
```
EXPM1(<expr> integer|double|decimal)
```

支持引擎：SparkSQL

使用说明：返回EXP(expr)-1

返回类型：double

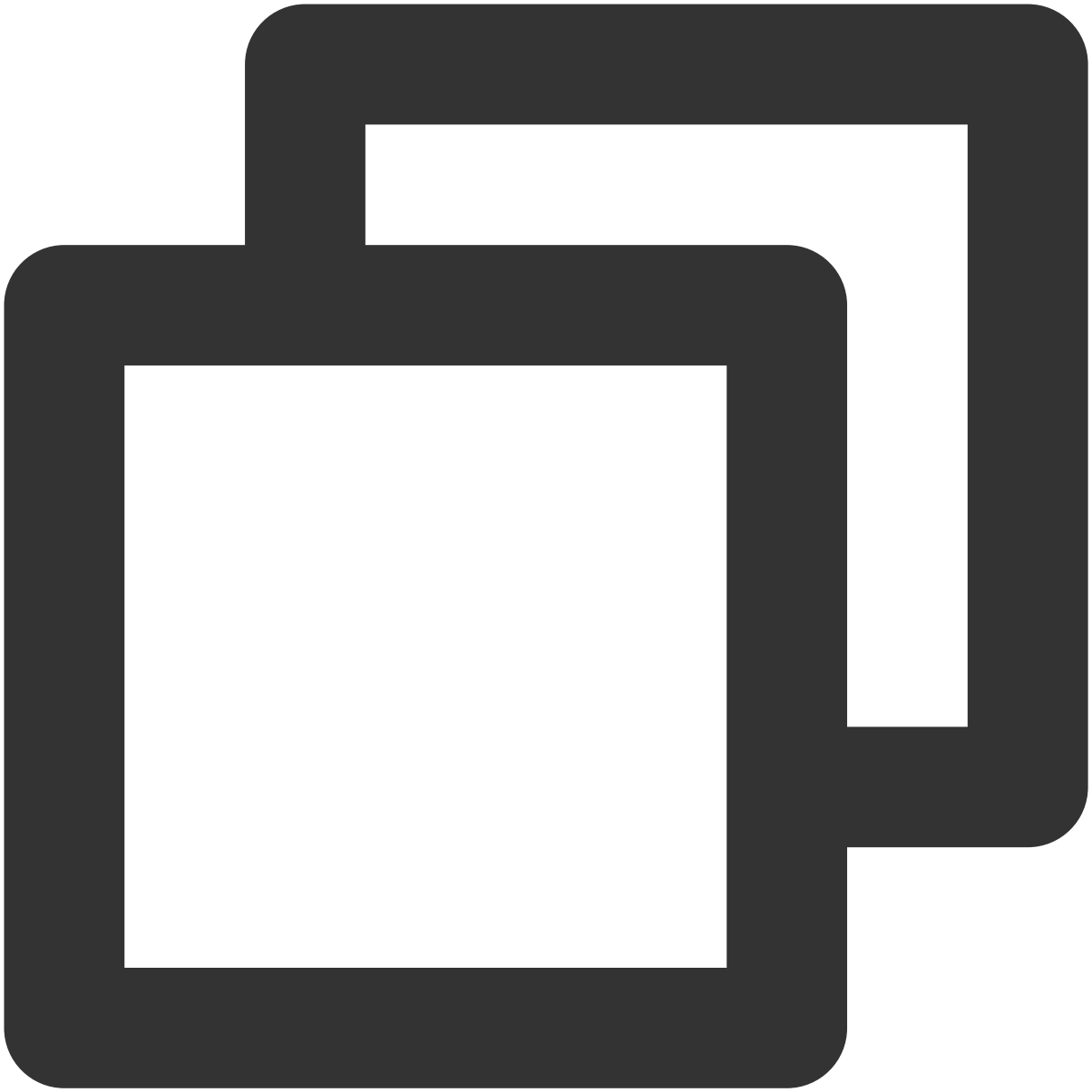
示例：



```
> SELECT expm1(0);
 0.0
```

FLOOR

函数语法：



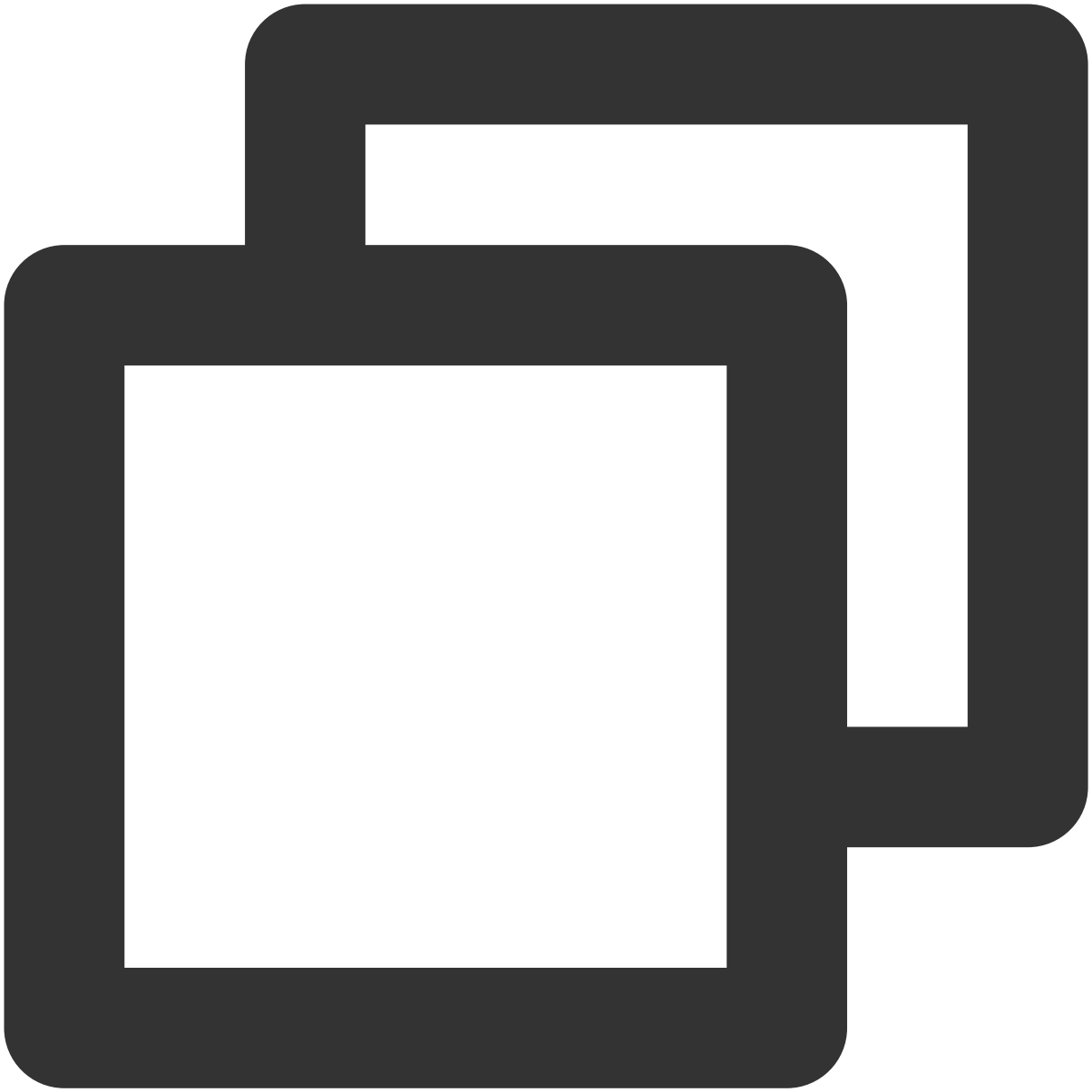
```
FLOOR(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回不大于expr的最大整数

返回类型：integer

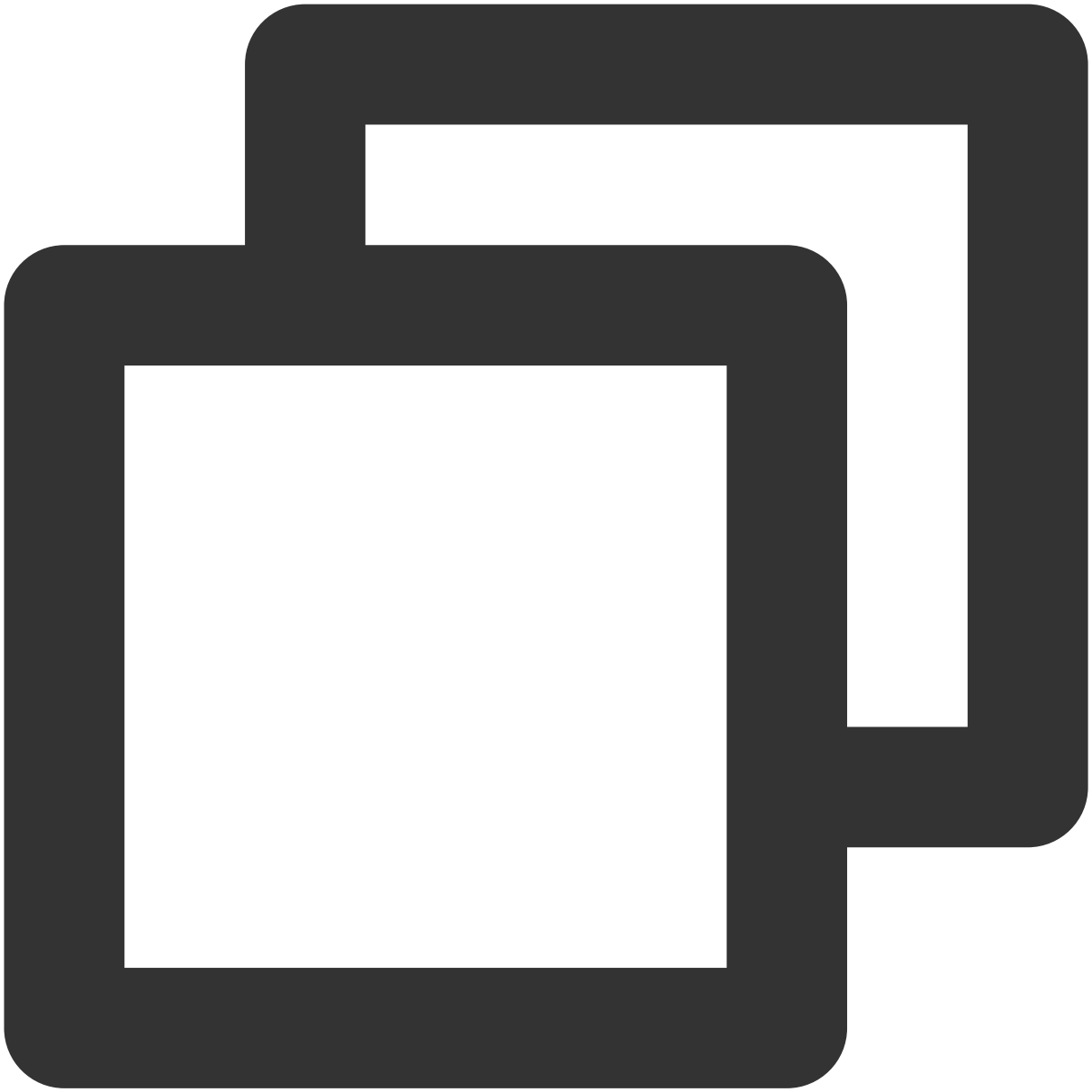
示例：



```
> SELECT floor(-0.1);
 -1
> SELECT floor(5);
 5
```

FACTORIAL

函数语法：



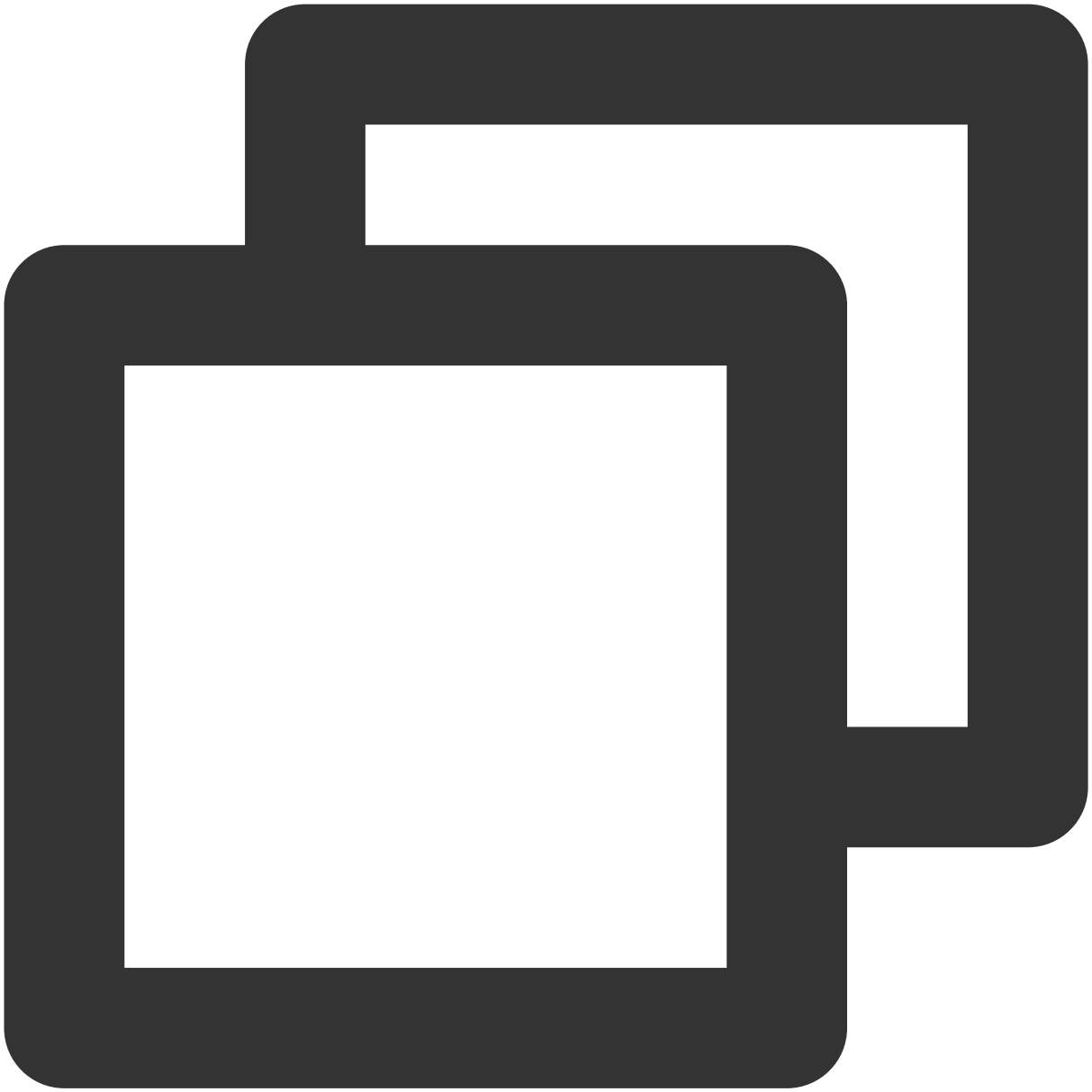
```
FACTORIAL(<expr> integer)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的阶乘。 expr为[0..20]。 否则为NULL。

返回类型：bigint

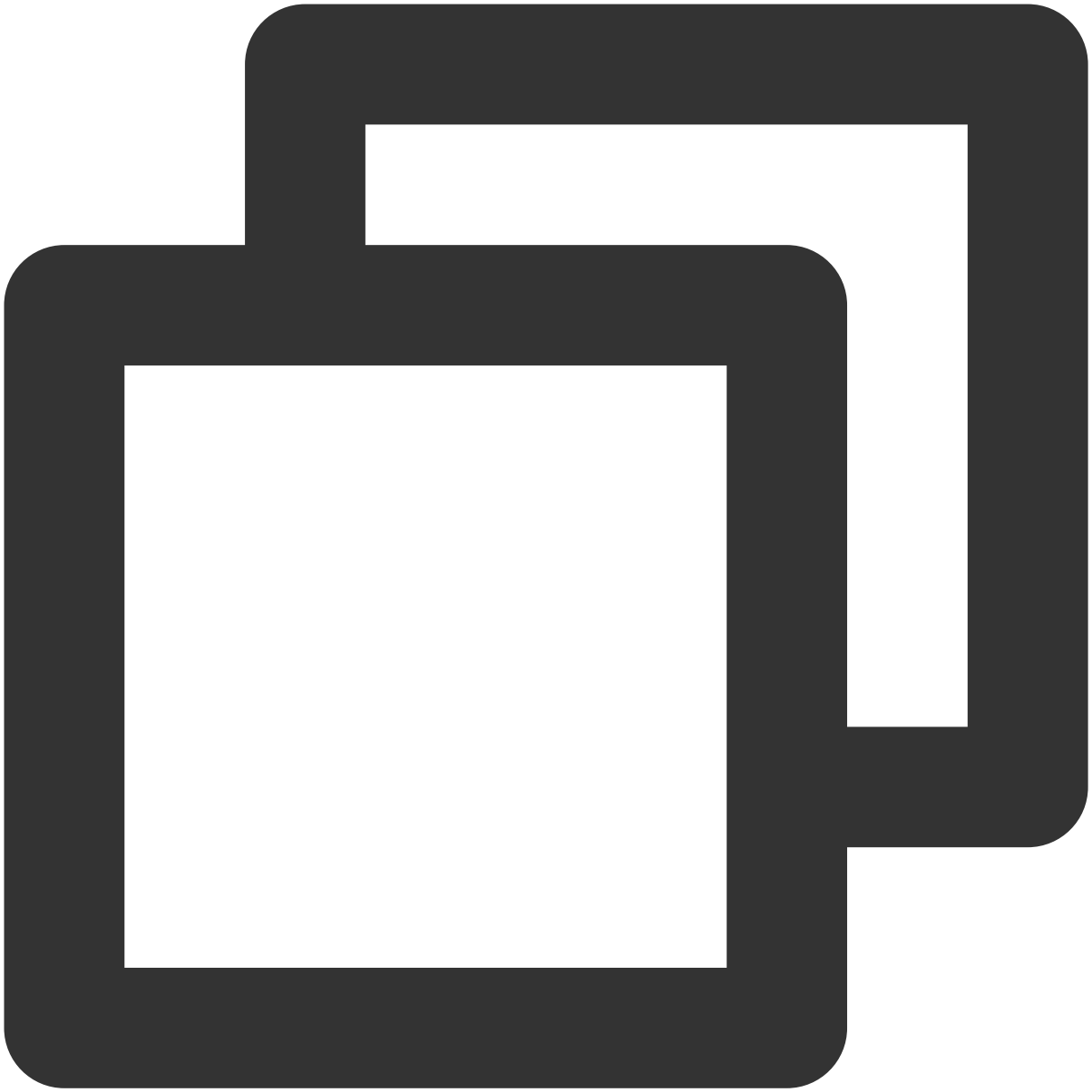
示例：



```
> SELECT factorial(5);
 120
```

HEX

函数语法：



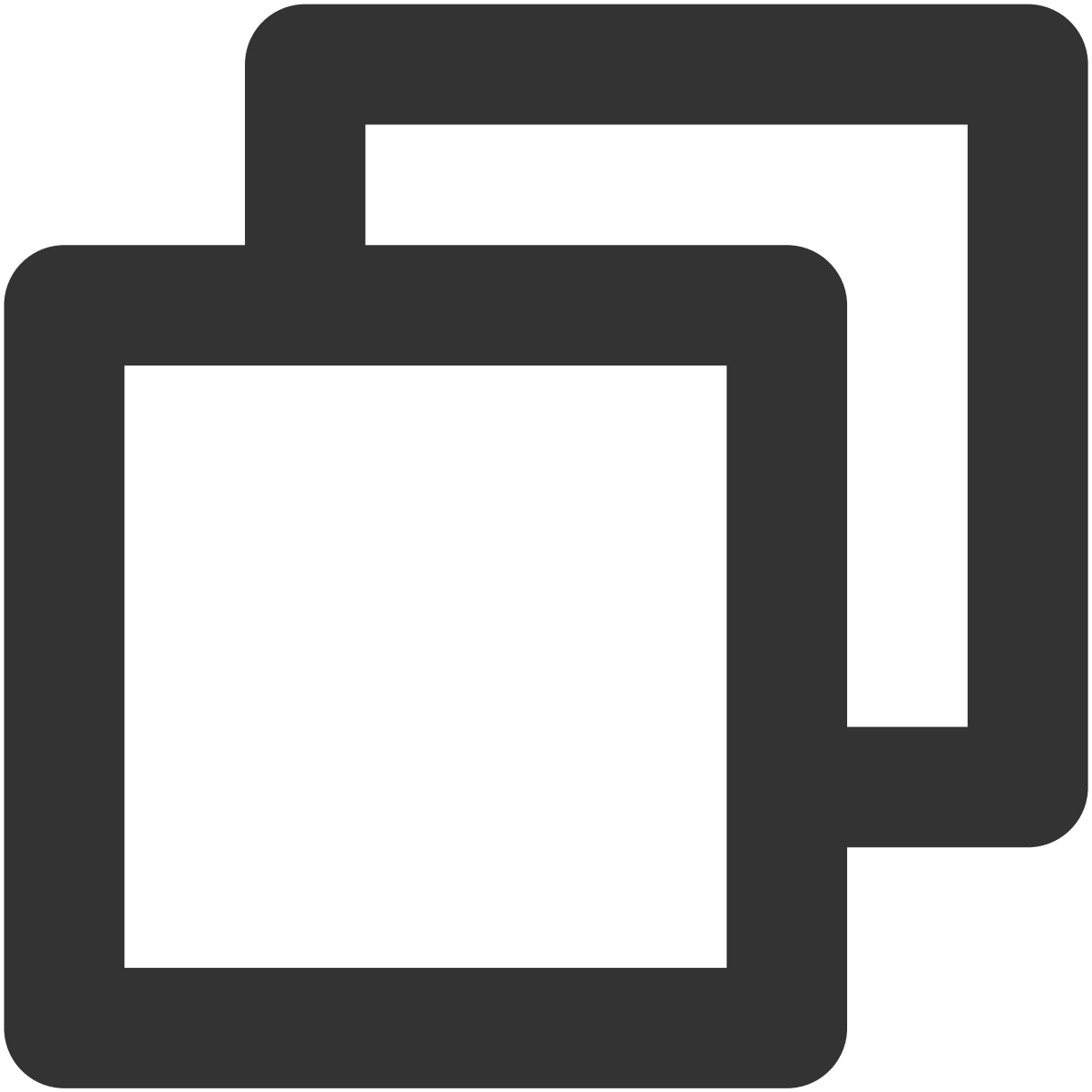
```
HEX(<expr> bigint|string)
```

支持引擎：SparkSQL、Presto

使用说明：以十六进制返回expr

返回类型：string

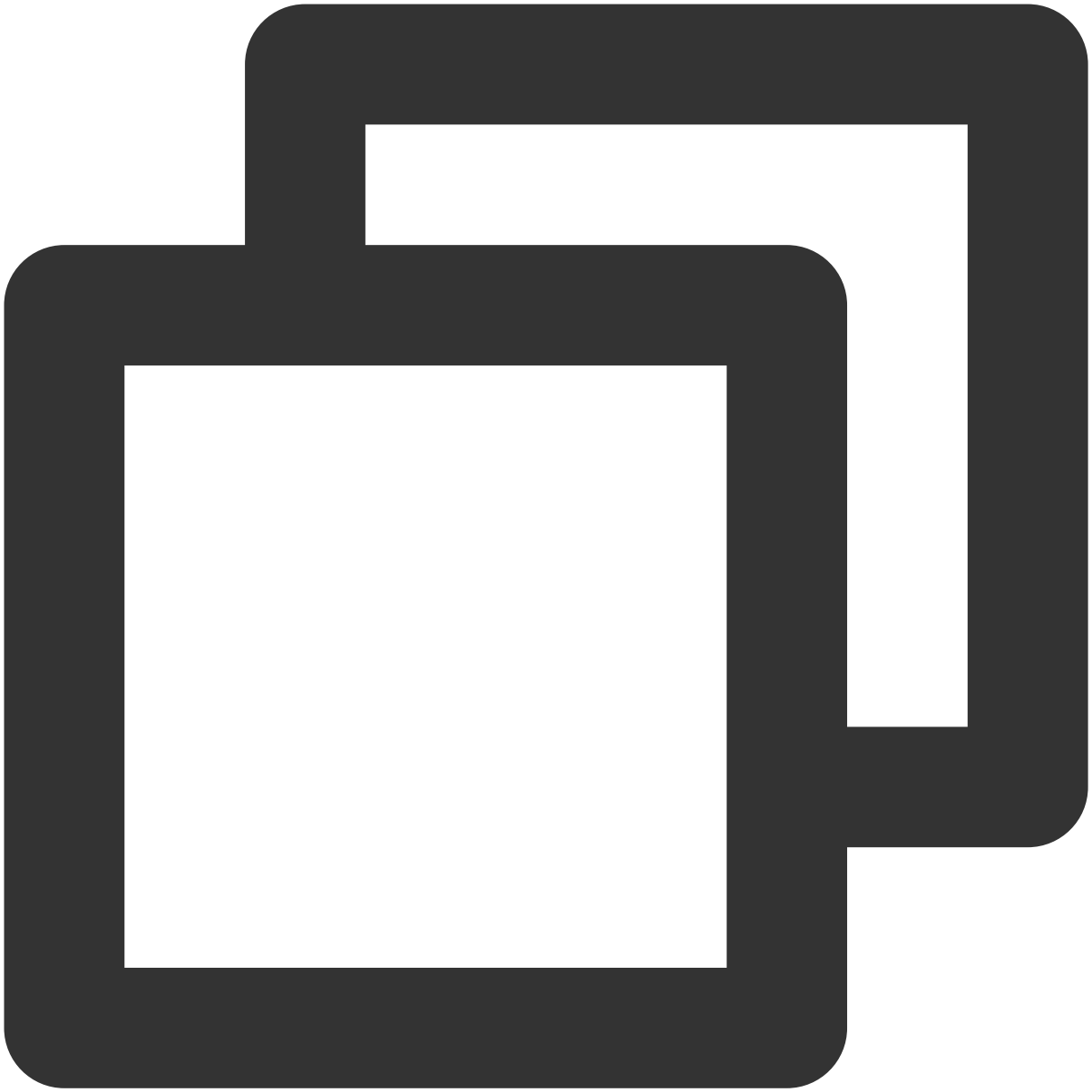
示例：



```
> SELECT hex(17);
 11
> SELECT hex('Spark SQL');
 537061726B2053514C
```

HYPOT

函数语法：



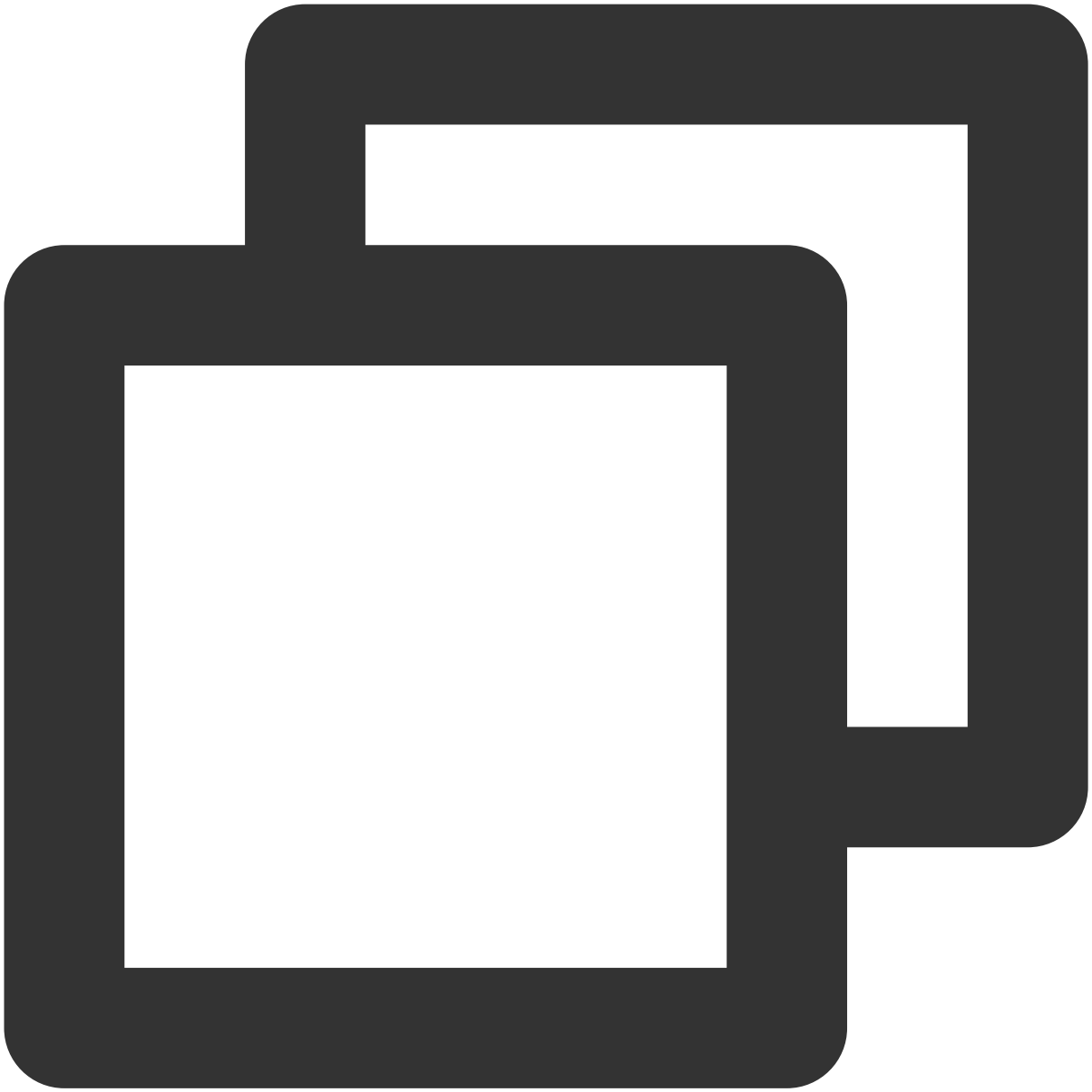
```
HYPOT(<expr1> integer|double|decimal, <expr2> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回sqrt(pow(expr1, 2) + pow(expr2, 2))

返回类型：double

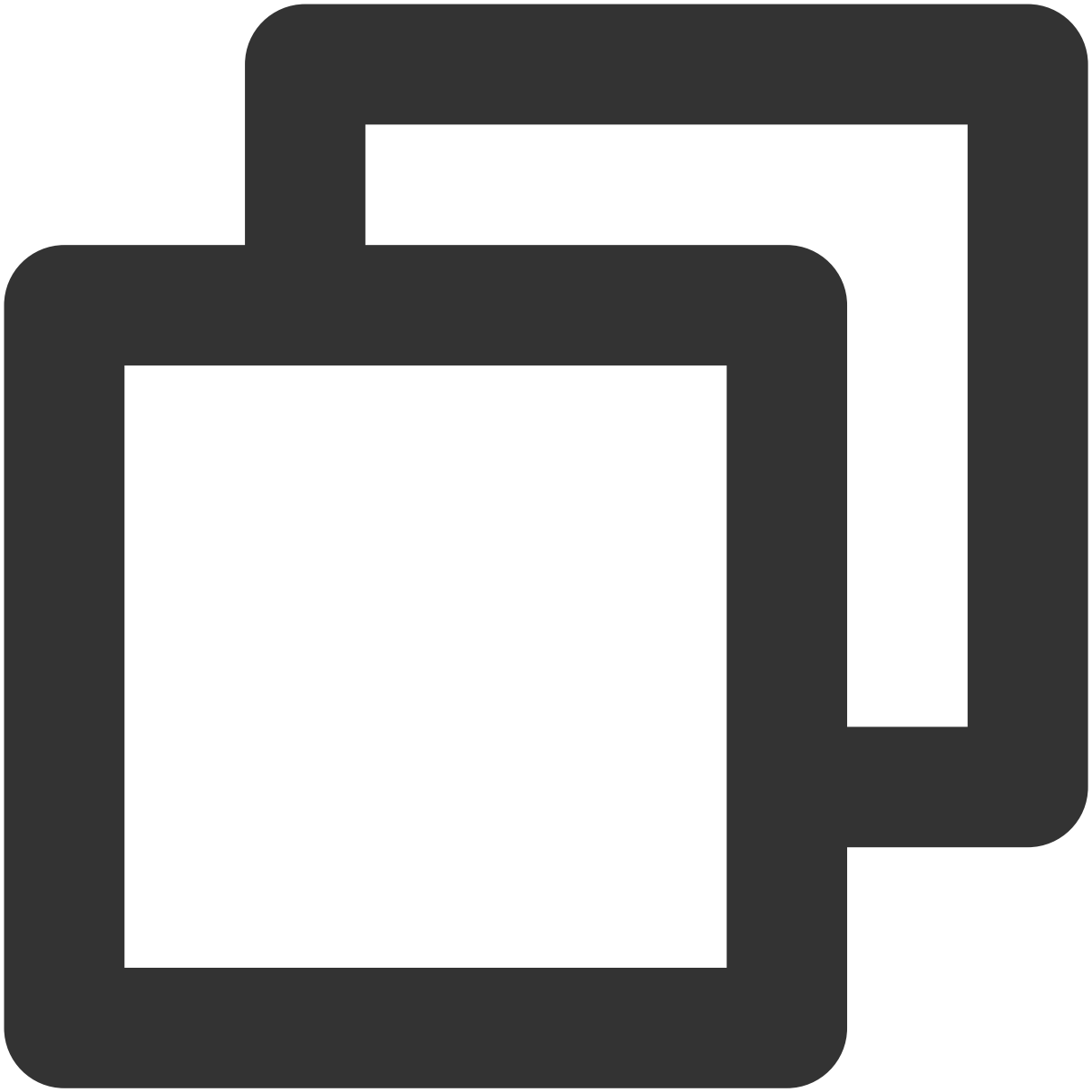
示例：



```
> SELECT hypot(3, 4);
 5.0
```

LOG

函数语法：



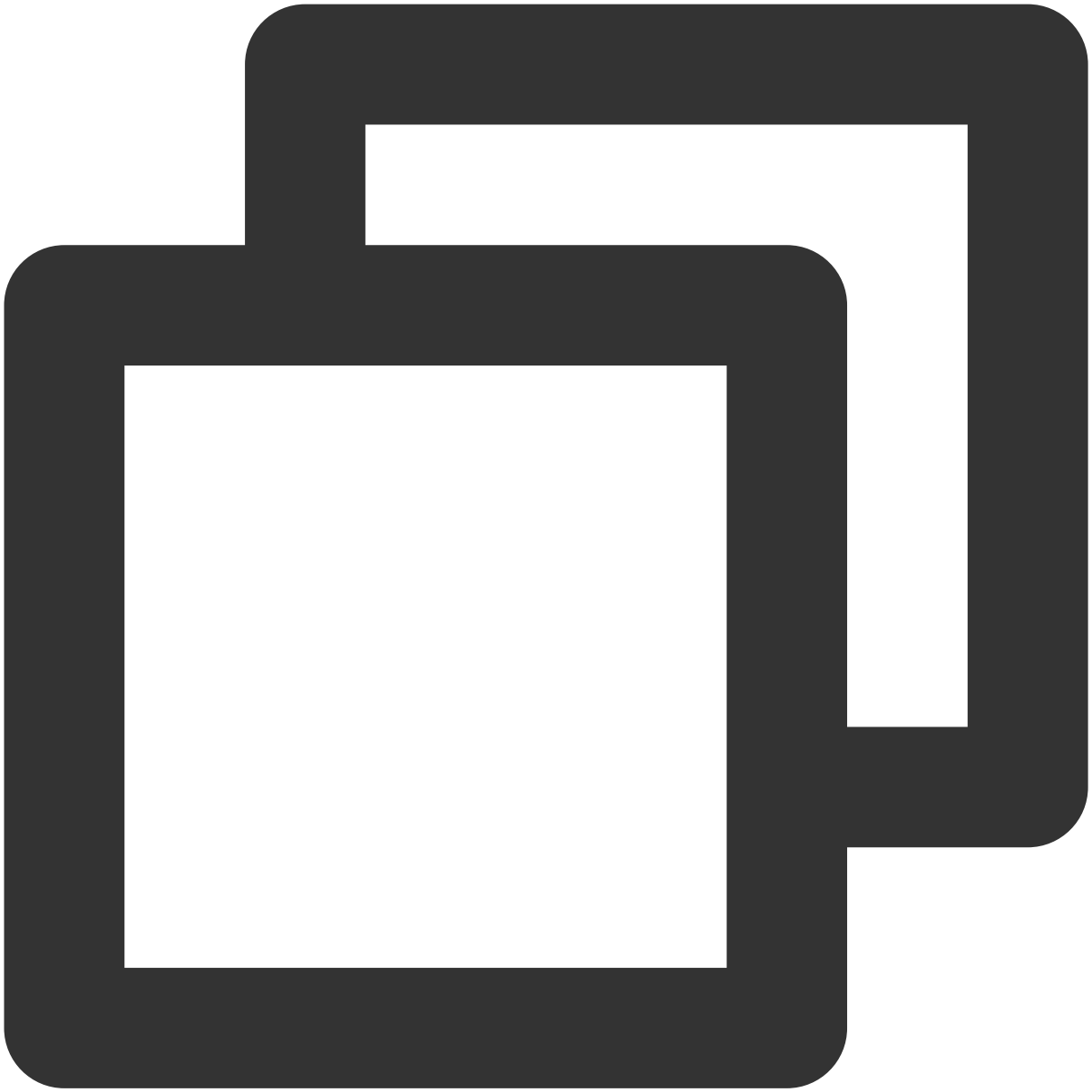
```
LOG(<base> integer|double|decimal, <expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr以base为底数的对数。

返回类型：double

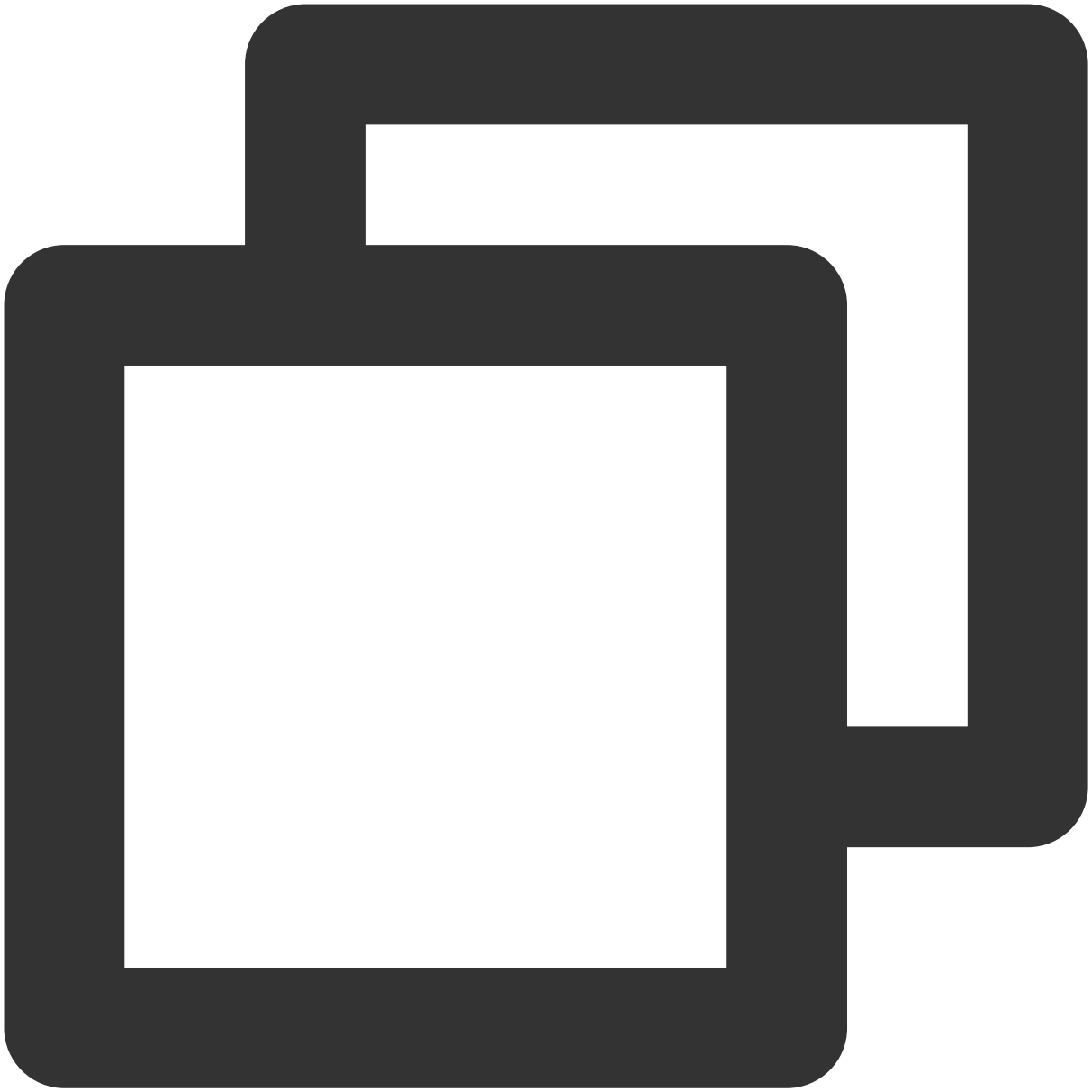
示例：



```
> SELECT log(10, 100);
 2.0
```

LOG10

函数语法：



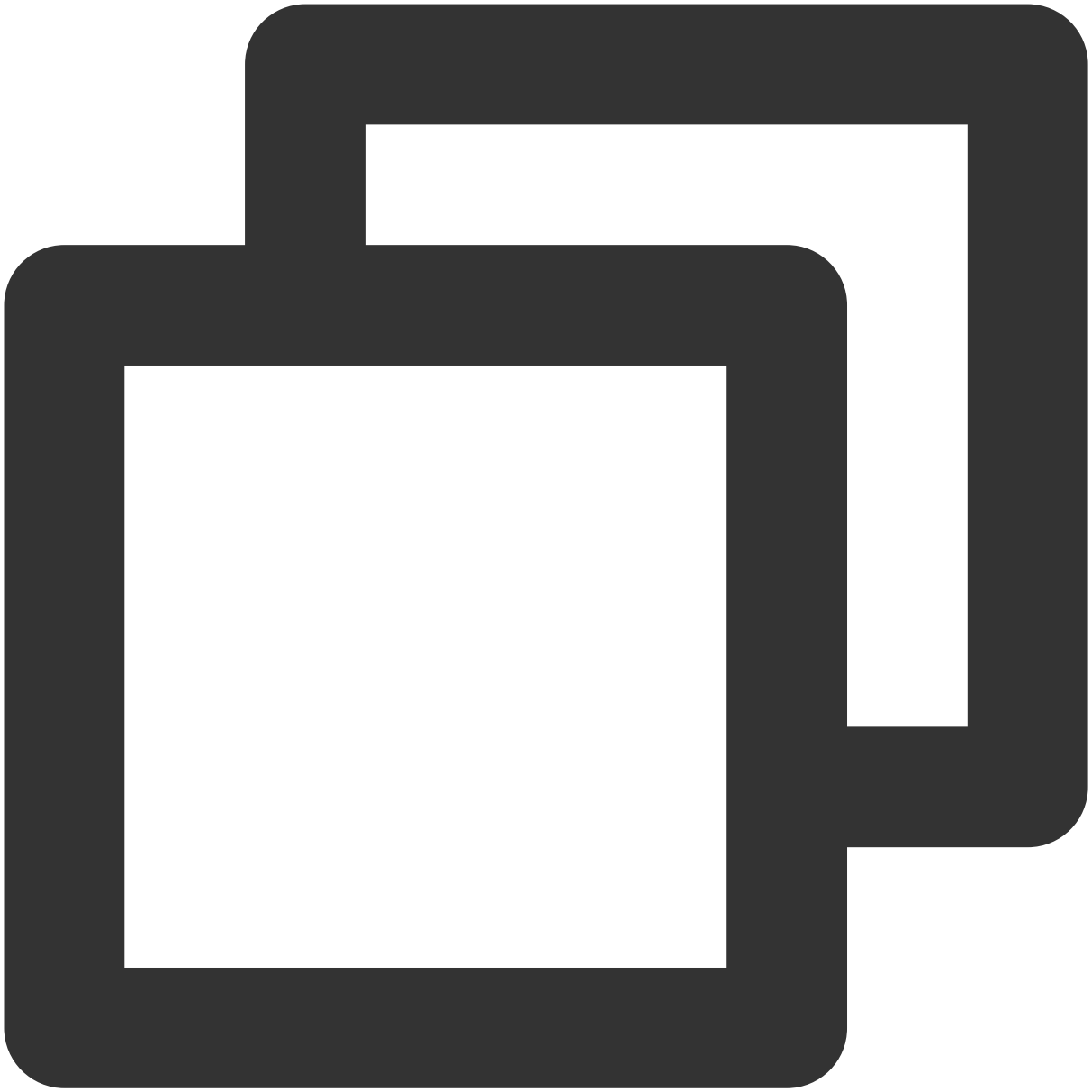
```
LOG10(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr以10为底数的对数。

返回类型：double

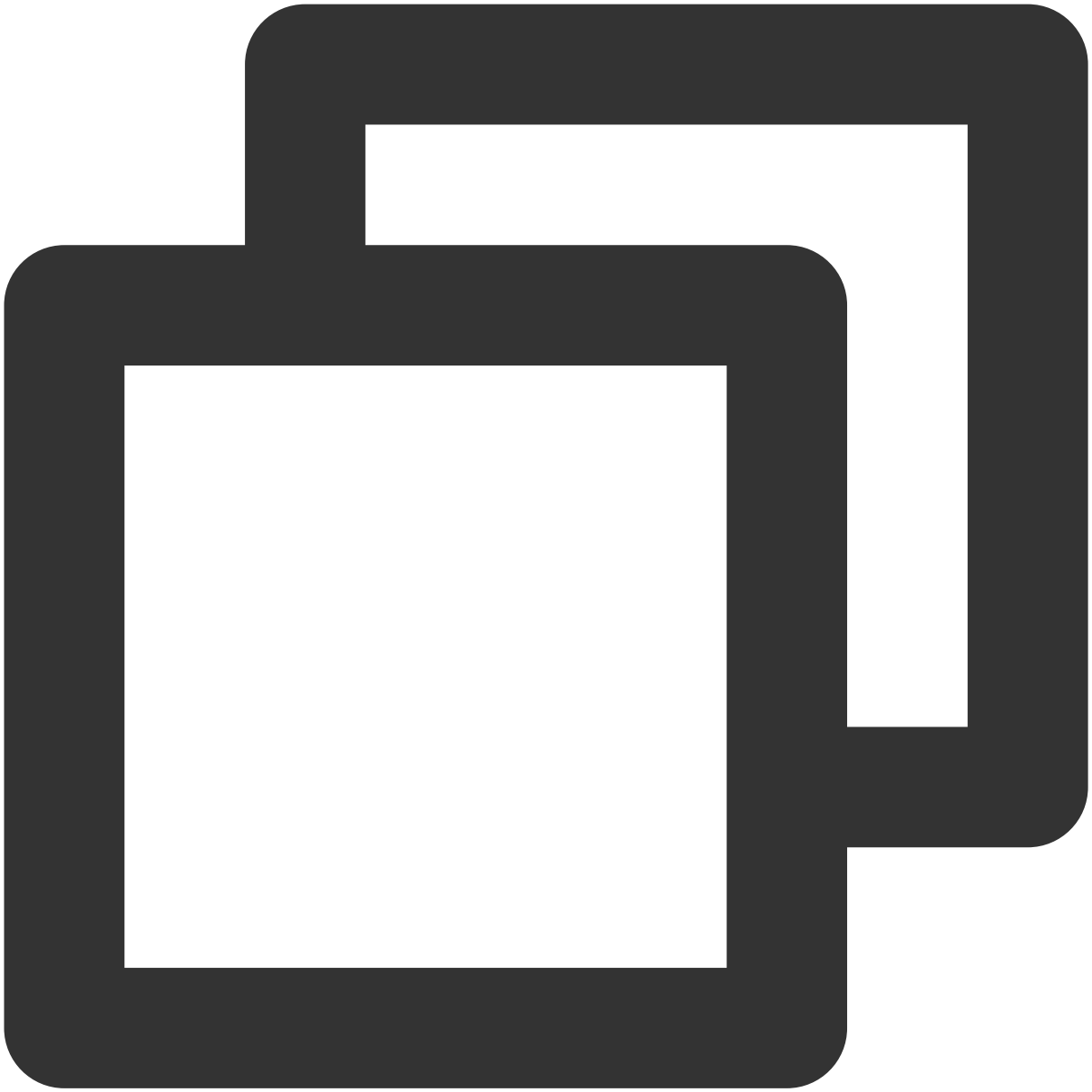
示例：



```
> SELECT log10(10);
 1.0
```

LOG1P

函数语法：



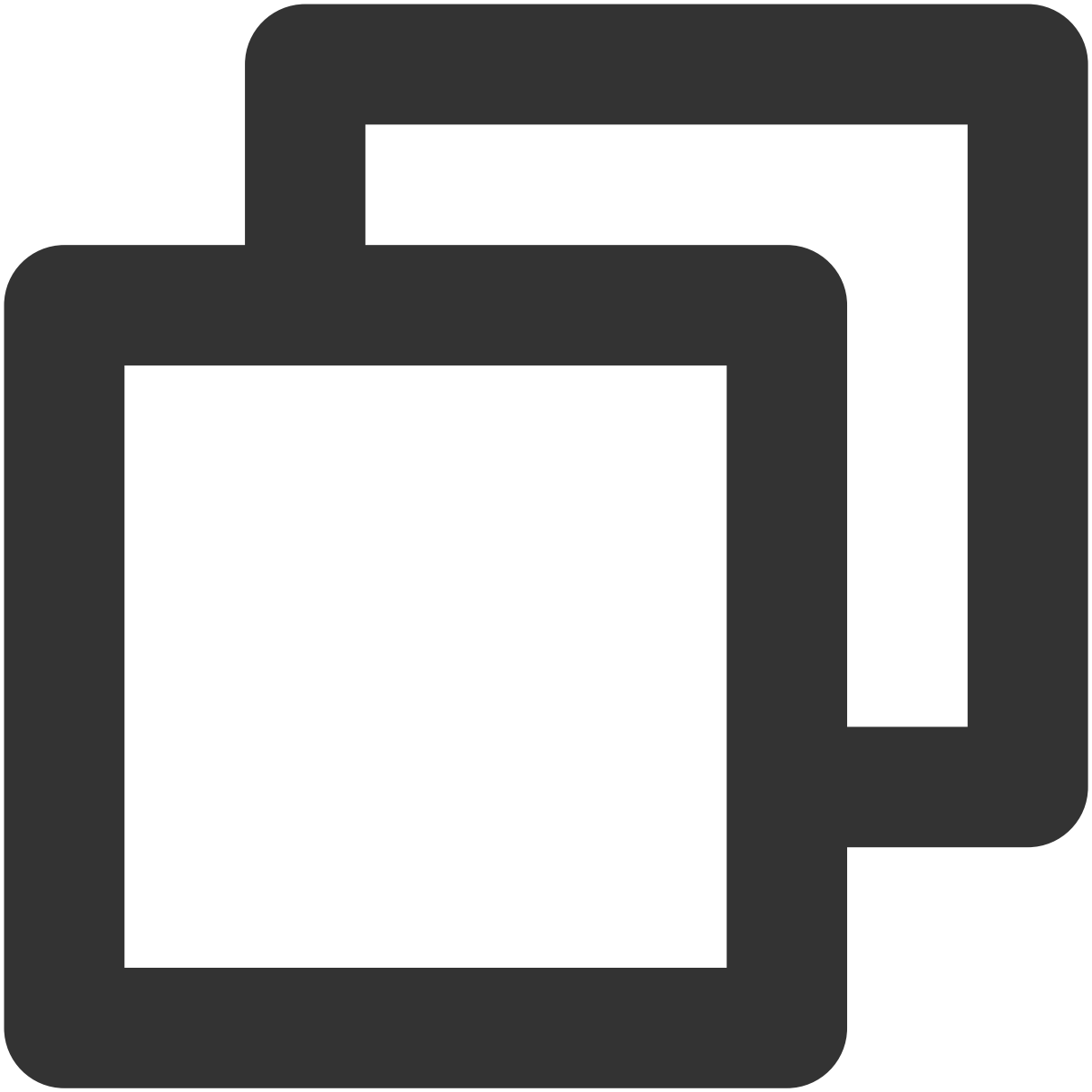
```
LOG1P(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回log(1 + expr)

返回类型：double

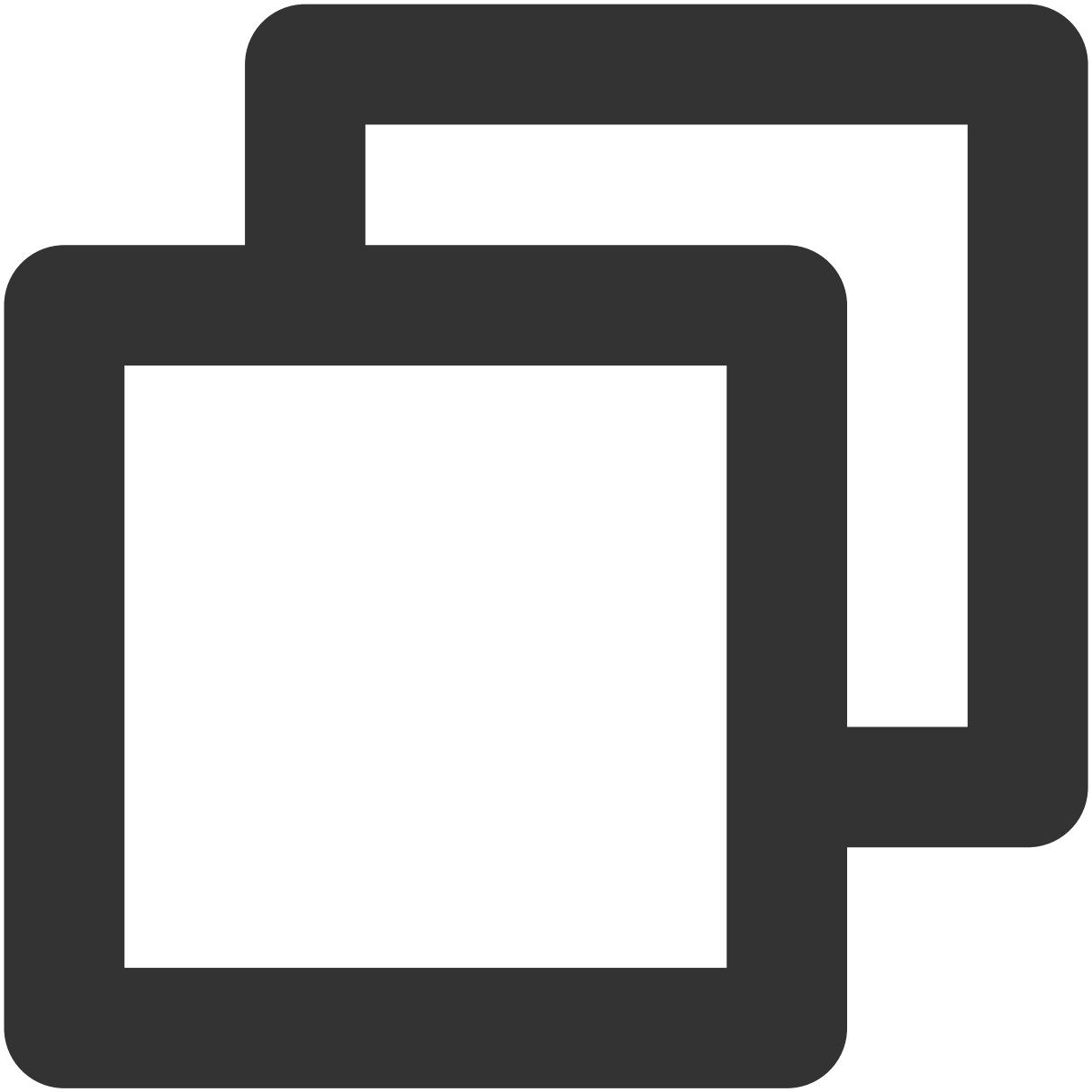
示例：



```
> SELECT log1p(0);
 0.0
```

LOG2

函数语法：



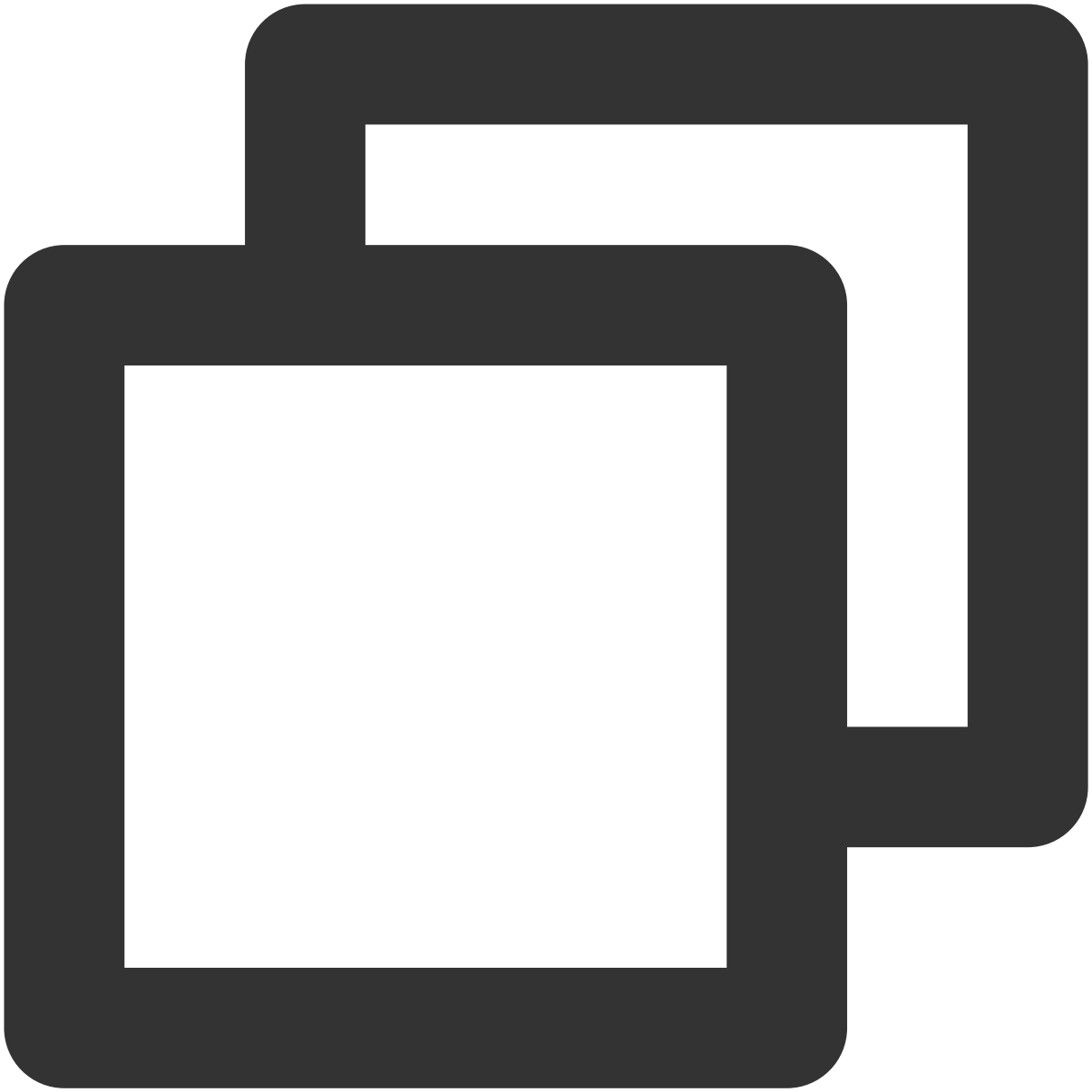
```
LOG2(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回以2为底的expr的对数。

返回类型：double

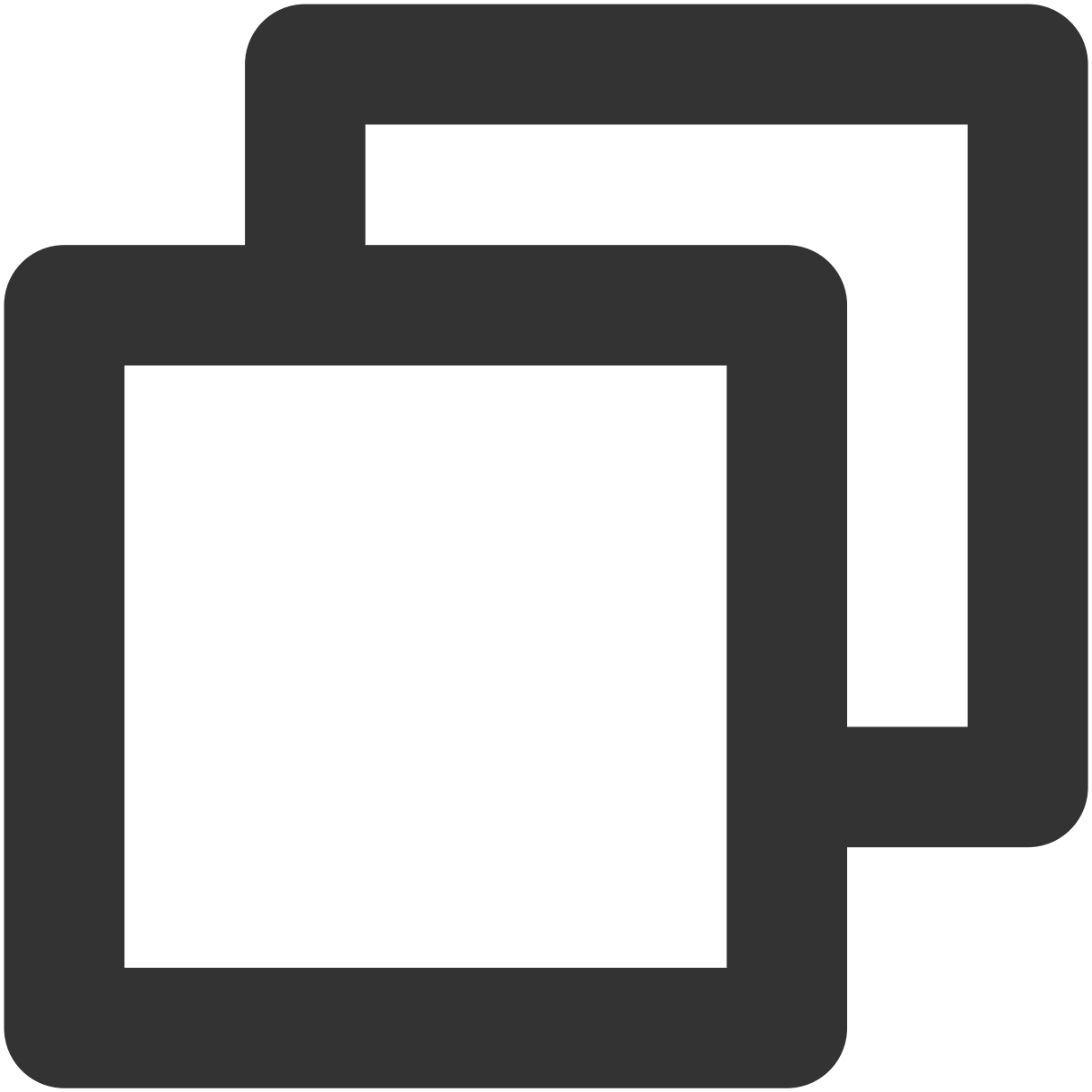
示例：



```
> SELECT log2(2);
 1.0
```

LN

函数语法：



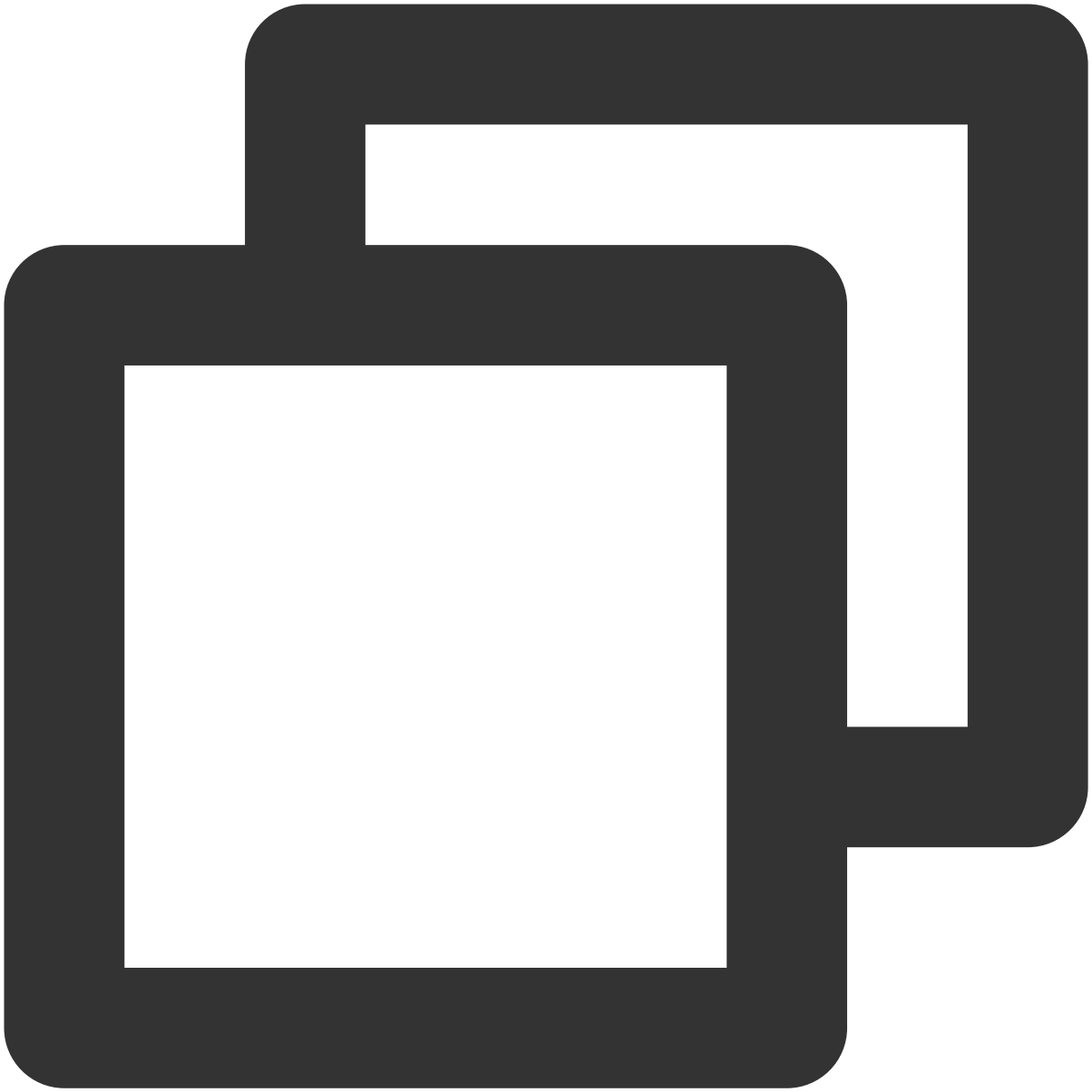
```
LN(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的自然对数（以e为底）。

返回类型：double

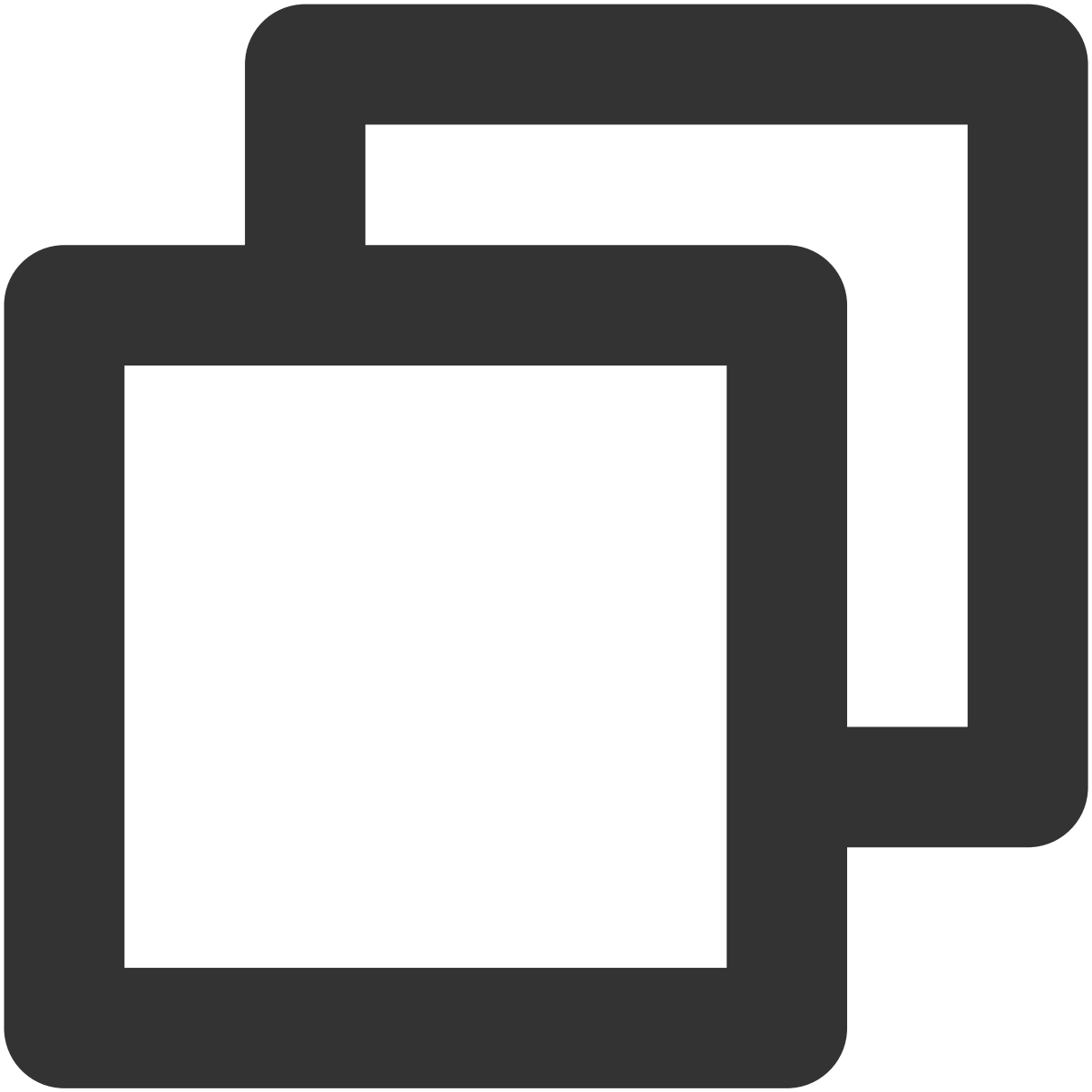
示例：



```
> SELECT ln(1);
 0.0
```

MOD

函数语法：



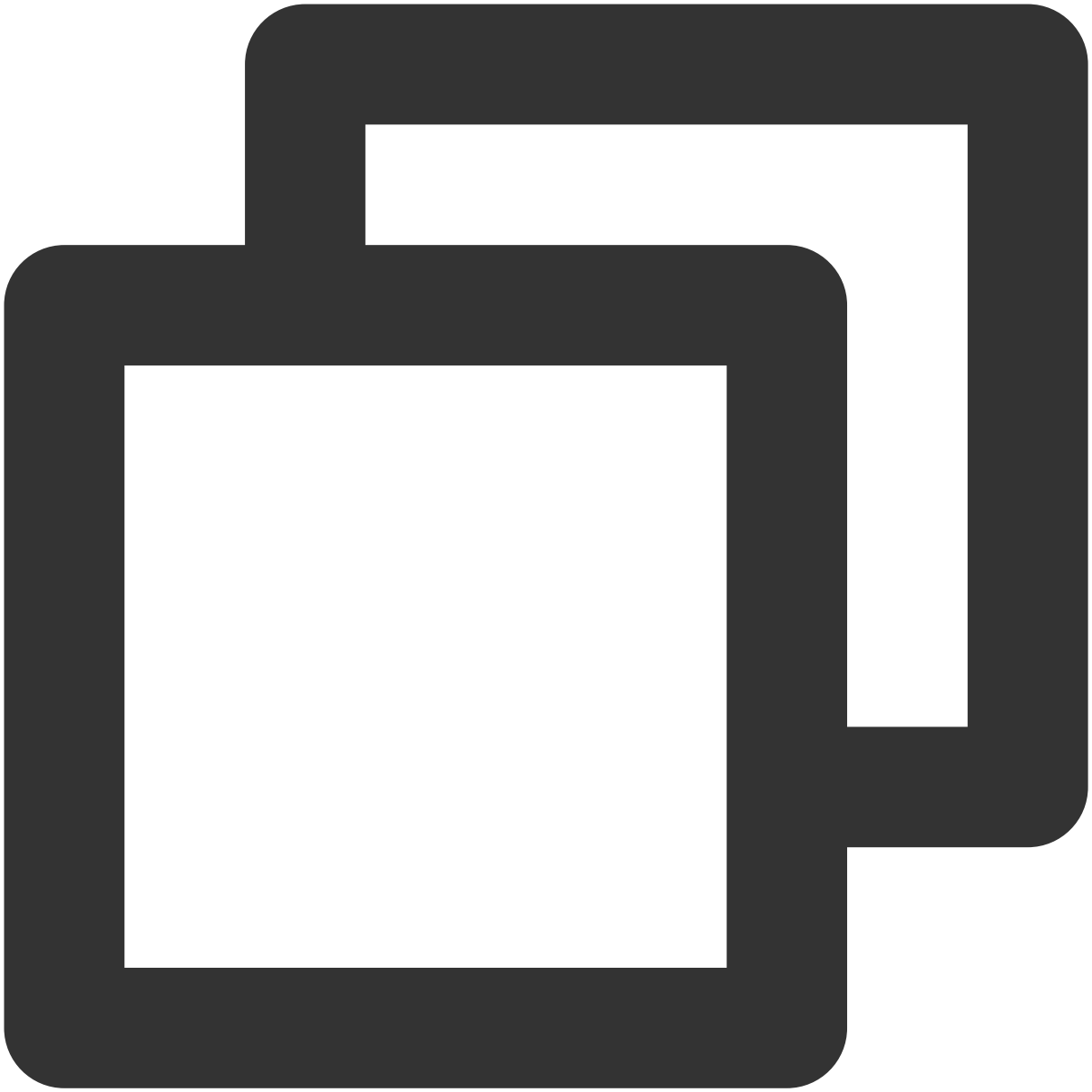
```
MOD(<expr1> integer|double|decimal, <expr2> integer|double|decimal)
<expr1> MOD <expr2>
```

支持引擎：SparkSQL、Presto

使用说明：返回expr1/expr2之后的余数。

返回类型：double|integer

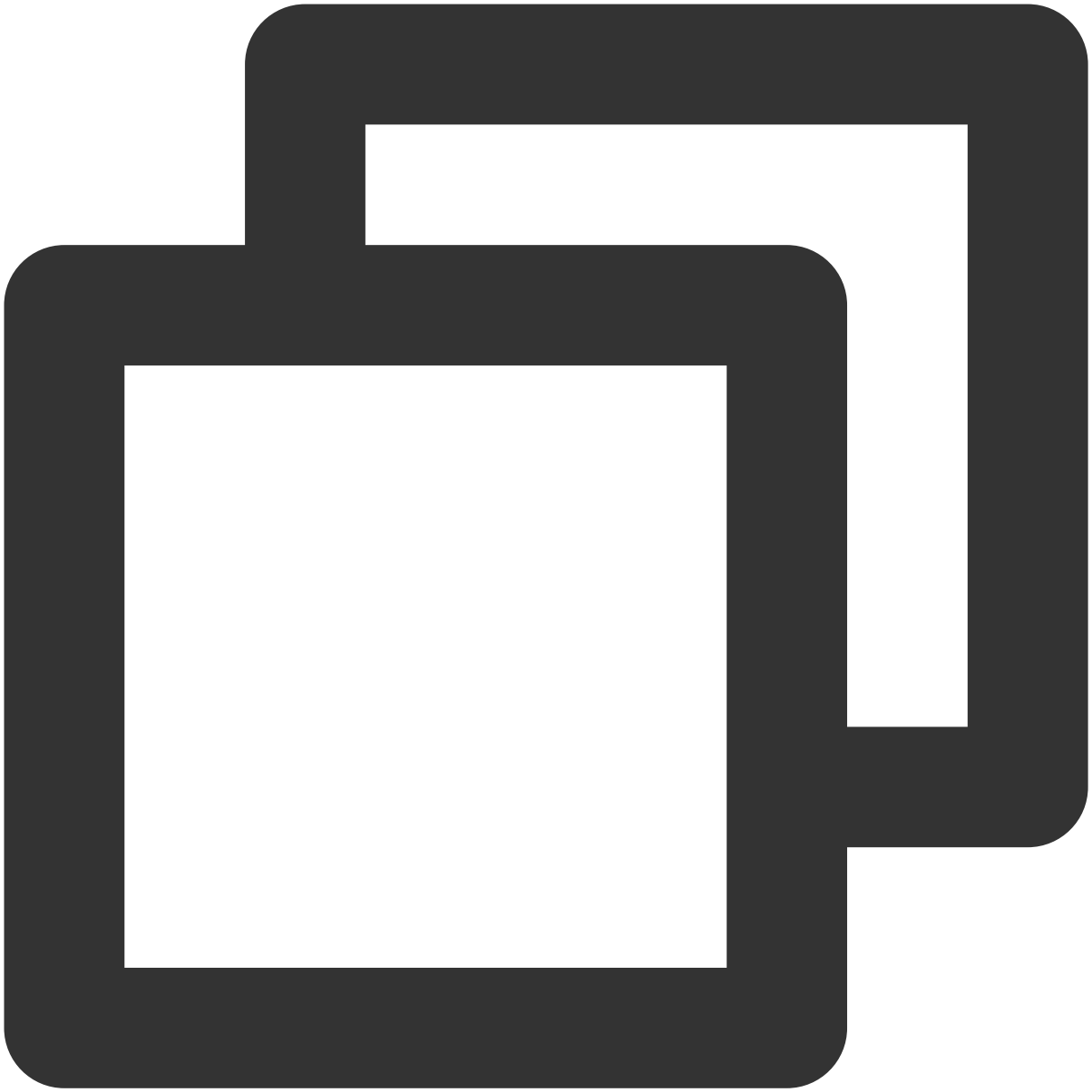
示例：



```
> SELECT 2 % 1.8;
 0.2
> SELECT MOD(2, 1.8);
 0.2
```

NEGATIVE

函数语法：



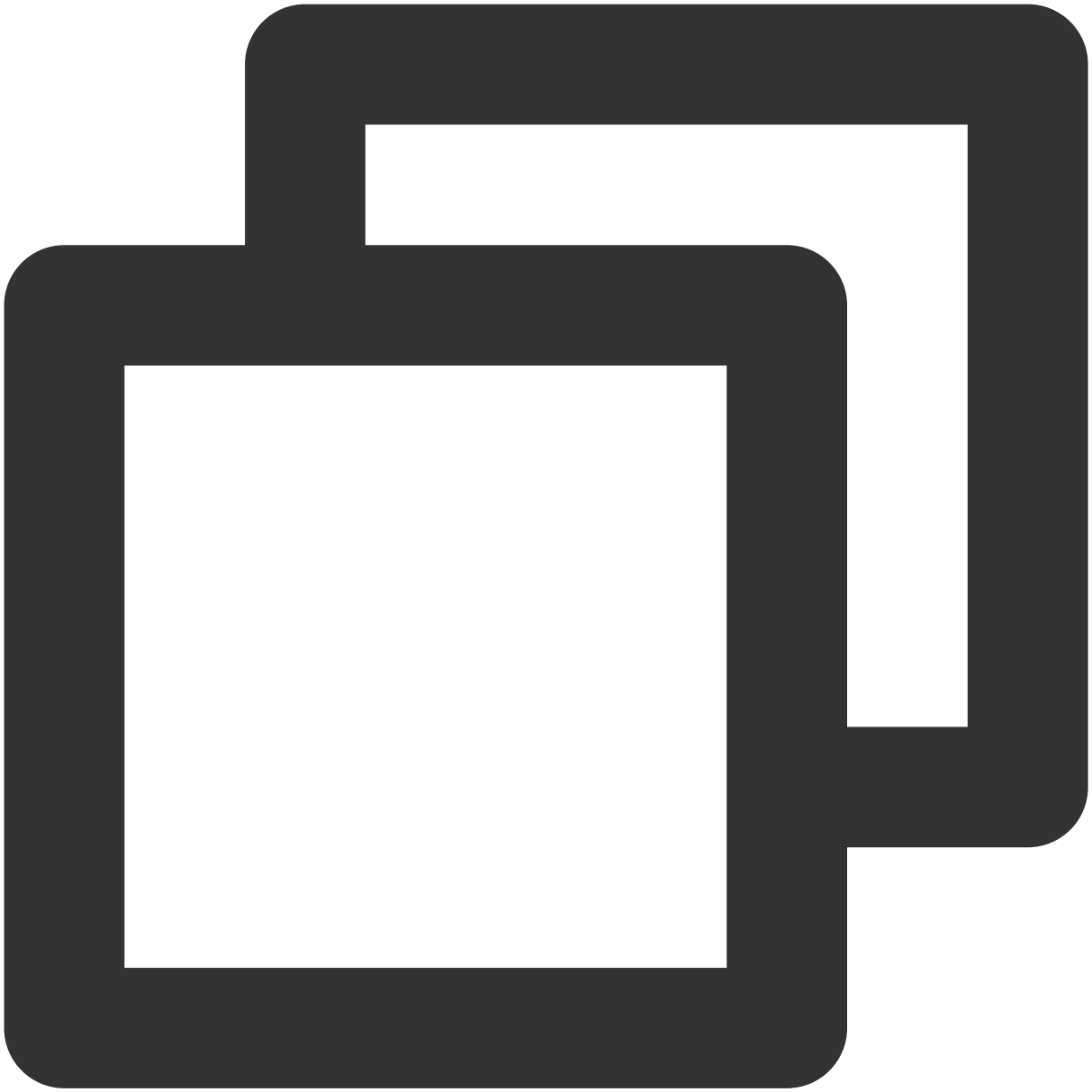
```
NEGATIVE(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的负数

返回类型：与expr一致

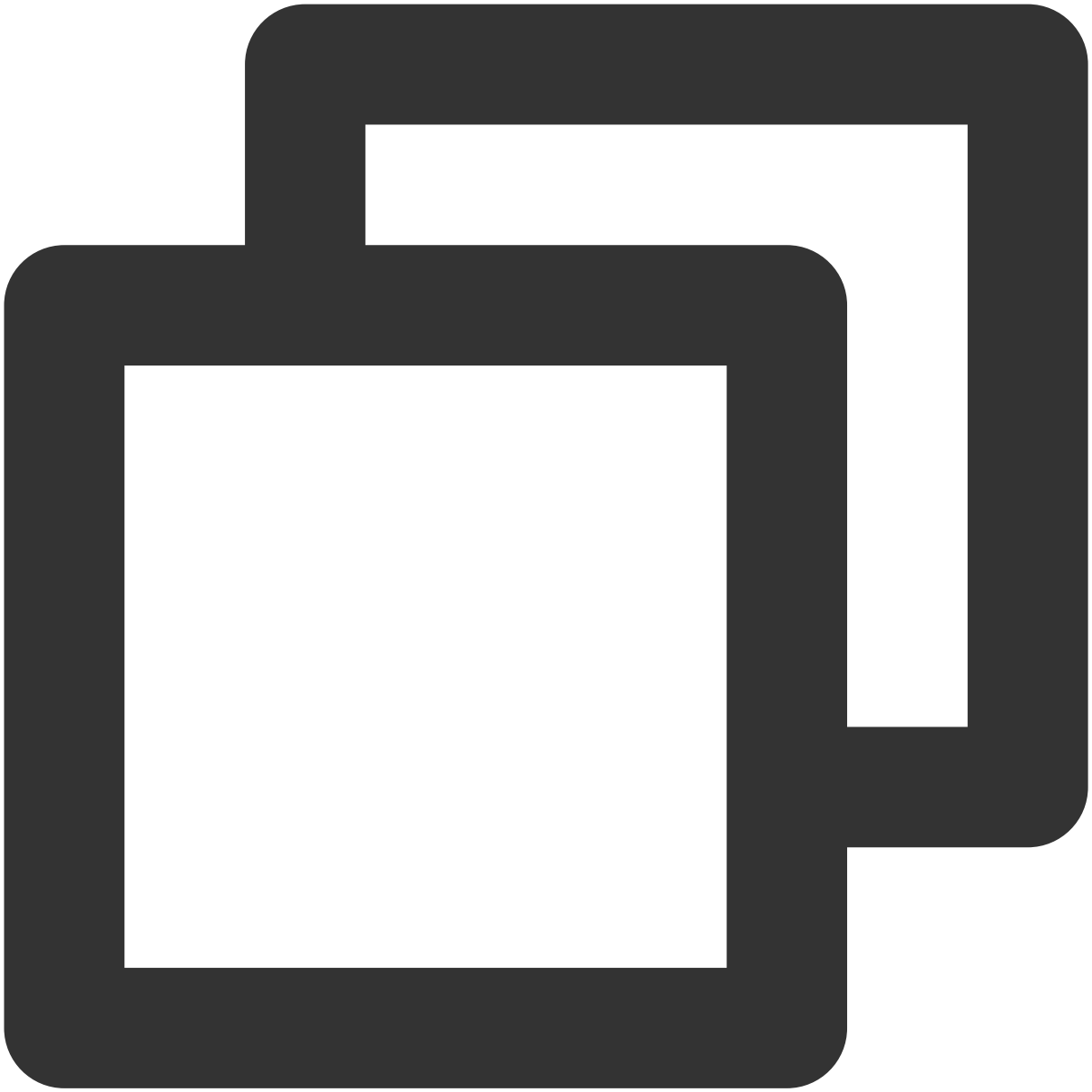
示例：



```
> SELECT negative(1);
 -1
```

PI

函数语法：



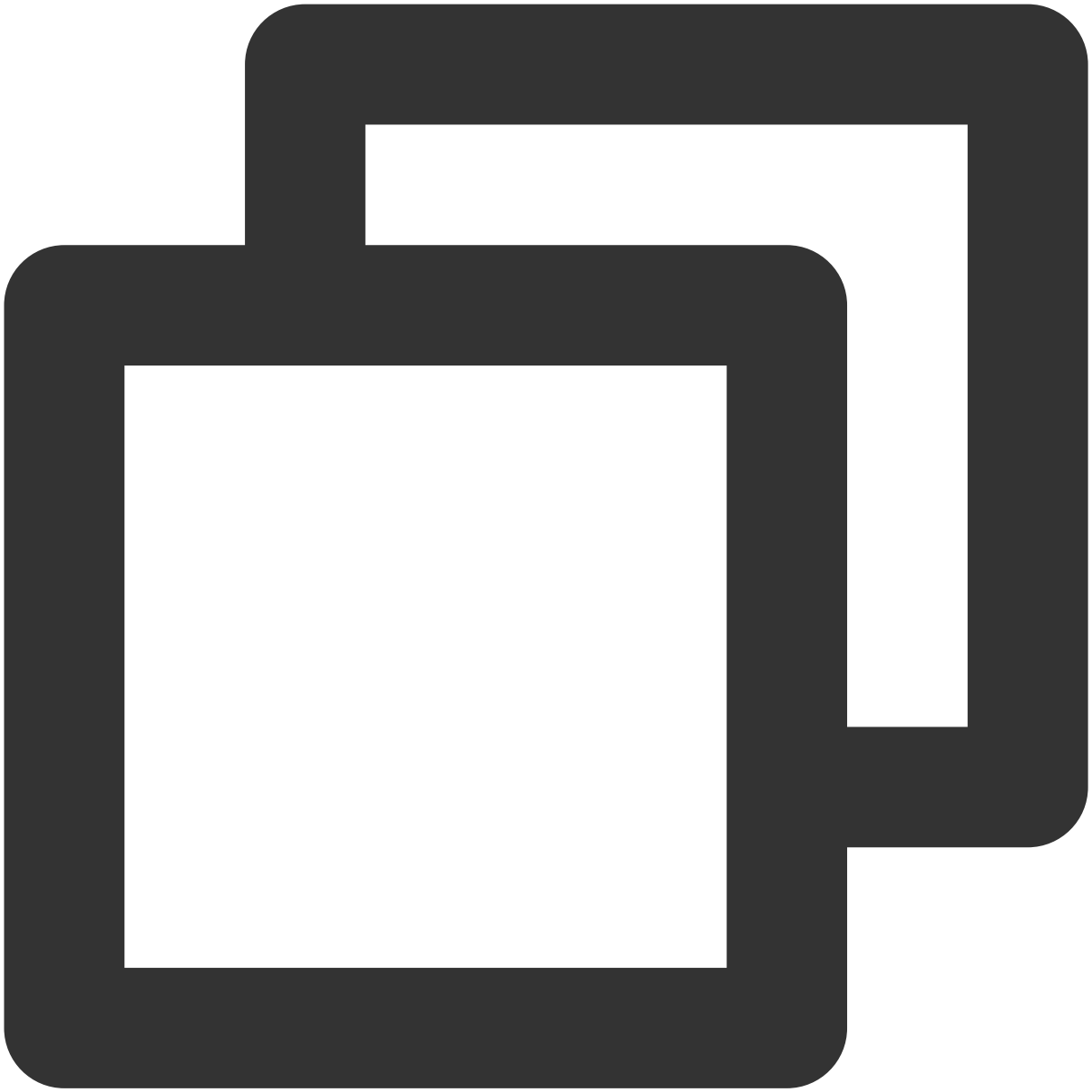
```
PI()
```

支持引擎：SparkSQL、Presto

使用说明：返回PI

返回类型：double

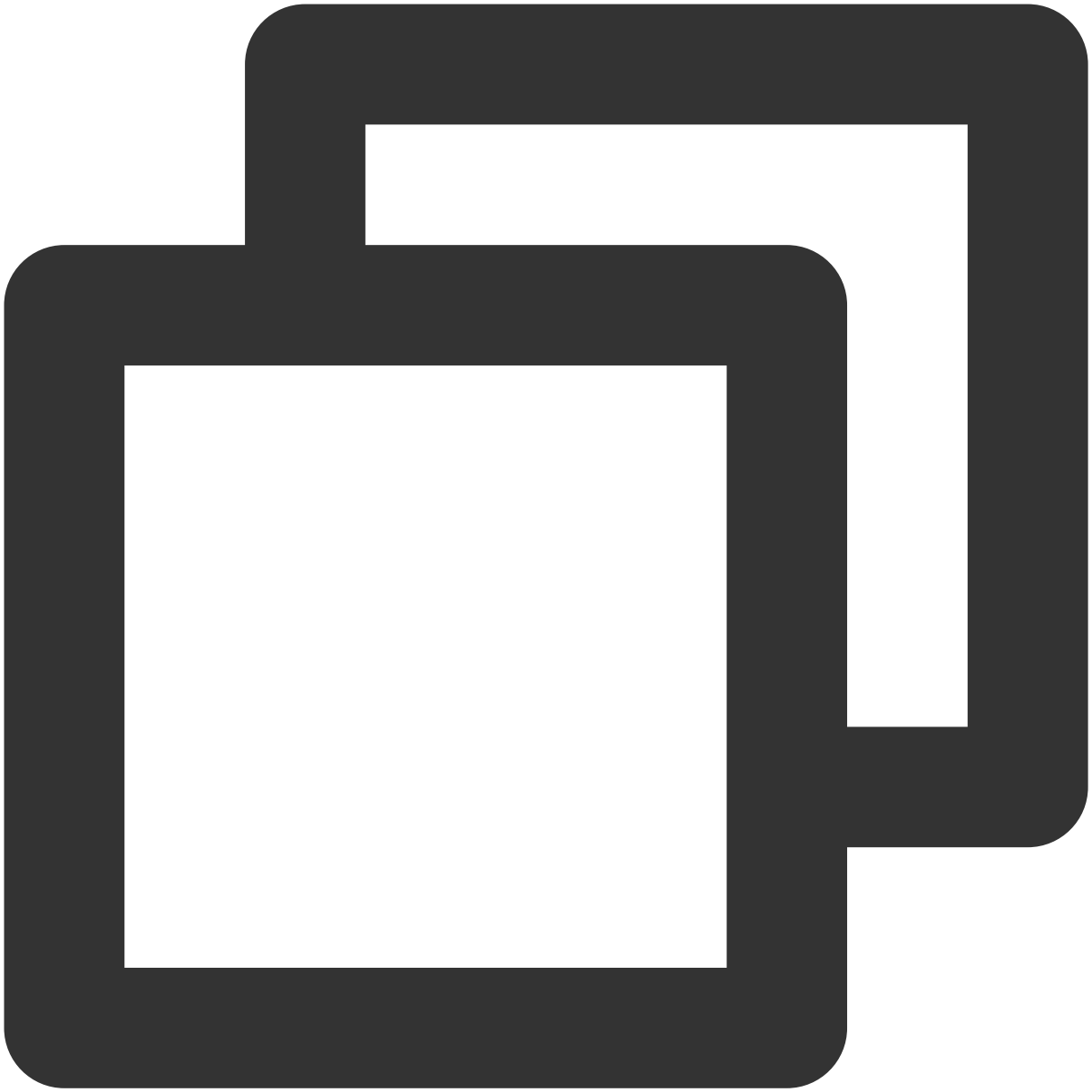
示例：



```
> SELECT pi();
 3.141592653589793
```

PMOD

函数语法：



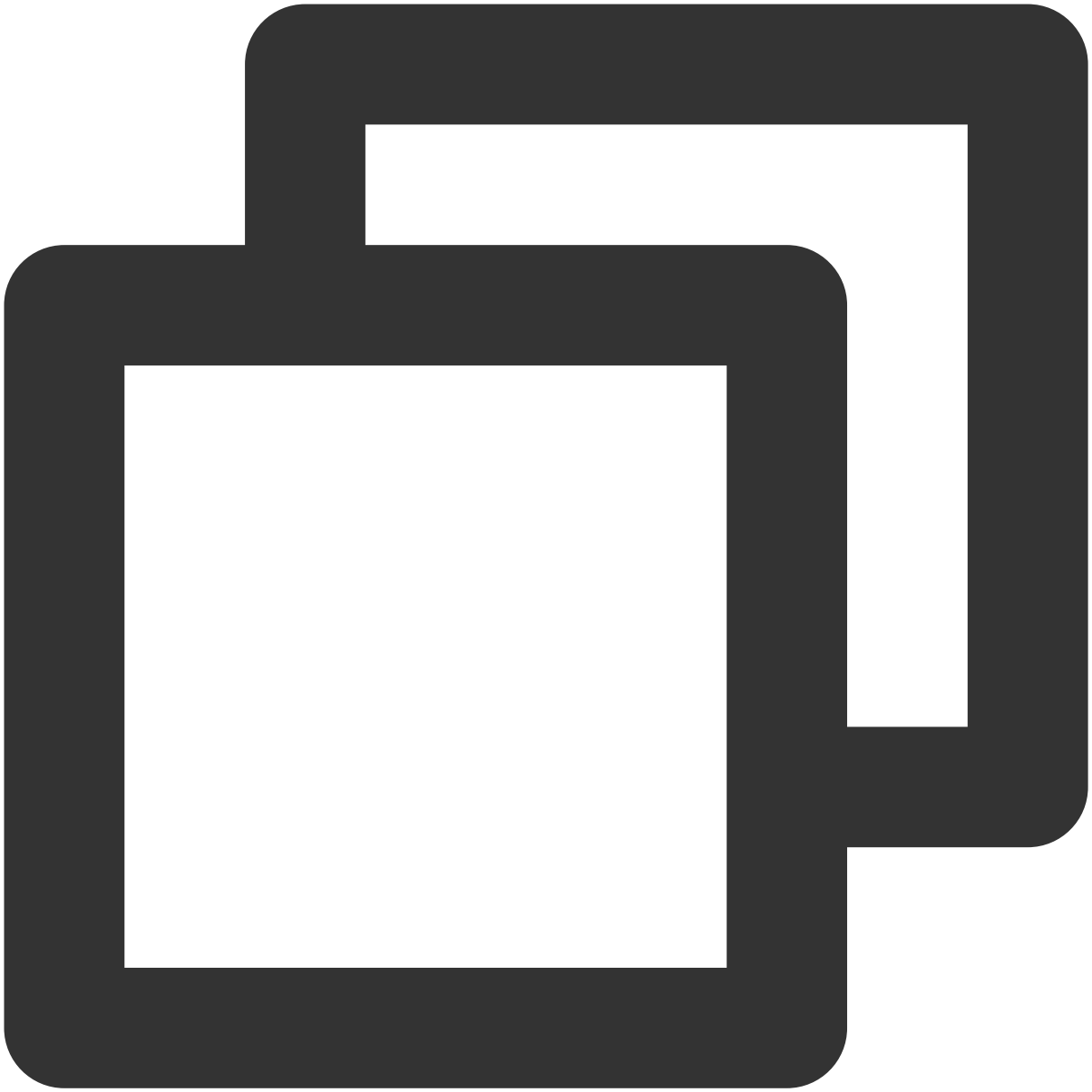
```
PMOD(<expr1> integer|double|decimal, <expr2> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr1 mod expr2的正值。

返回类型：double

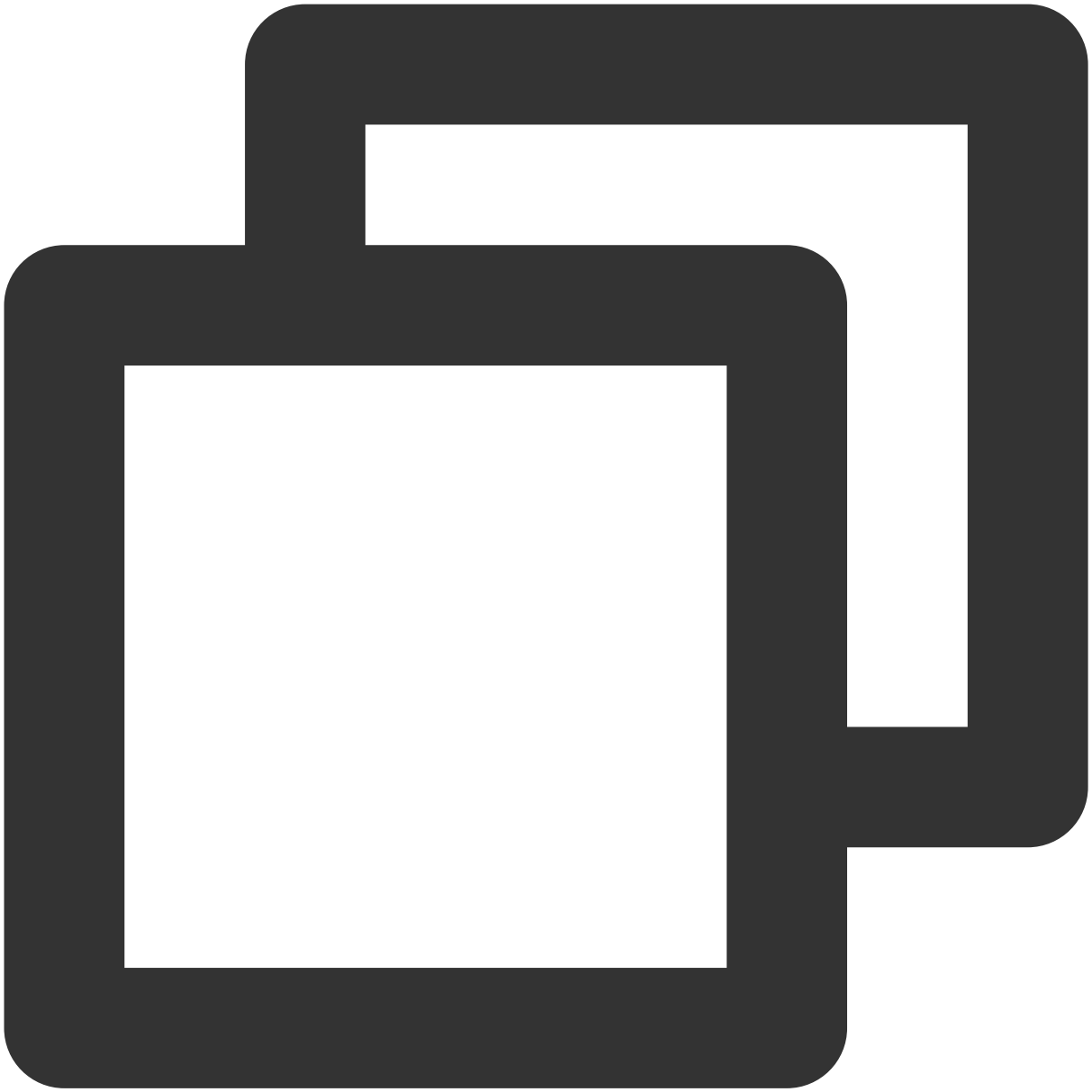
示例：



```
> SELECT pmod(10, 3);
 1
> SELECT pmod(-10, 3);
 2
```

POSITIVE

函数语法：



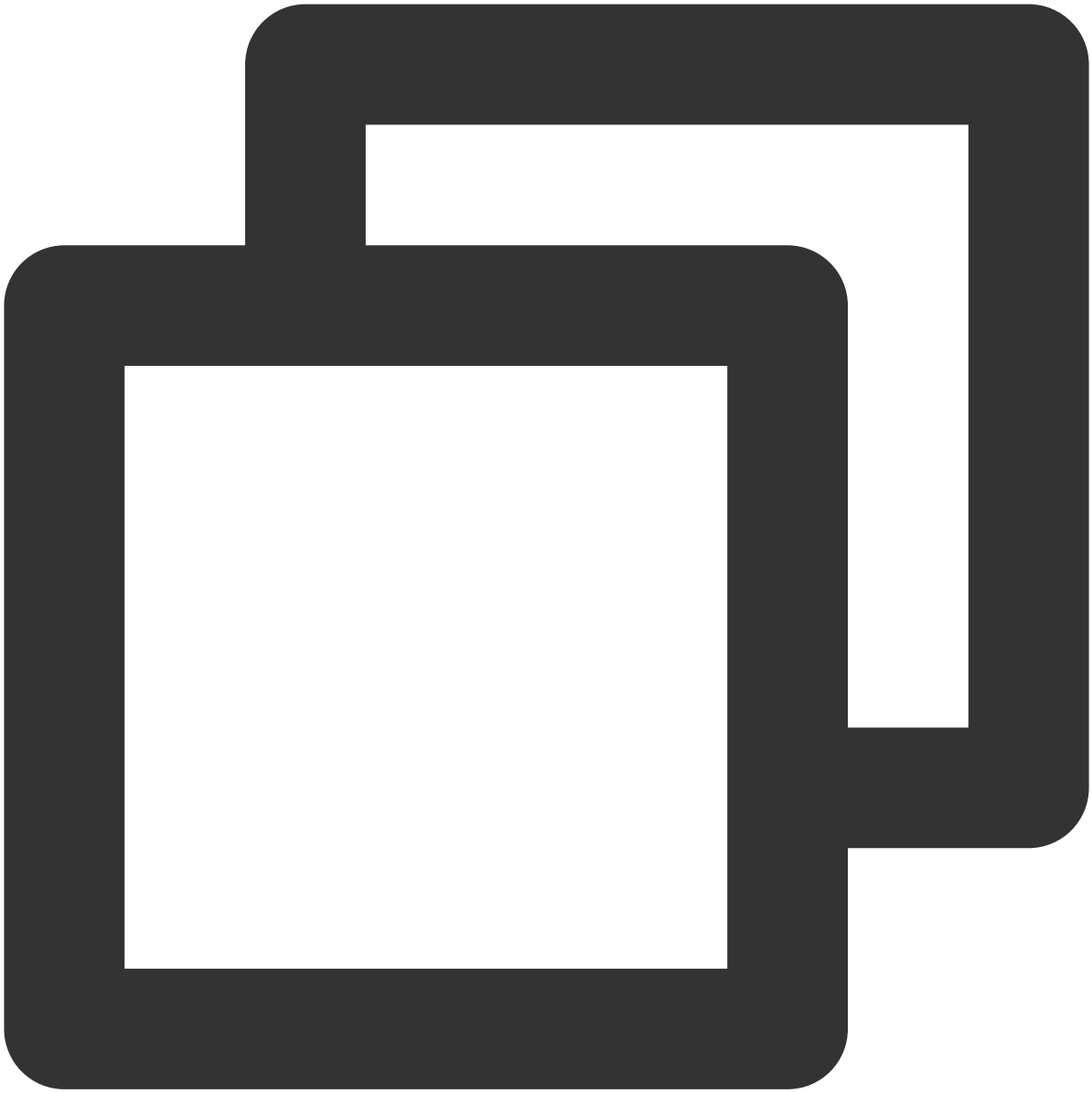
```
POSITIVE(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr

返回类型：与expr一致

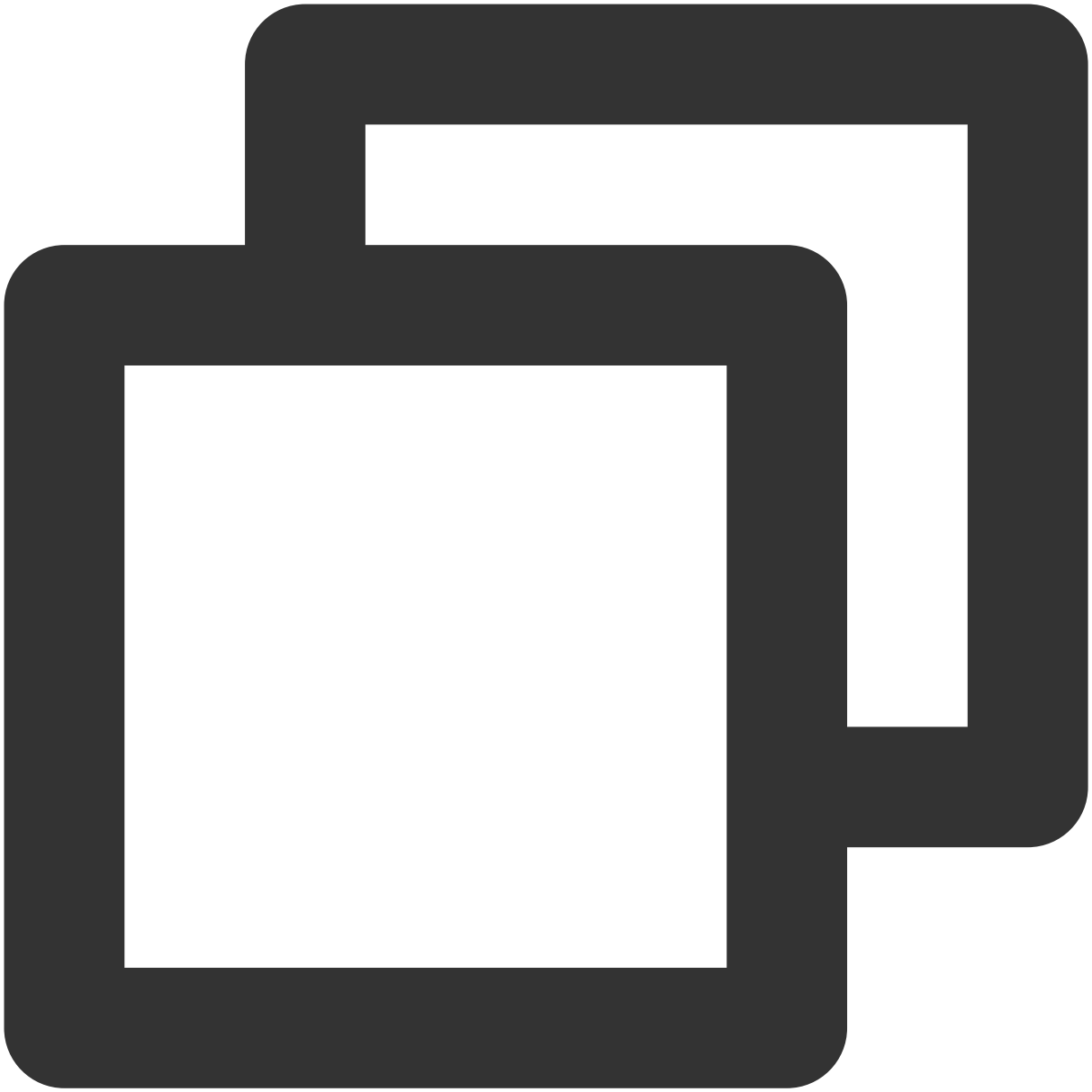
示例：



```
> SELECT positive(1);
 1
```

POWER

函数语法：



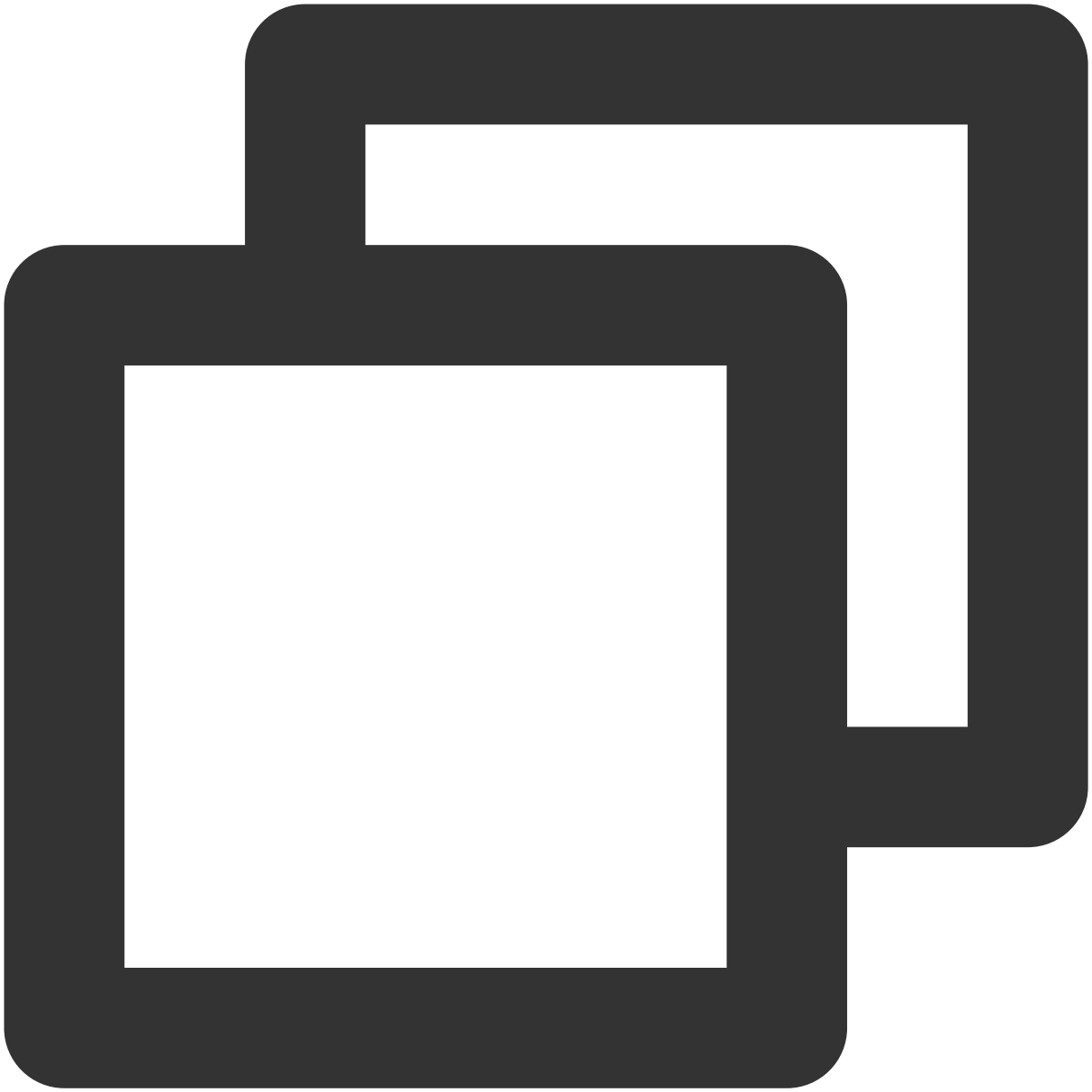
```
POWER(<base> integer|double|decimal, <number> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回base的number次方

返回类型：double

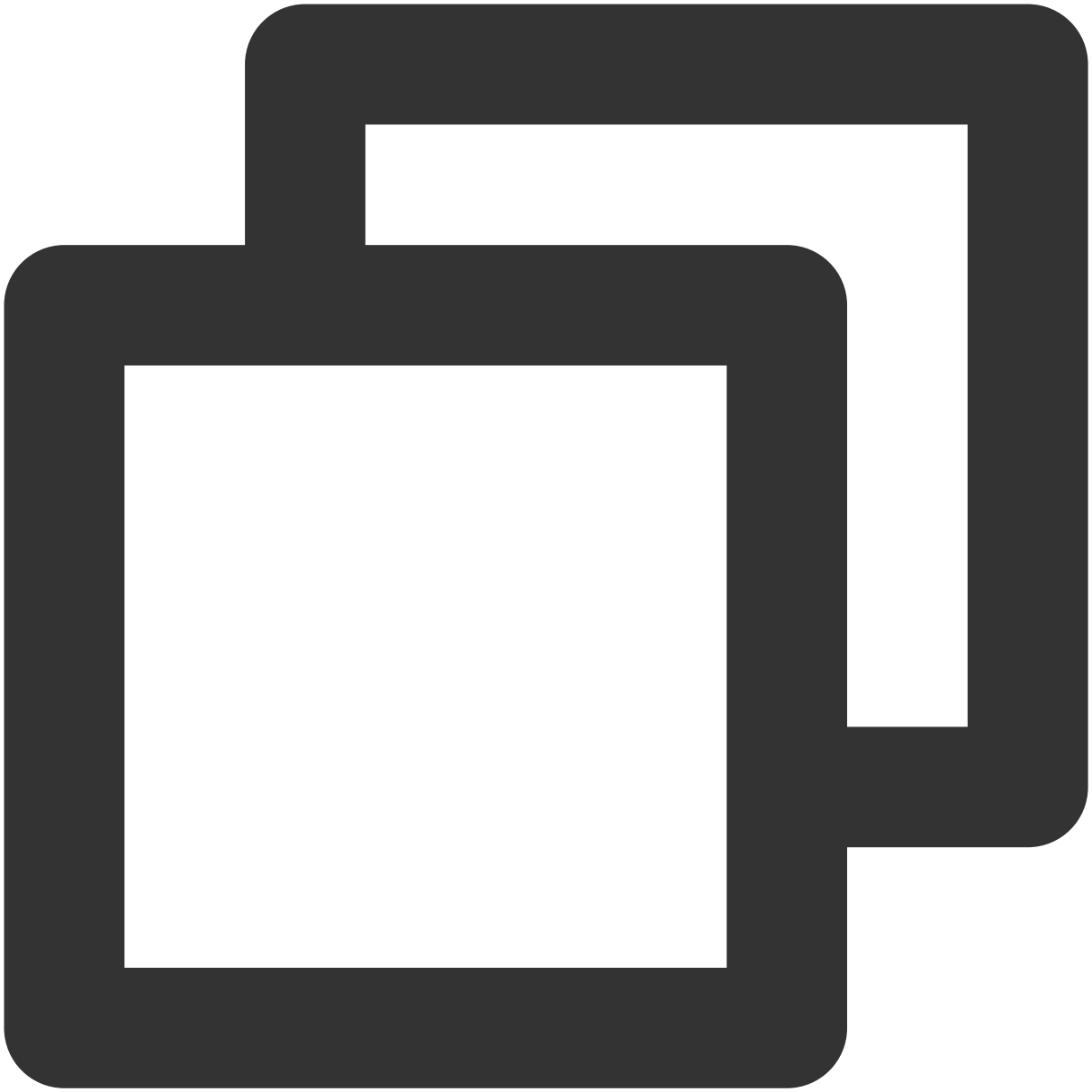
示例：



```
> SELECT power(2, 3);
 8.0
```

POW

函数语法：



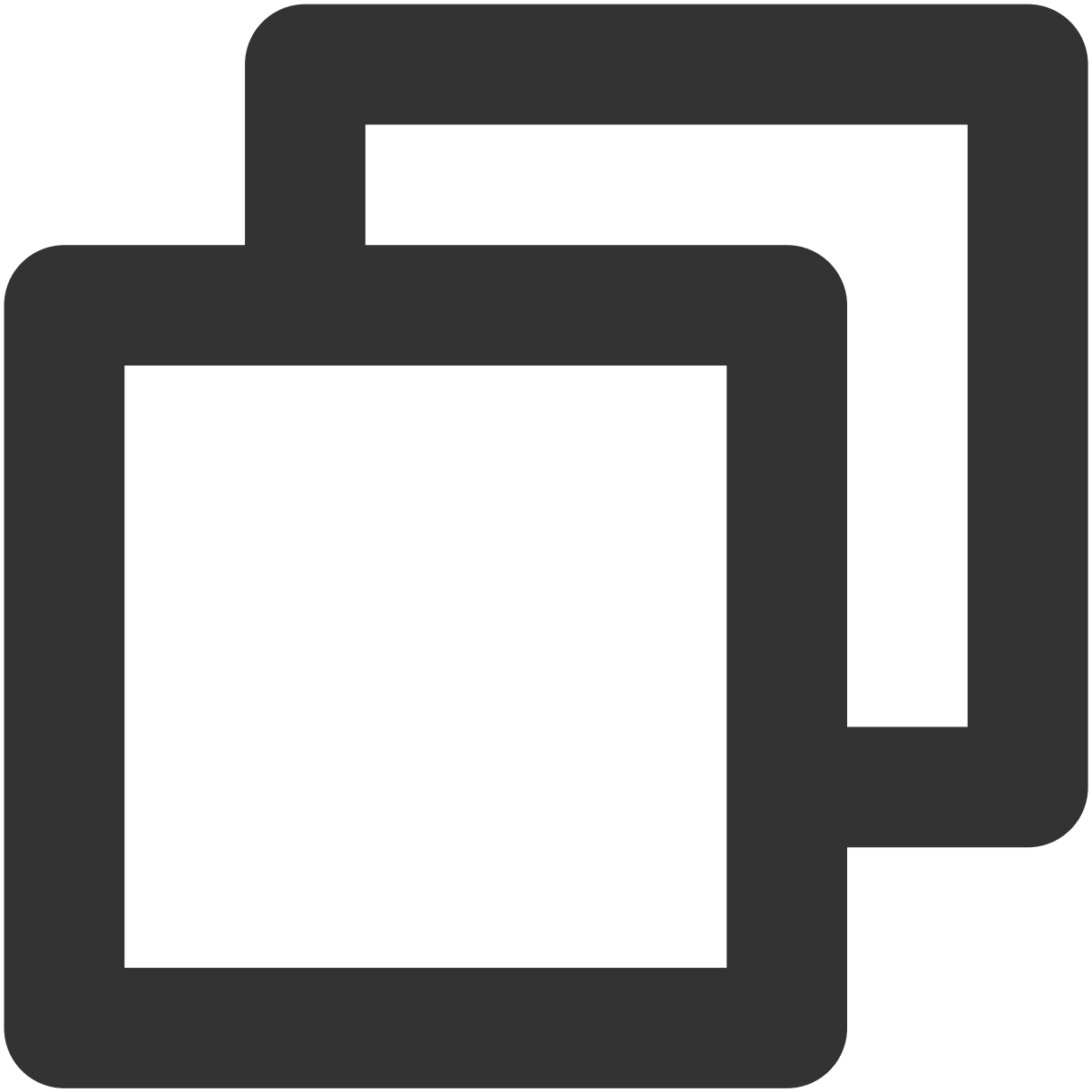
```
POW(<base> integer|double|decimal, <number> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回base的number次方

返回类型：double

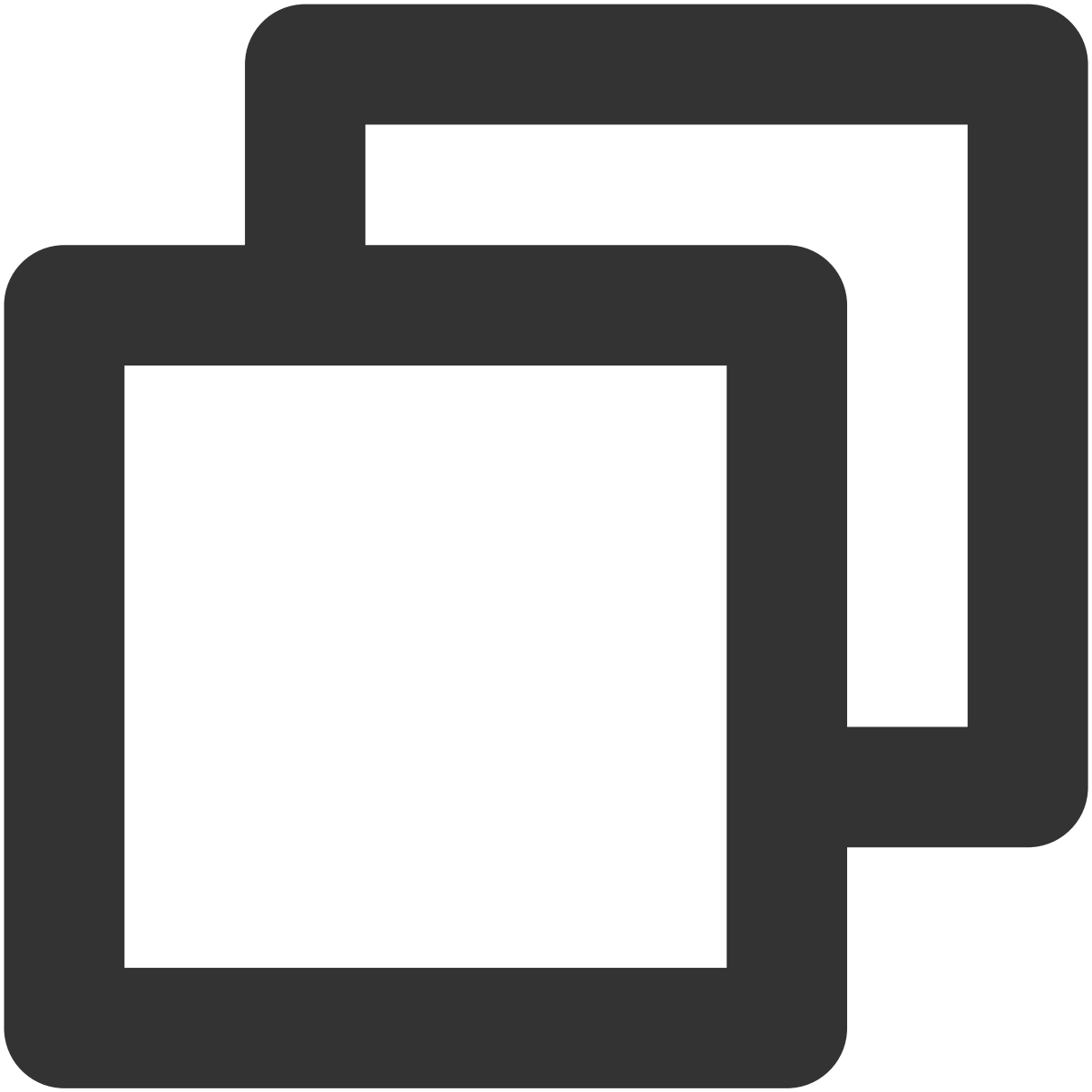
示例：



```
> SELECT pow(2, 3);
 8.0
```

RADIANS

函数语法：



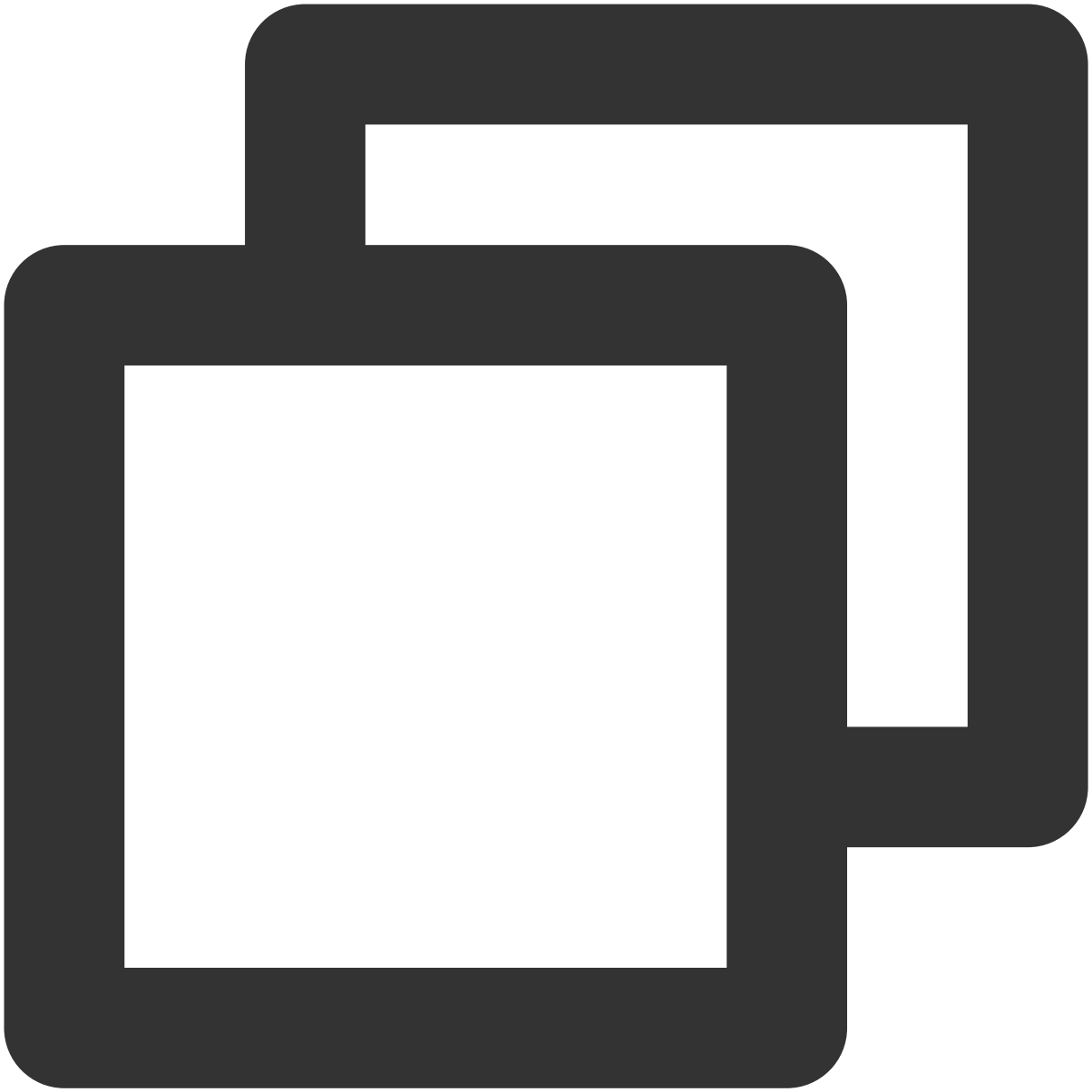
```
RADIANS(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：将度转换为弧度

返回类型：double

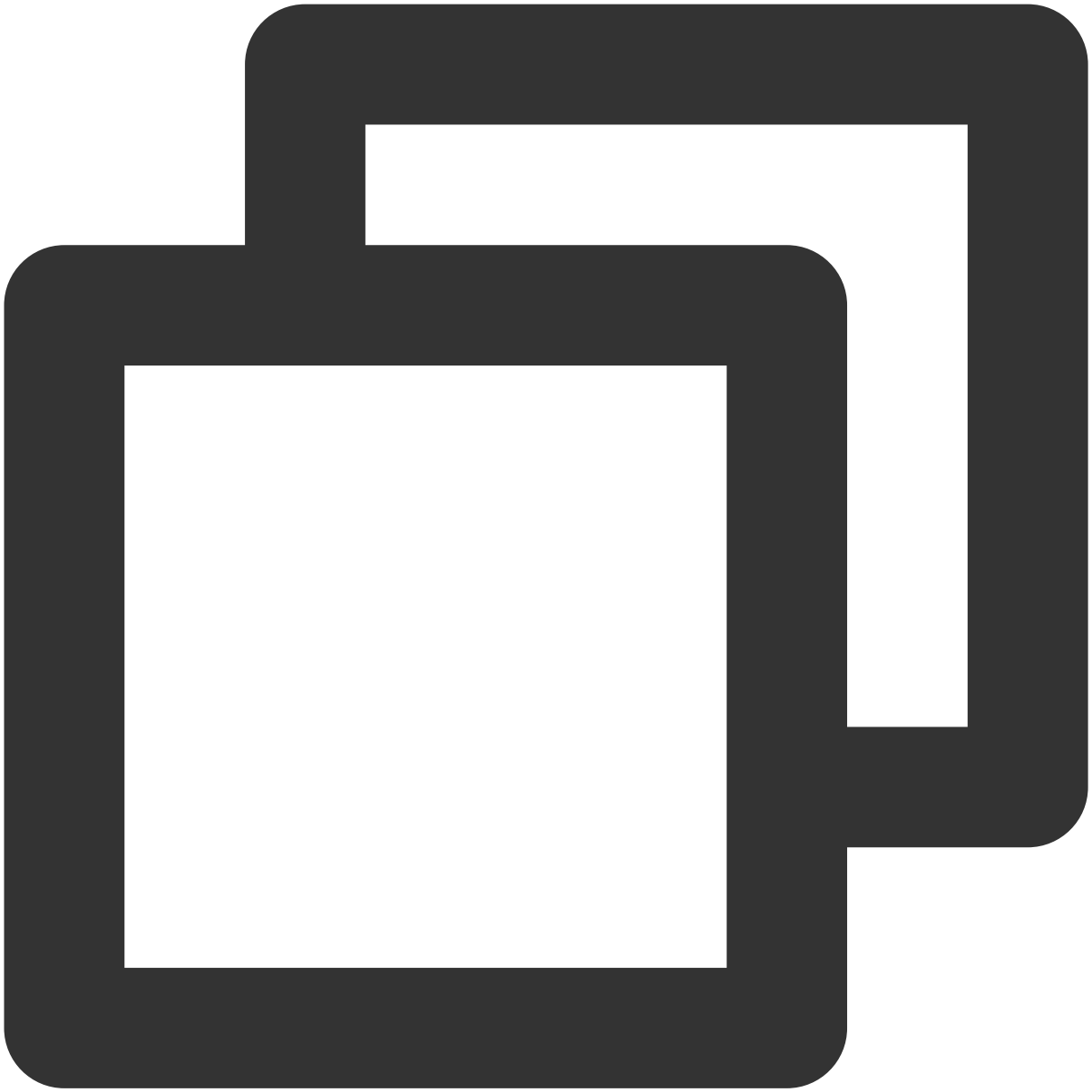
示例：



```
> SELECT radians(180);
 3.141592653589793
```

RINT

函数语法：



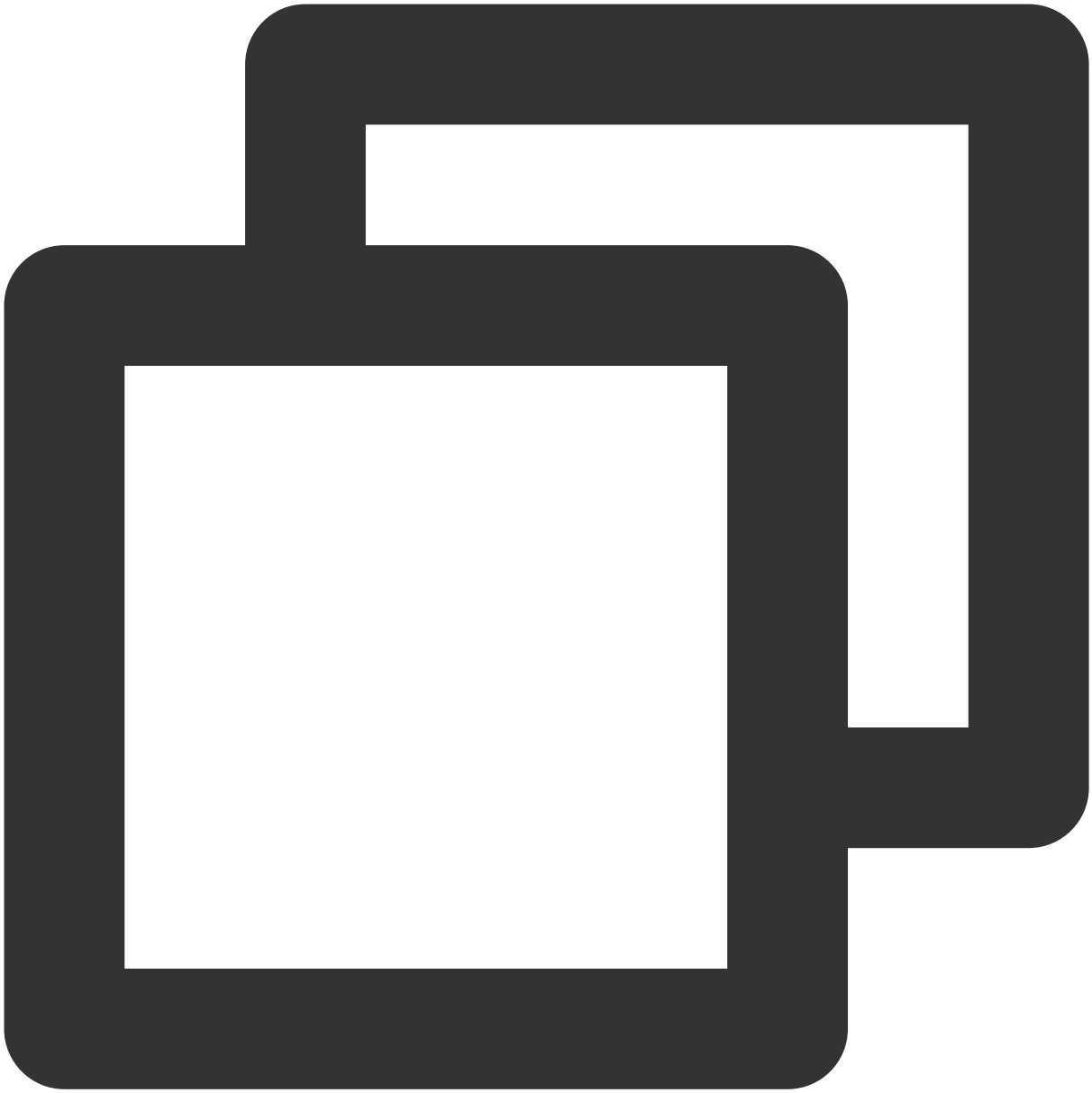
```
RINT(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回与参数值最近且等于数学整数的双精度值

返回类型：double

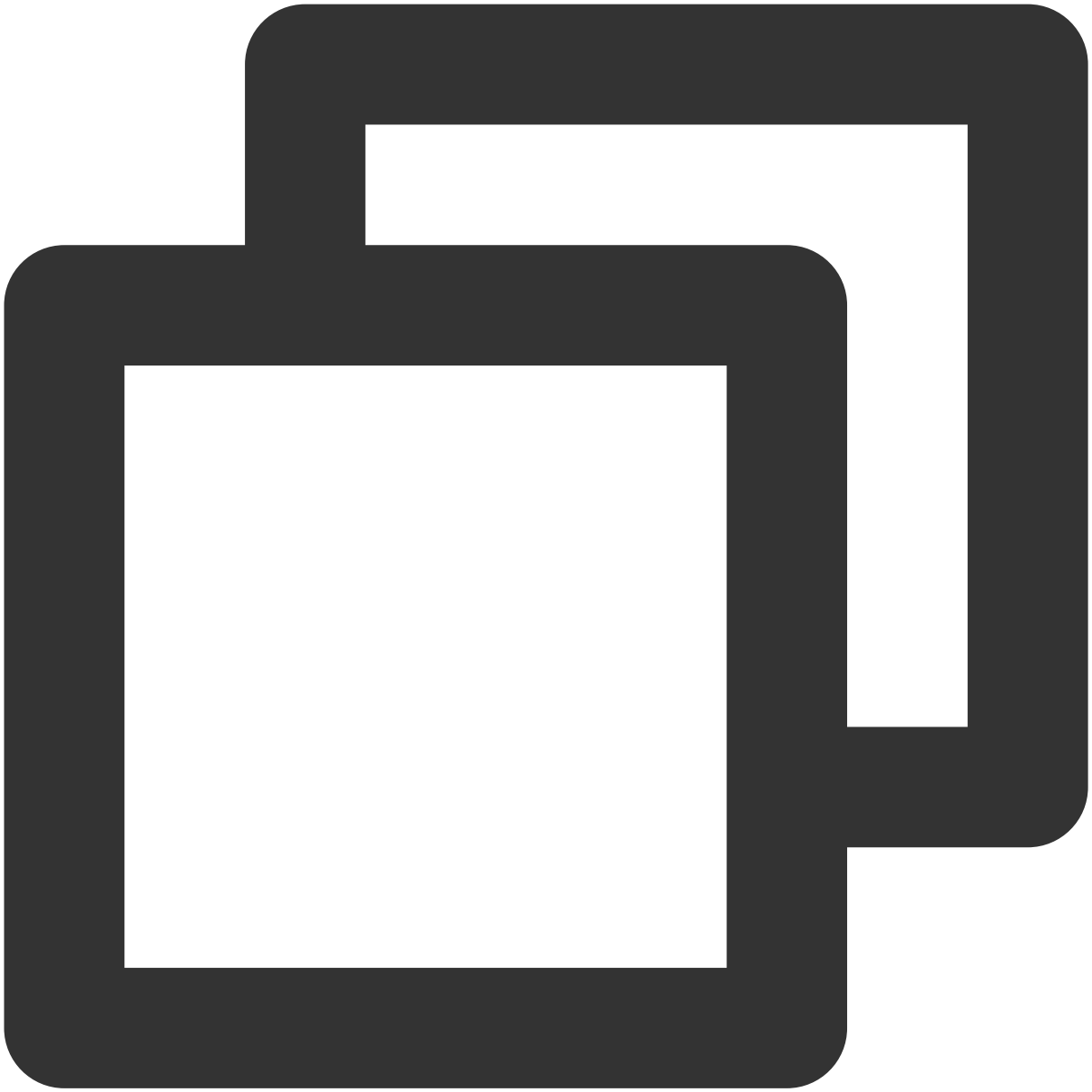
示例：



```
> SELECT rint(12.3456);
 12.0
```

ROUND

函数语法：



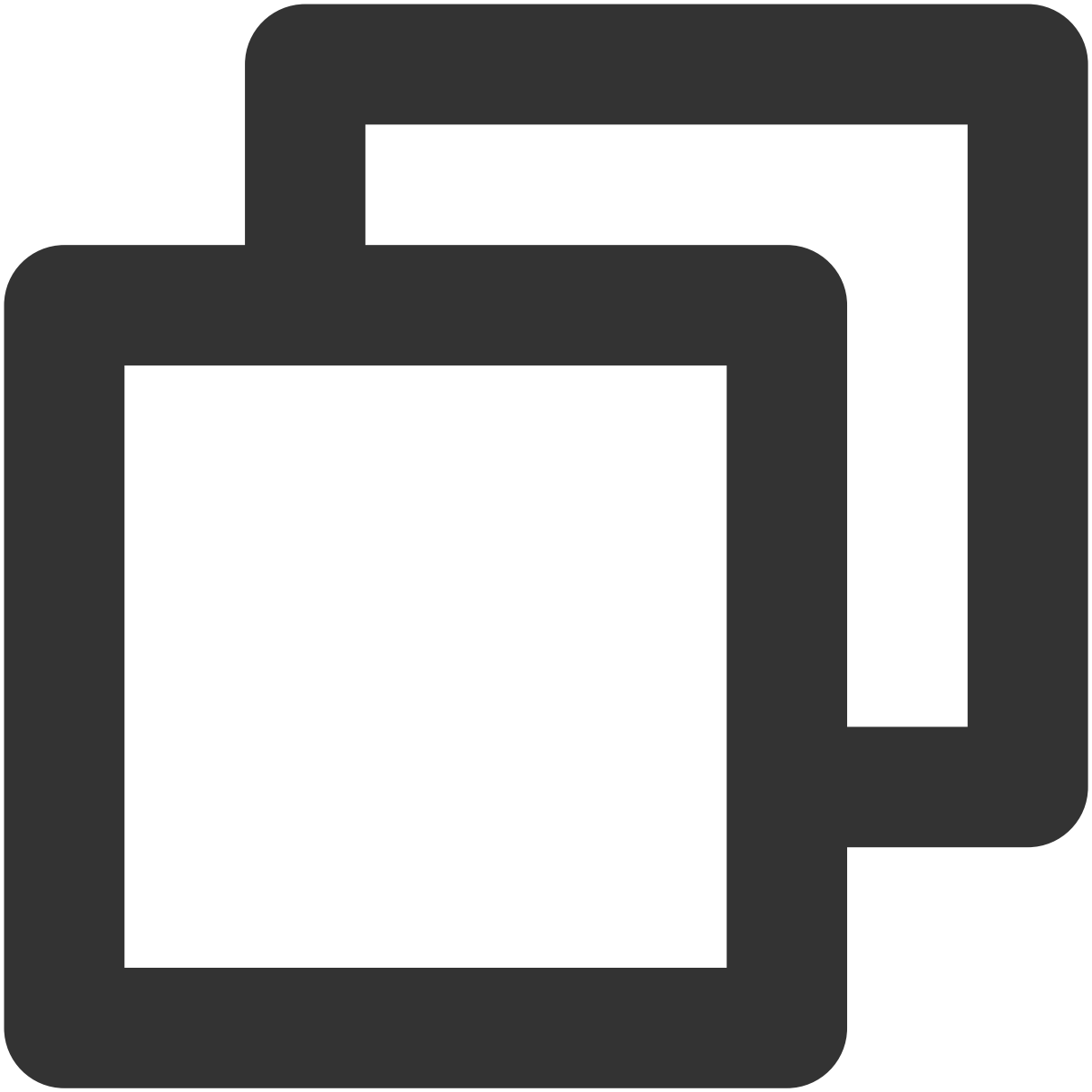
```
ROUND(<expr> integer|double|decimal, <d> integer)
```

支持引擎：SparkSQL、Presto

使用说明：使用半向上舍入模式将expr舍入到d位小数。

返回类型：double

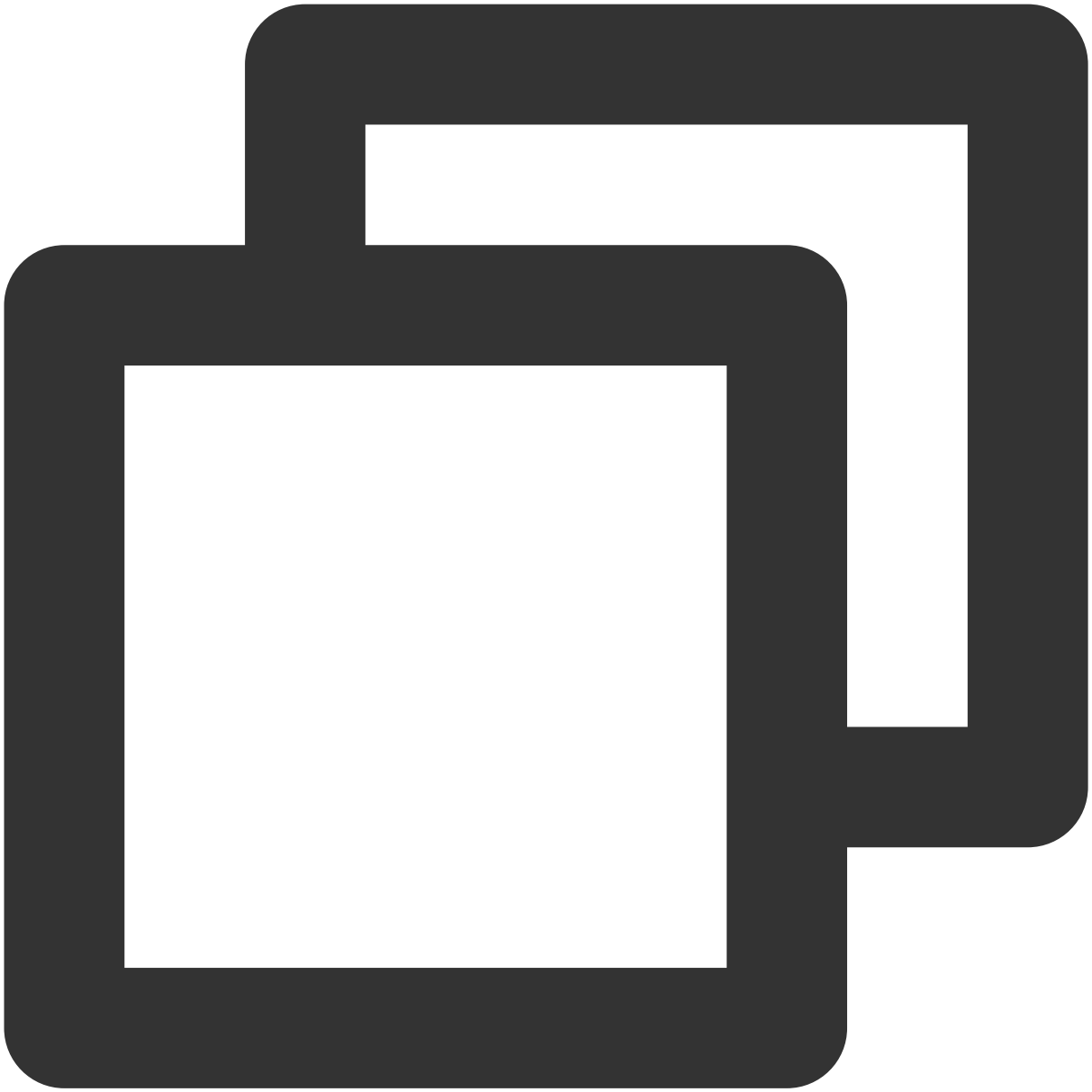
示例：



```
> SELECT round(2.5, 0);
 3
```

SHIFTLEFT

函数语法：



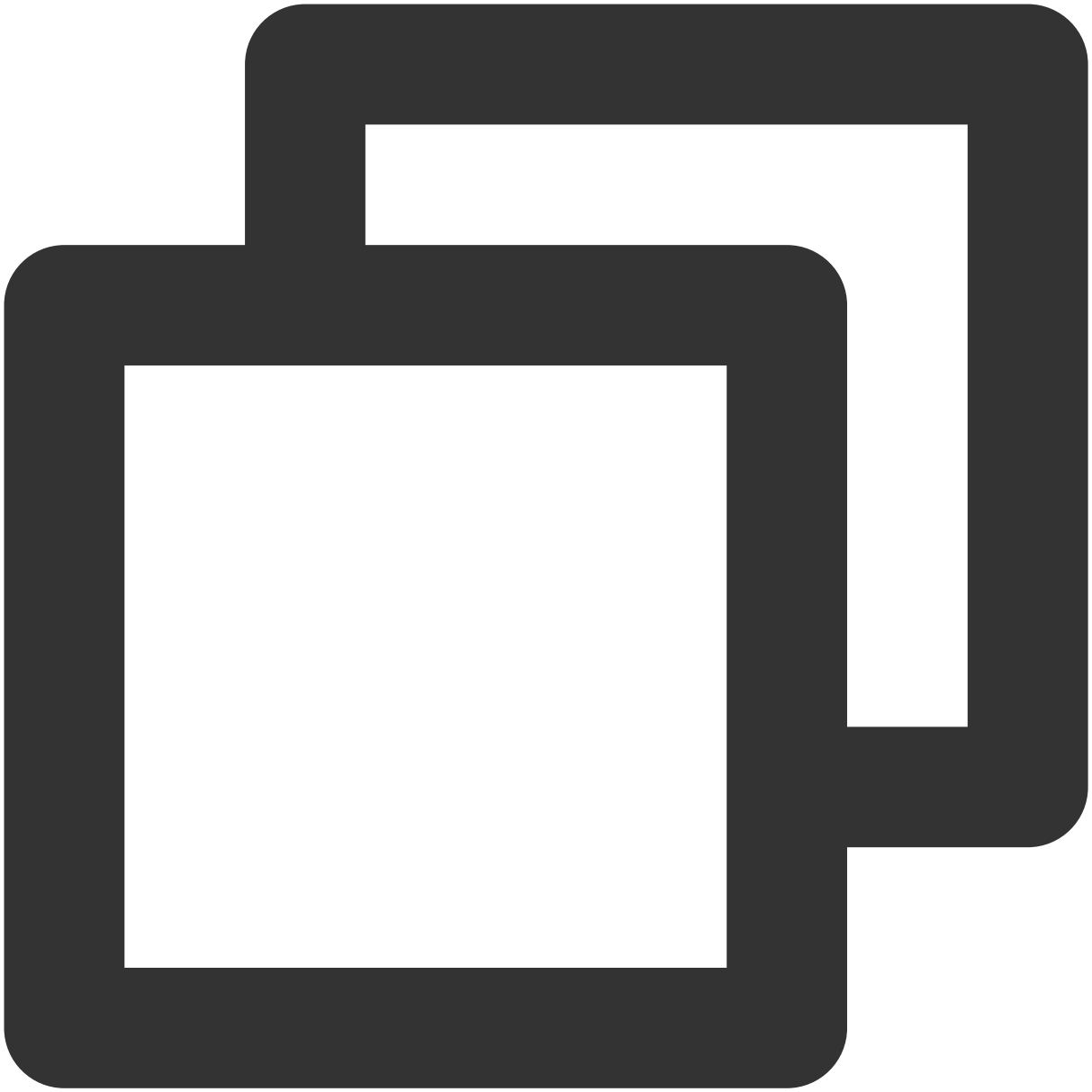
```
SHIFTLEFT(<base> integer|double|decimal, <expr> integer)
```

支持引擎：SparkSQL、Presto

使用说明：按位向左移位。

返回类型：int |bigint

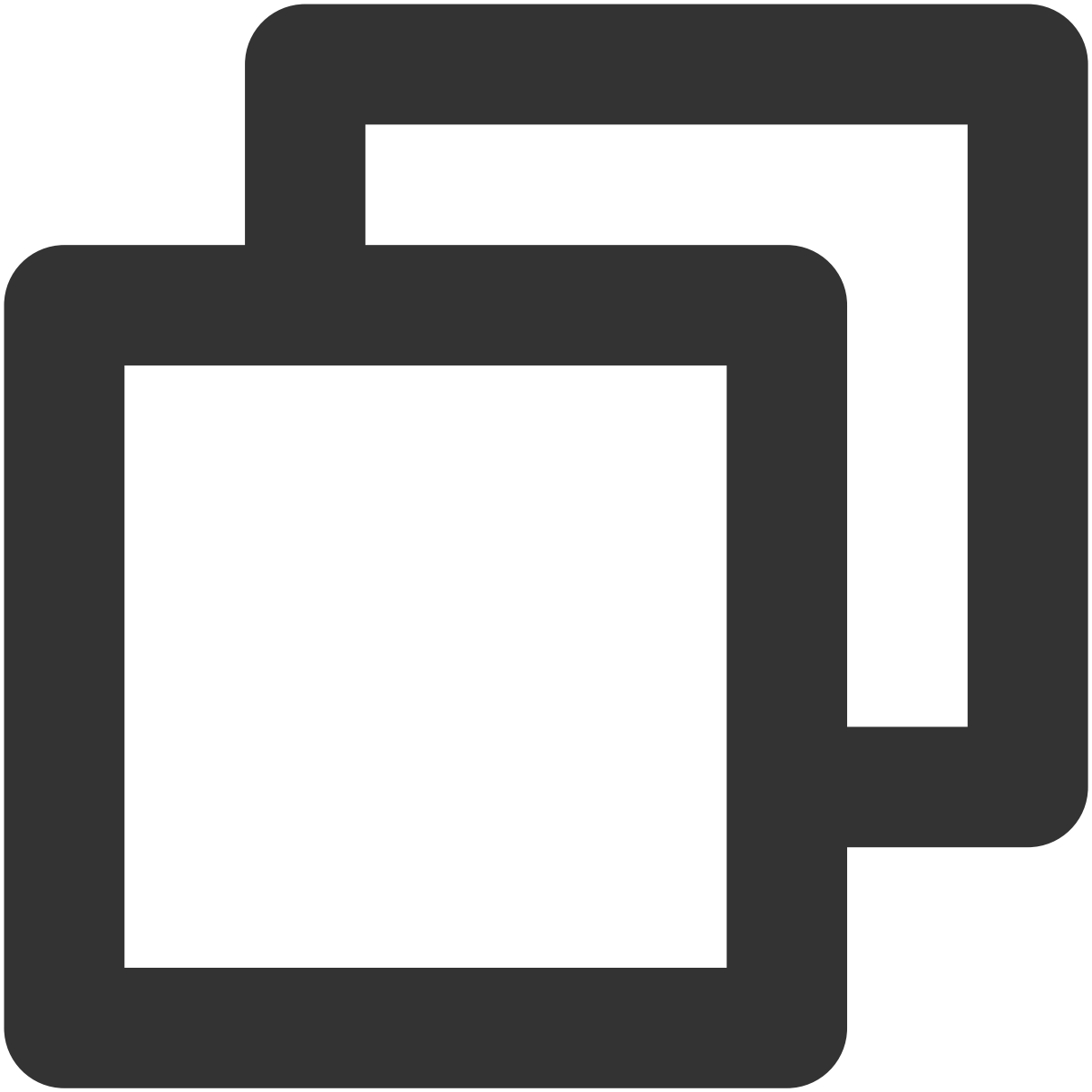
示例：



```
> SELECT shiftleft(2, 1);
 4
```

SHIFTRIGHT

函数语法：



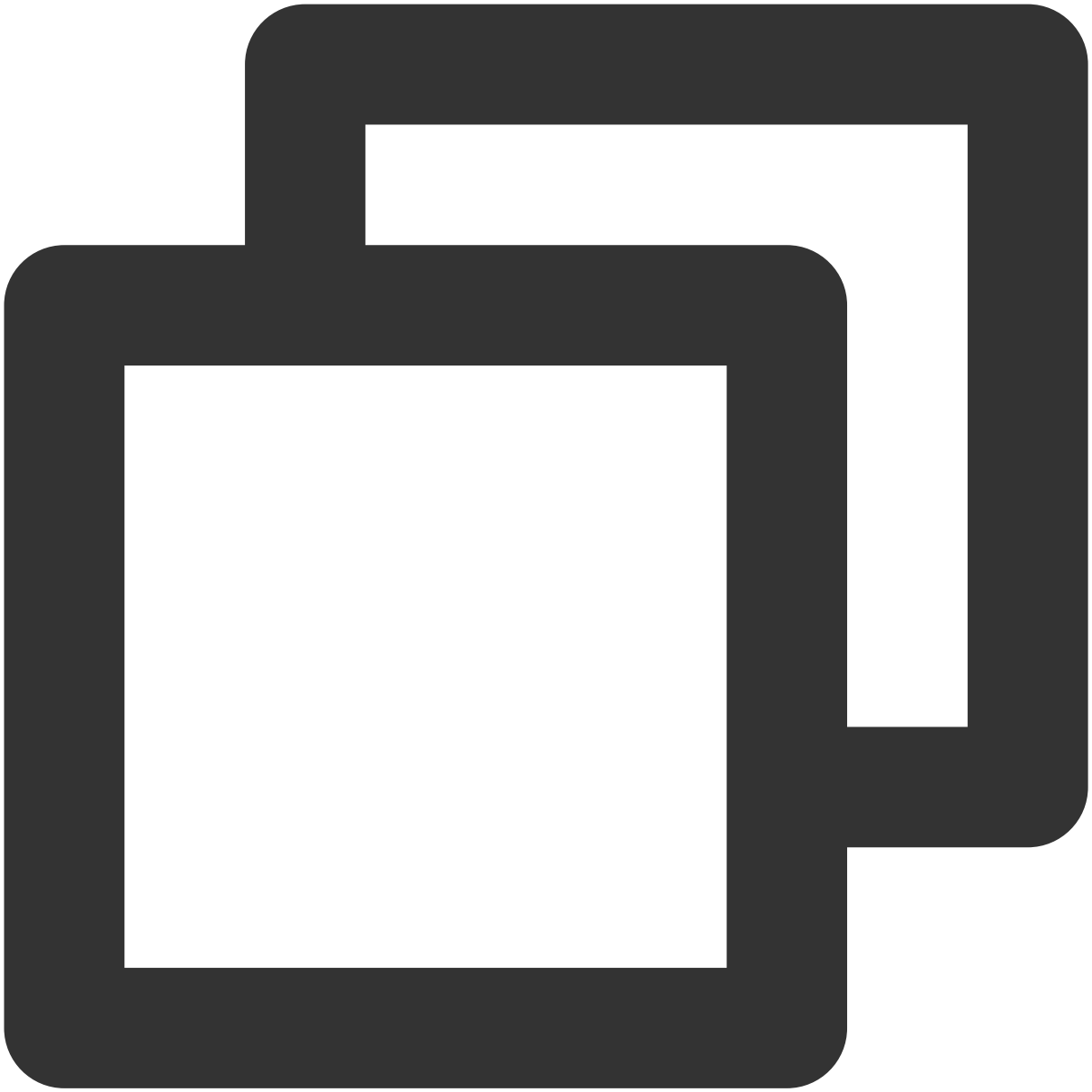
```
SHIFTRIGHT(<base> integer|double|decimal, <expr> integer)
```

支持引擎：SparkSQL、Presto

使用说明：按位向右移位。

返回类型：int |bigint

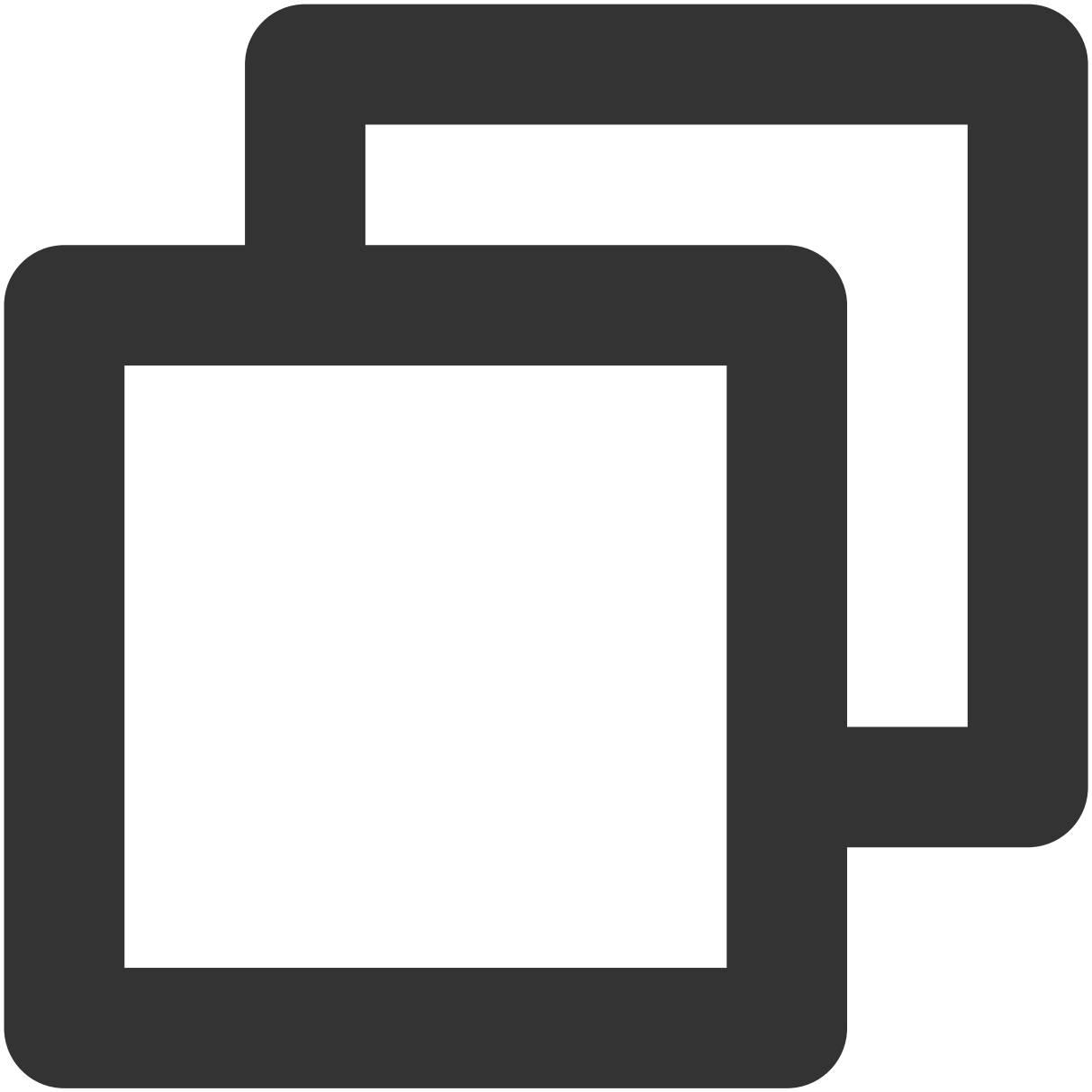
示例：



```
> SELECT shiftright(4, 1);
 2
```

SHIFTRIGHTUNSIGNED

函数语法：



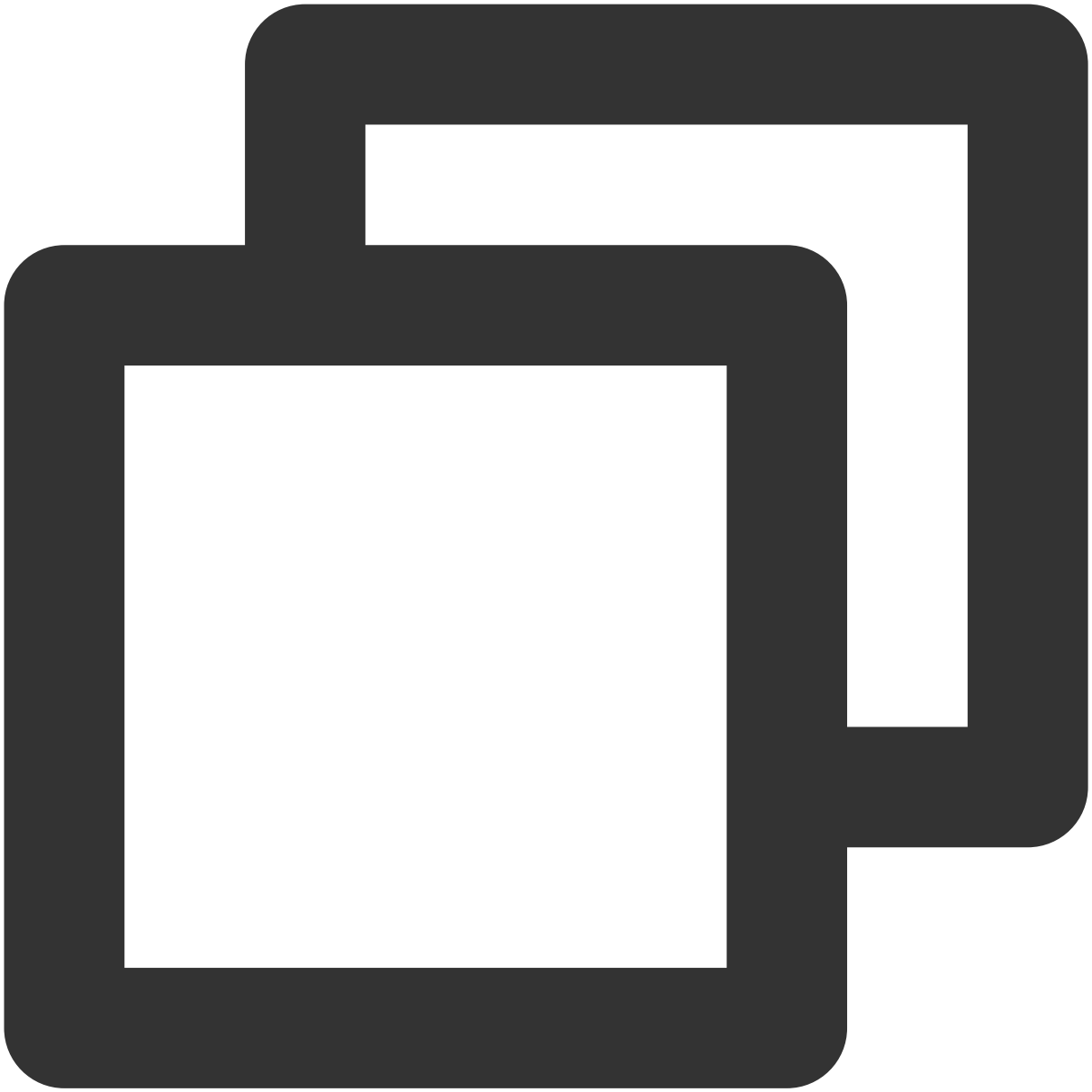
```
SHIFTRIGHTUNSIGNED(<base> integer|double|decimal, <expr> integer)
```

支持引擎：SparkSQL、Presto

使用说明：按位无符号右移。

返回类型：int |bigint

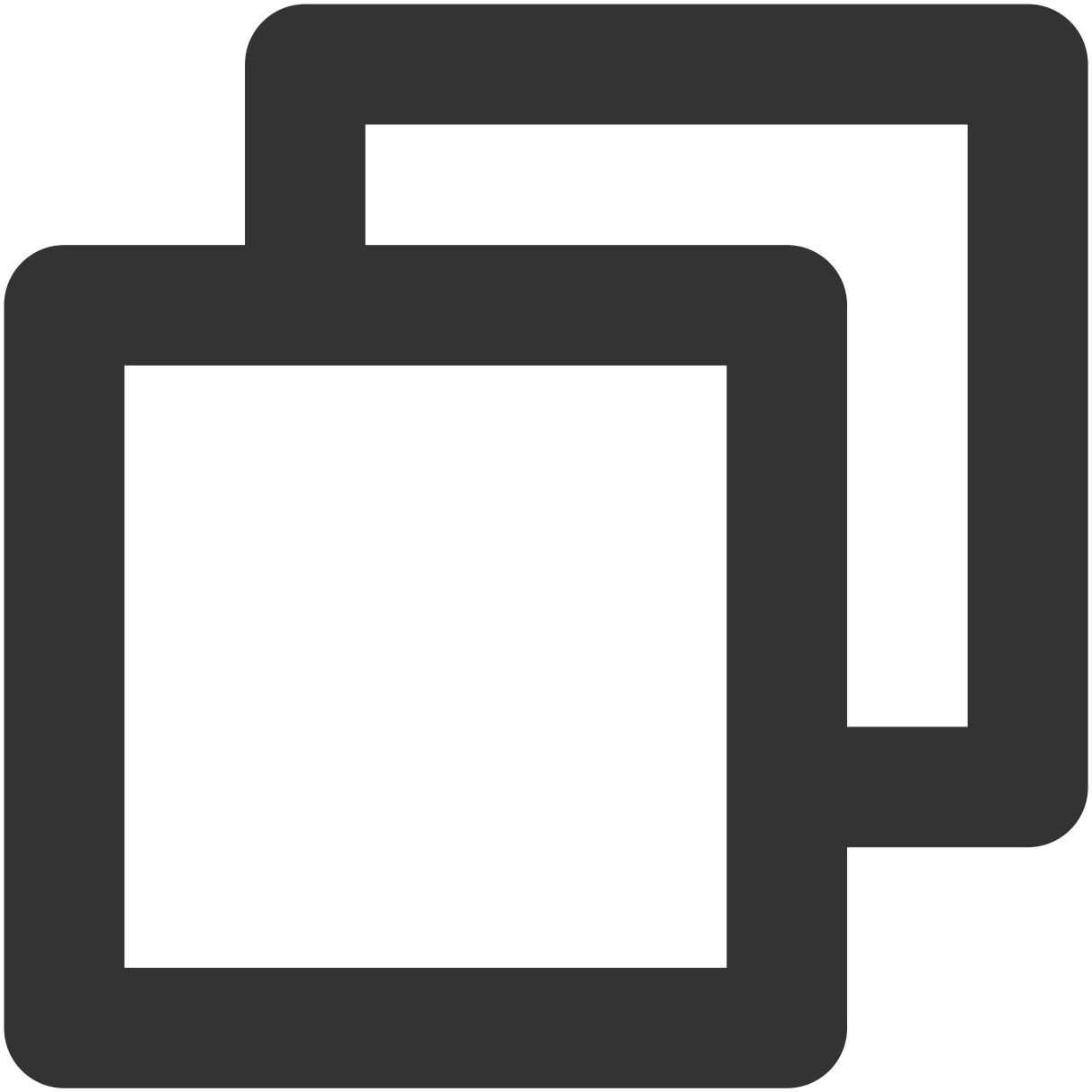
示例：



```
> SELECT shiftrightunsigned(4, 1);
 2
```

SIGN

函数语法：



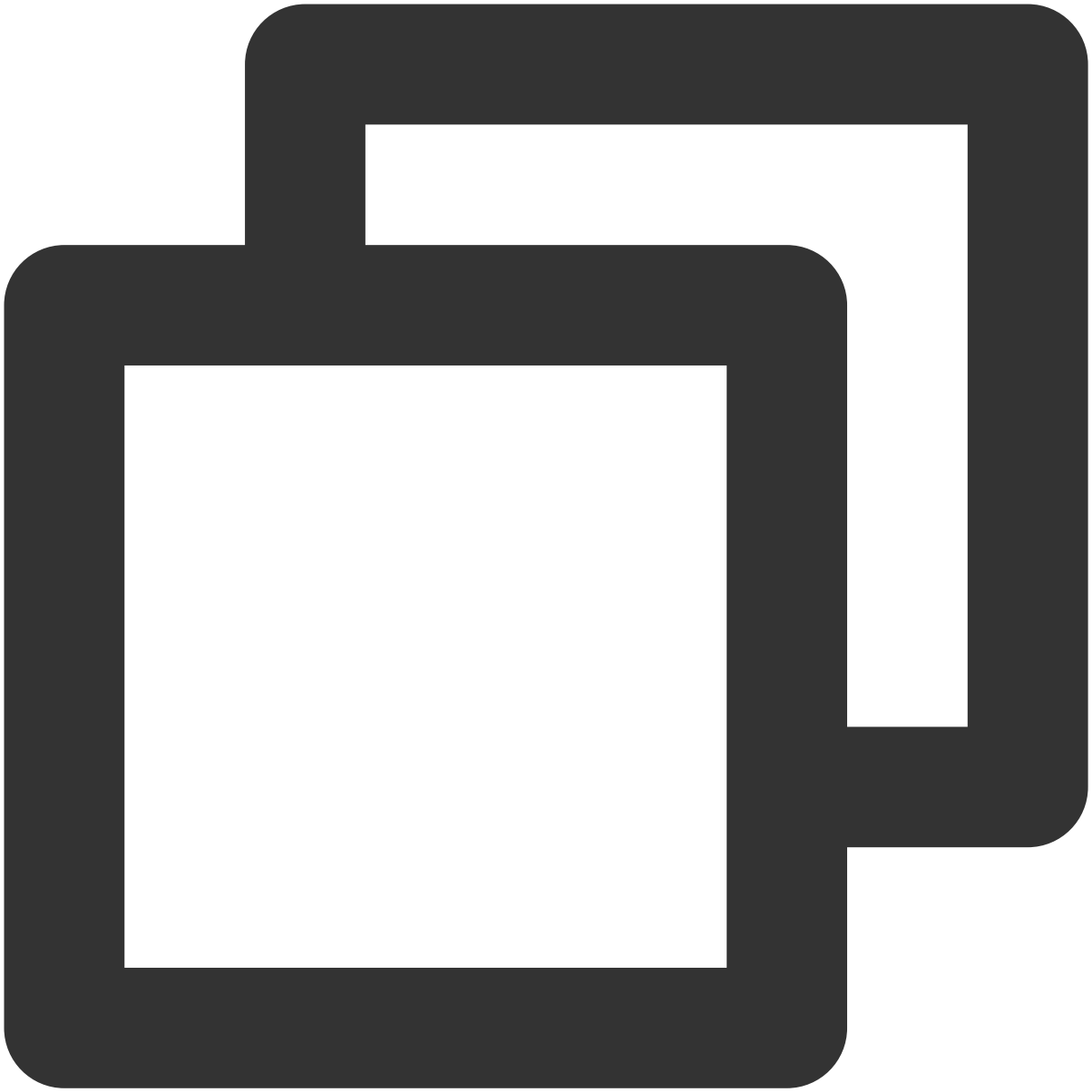
```
SIGN(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：当expr为负、0或正时，返回-1.0、0.0或1.0。

返回类型：double

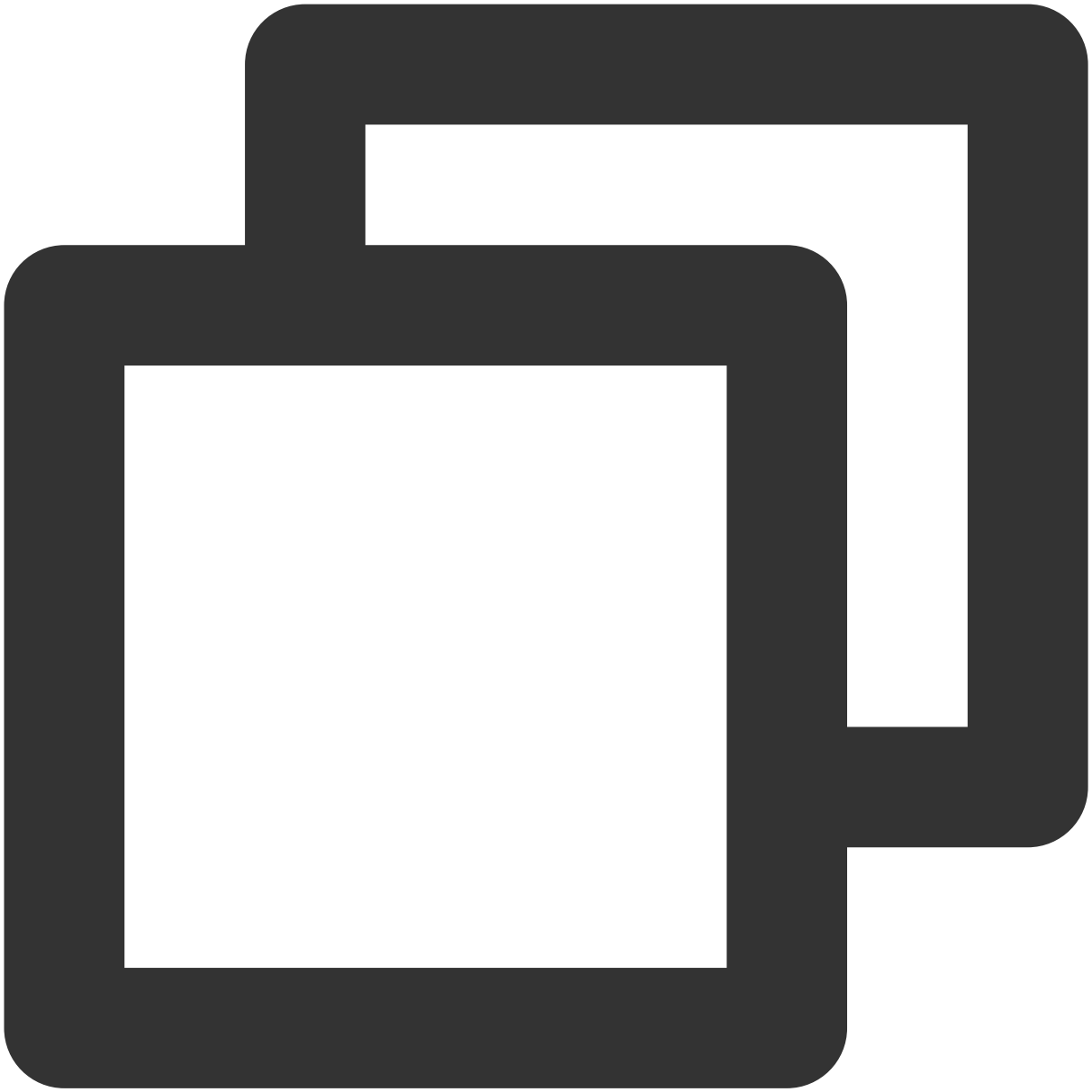
示例：



```
> SELECT sign(40);
 1.0
```

SIGNUM

函数语法：



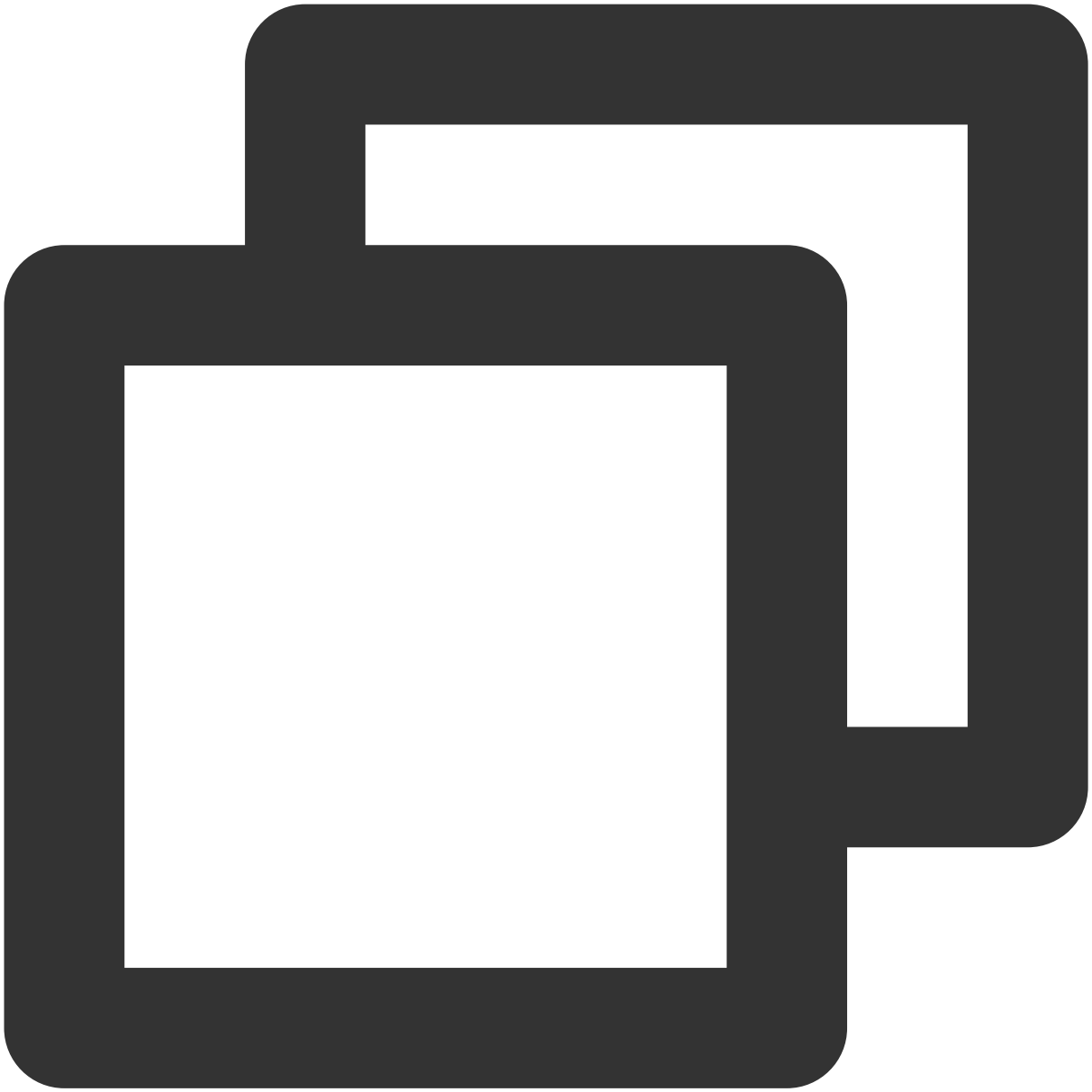
```
SIGNUM(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：当expr为负、0或正时，返回-1.0、0.0或1.0。

返回类型：double

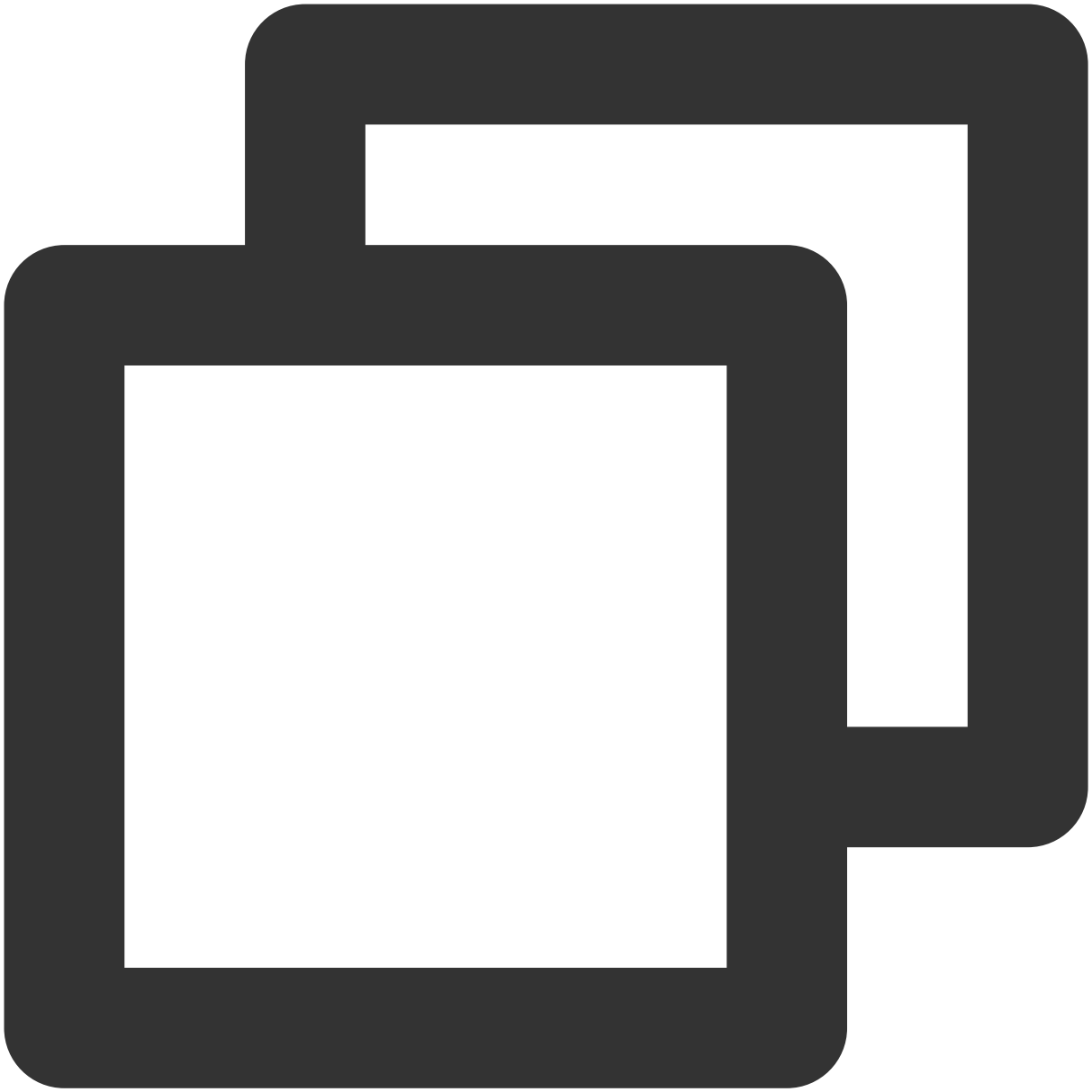
示例：



```
> SELECT signum(40);
 1.0
```

SIN

函数语法：



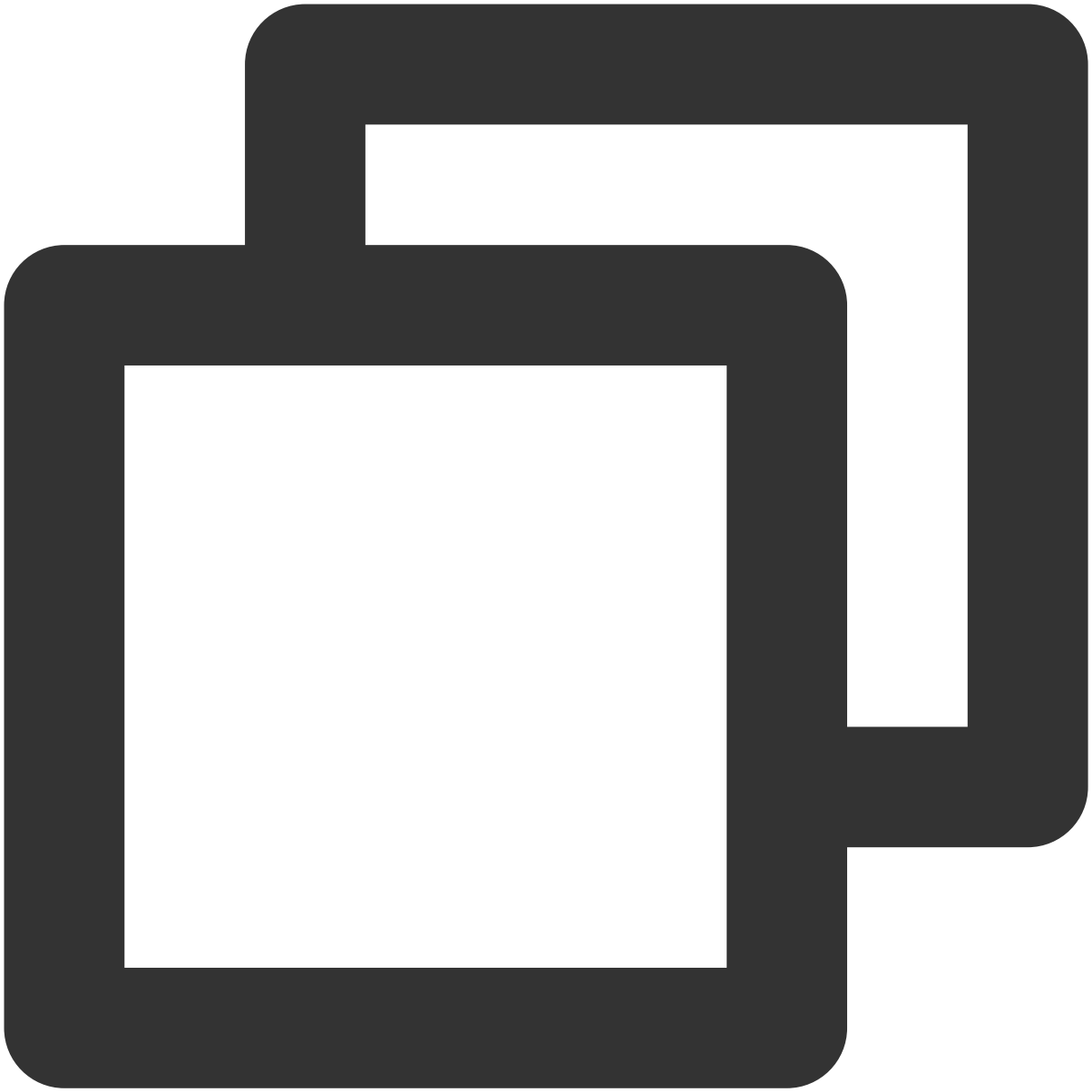
```
SIN(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的正弦值

返回类型：double

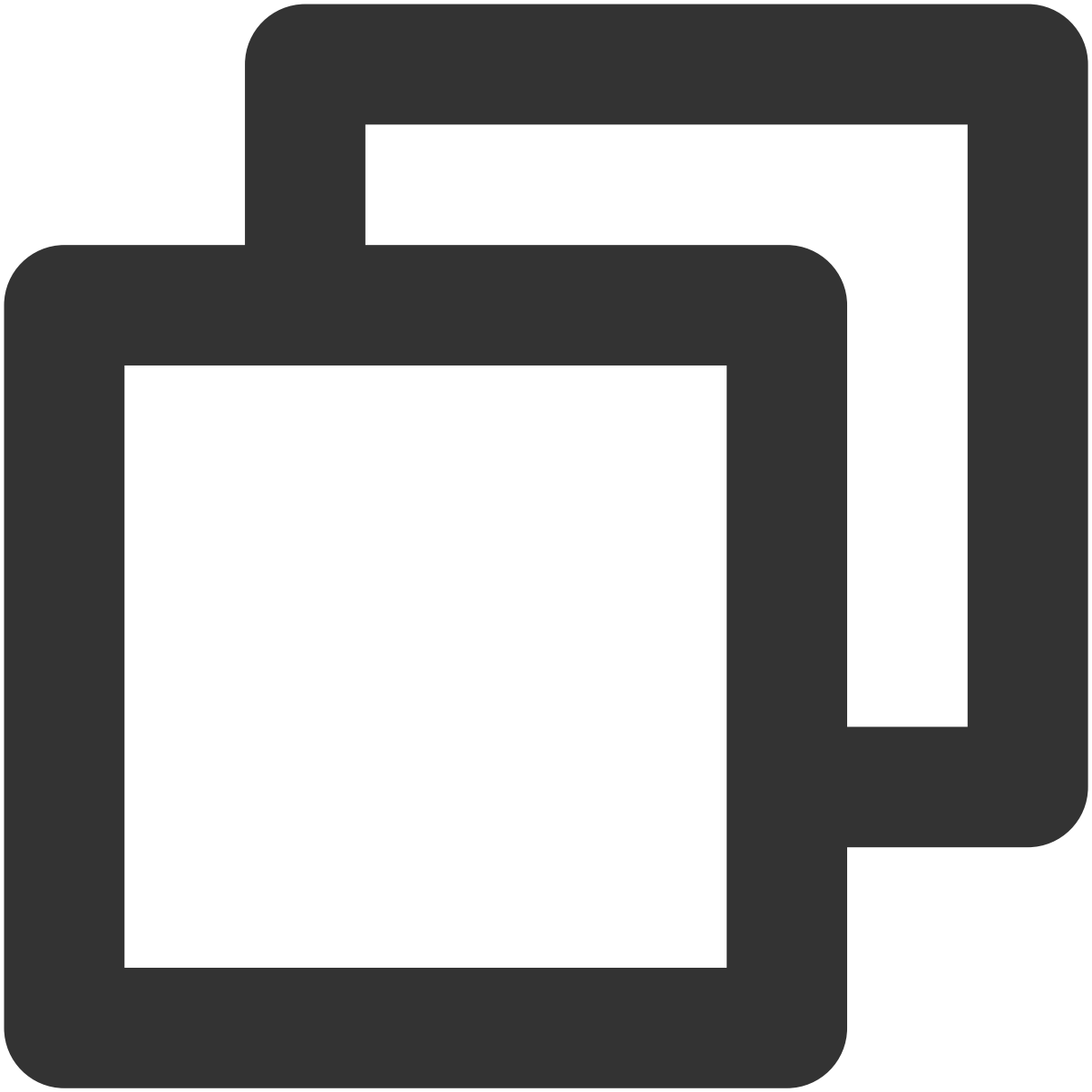
示例：



```
> SELECT sin(0);
 0.0
```

SINH

函数语法：



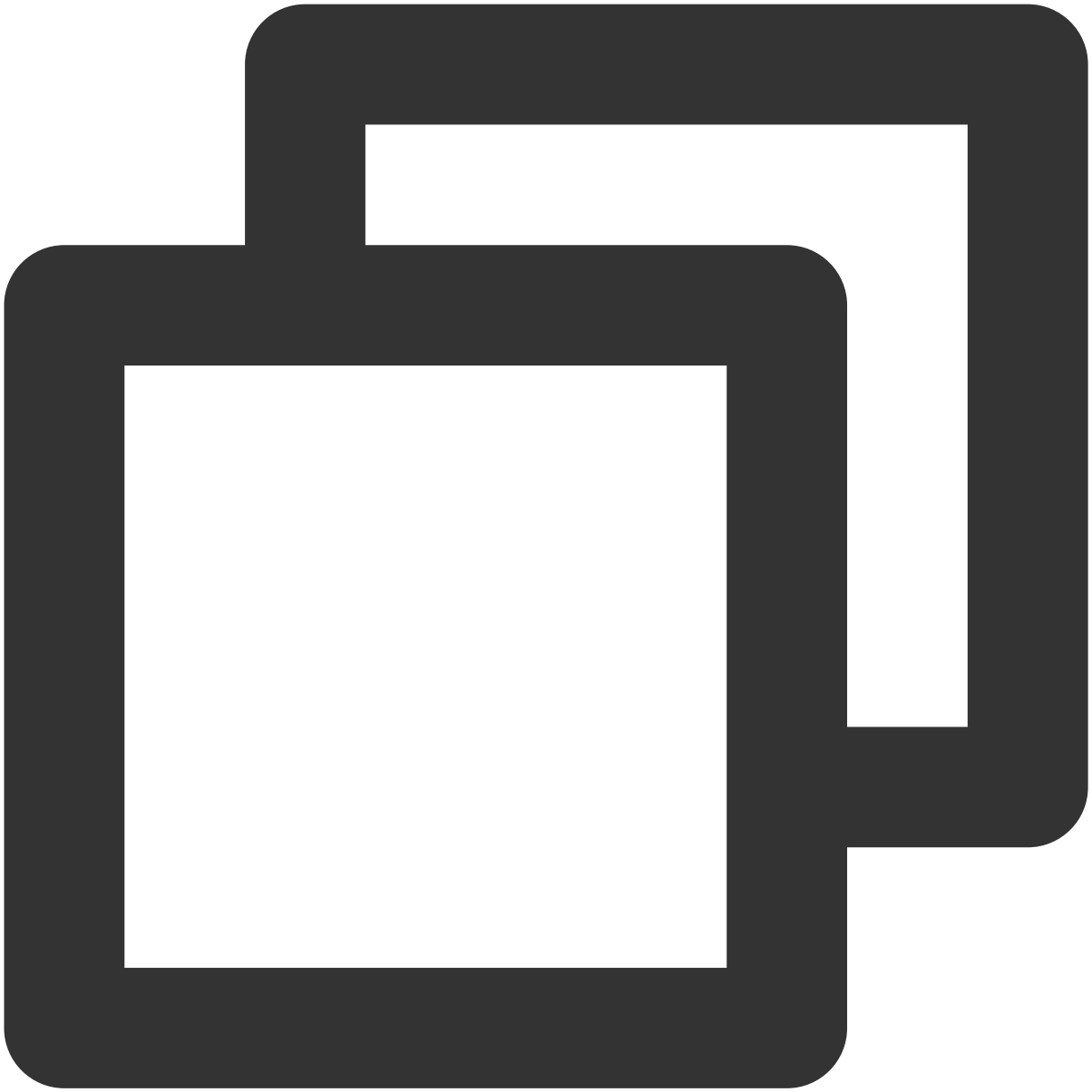
```
SINH(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的双曲正弦

返回类型：double

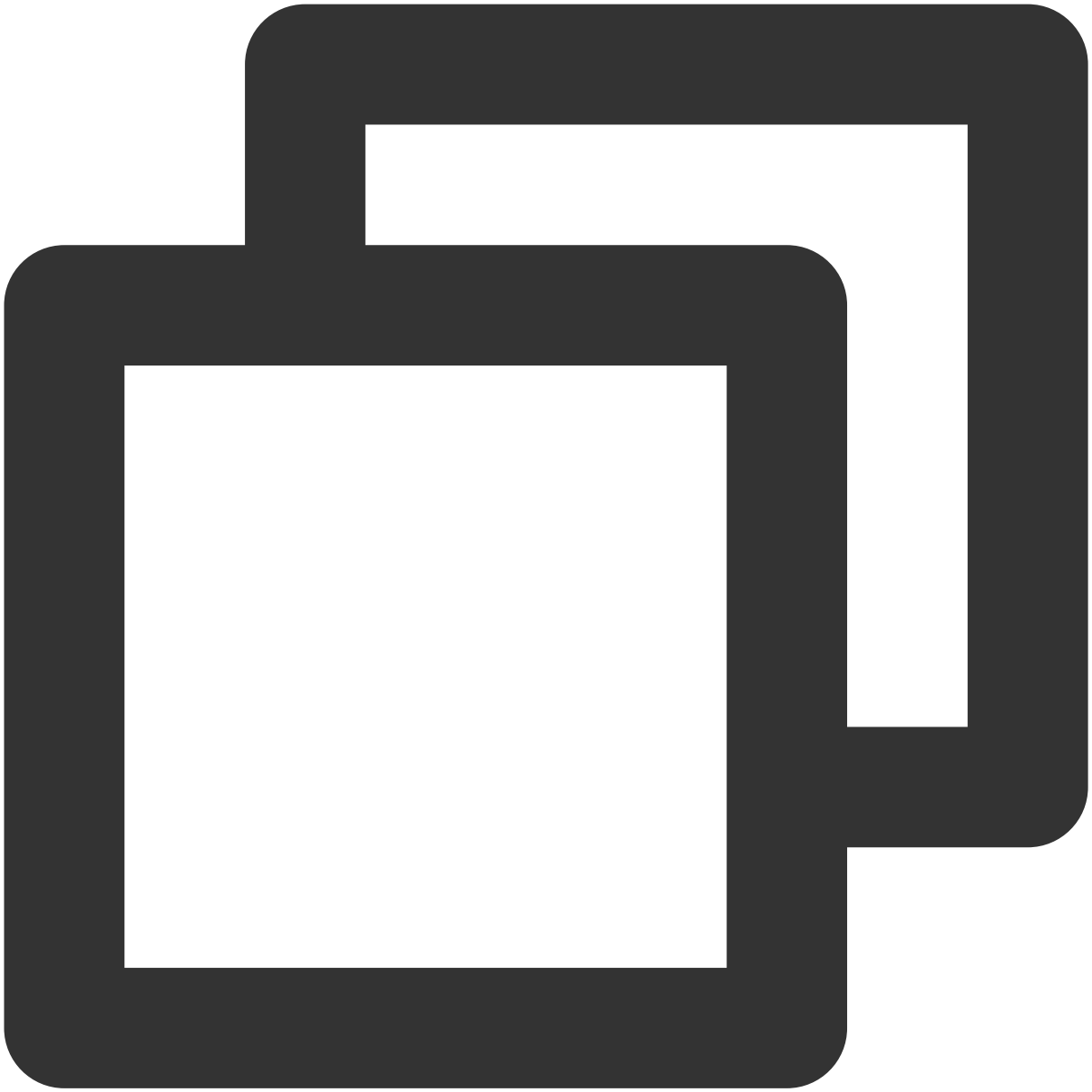
示例：



```
> SELECT sinh(0);
 0.0
```

SQRT

函数语法：



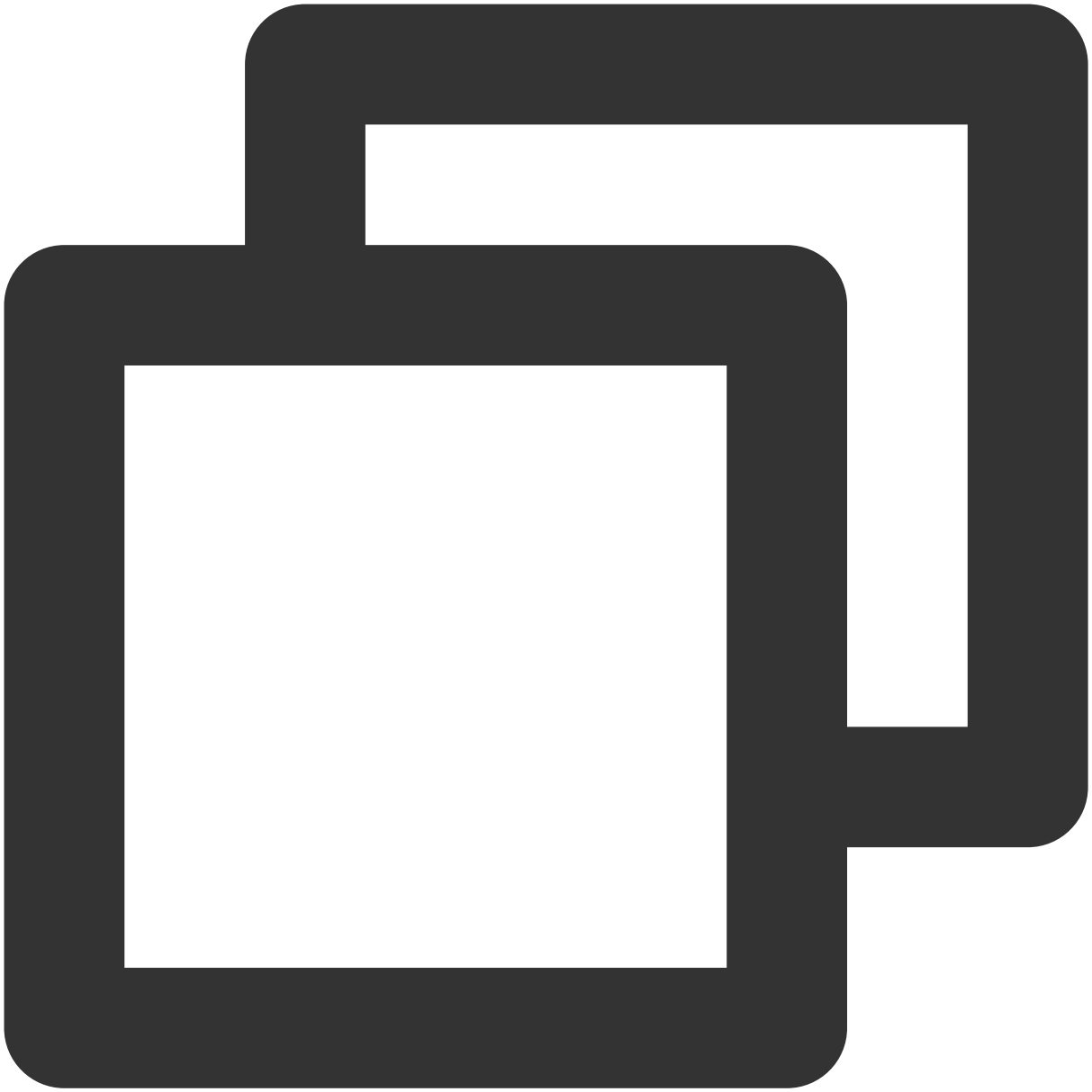
```
SQRT(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的平方根

返回类型：double

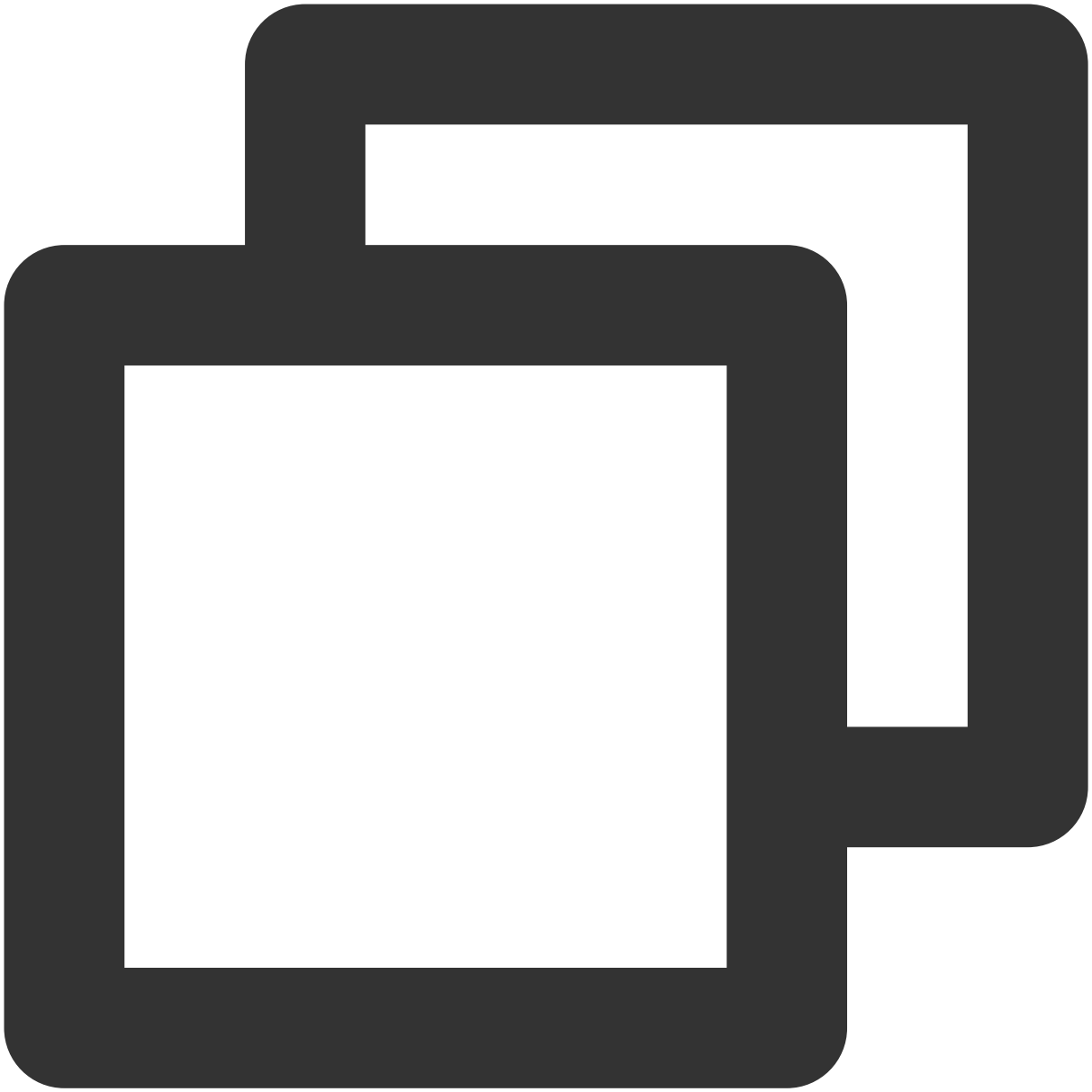
示例：



```
> SELECT sqrt(4);
 2.0
```

TAN

函数语法：



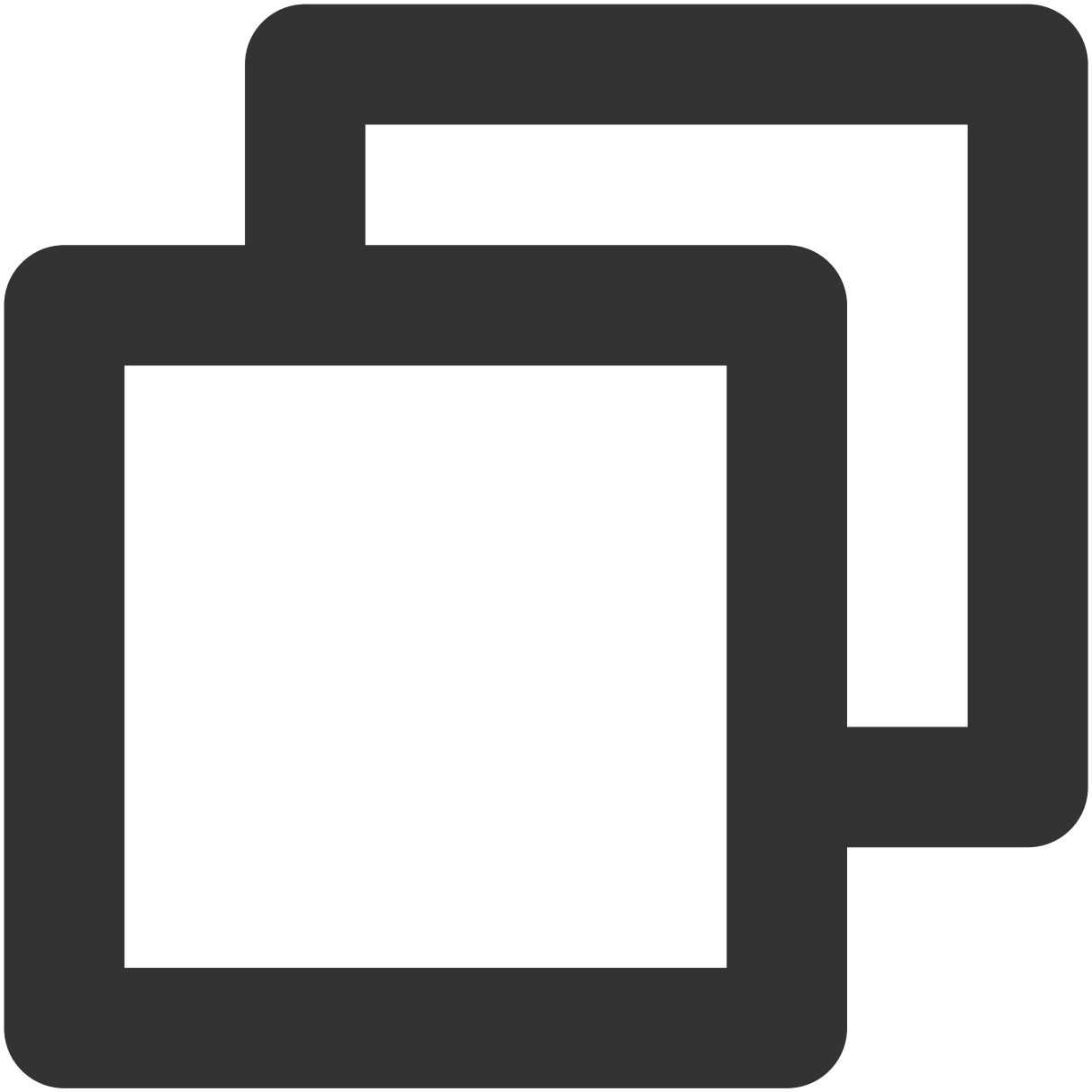
```
TAN(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的正切值

返回类型：double

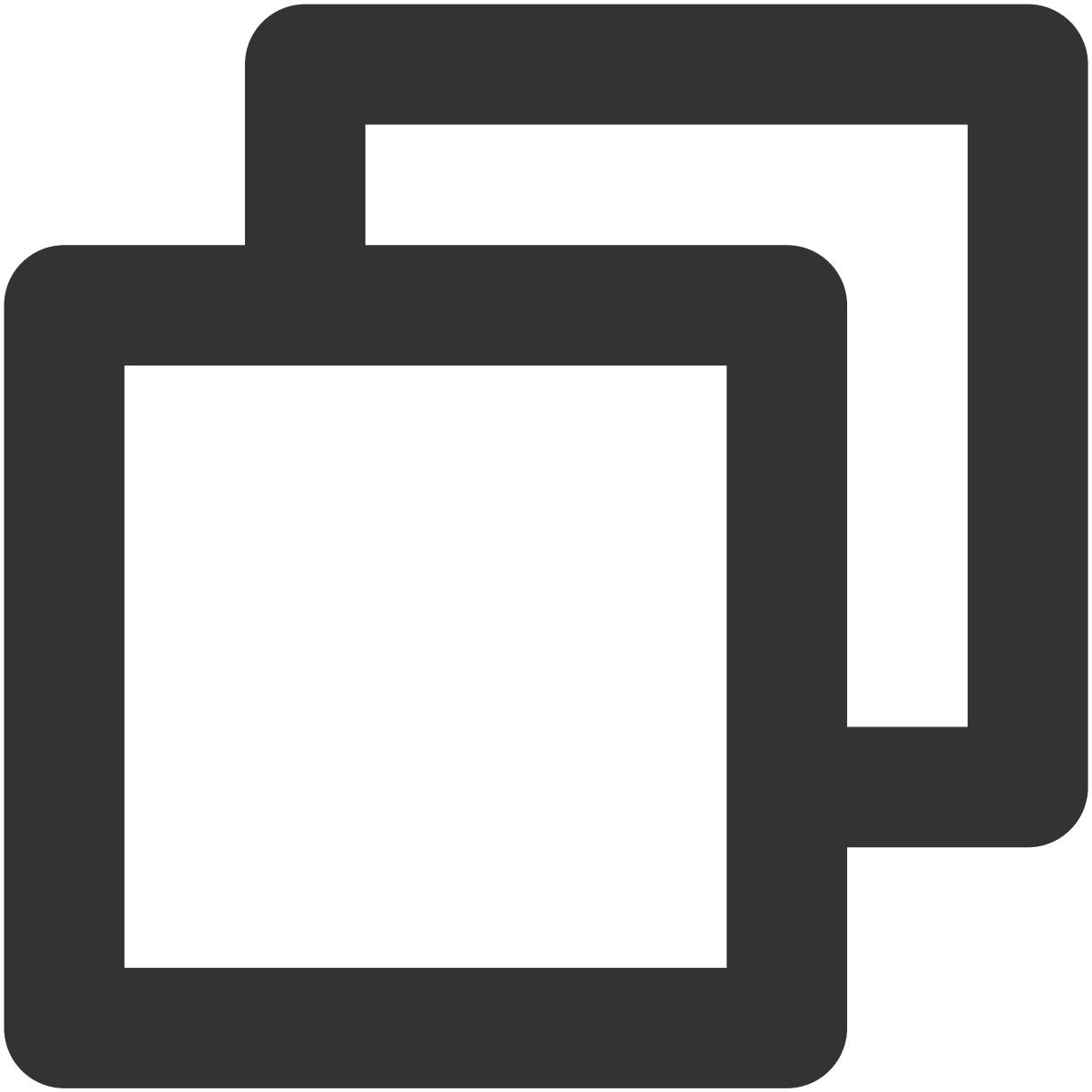
示例：



```
> SELECT tan(0);
 0.0
```

COT

函数语法：



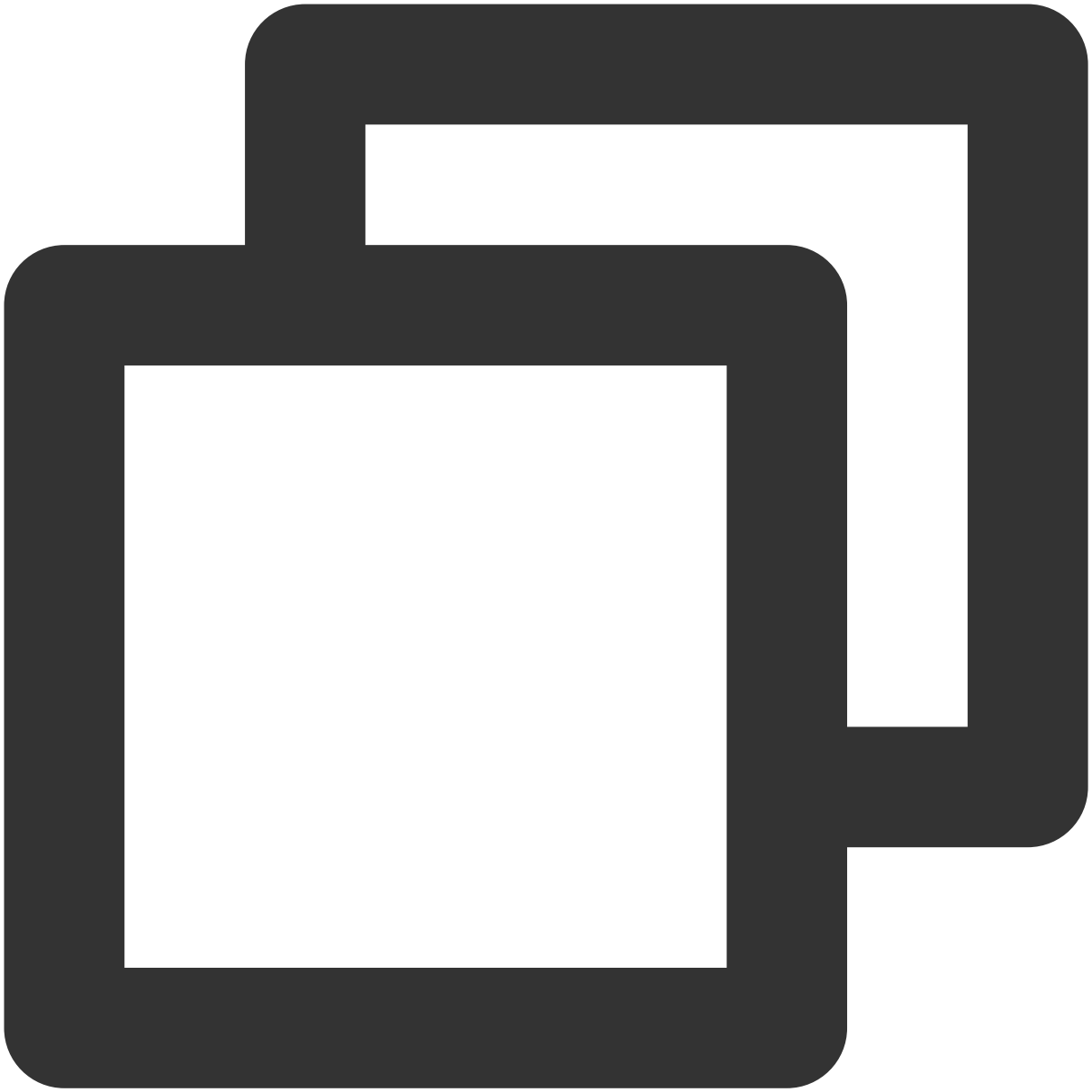
```
COT(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回表达式的余切

返回类型：double

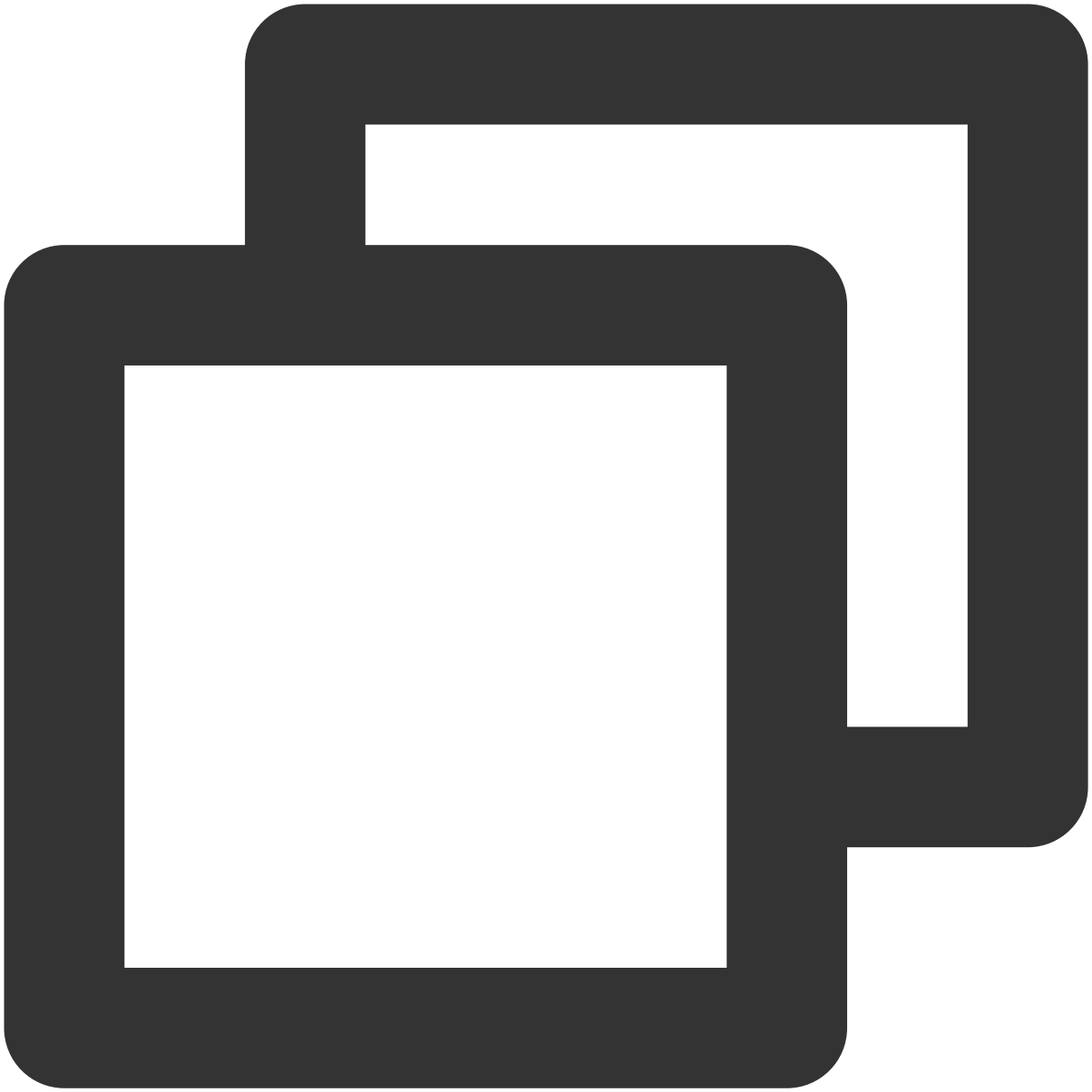
示例：



```
> SELECT cot(1);
 0.6420926159343306
```

TANH

函数语法：



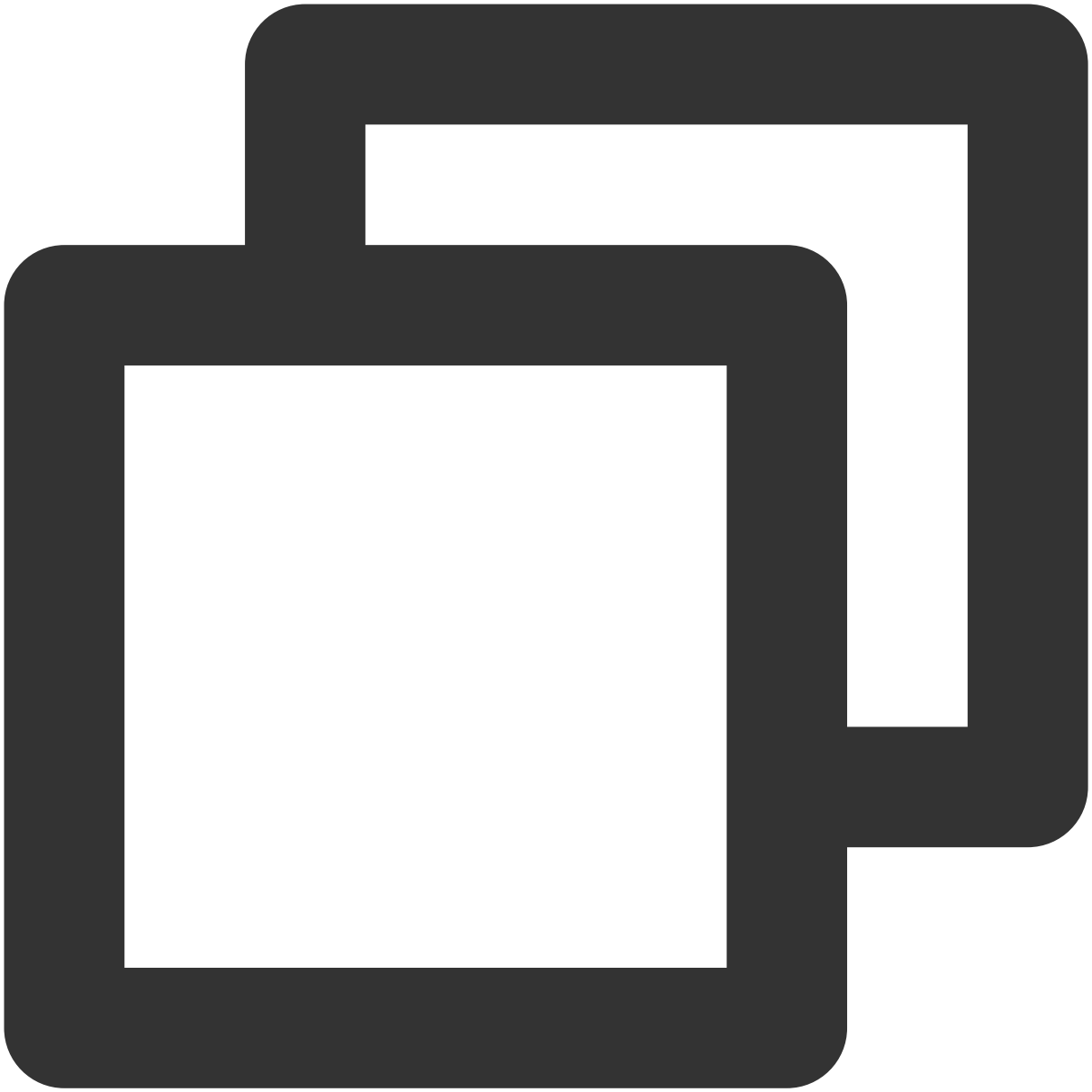
```
TANH(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的双曲正切

返回类型：double

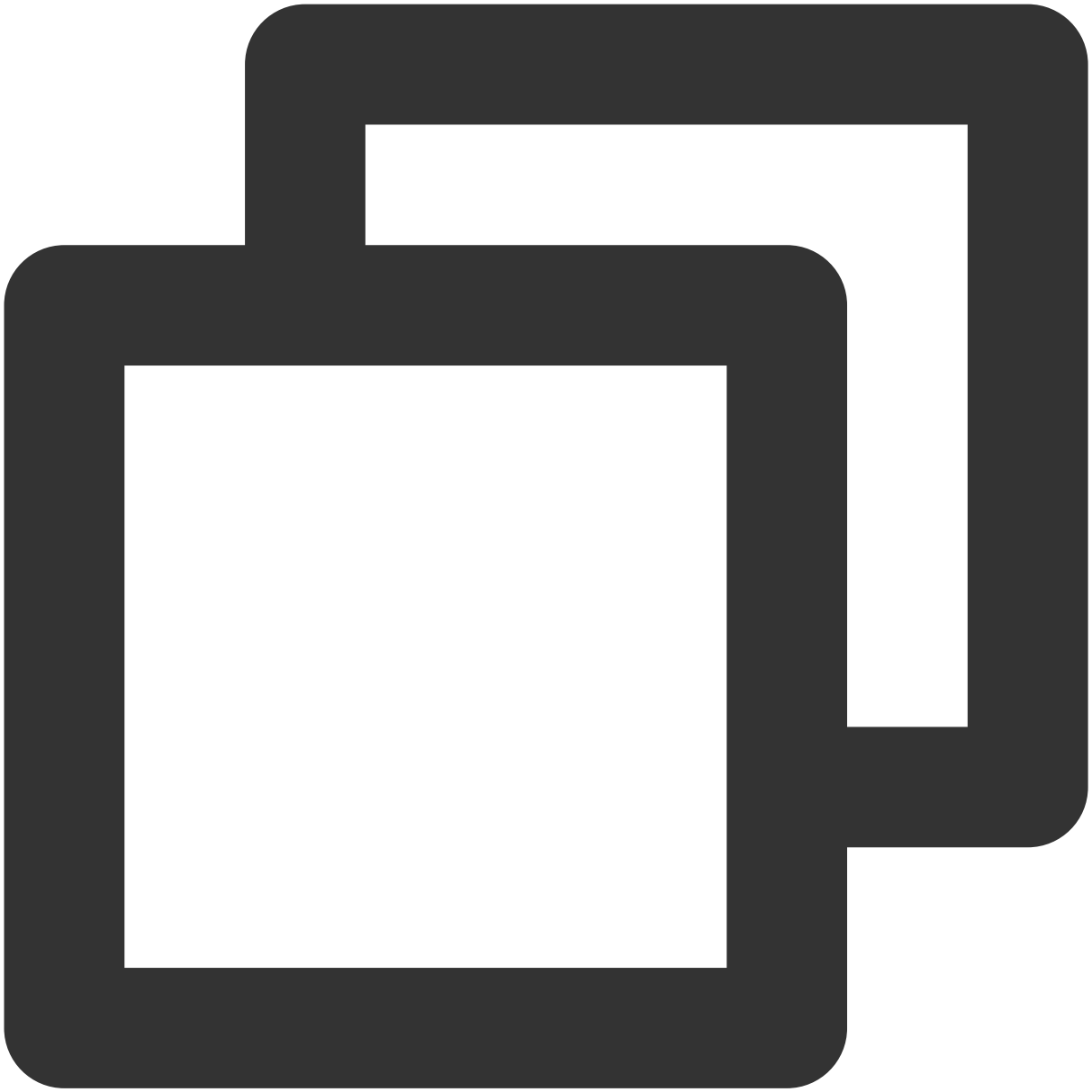
示例：



```
> SELECT tanh(0);
 0.0
```

WIDTH_BUCKET

函数语法：



```
WIDTH_BUCKET(<value> integer|double|decimal, <min> integer|double|decimal, <max> in
```

支持引擎：SparkSQL、Presto

使用说明：从min到max的等分为num_backet个分组，返回value落入的分组编号

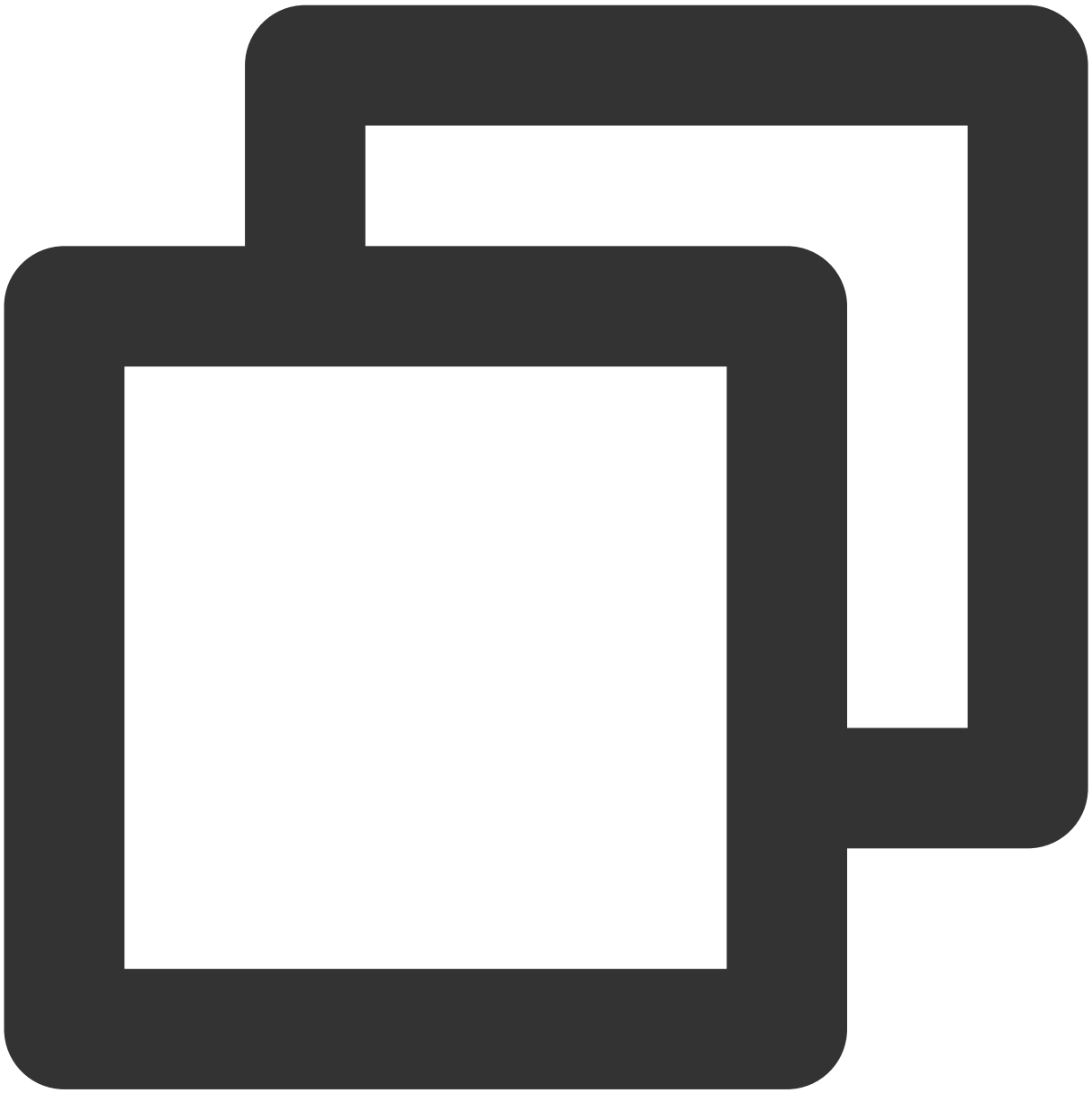
返回类型：integer

示例：

```
> SELECT width_bucket(5.3, 0.2, 10.6, 5);
 3
> SELECT width_bucket(-2.1, 1.3, 3.4, 3);
 0
> SELECT width_bucket(8.1, 0.0, 5.7, 4);
 5
> SELECT width_bucket(-0.9, 5.2, 0.5, 2);
 3
```

TRY_ADD

函数语法：



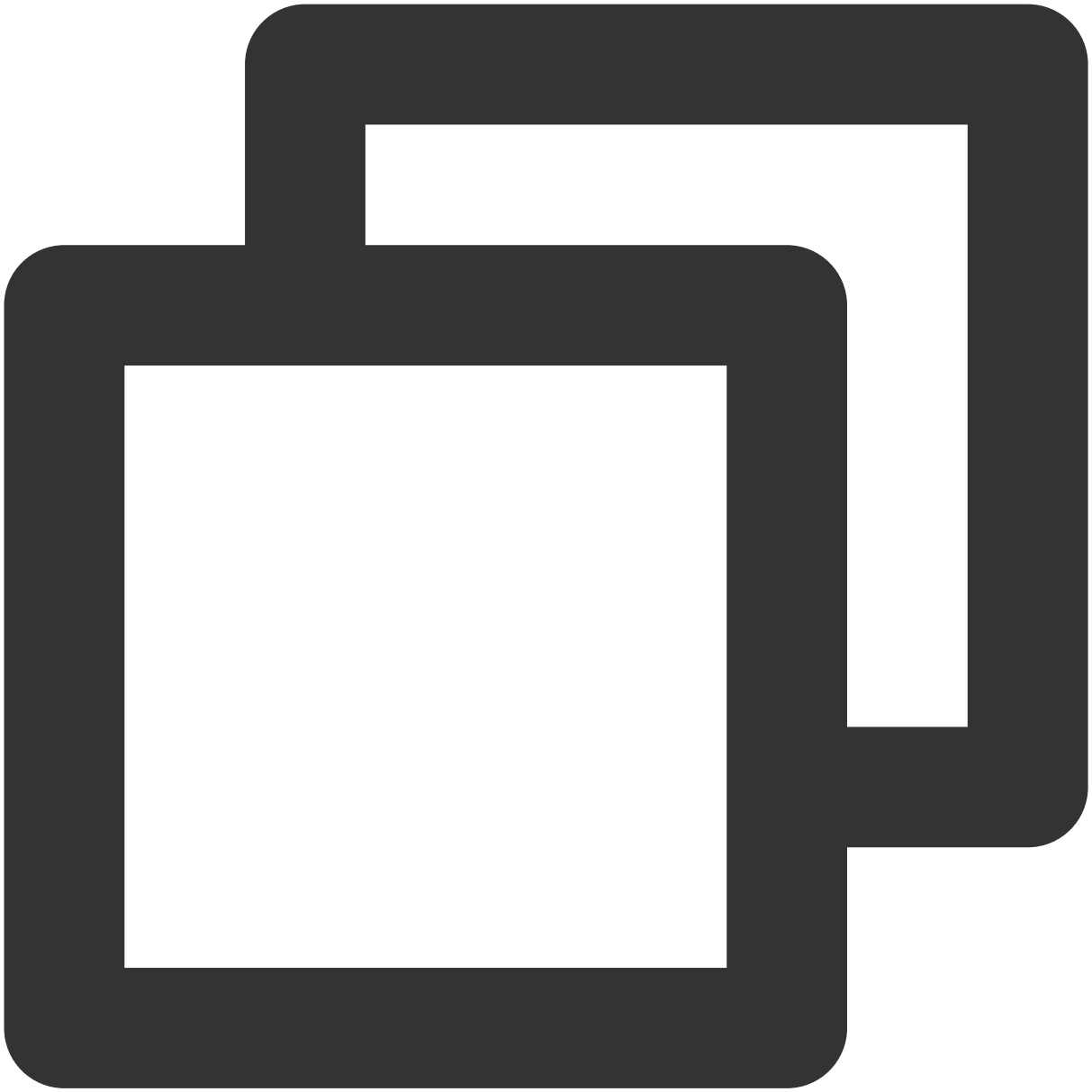
```
TRY_ADD(<expr1> integer|double|decimal|date|timestamp, <expr2> integer|double|decim
```

支持引擎：SparkSQL、Presto

使用说明：返回expr1和expr2的总和，溢出时结果为null

返回类型：integer|double|decimal|date|timestamp

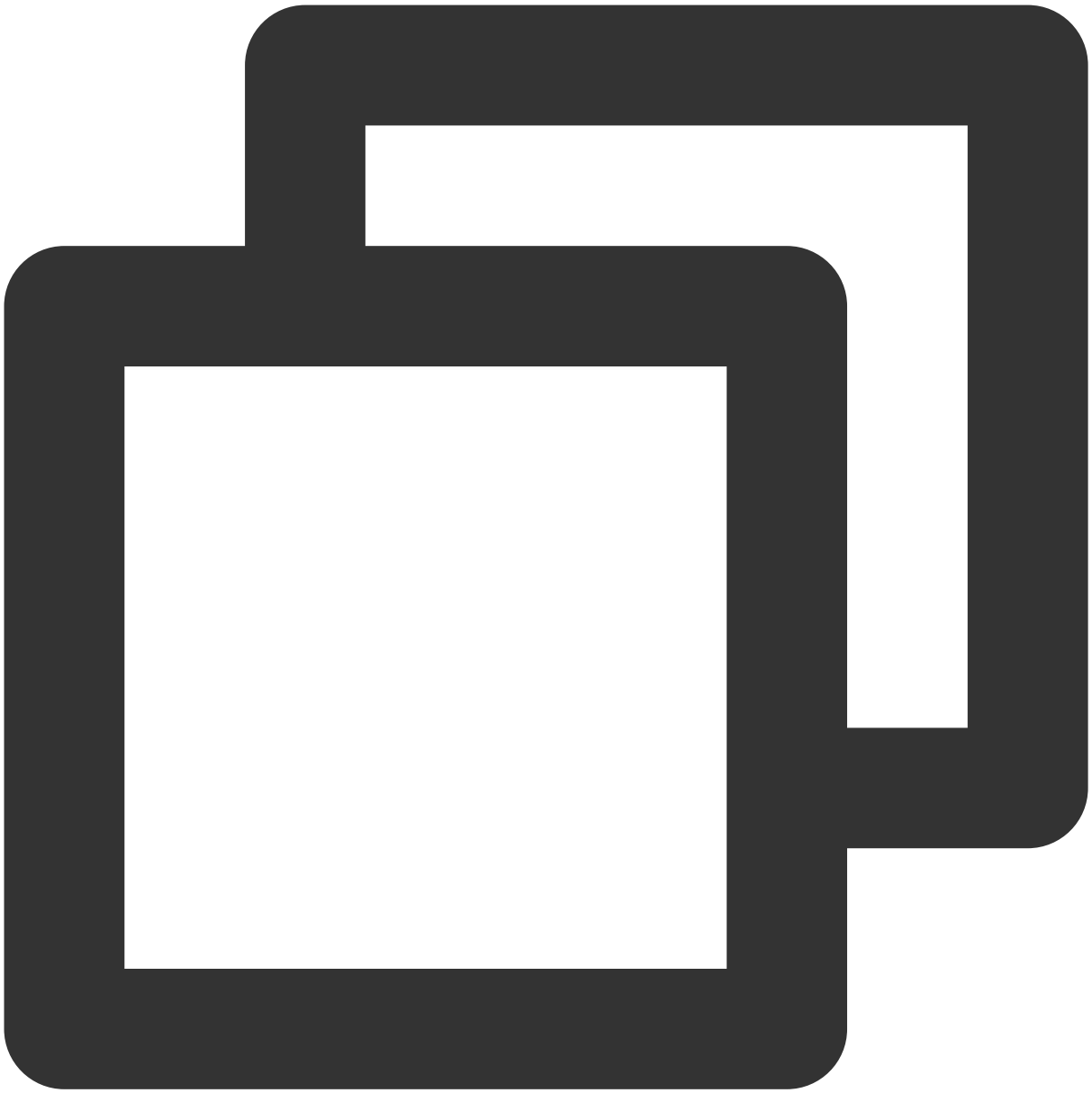
示例：



```
> SELECT try_add(1, 2);
 3
> SELECT try_add(2147483647, 1);
 NULL
> SELECT try_add(date'2021-01-01', 1);
 2021-01-02
```

TRY_DIVIDE

函数语法：



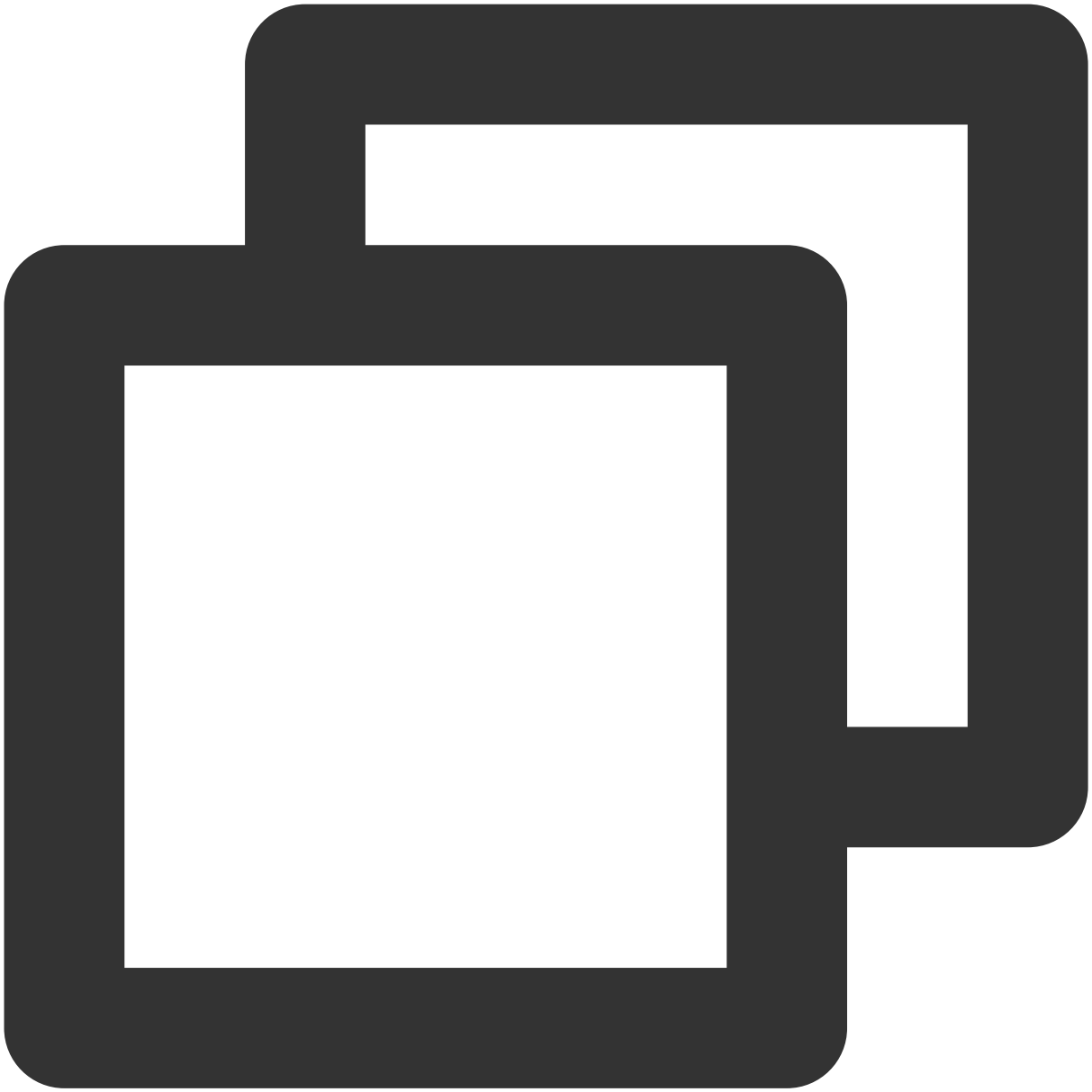
```
TRY_DIVIDE(<dividend> integer|double|decimal, <divisor> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回dividend/divisor

返回类型：double

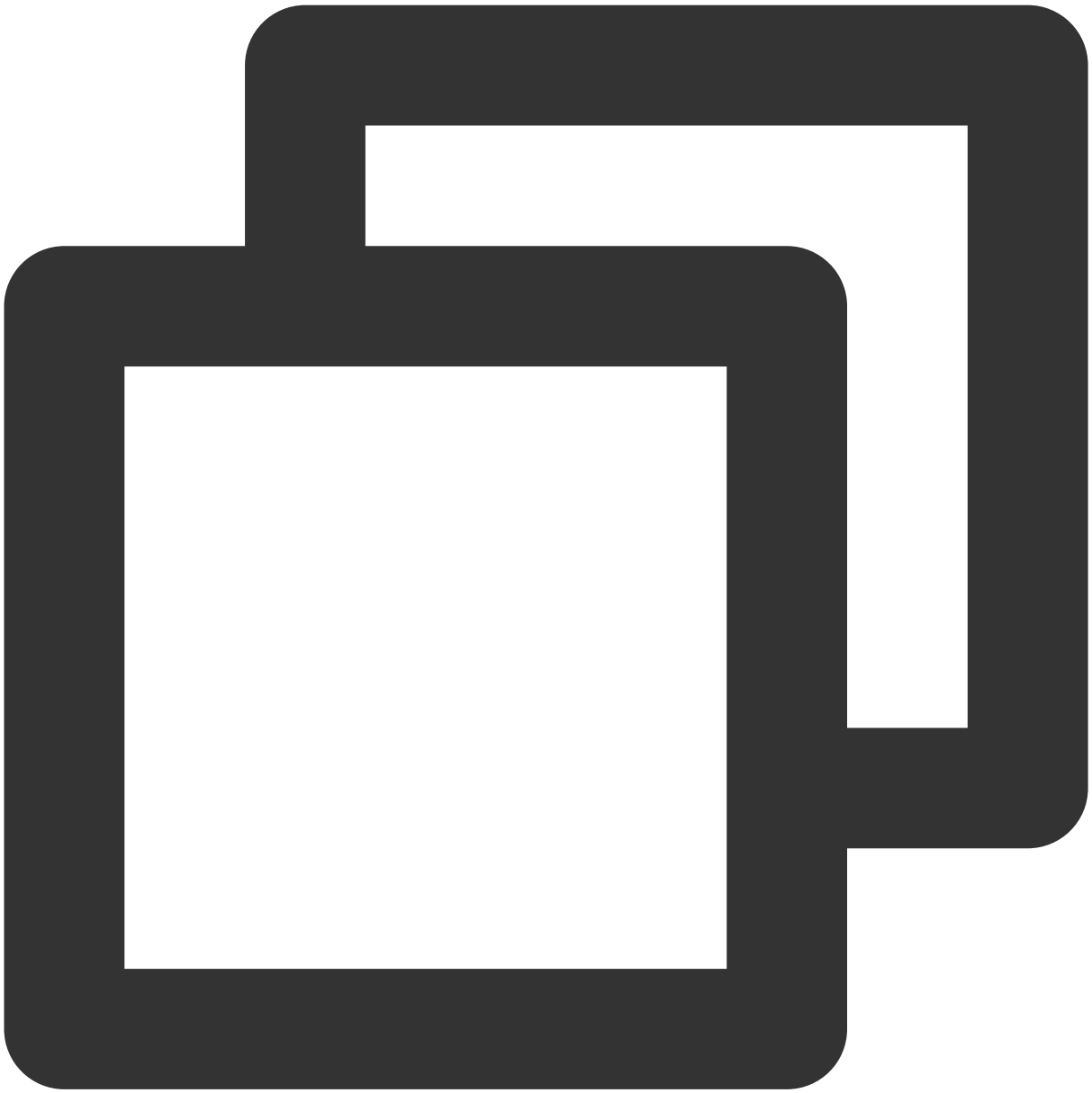
示例：



```
> SELECT try_divide(3, 2);
 1.5
> SELECT try_divide(2L, 2L);
 1.0
> SELECT try_divide(1, 0);
 NULL
```

RAND

函数语法：



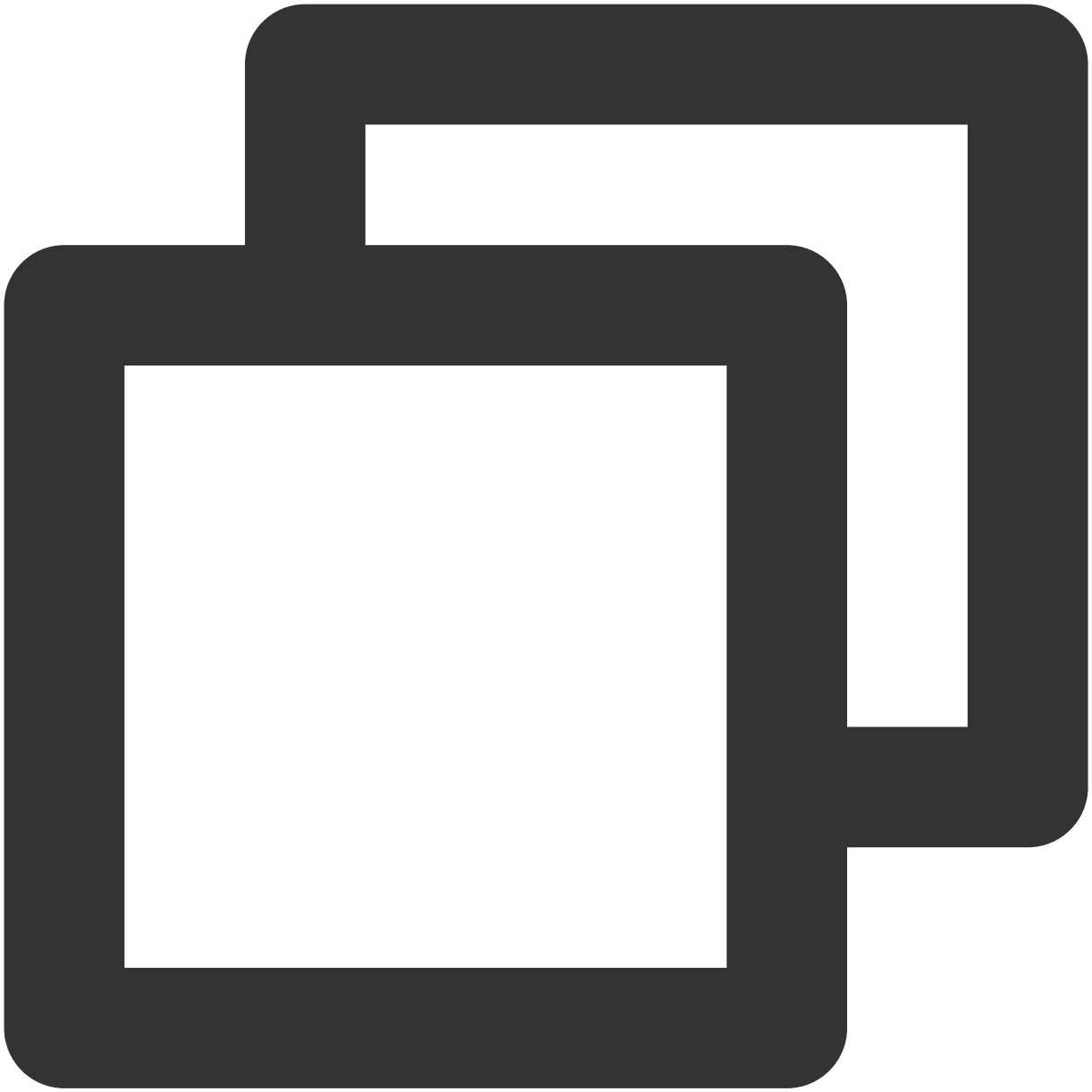
```
RAND([<seed> integer])
```

支持引擎：SparkSQL、Presto

使用说明：返回一个随机值，该随机值为[0，1]中的值独立且均匀分布。

返回类型：double

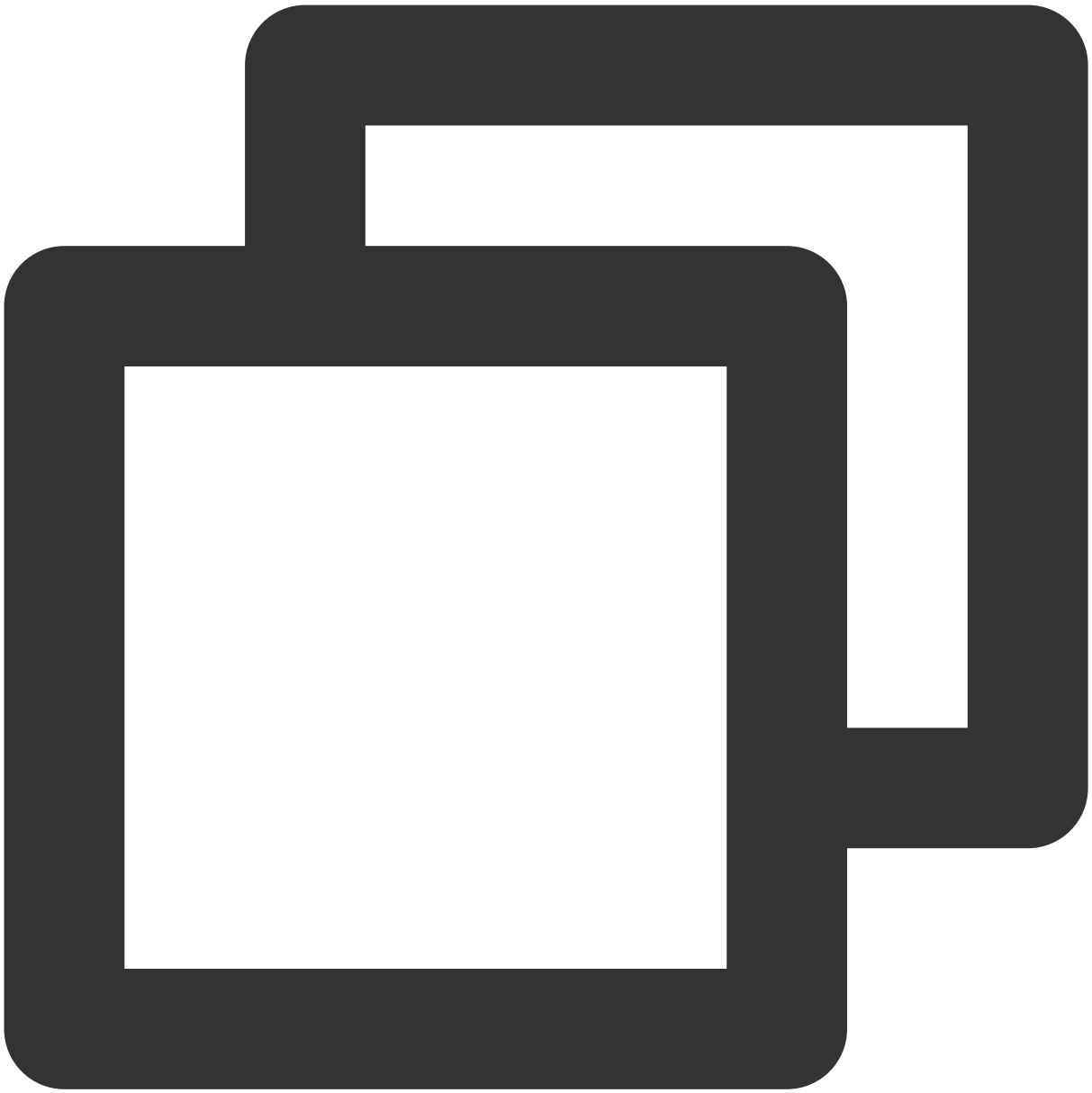
示例：



```
> SELECT rand();
 0.9629742951434543
> SELECT rand(0);
 0.7604953758285915
> SELECT rand(null);
 0.7604953758285915
```

RANDOM

函数语法：



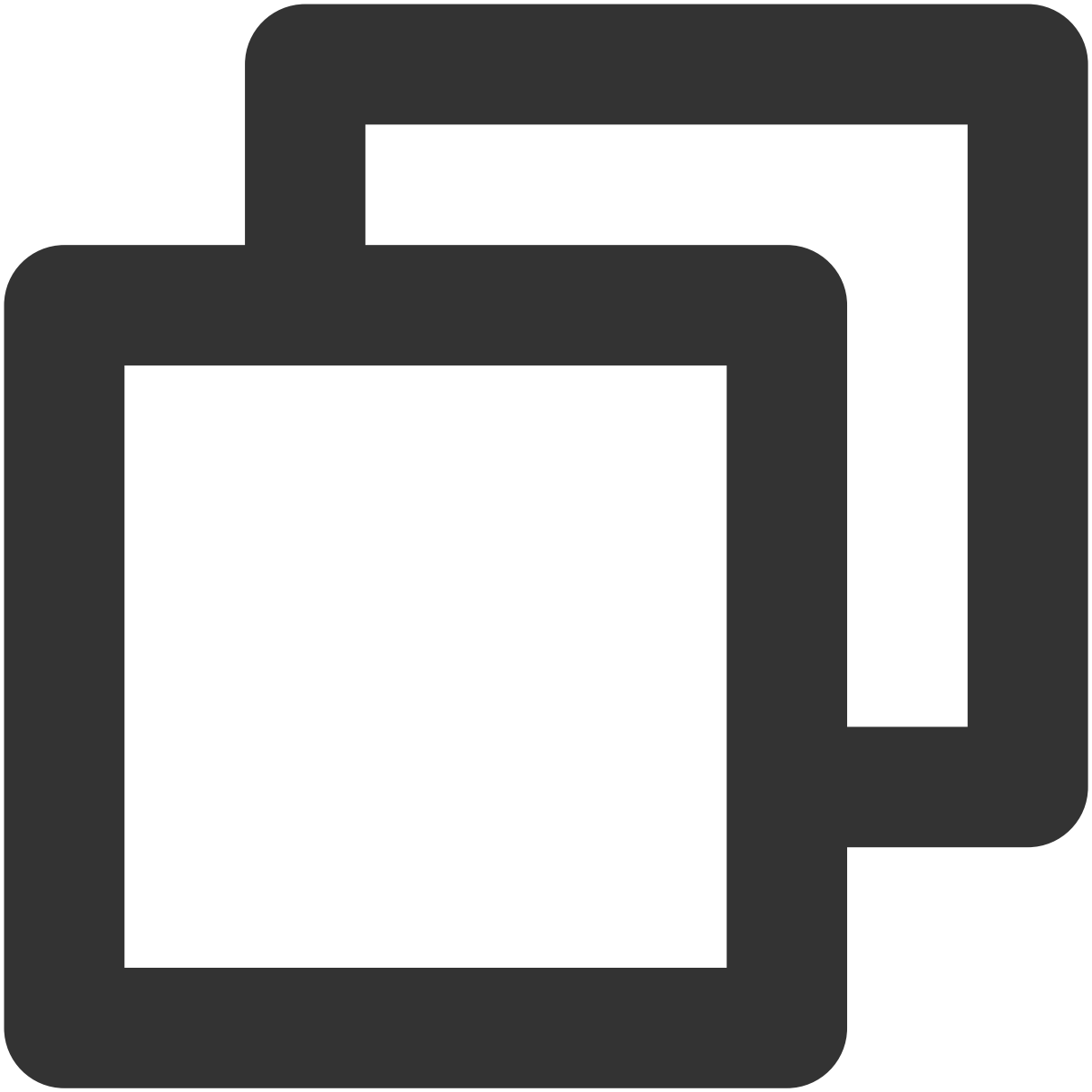
```
RANDOM([<seed> integer])
```

支持引擎：SparkSQL、Presto

使用说明：返回一个随机值，该随机值为[0，1]中的值独立且均匀分布。

返回类型：double

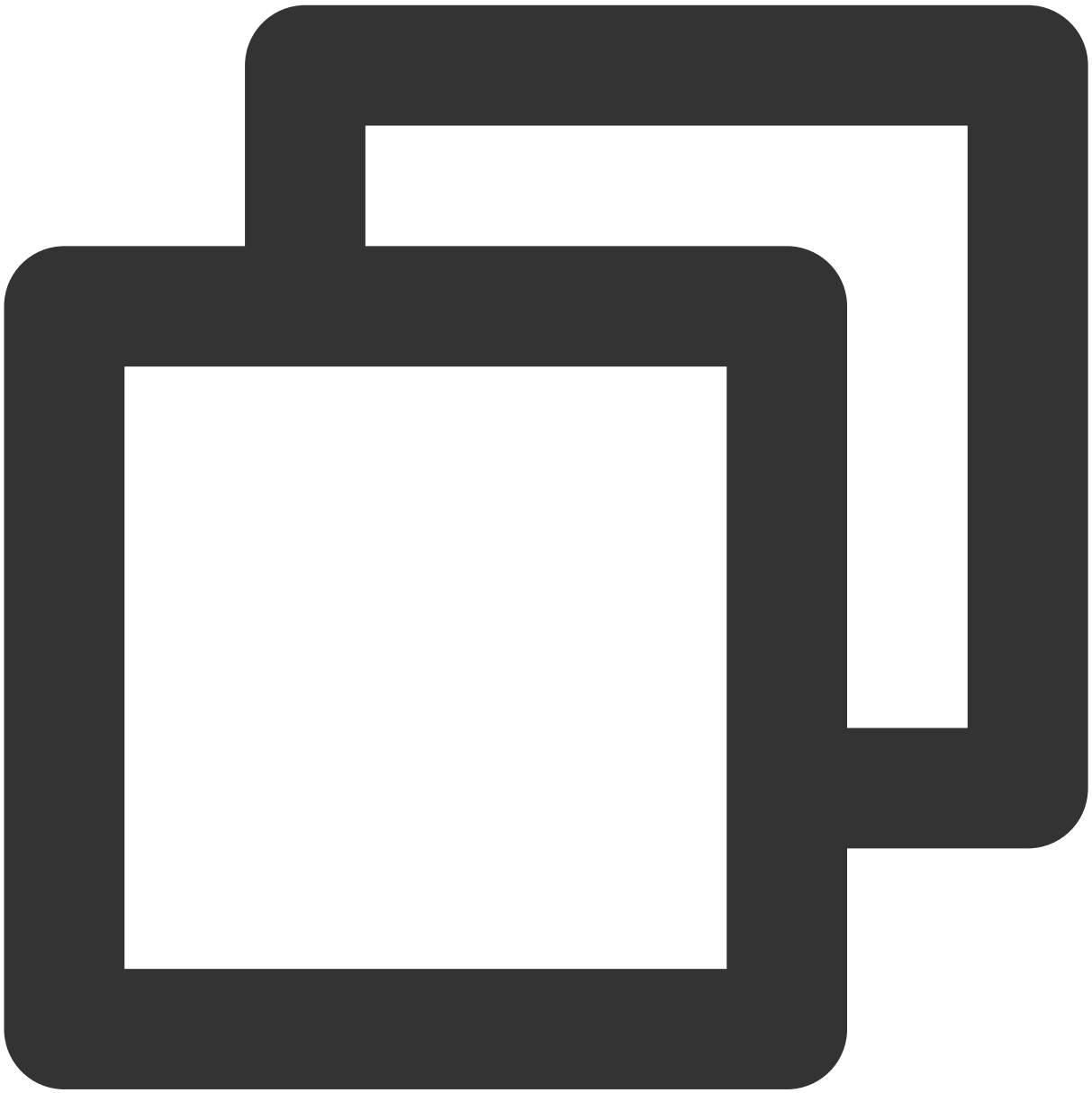
示例：



```
> SELECT rand();
 0.9629742951434543
> SELECT rand(0);
 0.7604953758285915
> SELECT rand(null);
 0.7604953758285915
```

RANDN

函数语法：



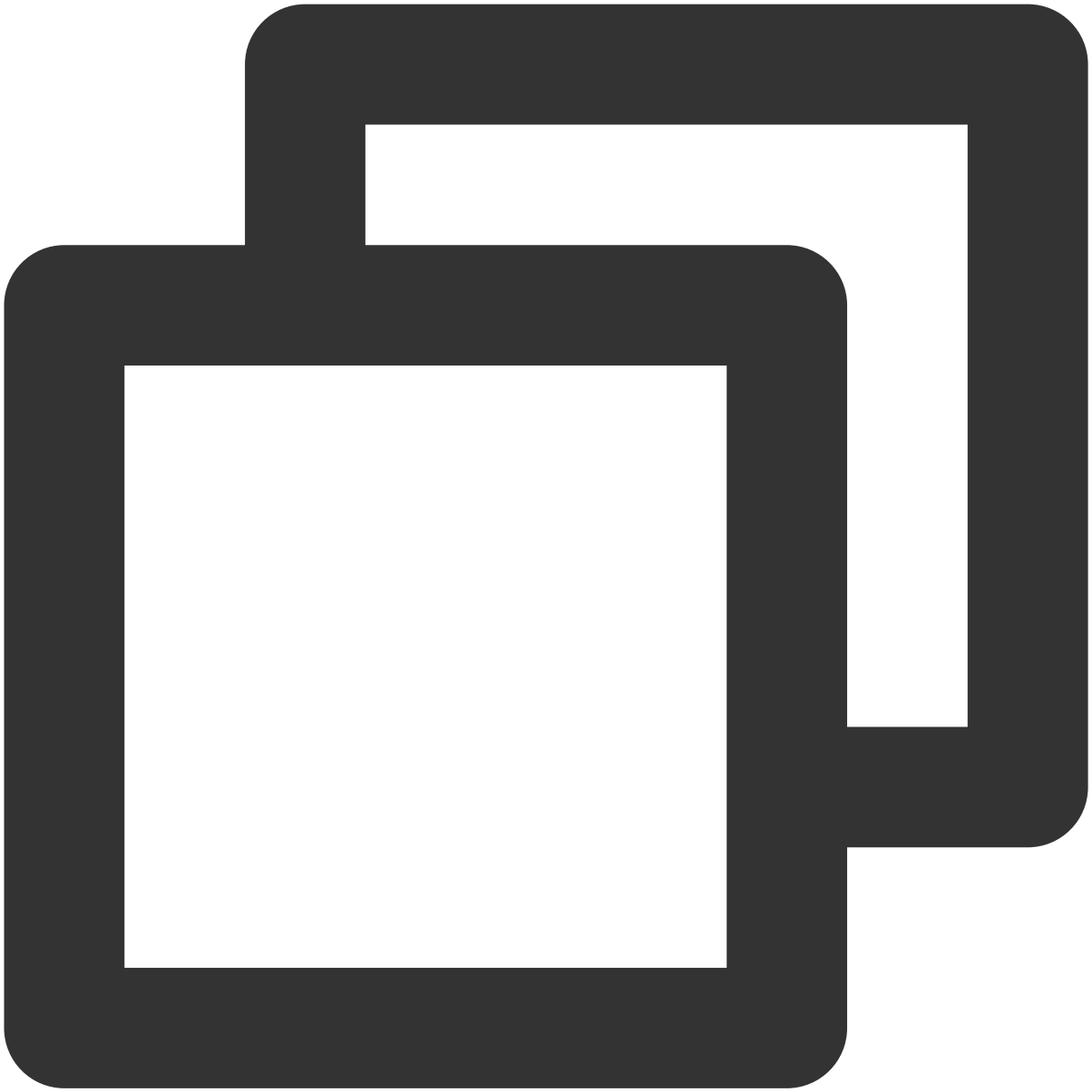
```
RANDN([<seed> integer])
```

支持引擎：SparkSQL、Presto

使用说明：返回一个随机值，其中包含从标准正态分布中提取的独立和同分布（i.i.d.）值。该函数在SPARKSQL与PRESTO中的实现方式不一致，使用相同的seed可能得到不同的结果。

返回类型：double

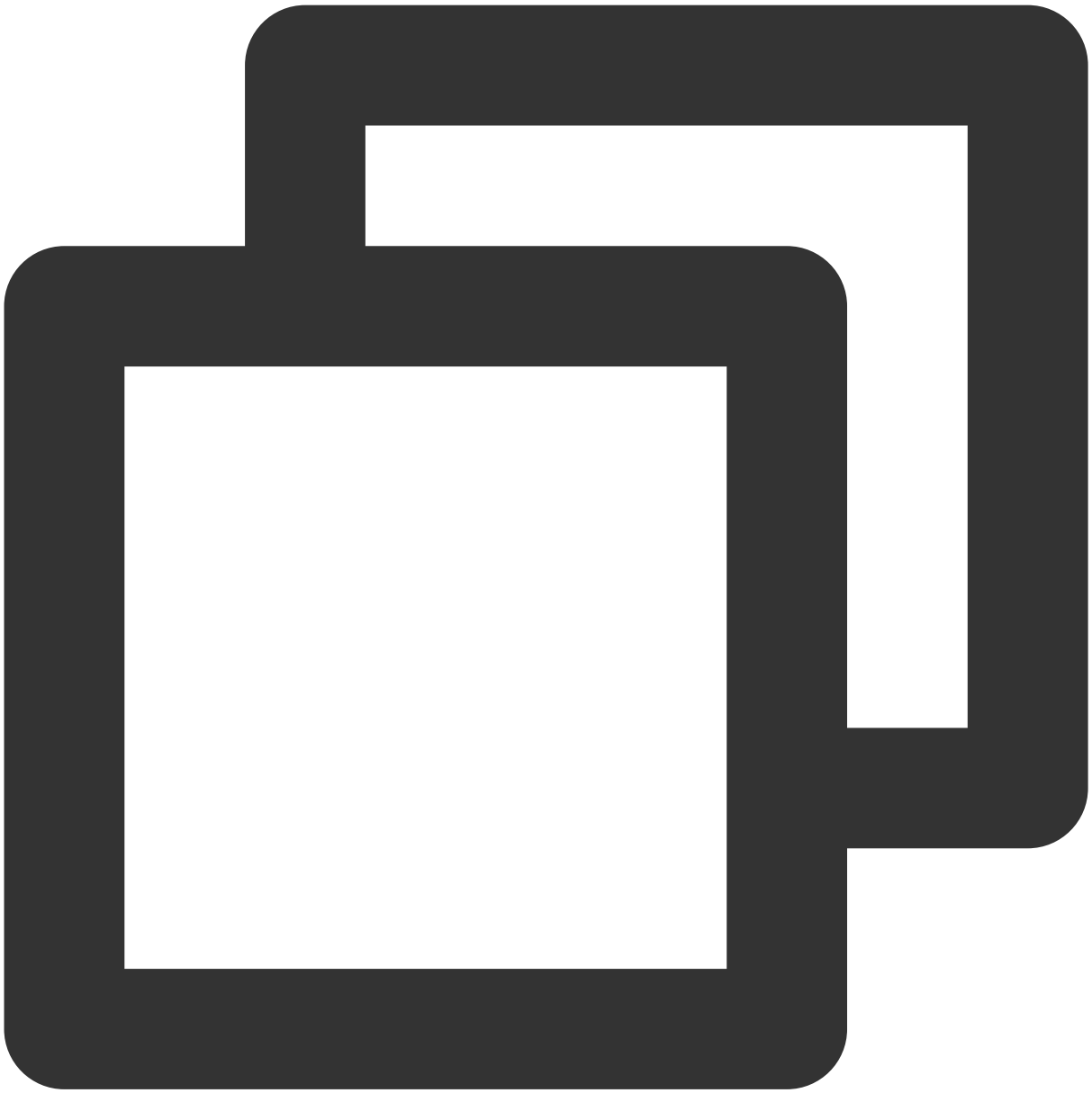
示例：



```
> SELECT randn();
 -0.3254147983080288
> SELECT randn(0);
 1.6034991609278433
> SELECT randn(null);
 1.6034991609278433
```

DIV

函数语法：



```
<expr1> DIV <expr2>
```

支持引擎：SparkSQL

使用说明：expr1除以expr2

返回类型：integer

示例：

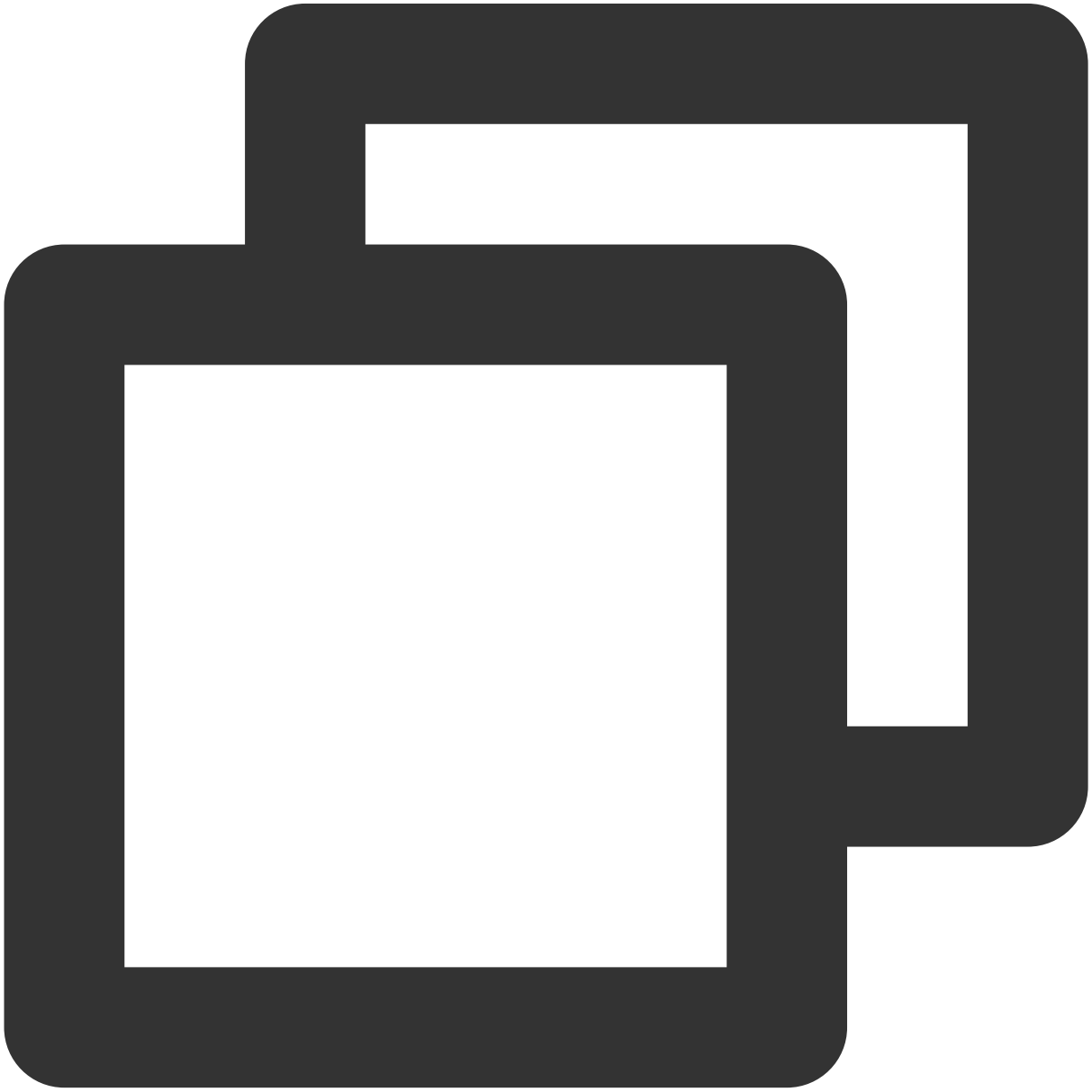
```
> SELECT 3 div 2;
 1
```

字符串函数

最近更新时间：2024-08-07 17:33:02

ASCII

函数语法：



```
ASCII(<str> string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回 str 的第一个字符的数值。

返回类型：integer。

示例：

```
> SELECT ascii('222');
 50
```

BASE64

函数语法：

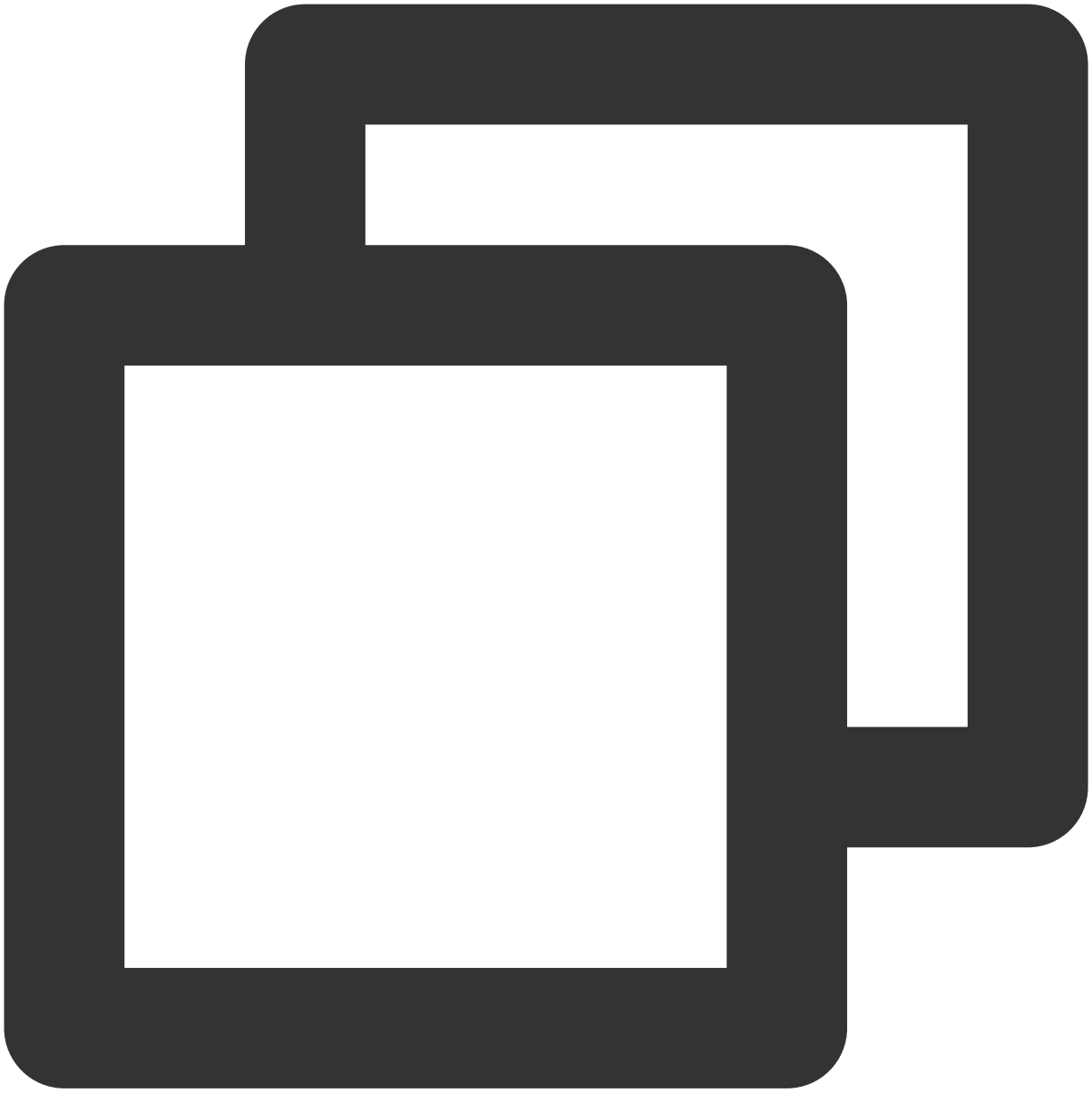
```
-- SparkSQL
BASE64(<str> string|binary)
-- Presto
BASE64(<str> binary)
```

支持引擎：SparkSQL、Presto。

使用说明：将参数转换为 base64 字符串。

返回类型：string。

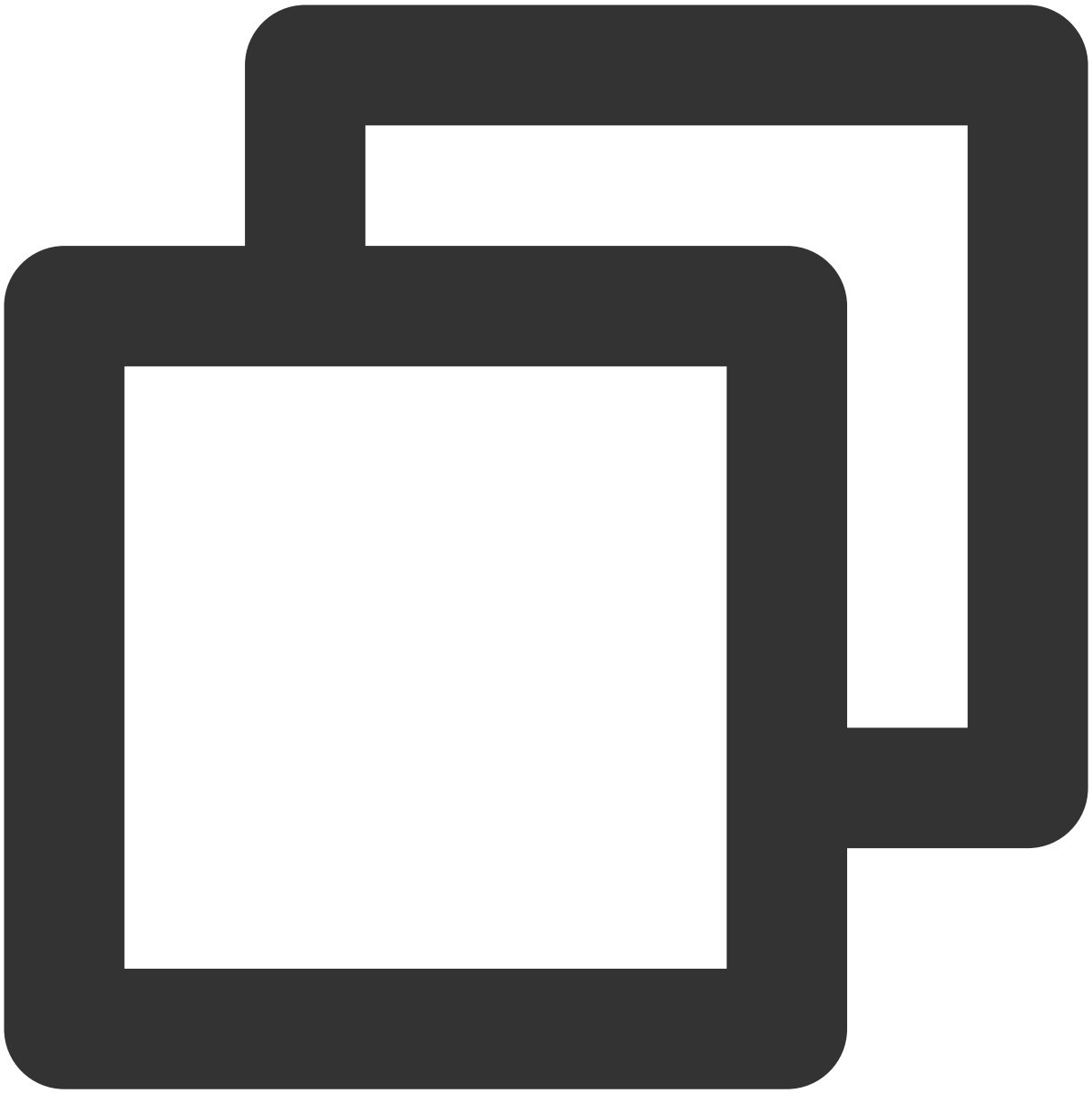
示例：



```
> SELECT base64('tencent');
 dGVuY2VudA==
```

BIT_LENGTH

函数语法：



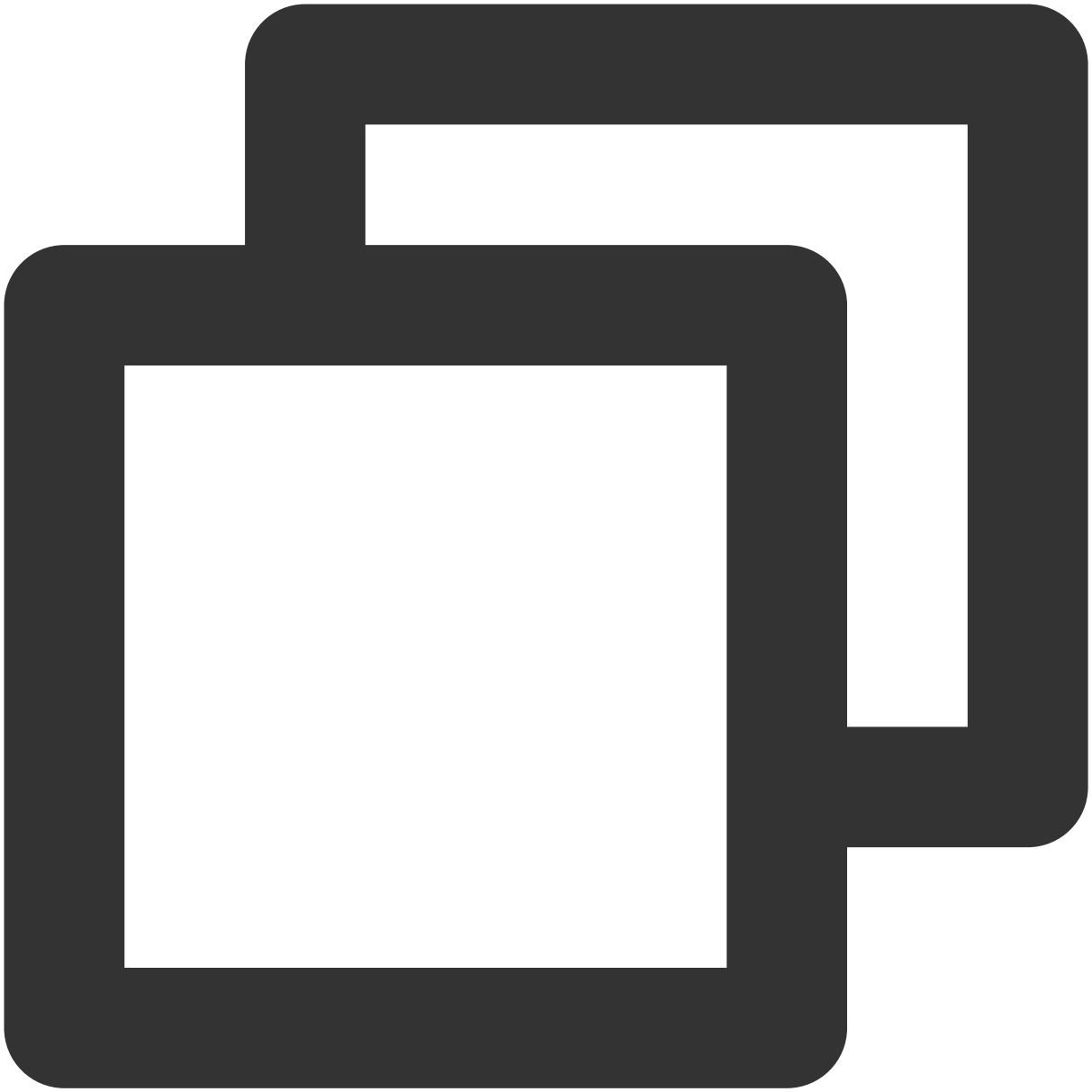
```
BIT_LENGTH(<expr> string|binary)
```

支持引擎：SparkSQL、Presto。

使用说明：返回字符串数据的位长度或二进制数据的位数。

返回类型：integer。

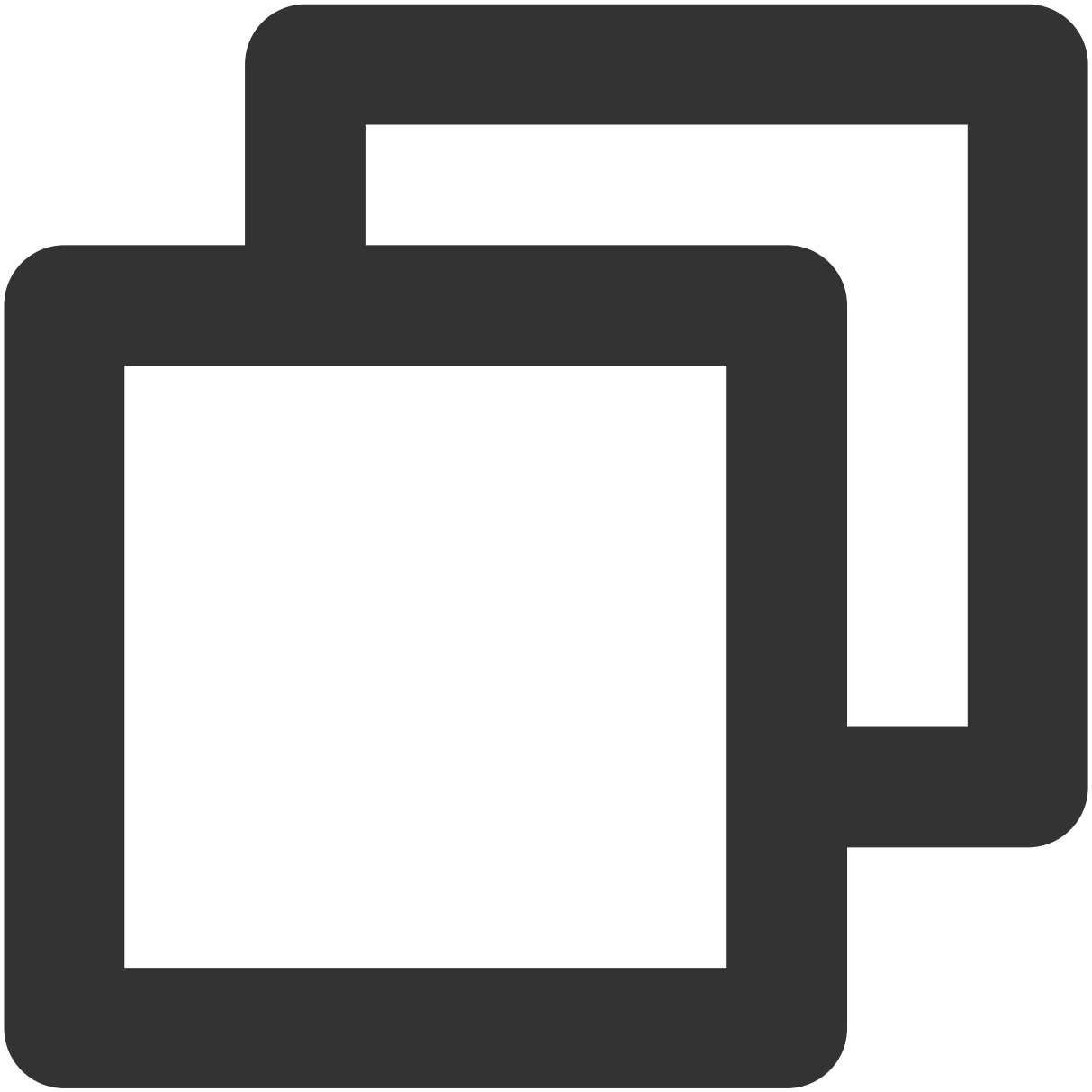
示例：



```
> select bit_length('tencent');
 56
> select bit_length(binary('tencent'));
 56
```

CHAR

函数语法：



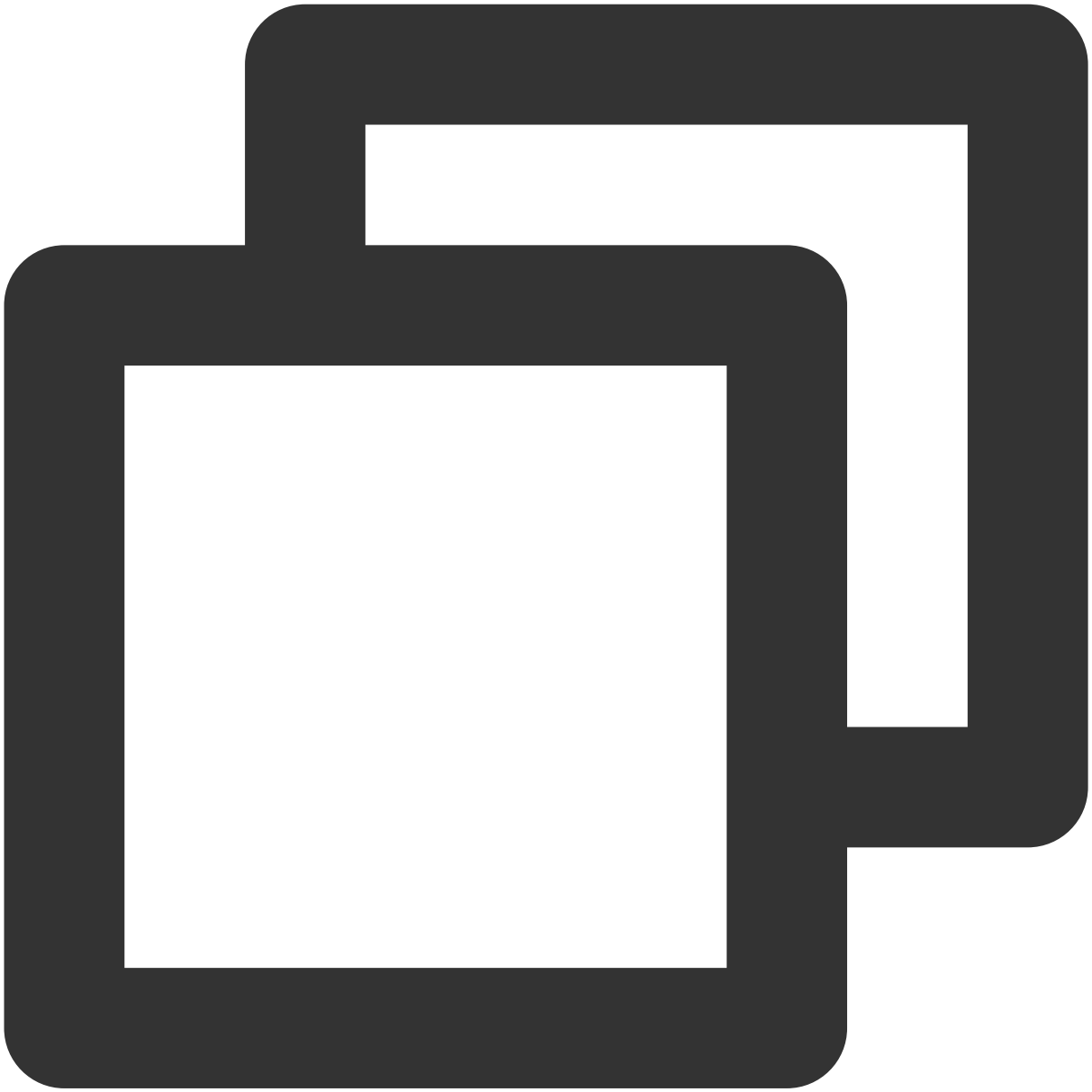
```
CHAR(<expr> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：返回 expr 的 ASCII 字符。如果 expr 大于256，则 expr=expr%256。

返回类型：string。

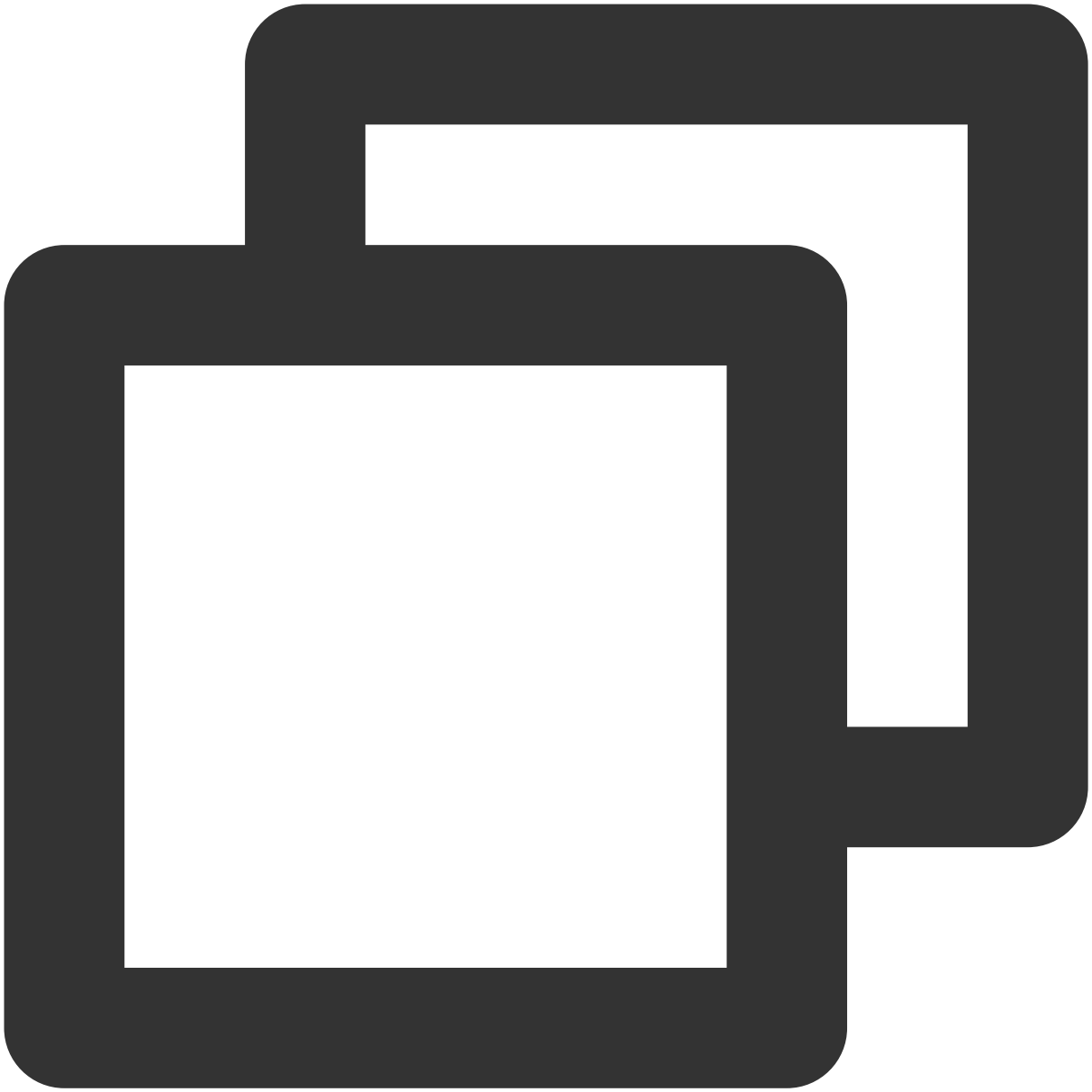
示例：



```
> SELECT char(65);
 A
```

CHR

函数语法：



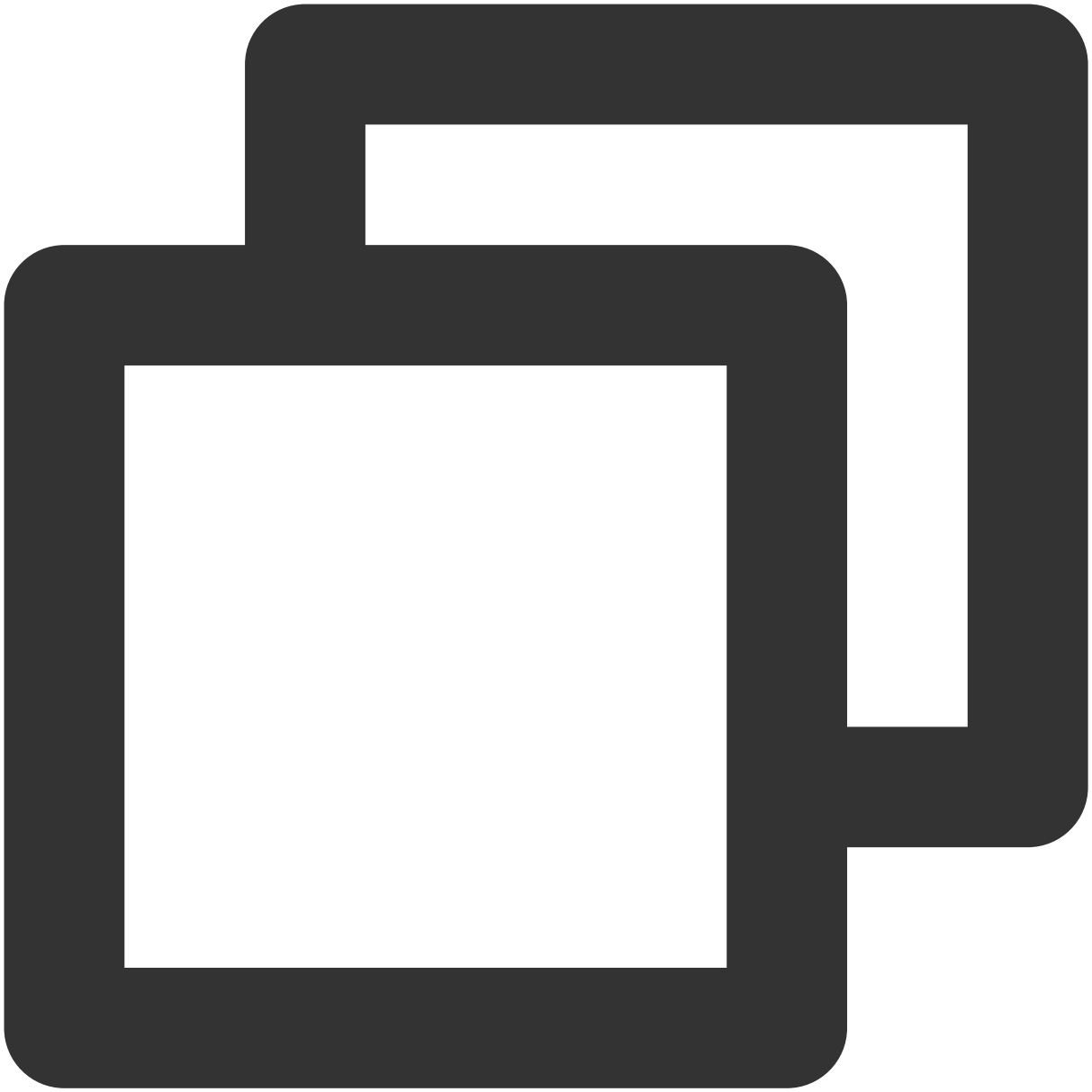
```
CHR(<expr> integer)
```

支持引擎：SparkSQL。

使用说明：返回 expr 的 ASCII 字符。如果 expr 大于256，则 expr=expr%256。

返回类型：string。

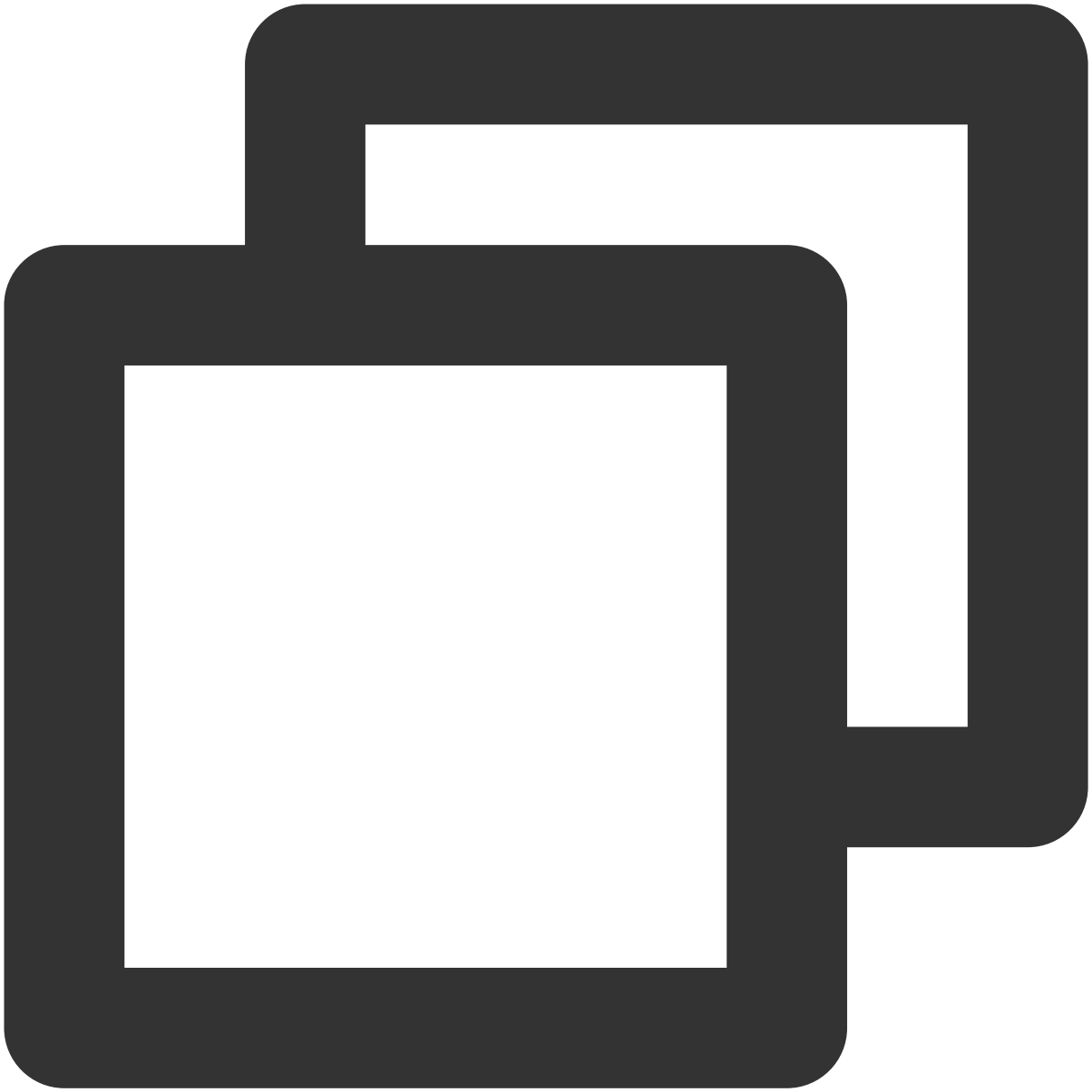
示例：



```
> SELECT chr(65);
 A
```

CHAR_LENGTH

函数语法：



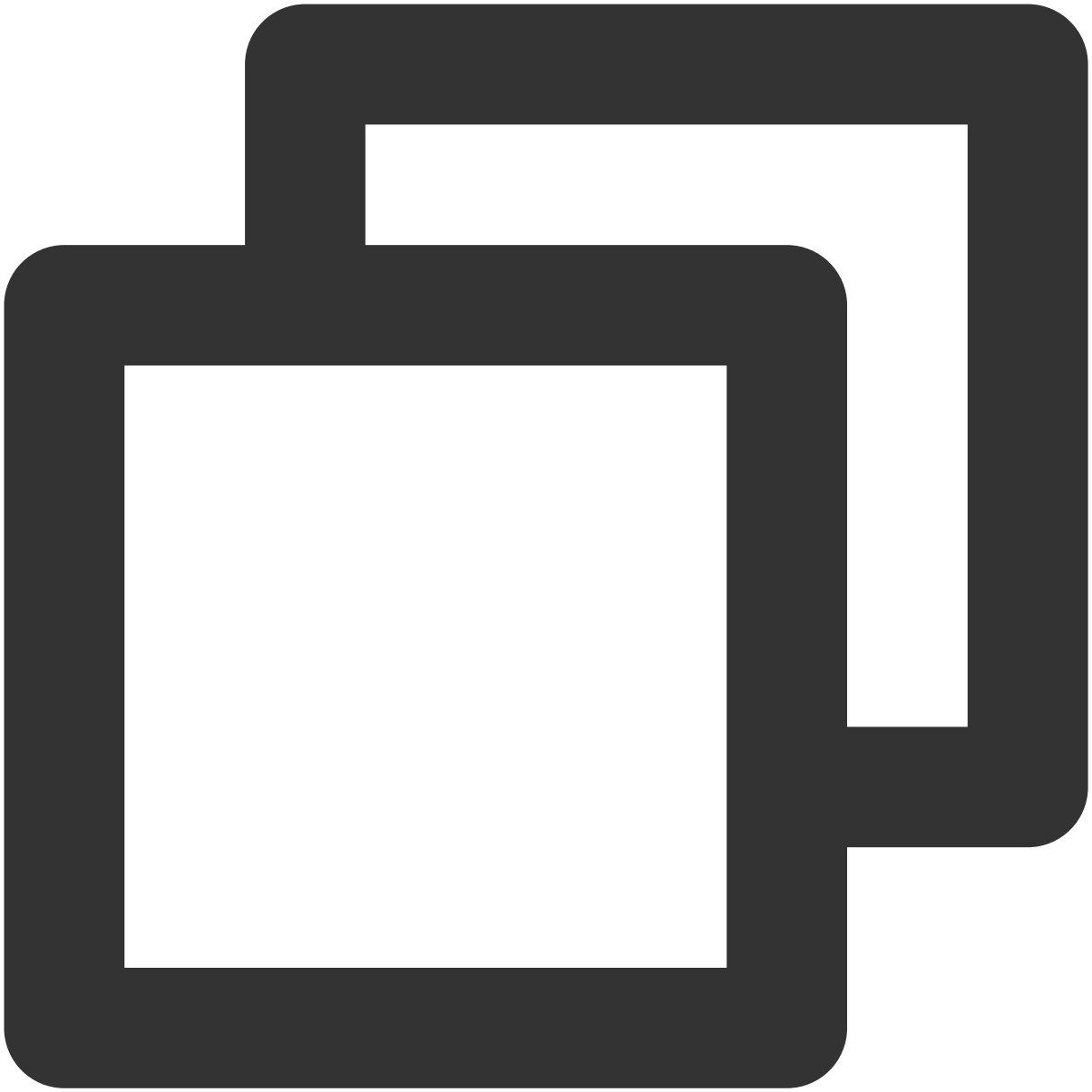
```
CHAR_LENGTH(<expr> string|binary)
```

支持引擎：SparkSQL、Presto。

使用说明：返回字符串数据的字符长度或二进制数据的字节数。字符串数据的长度包括末尾空格。二进制数据的长度包括二进制零。

返回类型：integer。

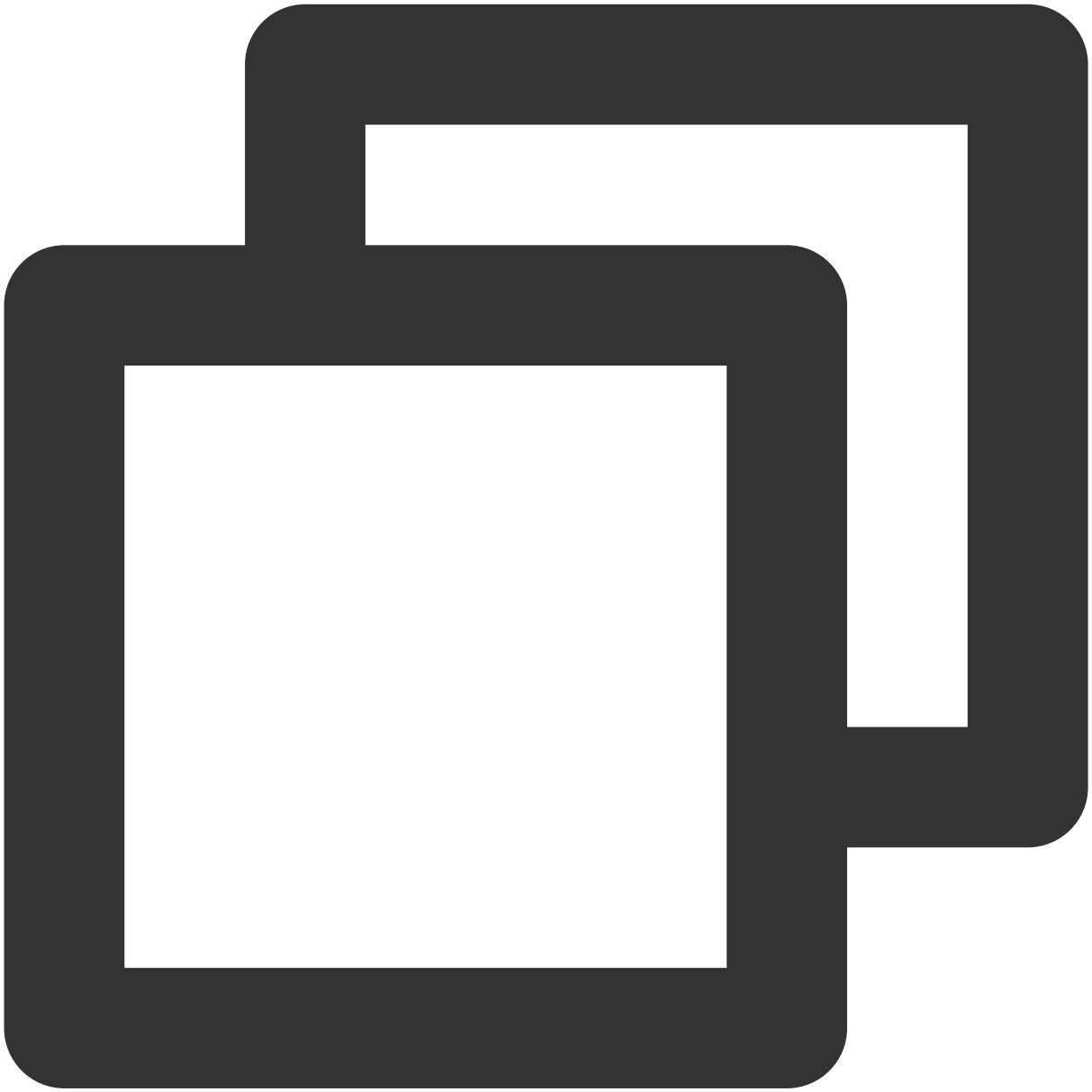
示例：



```
> select char_length(binary('tencent'));
 7
> select char_length('tencent');
 7
```

CHARACTER_LENGTH

函数语法：



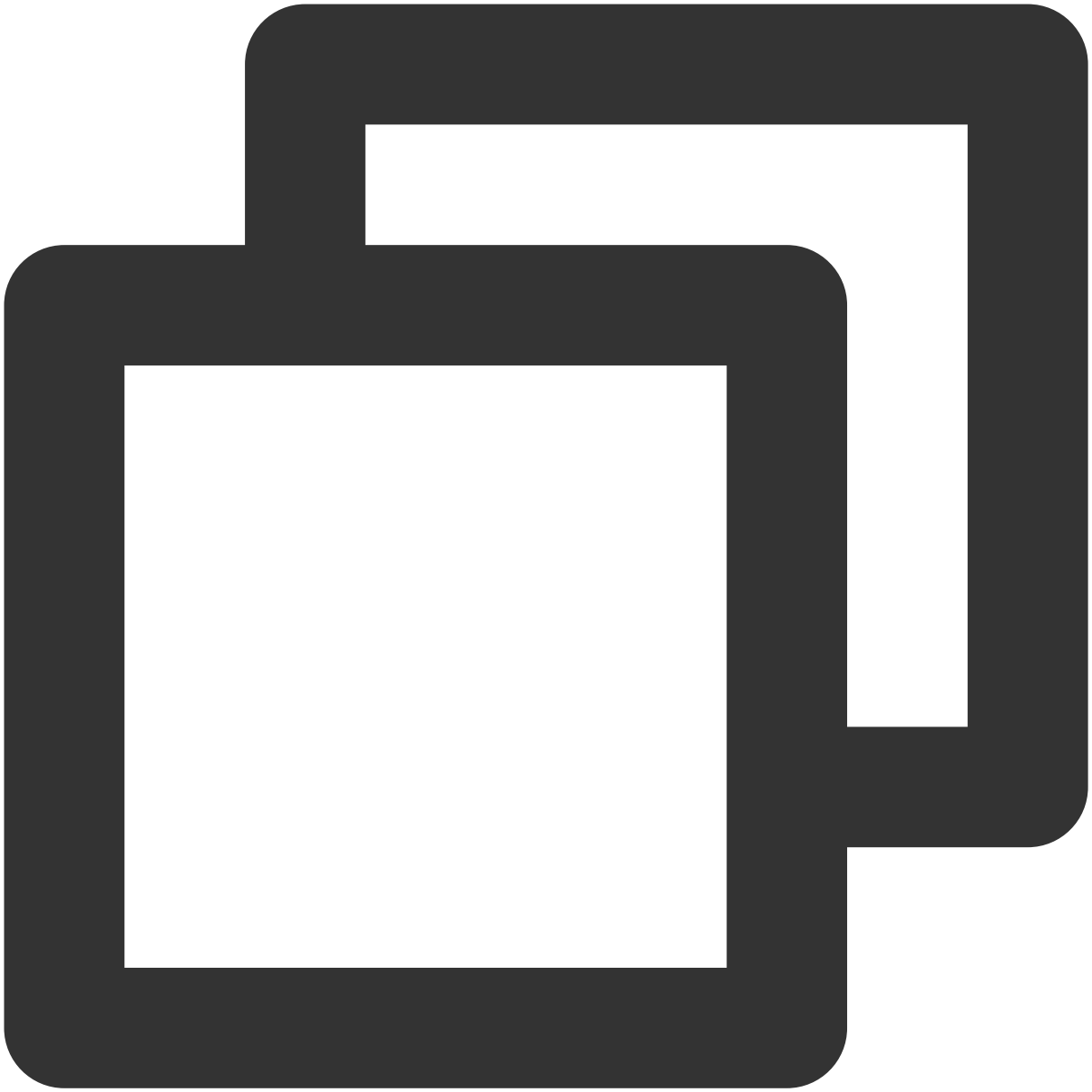
```
CHARACTER_LENGTH(<expr> string|binary)
```

支持引擎：SparkSQL、Presto。

使用说明：返回字符串数据的字符长度或二进制数据的字节数。字符串数据的长度包括末尾空格。二进制数据的长度包括二进制零。

返回类型：integer。

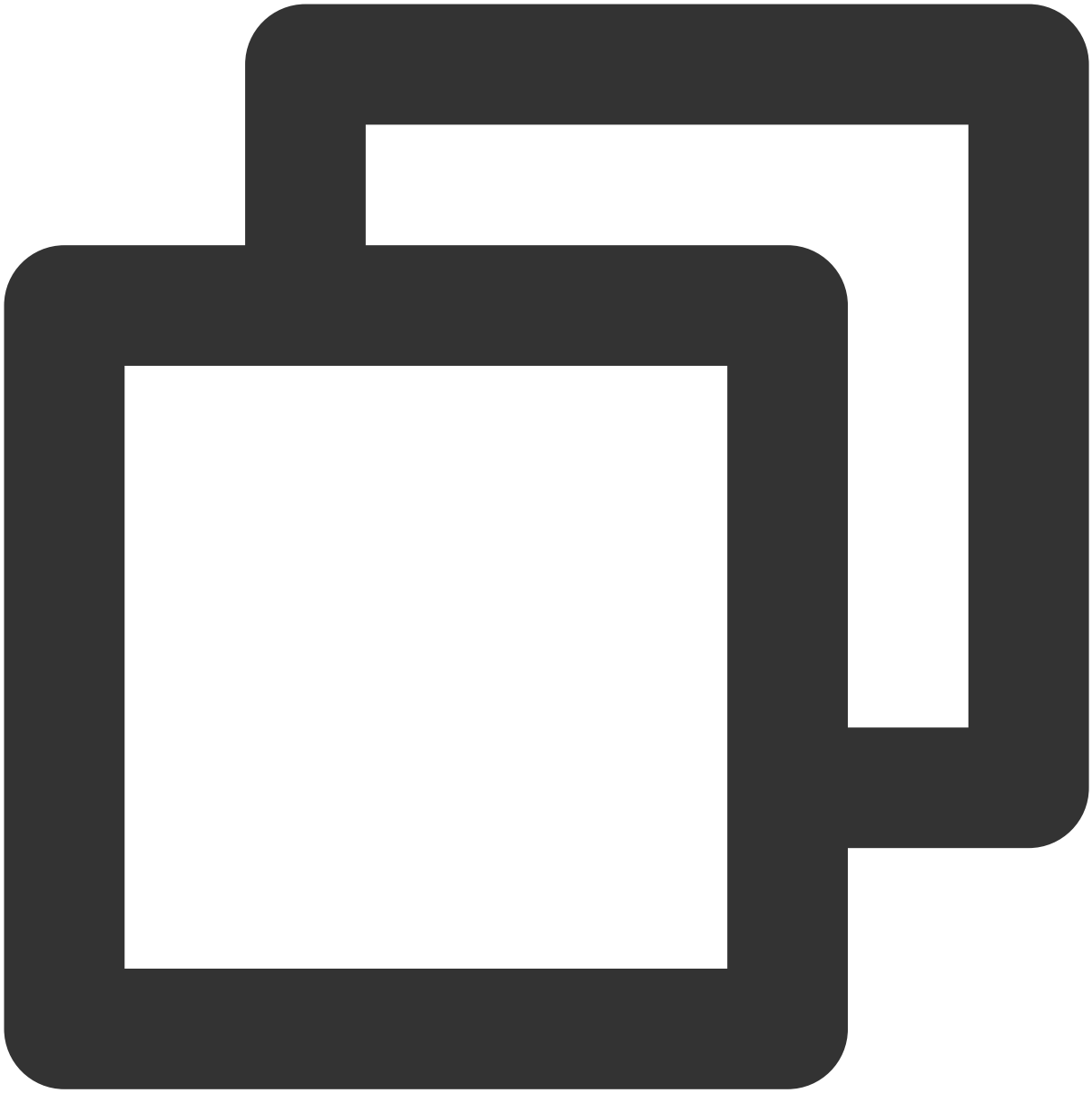
示例：



```
> select character_length(binary('tencent'));
 7
> select character_length('tencent');
 7
```

CONCAT_WS

函数语法：



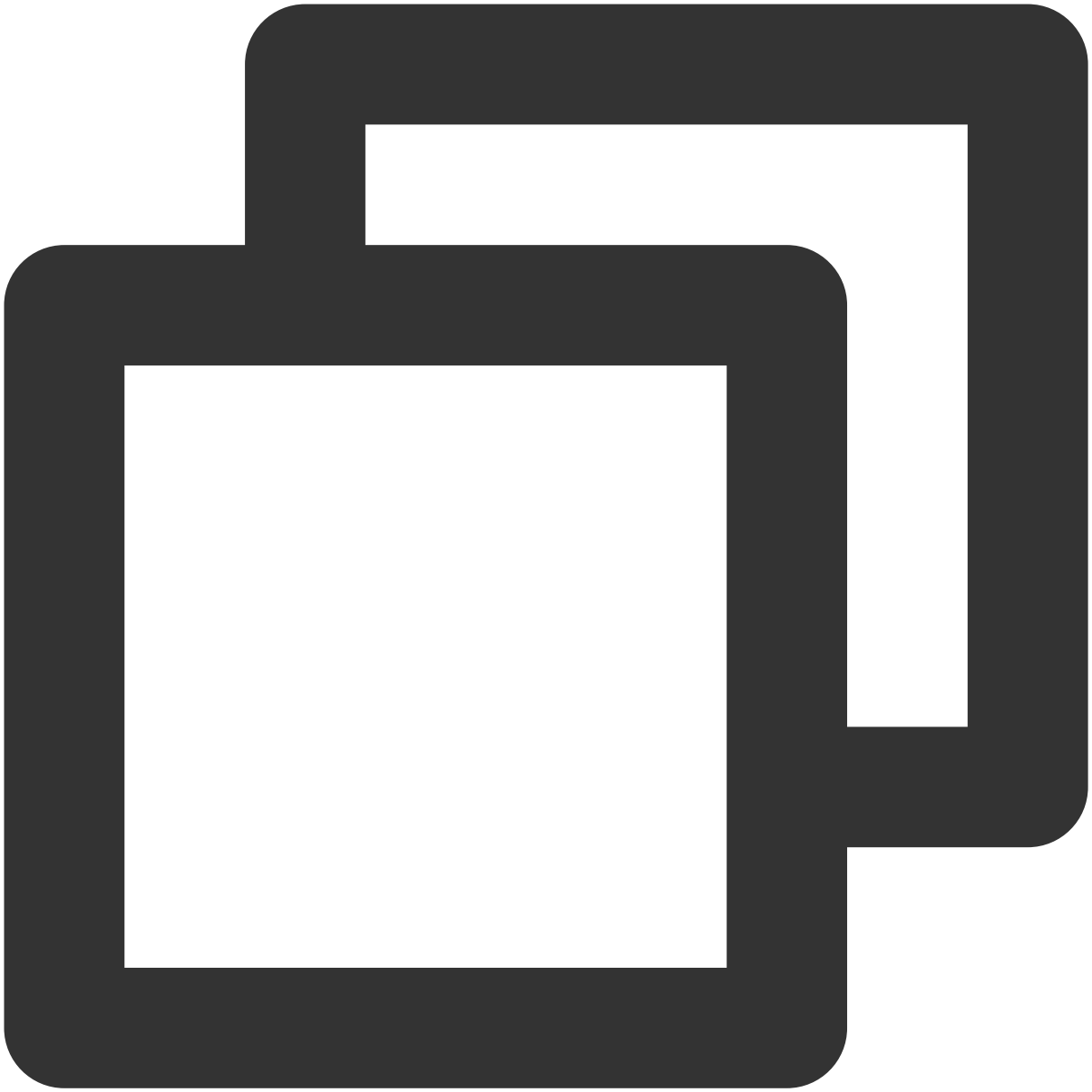
```
CONCAT_WS(<sep> string[, <s> string|array<string>}]+)
```

支持引擎：SparkSQL、Presto。

使用说明：返回由 sep 分隔的字符串。

返回类型：string。

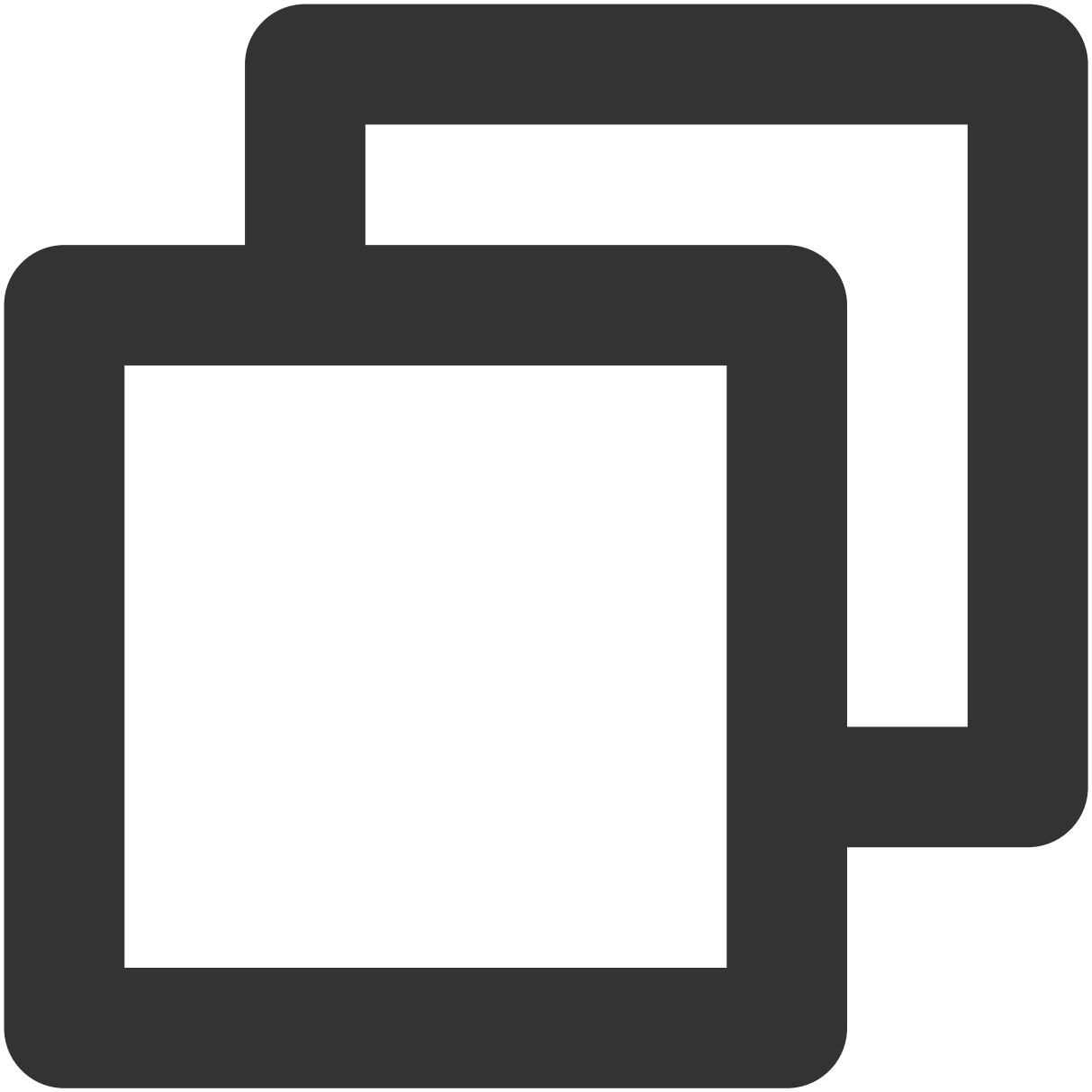
示例：



```
> SELECT concat_ws(' ', 'tencent', 'dlc');
tencent dlc
```

DECODE

函数语法：



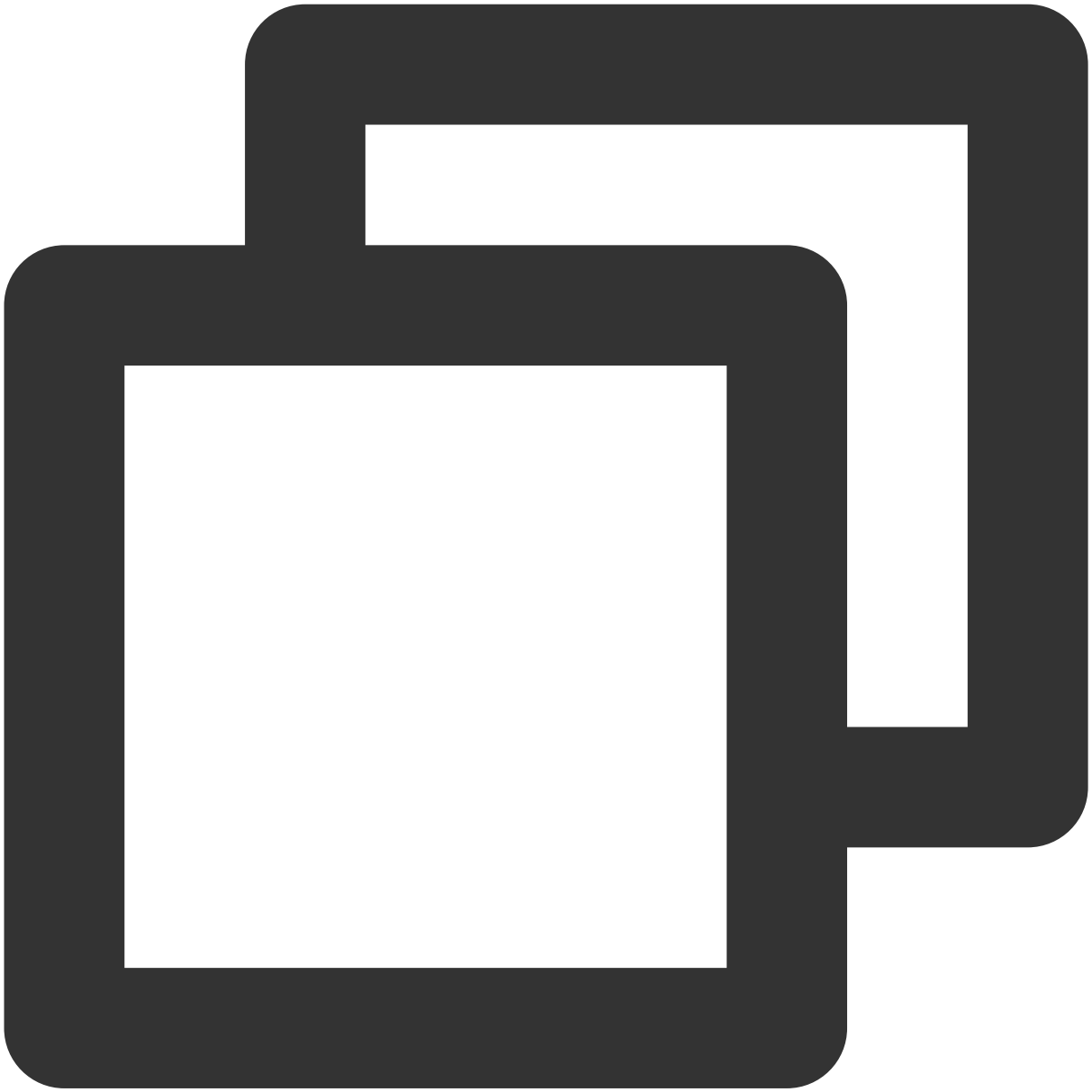
```
DECODE(<expr> binary|string, <charset>string)
```

支持引擎：SparkSQL、Presto。

使用说明：使用第二个参数字符集解码第一个参数。

返回类型：string。

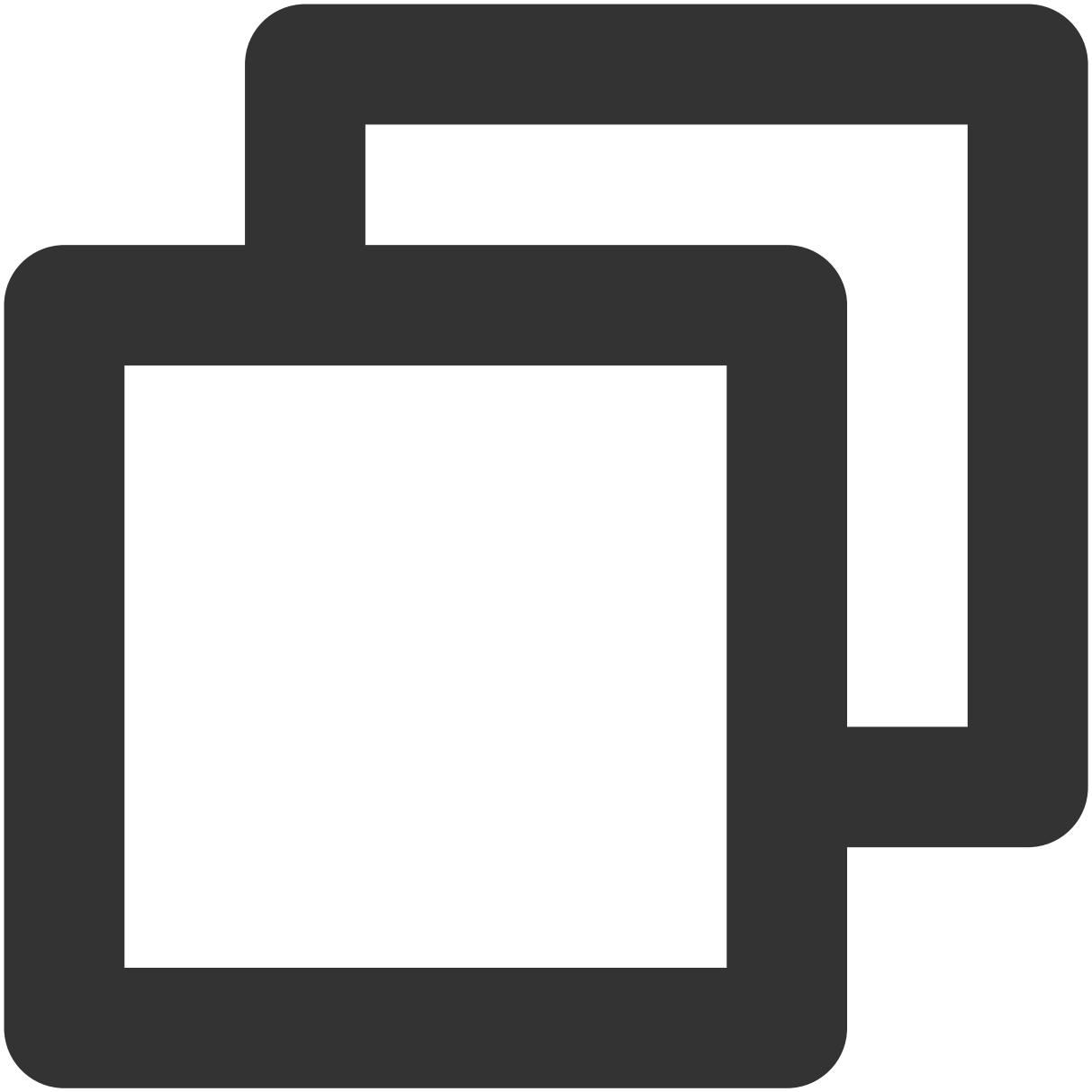
示例：



```
> SELECT decode(encode('abc', 'utf-8'), 'utf-8');
 abc
```

ELT

函数语法：



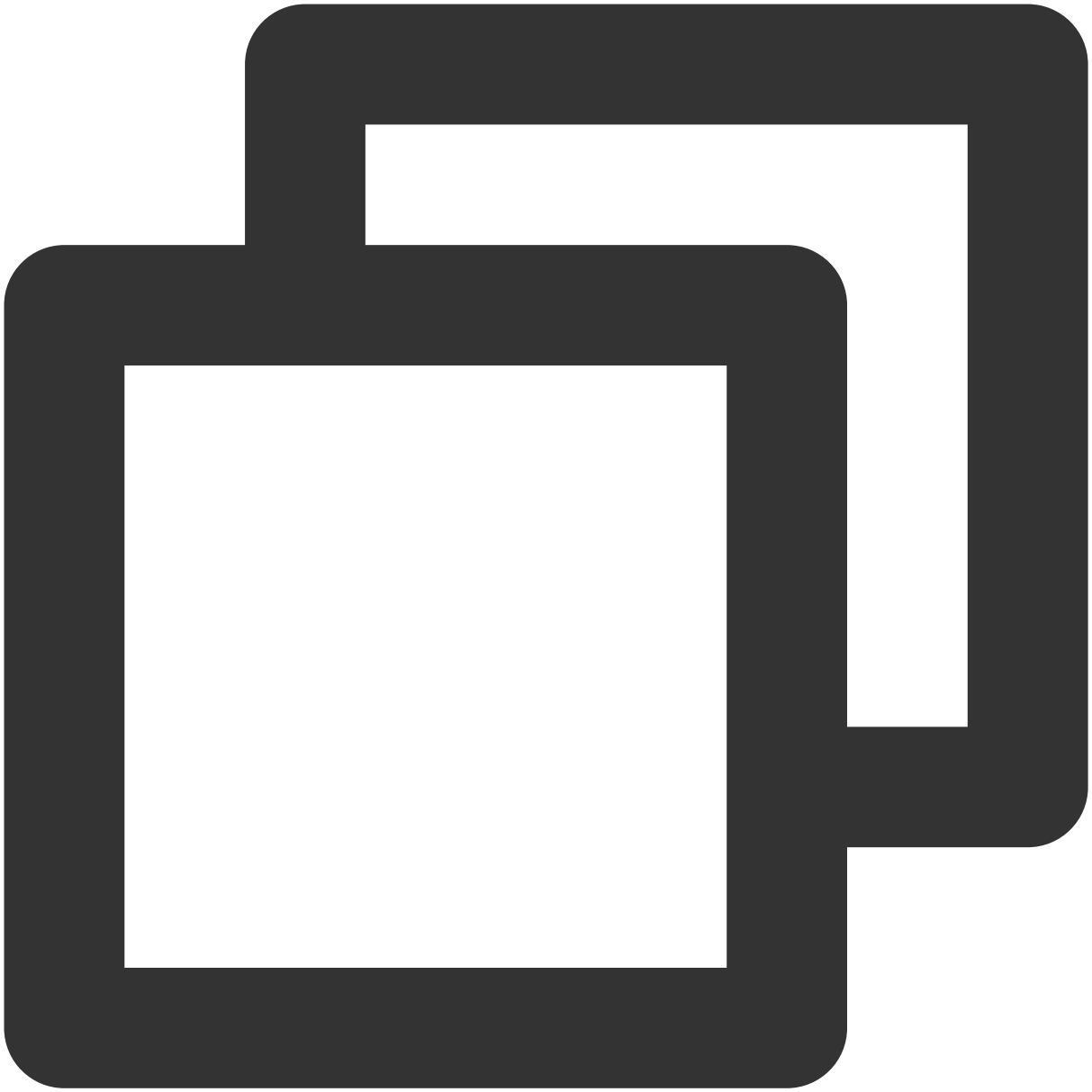
```
ELT(<n> integer, <s1> string, <s2> string, ...)
```

支持引擎：SparkSQL、Presto。

使用说明：返回第 n 个元素。

返回类型：string。

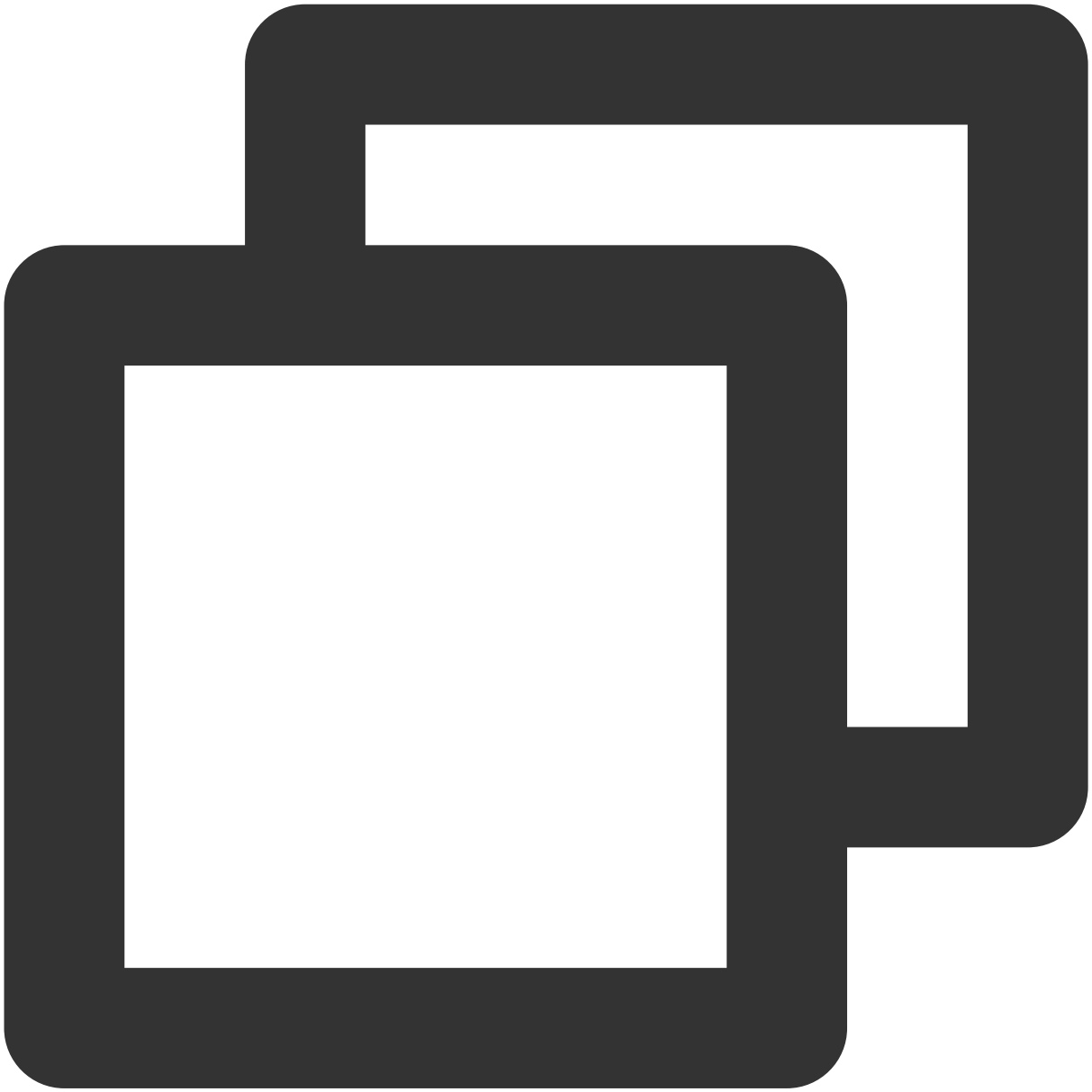
示例：



```
> SELECT elt(1, 'scala', 'java');
 scala
```

ENCODE

函数语法：



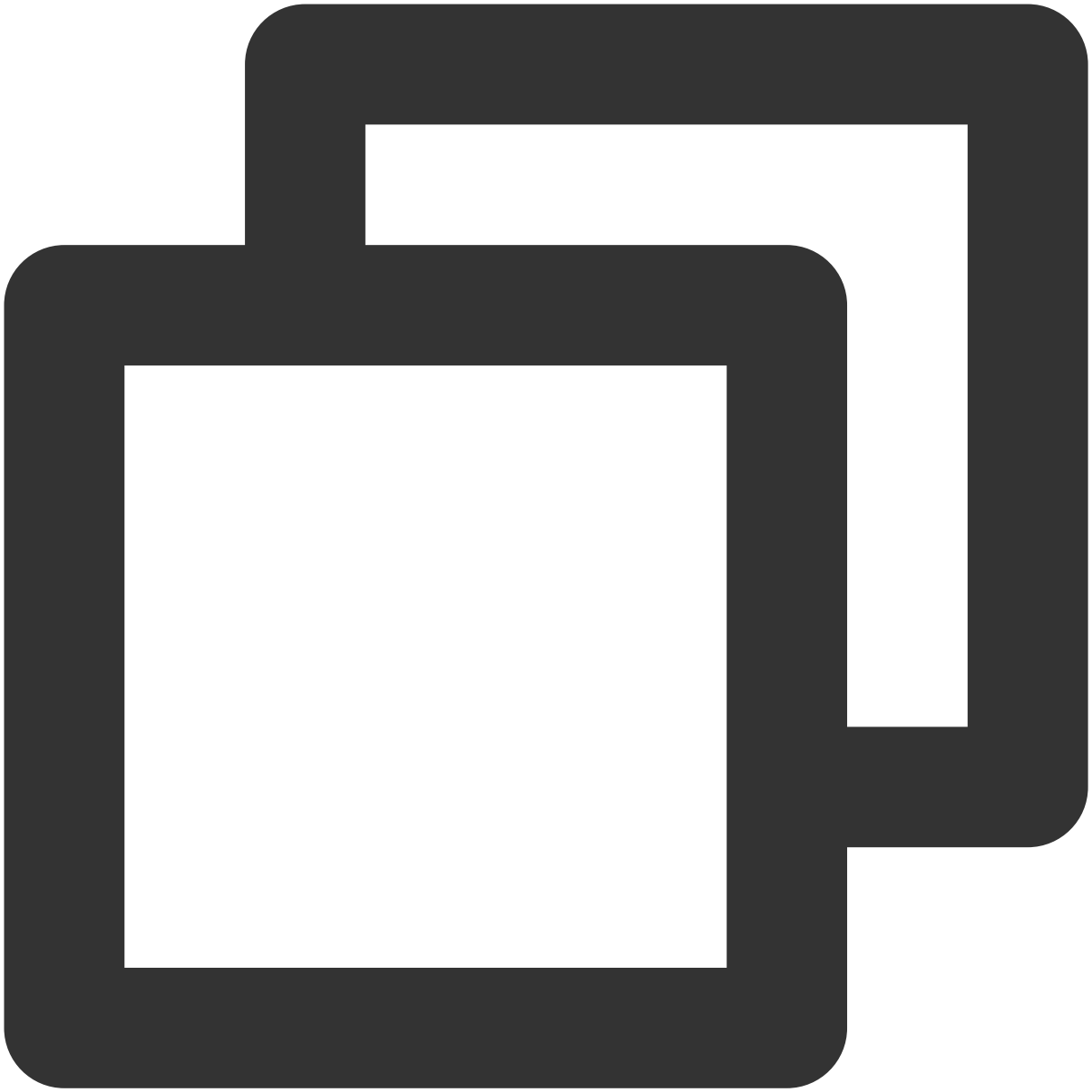
```
ENCODE(<expr> binary|string, <charset> string)
```

支持引擎：SparkSQL、Presto。

使用说明：使用第二个参数字符集对第一个参数进行编码。

返回类型：string。

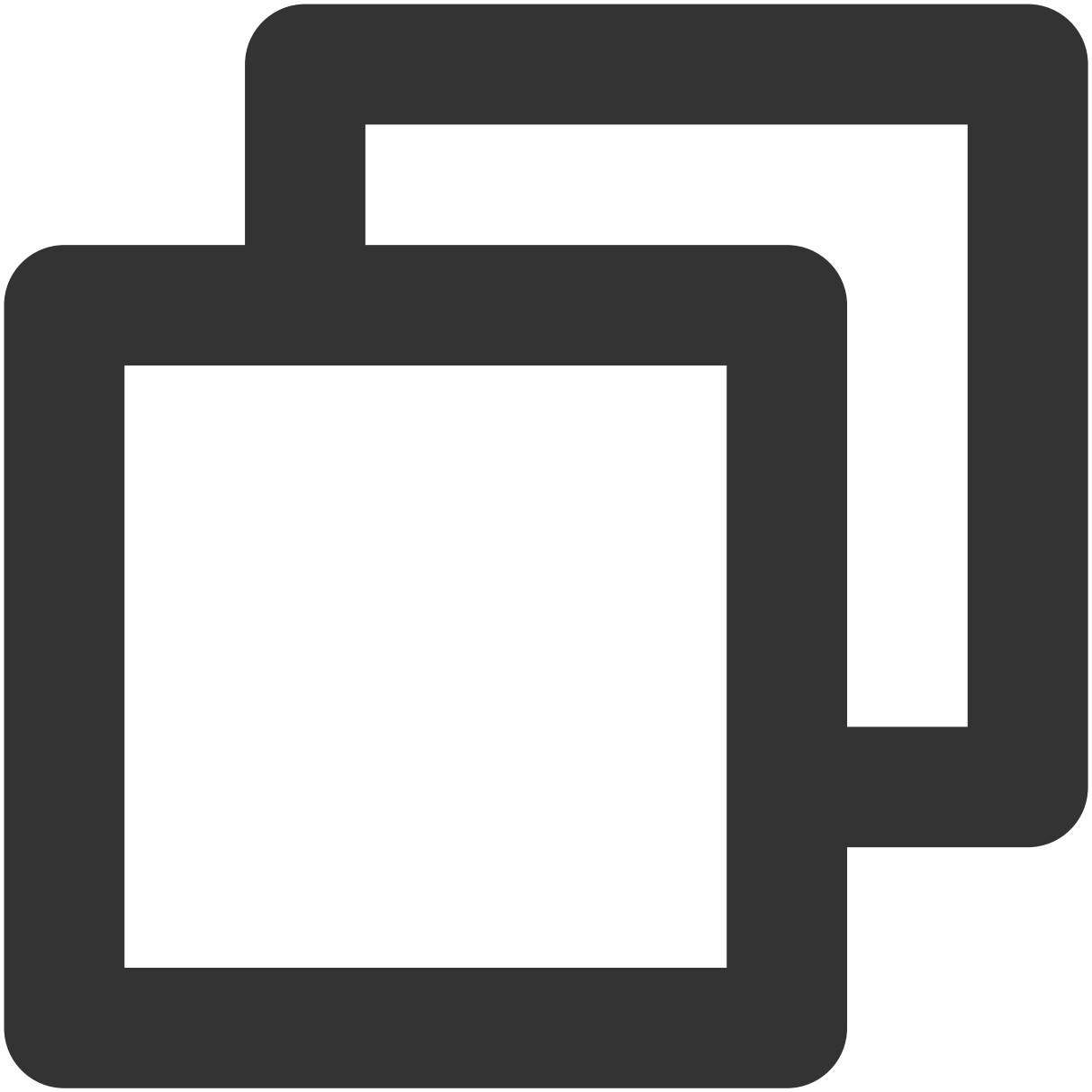
示例：



```
> SELECT encode('abc', 'utf-8');
 abc
> SELECT encode(x'616263', 'utf-8');
 abc
```

FIND_IN_SET

函数语法：



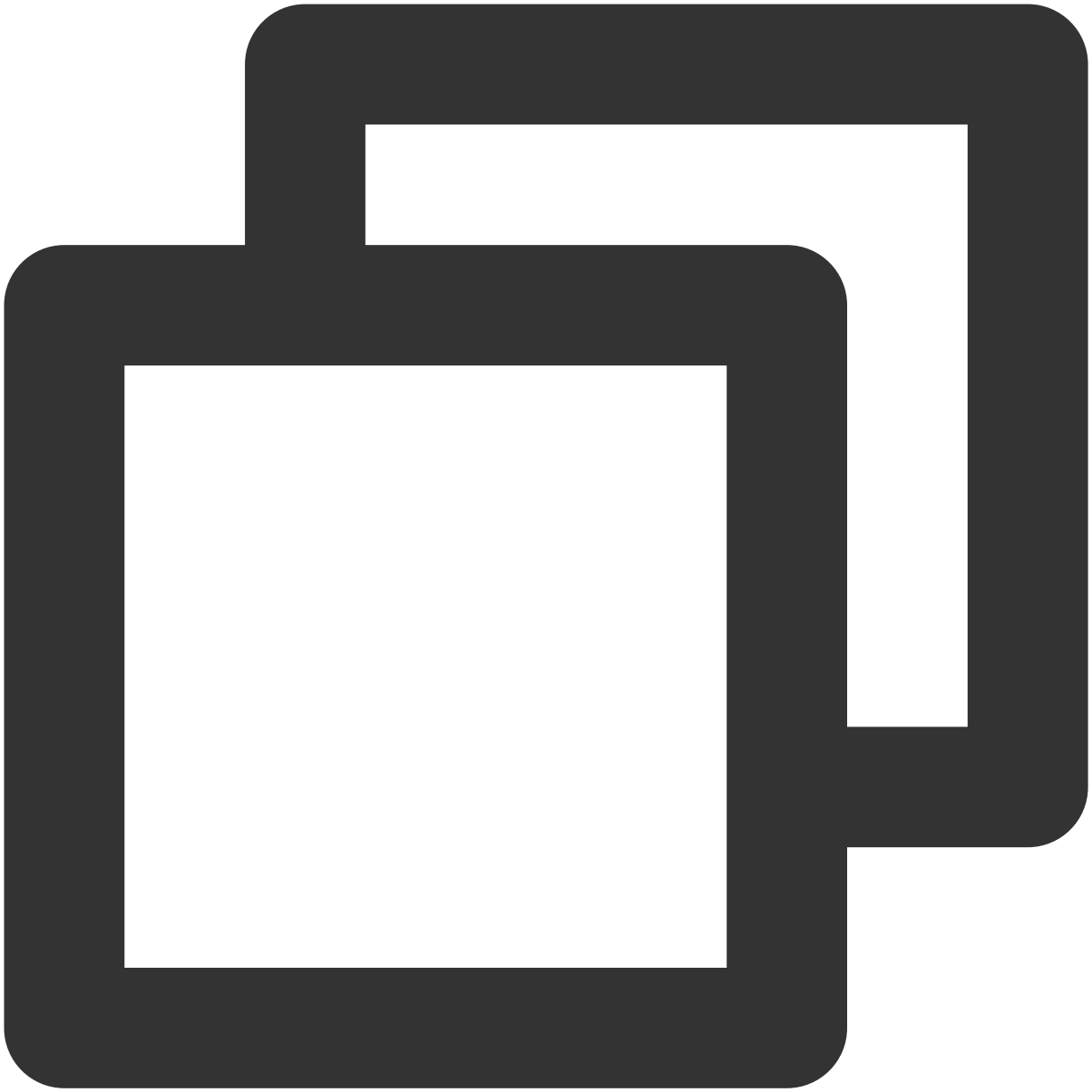
```
FIND_IN_SET(<str> string, <str_array> string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回逗号分隔列表 atr_array 中给定字符串 str 的索引（从1开始计数）。如果未找到字符串或 str 包含逗号，则返回0。

返回类型：integer。

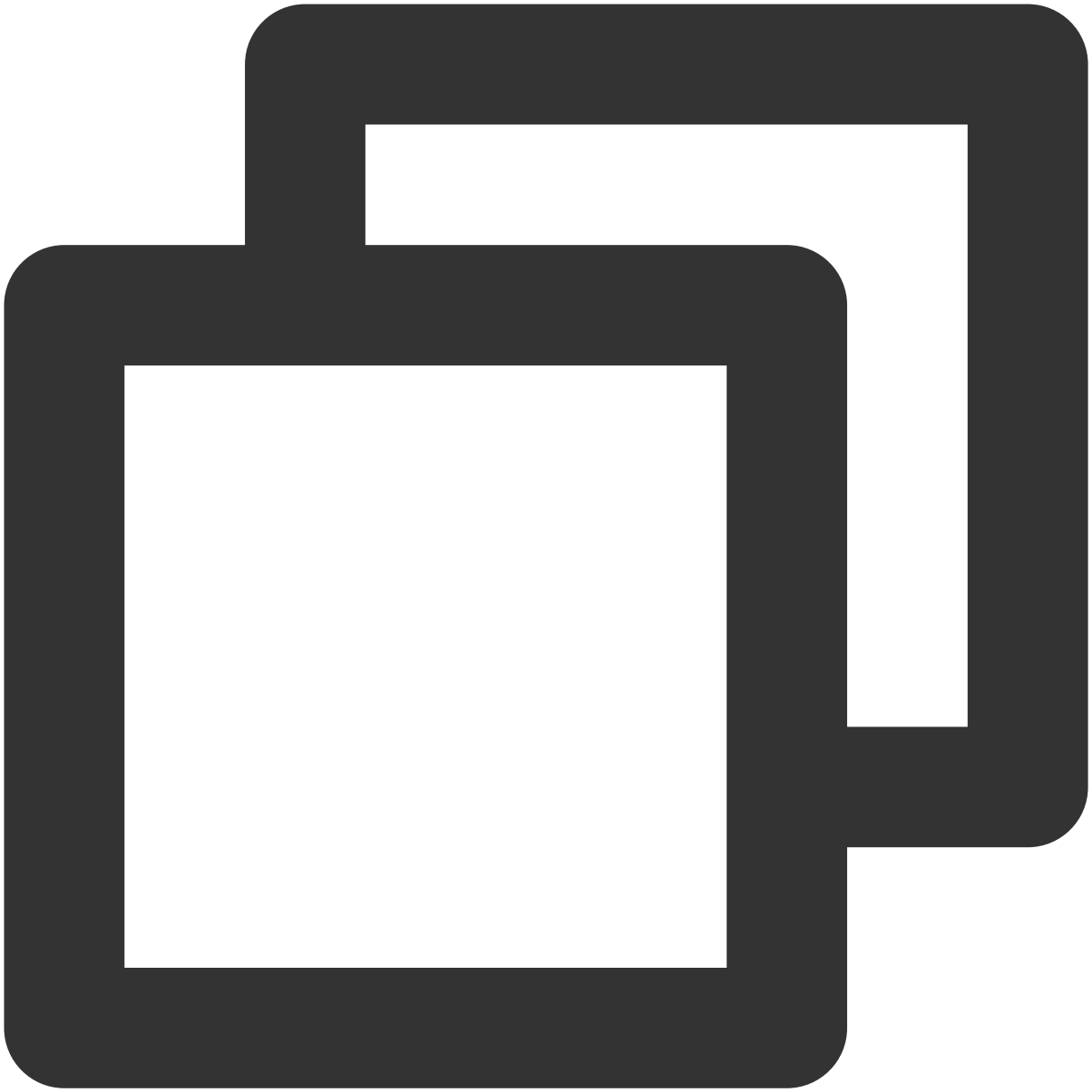
示例：



```
> SELECT find_in_set('ab','abc,b,ab,c,def');
 3
```

FORMAT_NUMBER

函数语法：



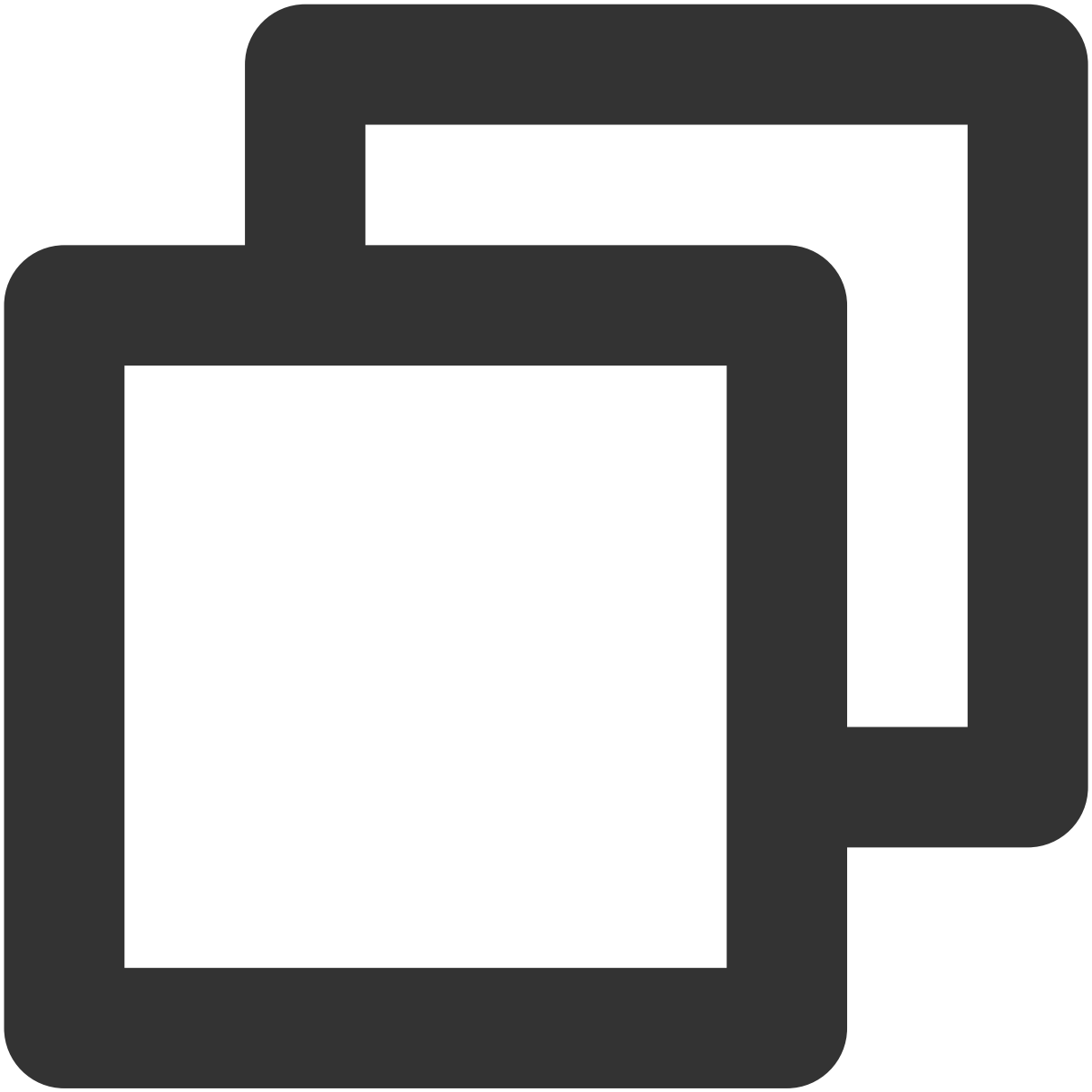
```
FORMAT_NUMBER(<expr1> integer|double|decimal, <expr2> string|integer)
```

支持引擎：SparkSQL、Presto。

使用说明：将 expr1 的格式设置为'#,####,####.##'，四舍五入到 expr2 小数位。如果 expr2 为0，则结果没有小数点或小数部分。

返回类型：string。

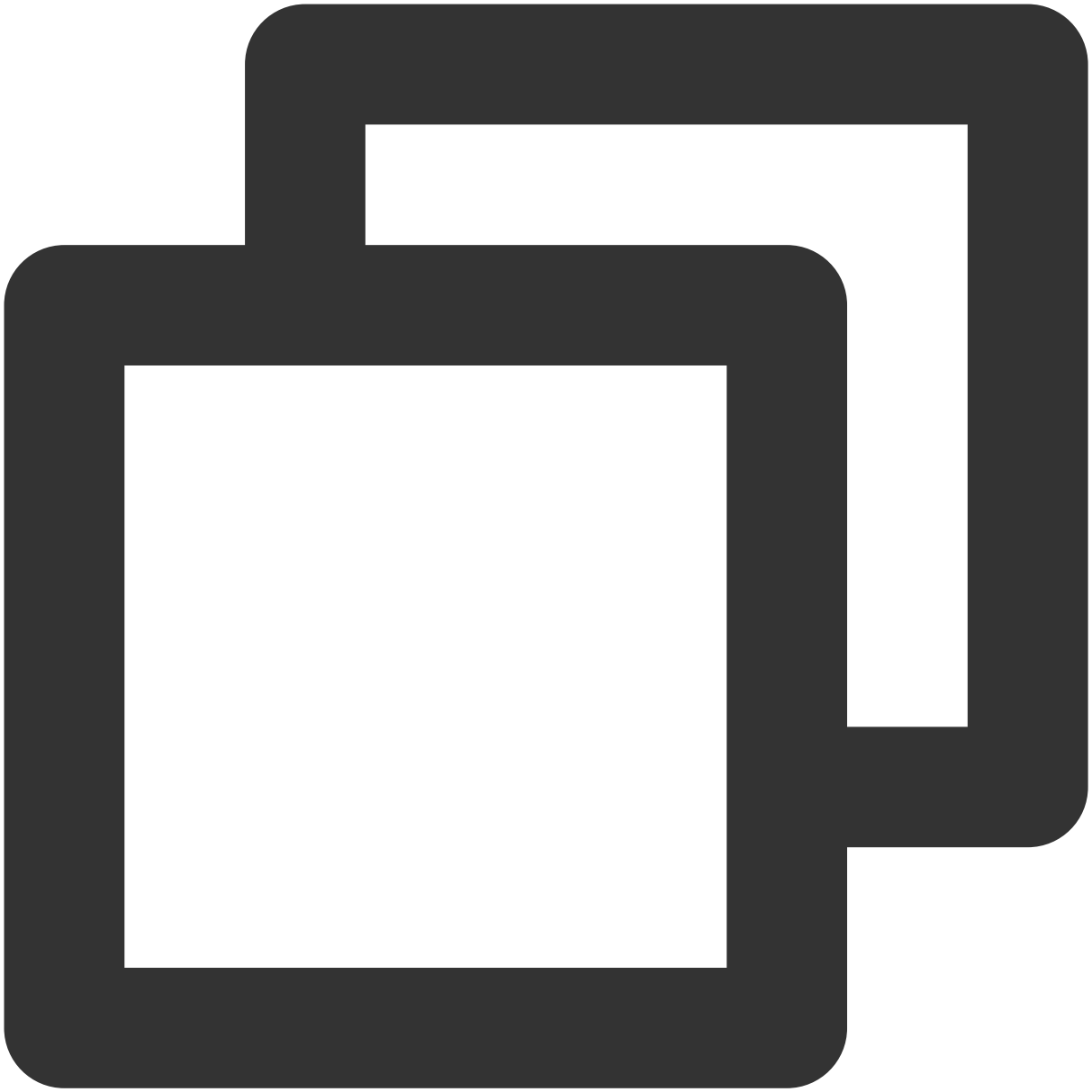
示例：



```
> SELECT format_number(12332.123456, 4);
 12,332.1235
> SELECT format_number(12332.123456, '##################.###');
 12332.123
```

FORMAT_STRING

函数语法：



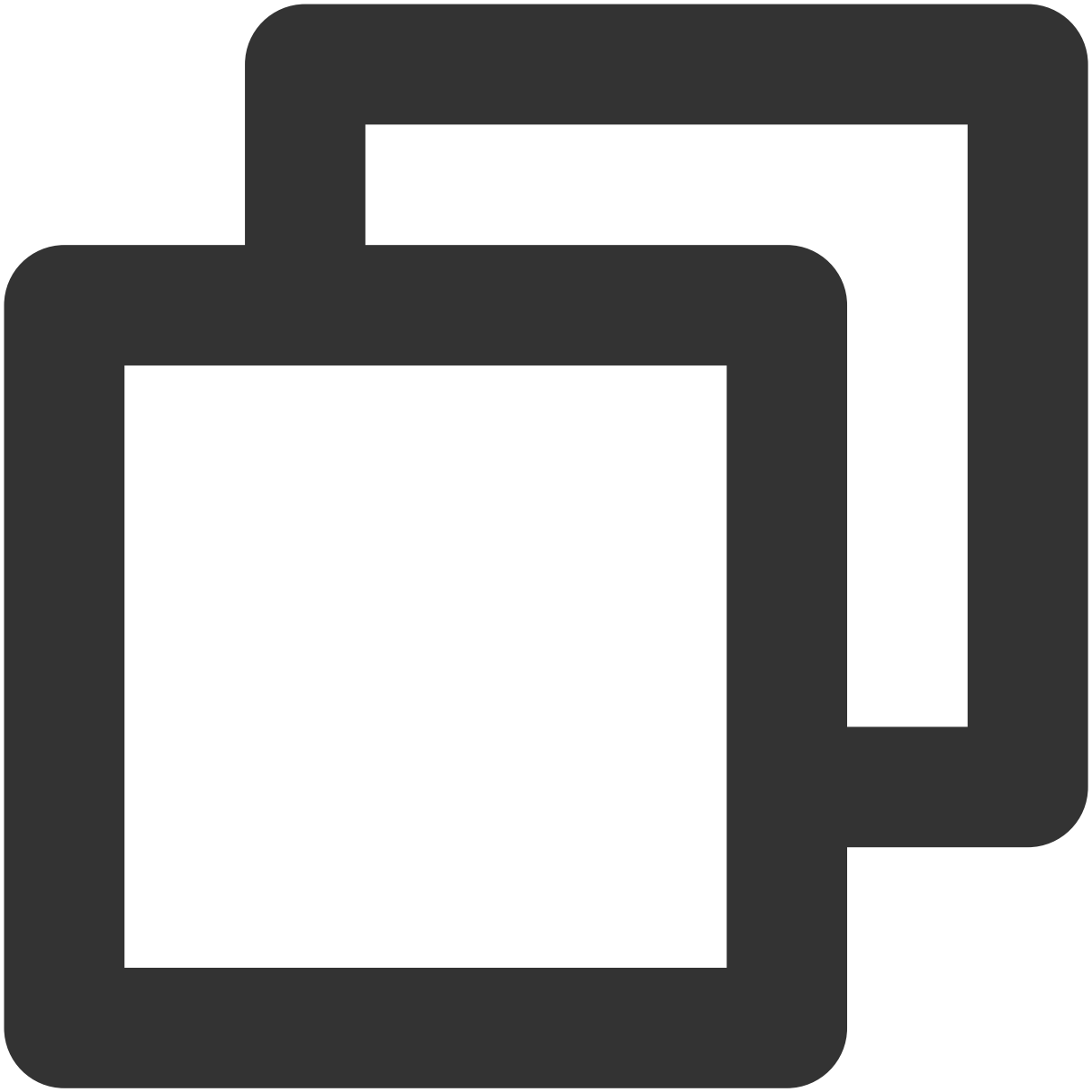
```
FORMAT_STRING(<str> string, obj <T>, ...)
```

支持引擎：SparkSQL、Presto。

使用说明：返回 printf 样式格式字符串中的格式化字符串。

返回类型：string。

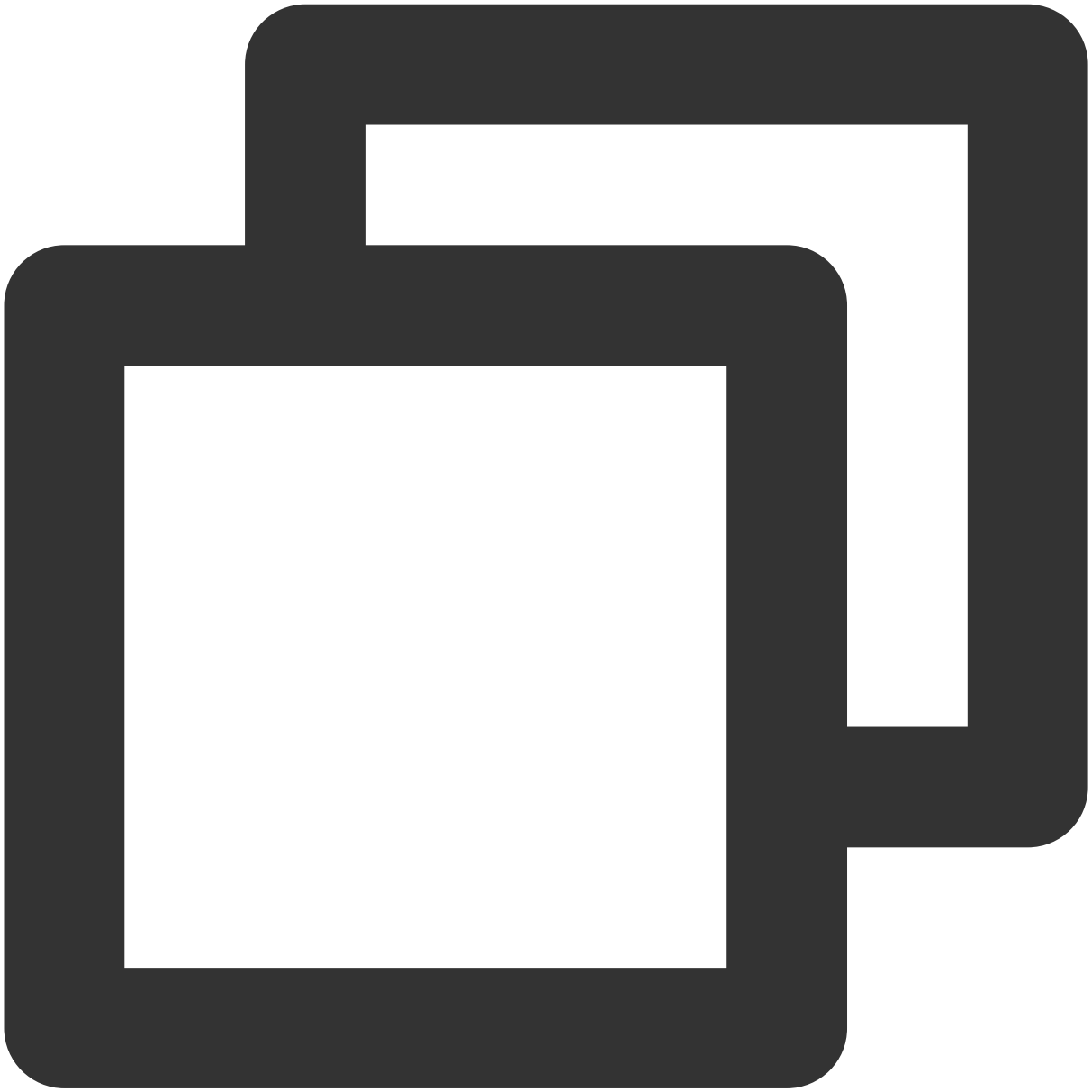
示例：



```
> SELECT format_string("Hello World %d %s", 100, "days");
 Hello World 100 days
```

INITCAP

函数语法：



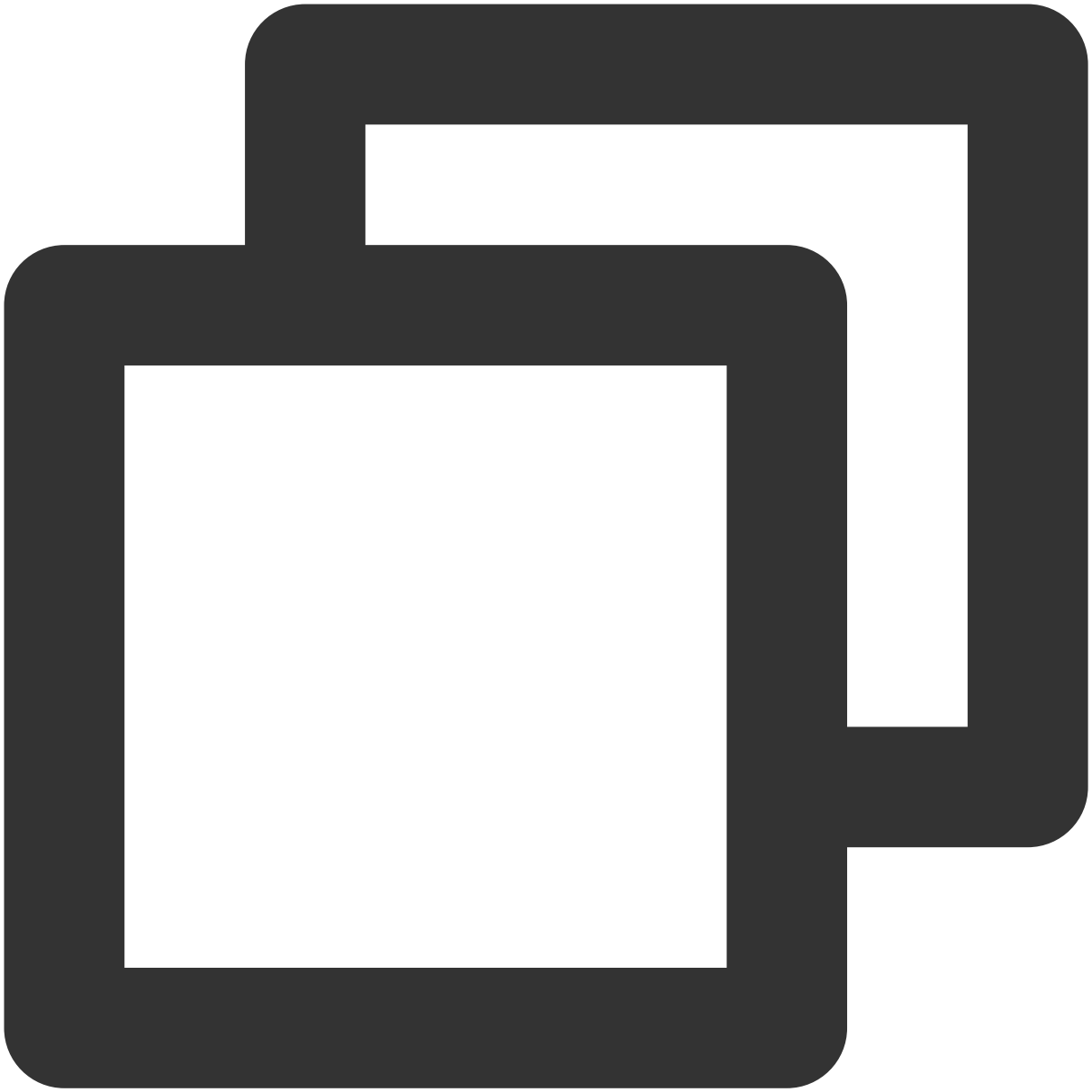
```
INITCAP(<str> string)
```

支持引擎：SparkSQL、Presto。

使用说明：每个单词的第一个字母都改为大写，所有其他字母均为小写。

返回类型：string。

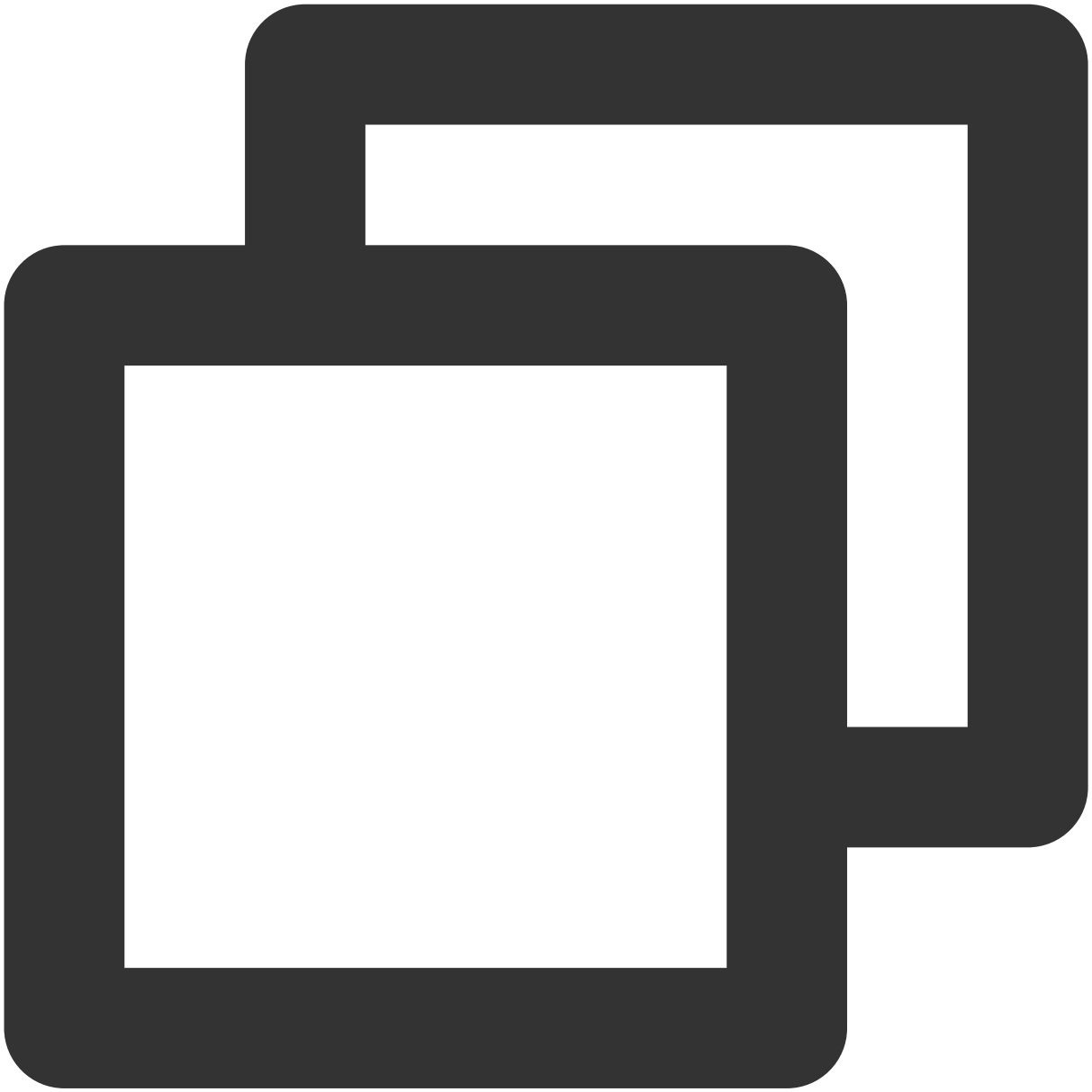
示例：



```
> SELECT initcap('sPark sql');
 Spark Sql
```

INSTR

函数语法：



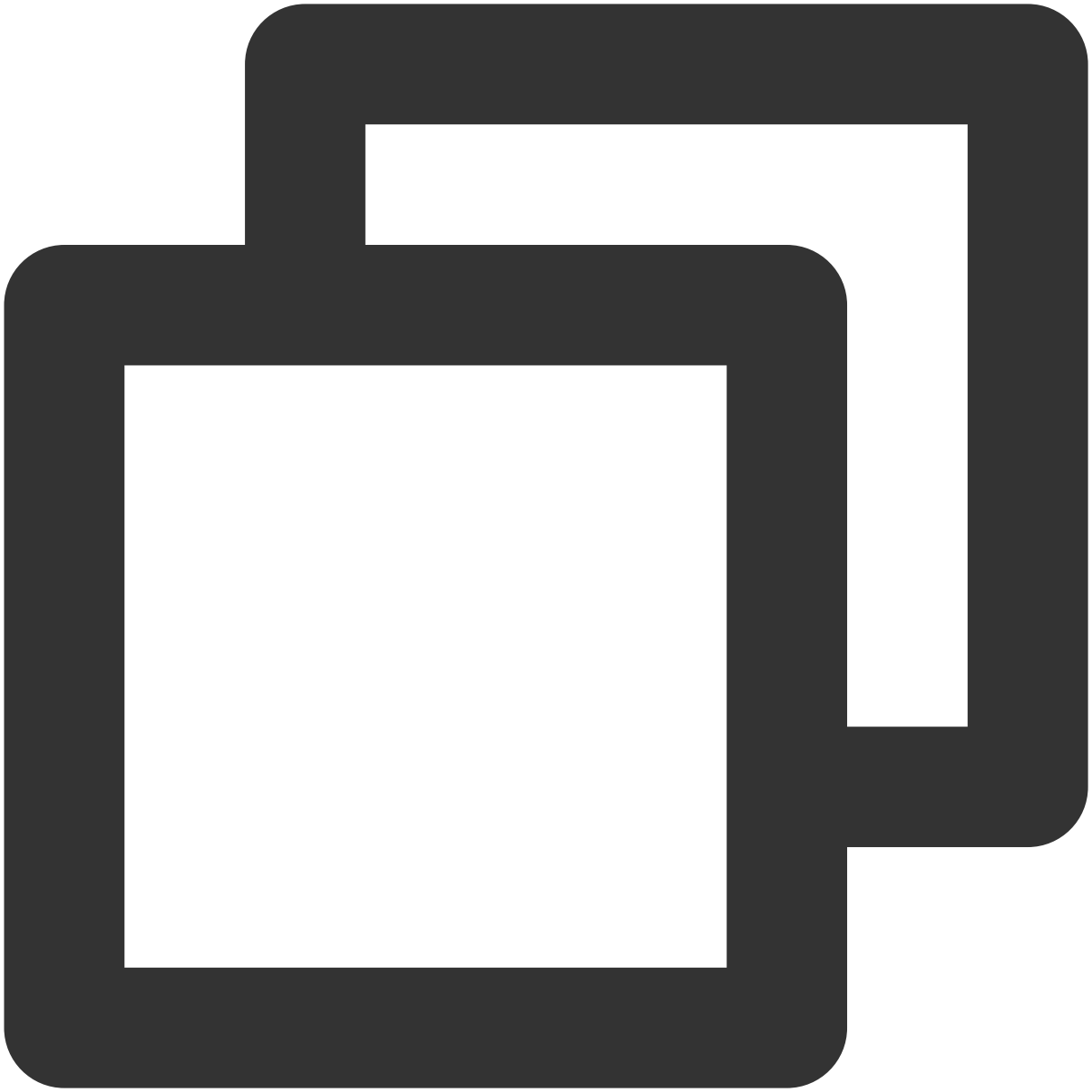
```
INSTR(<str> string, <substr> string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回 str 中第一次出现 substr 的索引（从1开始计数）。

返回类型：integer。

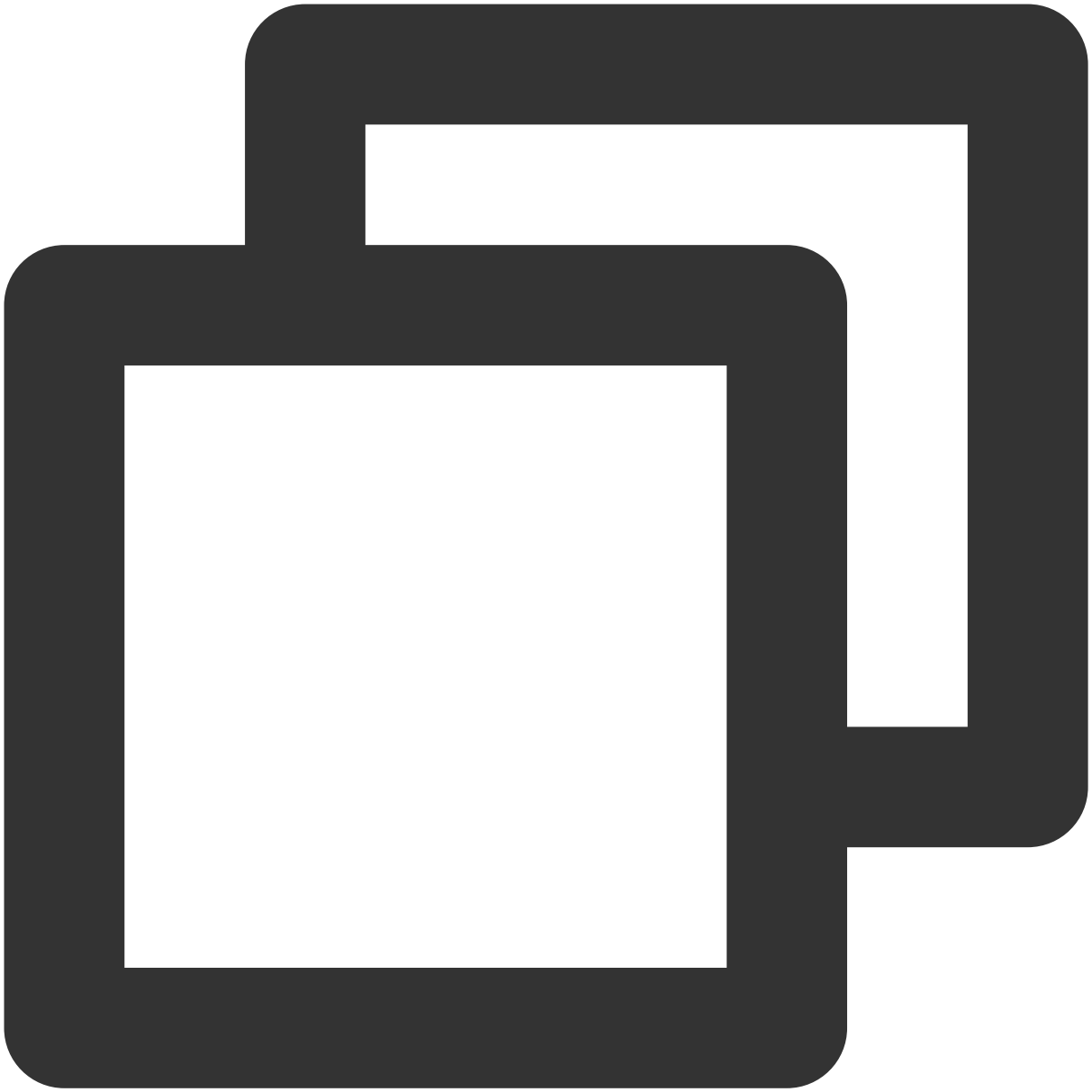
示例：



```
> SELECT instr('SparkSQL', 'SQL');
 6
```

LCASE

函数语法：



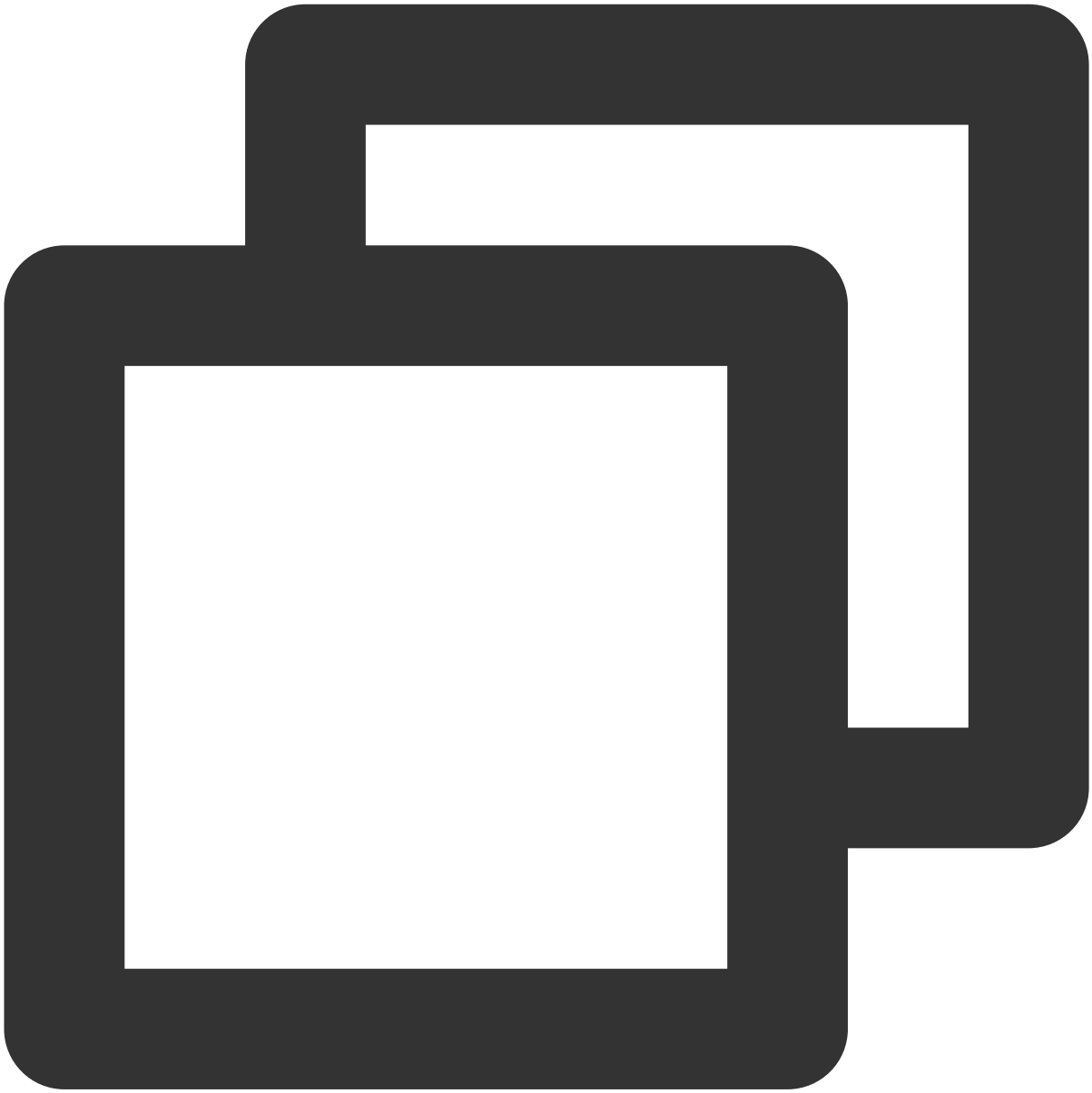
```
LCASE(<str> string)
```

支持引擎：SparkSQL、Presto。

使用说明：str 改为小写。

返回类型：string。

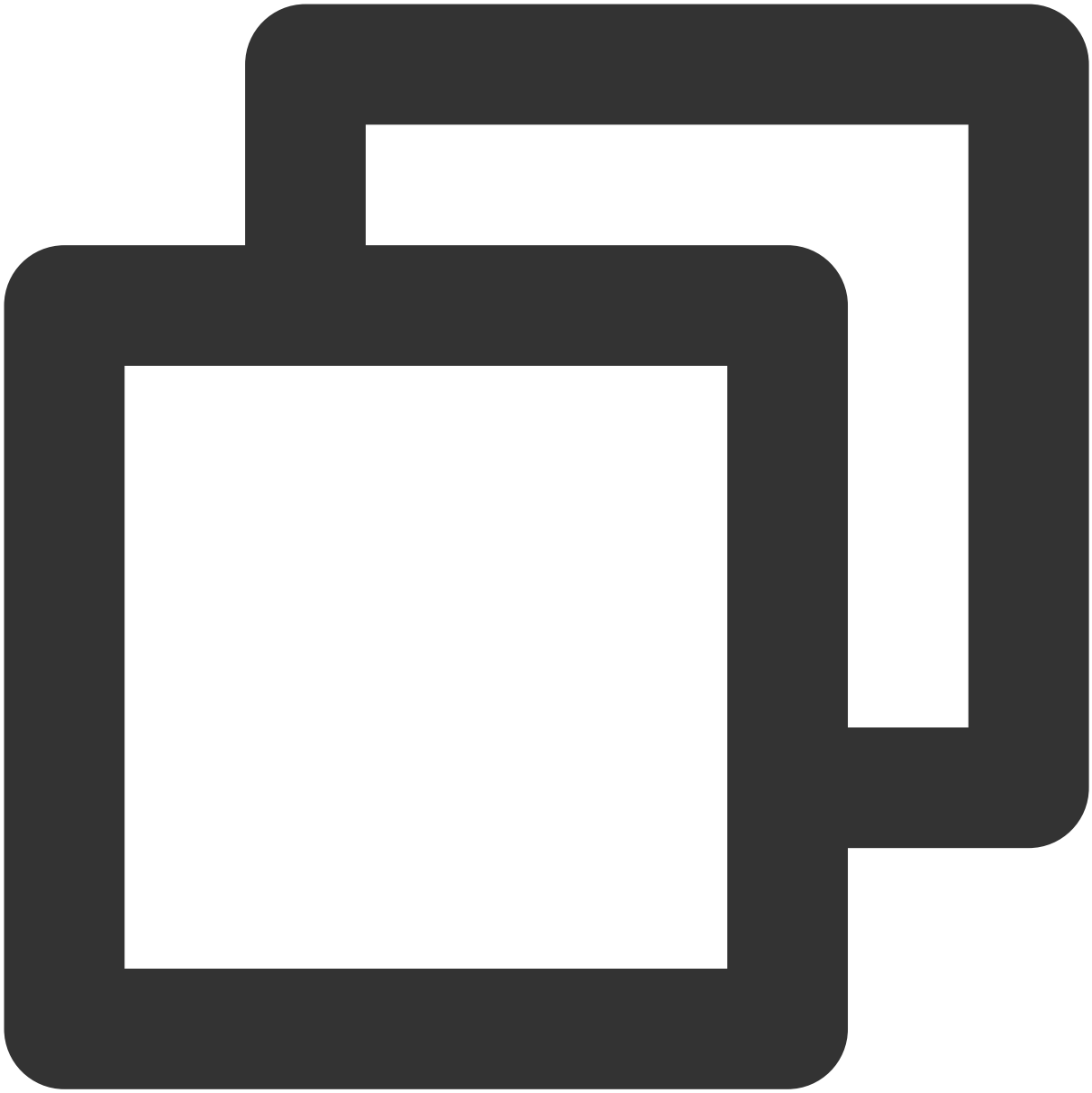
示例：



```
> SELECT lcase('SparkSQL');
 sparksql
```

LENGTH

函数语法：



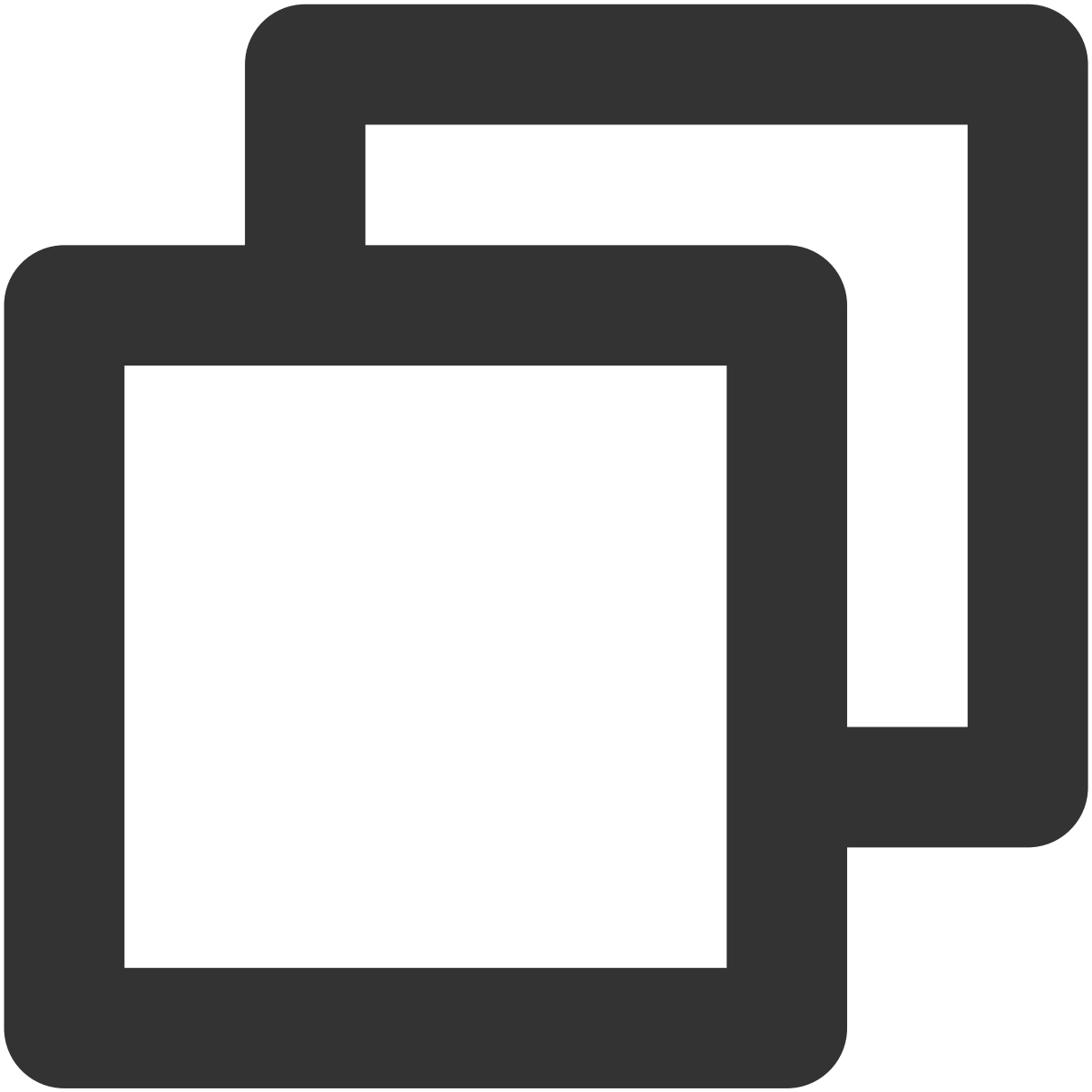
```
LENGTH(<expr> string|binary)
```

支持引擎：SparkSQL、Presto。

使用说明：返回字符串数据的字符长度或二进制数据的字节数。字符串数据的长度包括最后的空格。二进制数据的长度包括二进制零。

返回类型：integer。

示例：



```
> SELECT length('Spark SQL ');
 10
```

LEVENSHTEIN

函数语法：

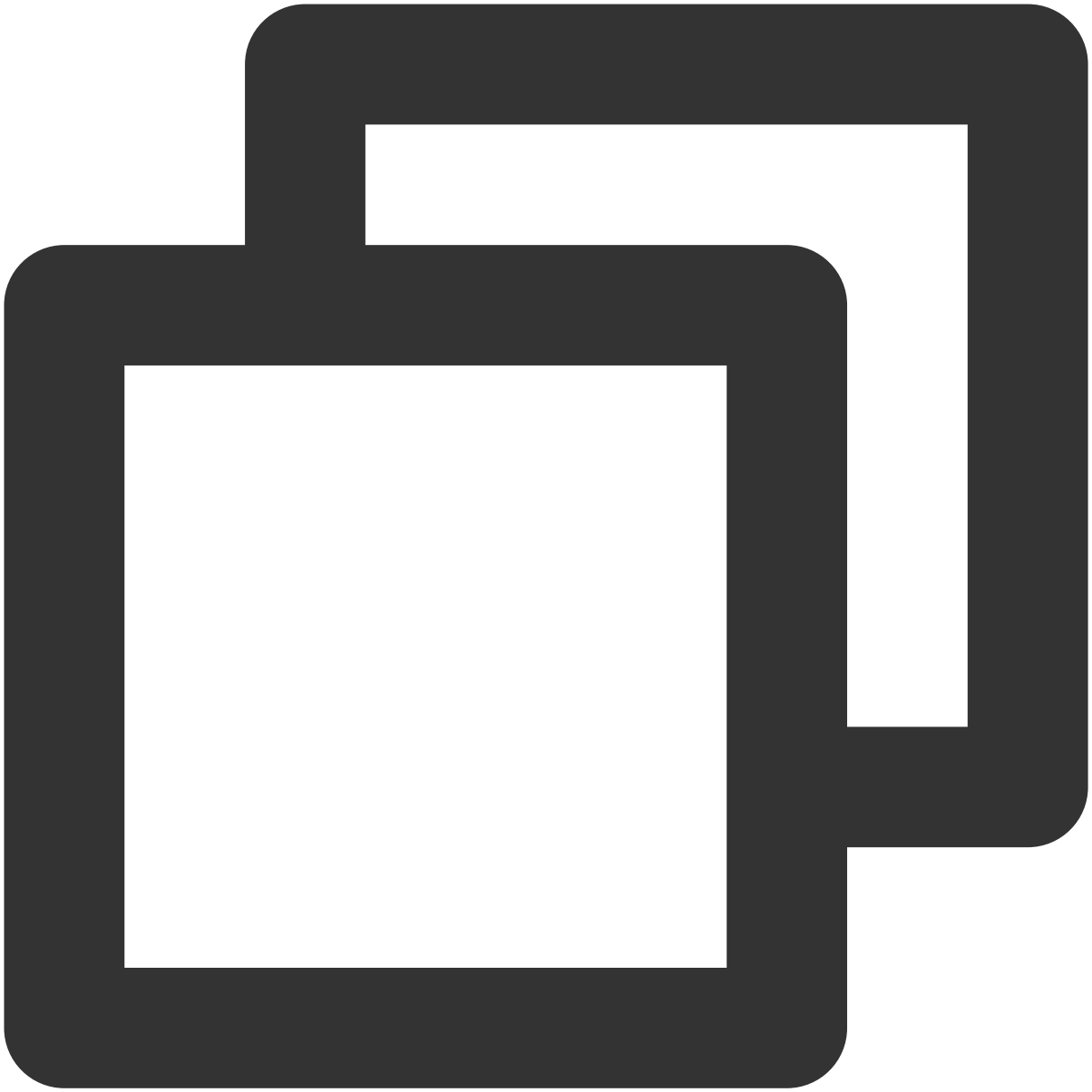
```
LEVENSHTEIN(<s1> string, <s2> string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回两个给定字符串之间的 Levenshtein 距离。

返回类型：integer。

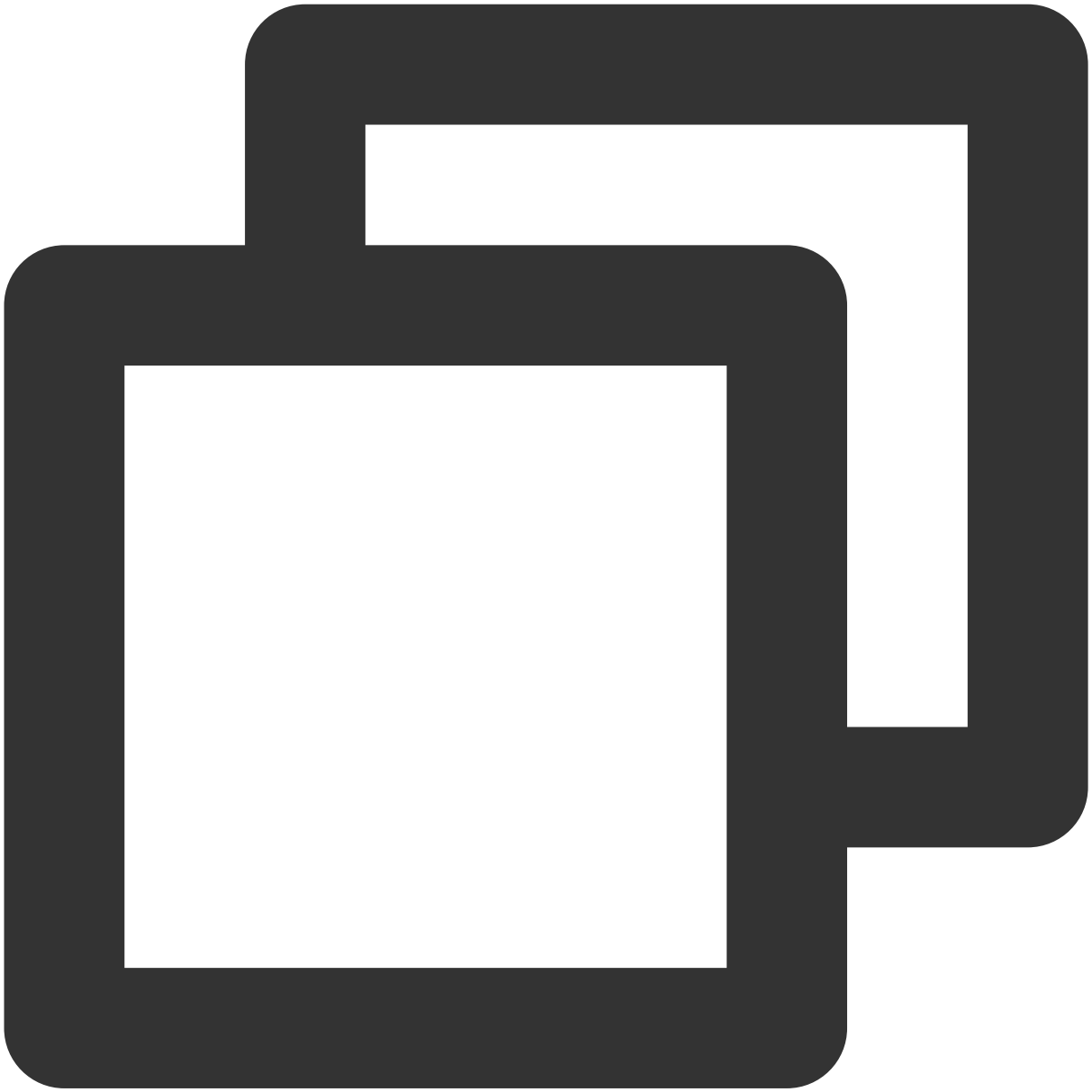
示例：



```
> SELECT levenshtein('kitten', 'sitting');
 3
```

LIKE

函数语法：



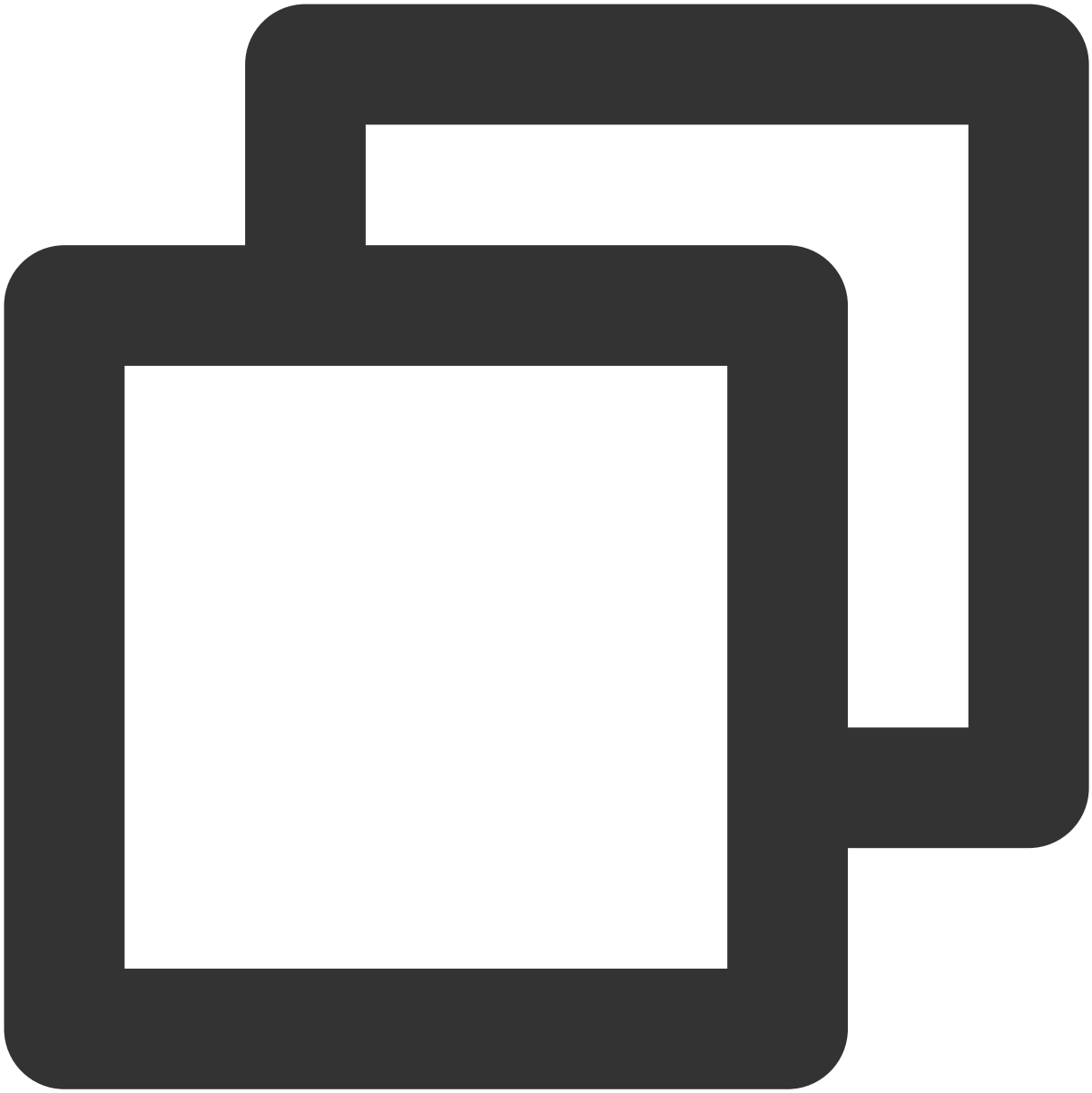
```
LIKE(<s1> str, <s2> pattern)
<str> like <pattern>[ ESCAPE <escape>]
```

支持引擎：SparkSQL、Presto。

使用说明：如果 str 匹配带转义 escape 的 pattern，则返回 true；如果任何参数为 null，则返回 null；否则返回 false。

返回类型：boolean。

示例：

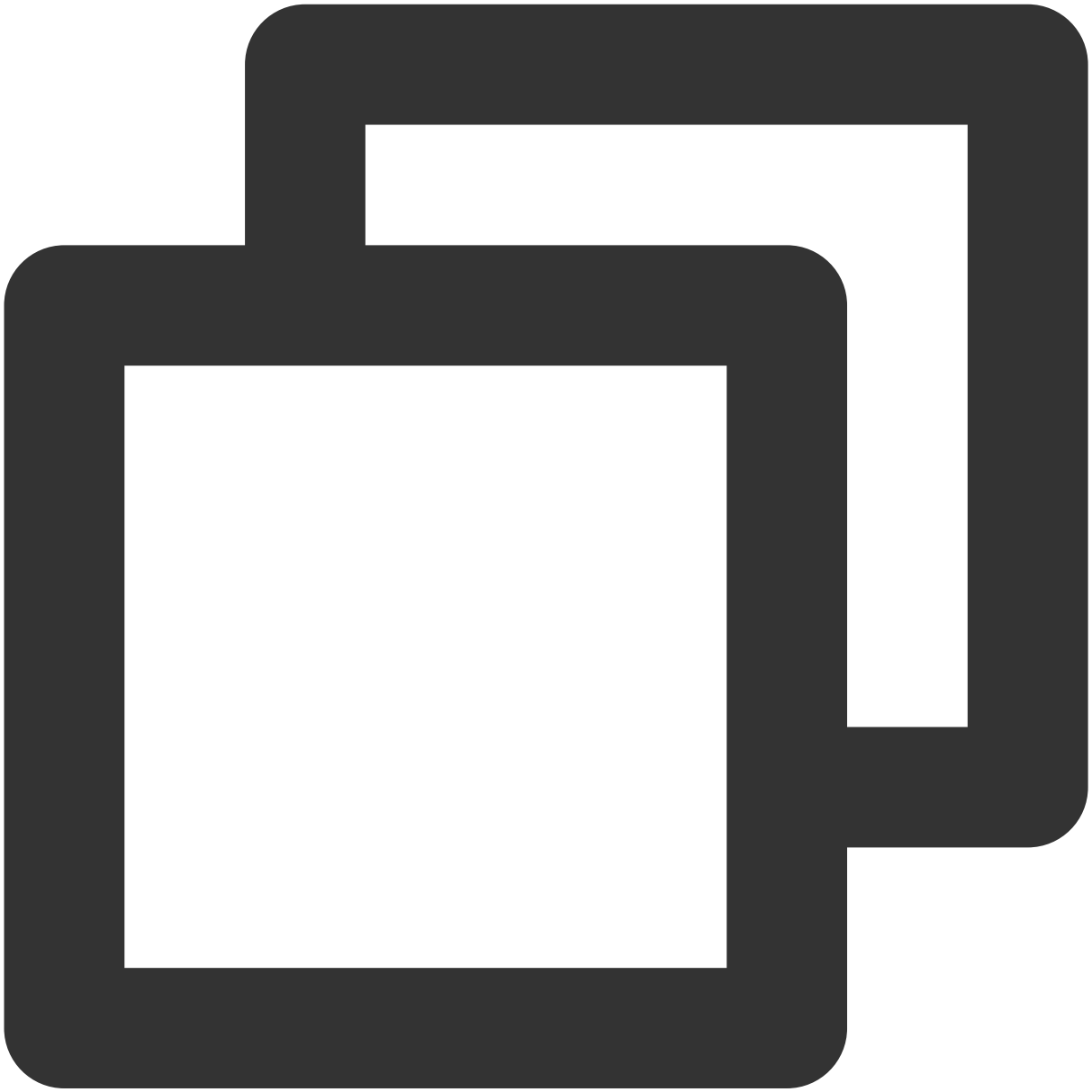


```
> SELECT like('Spark', '_park');
true
> SET spark.sql.parser.escapedStringLiterals=true;
spark.sql.parser.escapedStringLiterals  true
> SELECT '%SystemDrive%\\Users\\John' like '\\%SystemDrive\\%\\\\Users%';
true
> SET spark.sql.parser.escapedStringLiterals=false;
spark.sql.parser.escapedStringLiterals  false
> SELECT '%SystemDrive%\\\\Users\\\\John' like '\\%SystemDrive\\%\\\\\\\\Users%';
false
> SELECT '%SystemDrive%/Users/John' like '/%SystemDrive/%//Users%' ESCAPE '/';
```

```
true
```

LOWER

函数语法：



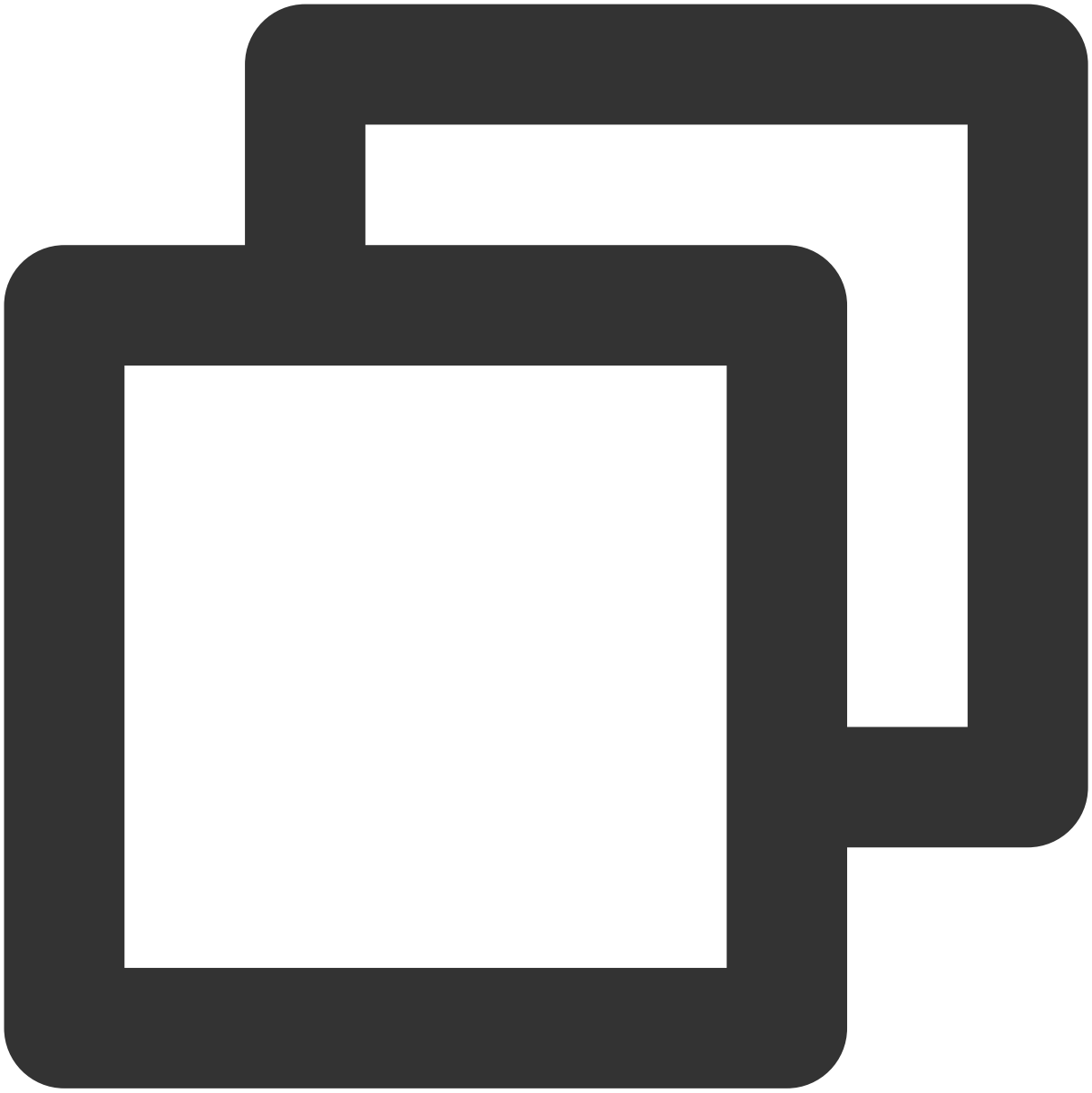
```
LOWER(<str> string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回所有字符都更改为小写的 str。

返回类型：string。

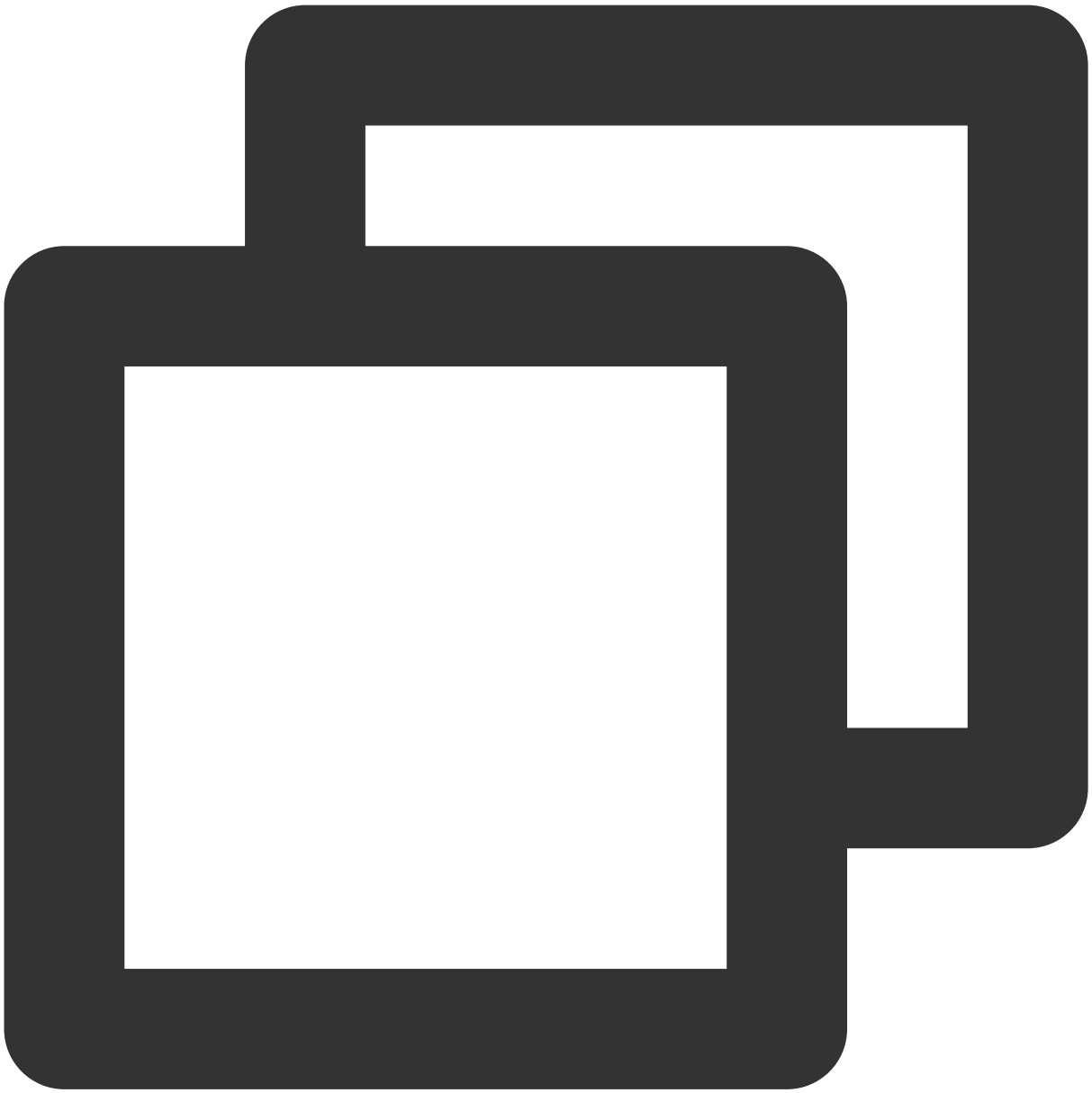
示例：



```
> SELECT lower('TENCENT');
 tencent
```

LOCATE

函数语法：



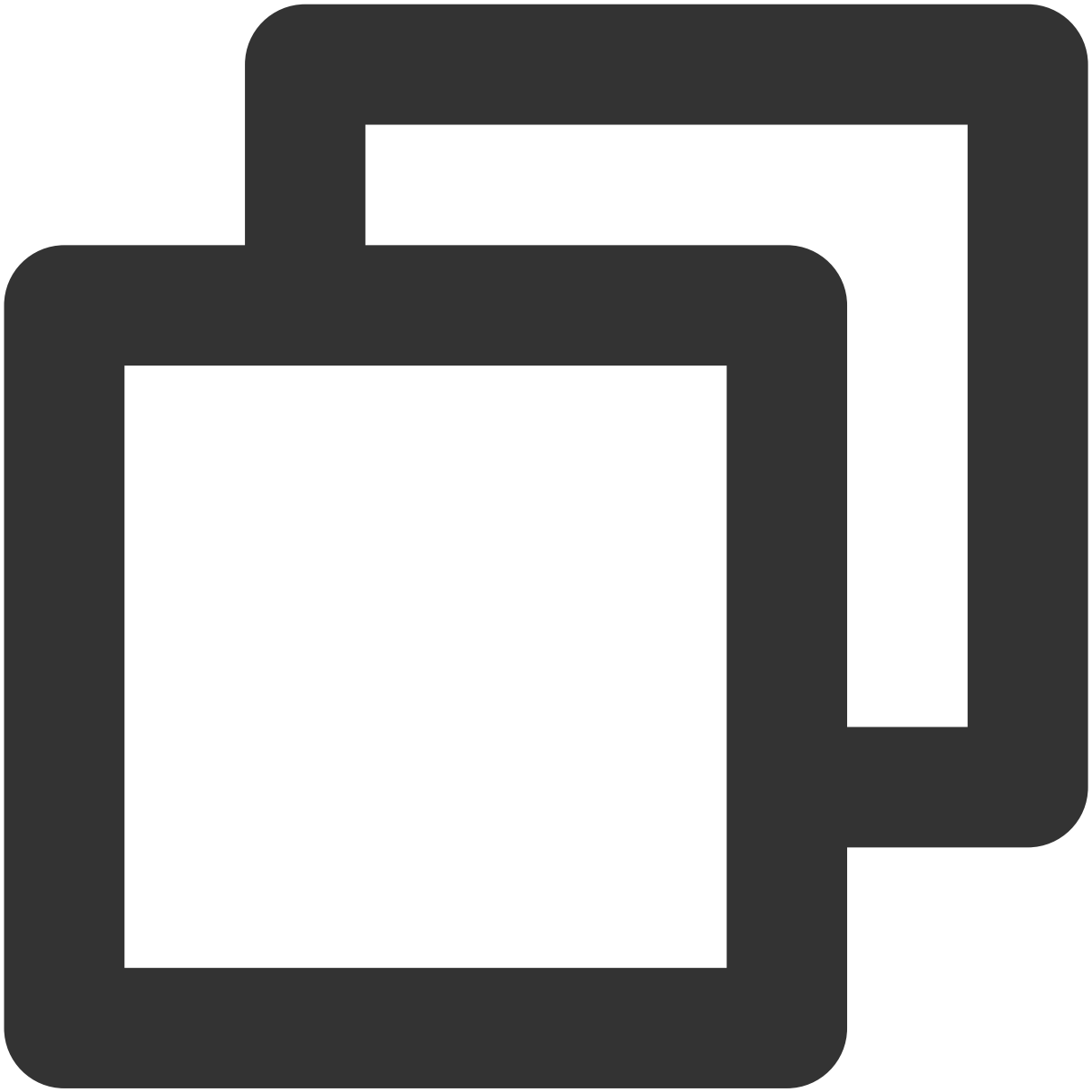
```
LOCATE(<substr> string, <str> string[, <pos> integer])
```

支持引擎：SparkSQL、Presto。

使用说明：返回在 str 中第 pos 位之后中第一次出现 substr 的位置。

返回类型：integer。

示例：



```
> SELECT locate('bar', 'foobarbar');
 4
> SELECT locate('bar', 'foobarbar', 5);
 7
```

OCTET_LENGTH

函数语法：

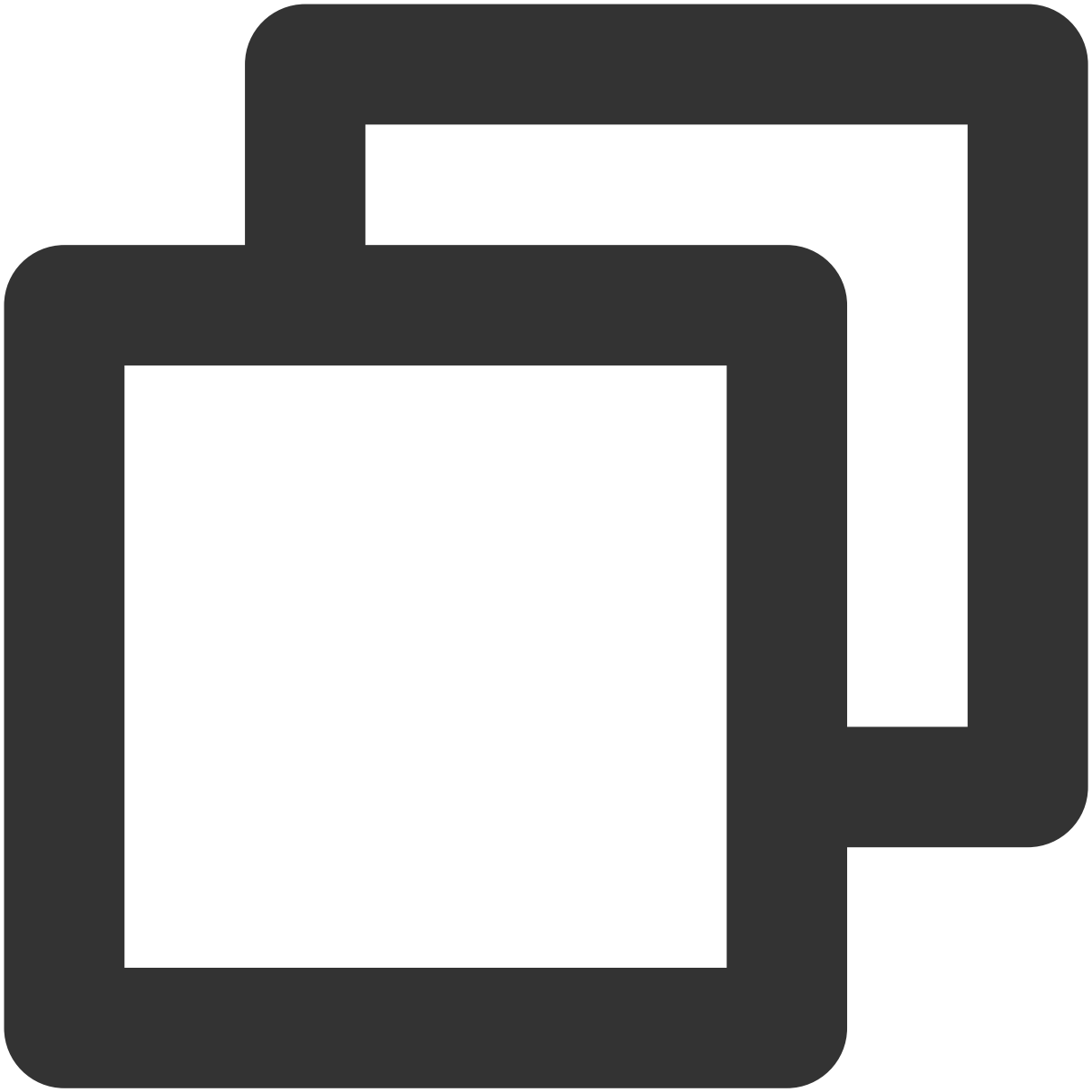
```
OCTET_LENGTH(<expr> string|binary)
```

支持引擎：SparkSQL、Presto。

使用说明：返回字符串数据的字节长度或二进制数据的字节数。

返回类型：integer。

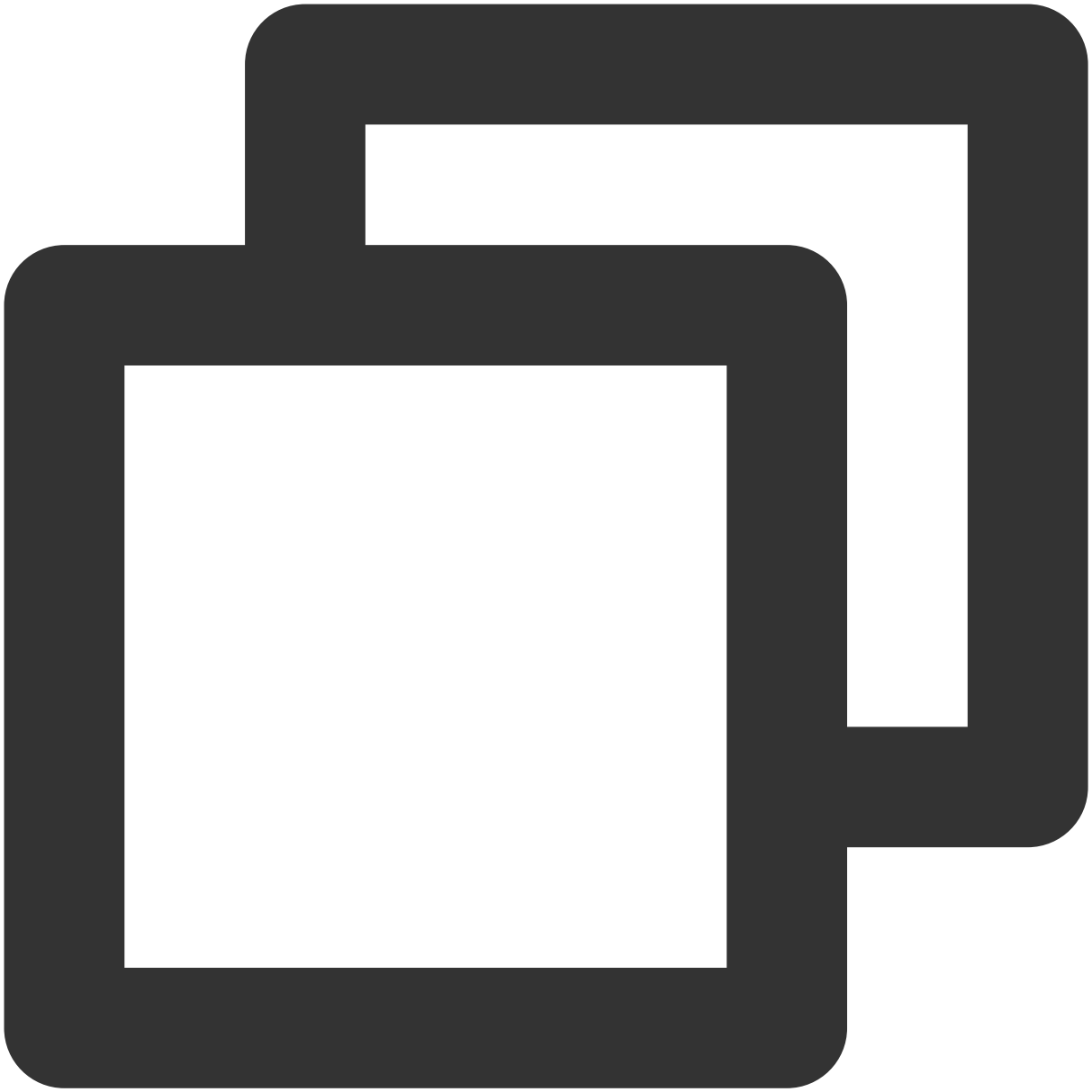
示例：



```
> SELECT octet_length('Spark SQL');
 9
```

LPAD

函数语法：



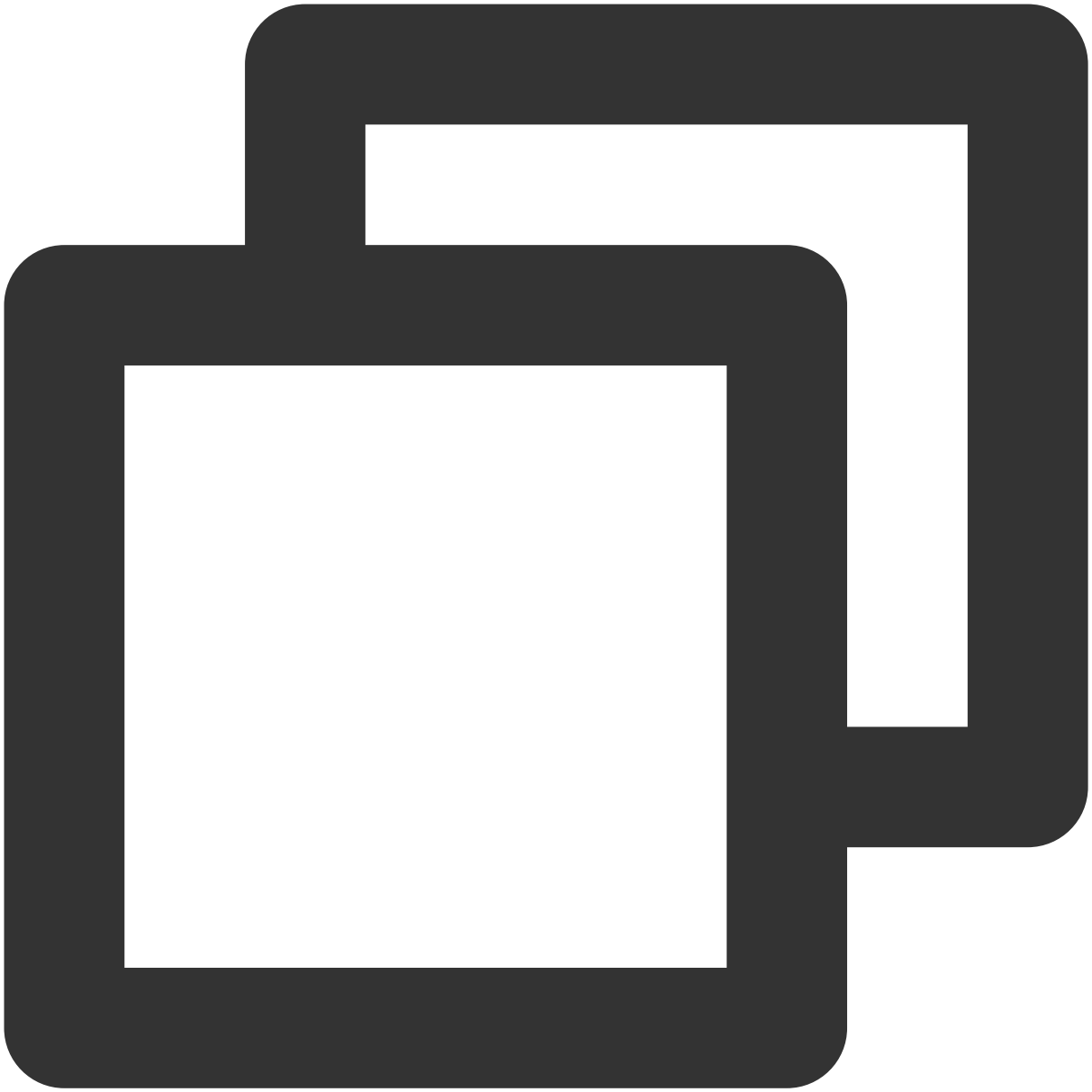
```
LPAD(<str> string, <len> integer[, <pad> string])
```

支持引擎：SparkSQL、Presto。

使用说明：返回 str，在左填充 pad 到长度 len。如果 str 长于 len，则返回值缩短为 len 个字符。如果未指定 pad，str 将用空格字符填充。

返回类型：string。

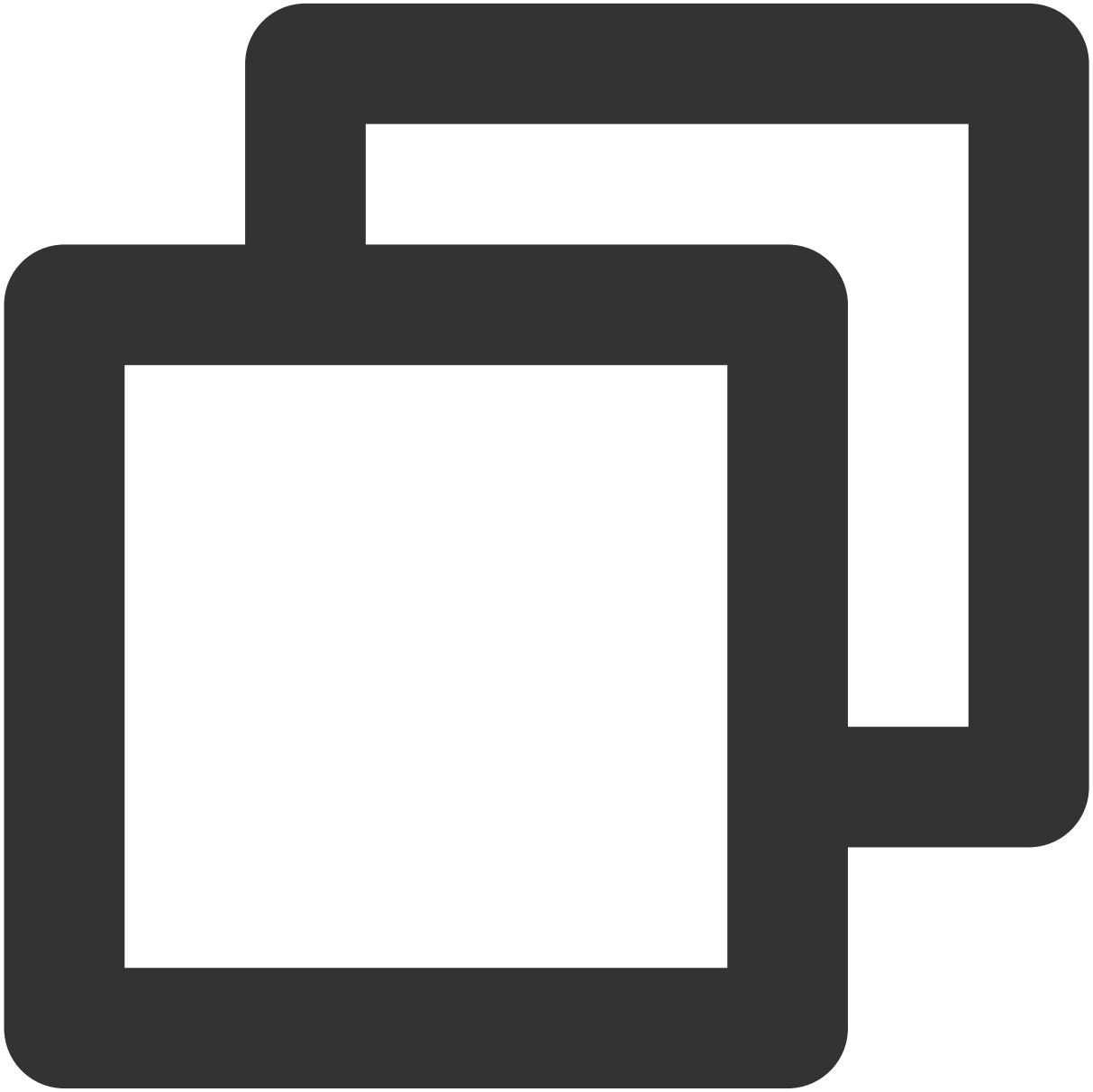
示例：



```
> SELECT lpad('hi', 5, '??');
 ???hi
> SELECT lpad('hi', 1, '??');
 h
> SELECT lpad('hi', 5);
  hi
```

LTRIM

函数语法：



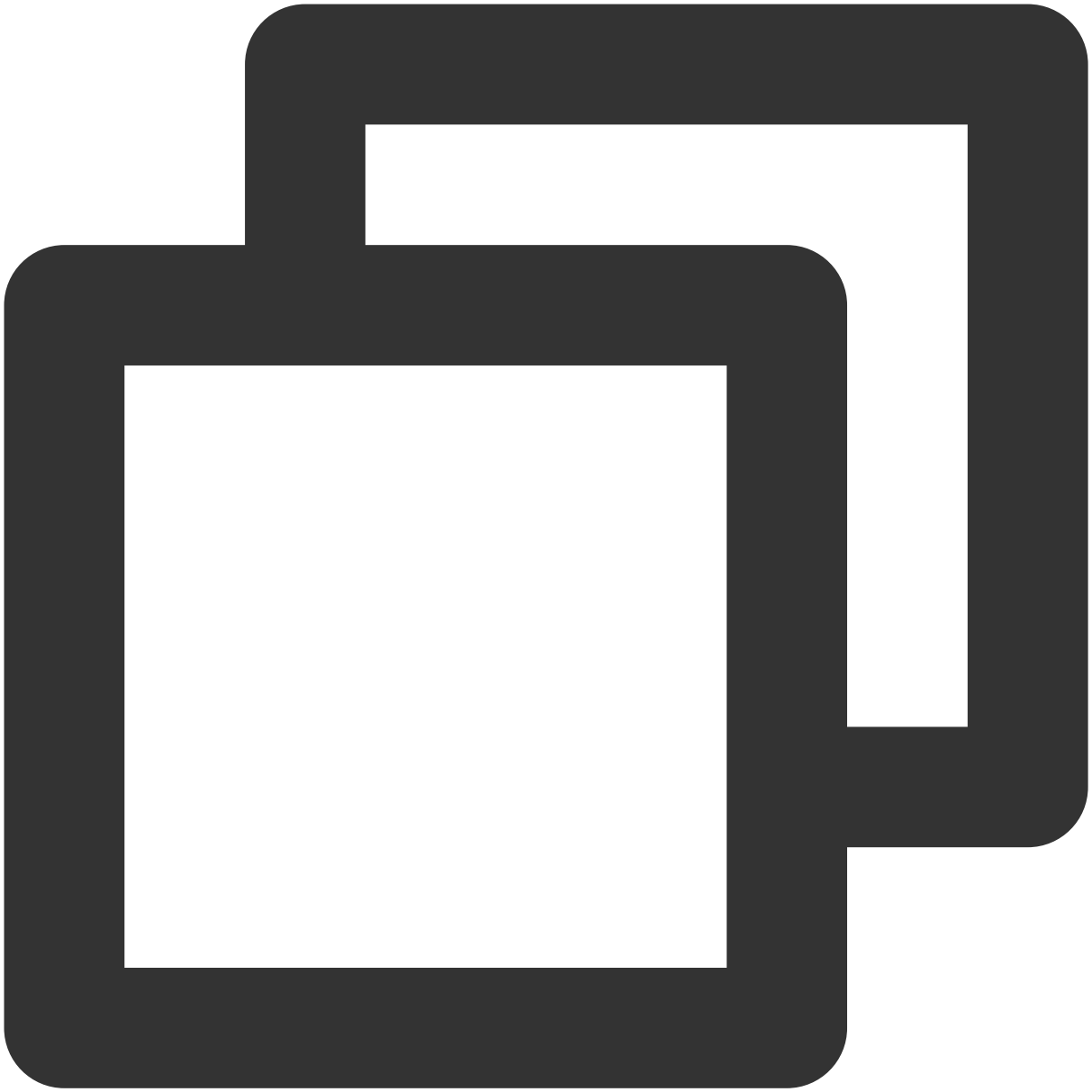
```
LTRIM(<str> string)
```

支持引擎：SparkSQL、Presto。

使用说明：从 str 中删除前导空格字符。

返回类型：string。

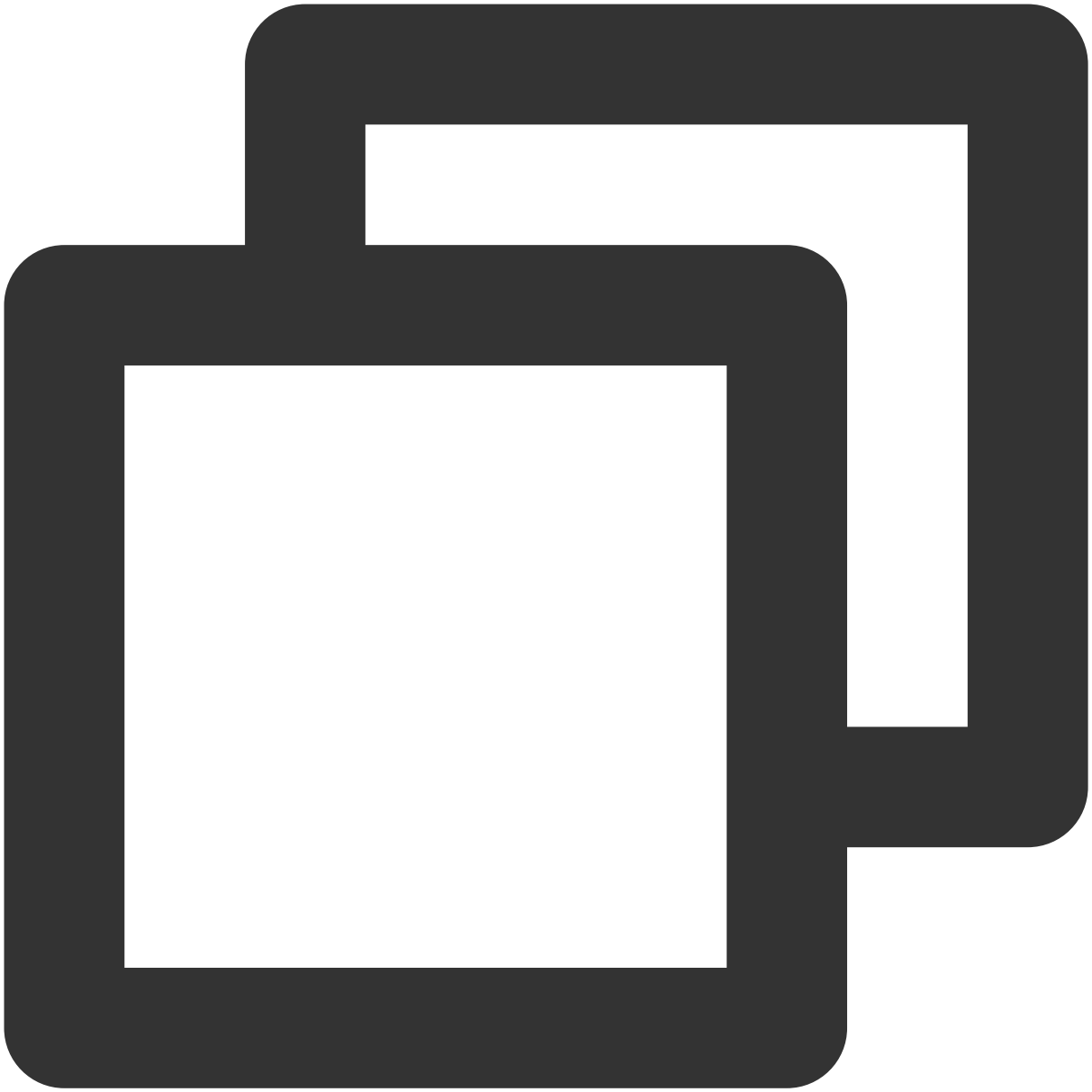
示例：



```
> SELECT ltrim('    SparkSQL   ');
 SparkSQL
```

PARSE_URL

函数语法：



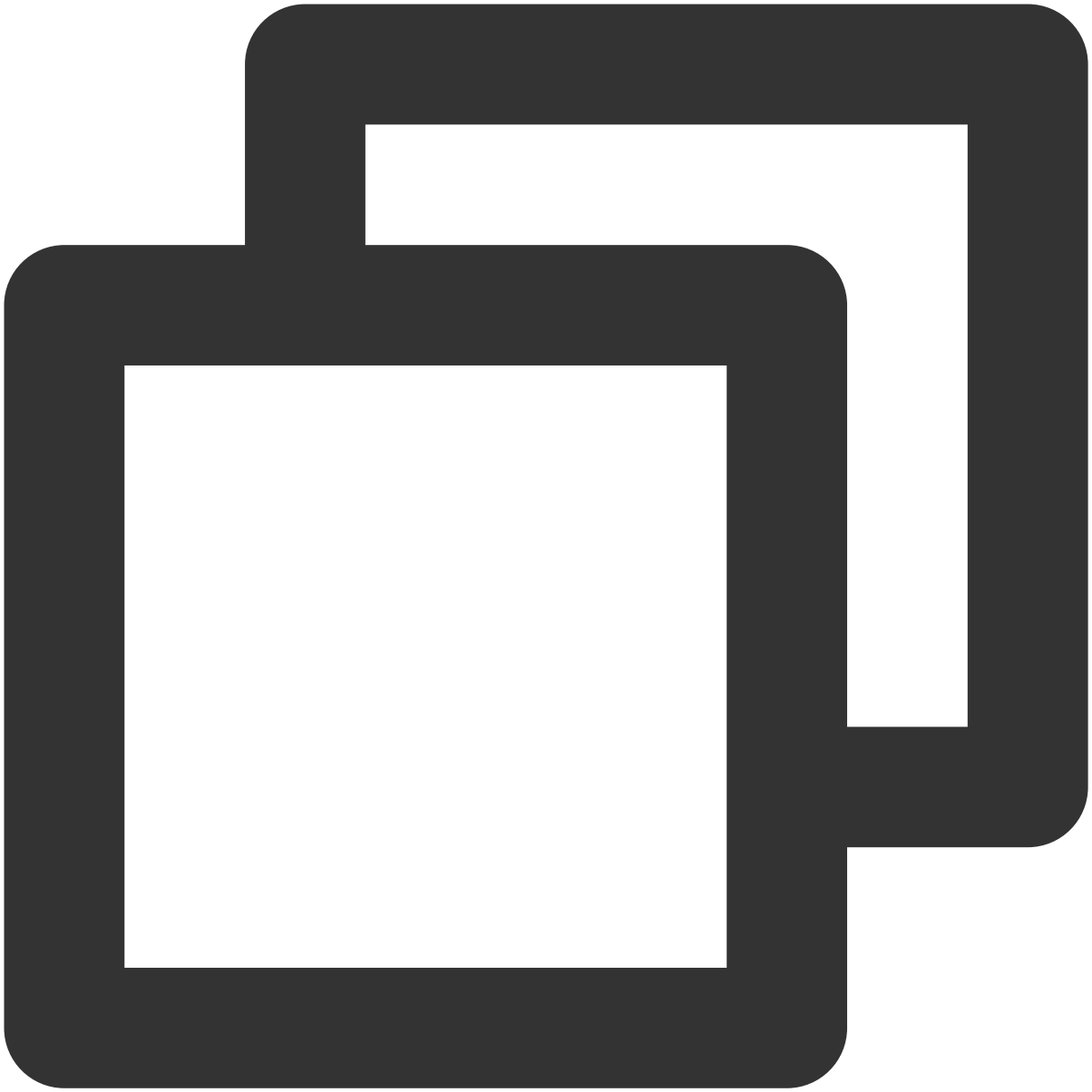
```
PARSE_URL(<url> string, <path> string[, <key> string])
```

支持引擎：SparkSQL、Presto。

使用说明：从 url 中提取 path。

返回类型：string。

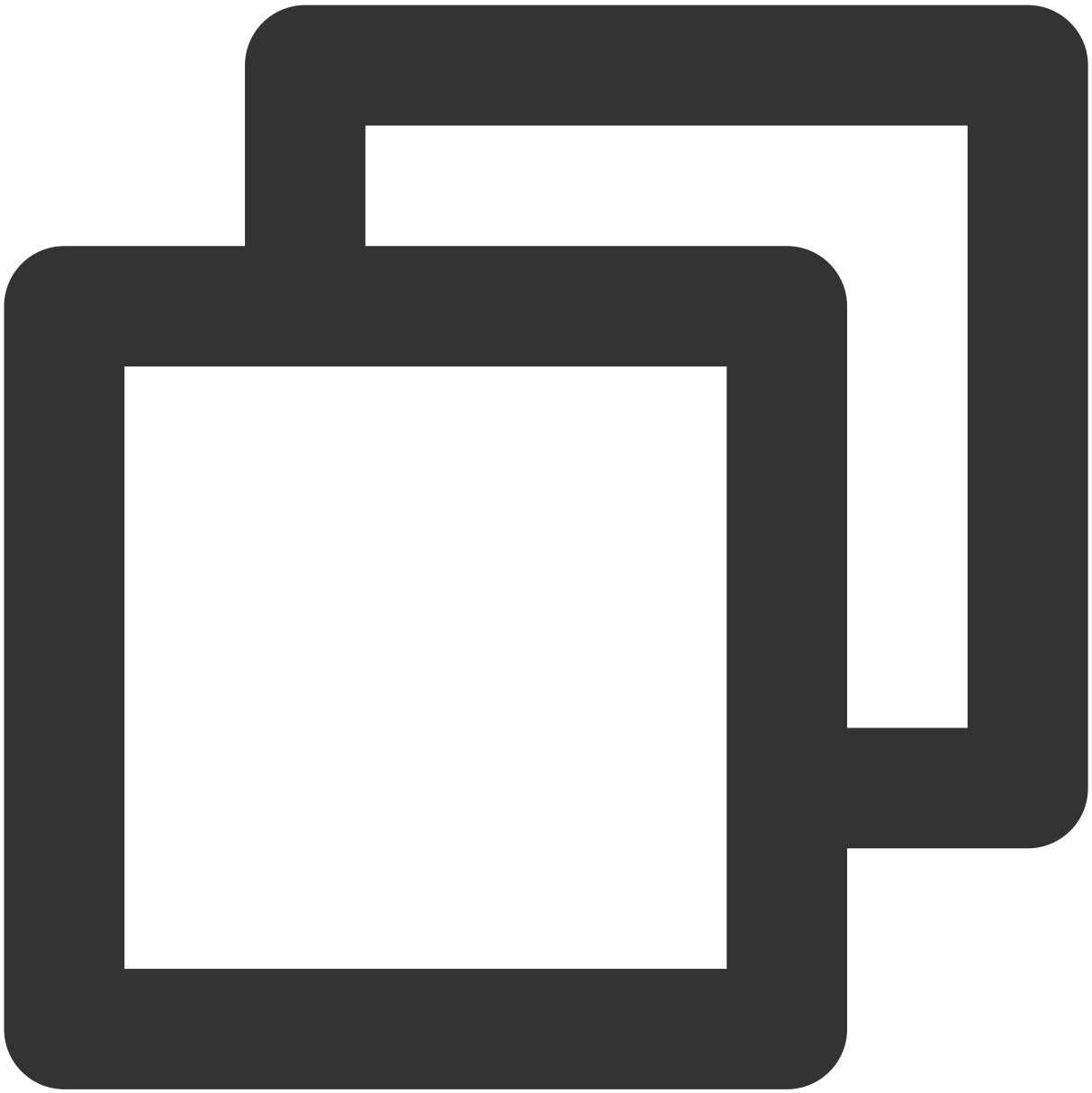
示例：



```
> SELECT parse_url('http://spark.apache.org/path?query=1', 'HOST');
 spark.apache.org
> SELECT parse_url('http://spark.apache.org/path?query=1', 'QUERY');
 query=1
> SELECT parse_url('http://spark.apache.org/path?query=1', 'QUERY', 'query');
 1
```

POSITION

函数语法：



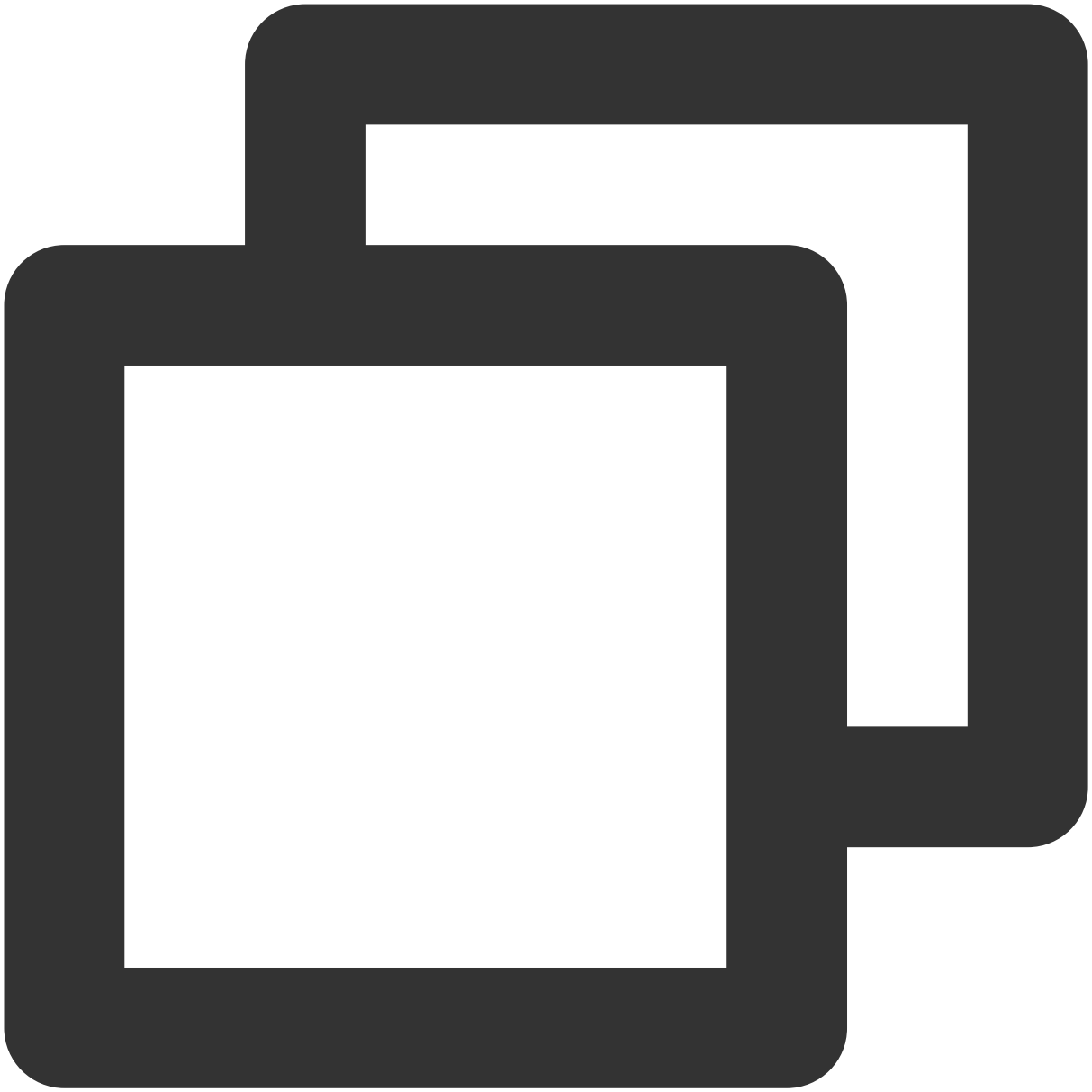
```
POSITION(<substr> string, <str> string[, <pos> integer])
```

支持引擎：SparkSQL、Presto。

使用说明：返回在 str 中第 pos 位之后中第一次出现 substr 的位置。

返回类型：integer。

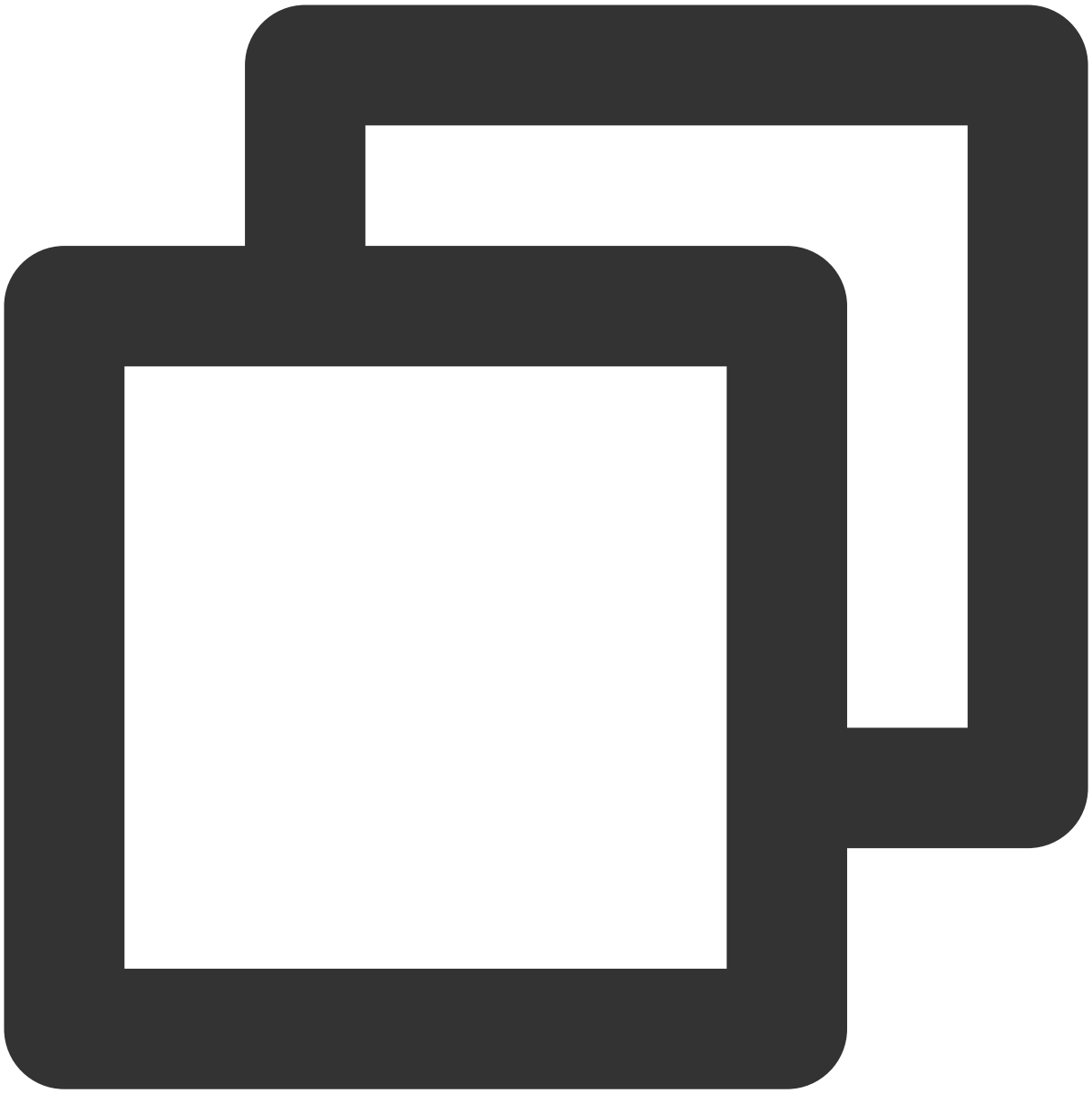
示例：



```
> SELECT position('bar', 'foobarbar');
 4
> SELECT position('bar', 'foobarbar', 5);
 7
> SELECT POSITION('bar' IN 'foobarbar');
 4
```

PRINTF

函数语法：



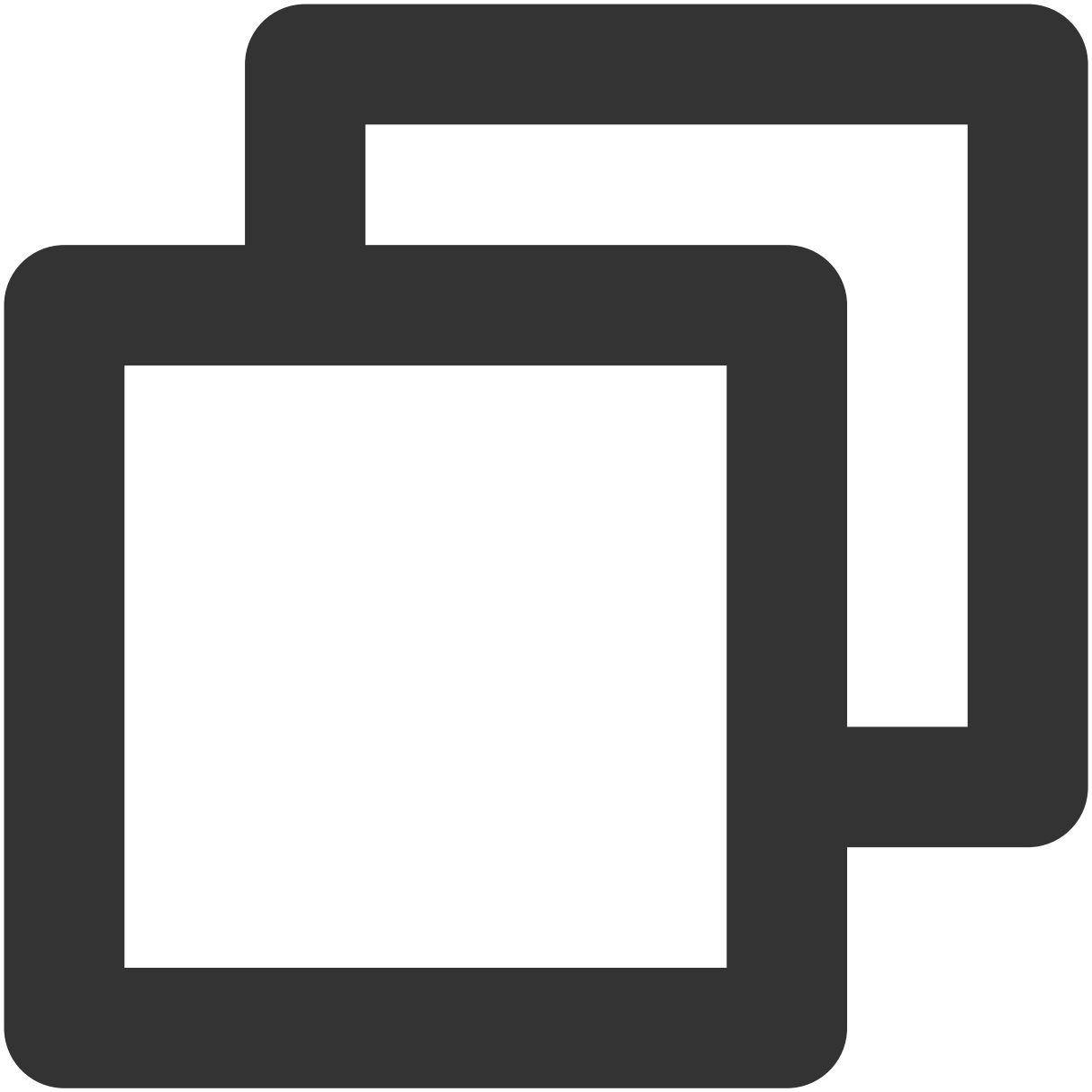
```
PRINTF(<str> string, obj <T>, ...)
```

支持引擎：SparkSQL、Presto。

使用说明：返回 printf 样式格式字符串中的格式化字符串。

返回类型：string。

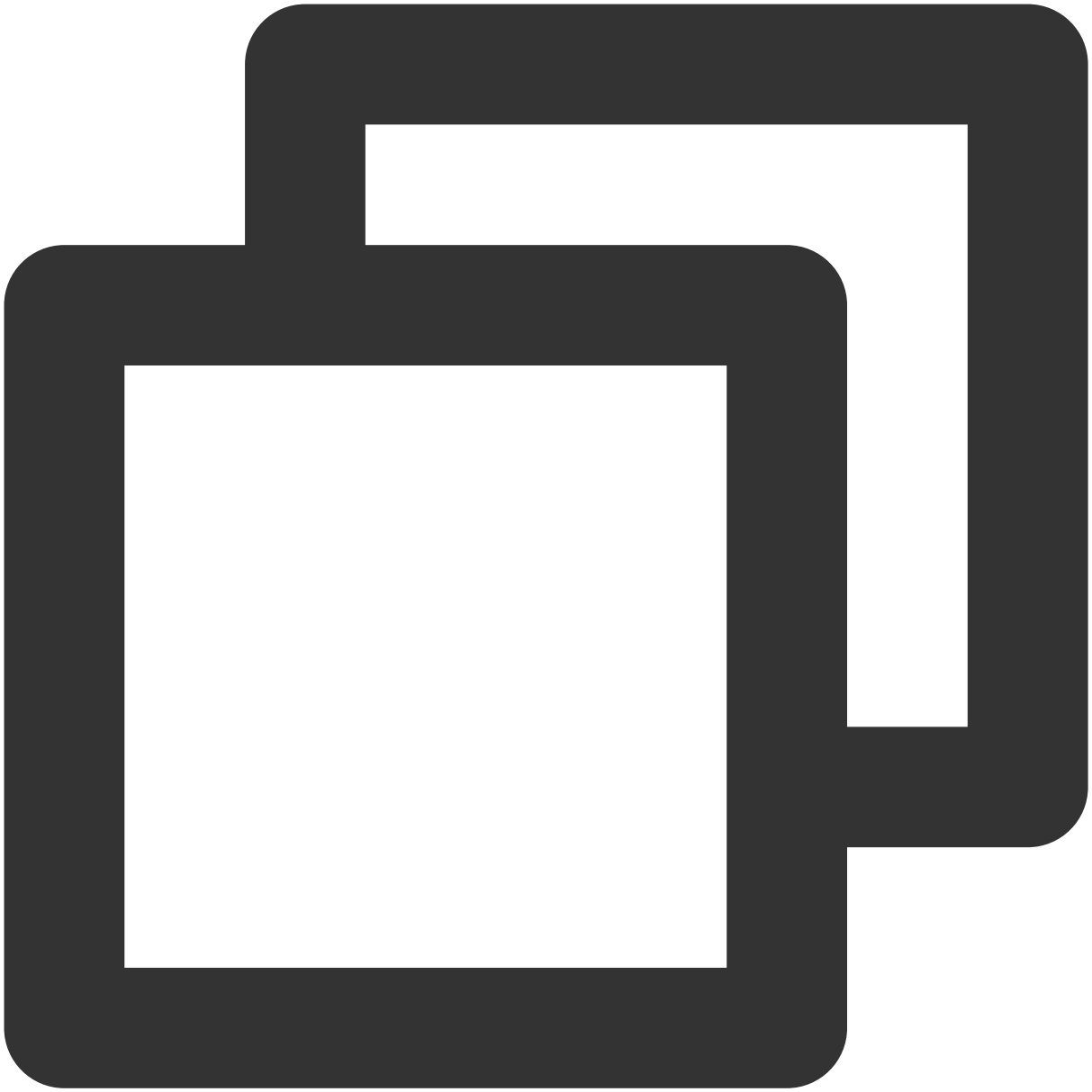
示例：



```
> SELECT printf("Hello World %d %s", 100, "days");
 Hello World 100 days
```

REPEAT

函数语法：



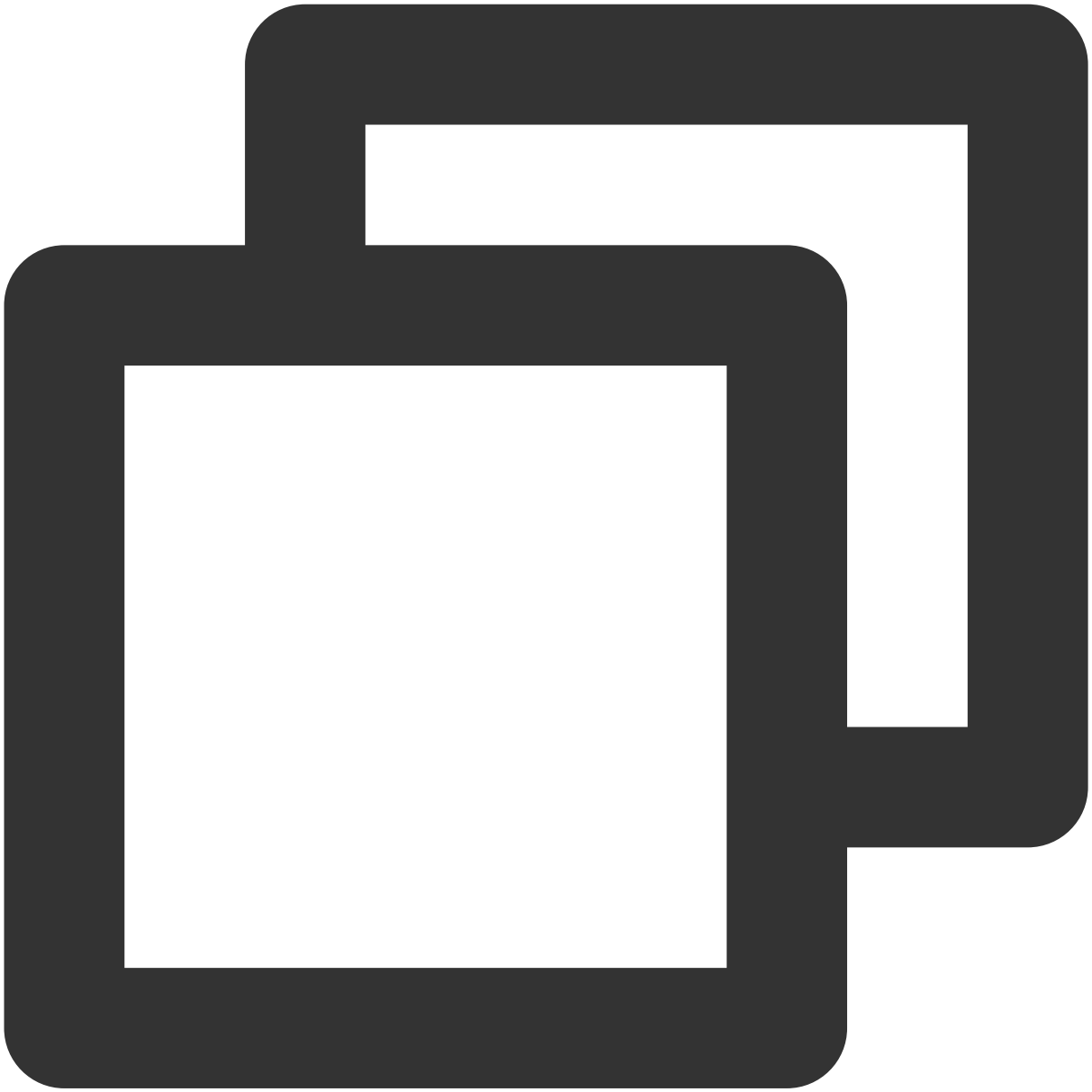
```
REPEAT(<str> string, <n> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：返回将给定字符串重复 n 次的字符串。

返回类型：string。

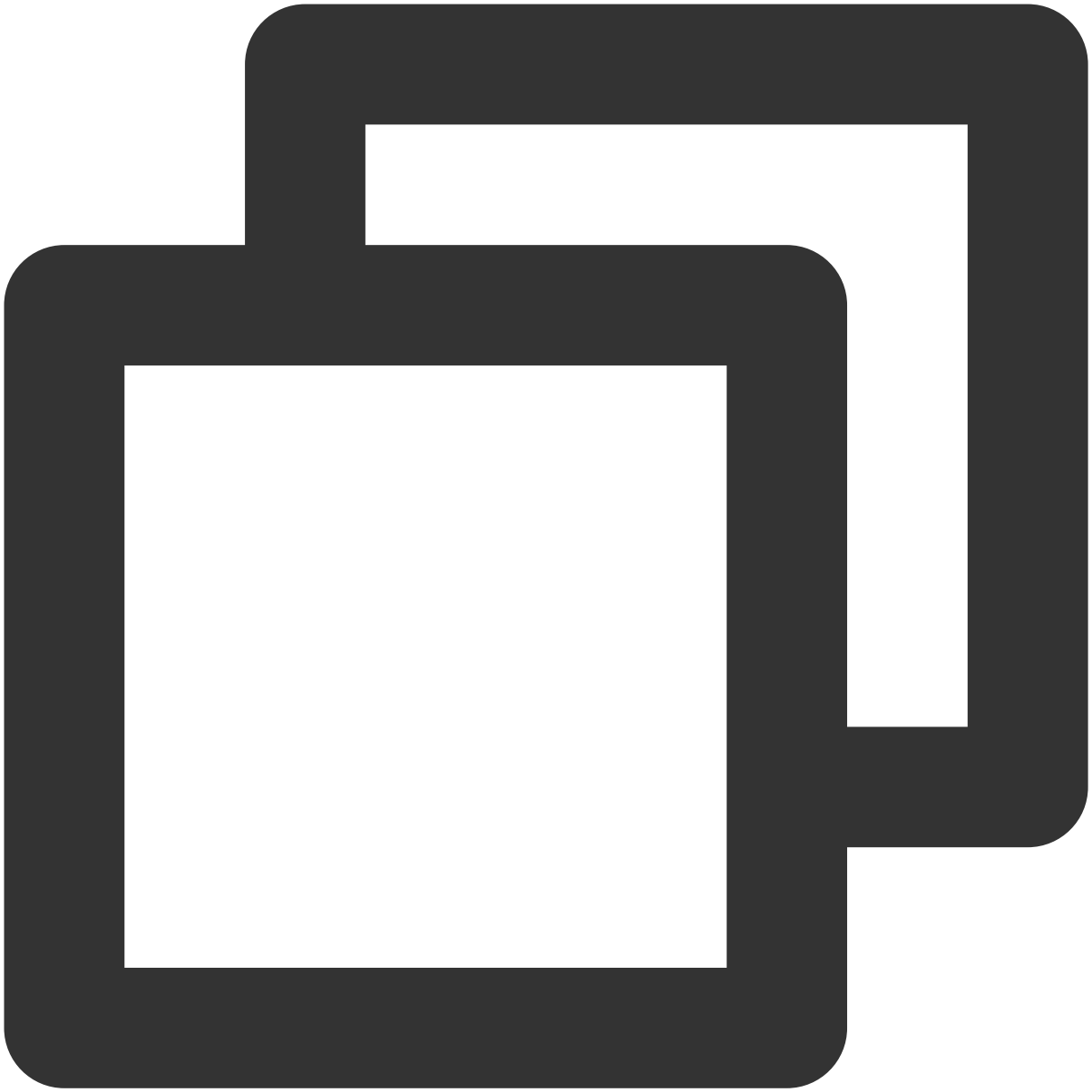
示例：



```
> SELECT repeat('123', 2);
 123123
```

REPLACE

函数语法：



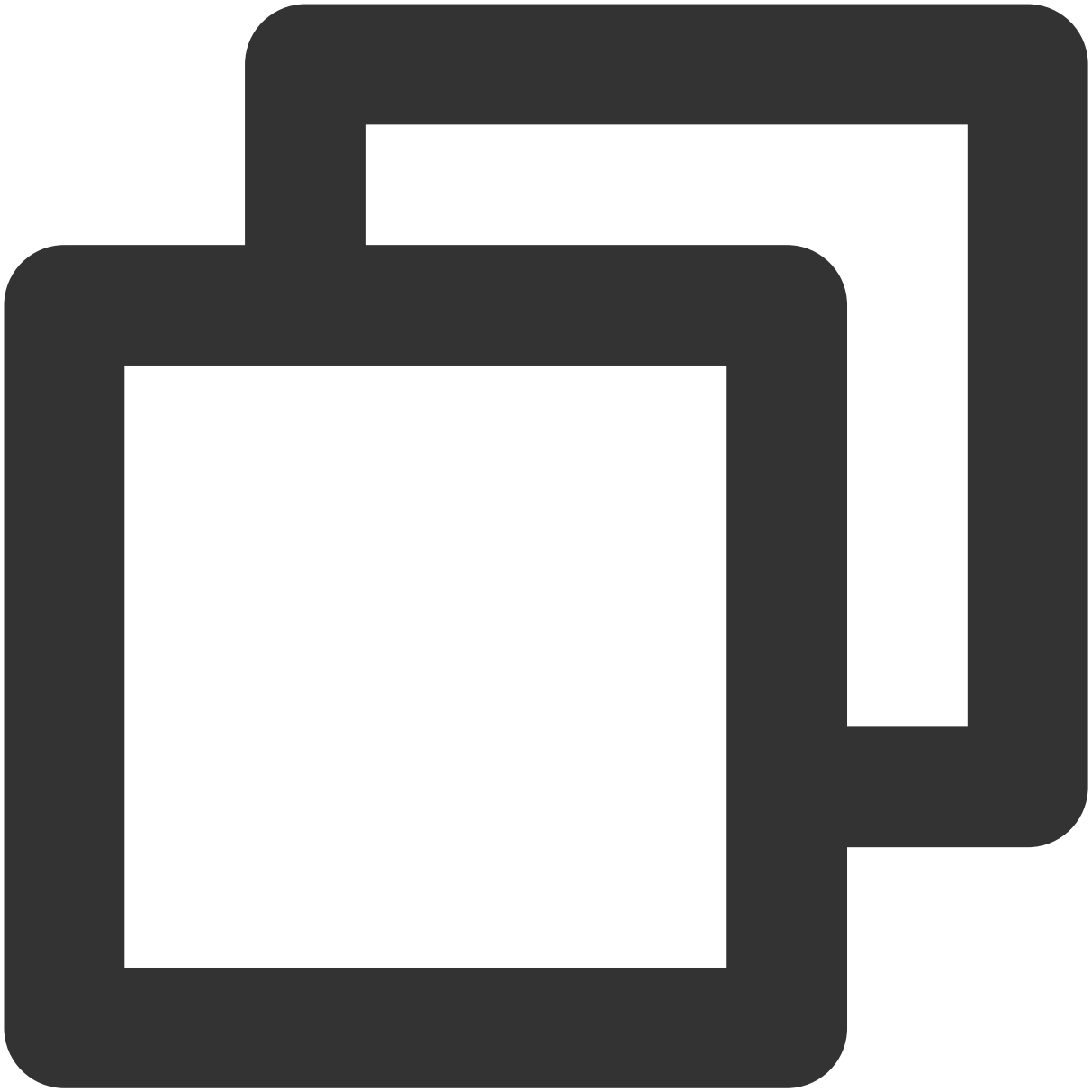
```
REPLACE(<str> string, <search> string[, <replace> string])
```

支持引擎：SparkSQL、Presto。

使用说明：用 replace 替换 str 中所有出现的 search。

返回类型：string。

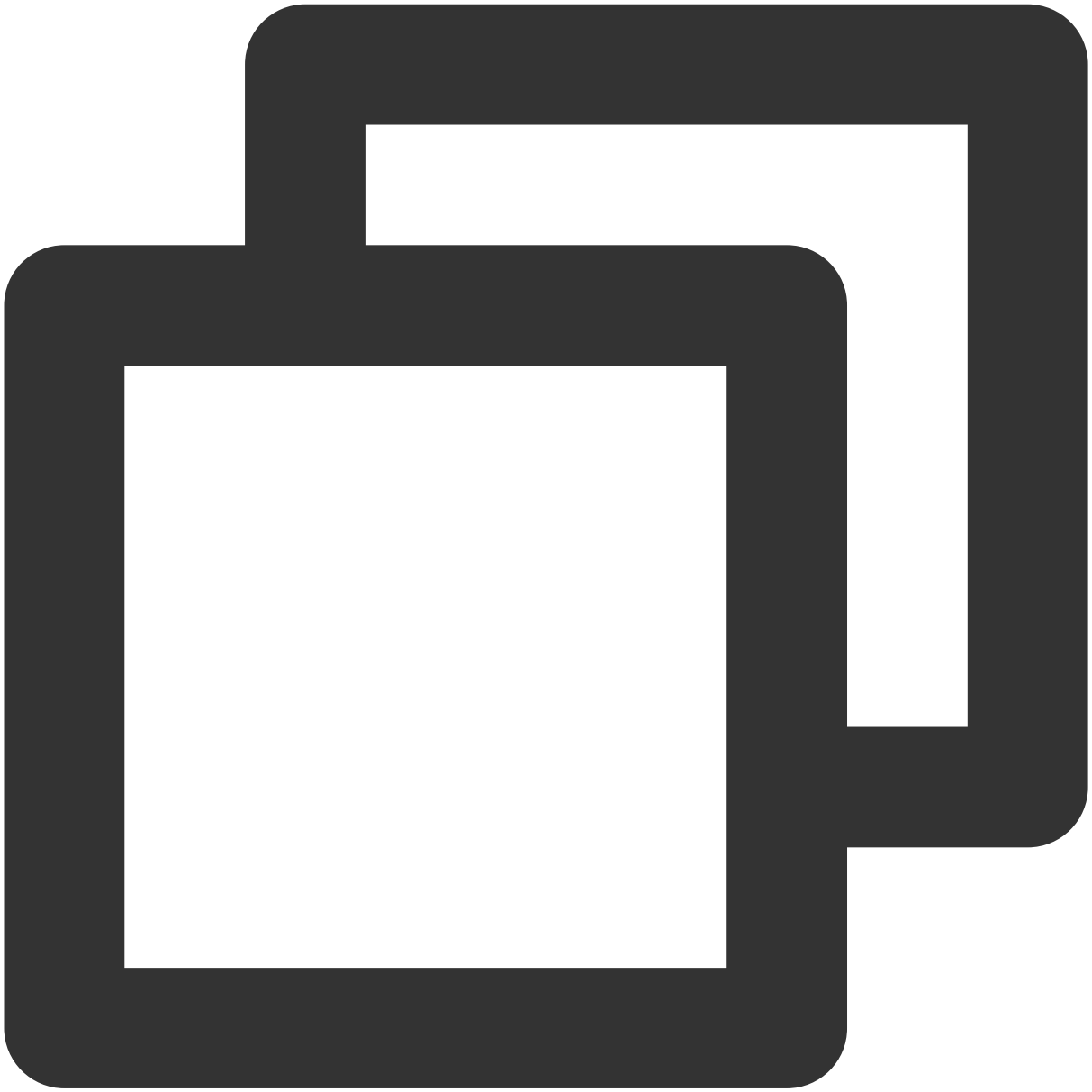
示例：



```
> SELECT replace('ABCabc', 'abc', 'DEF');
 ABCDEF
```

OVERLAY

函数语法：



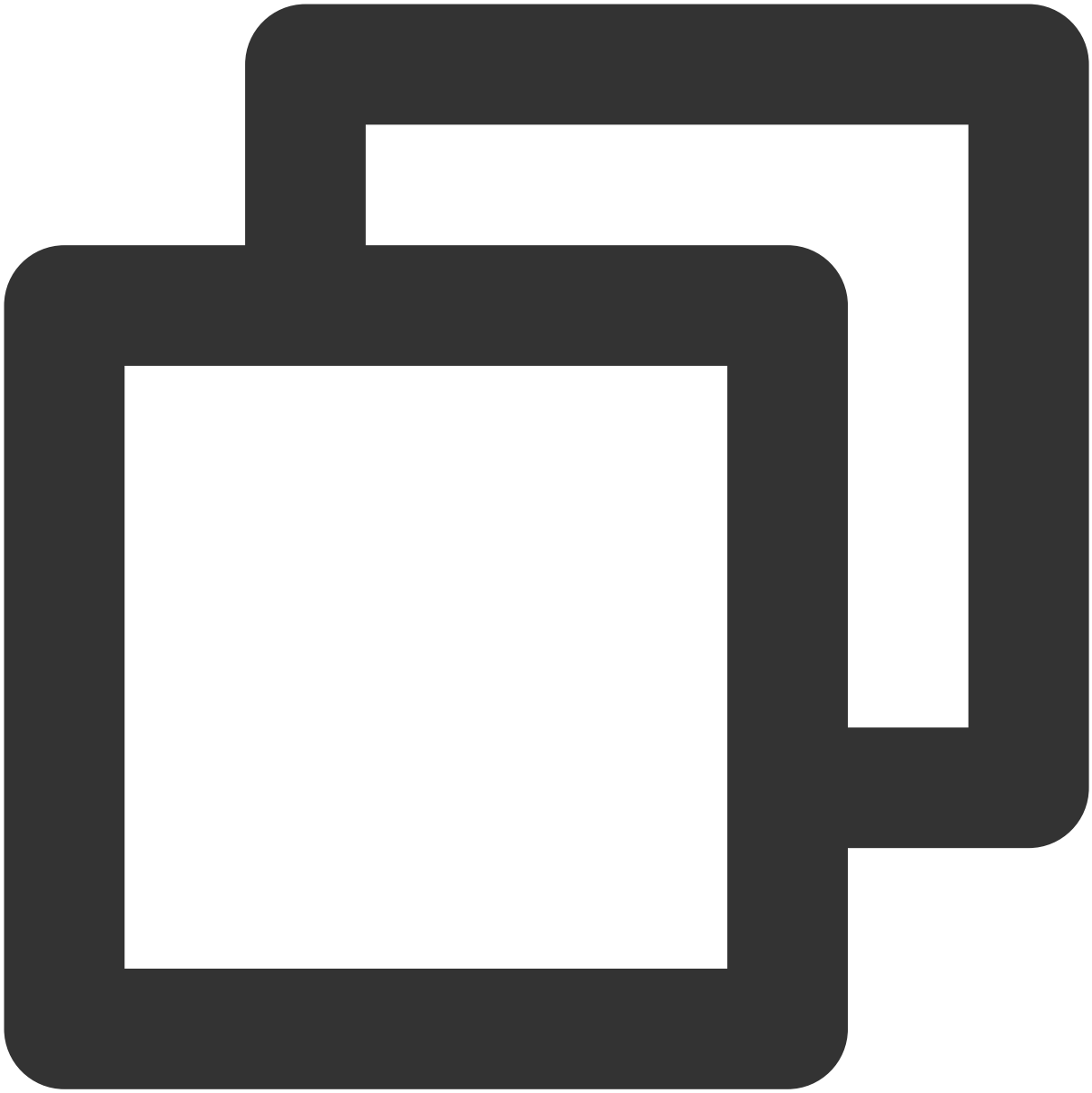
```
OVERLAY(<input> string, <replace> string, <pos> integer[, <len> integer])
```

支持引擎：SparkSQL。

使用说明：将 input 替换为从 pos 开始、长度为 len 的 replace。

返回类型：string。

示例：



```
> SELECT overlay('Spark SQL' PLACING '_' FROM 6);
 Spark_SQL
> SELECT overlay('Spark SQL' PLACING 'CORE' FROM 7);
 Spark CORE
> SELECT overlay('Spark SQL' PLACING 'ANSI ' FROM 7 FOR 0);
 Spark ANSI SQL
> SELECT overlay('Spark SQL' PLACING 'tructured' FROM 2 FOR 4);
 Structured SQL
> SELECT overlay(encode('Spark SQL', 'utf-8') PLACING encode('_', 'utf-8') FROM 6);
 Spark_SQL
> SELECT overlay(encode('Spark SQL', 'utf-8') PLACING encode('CORE', 'utf-8') FROM
```

```
 Spark CORE
> SELECT overlay(encode('Spark SQL', 'utf-8') PLACING encode('ANSI ', 'utf-8') FROM
 Spark ANSI SQL
> SELECT overlay(encode('Spark SQL', 'utf-8') PLACING encode('tructured', 'utf-8')
 Structured SQL
```

RPAD

函数语法：

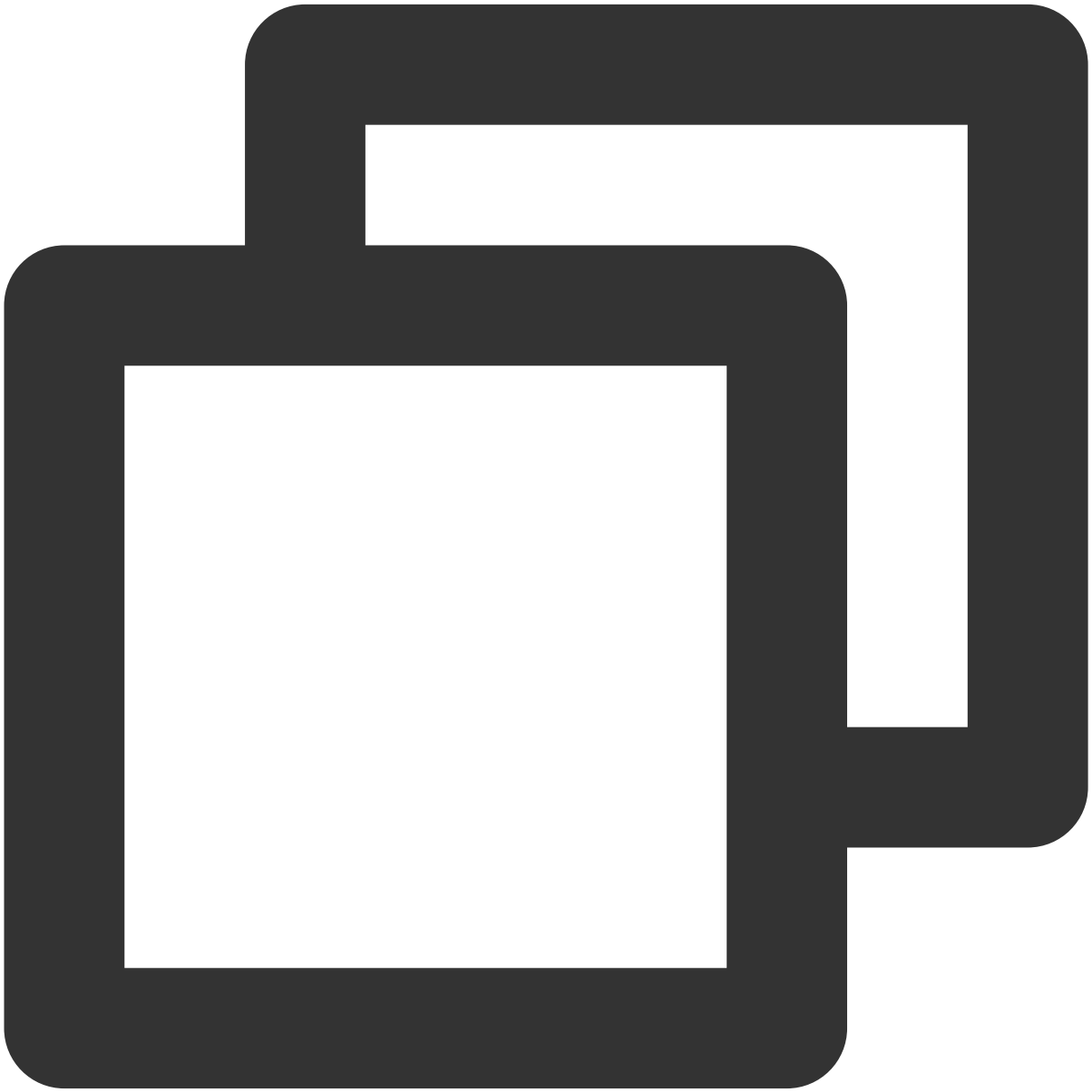
```
RPAD(<str> string, <len> integer[, <pad> string])
```

支持引擎：SparkSQL、Presto。

使用说明：返回 str，在右填充 pa d到长度 len。如果 str 长于 len，则返回值缩短为 len 个字符。如果未指定 pad，str 将用空格字符填充。

返回类型：string。

示例：



```
> SELECT rpad('hi', 5, '??');
 hi???
```

```
> SELECT rpad('hi', 1, '??');
 h
> SELECT rpad('hi', 5);
 hi
```

RTRIM

函数语法：

```
RTRIM(<str> string)
```

支持引擎：SparkSQL、Presto。

使用说明：从 str 中删除尾部空格字符。

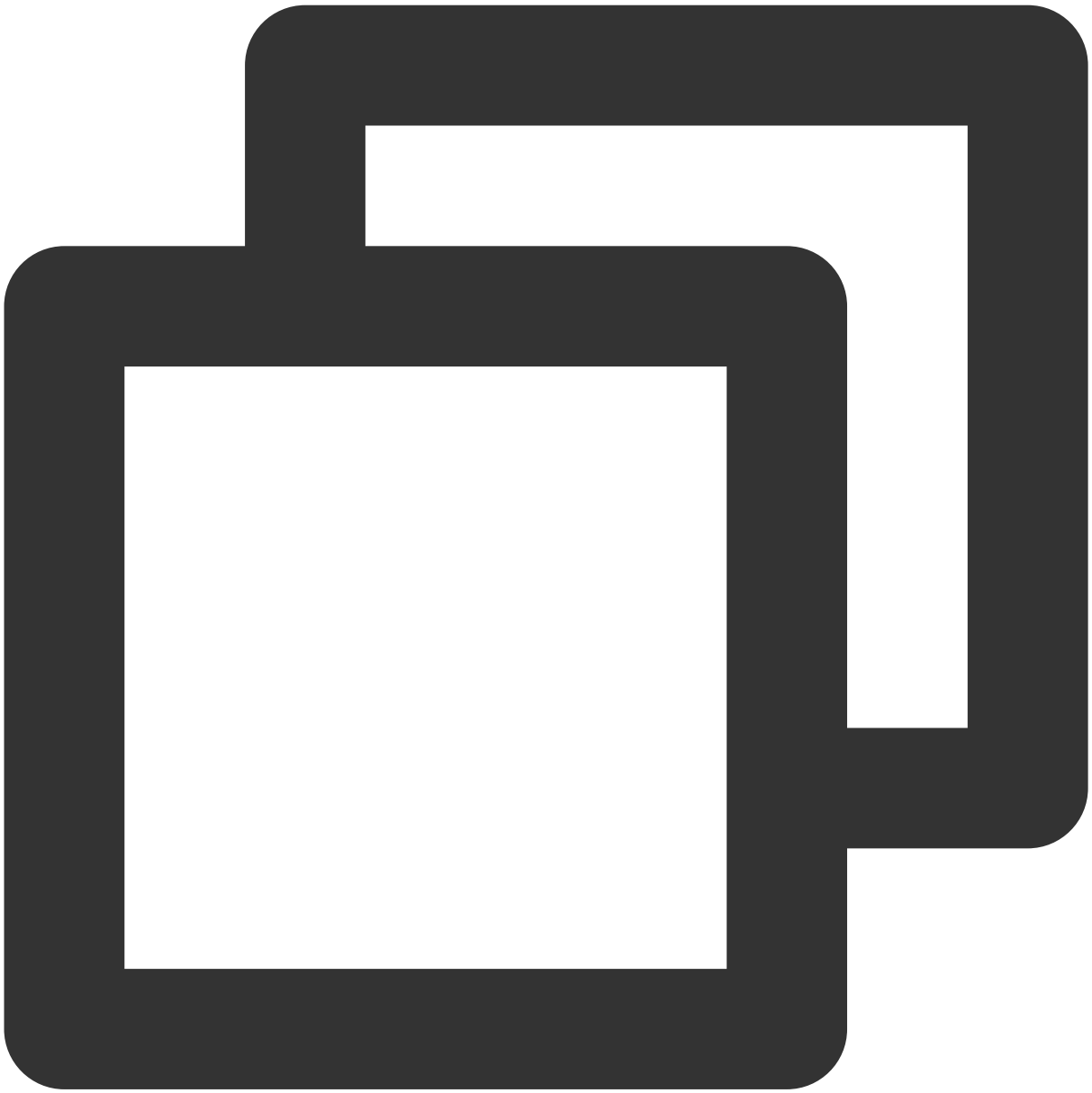
返回类型：string。

示例：

```
> SELECT rtrim('    SparkSQL   ');
   SparkSQL
```

SENTENCES

函数语法：



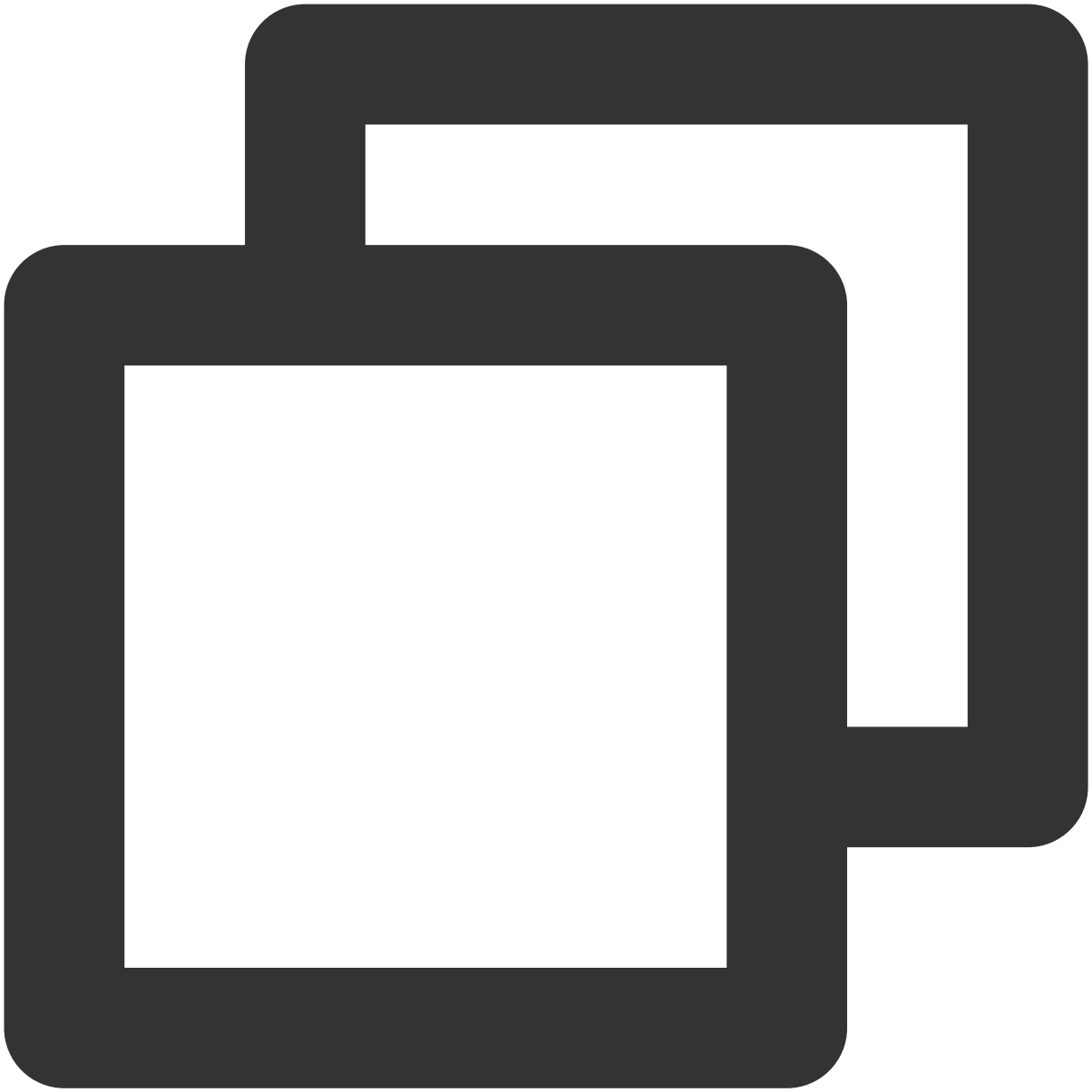
```
SENTENCES(<str> string[, <lang> string, <country> string])
```

支持引擎：SparkSQL、Presto。

使用说明：将 str 拆分为一个单词数组。

返回类型：array <string>。

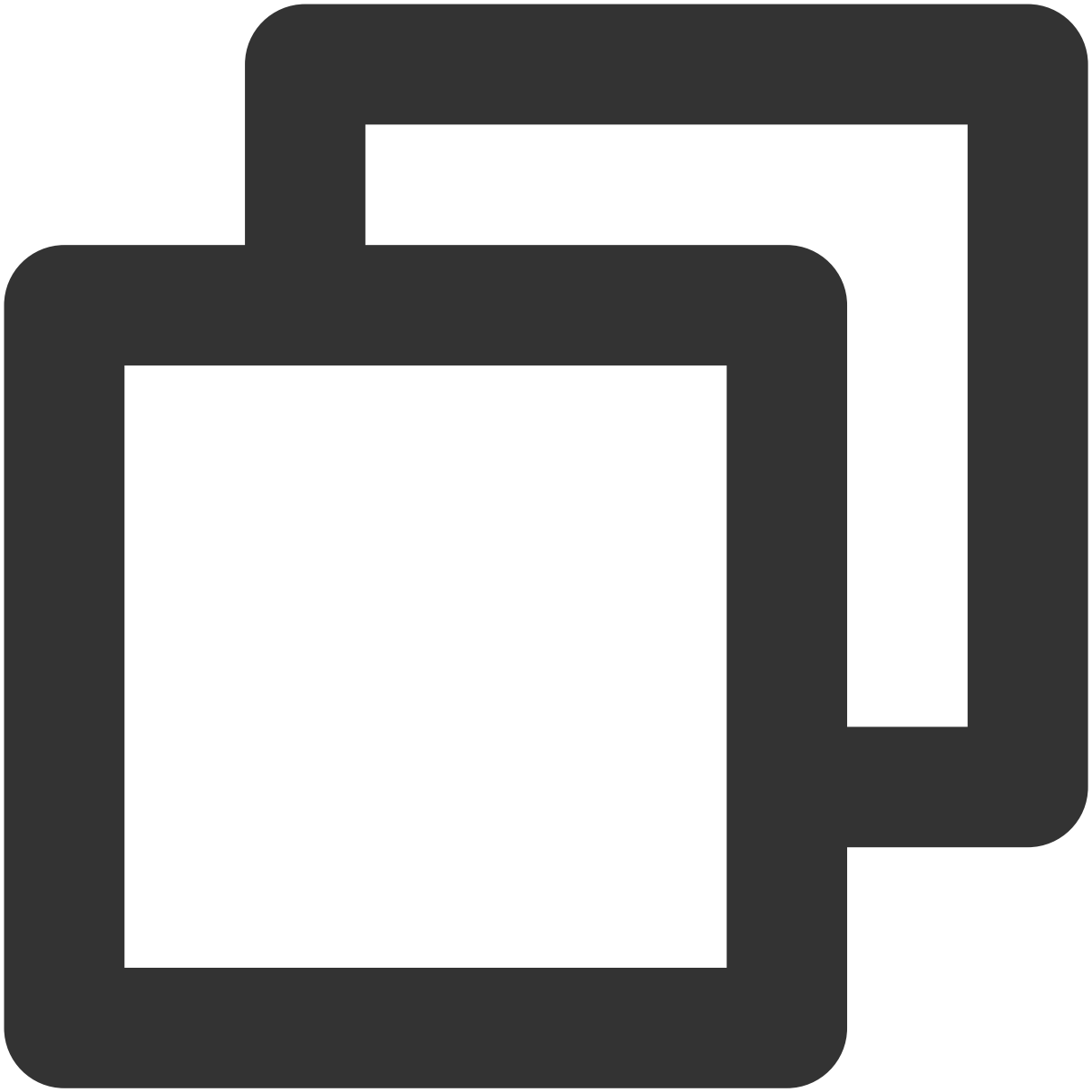
示例：



```
> SELECT sentences('Hi there! Good morning.');
 [["Hi","there"],["Good","morning"]]
```

SOUNDEX

函数语法：



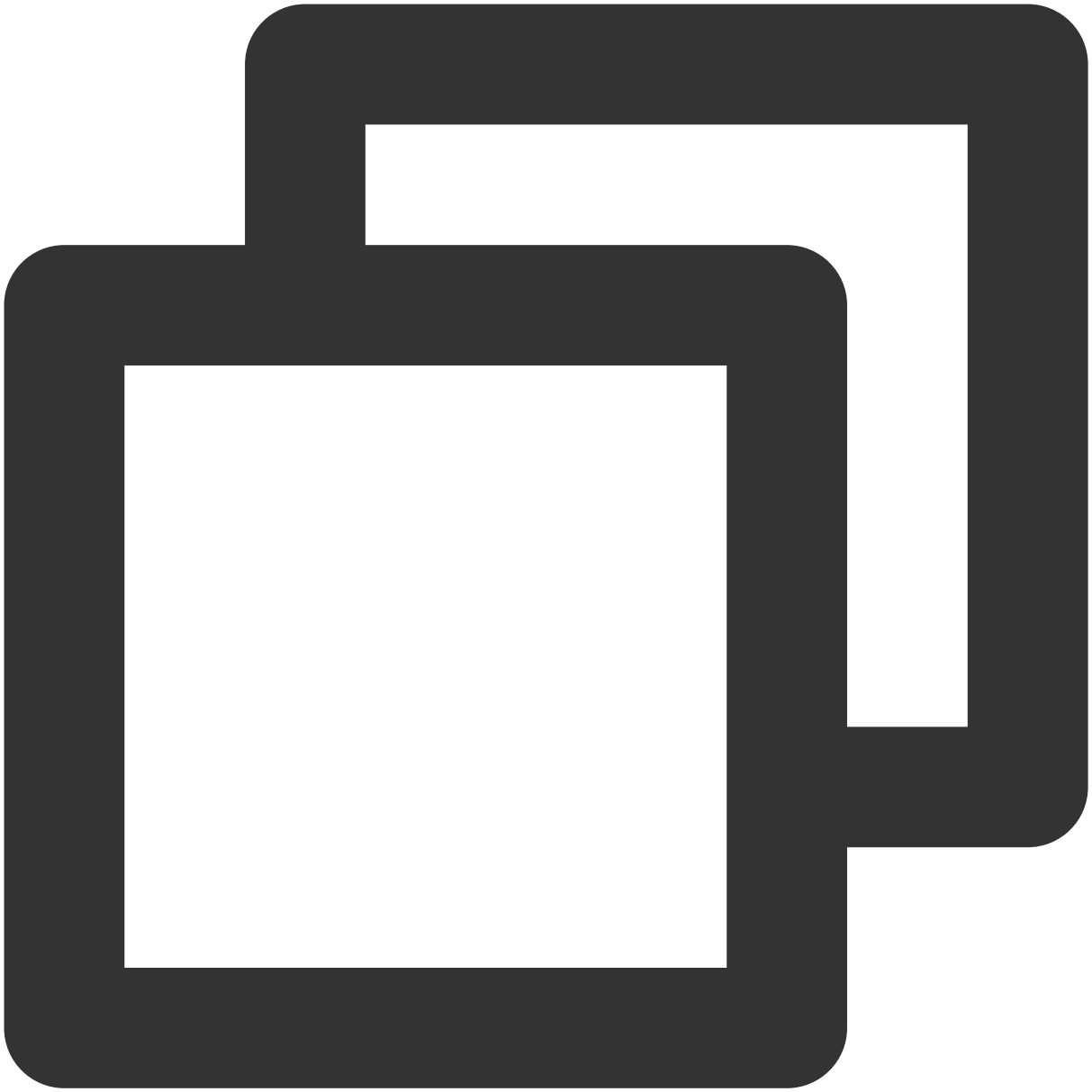
```
SOUNDEX(<str> string)
```

支持引擎：SparkSQL、Presto。

使用说明：返回字符串的 Soundex 编码。

返回类型：string。

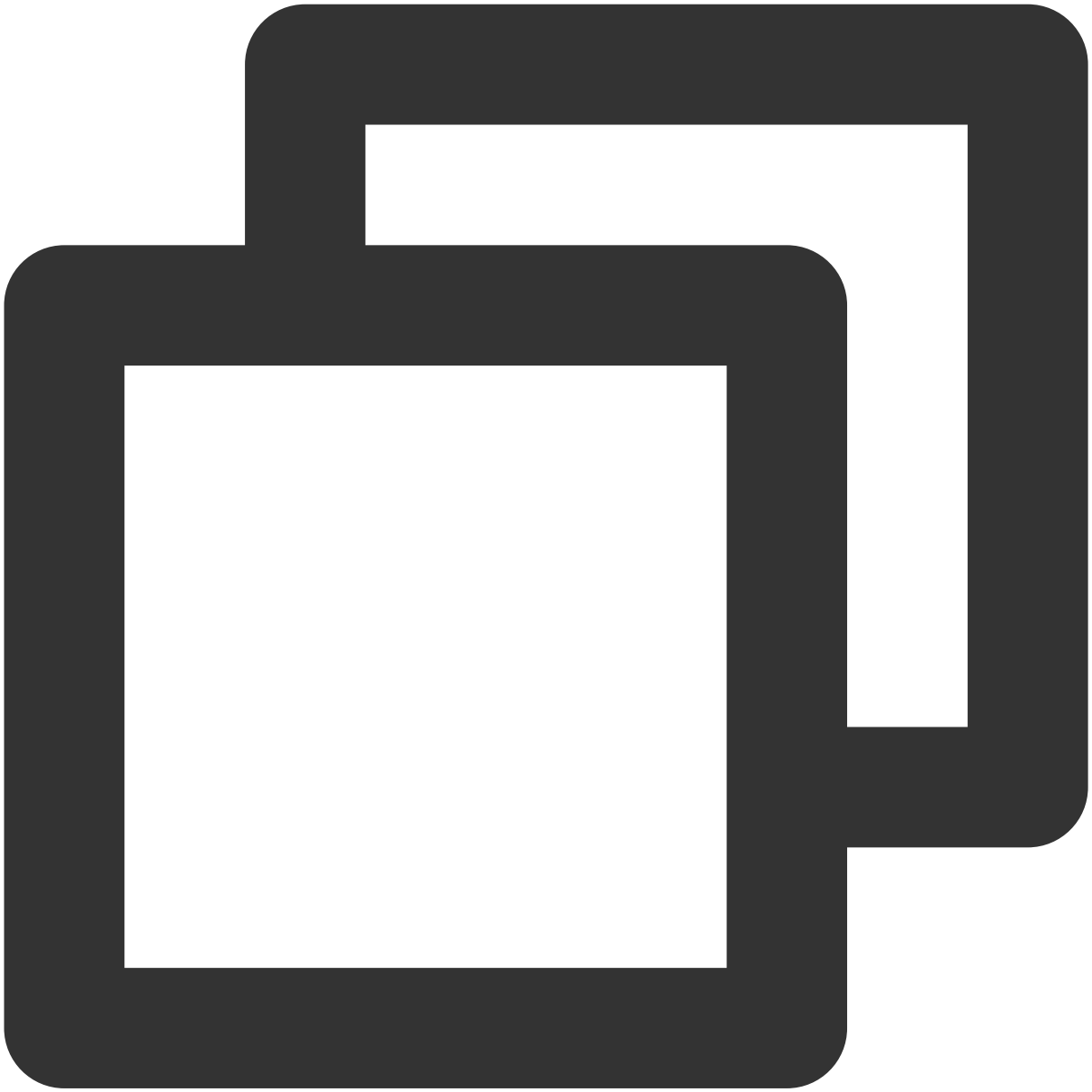
示例：



```
> SELECT soundex('Miller');
 M460
```

SPACE

函数语法：



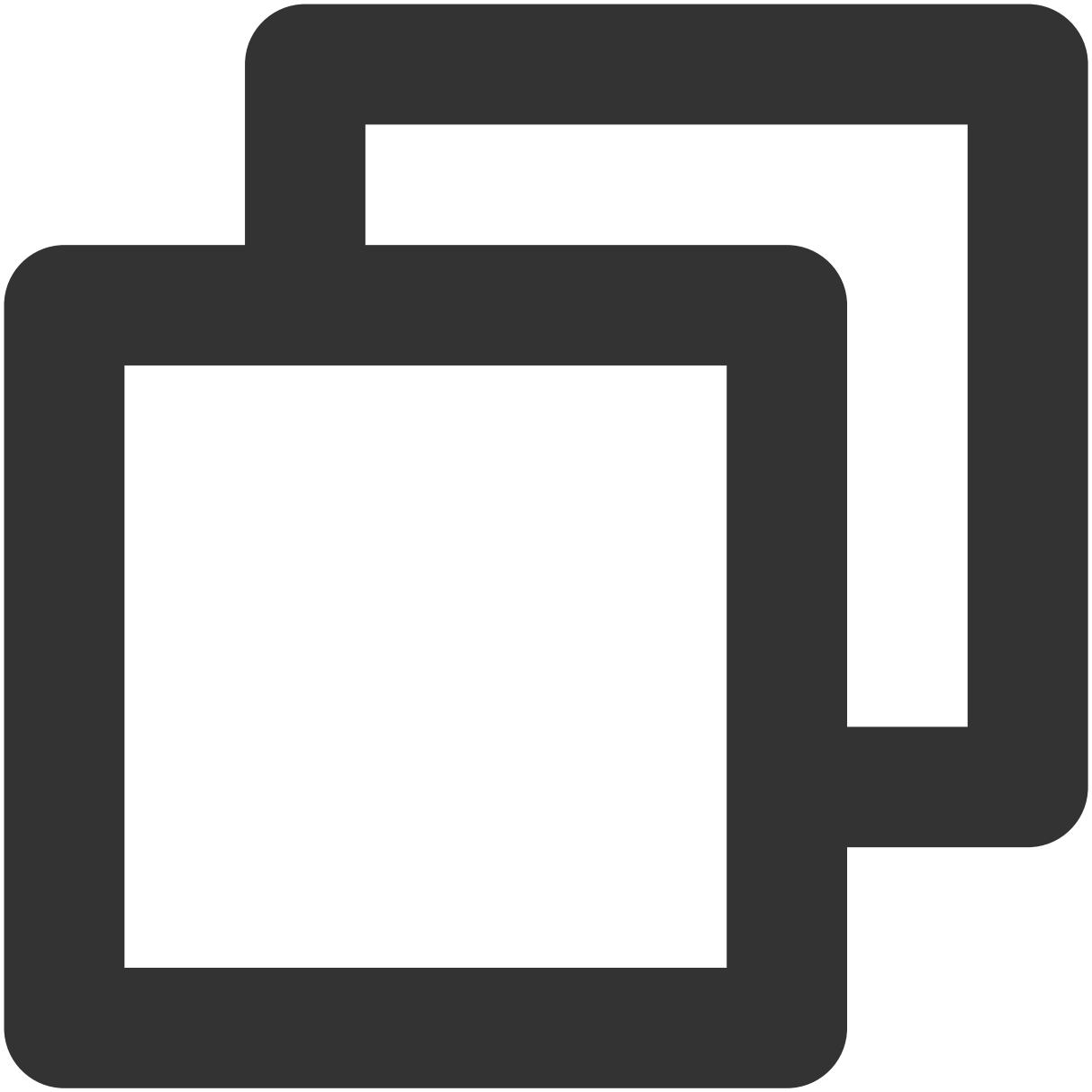
```
SPACE(<n> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：返回由 n 个空格组成的字符串。

返回类型：string。

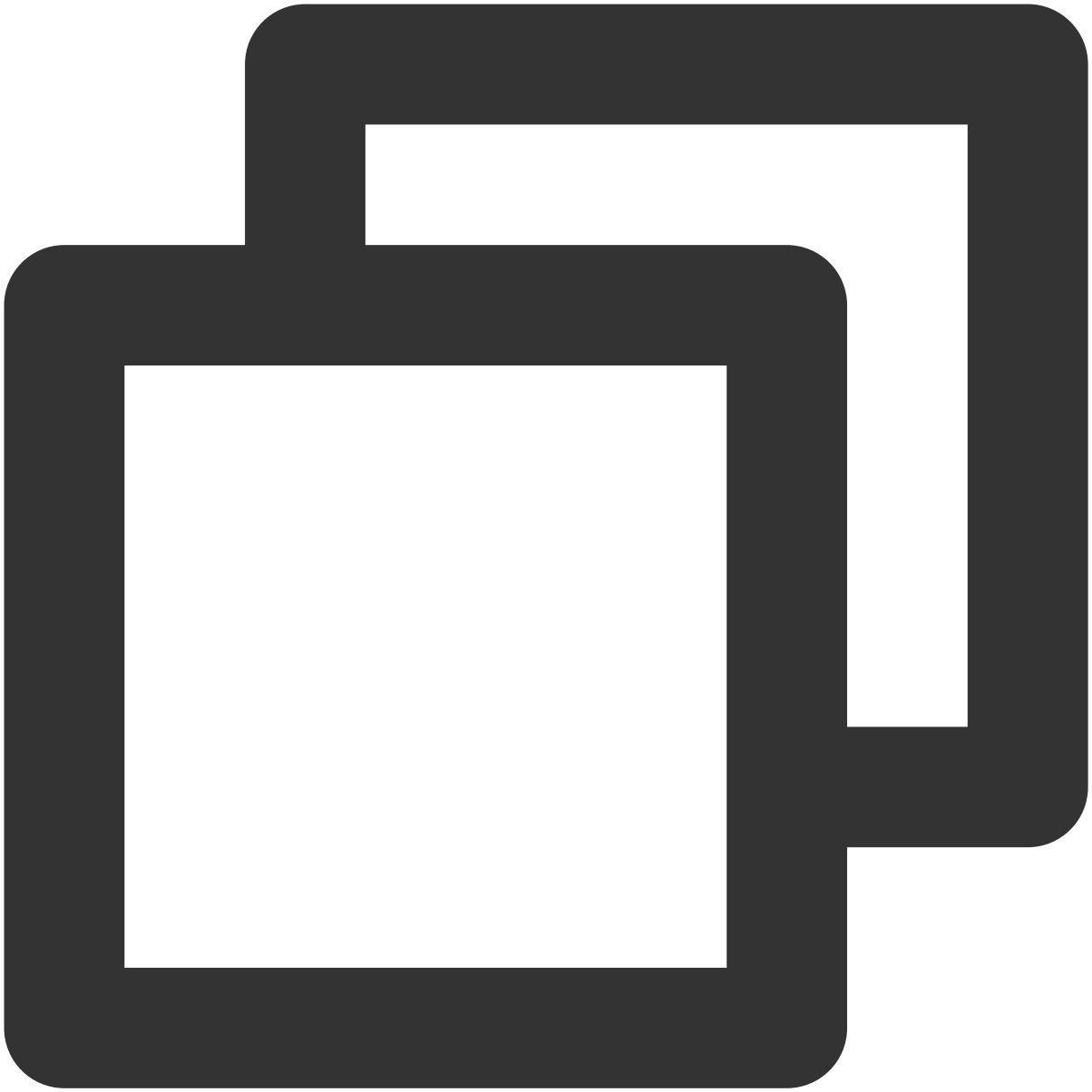
示例：



```
> SELECT concat(space(2), '1');
 1
```

SPLIT

函数语法：



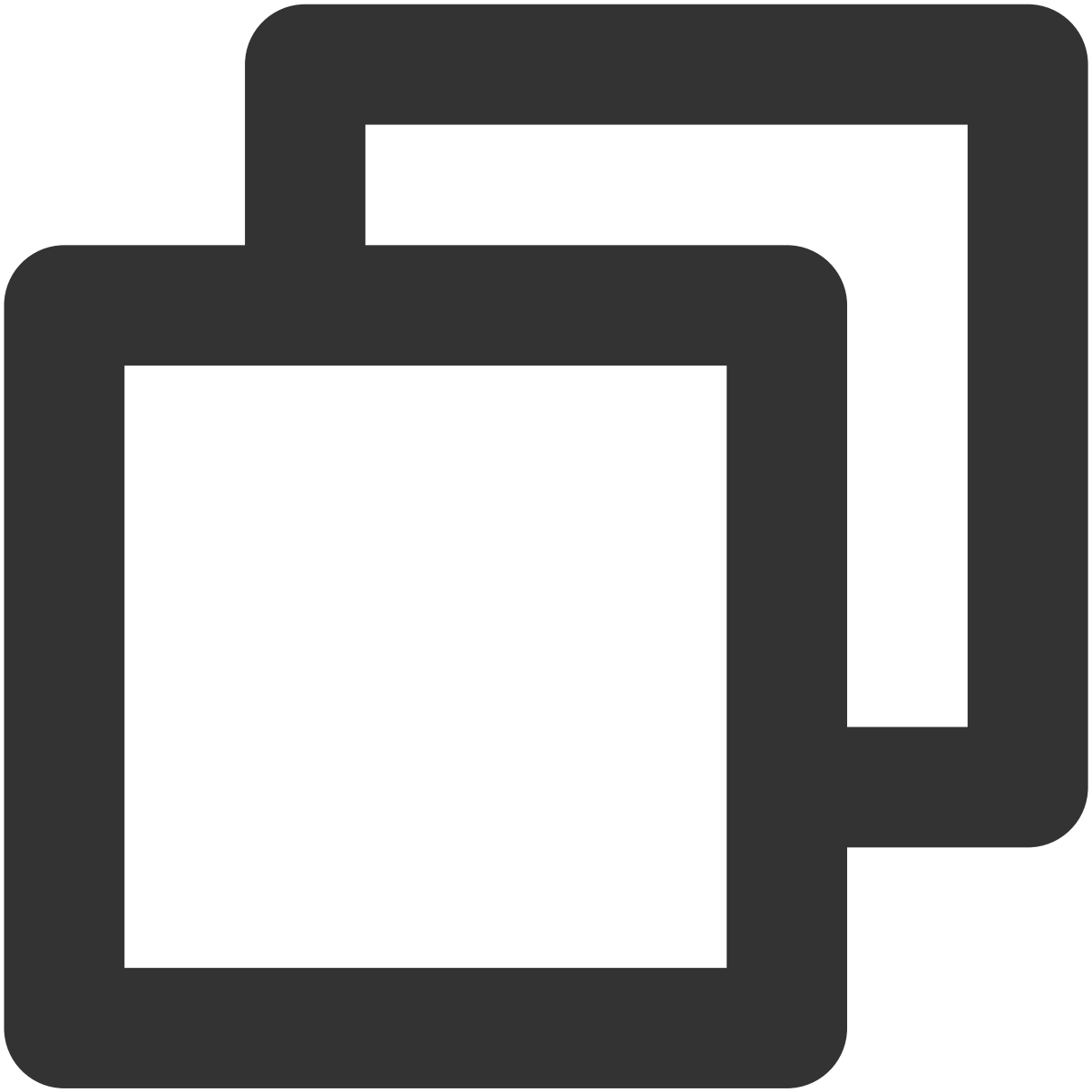
```
SPLIT(<str> string, <regex> string, <limit> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：以与regex匹配的字符串作为分隔符，拆分str，并返回长度最大为limit的数组。

返回类型：array <string>。

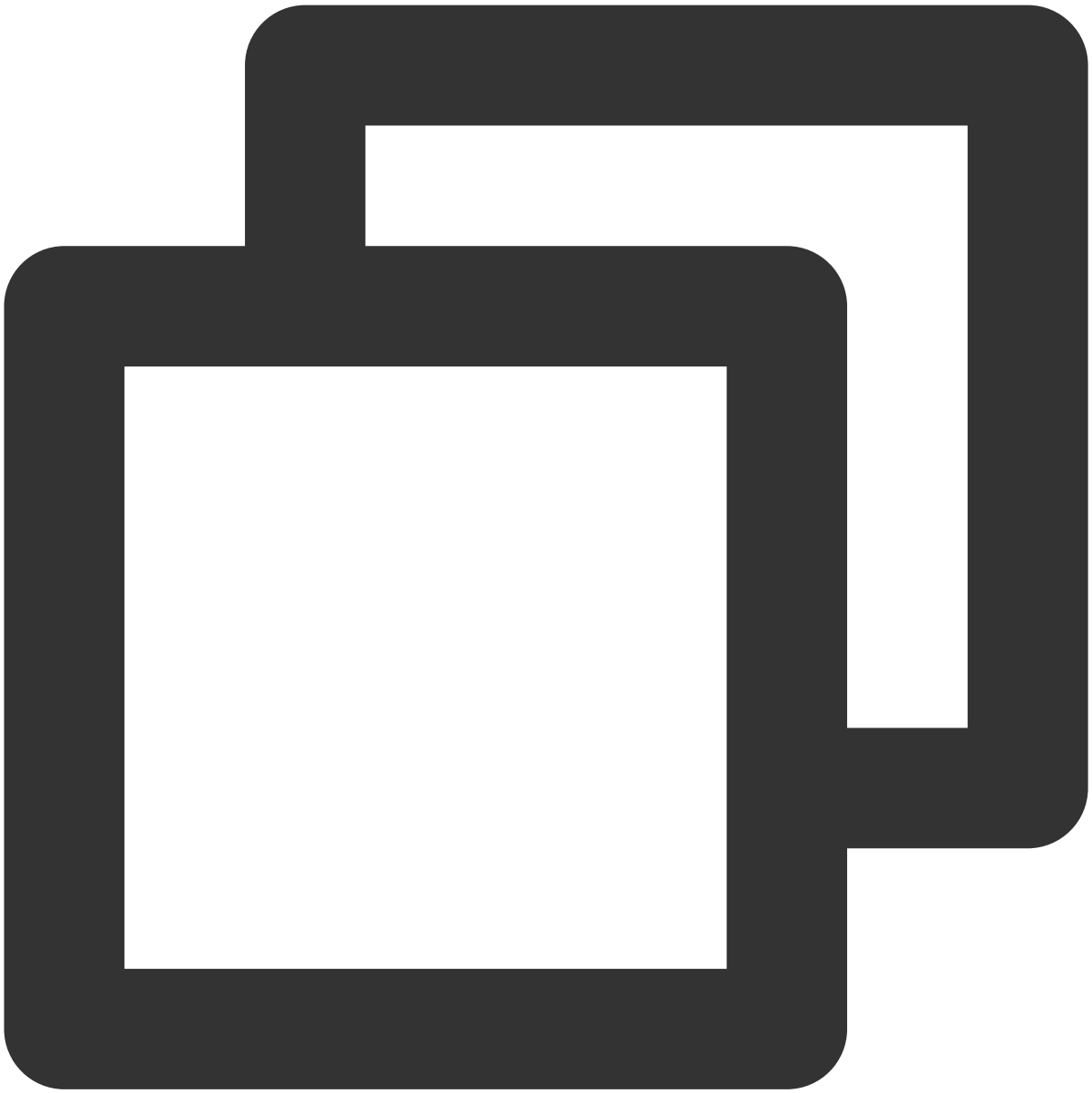
示例：



```
> SELECT split('oneAtwoBthreeC', '[ABC]');
 ["one","two","three",""]
> SELECT split('oneAtwoBthreeC', '[ABC]', -1);
 ["one","two","three",""]
> SELECT split('oneAtwoBthreeC', '[ABC]', 2);
 ["one","twoBthreeC"]
```

SUBSTRING

函数语法：



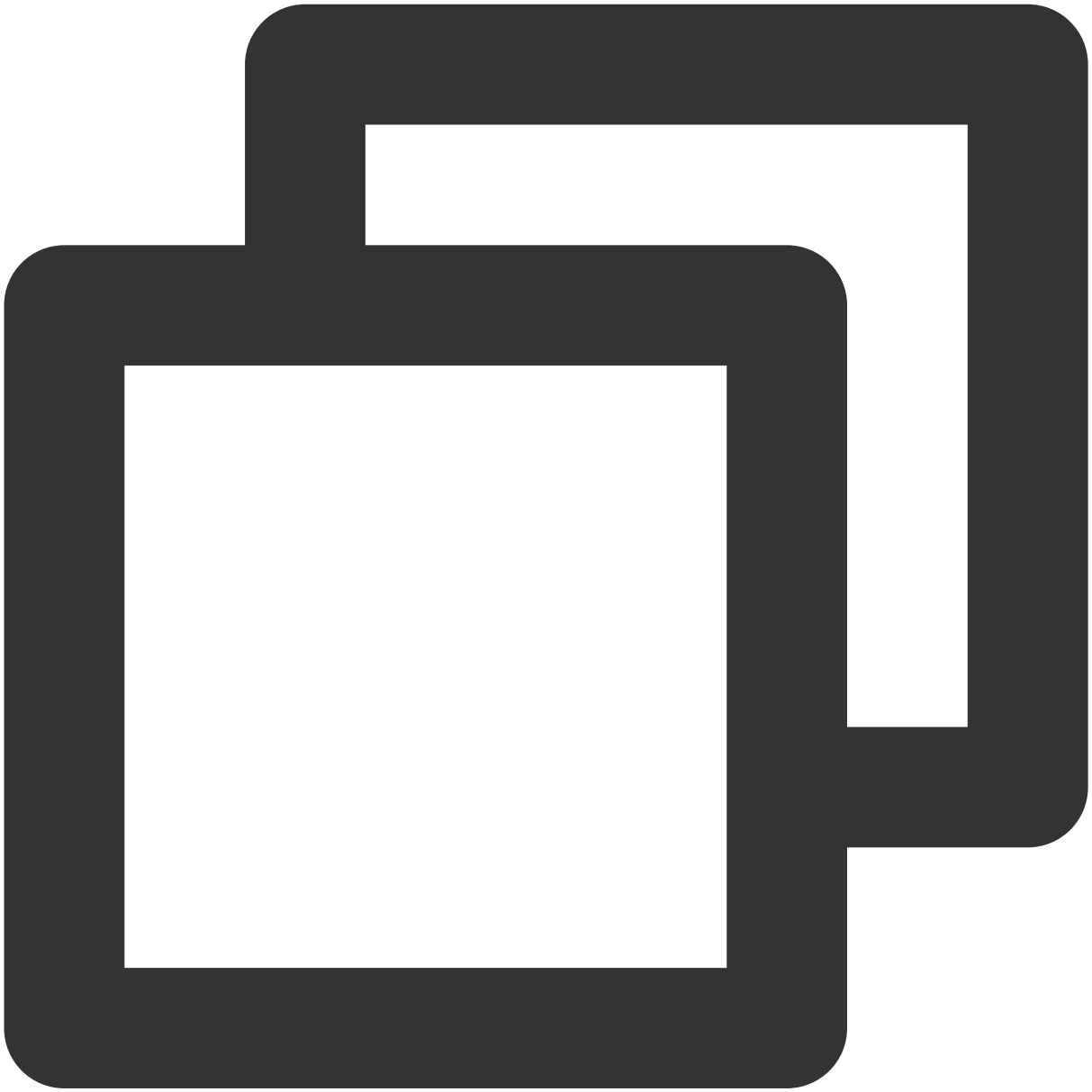
```
SUBSTRING(<str> string, <pos> integer[, <len> integer])
SUBSTRING(<str> FROM <pos>[ FOR <len>])
```

支持引擎：SparkSQL、Presto。

使用说明：返回从 pos 开始且长度为 len 的 str 子字符串，或从 pos 开始且长度为 len 的字节数组切片。

返回类型：string。

示例：

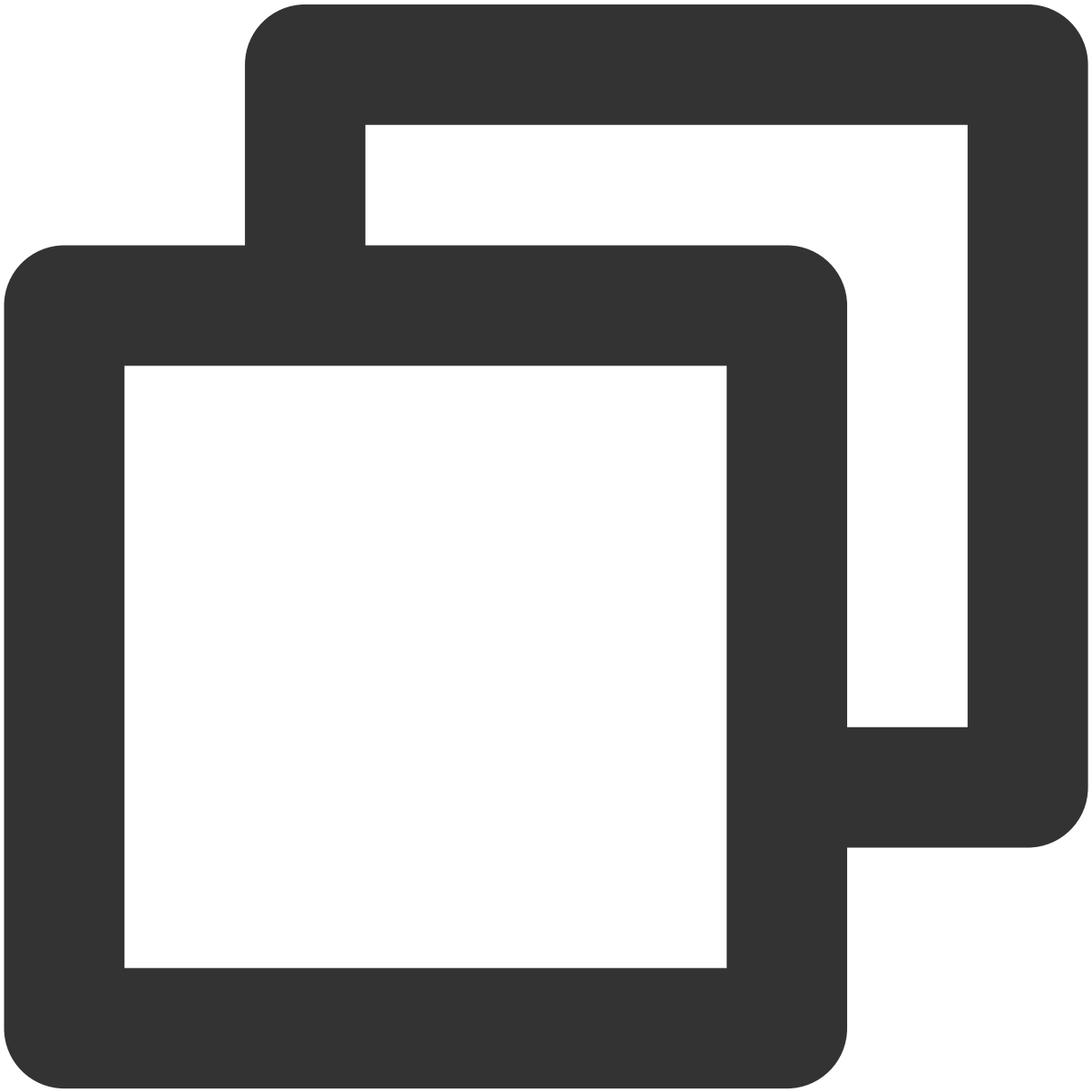


```
> SELECT substring('Spark SQL', 5);
 k SQL
> SELECT substring('Spark SQL', -3);
 SQL
> SELECT substring('Spark SQL', 5, 1);
 k
> SELECT substring('Spark SQL' FROM 5);
 k SQL
> SELECT substring('Spark SQL' FROM -3);
 SQL
> SELECT substring('Spark SQL' FROM 5 FOR 1);
```

```
k
```

SUBSTR

函数语法：



```
SUBSTR(<str> string, <pos> integer[, <len> integer])
SUBSTR(<str> FROM <pos>[ FOR <len>])
```

支持引擎：SparkSQL、Presto。

使用说明：返回从pos开始且长度为len的str子字符串，或从pos开始且长度为len的字节数组切片。

返回类型：string。

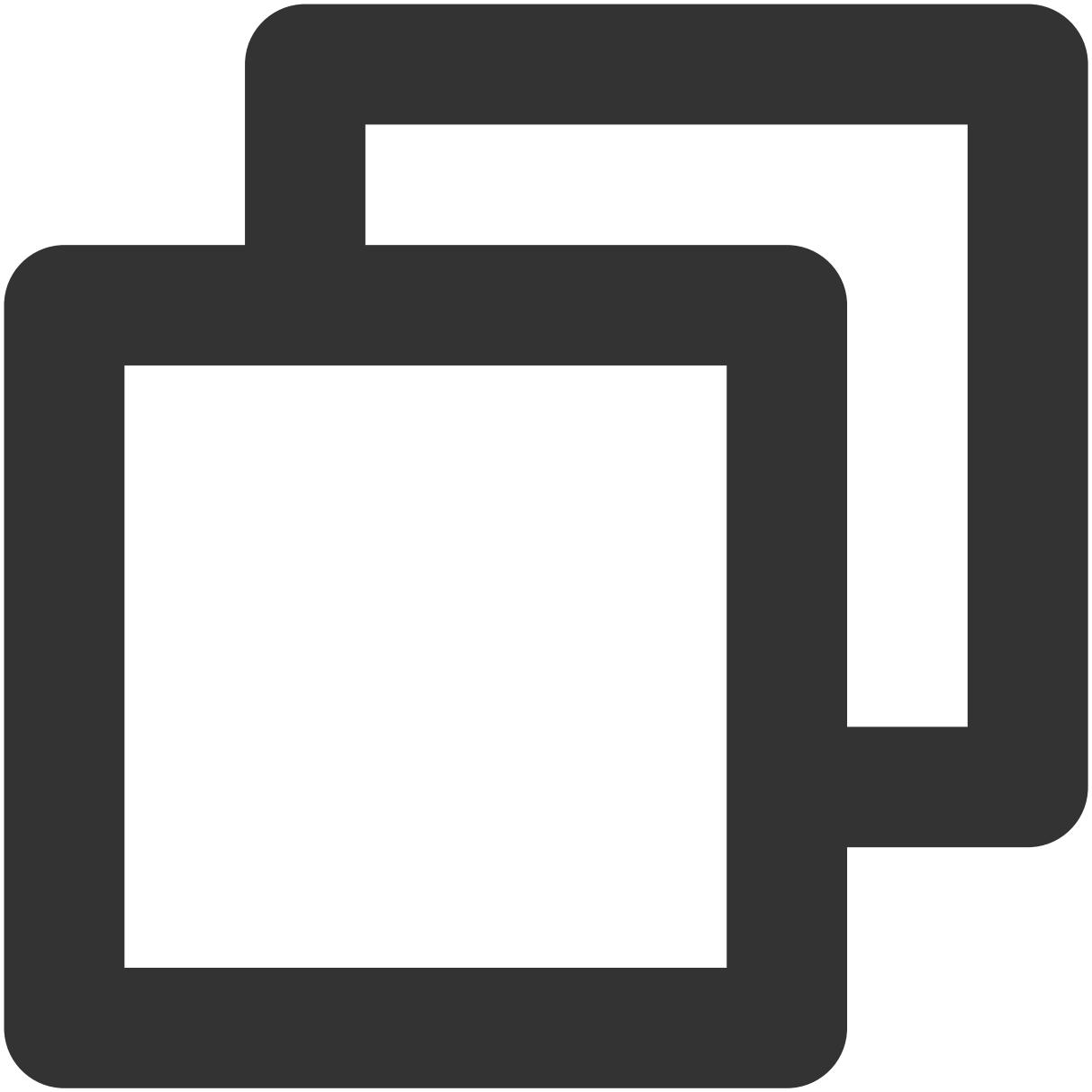
示例：

```
> SELECT substr('Spark SQL', 5);
 k SQL
> SELECT substr('Spark SQL', -3);
 SQL
> SELECT substr('Spark SQL', 5, 1);
 k
```

```
> SELECT substr('Spark SQL' FROM 5);
 k SQL
> SELECT substr('Spark SQL' FROM -3);
 SQL
> SELECT substr('Spark SQL' FROM 5 FOR 1);
 k
```

LEFT

函数语法：



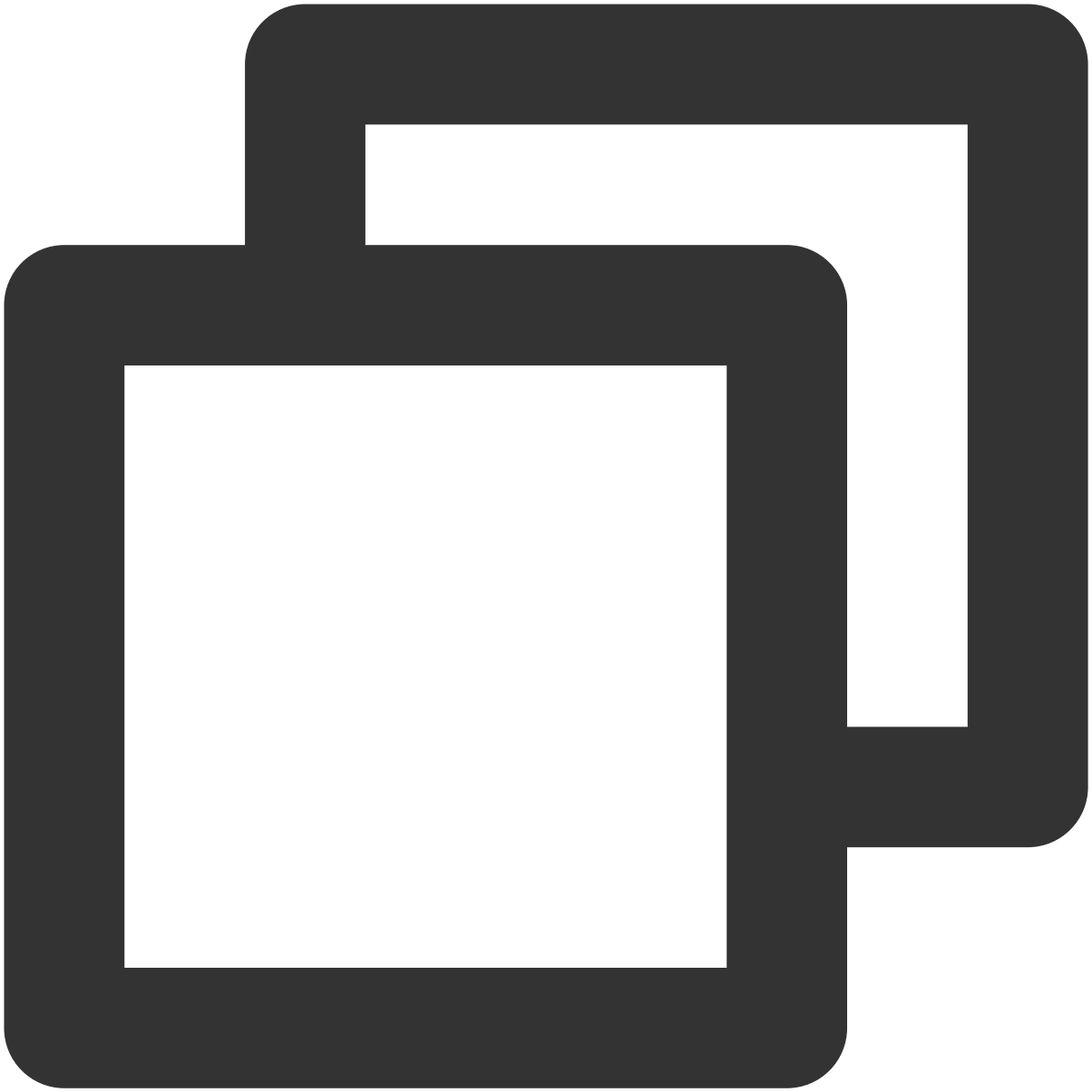
```
LEFT(<str> string, <len> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：返回字符串str中最左边的len字符，如果len小于或等于0，则结果为空字符串。

返回类型：string。

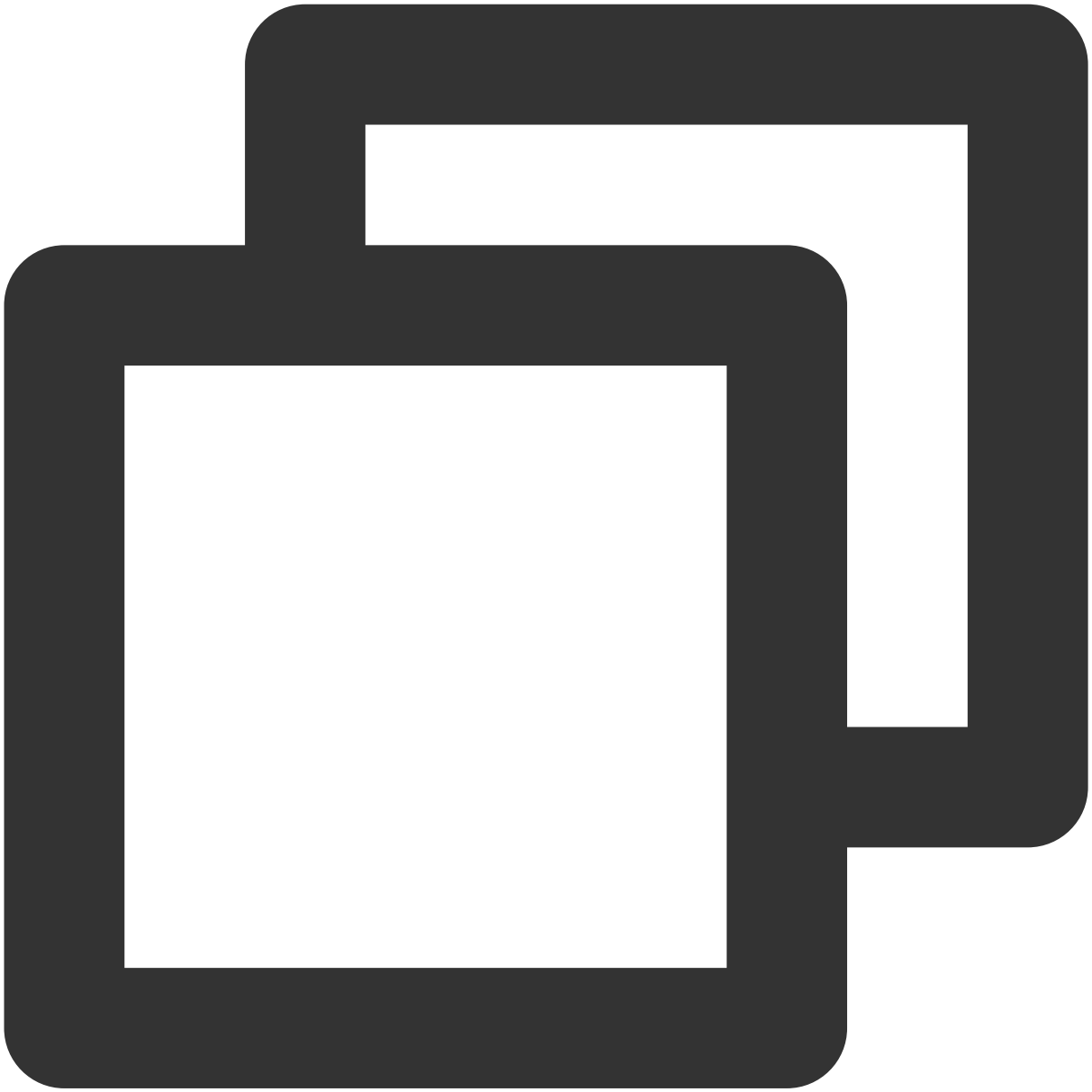
示例：



```
> SELECT left('tencent', 3);
 ten
```

RIGHT

函数语法：



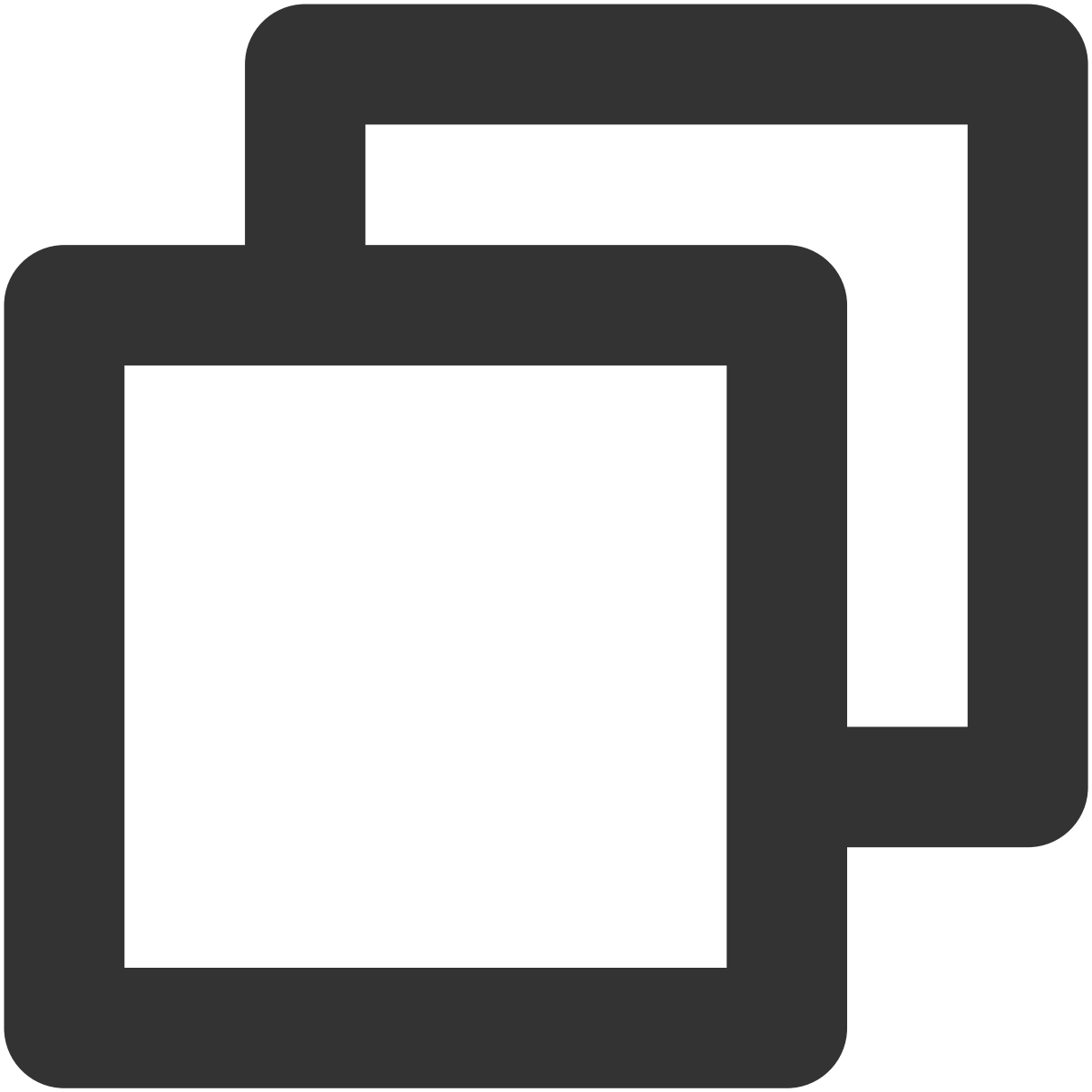
```
RIGHT(<str> string, <len> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：返回字符串str中最左边的len字符，如果len小于或等于0，则结果为空字符串。

返回类型：string。

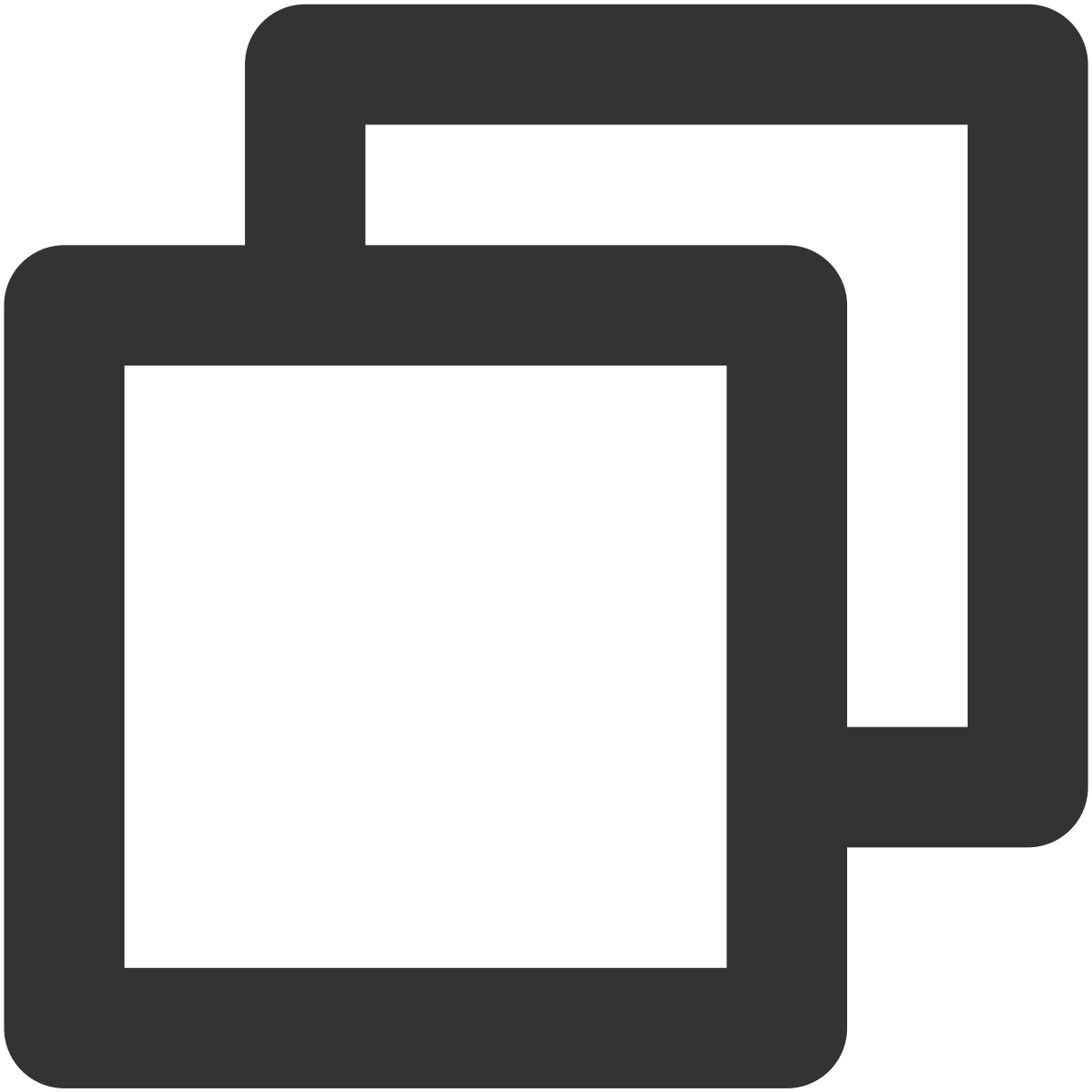
示例：



```
> SELECT left('tencent', 3);
 ten
```

SUBSTRING_INDEX

函数语法：



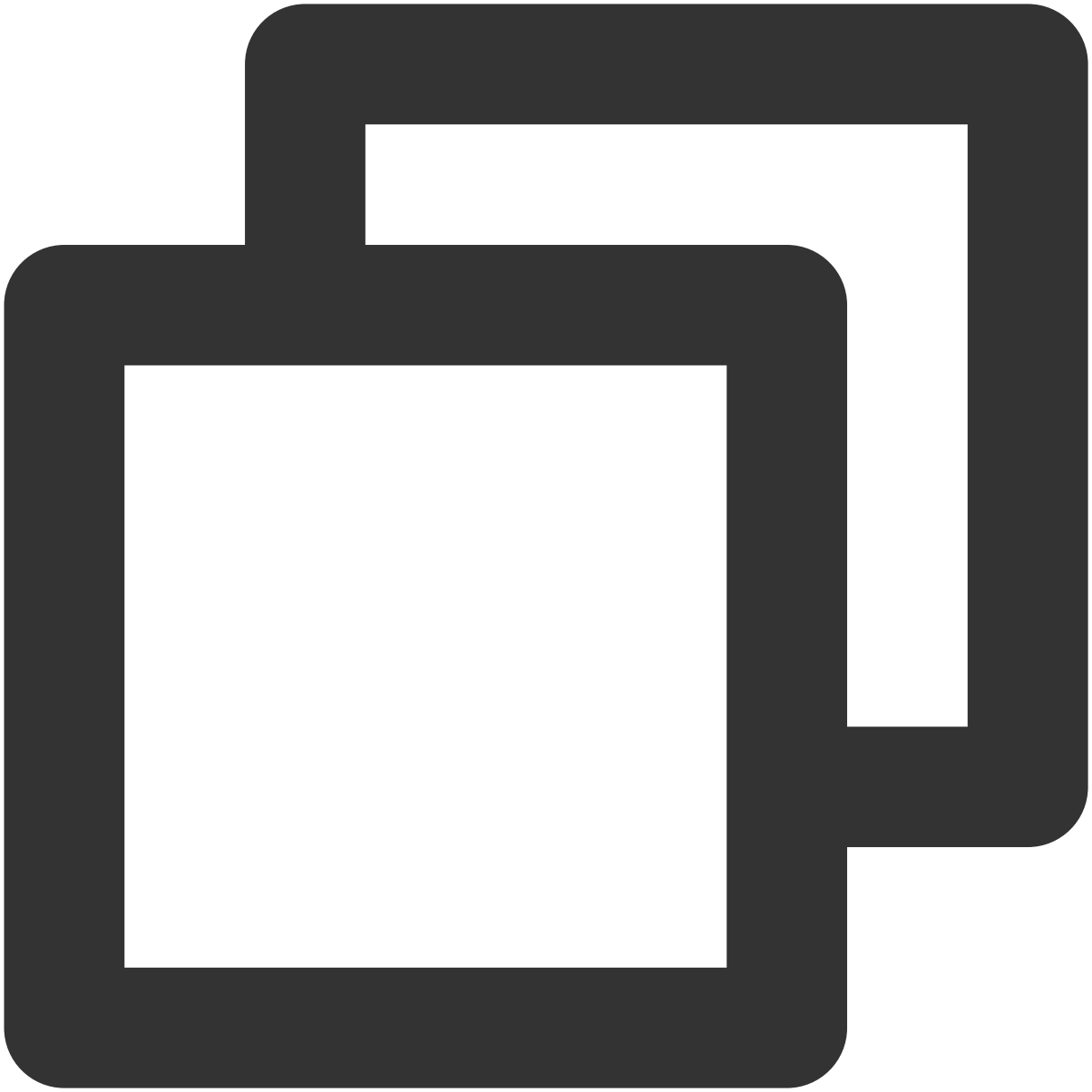
```
SUBSTRING_INDEX(<str> string, <delim> string, <count> integer)
```

支持引擎：SparkSQL、Presto。

使用说明：在delim的出现count之前，从str返回子字符串。如果count为正数，则返回最后定界符左侧的所有内容（从左侧计数）。如果计数为负数，则返回最终定界符右侧的所有内容（从右侧计数）。该函数在匹配delim时区分大小写。

返回类型：string。

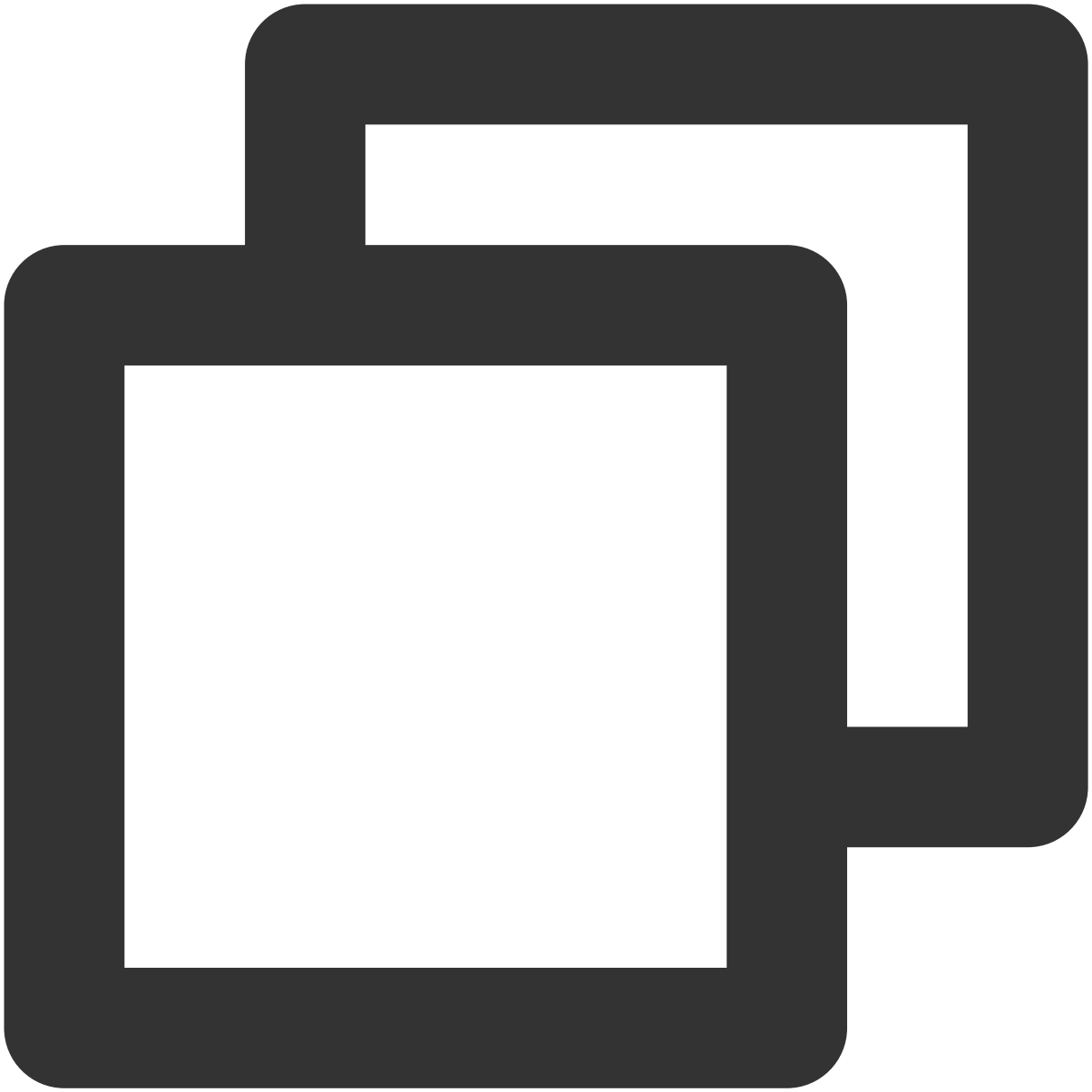
示例：



```
> SELECT substring_index('cloud.tencent.com', '.', 2);
 cloud.tencent
```

TRANSLATE

函数语法：



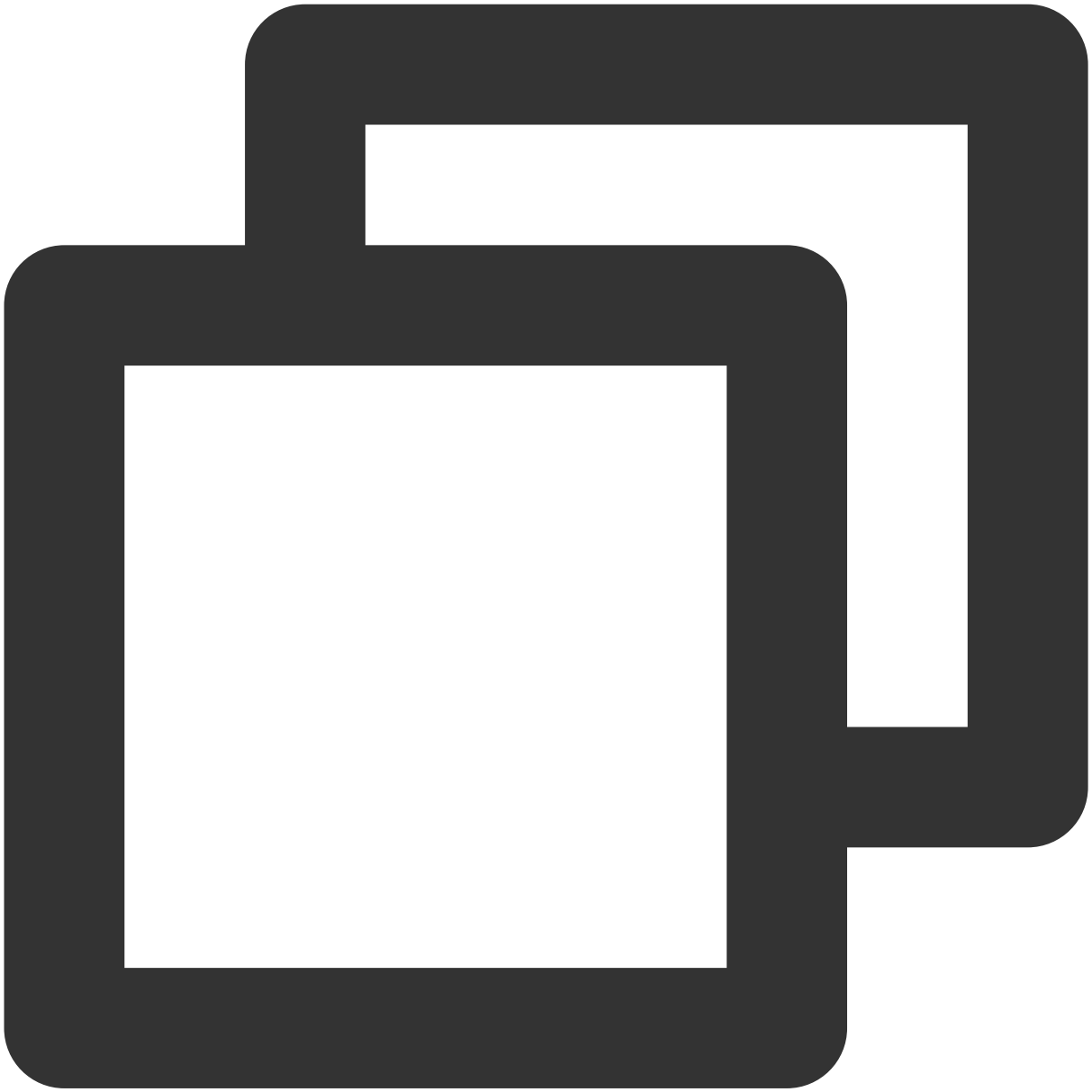
```
TRANSLATE(<input> string, <from> string, <to> string)
```

支持引擎：SparkSQL。

使用说明：通过将from字符串中的字符替换为to字符串中的相应字符来转换input字符串。

返回类型：string。

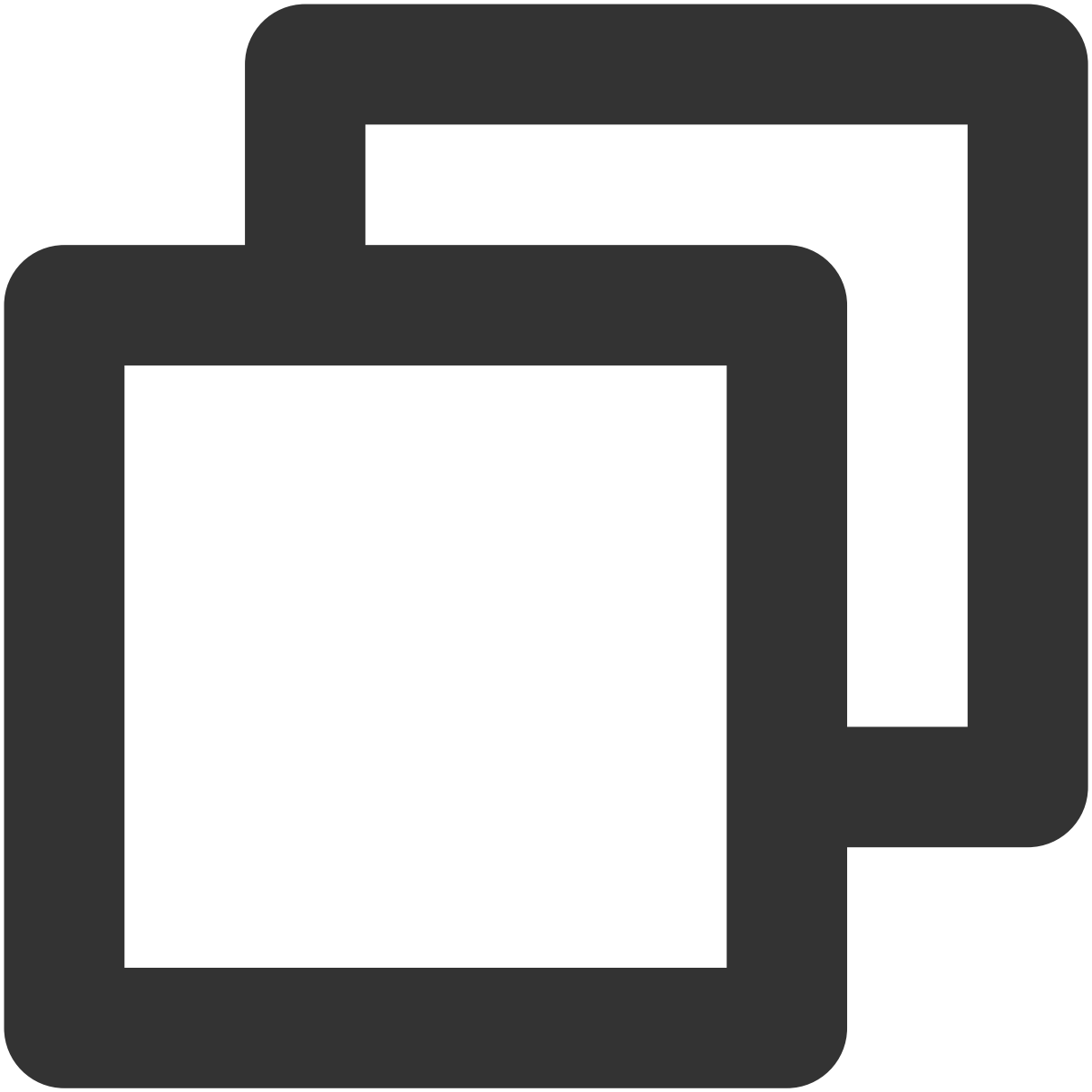
示例：



```
> SELECT translate('AaBbCc', 'abc', '123');
 A1B2C3
```

TRIM

函数语法：



```
TRIM(<str> string)
trim(BOTH FROM str)
trim(LEADING FROM str)
trim(TRAILING FROM str)
trim(trimStr FROM str)
trim(BOTH trimStr FROM str)
trim(LEADING trimStr FROM str)
trim(TRAILING trimStr FROM str)
```

支持引擎：SparkSQL、Presto

使用说明：

trim(str) - 从str中删除前导和末尾空格字符。

trim(BOTH FROM str) ：从str中删除前导和末尾空格字符。

trim(LEADING FROM str) ：从str中删除前导空格字符。

trim(TRAILING FROM str) ：从str中删除末尾空格字符。

trim(trimStr FROM str) ：从str中删除开头和结尾的trimStr字符。

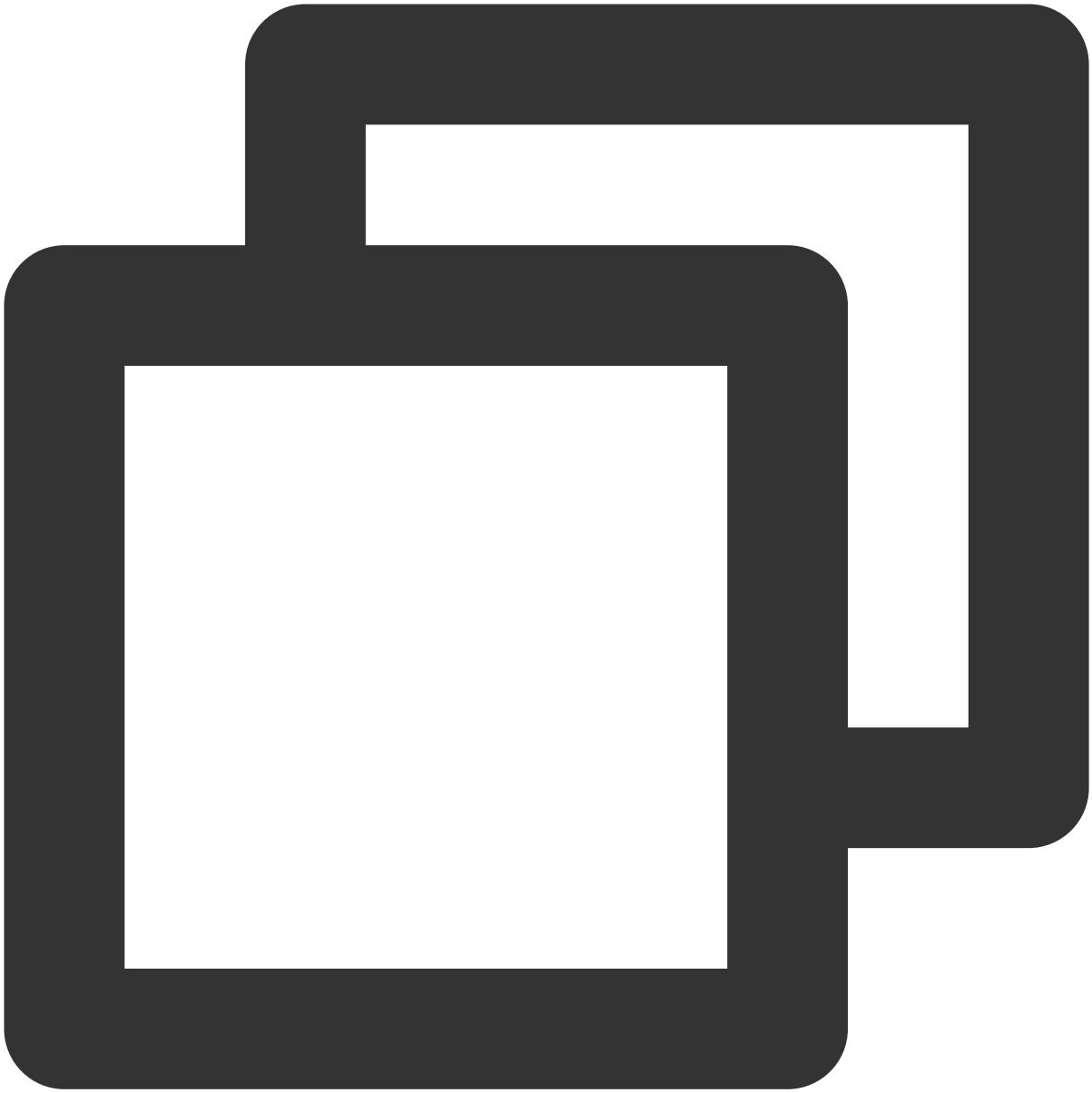
trim(BOTH trimStr FROM str) ：从str中删除开头和结尾的trimStr字符。

trim(LEADING trimStr FROM str) ：从str中删除前导trimStr字符。

trim(TRAILING trimStr FROM str)：从str中删除尾部trimStr字符。

返回类型：string

示例：



```
> SELECT trim('    SparkSQL   ');
 SparkSQL
> SELECT trim(BOTH FROM '    SparkSQL   ');
 SparkSQL
> SELECT trim(LEADING FROM '    SparkSQL   ');
 SparkSQL
> SELECT trim(TRAILING FROM '    SparkSQL   ');
   SparkSQL
> SELECT trim('SL' FROM 'SSparkSQLS');
 parkSQ
> SELECT trim(BOTH 'SL' FROM 'SSparkSQLS');
```

```
 parkSQ
> SELECT trim(LEADING 'SL' FROM 'SSparkSQLS');
 parkSQLS
> SELECT trim(TRAILING 'SL' FROM 'SSparkSQLS');
 SSparkSQ
```

BTRIM

函数语法：

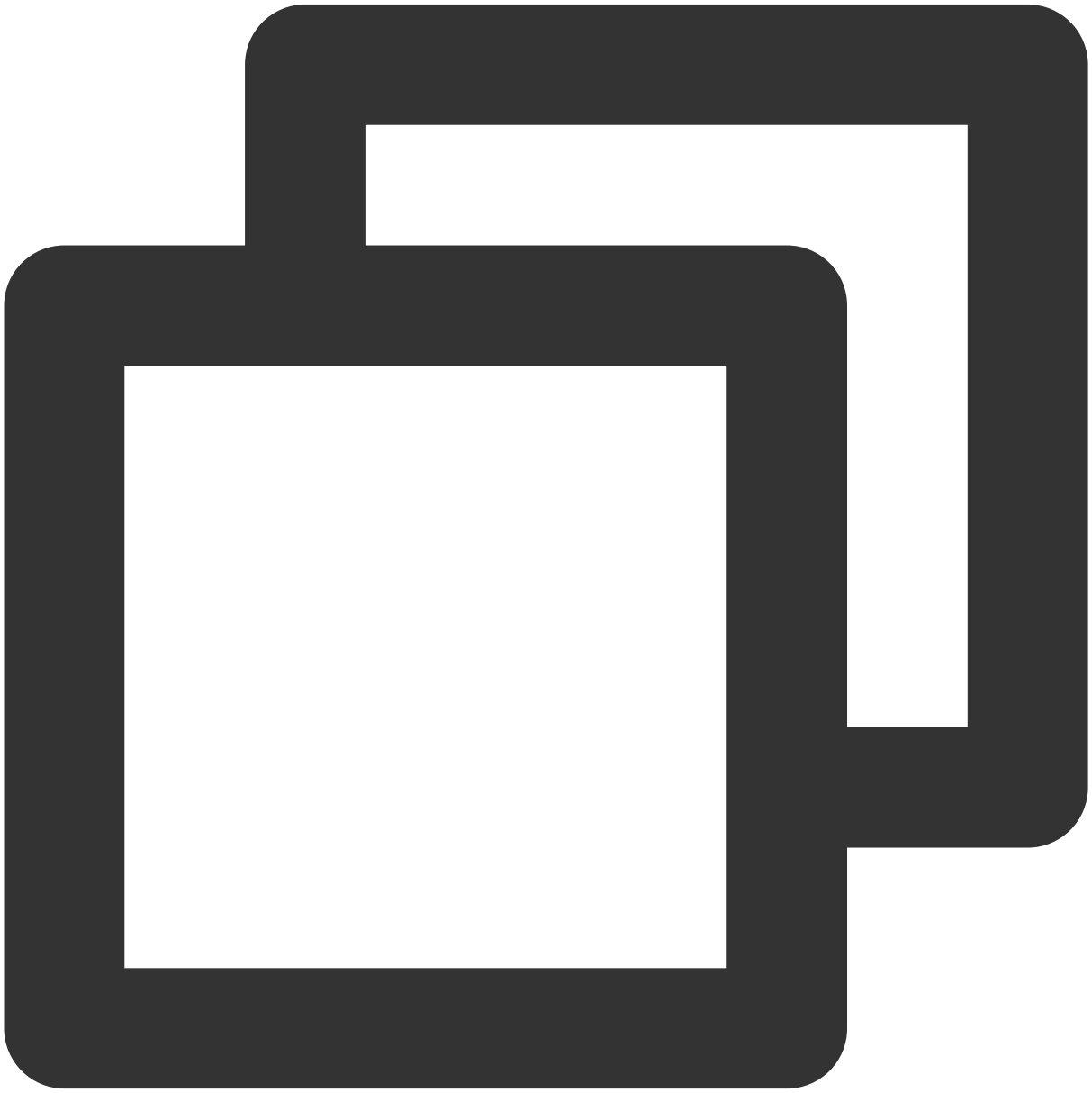
```
BTRIM(<str> string[, <trimStr> string])
```

支持引擎：SparkSQL、Presto

使用说明：从str中删除开头和结尾trimStr（默认为空格）字符。

返回类型：string

示例：



```
> SELECT btrim('    SparkSQL   ');
 SparkSQL
> SELECT btrim(encode('    SparkSQL   ', 'utf-8'));
```

```
 SparkSQL
> SELECT btrim('SSparkSQLS', 'SL');
 parkSQ
> SELECT btrim(encode('SSparkSQLS', 'utf-8'), encode('SL', 'utf-8'));
 parkSQ
```

UCASE

函数语法：

```
UCASE(<str> string)
```

支持引擎：SparkSQL、Presto

使用说明：返回所有字符都改为大写的str。

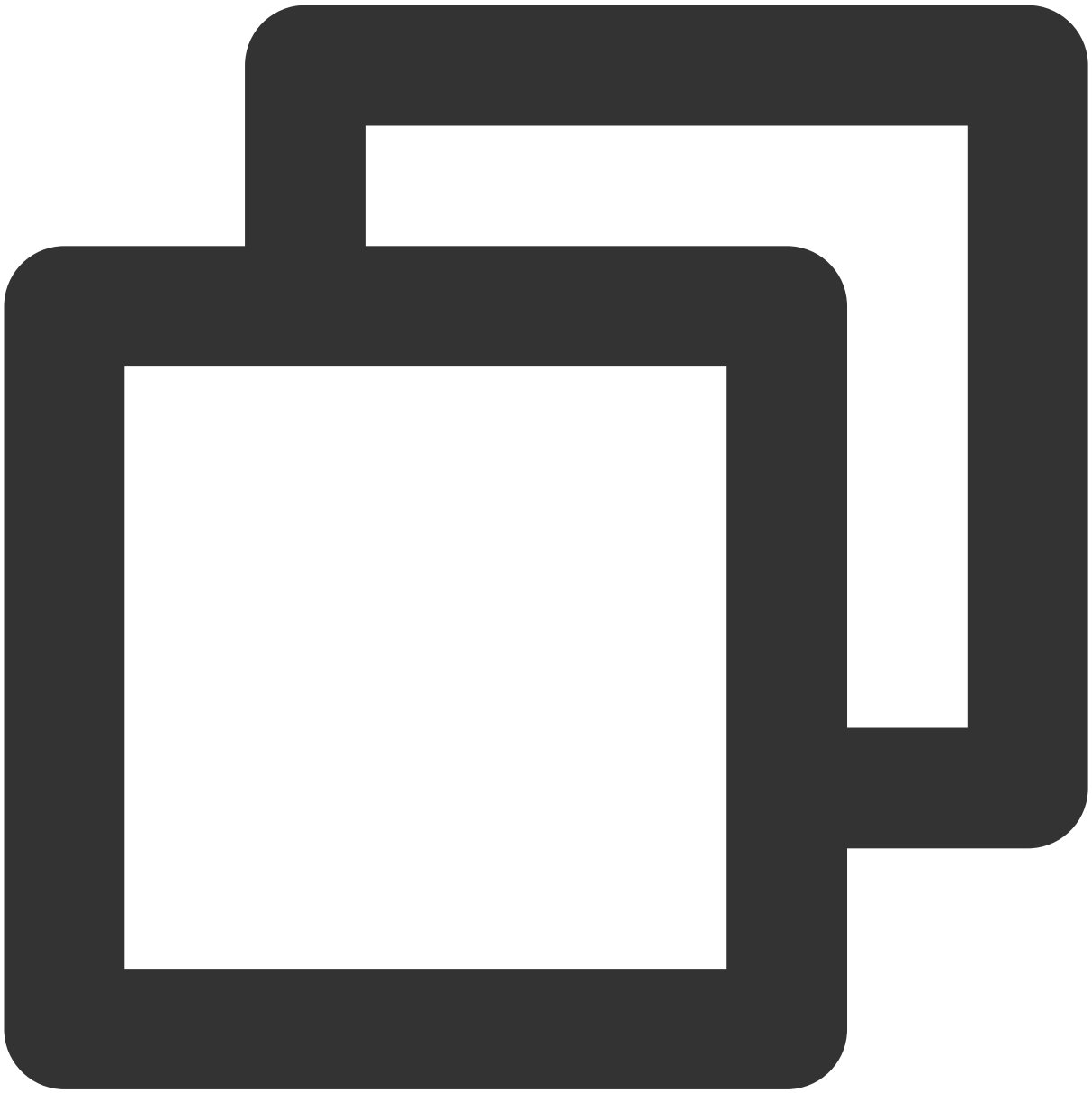
返回类型：string

示例：

```
> SELECT ucase('SparkSQL');
 SPARKSQL
```

UNBASE64

函数语法：



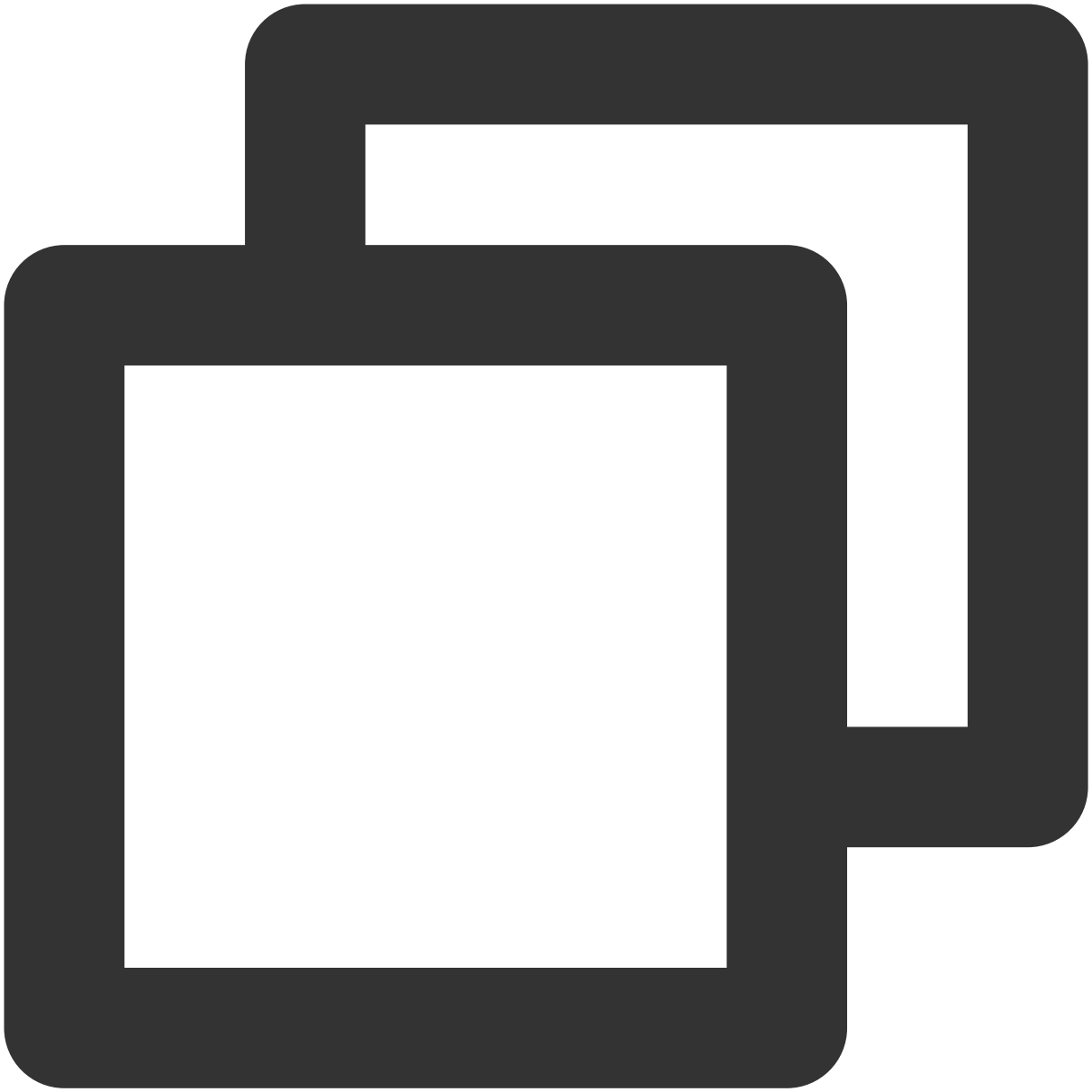
```
UNBASE64(<str> string)
```

支持引擎：SparkSQL、Presto

使用说明：将str从base64字符串转换为二进制。

返回类型：binary

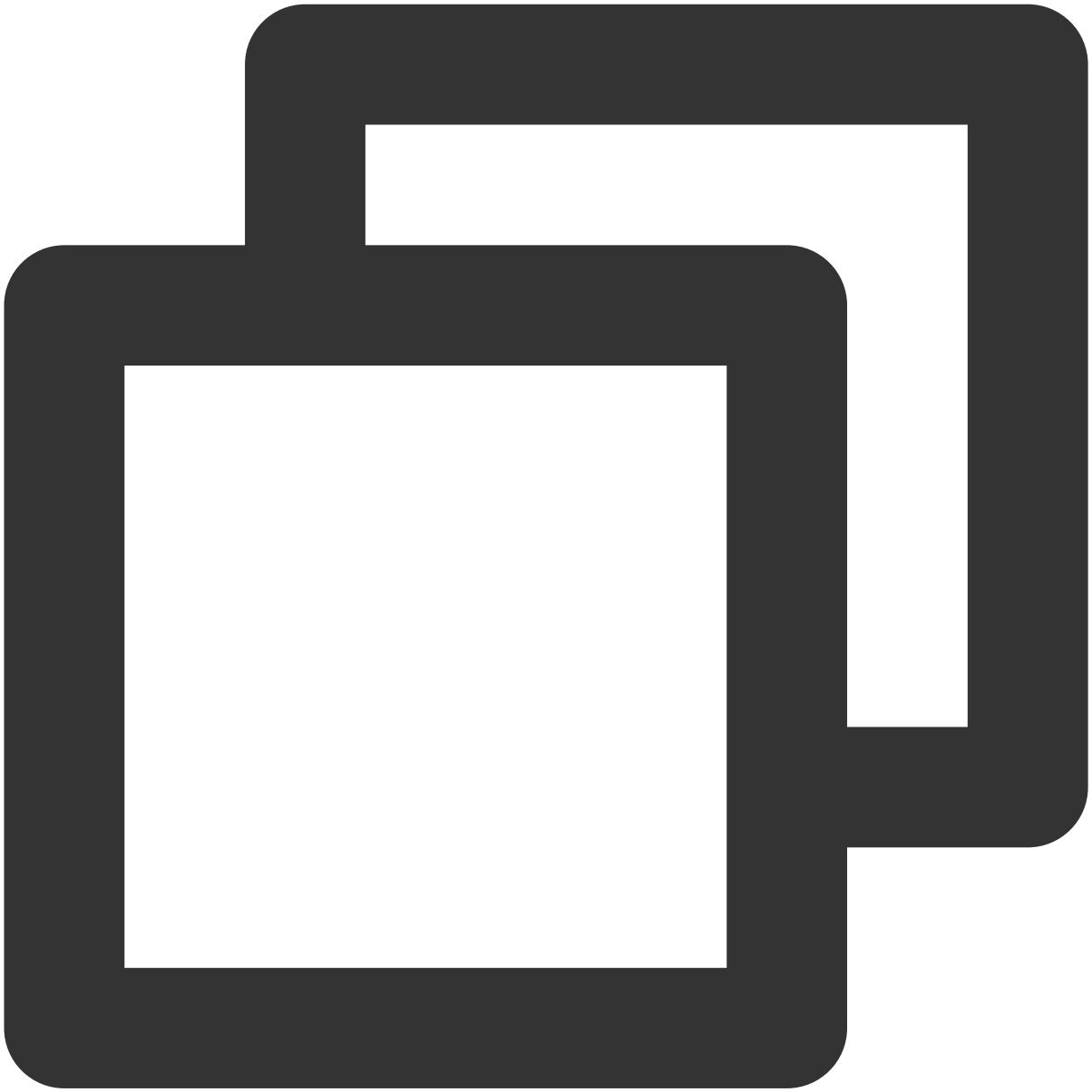
示例：



```
> SELECT unbase64('U3BhcmsgU1FM');
 Spark SQL
```

UNHEX

函数语法：



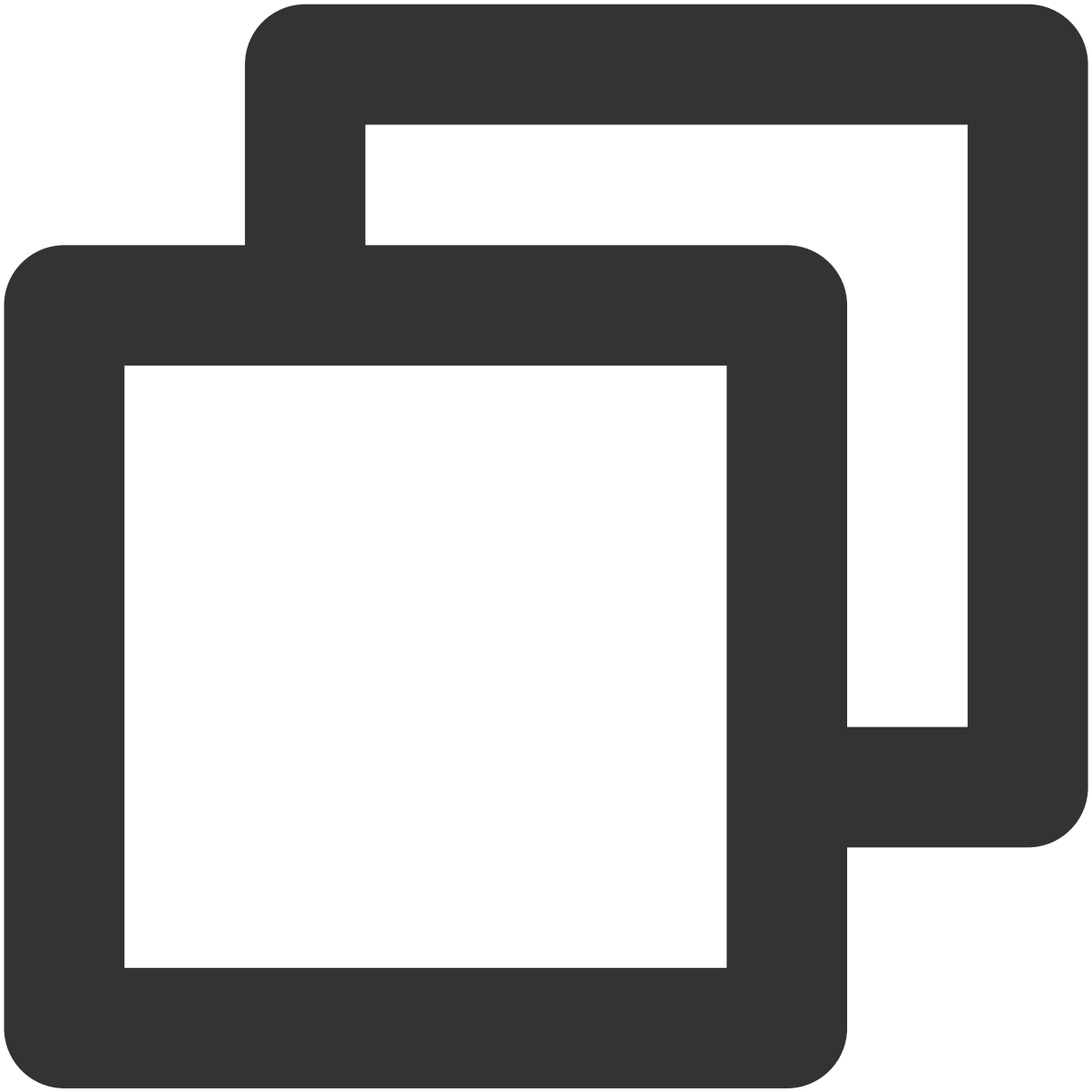
```
UNHEX(<str> string)
```

支持引擎：SparkSQL、Presto

使用说明：将十六进制的str转换为二进制。

返回类型：binary

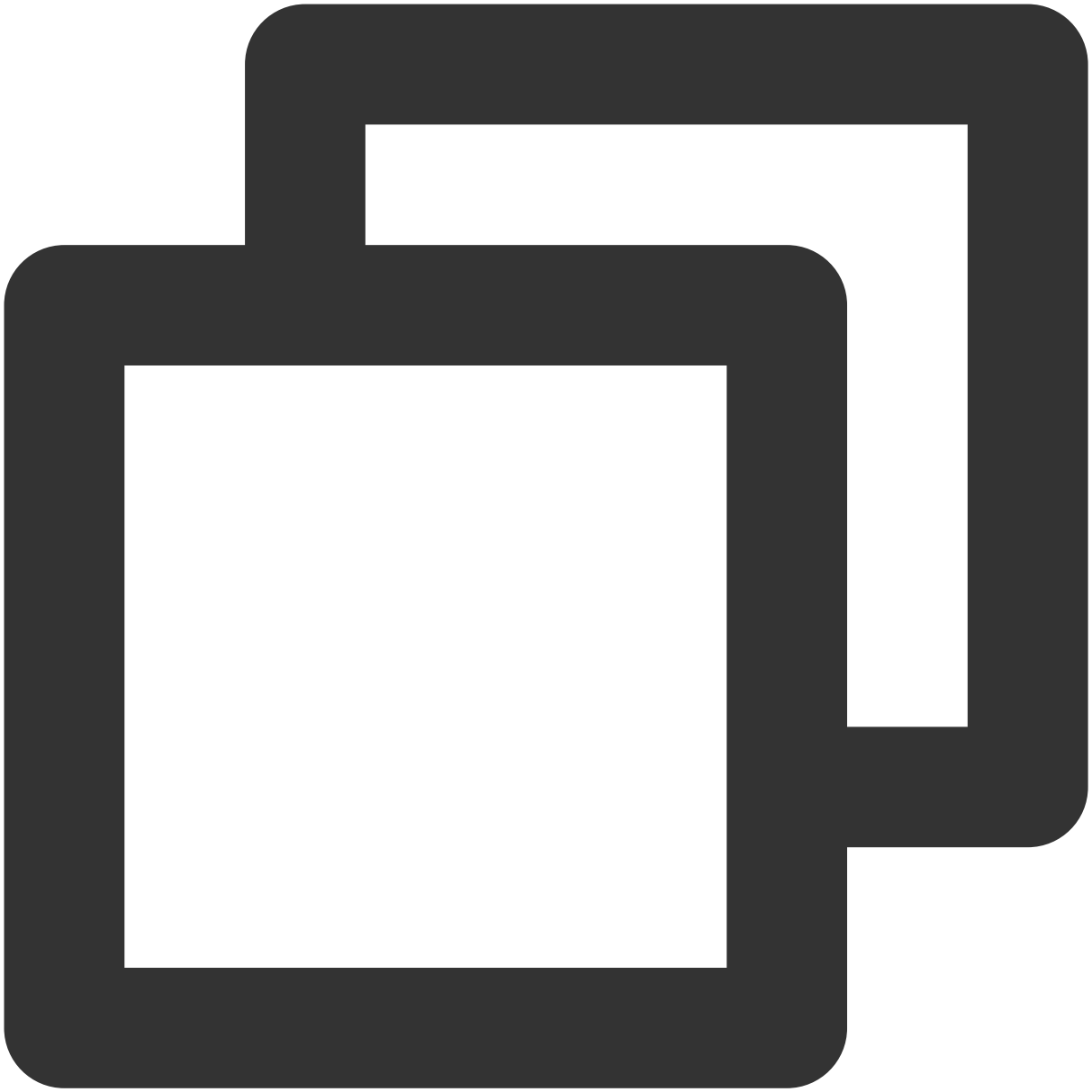
示例：



```
> select unhex('74656E63656E74');
 tencent
```

UPPER

函数语法：



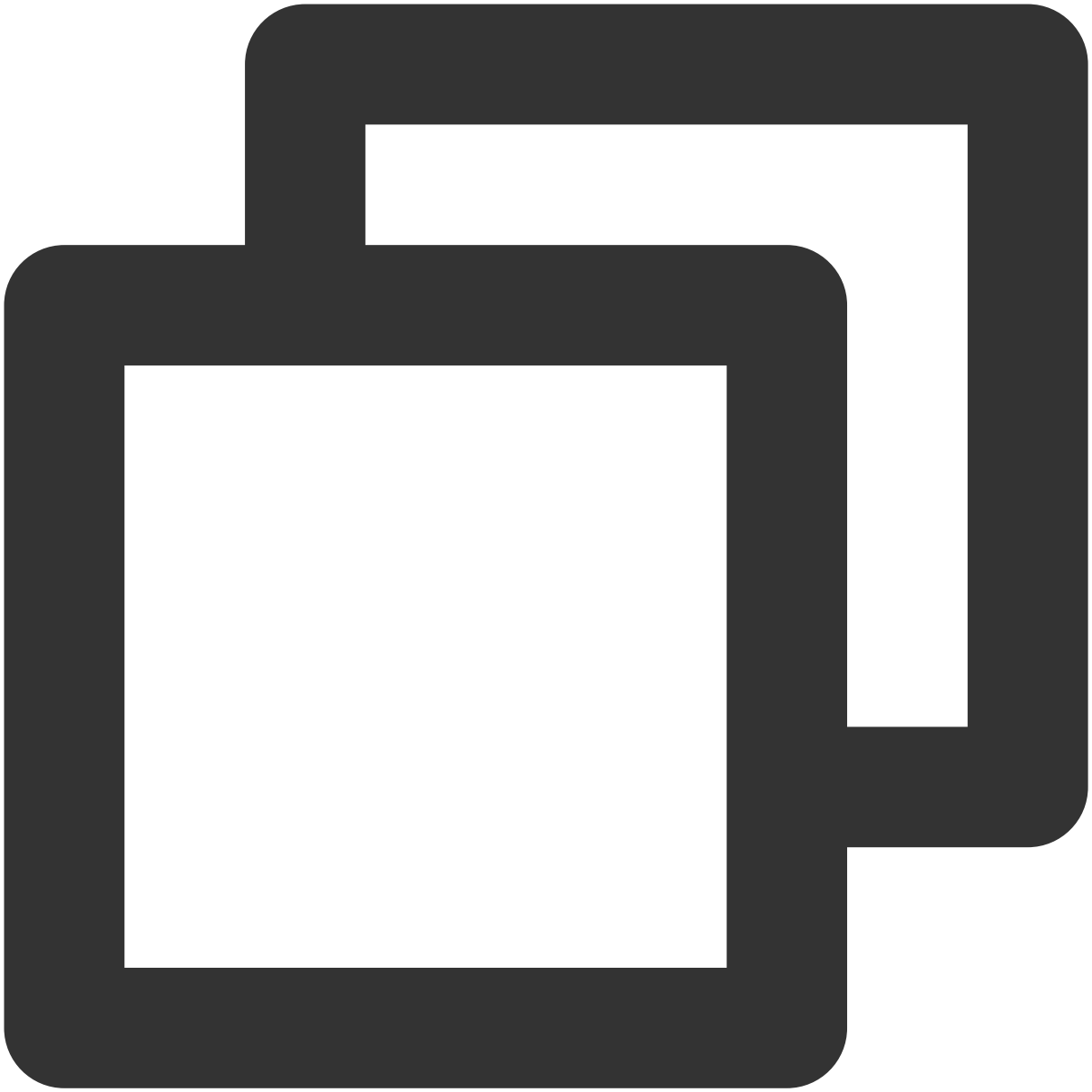
```
UPPER(<str> string)
```

支持引擎：SparkSQL、Presto

使用说明：返回所有字符都改为大写的str。

返回类型：string

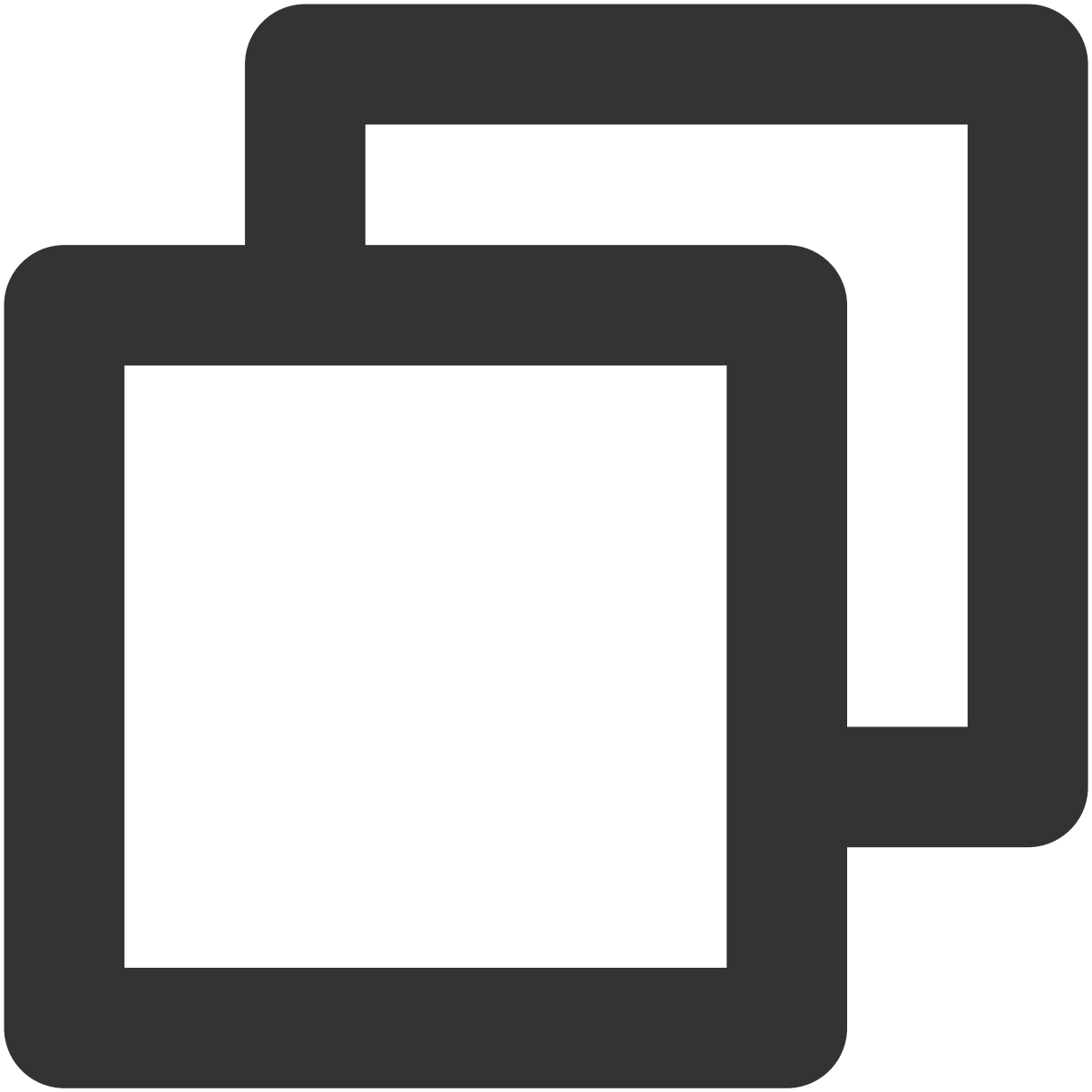
示例：



```
> SELECT upper('tencent');
TENCENT
```

UUID

函数语法：



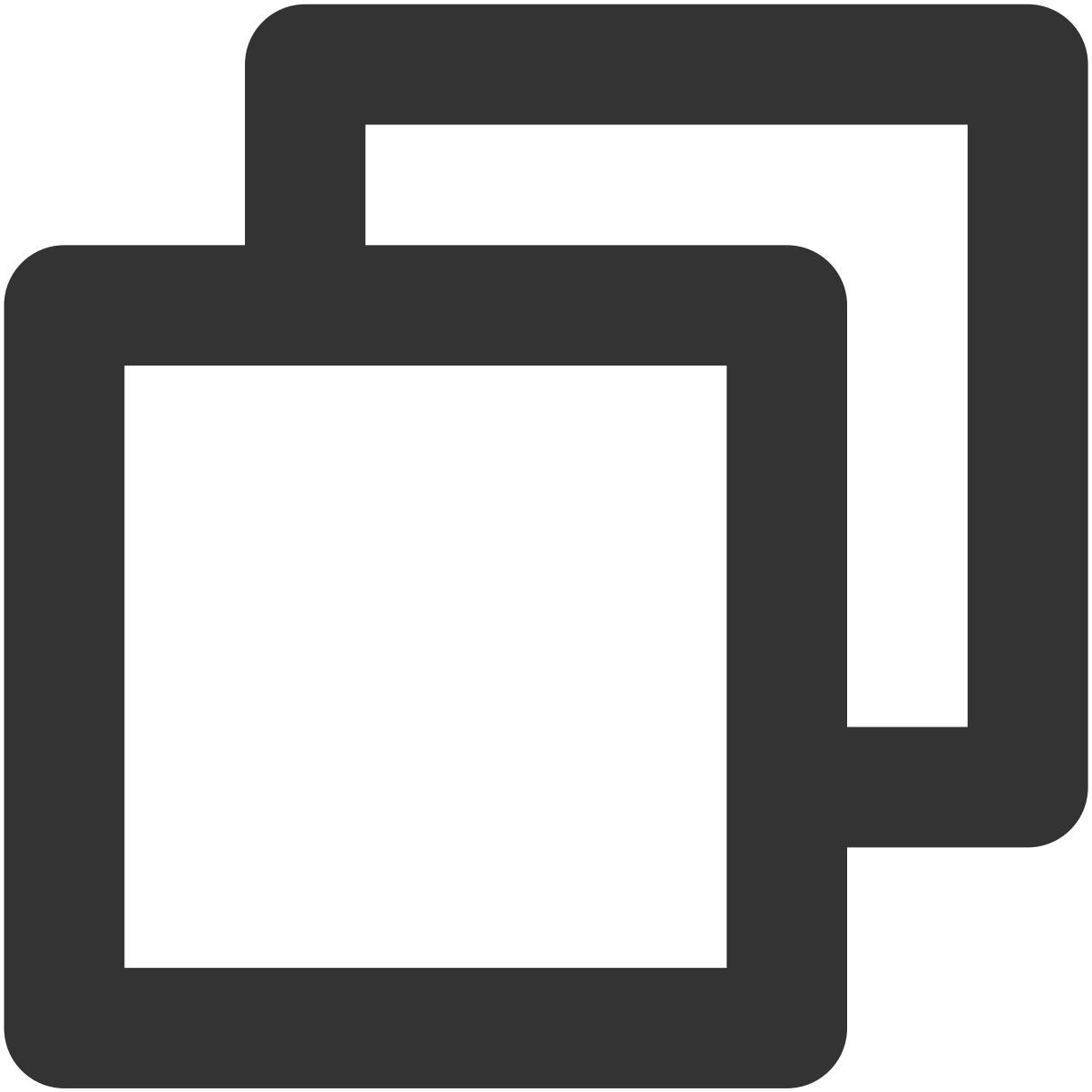
```
UUID()
```

支持引擎：SparkSQL、Presto

使用说明：返回一个36字符的UUID

返回类型：string

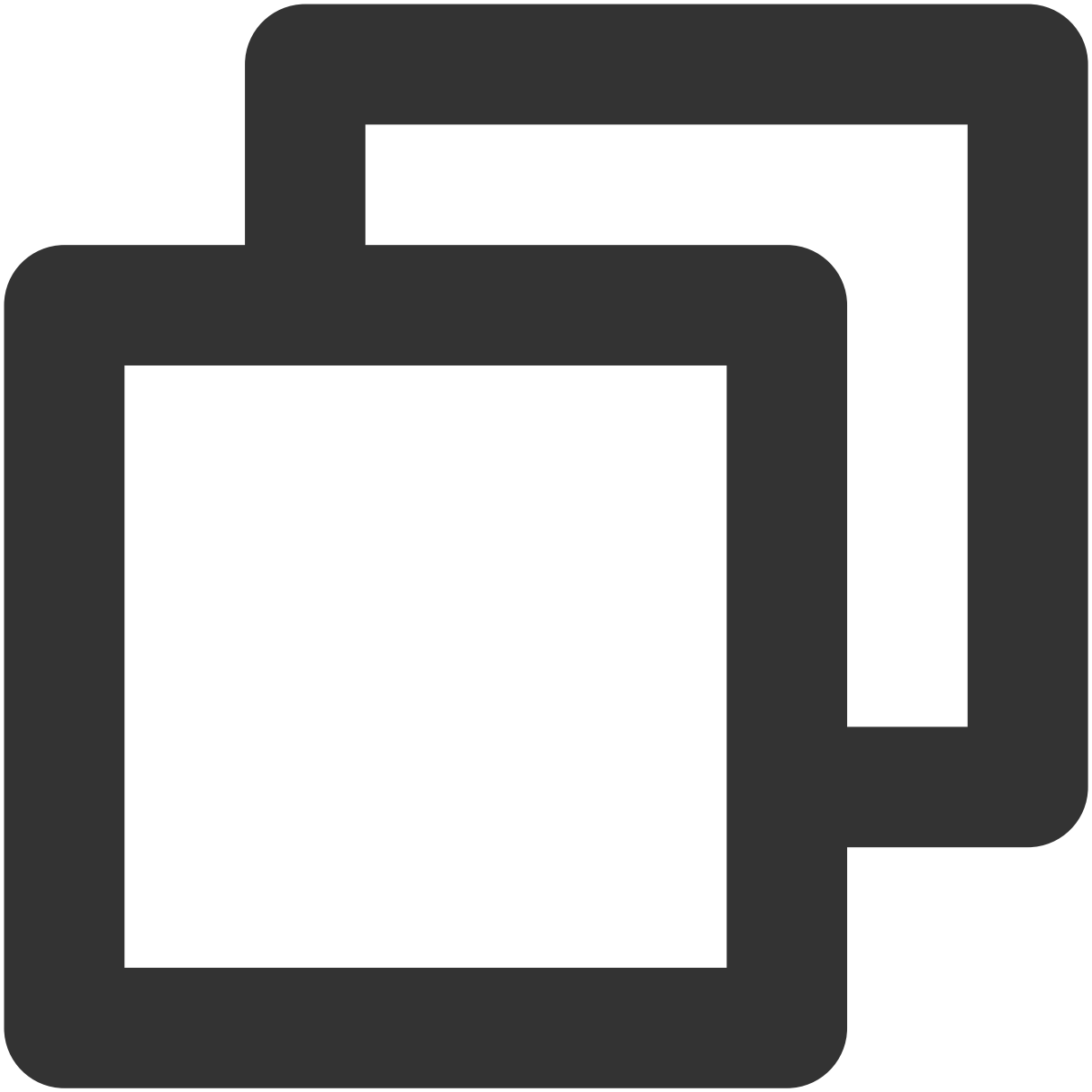
示例：



```
> SELECT uuid();
 46707d92-02f4-4817-8116-a4c3b23e6266
```

XPATH

函数语法：



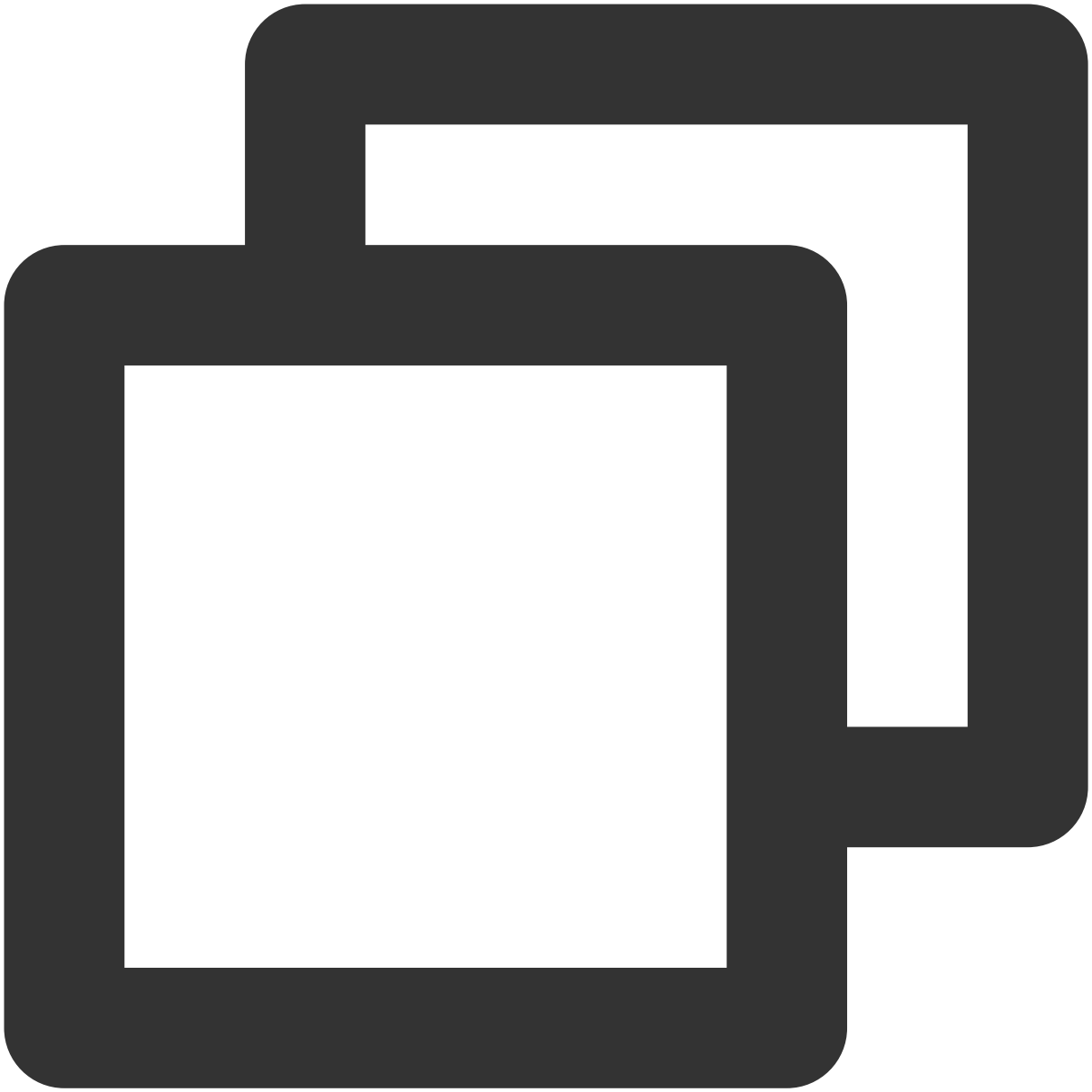
```
XPATH(<xml> string, <xpath> string)
```

支持引擎：SparkSQL、Presto

使用说明：返回xml节点中与XPath表达式匹配的字符串数组。

返回类型：array <string>

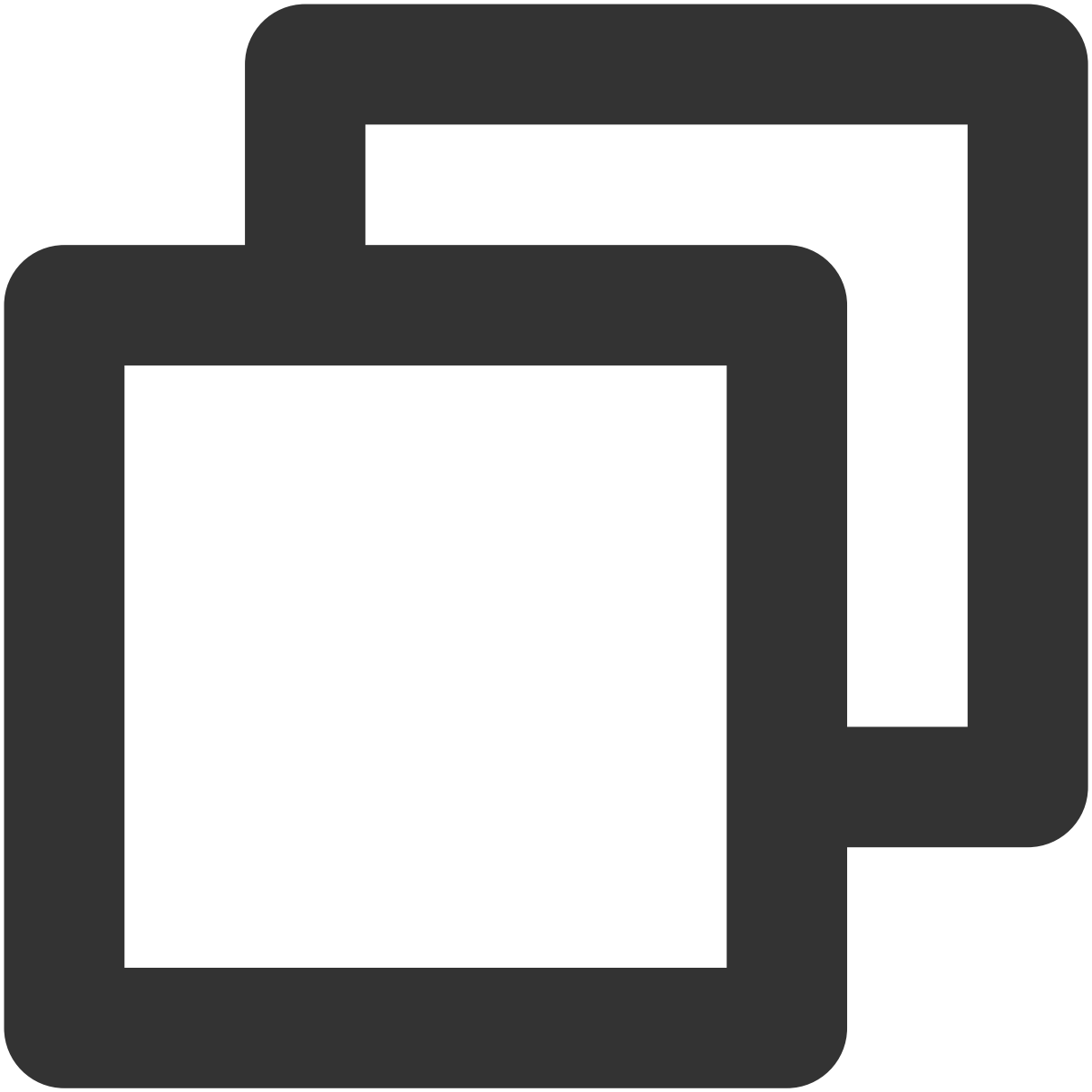
示例：



```
> SELECT xpath('<a><b>b1</b><b>b2</b><b>b3</b><c>c1</c><c>c2</c></a>','a/b/text()')
 ["b1","b2","b3"]
```

XPATH_BOOLEAN

函数语法：



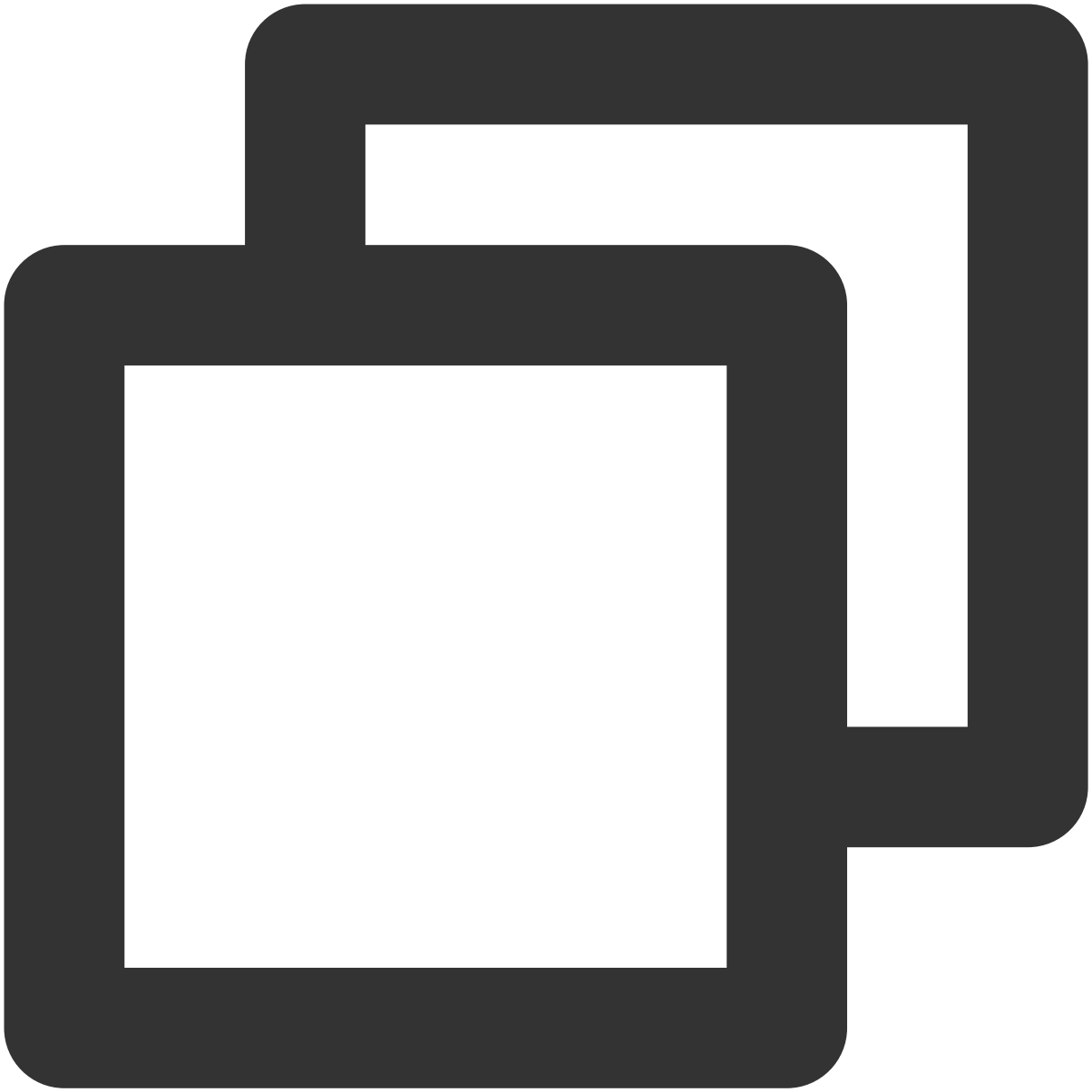
```
XPATH_BOOLEAN(<xml> string, <xpath> string)
```

支持引擎：SparkSQL、Presto

使用说明：如果XPath表达式的计算结果为true，或者如果找到匹配的节点，则返回true。

返回类型：boolean

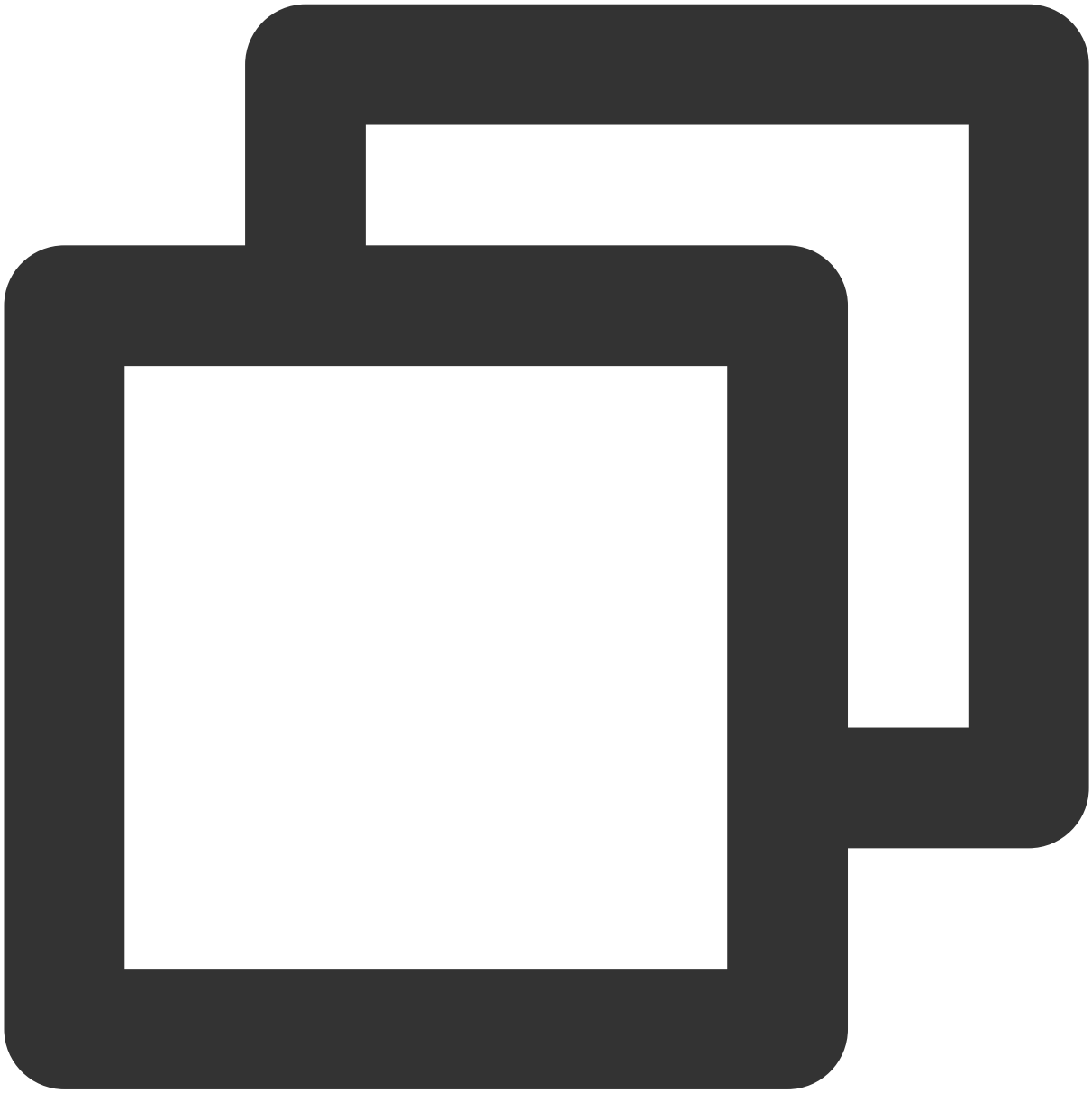
示例：



```
> SELECT xpath_boolean('<a><b>1</b></a>','a/b');
 true
```

XPATH_DOUBLE

函数语法：



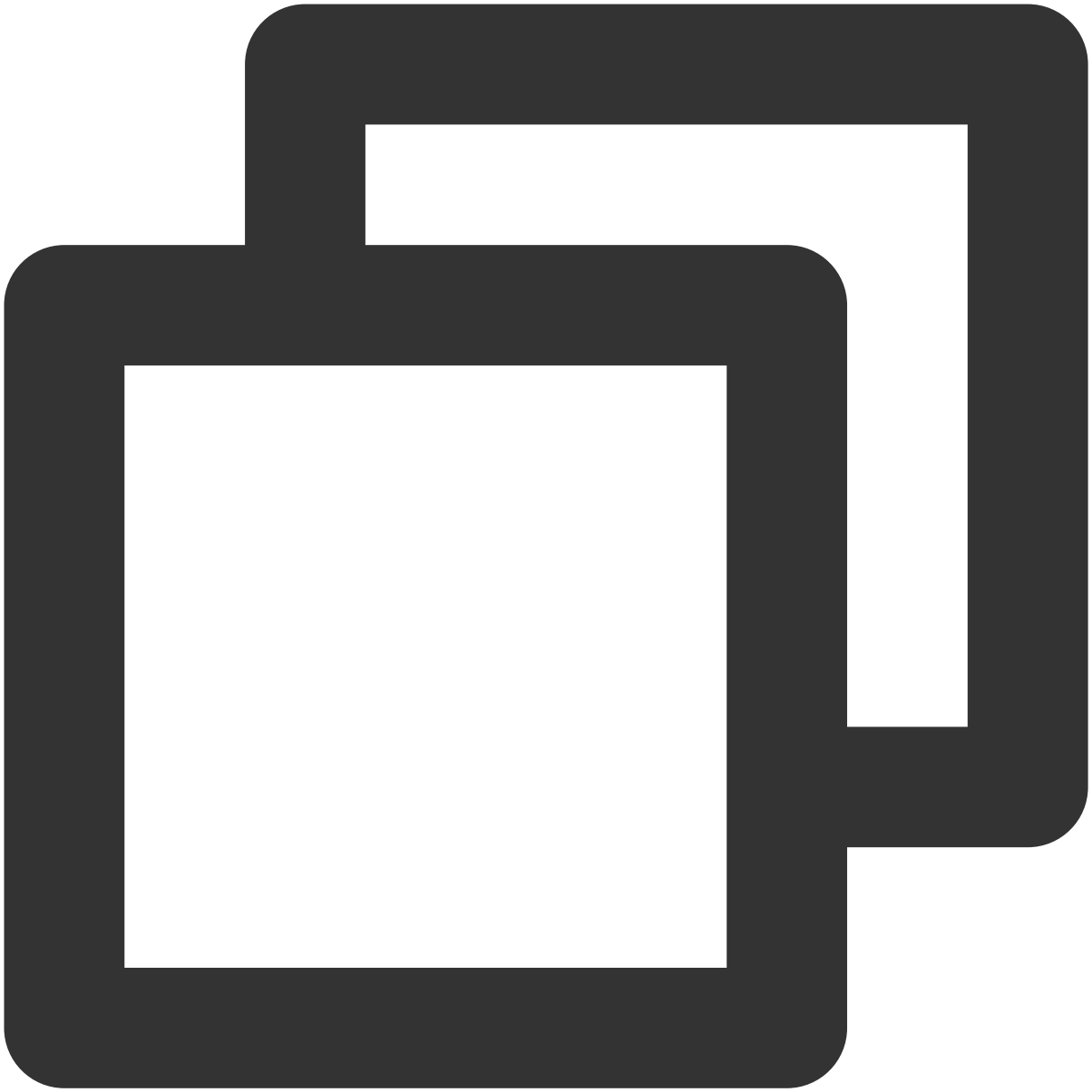
```
XPATH_DOUBLE(<xml> string, <xpath> string)
```

支持引擎：SparkSQL、Presto

使用说明：返回一个double类型的值，如果未找到匹配项，则返回零；如果找到匹配项但该值不是数字，则返回NaN。

返回类型：double

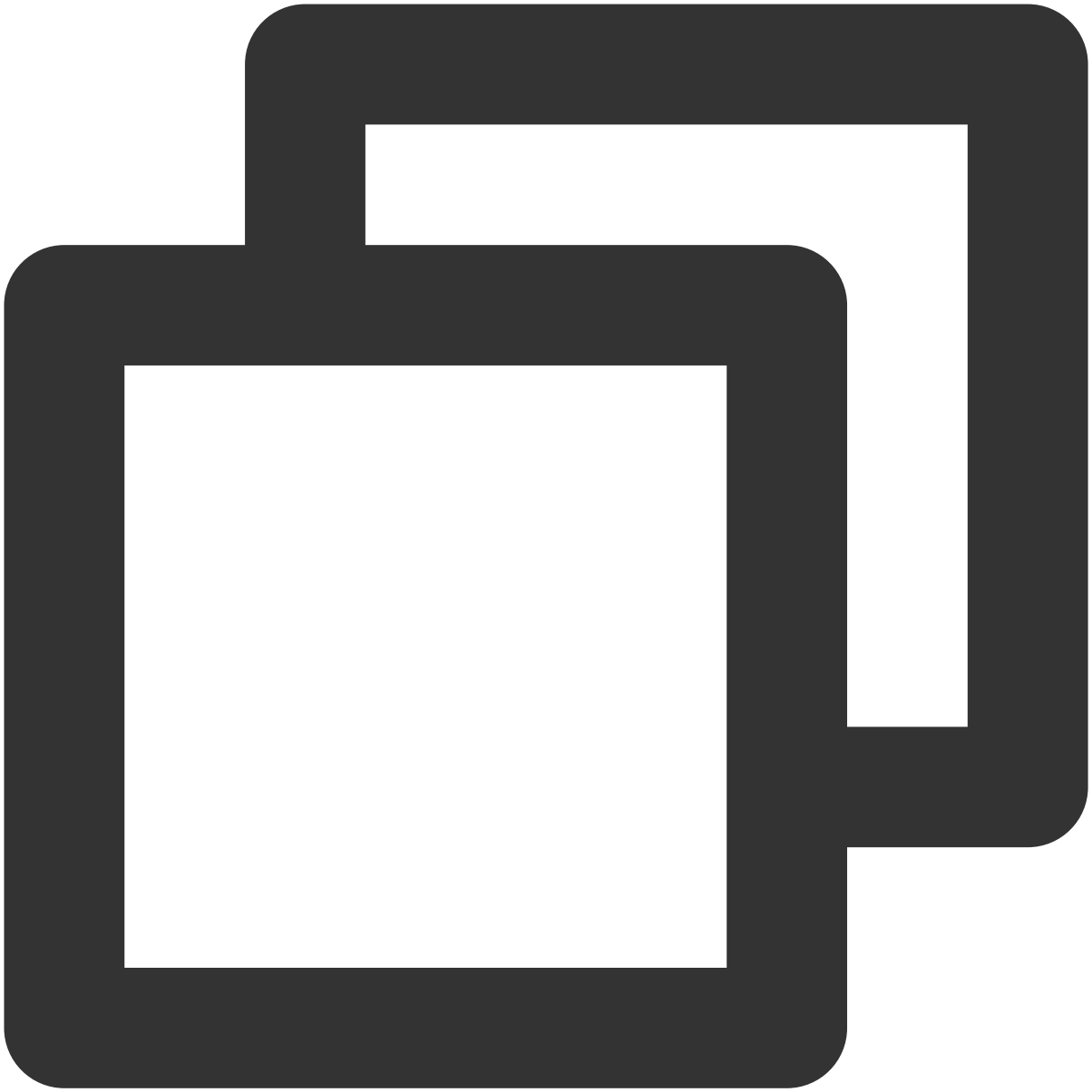
示例：



```
> SELECT xpath_double('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3.0
```

XPATH_NUMBER

函数语法：



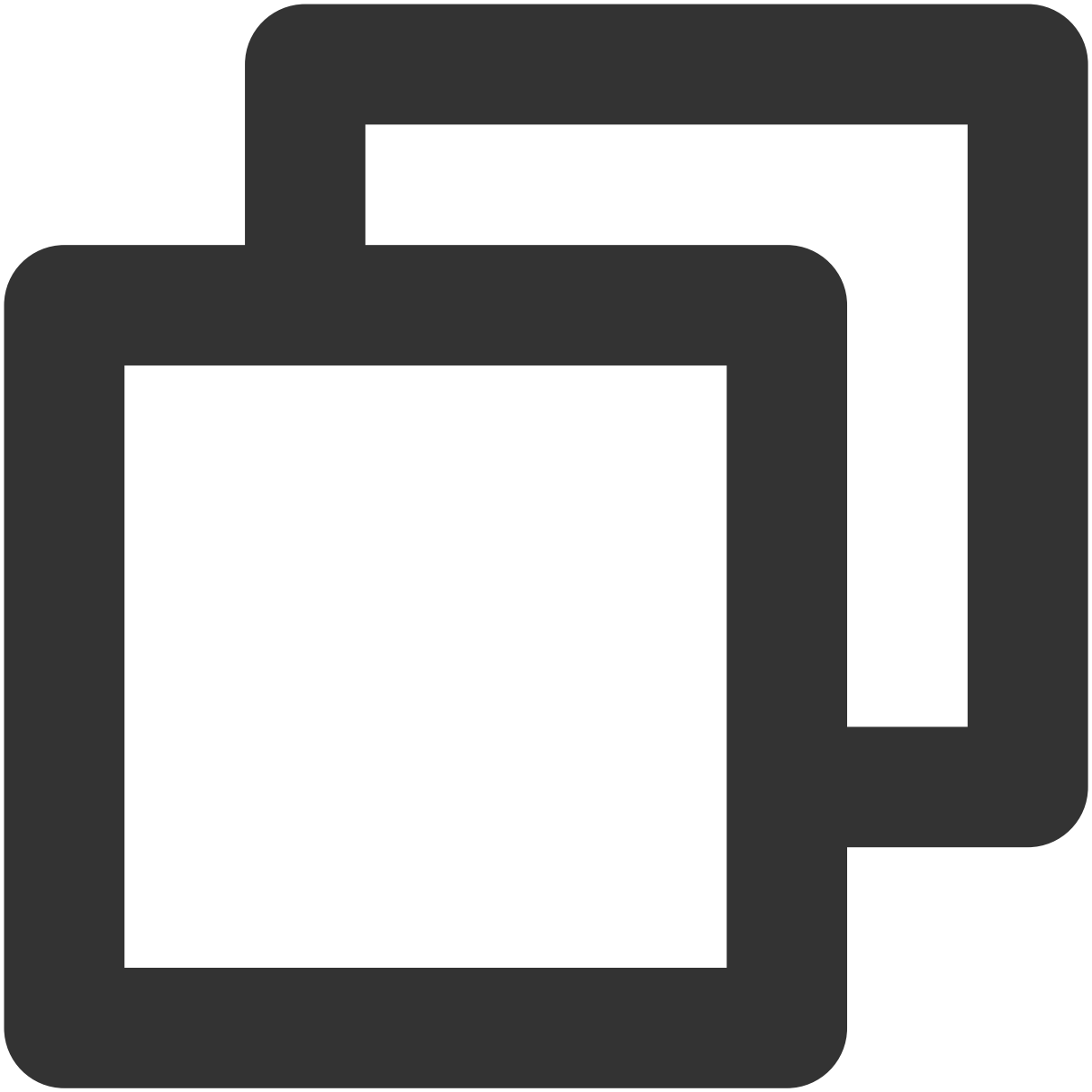
```
XPATH_NUMBER(<xml> string, <xpath> string)
```

支持引擎：SparkSQL、Presto

使用说明：返回一个double类型的值，如果未找到匹配项，则返回零；如果找到匹配项但该值不是数字，则返回NaN。

返回类型：double

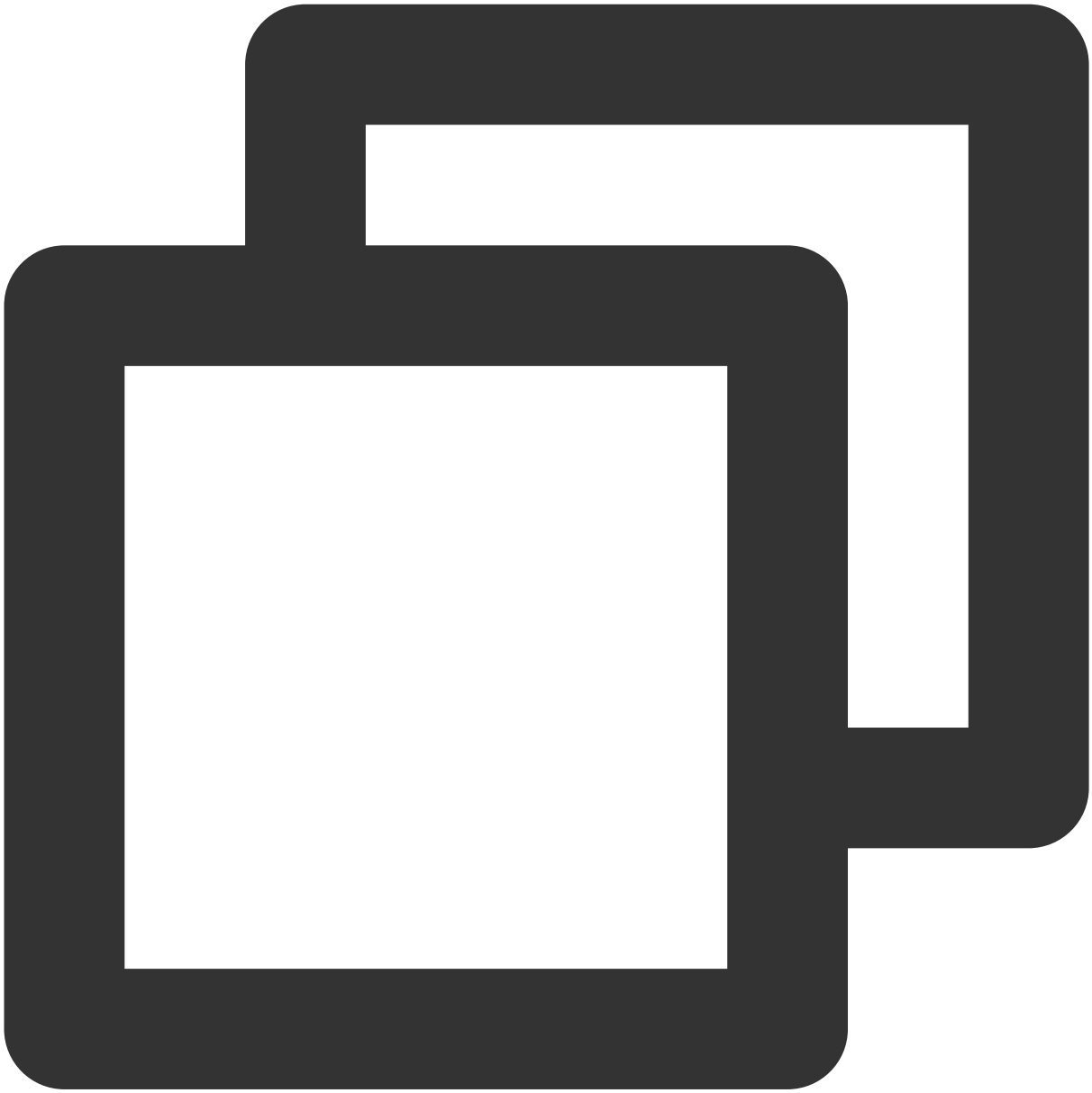
示例：



```
> SELECT xpath_number('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3.0
```

XPATH_FLOAT

函数语法：



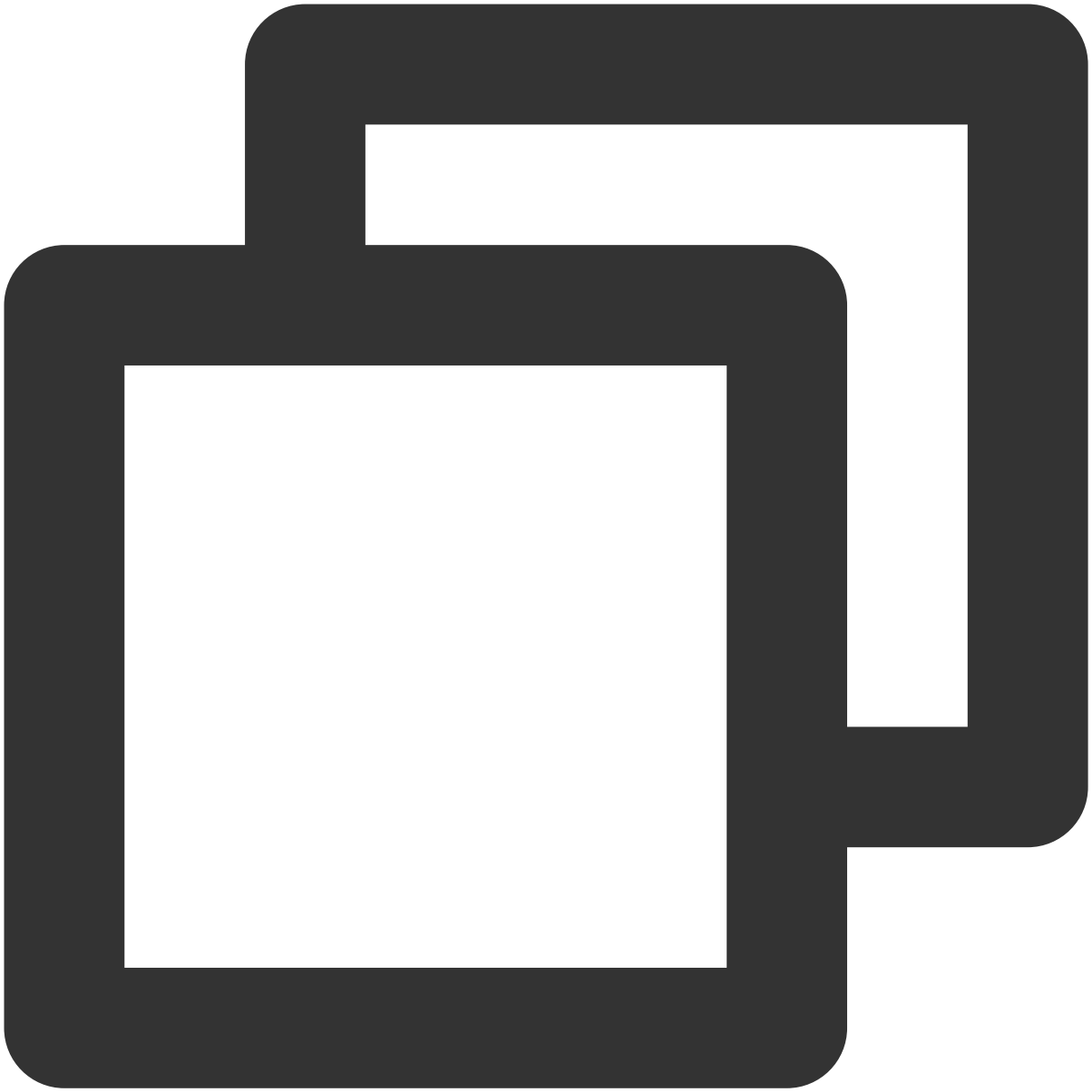
```
XPATH_FLOAT(<xml> string, <xpath> string)
```

支持引擎：SparkSQL、Presto

使用说明：返回float类型的值，如果未找到匹配项，则返回零；如果找到匹配项但该值为非数字，则返回NaN。

返回类型：float

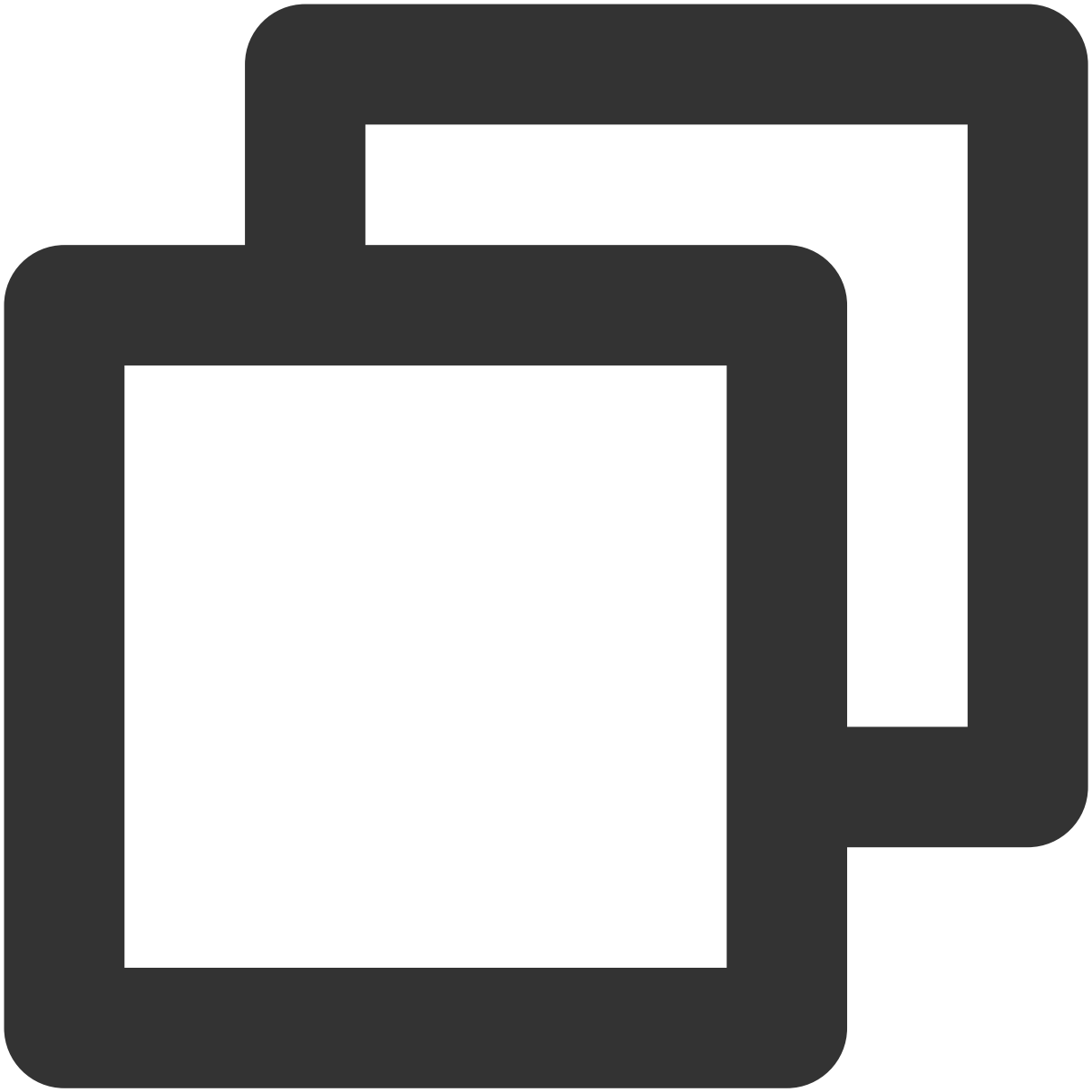
示例：



```
> SELECT xpath_float('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3.0
```

XPATH_INT

函数语法：



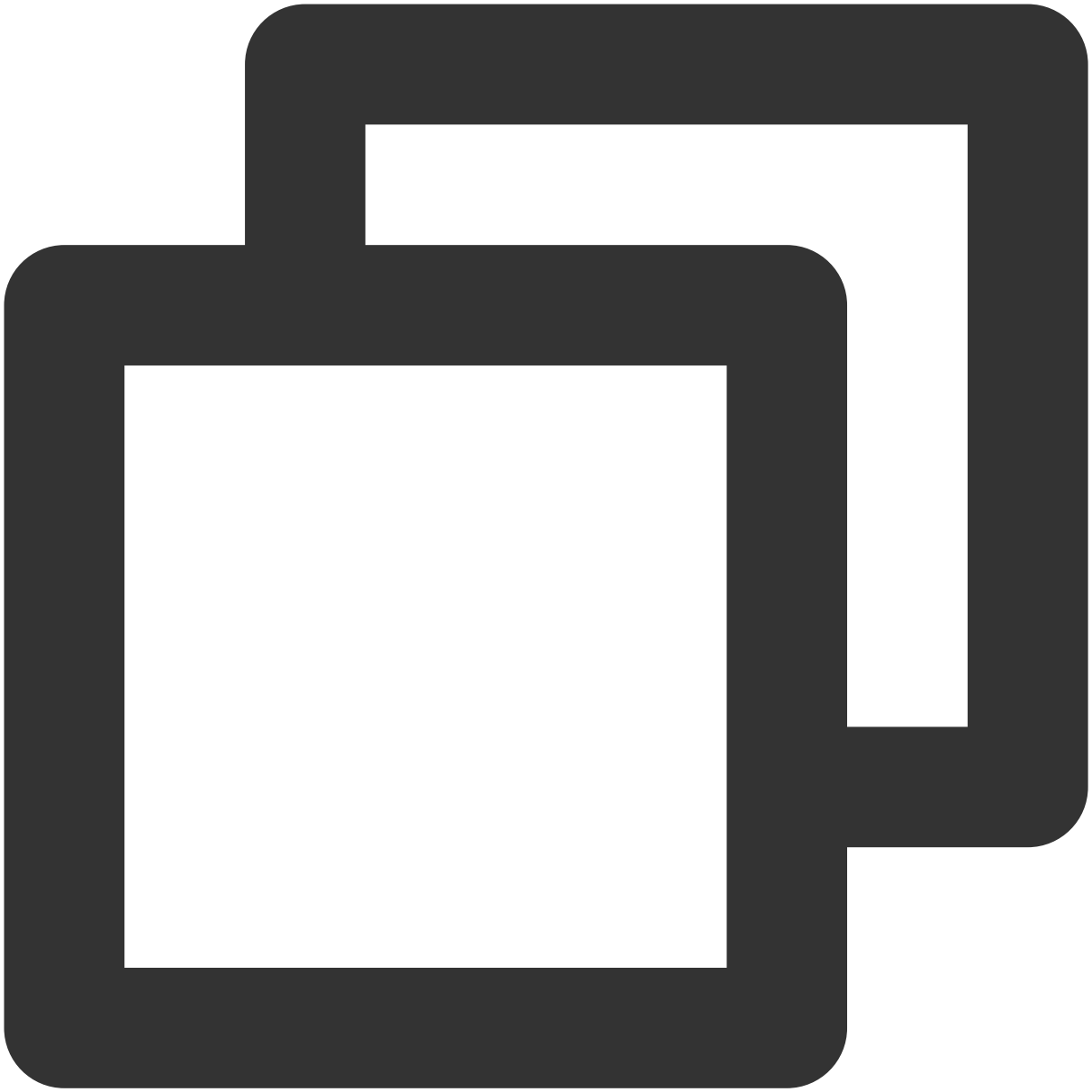
```
XPATH_INT(<xml> string, <xpath> string)
```

支持引擎：SparkSQL、Presto

使用说明：返回int类型的值，如果未找到匹配项，则返回零；如果找到匹配项但该值为非数字，则返回NaN。

返回类型：integer

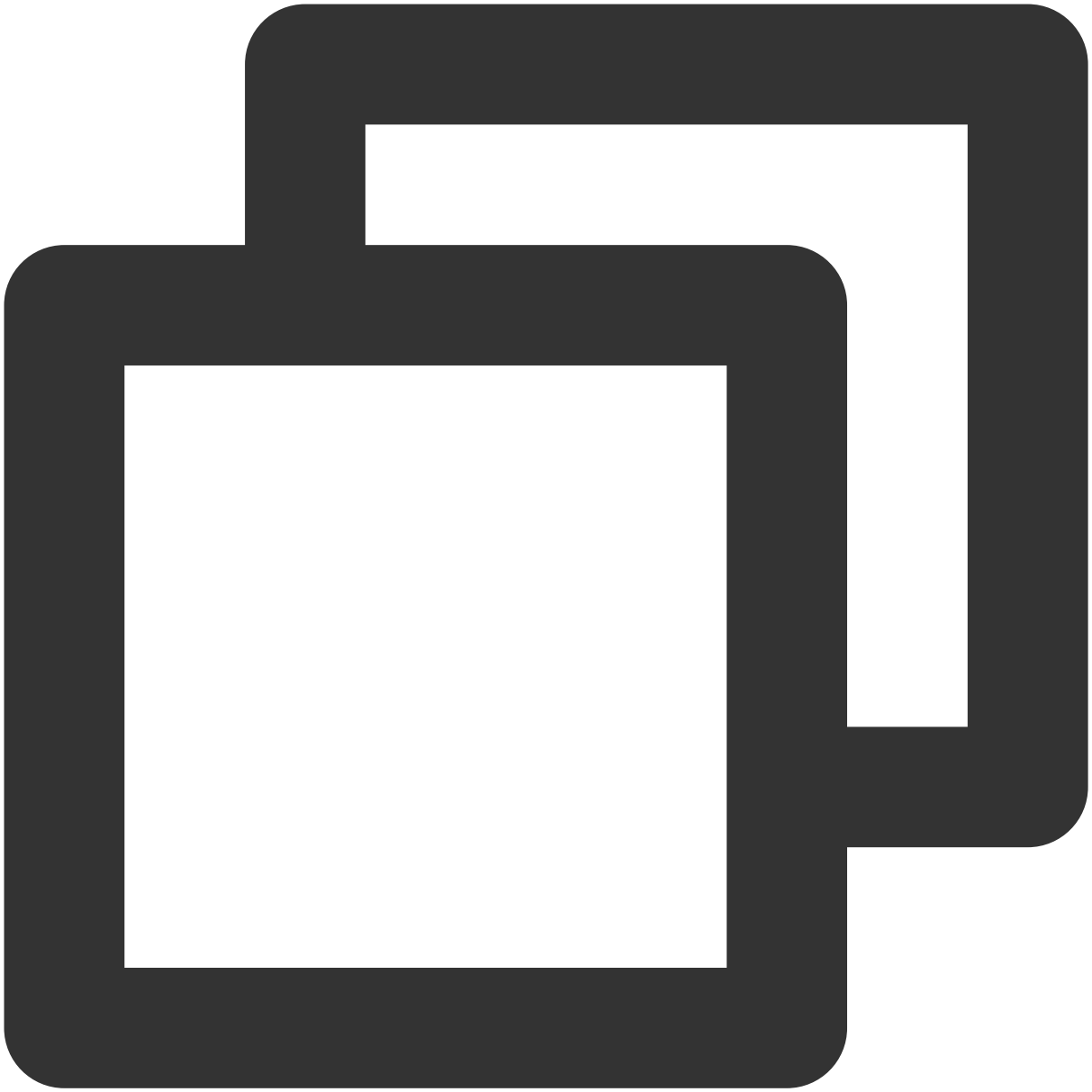
示例：



```
> SELECT xpath_int('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3
```

XPATH_LONG

函数语法：



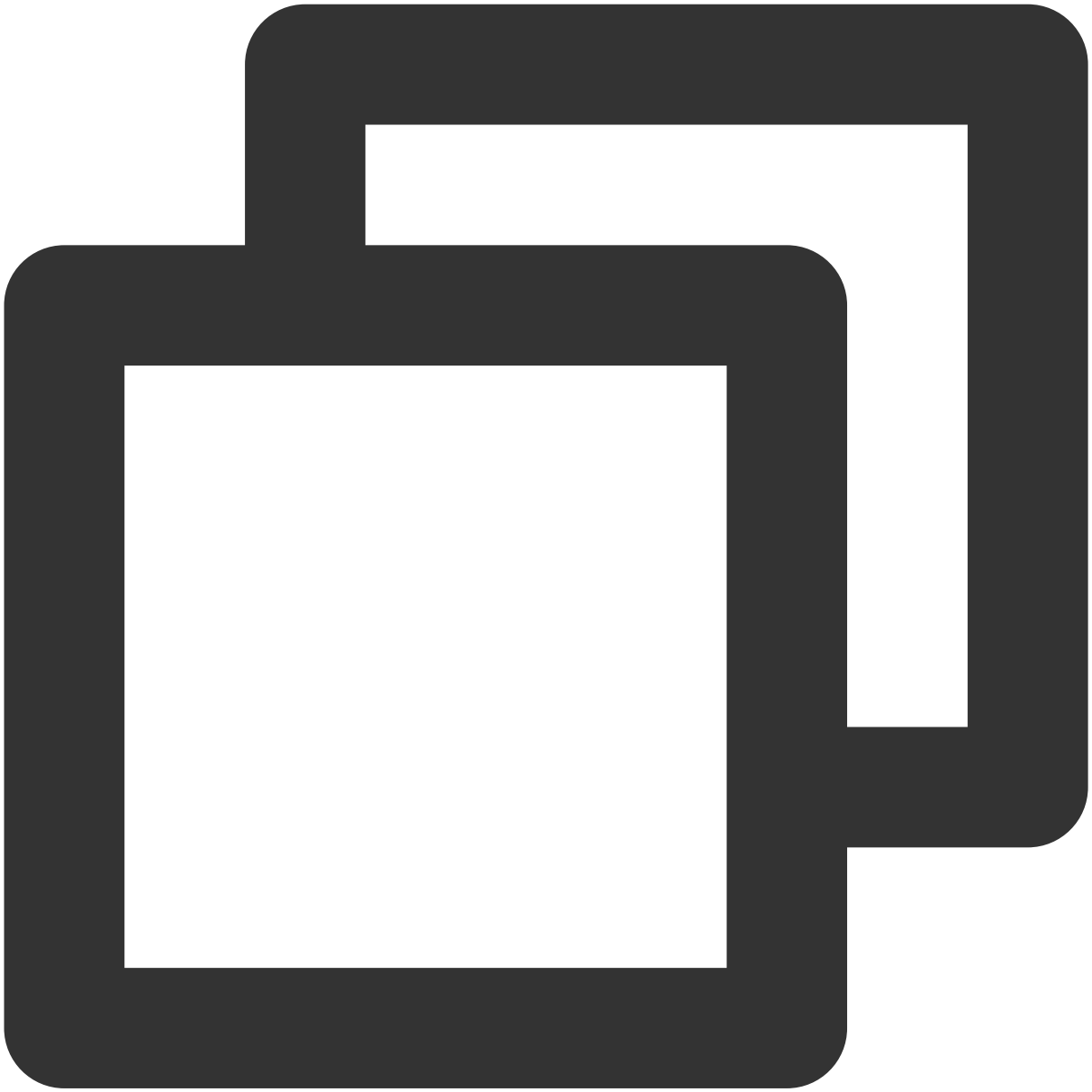
```
XPATH_LONG(<xml> string, <xpath> string)
```

支持引擎：SparkSQL、Presto

使用说明：返回bigint类型的值，如果未找到匹配项，则返回零；如果找到匹配项但该值为非数字，则返回NaN。

返回类型：bigint

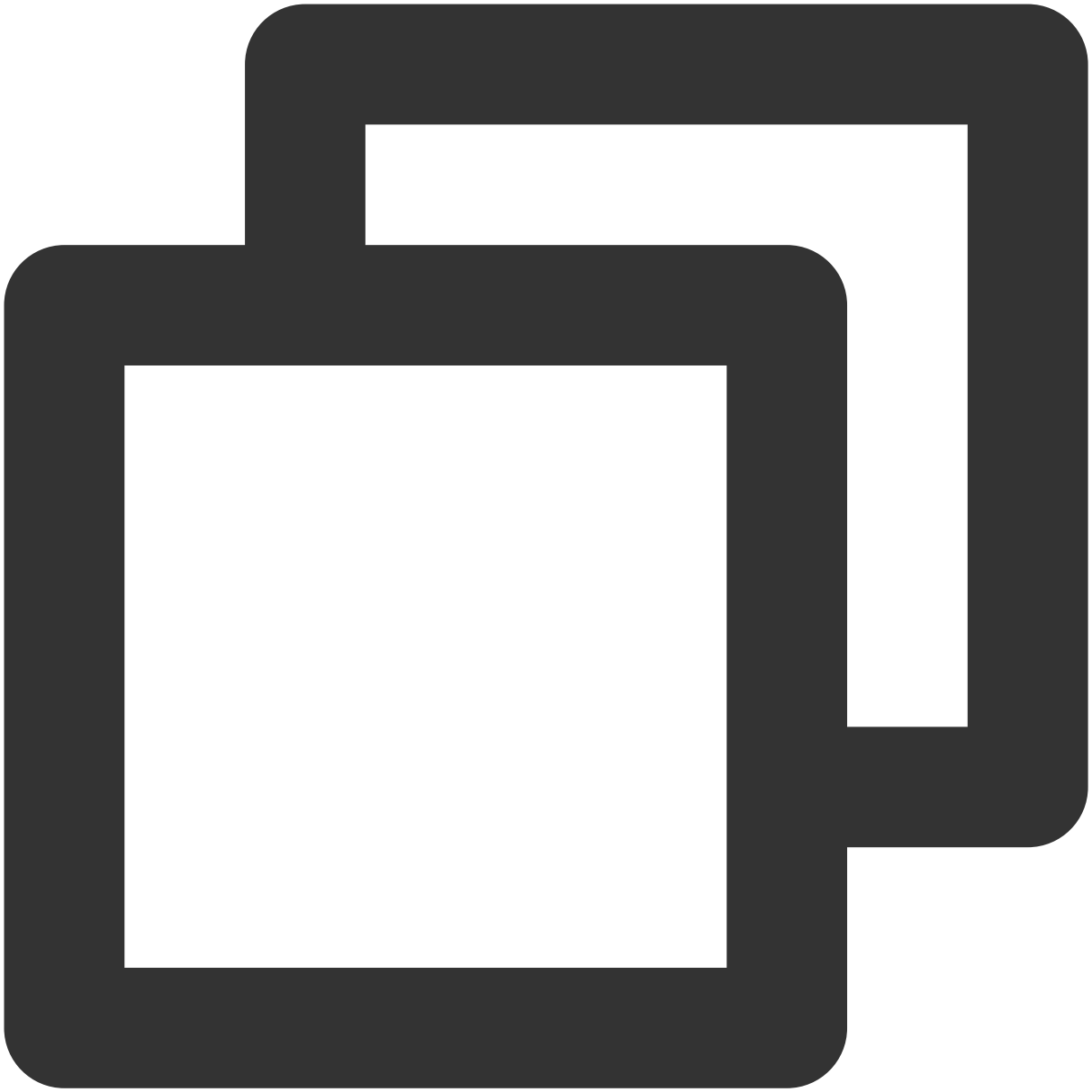
示例：



```
> SELECT xpath_long('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3
```

XPATH_SHORT

函数语法：



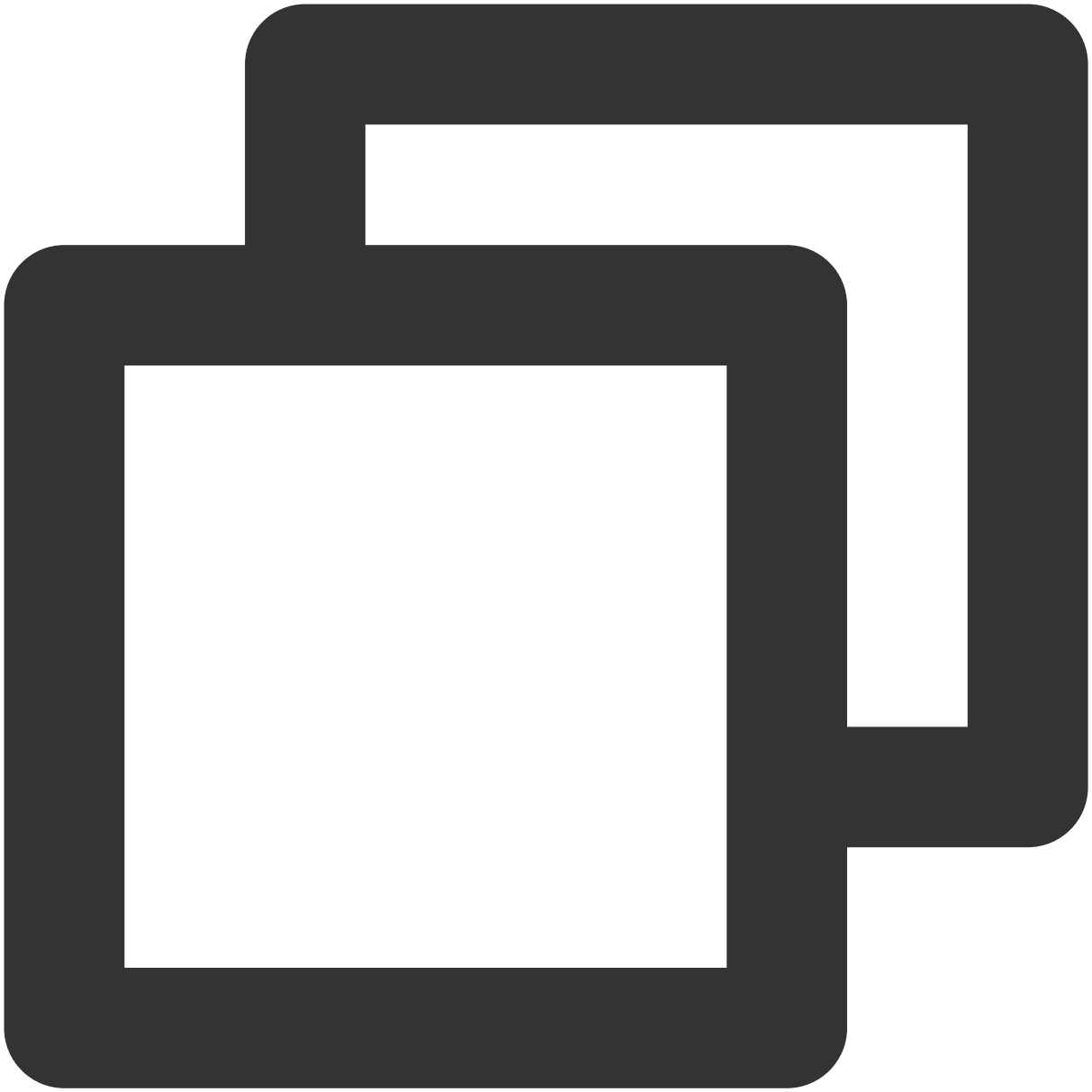
```
XPATH_SHORT(<xml> string, <xpath> string)
```

支持引擎：SparkSQL、Presto

使用说明：返回short类型的值，如果未找到匹配项，则返回零；如果找到匹配项但该值为非数字，则返回NaN。

返回类型：short

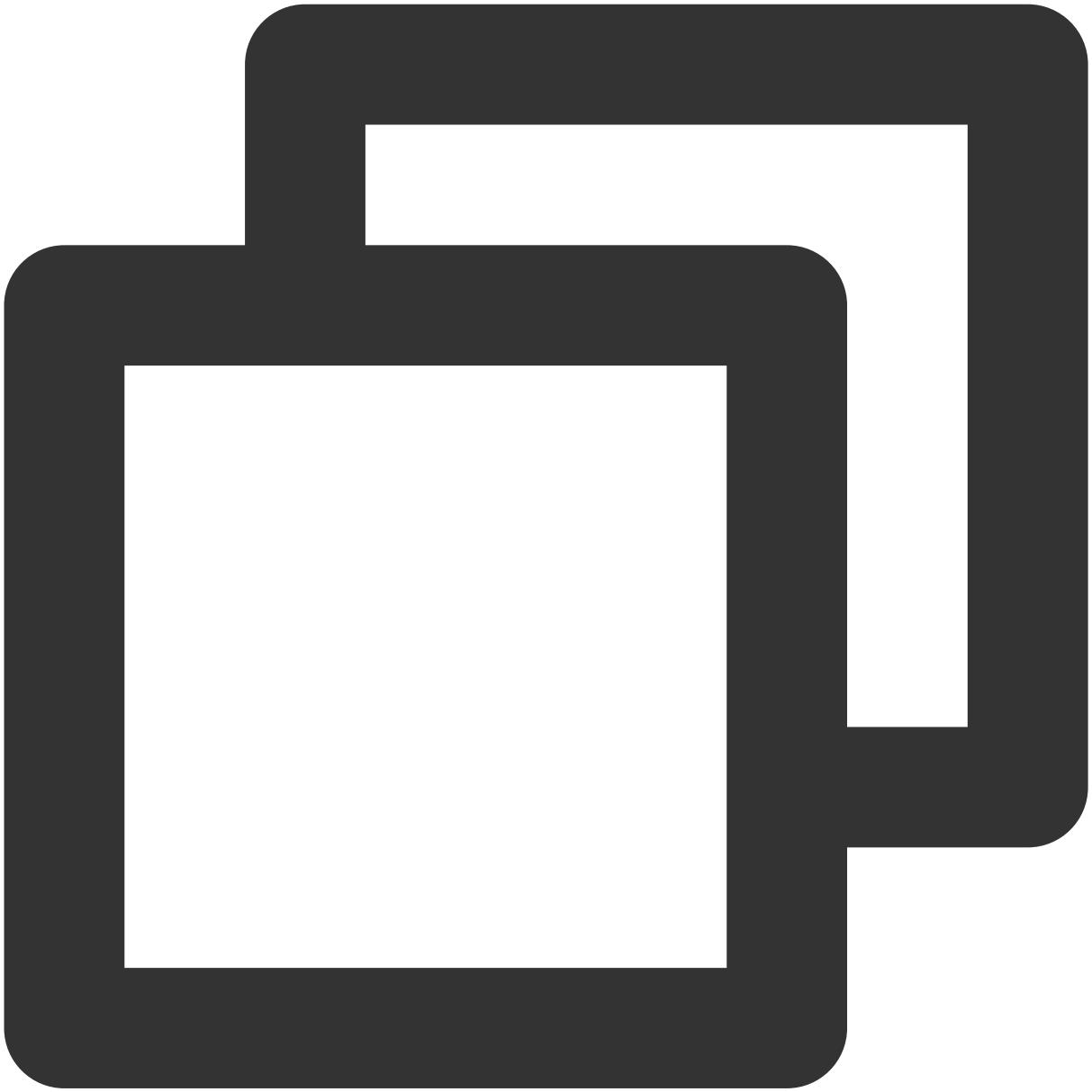
示例：



```
> SELECT xpath_short('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3
```

XPATH_STRING

函数语法：



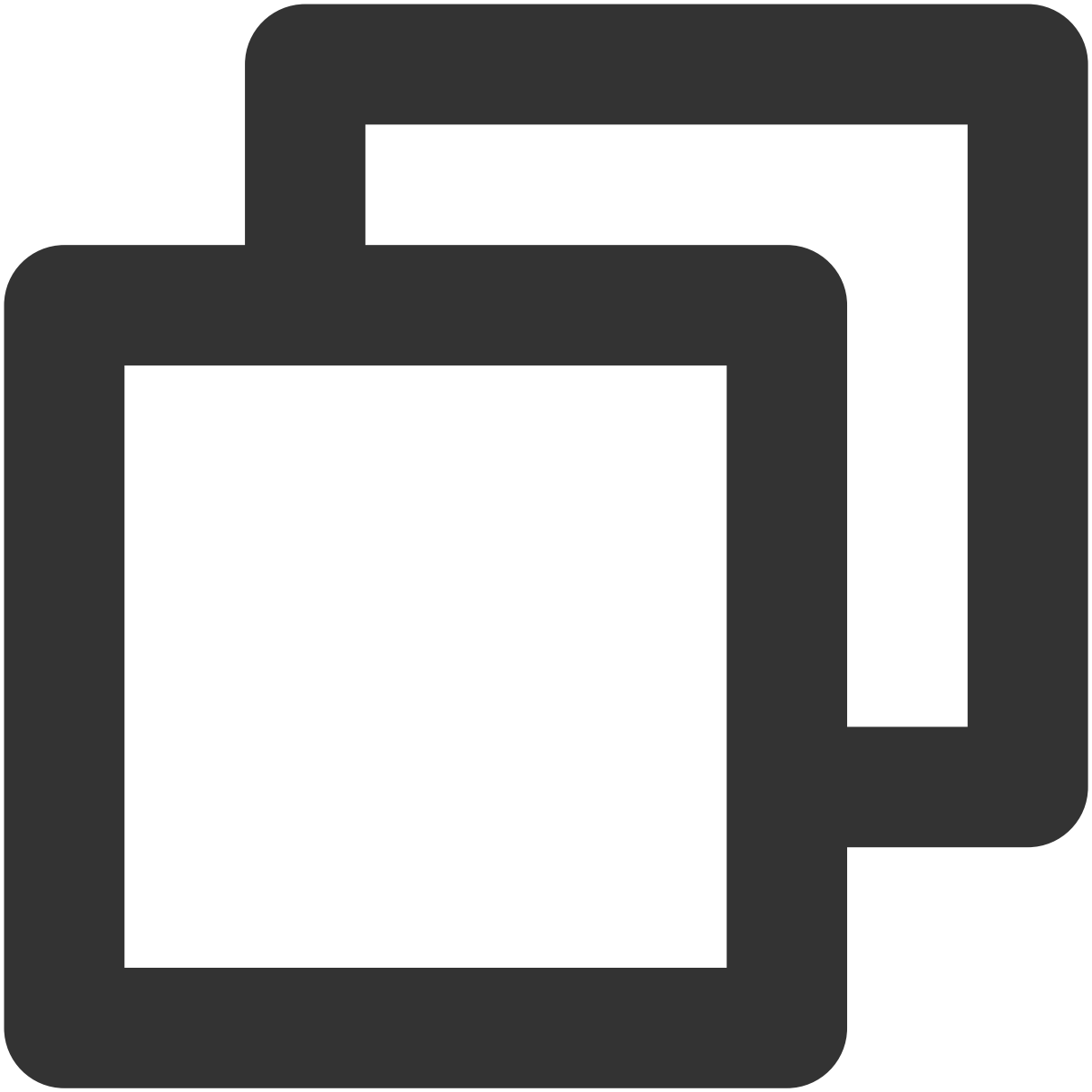
```
XPATH_STRING(<xml> string, <xpath> string)
```

支持引擎：SparkSQL、Presto

使用说明：返回与XPath表达式匹配的第一个xml节点的文本内容。

返回类型：string

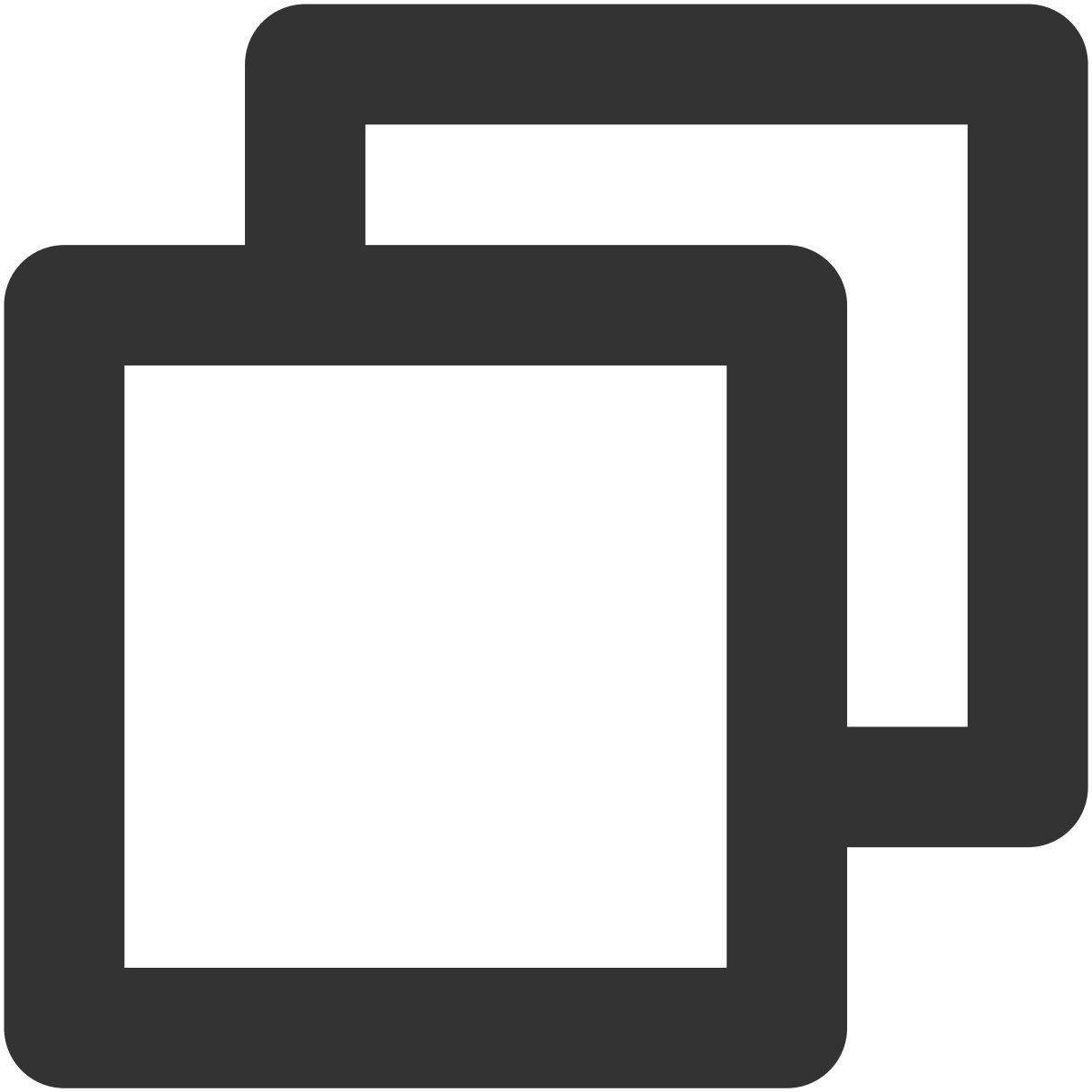
示例：



```
> SELECT xpath_string('<a><b>b</b><c>cc</c></a>','a/c');
 cc
```

REGEXP_EXTRACT

函数语法：



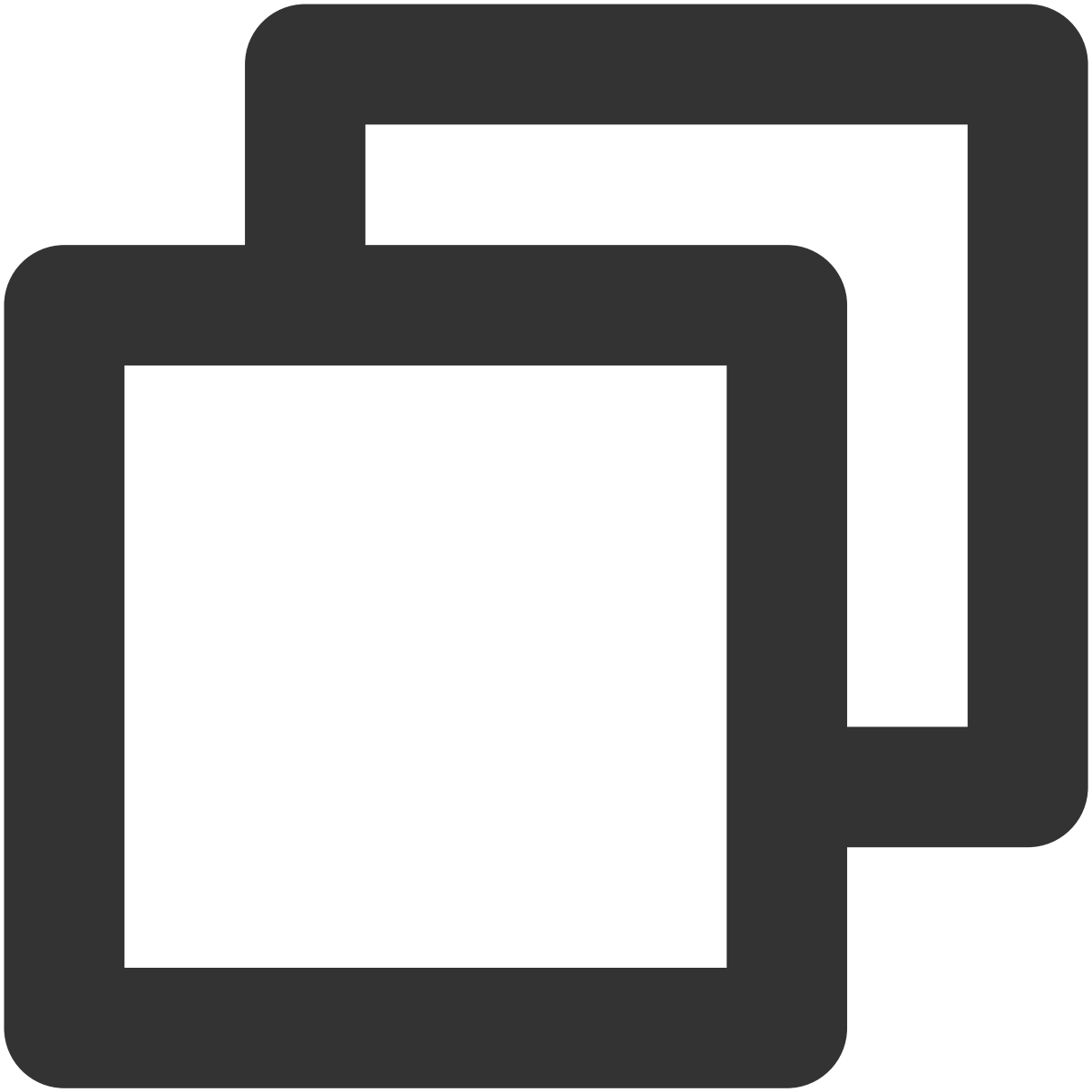
```
REGEXP_EXTRACT(<str> string, <regexp> string[, <idx> integer])
```

支持引擎：SparkSQL、Presto

使用说明：提取str中与regexp表达式匹配并对应于regex组索引idx的第一个字符串。

返回类型：string

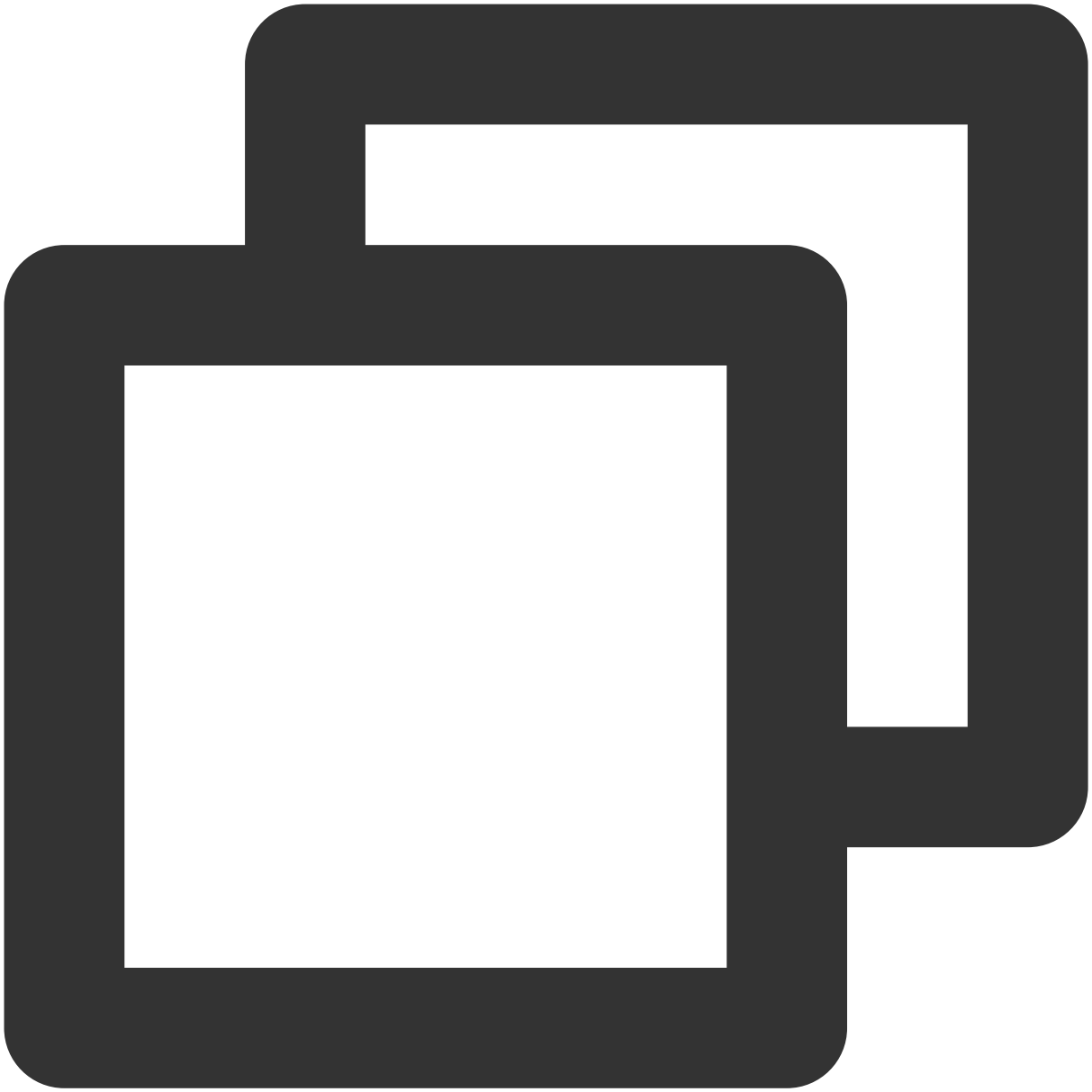
示例：



```
> SELECT regexp_extract('100-200', '(\\\\d+)-(\\\\d+)', 1);
 100
```

REGEXP_EXTRACT_ALL

函数语法：



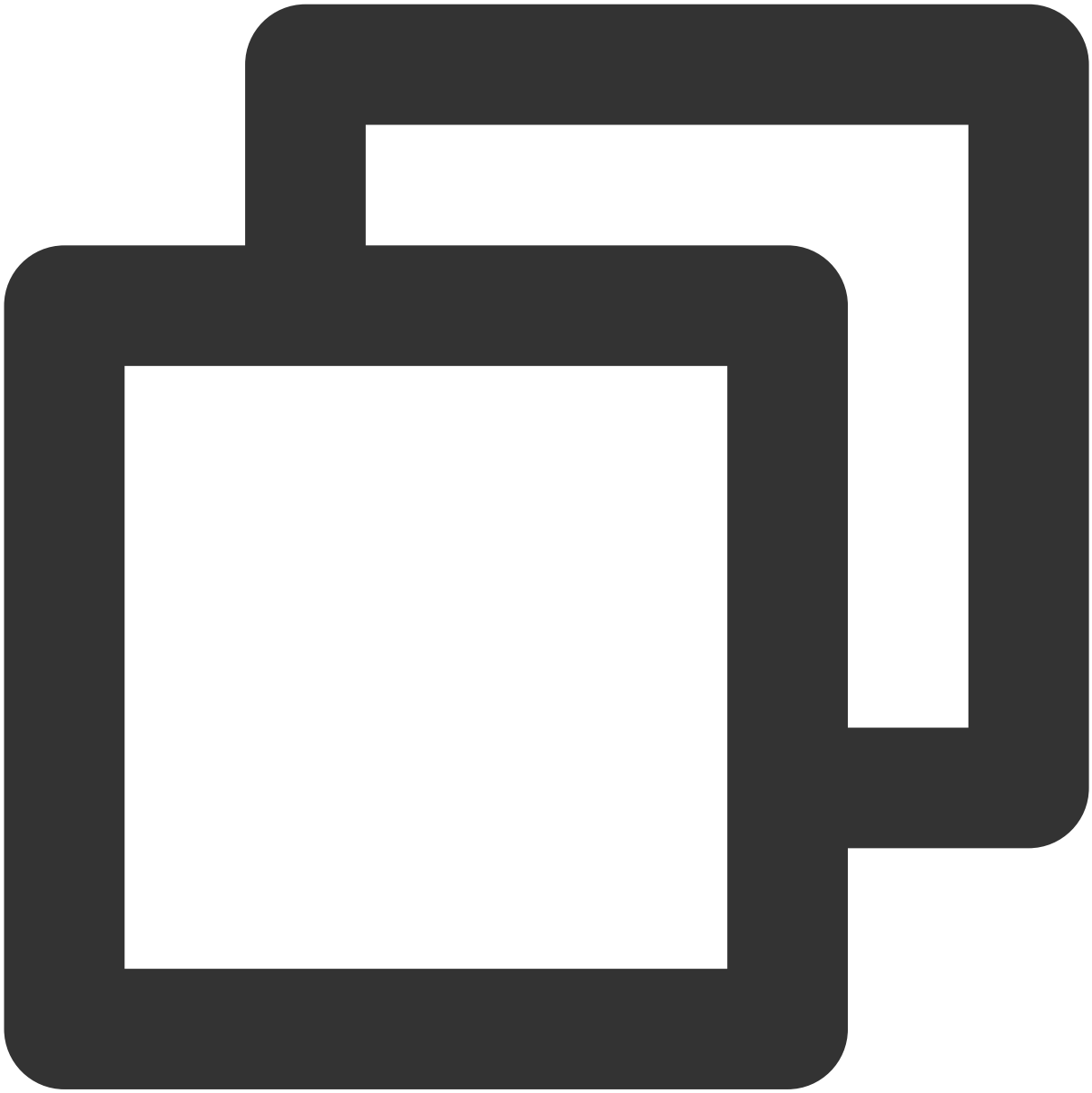
```
REGEXP_EXTRACT_ALL(<str> string, <regexp> string[, <idx> integer])
```

支持引擎：SparkSQL、Presto

使用说明：提取str中与regexp表达式匹配并对应于regex组索引的所有字符串。

返回类型：array <string>

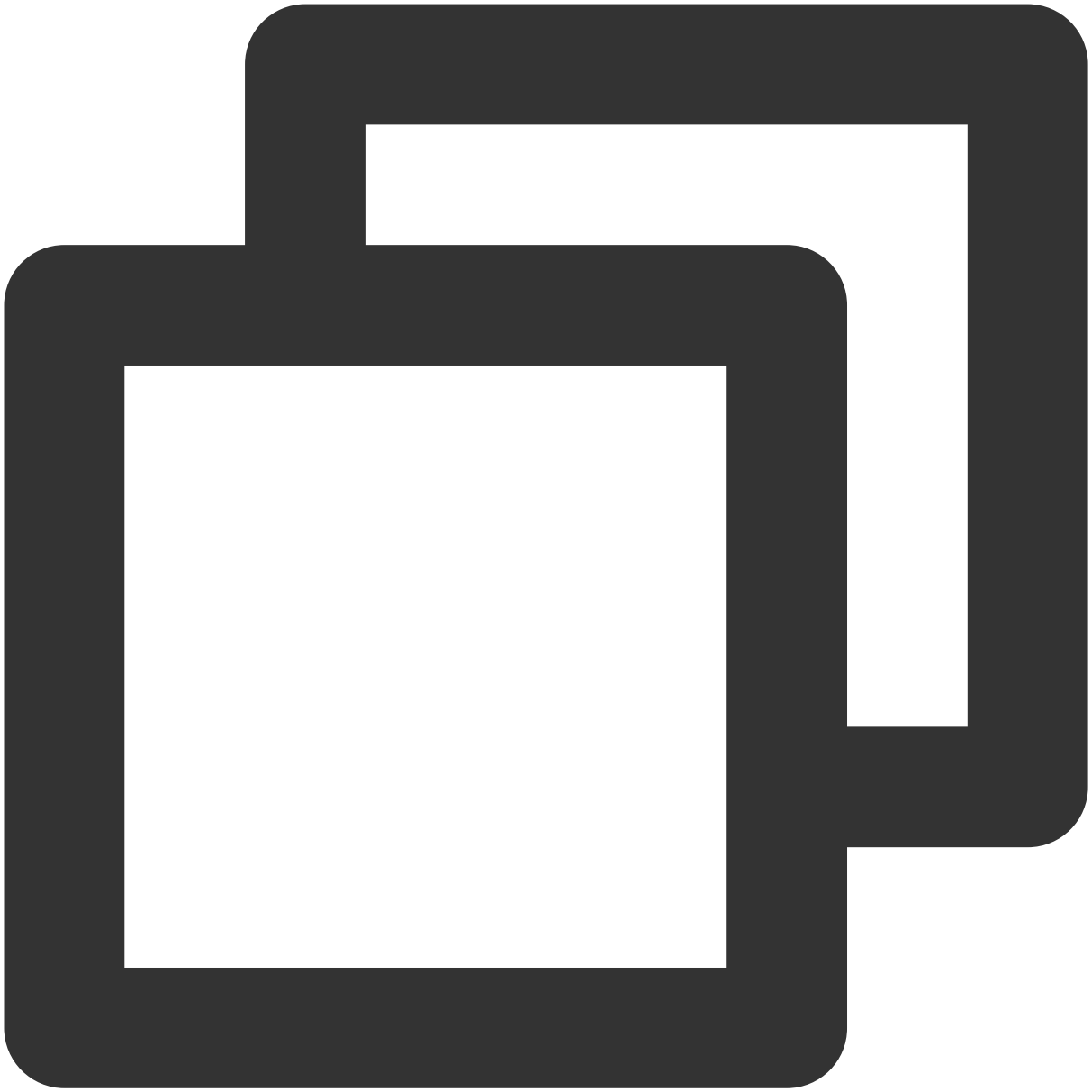
示例：



```
> SELECT regexp_extract_all('100-200, 300-400', '(\\\\d+)-(\\\\d+)', 1);
 ["100","300"]
```

REGEXP_REPLACE

函数语法：



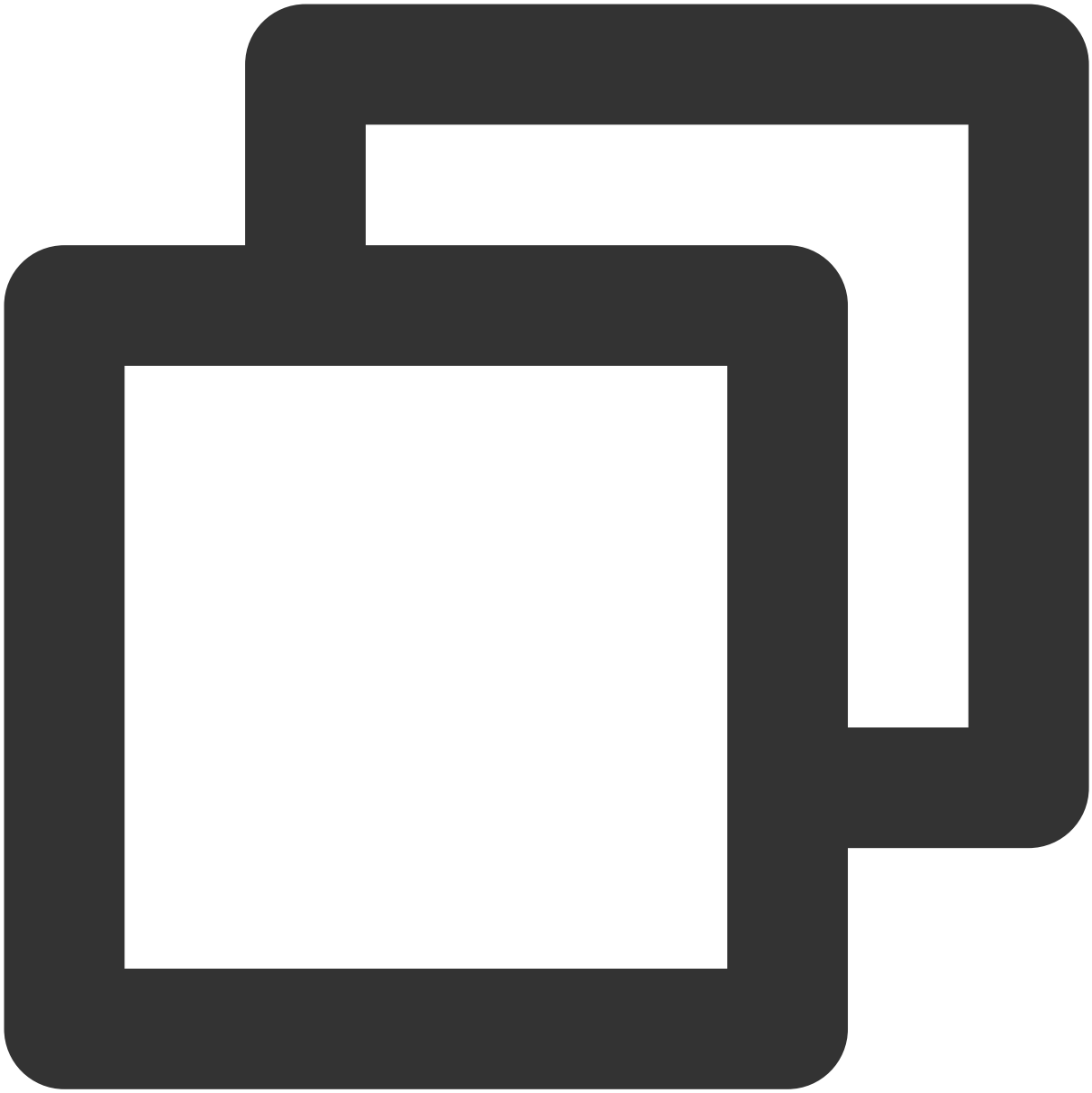
```
REGEXP_REPLACE(<str> string, <regexp> string, <rep> string[, <position> integer])
```

支持引擎：SparkSQL、Presto

使用说明：用rep替换str中与regexp匹配的所有子字符串。

返回类型：string

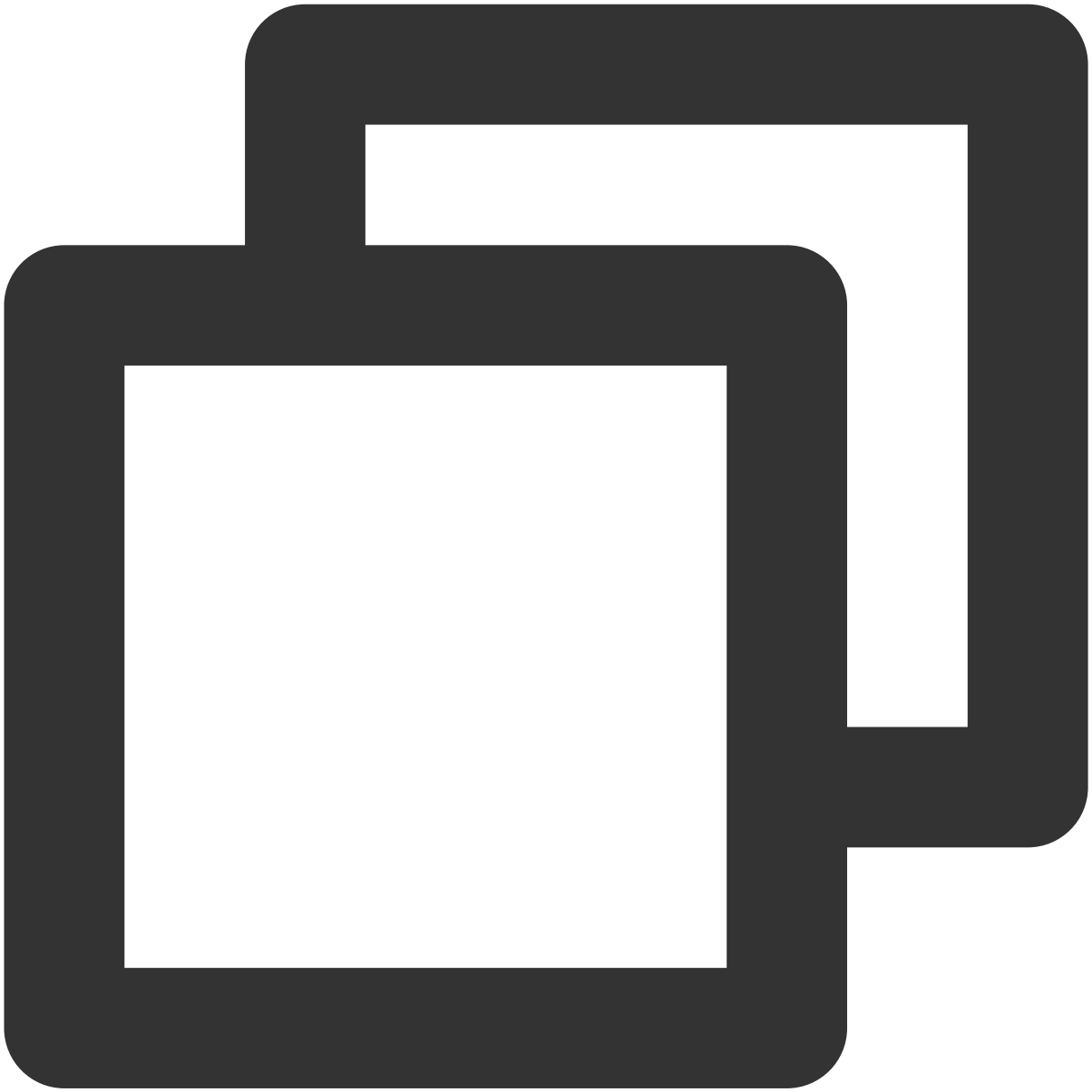
示例：



```
> SELECT regexp_replace('100-200', '(\\\\d+)', 'num');
 num-num
```

REGEXP_LIKE

函数语法：



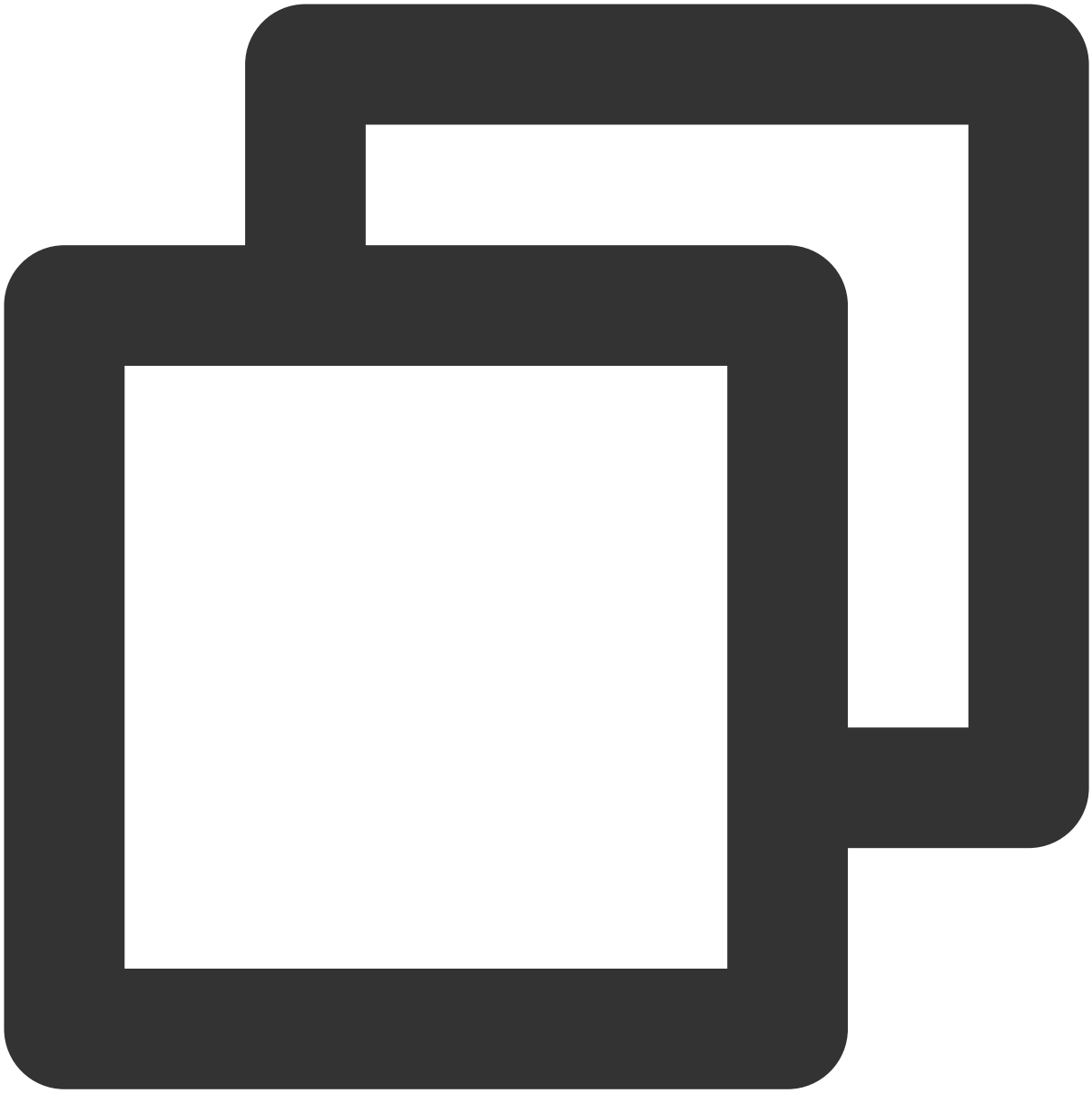
```
REGEXP_LIKE(<str> string, <regexp> string)
```

支持引擎：SparkSQL

使用说明：如果字符串匹配正则表达式，则返回 true，否则返回 false

返回类型：boolean

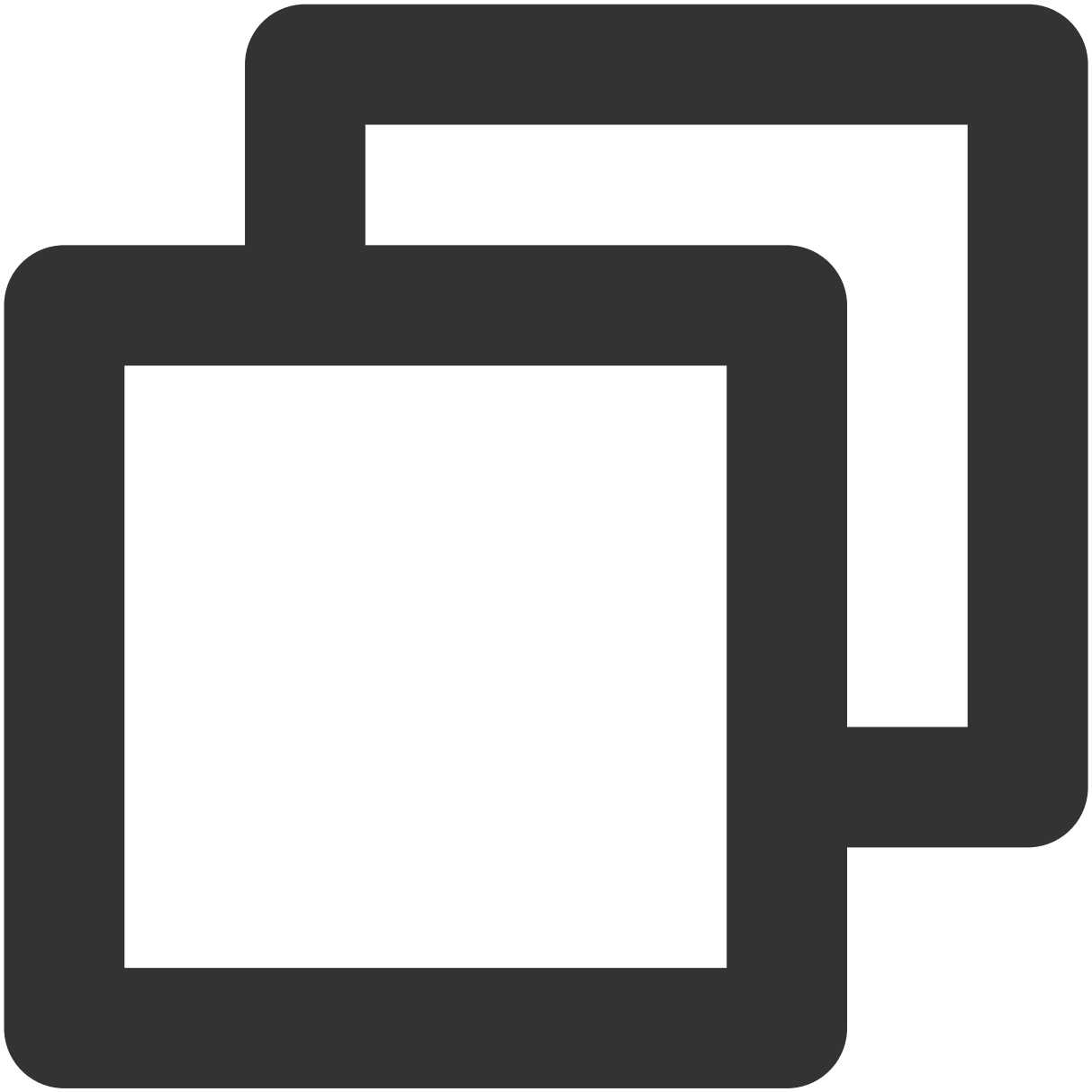
示例：



```
> SELECT regexp_like('%SystemDrive%\\Users\\John', '%SystemDrive%\\Users.*');
 true
```

REGEXP

函数语法：



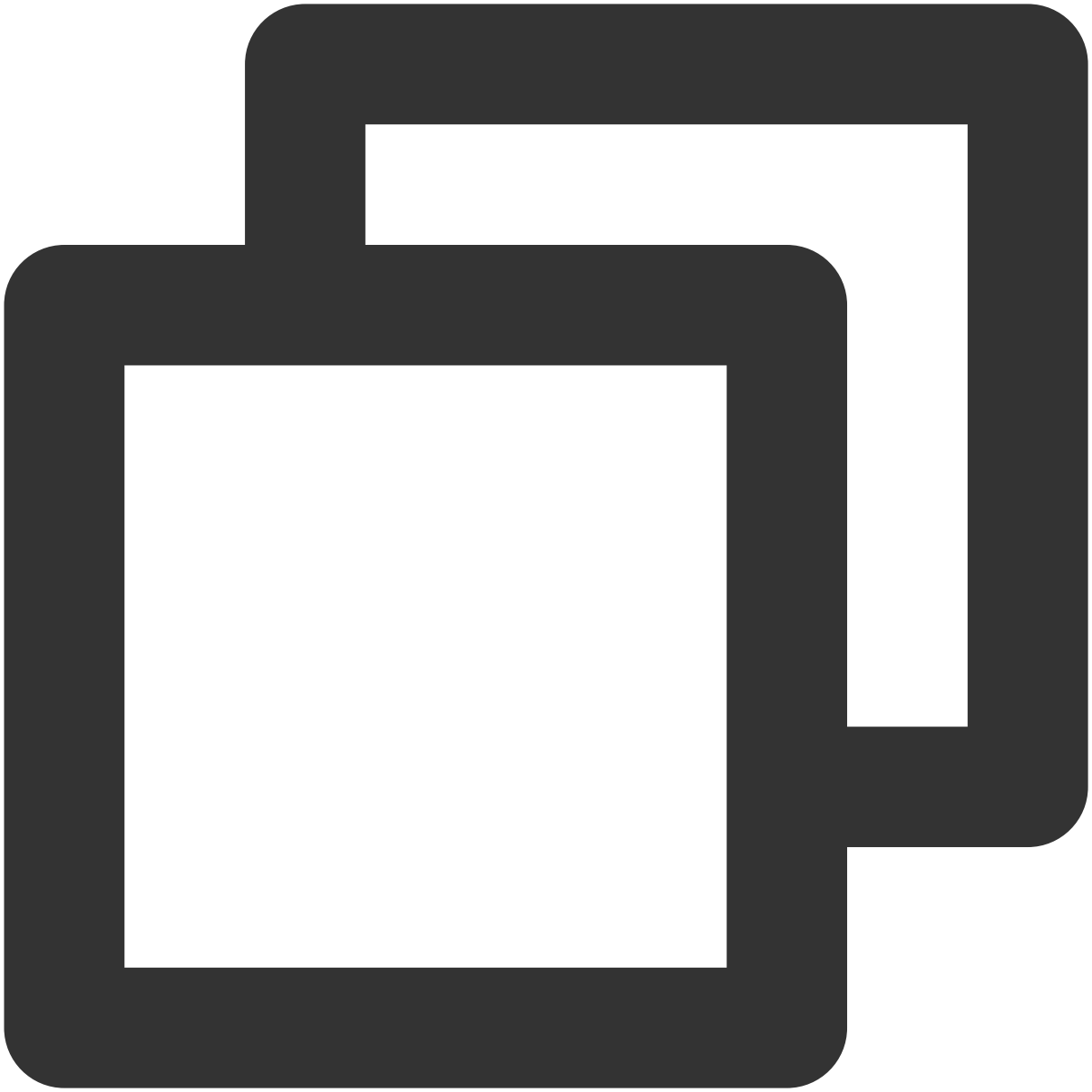
```
REGEXP(<str> string, <regexp> string)
```

支持引擎：SparkSQL

使用说明：如果字符串匹配正则表达式，则返回 true，否则返回 false

返回类型：boolean

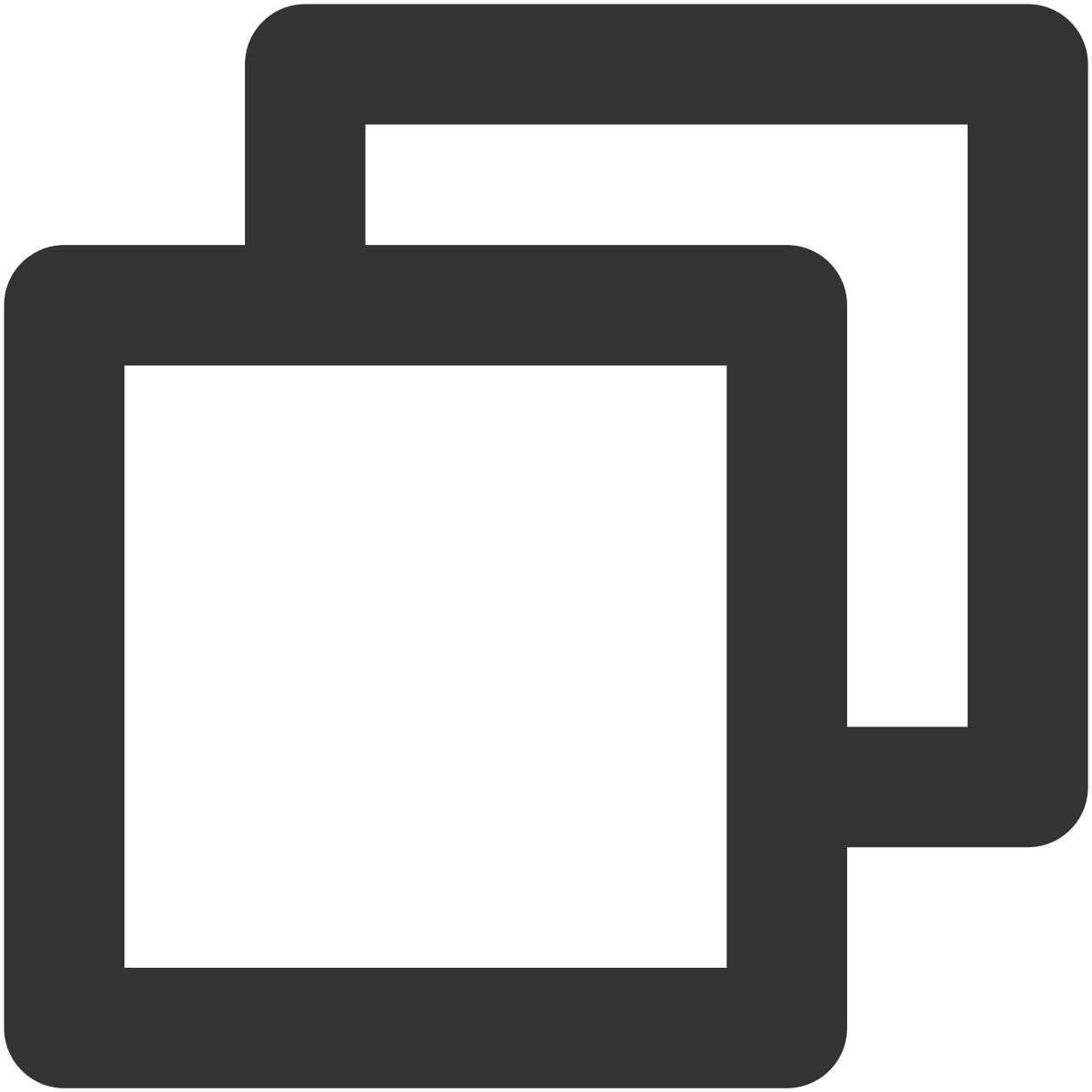
示例：



```
> SELECT regexp('%SystemDrive%\\Users\\John', '%SystemDrive%\\Users.*');
true
```

CONCAT

函数语法：



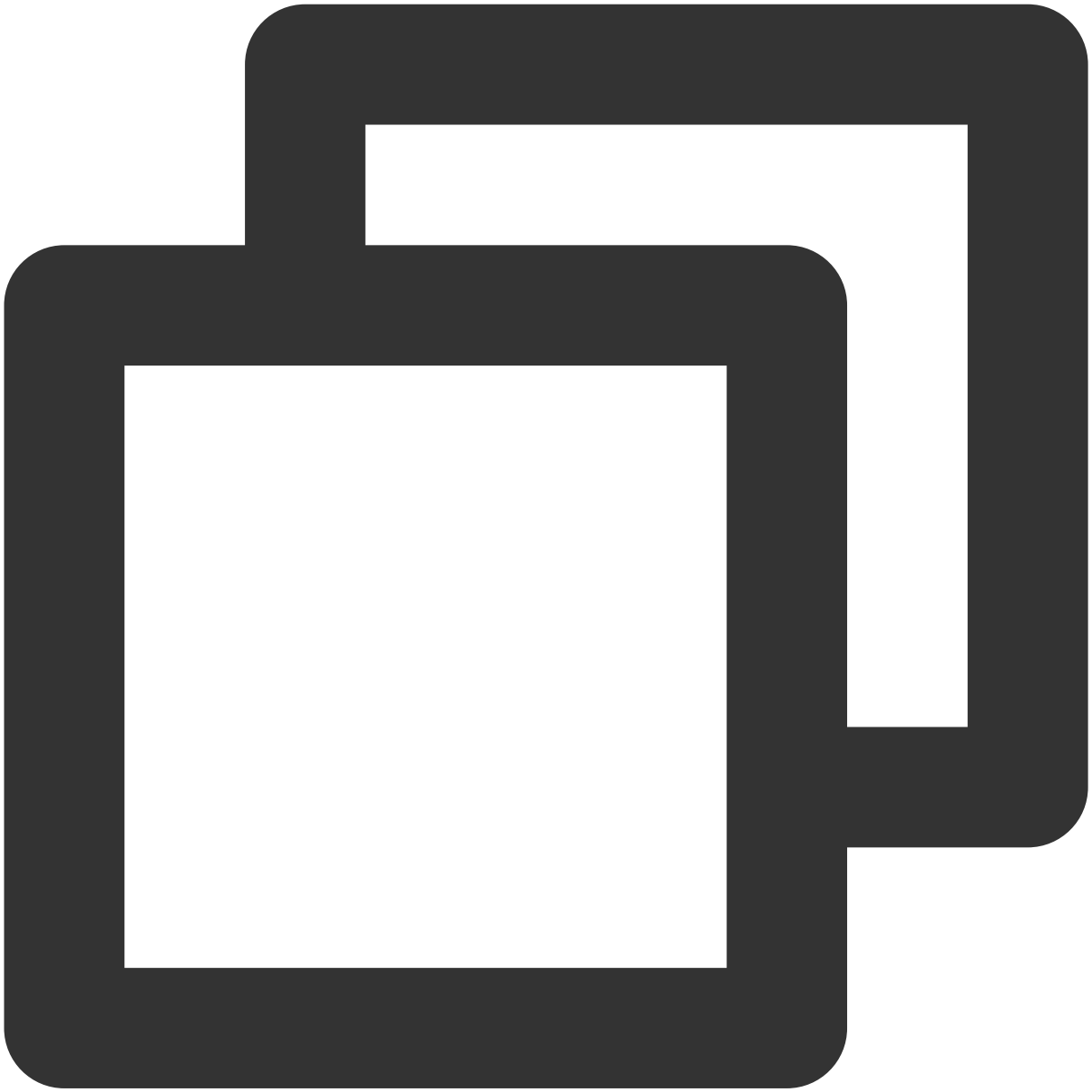
```
CONCAT(<s1> string, <s2> string, ...)
```

支持引擎：SparkSQL、Presto

使用说明：连接s1, s2, ...

返回类型：string

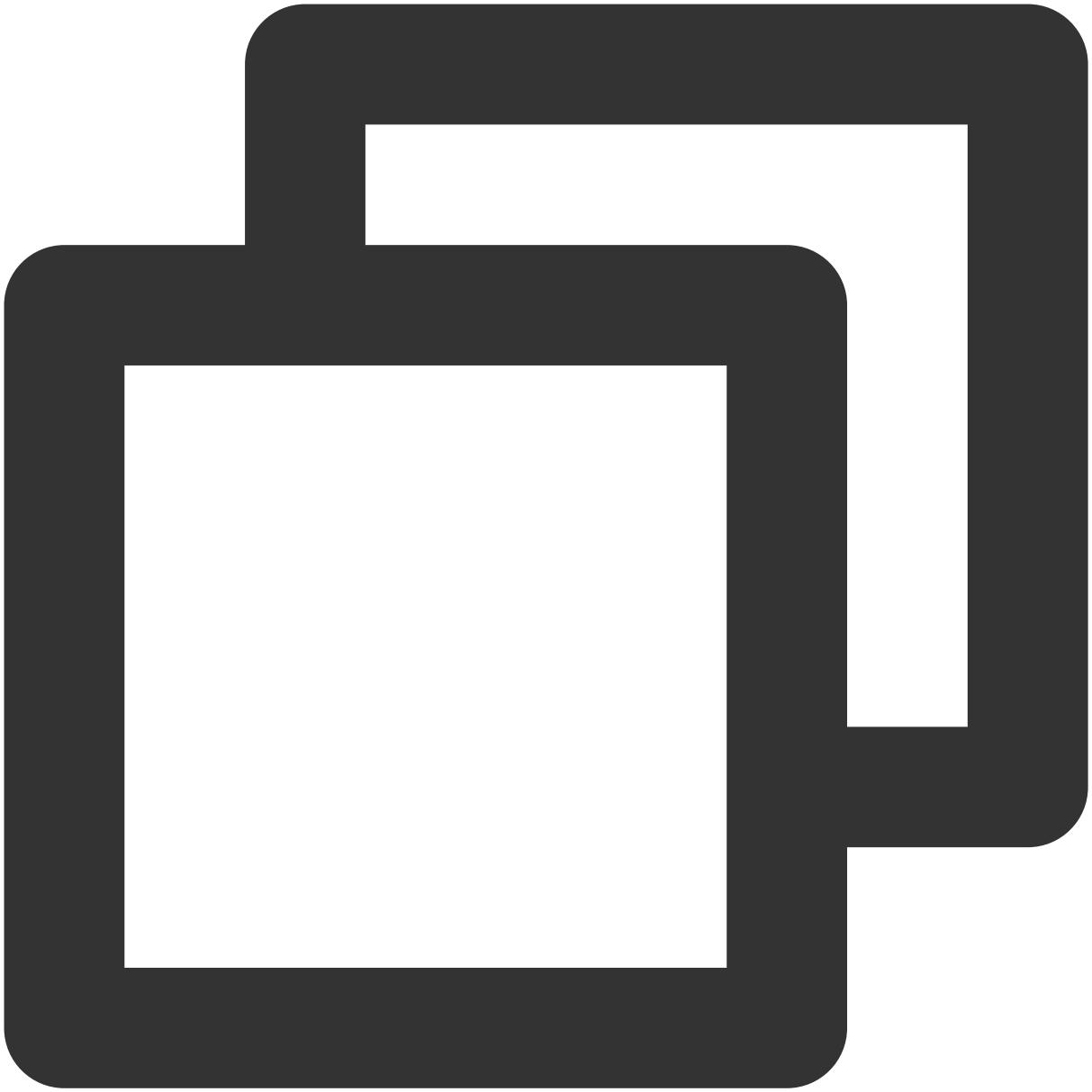
示例：



```
> SELECT concat('Spark', 'SQL');
 SparkSQL
```

STR_TO_MAP

函数语法：



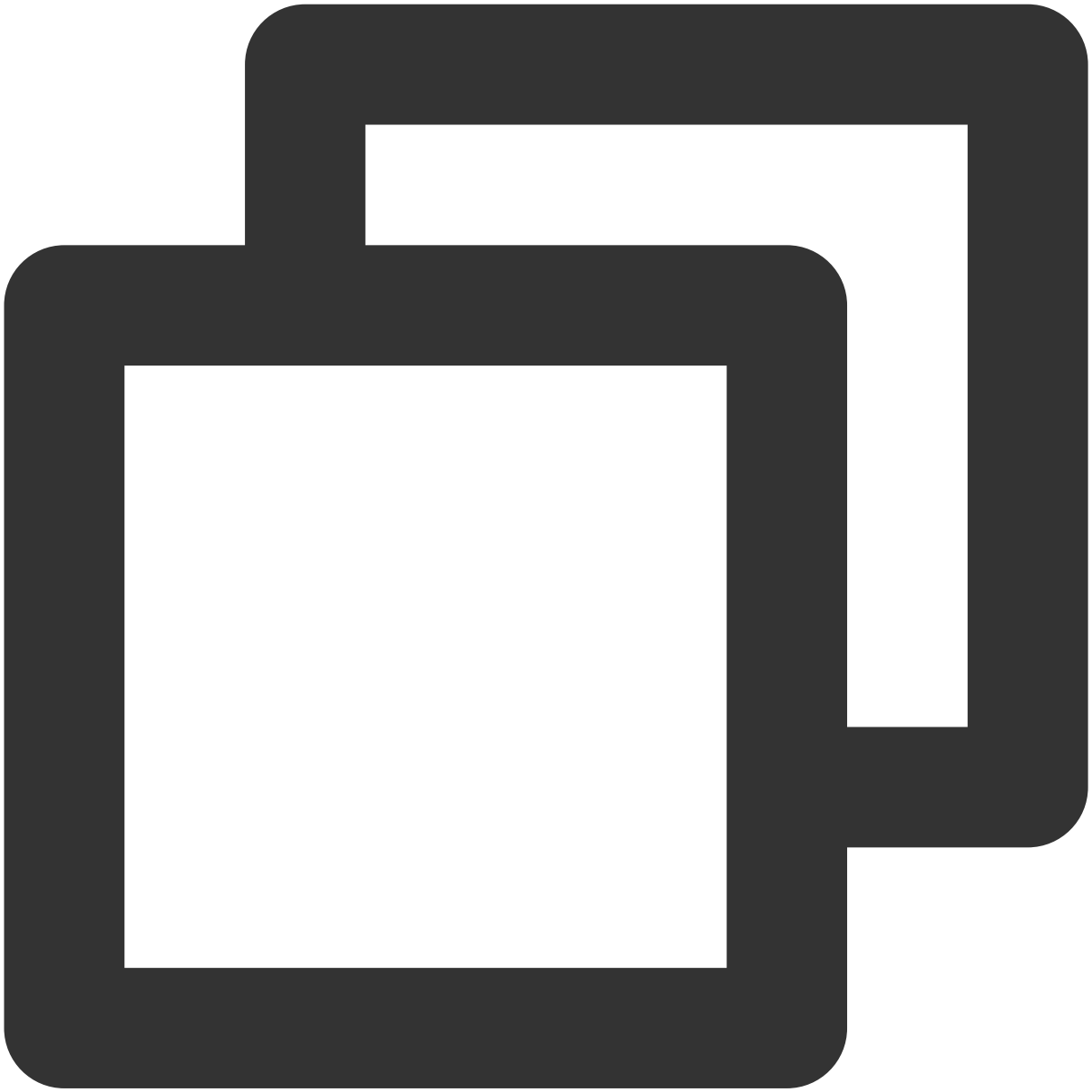
```
str_to_map(<text> string[, <pairDelim> string[, <keyValueDelim> string]])
```

支持引擎：SparkSQL、Presto

使用说明：使用分隔符将text拆分为键/值对后创建map。 对于pairDelim，默认分隔符是','，对于 keyValueDelim，默认分隔符是':'。pairDelim和keyValueDelim都被视为正则表达式。

返回类型：map <string, string>

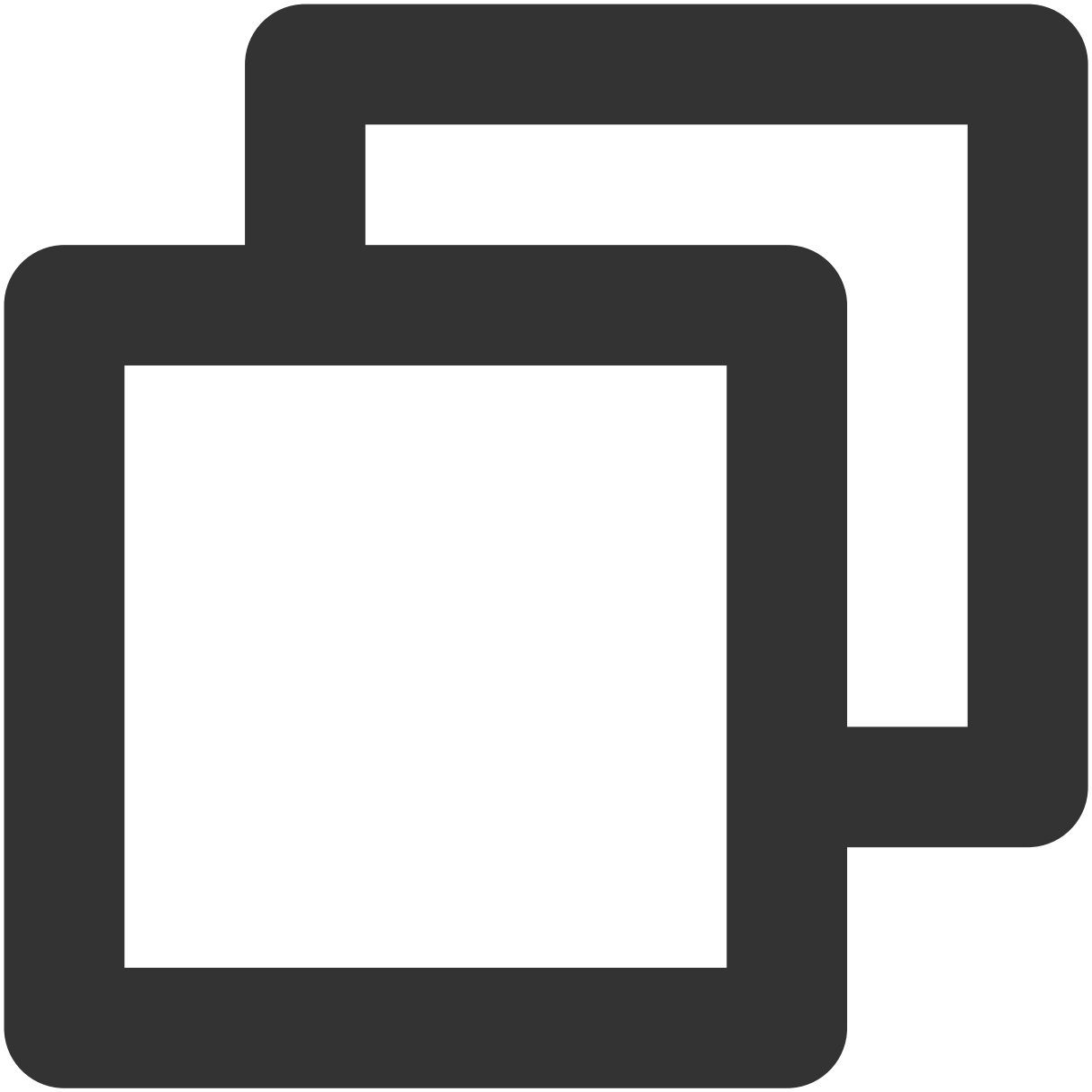
示例：



```
> SELECT str_to_map('a:1,b:2,c:3', ',', ':');
 {"a":"1","b":"2","c":"3"}
> SELECT str_to_map('a');
 {"a":null}
```

REVERSE

函数语法：



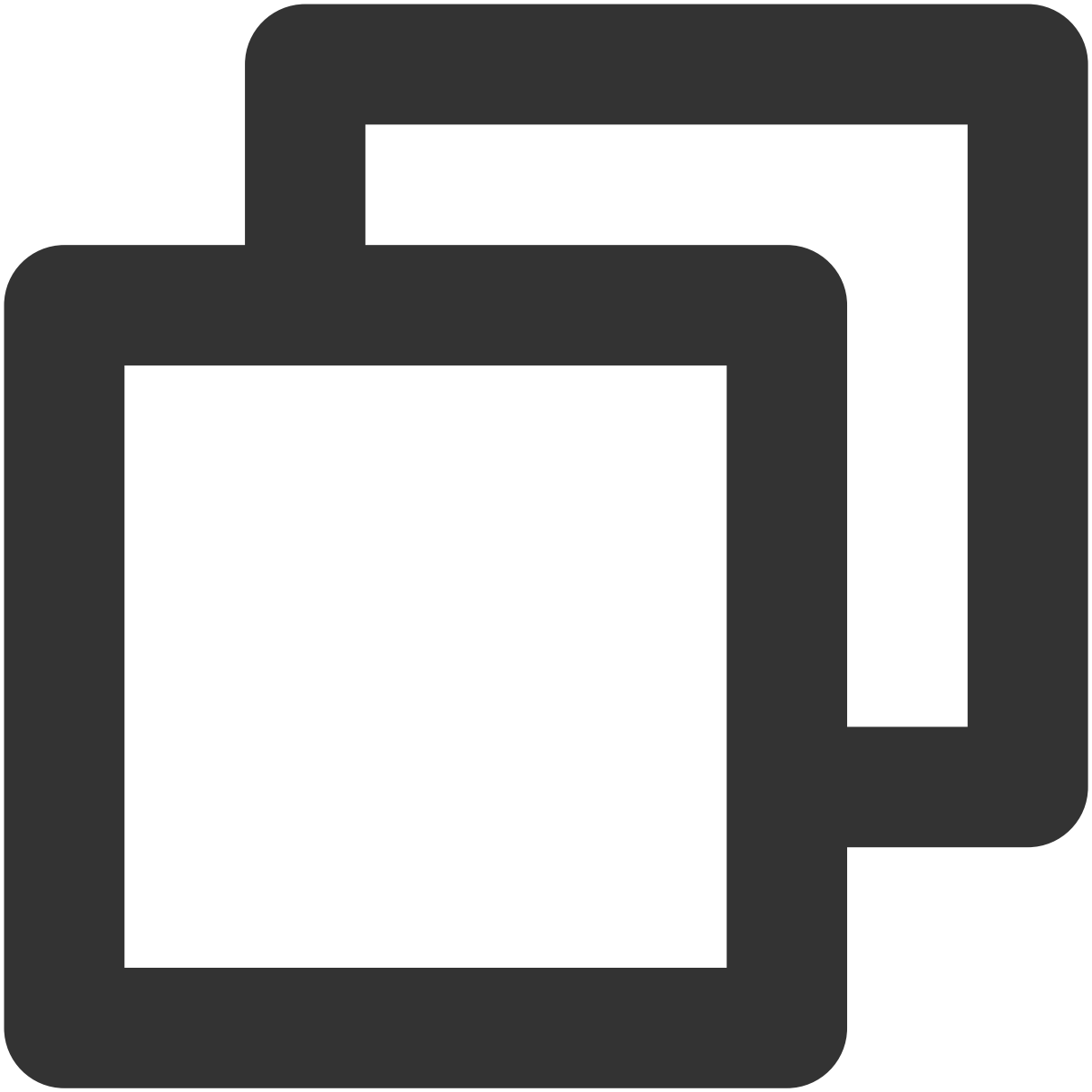
```
REVERSE(<str> string)
```

支持引擎：SparkSQL、Presto

使用说明：返回一个反转的字符串。

返回类型：string

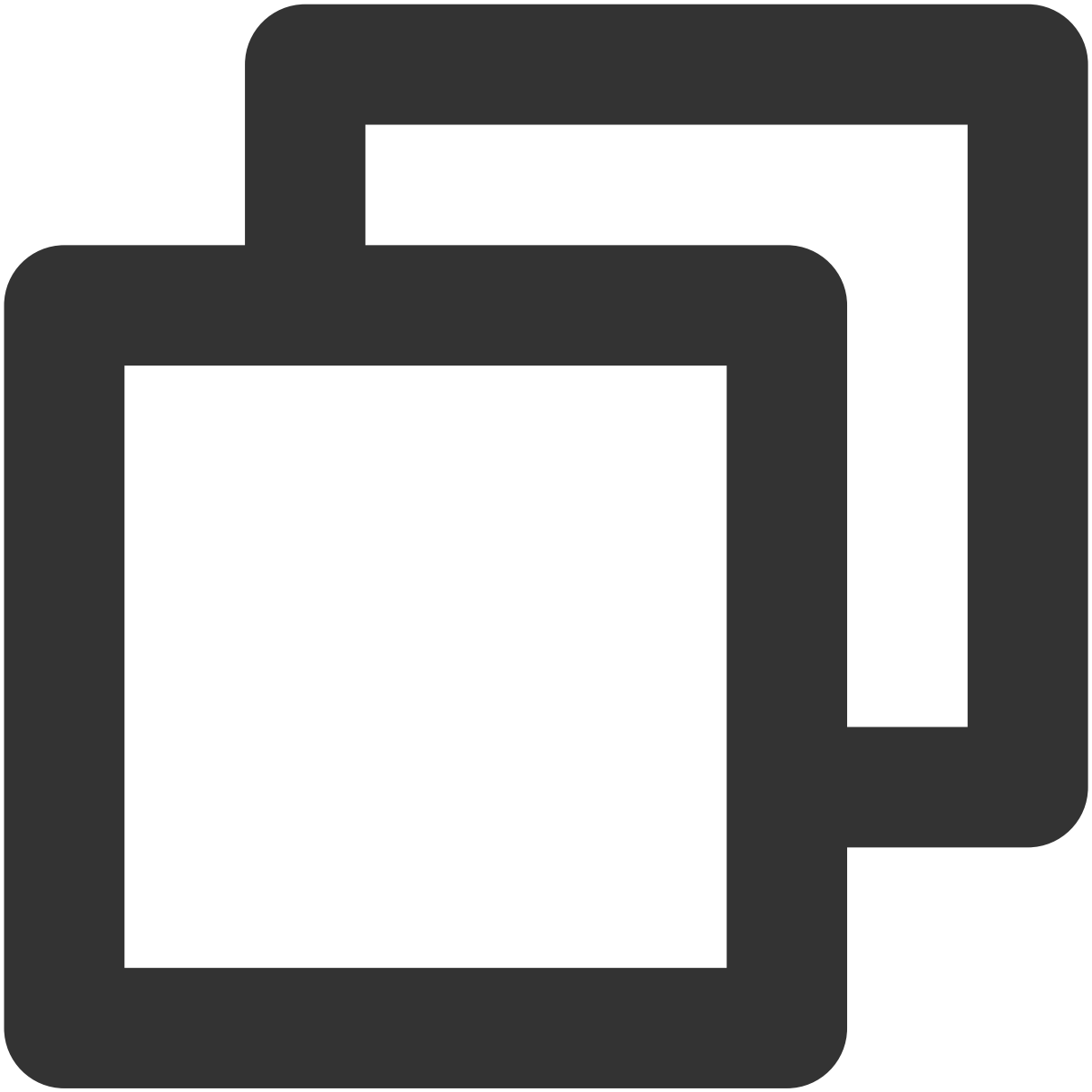
示例：



```
> SELECT reverse('Spark SQL');
 LQS krapS
```

RLIKE

函数语法：



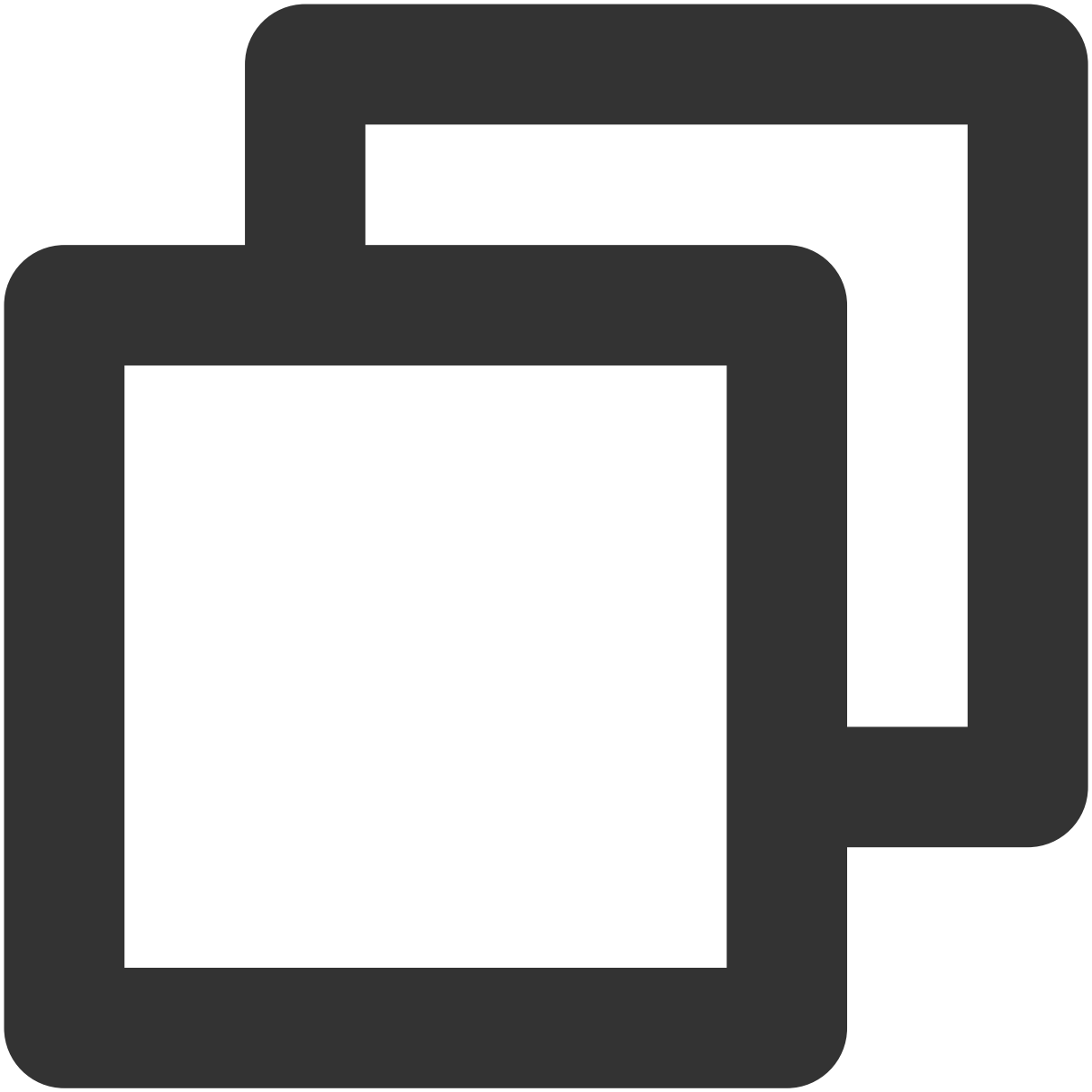
```
RLIKE(<str> string, <regexp> string)
```

支持引擎：SparkSQL、Presto

使用说明：如果str匹配regexp，则返回true，否则返回false。

返回类型：boolean

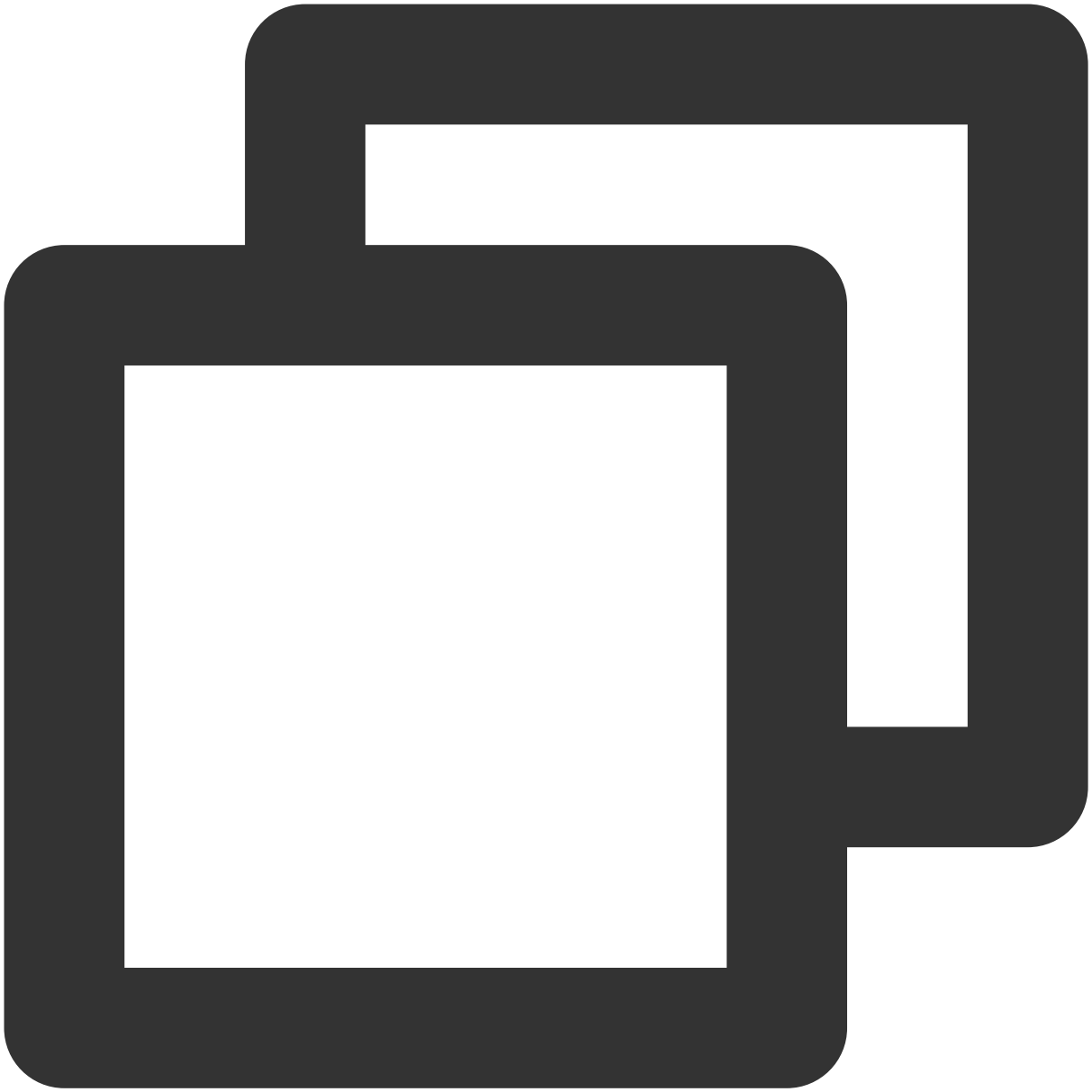
示例：



```
> SELECT rlike('%SystemDrive%\\Users\\John', '%SystemDrive%\\Users.*');
true
```

FROM_CSV

函数语法：



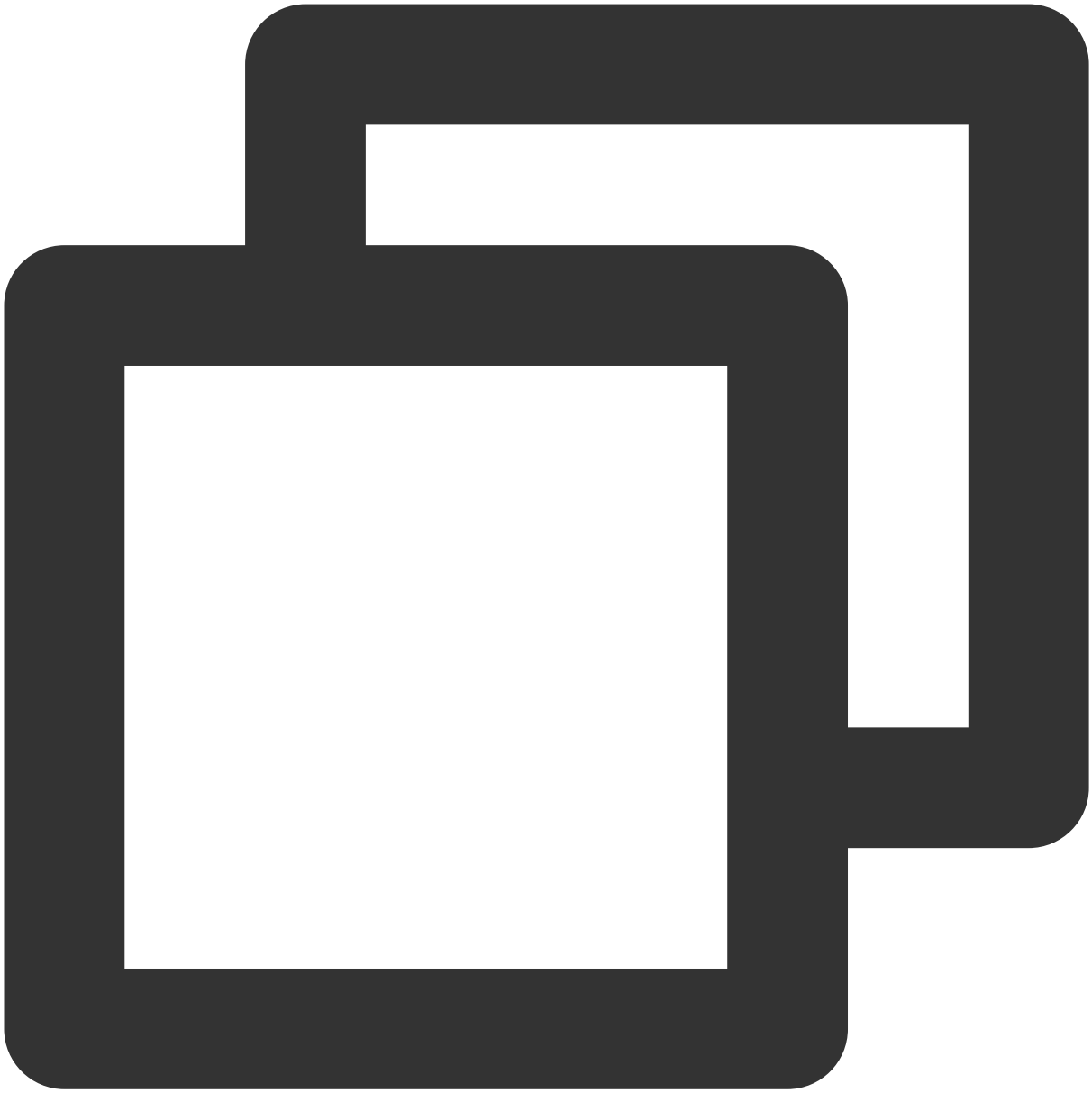
```
FROM_CSV(<cvsStr> string, <schema> string, <options> map<string, string>)
```

支持引擎：SparkSQL

使用说明：返回具有给定csvStr和schema的结构值。

返回类型：struct

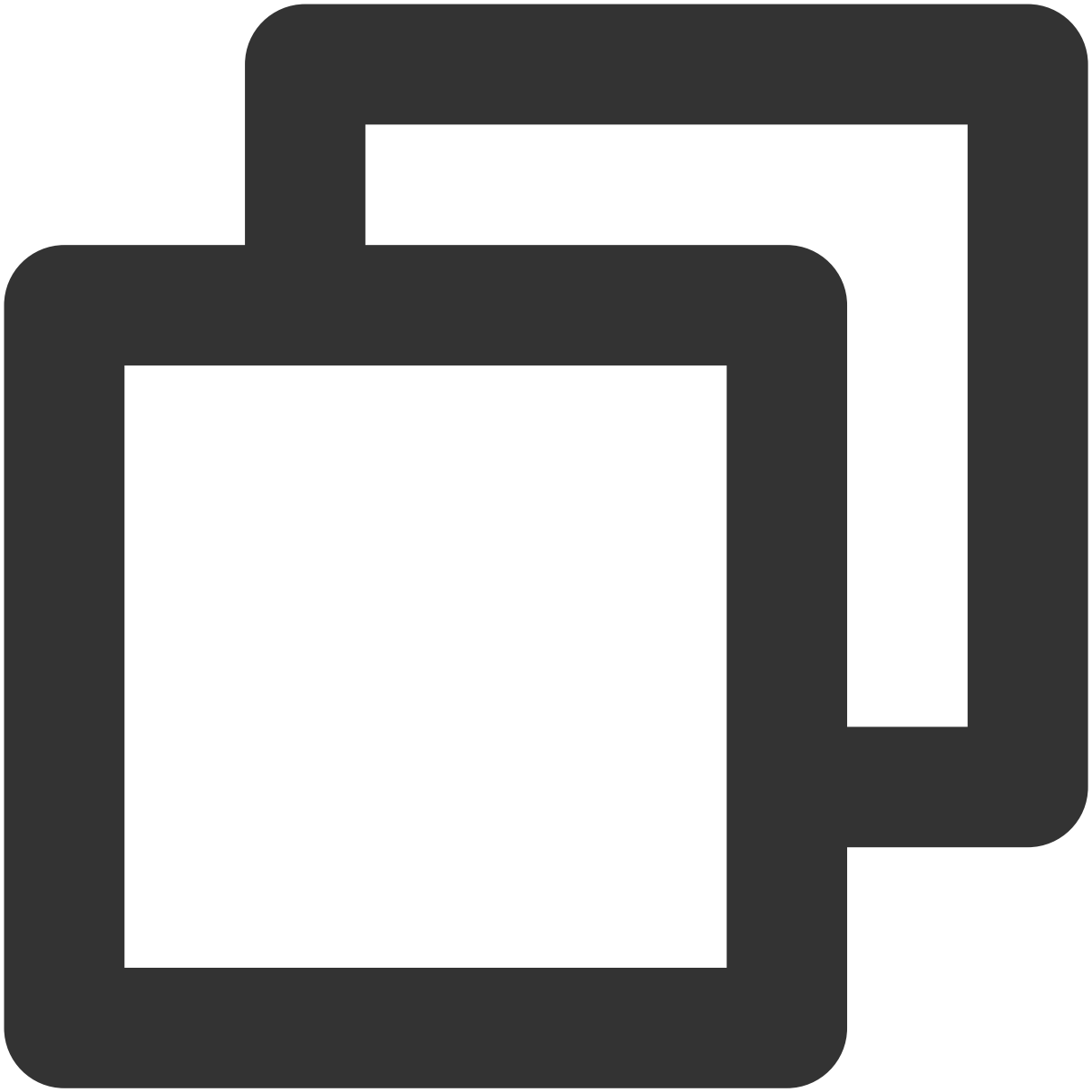
示例：



```
> SELECT from_csv('1, 0.8', 'a INT, b DOUBLE');
 {"a":1,"b":0.8}
> SELECT from_csv('26/08/2015', 'time Timestamp', map('timestampFormat', 'dd/MM/yyy
 {"time":2015-08-26 00:00:00}
```

SCHEMA_OF_CSV

函数语法：



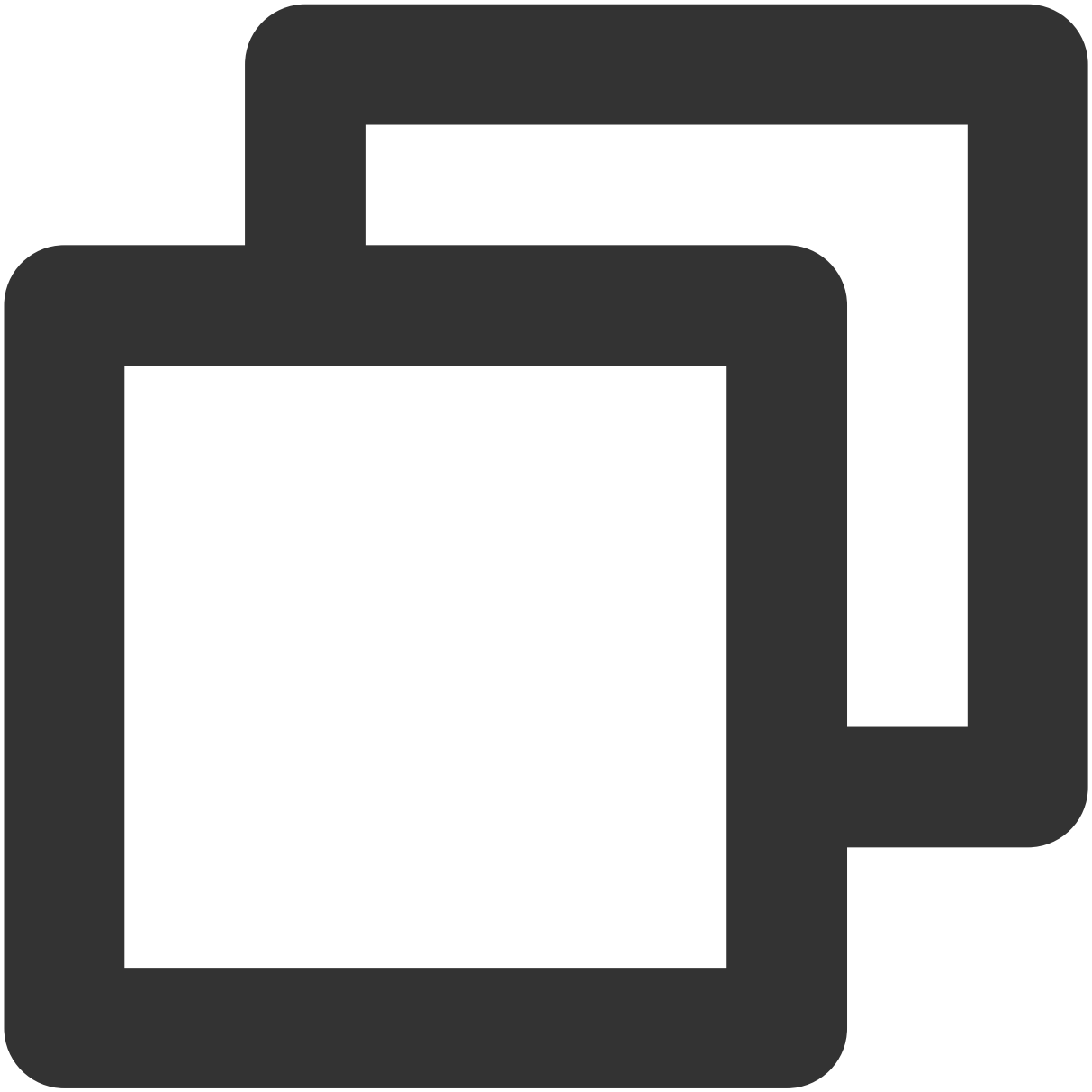
```
SCHEMA_OF_CSV(<csvStr> string[, options map<string, string>])
```

支持引擎：SparkSQL

使用说明：返回csv字符串的schema。

返回类型：string

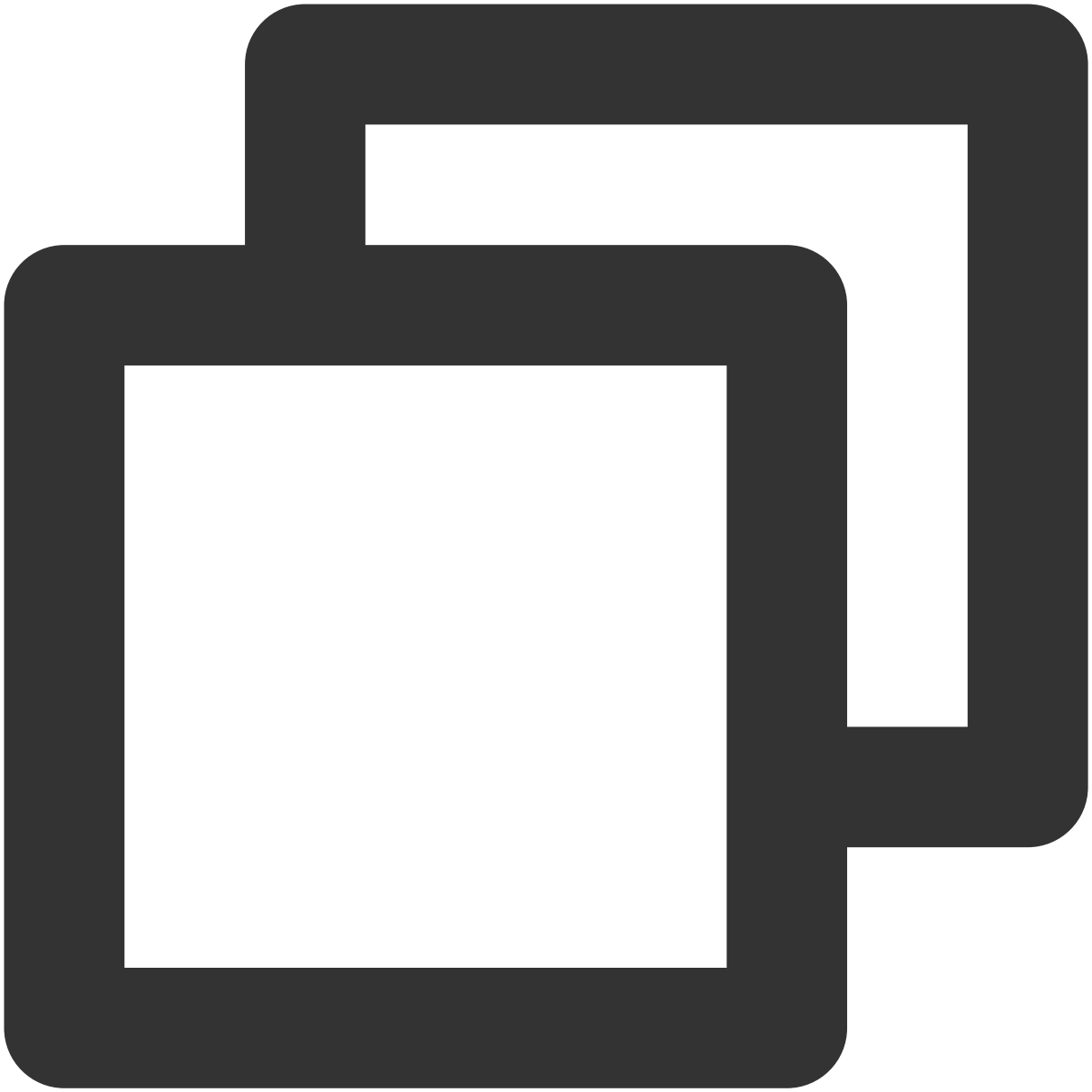
示例：



```
> SELECT schema_of_csv('1,abc');
 STRUCT<`_c0`: INT, `_c1`: STRING>
```

TO_CSV

函数语法：



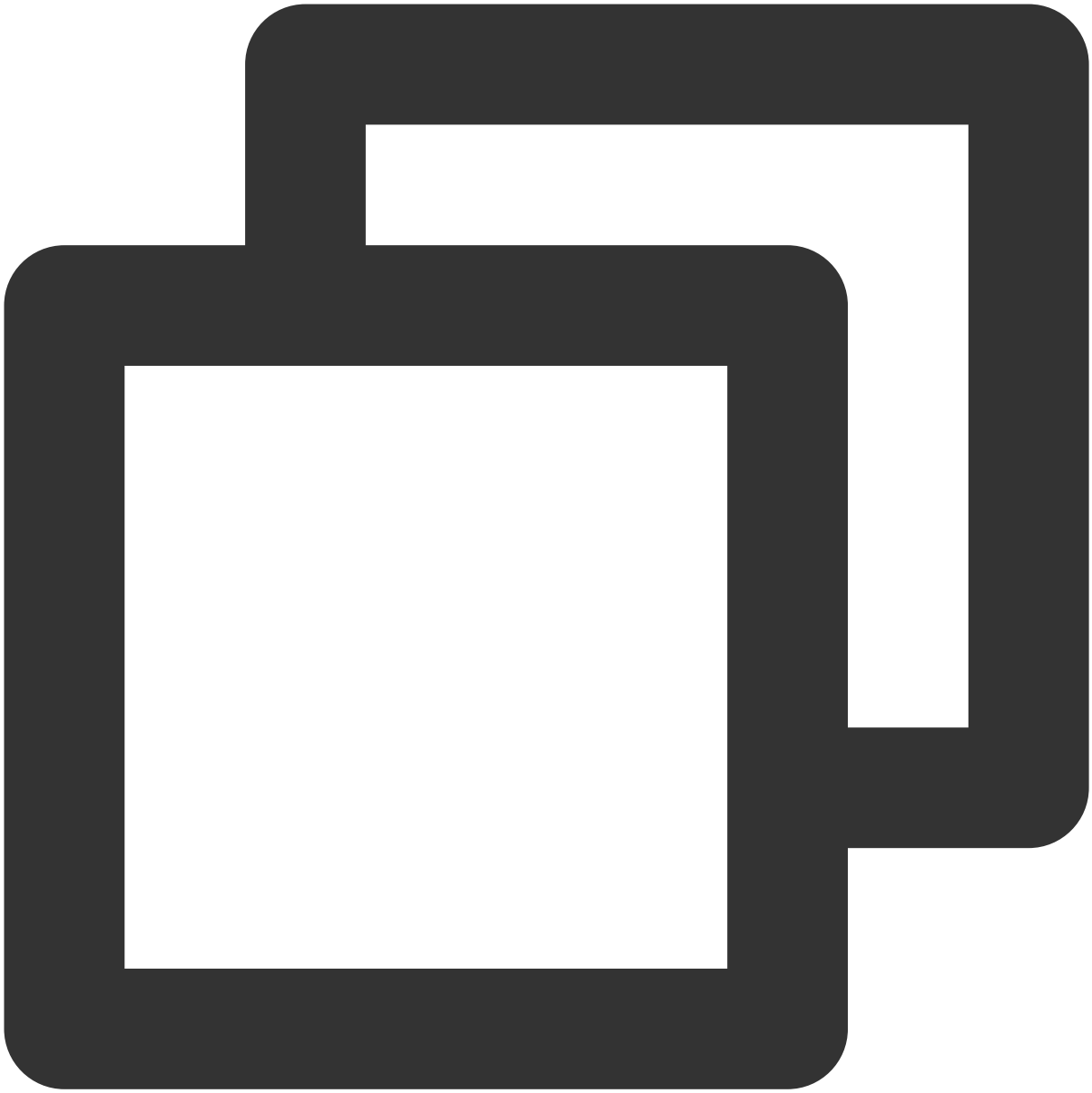
```
TO_CSV(<expr> struct[, <options> map<string, string>])
```

支持引擎：SparkSQL

使用说明：返回具有给定结构值的csv字符串

返回类型：string

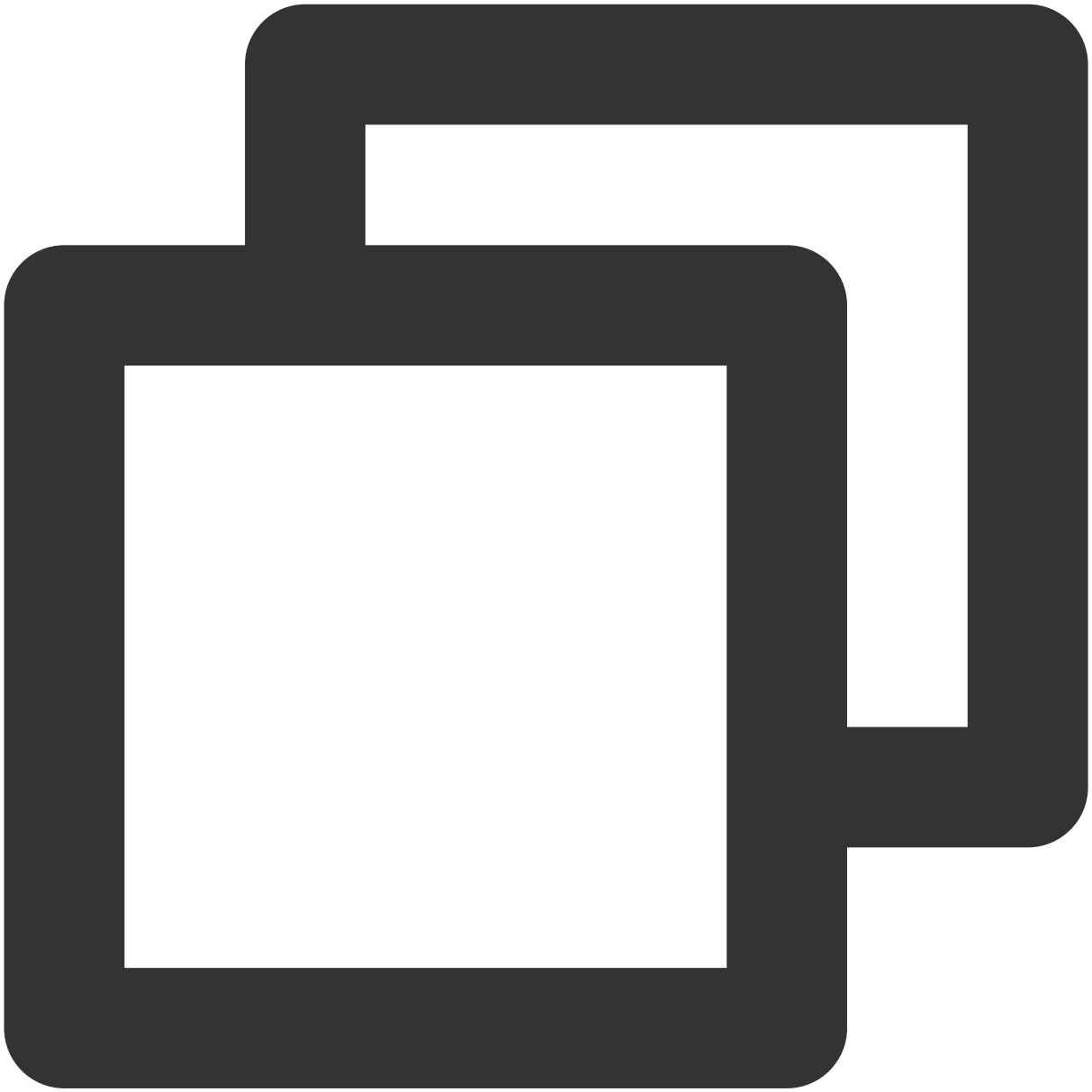
示例：



```
> SELECT to_csv(named_struct('a', 1, 'b', 2));
 1,2
> SELECT to_csv(named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd')), map
 26/08/2015
```

NGRAMS

函数语法：



```
NGRAMS(<a> array<array<string>>, <N> integer, <K> integer, <pf> integer)
```

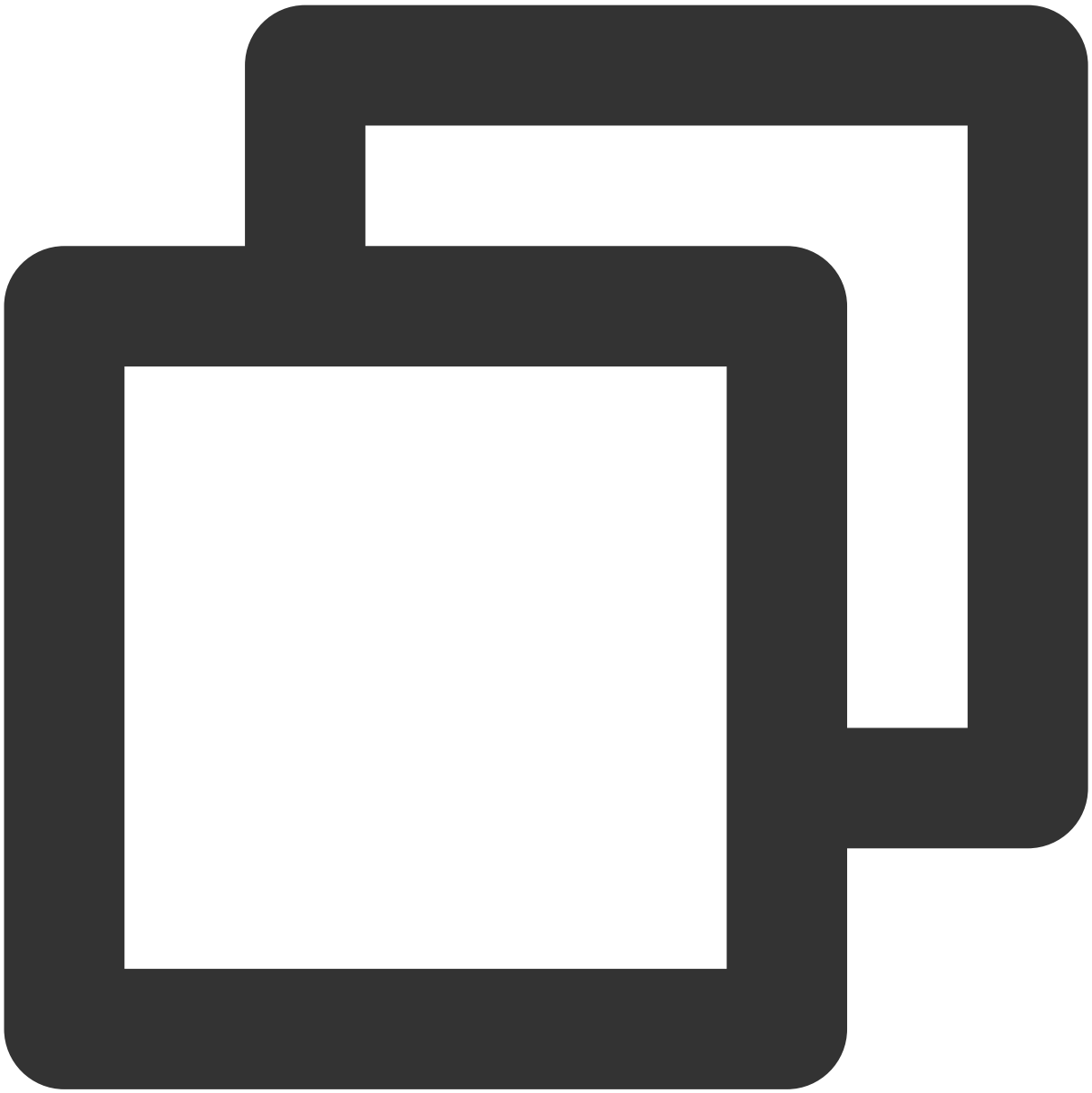
支持引擎：Presto。

使用说明：返回一组标记化句子中的前 k 个 N-grams。

返回类型：array <struct <string,double>>。

CONTEXT_NGRAMS

函数语法：



```
CONTEXT_NGRAMS((array &lt;array &lt;string>>, array &lt;string>, int, int))
```

支持引擎：Presto

使用说明：给定一个contextual N-grams，从一组标记化的句子中返回前 k 个上下文 N-gram。

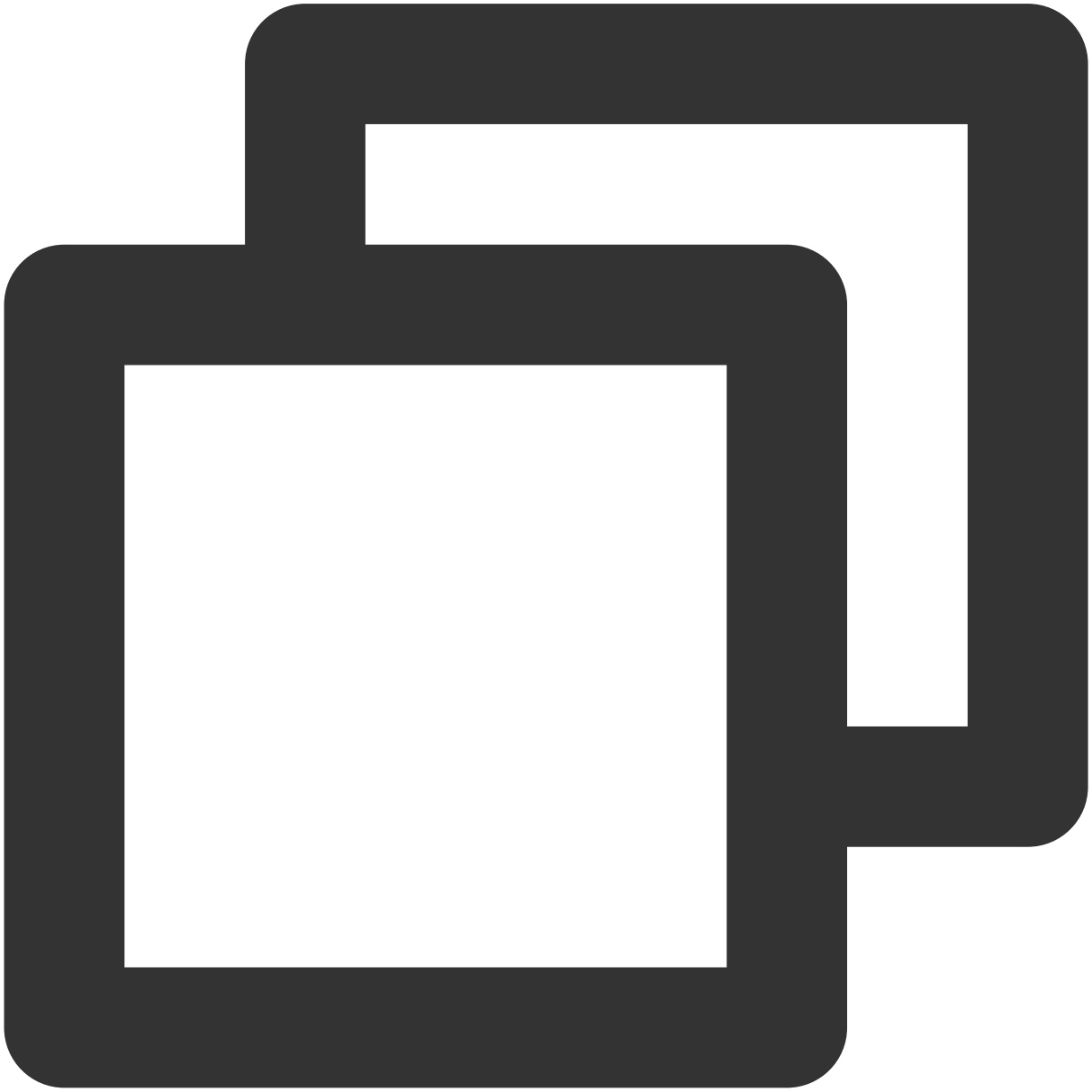
返回类型：array <struct <string, double>>

聚合函数

最近更新时间：2024-08-07 17:33:16

APPROX_COUNT_DISTINCT

函数语法：



```
APPROX_COUNT_DISTINCT(<expr> any [, <relativeSD> integer|double|decimal])
```

支持引擎：SparkSQL

使用说明：返回HyperLogLog++估计的基数。relativeSD定义了允许的最大相对标准差。

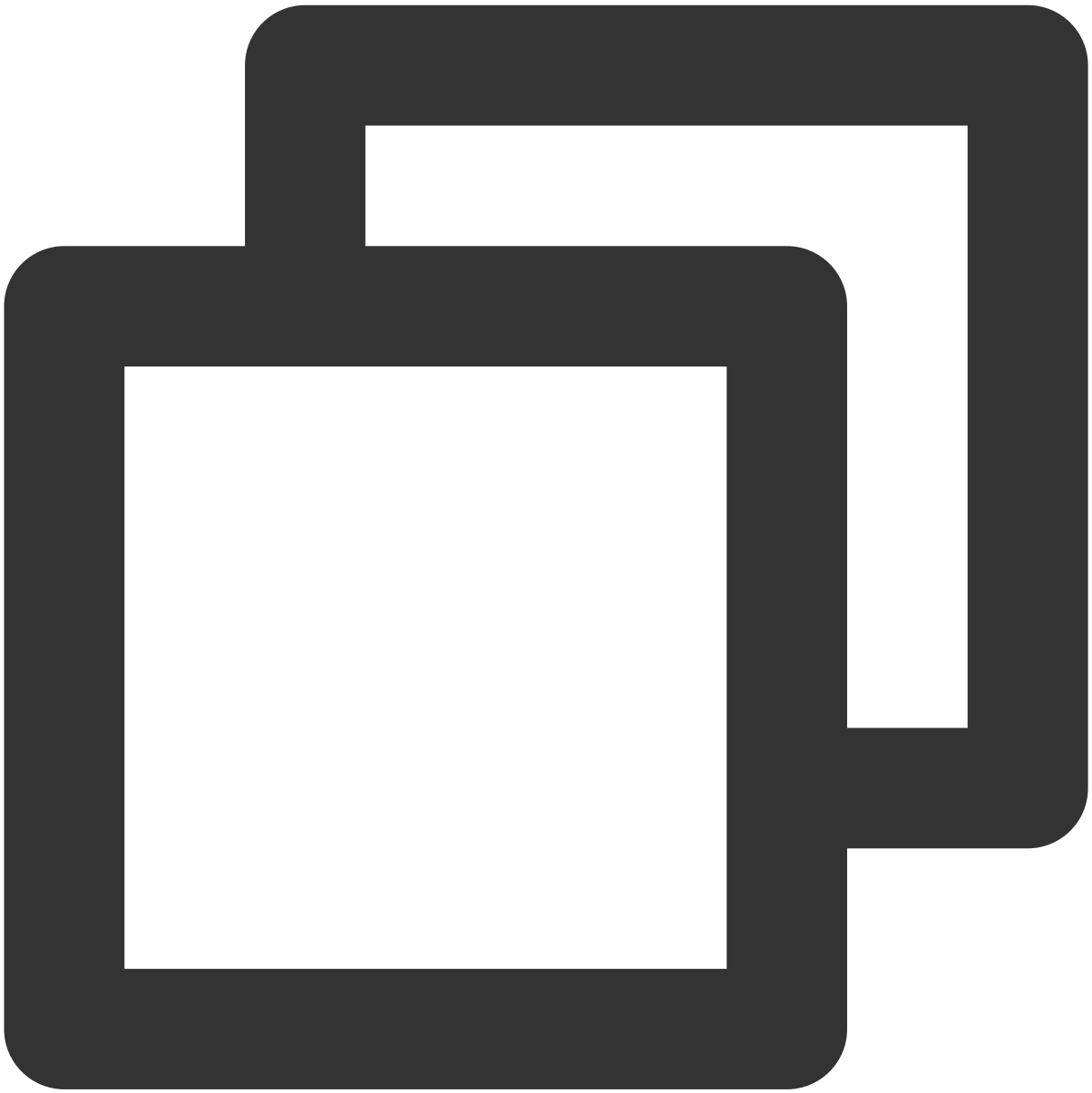
返回类型：bigint

示例：

```
SELECT approx_count_distinct(col1) FROM (VALUES (1), (1), (2), (2), (3)) tab(col1);
 3
```

AVG

函数语法：



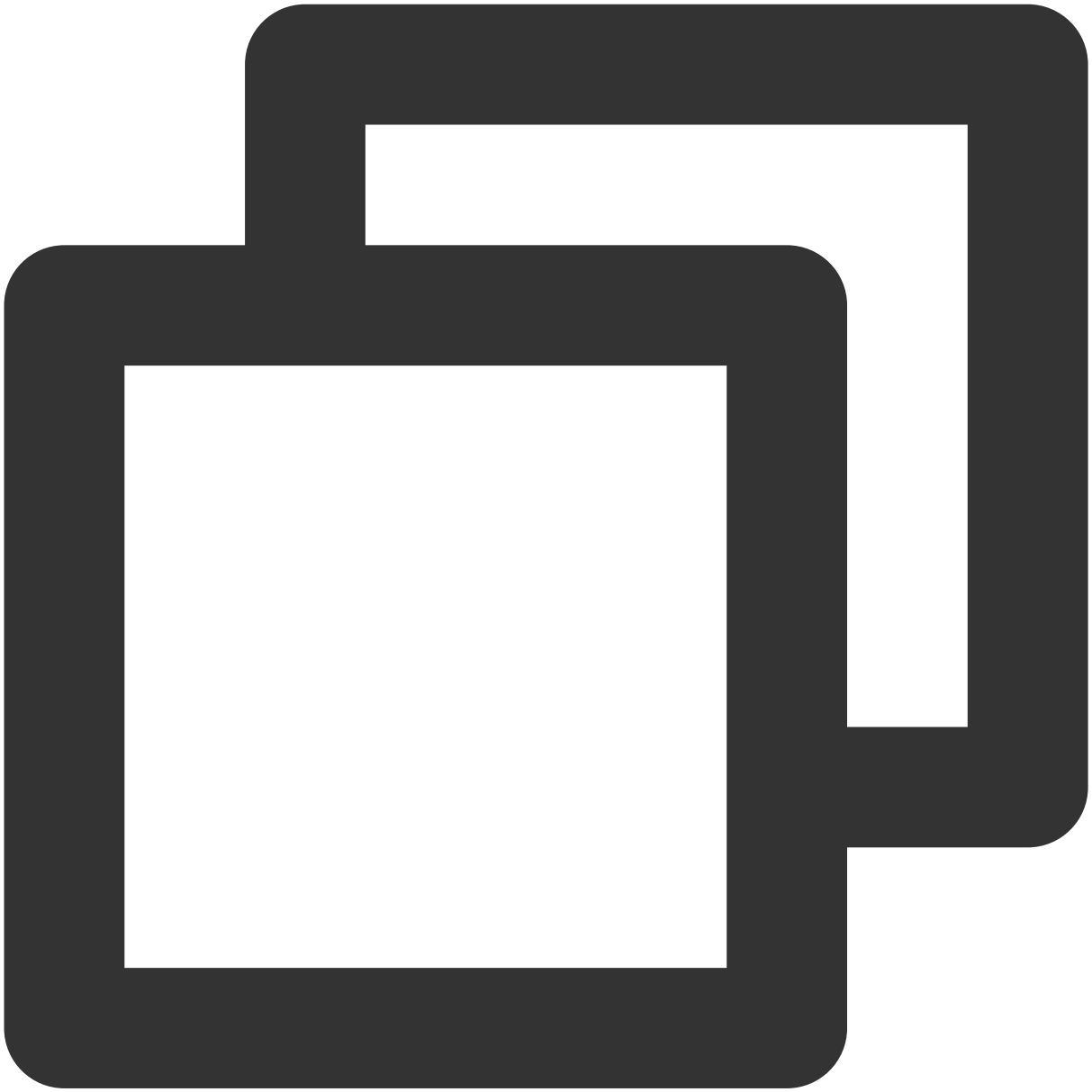
```
AVG(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回根据组的值计算的平均值

返回类型：double

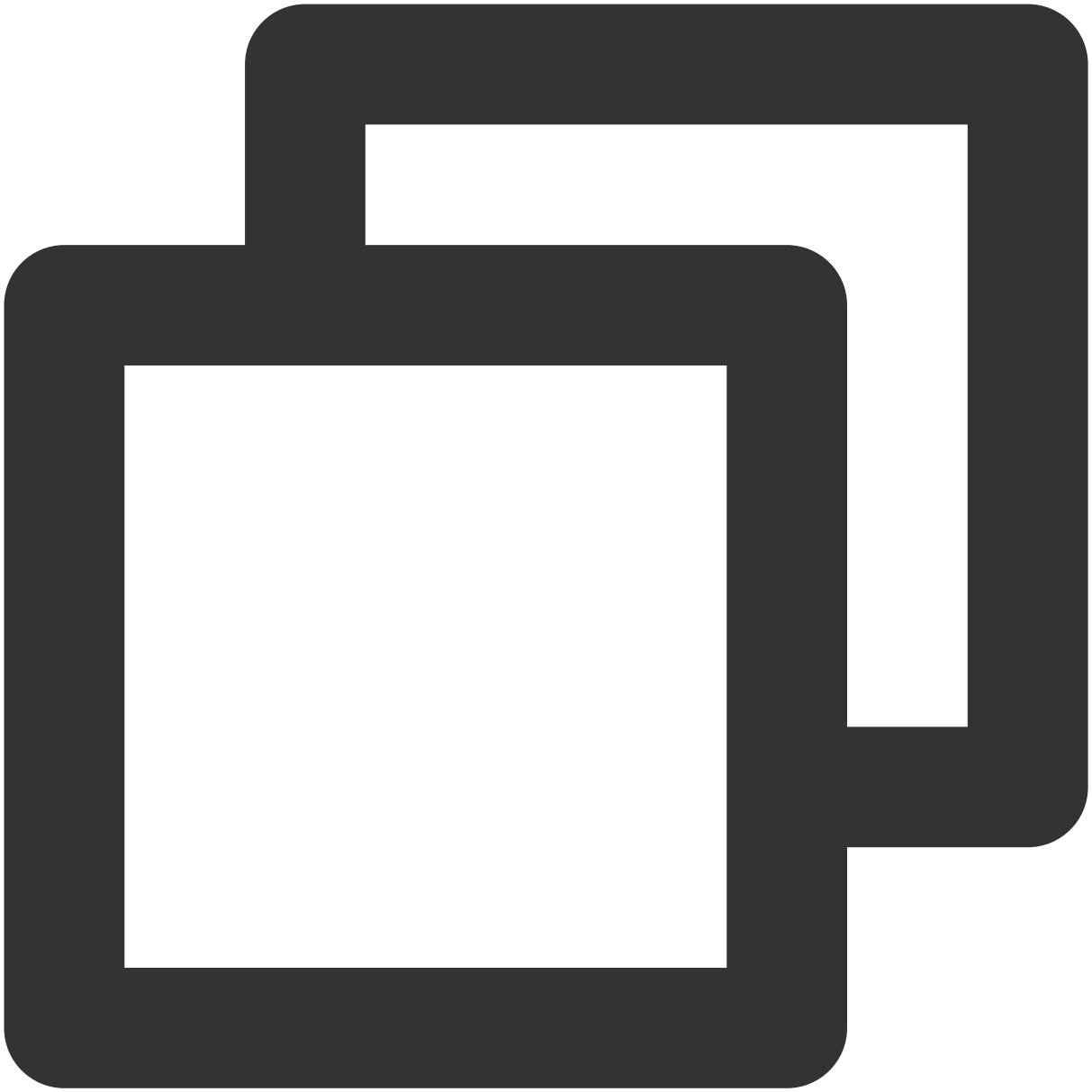
示例：



```
> SELECT avg(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 2.0
> SELECT avg(col) FROM (VALUES (1), (2), (NULL)) AS tab(col);
 1.5
```

CORR

函数语法：



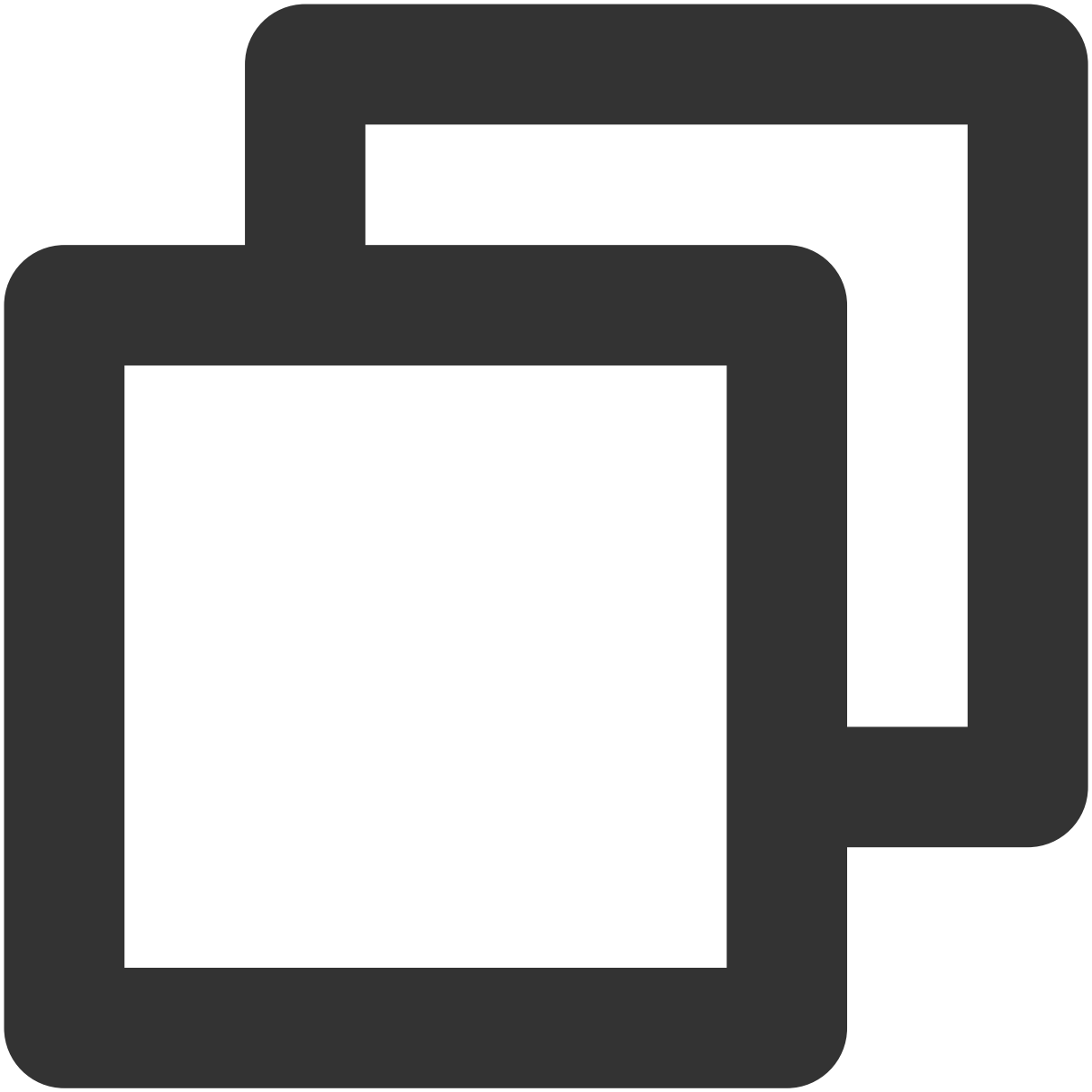
```
CORR(<expr> integer|double|decimal, <expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回一组数字对之间的Pearson相关系数。

返回类型：double

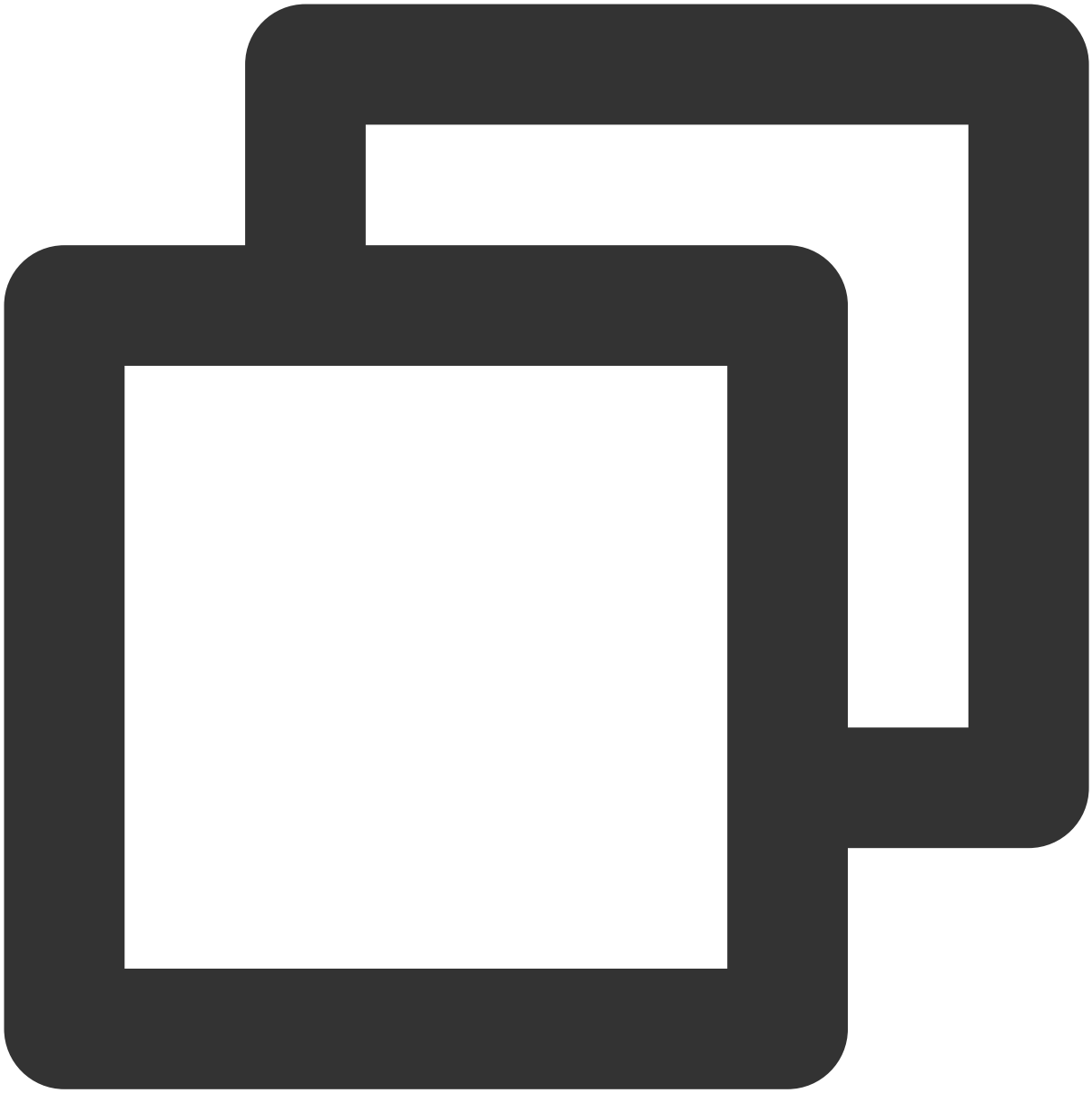
示例：



```
> SELECT corr(c1, c2) FROM (VALUES (3, 2), (3, 3), (6, 4)) as tab(c1, c2);
 0.8660254037844387
```

COUNT

函数语法：



```
COUNT(*)
COUNT([DISTINCT] <col1> ANY, <col2> ANY, ...)
```

支持引擎：SparkSQL、Presto

使用说明：

COUNT(*)：返回检索到的行总数，包括包含null的行。

COUNT(<col1> ANY, <col2> ANY, ...)：返回提供的表达式均为非空的行数。

COUNT([DISTINCT] <col1> ANY,<col2> ANY, ...)：返回提供的表达式唯一且非空的行数。

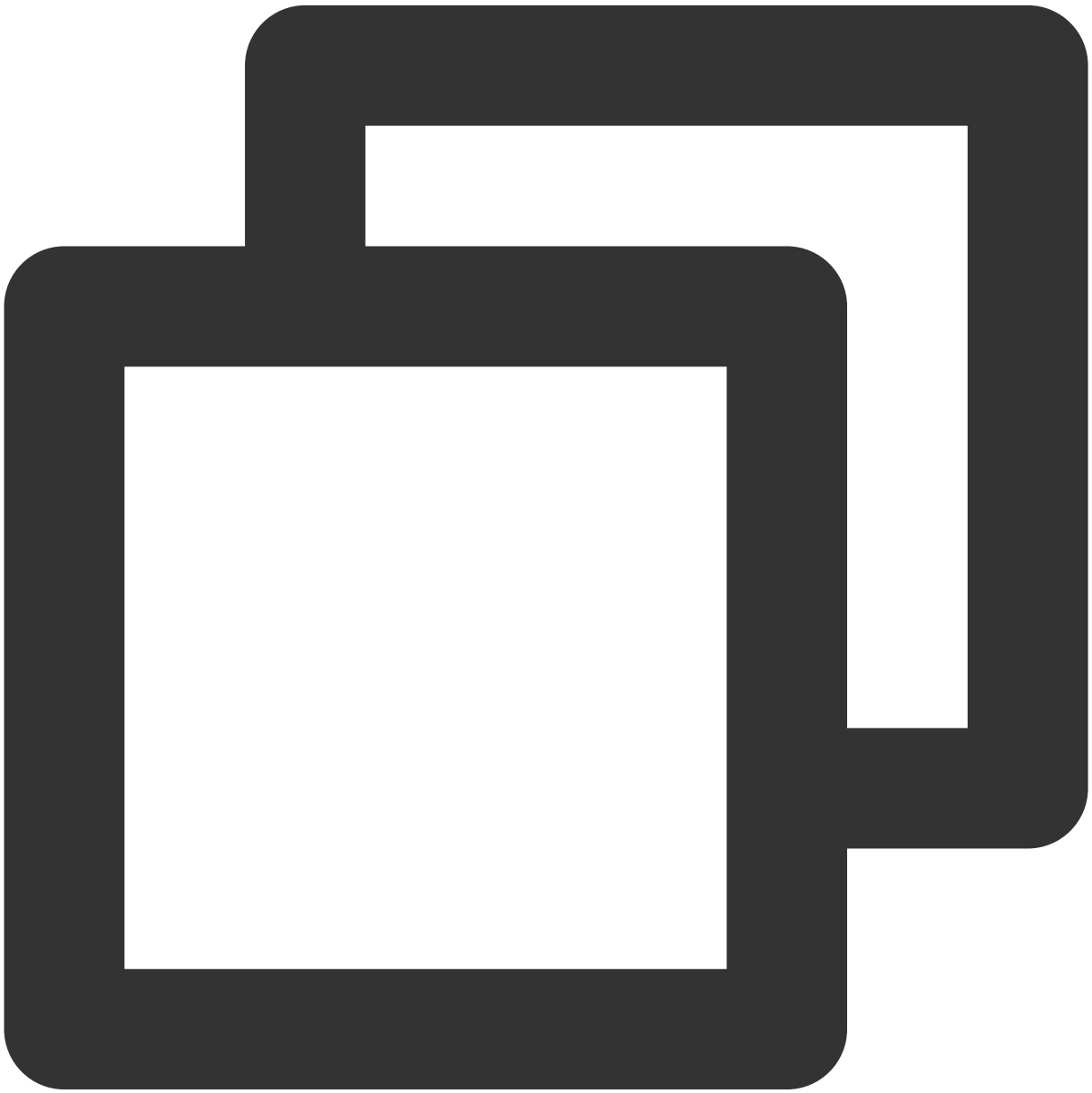
返回类型：integer

示例：

```
> SELECT count(*) FROM (VALUES (NULL), (5), (5), (20)) AS tab(col);
 4
> SELECT count(col) FROM (VALUES (NULL), (5), (5), (20)) AS tab(col);
 3
> SELECT count(DISTINCT col) (FROM VALUES (NULL), (5), (5), (10)) AS tab(col);
 2
```

COUNT_IF

函数语法：



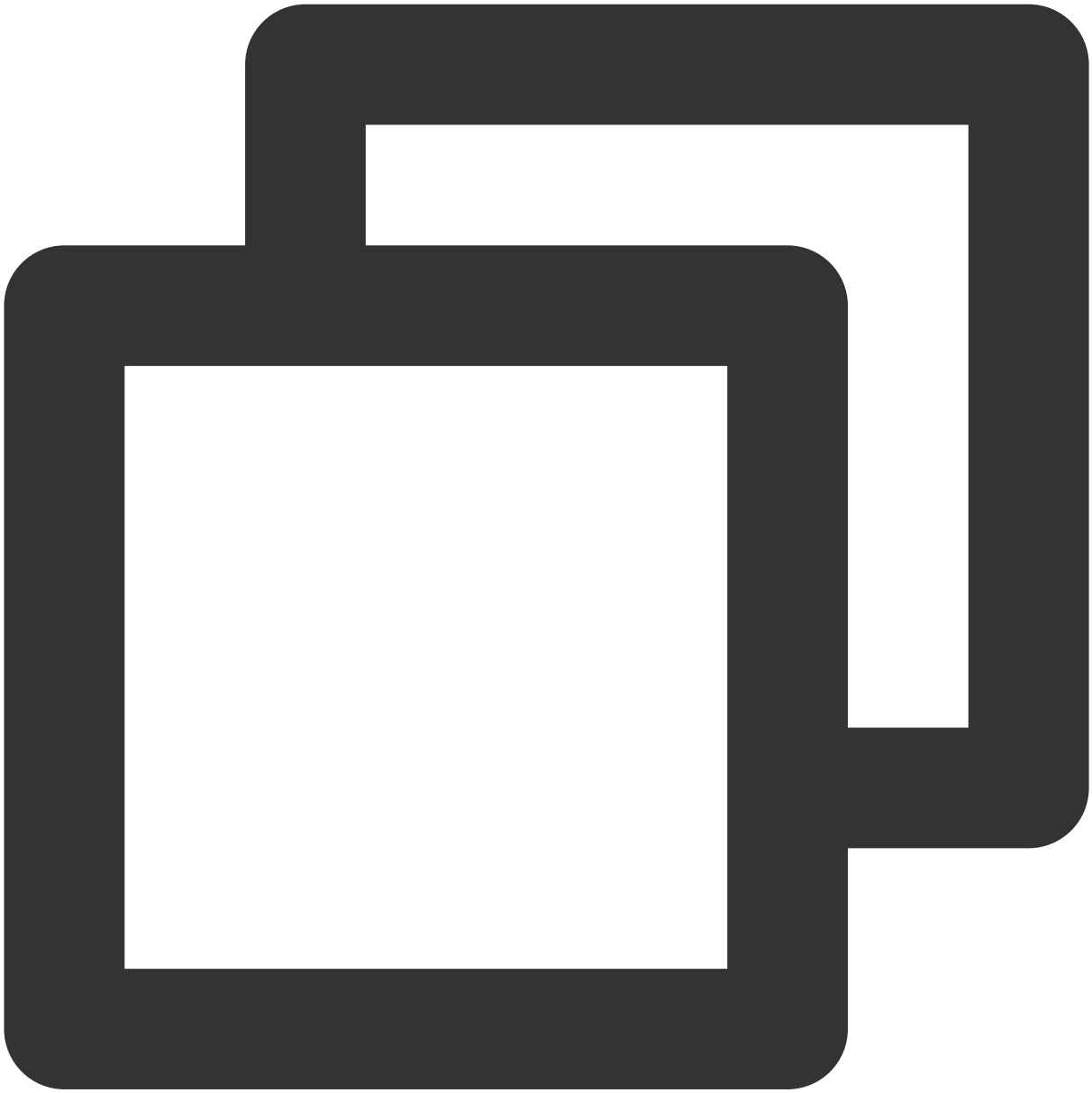
```
COUNT_IF(<expr> ANY)
```

支持引擎：SparkSQL、Presto

使用说明：返回表达式为TRUE的行数。

返回类型：int

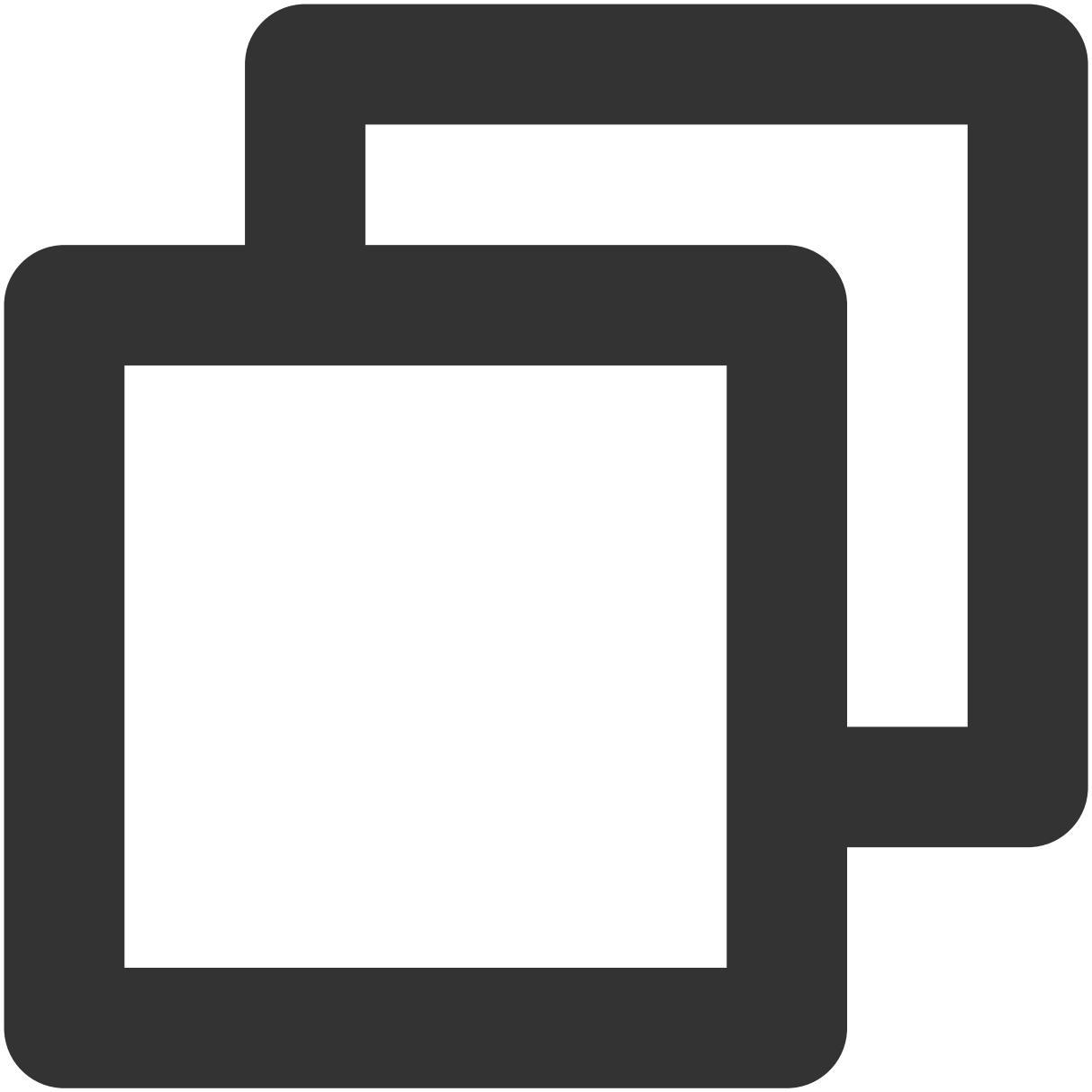
示例：



```
> SELECT count_if(col % 2 = 0) FROM (VALUES (NULL), (0), (1), (2), (3)) AS tab(col)
 2
> SELECT count_if(col IS NULL) FROM (VALUES (NULL), (0), (1), (2), (3)) AS tab(col)
 1
```

COVER_POP

函数语法：



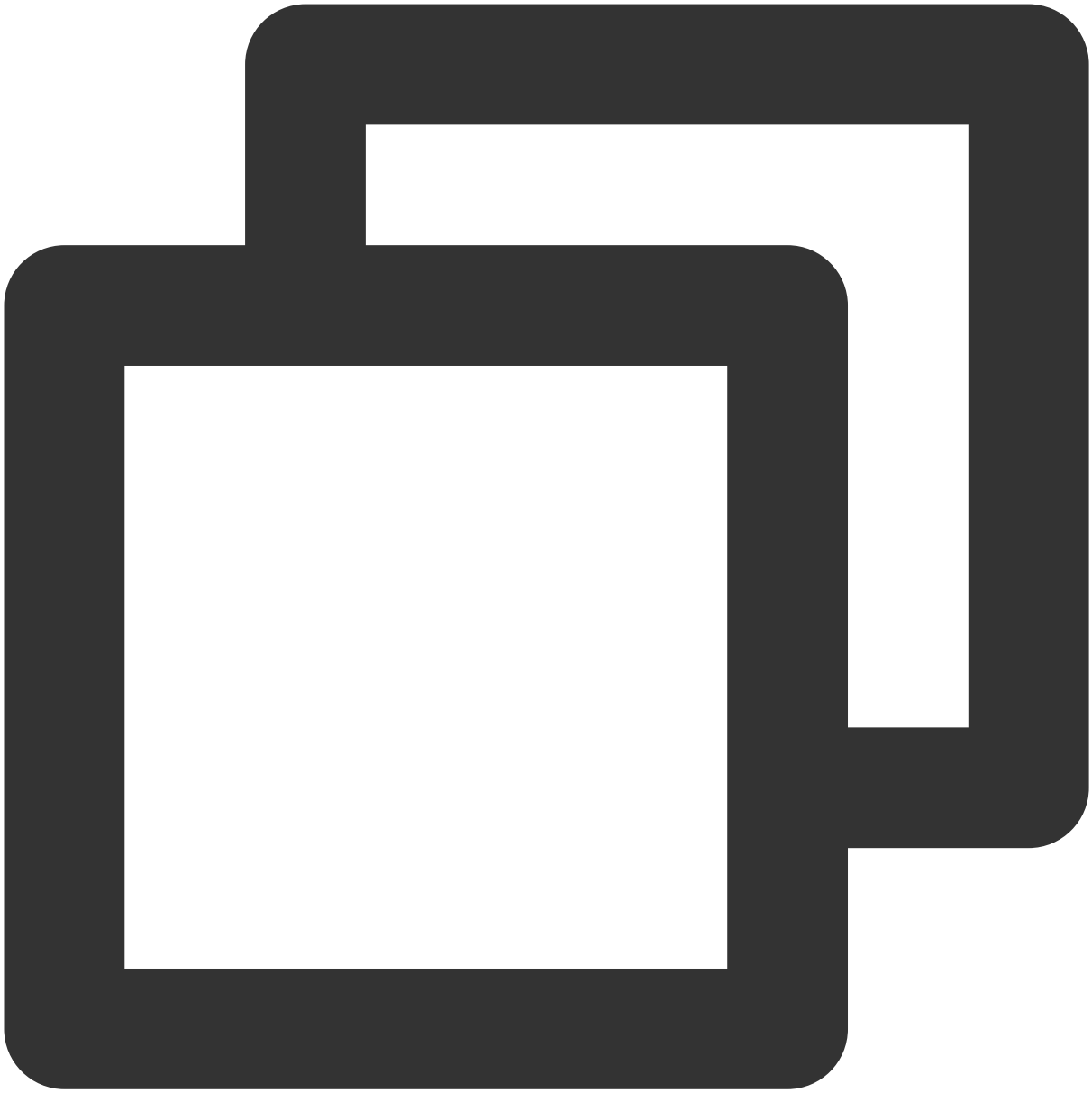
```
COVAR_POP(<expr1> integer|double|decimal, <expr2> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回一组数字对的总体协方差。

返回类型：double

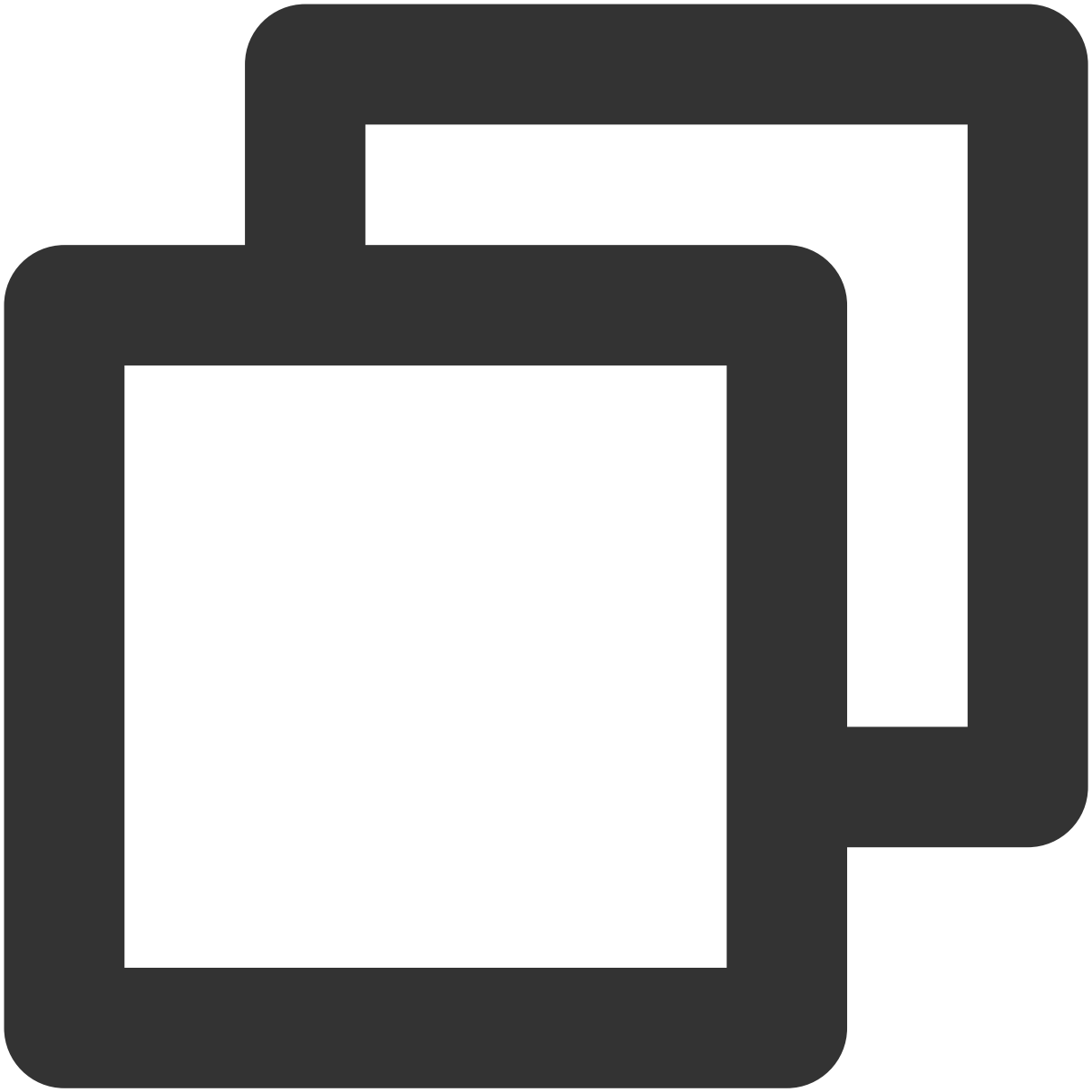
示例：



```
> SELECT covar_pop(c1, c2) FROM (VALUES (1,1), (2,2), (3,3)) AS tab(c1, c2);
 0.6666666666666666
```

COVER_SAMP

函数语法：



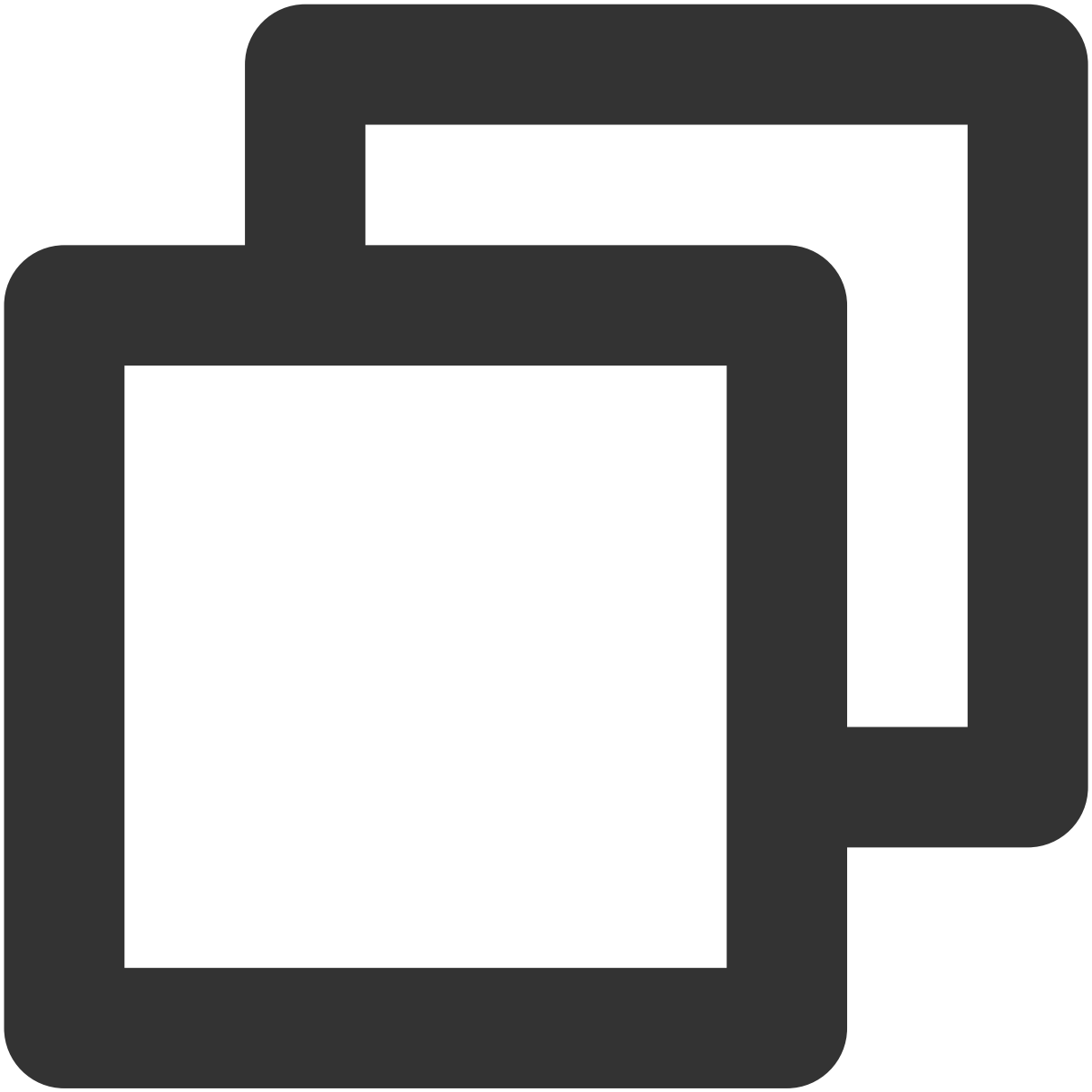
```
COVAR_SAMP(<expr1> integer|double|decimal, <expr2> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回一组数字对的样本协方差。

返回类型：double

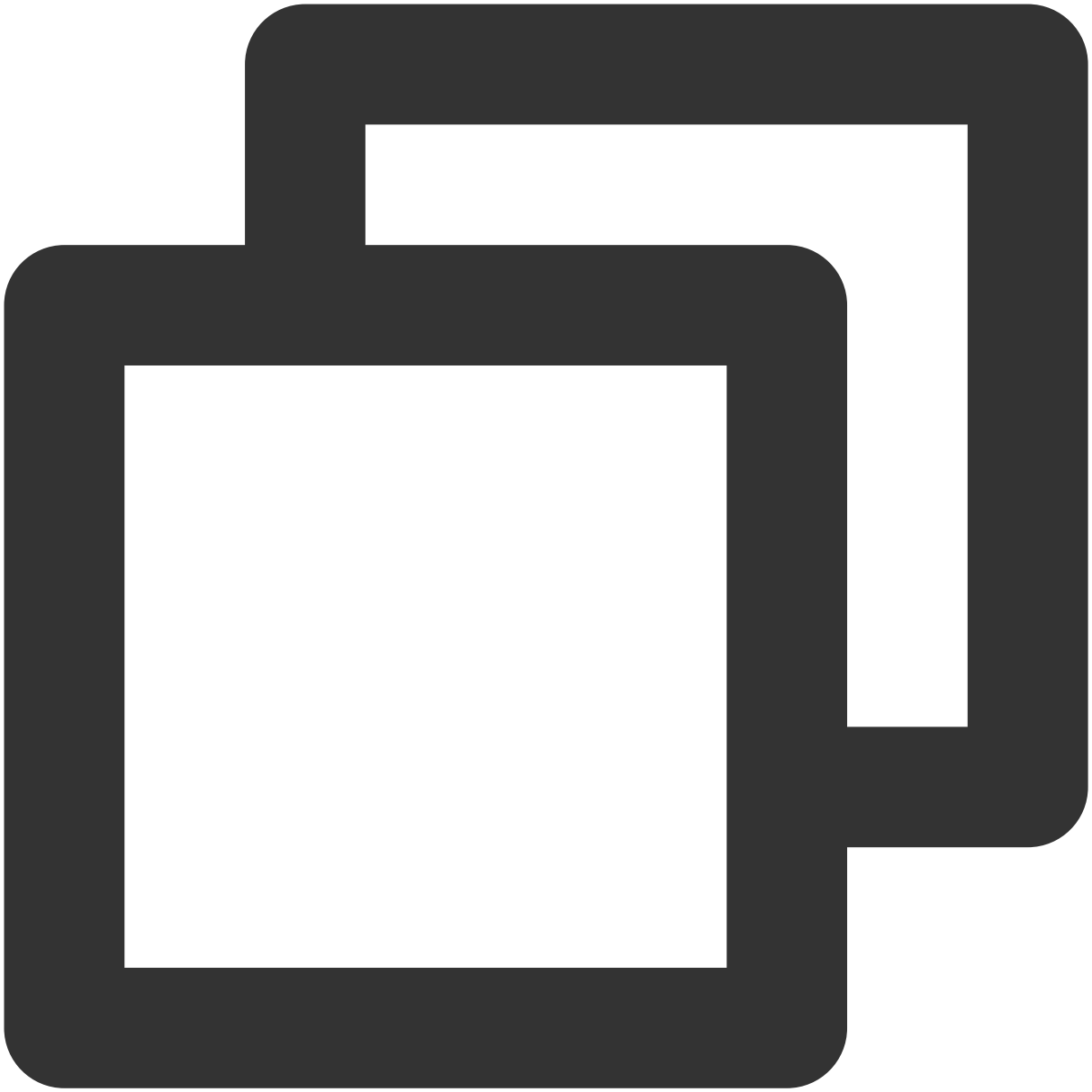
示例：



```
> SELECT covar_samp(c1, c2) FROM (VALUES (1,1), (2,2), (3,3)) AS tab(c1, c2);
 1.0
```

FIRST_VALUE

函数语法：



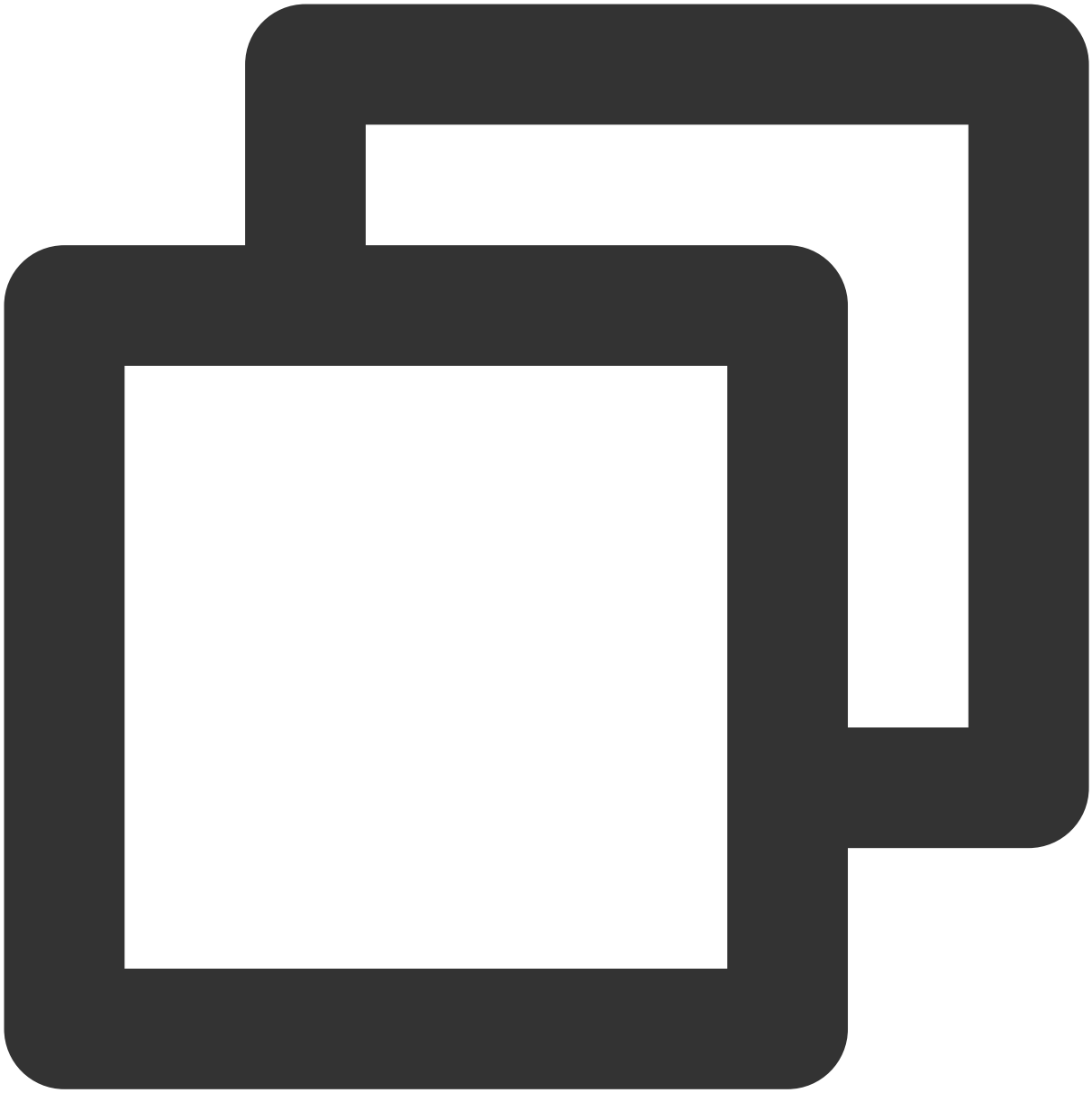
```
FIRST_VALUE(<expr> T[, <isIgnoreNull> boolean])
```

支持引擎：SparkSQL、Presto

使用说明：返回一组行的expr的第一个值。如果isIgnoreNull为true，则仅返回非null值。

返回类型：T

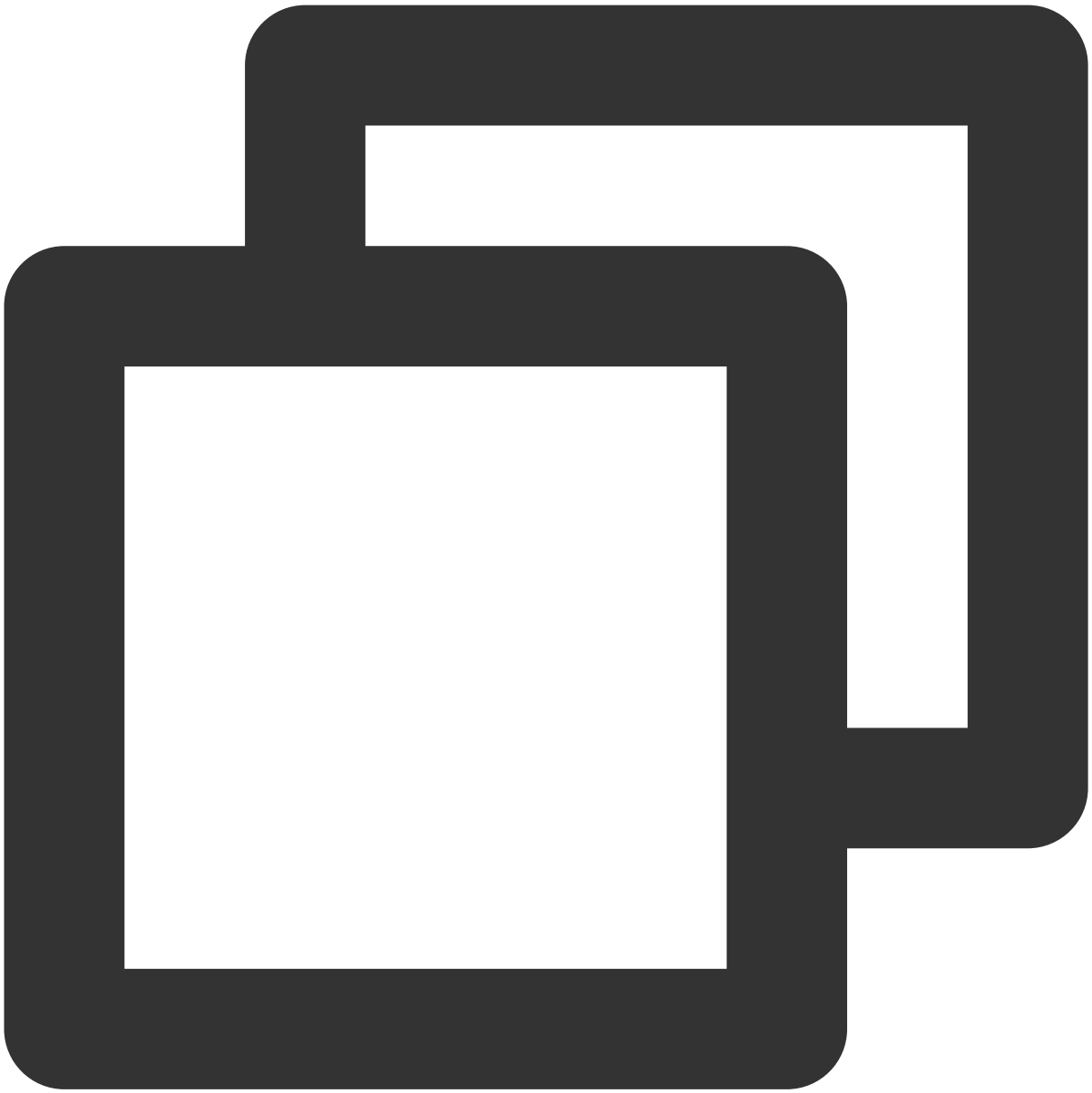
示例：



```
> SELECT first_value(col) FROM (VALUES (10), (5), (20)) AS tab(col);
 10
> SELECT first_value(col) FROM (VALUES (NULL), (5), (20)) AS tab(col);
 NULL
> SELECT first_value(col, true) FROM (VALUES (NULL), (5), (20)) AS tab(col);
 5
```

FIRST

函数语法：



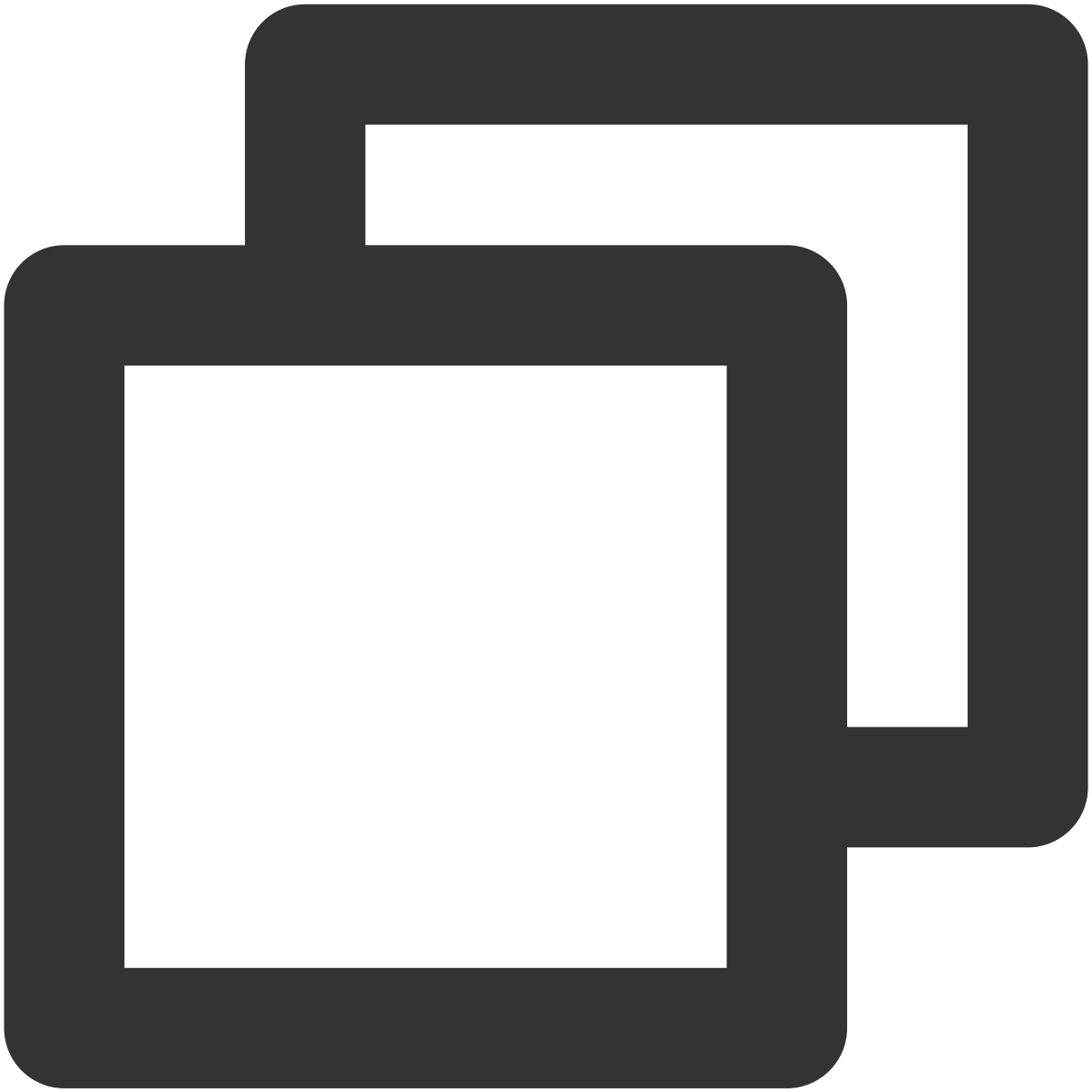
```
FIRST(<expr> T[, <isIgnoreNull> boolean])
```

支持引擎：SparkSQL

使用说明：返回一组行的expr的第一个值。如果isIgnoreNull为true，则仅返回非null值。

返回类型：T

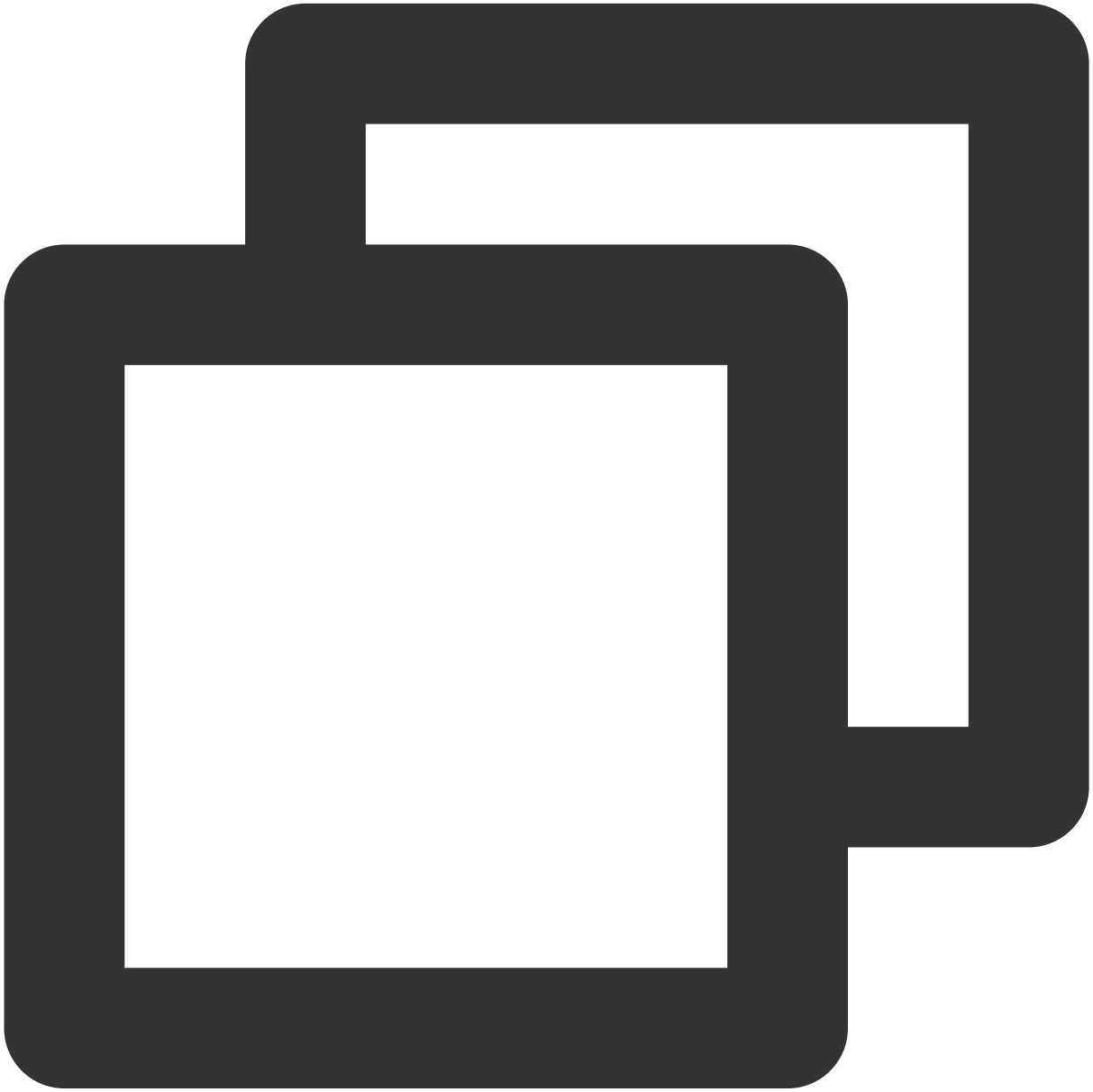
示例：



```
> SELECT first(col) FROM (VALUES (10), (5), (20)) AS tab(col);
 10
> SELECT first(col) FROM (VALUES (NULL), (5), (20)) AS tab(col);
 NULL
> SELECT first(col, true) FROM (VALUES (NULL), (5), (20)) AS tab(col);
 5
```

KURTOSIS

函数语法：



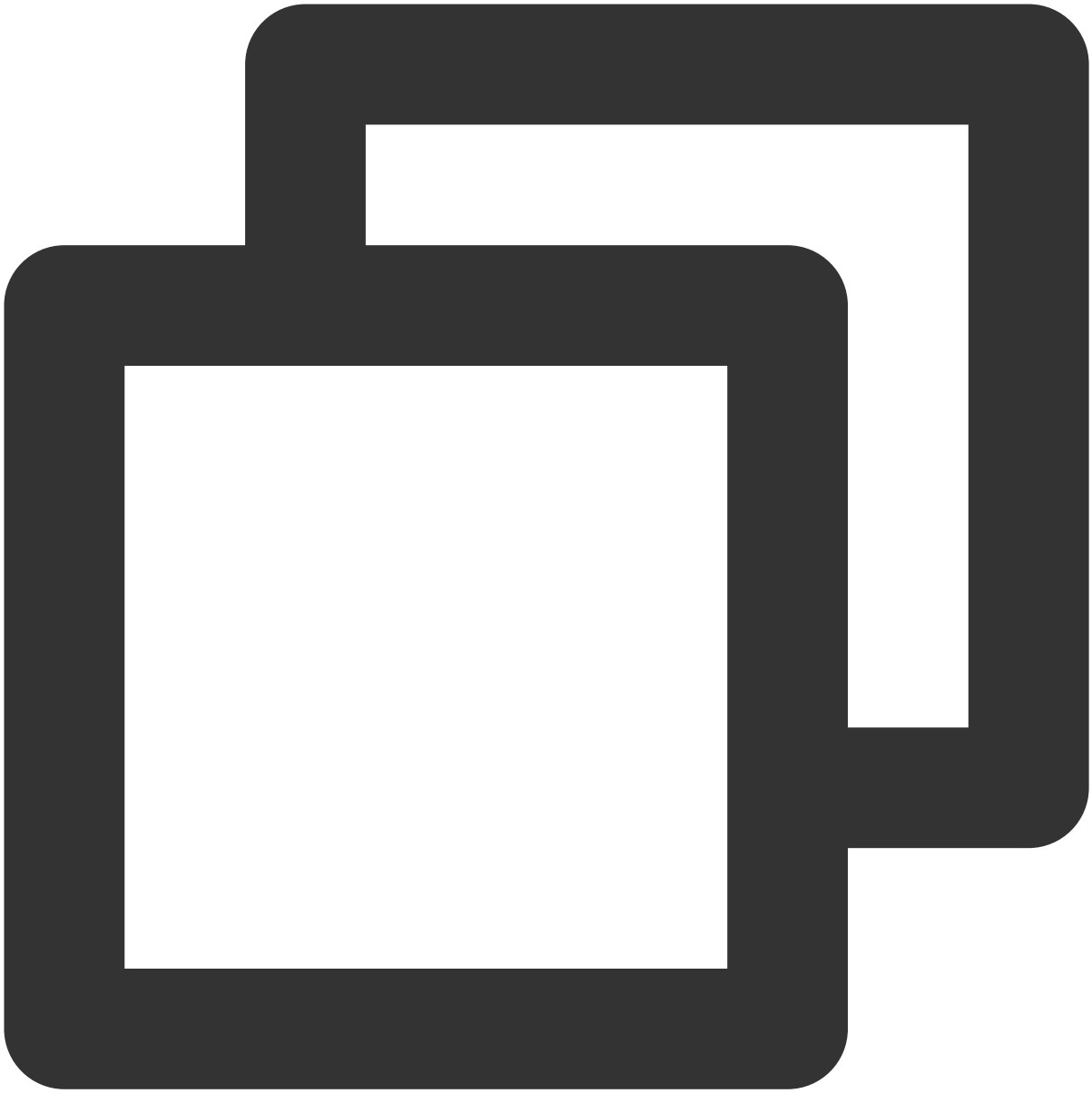
```
KURTOSIS(<expr> integer|double|decimal)
```

支持引擎：SparkSQL

使用说明：返回根据组的值计算得出的峰度值。

返回类型：double

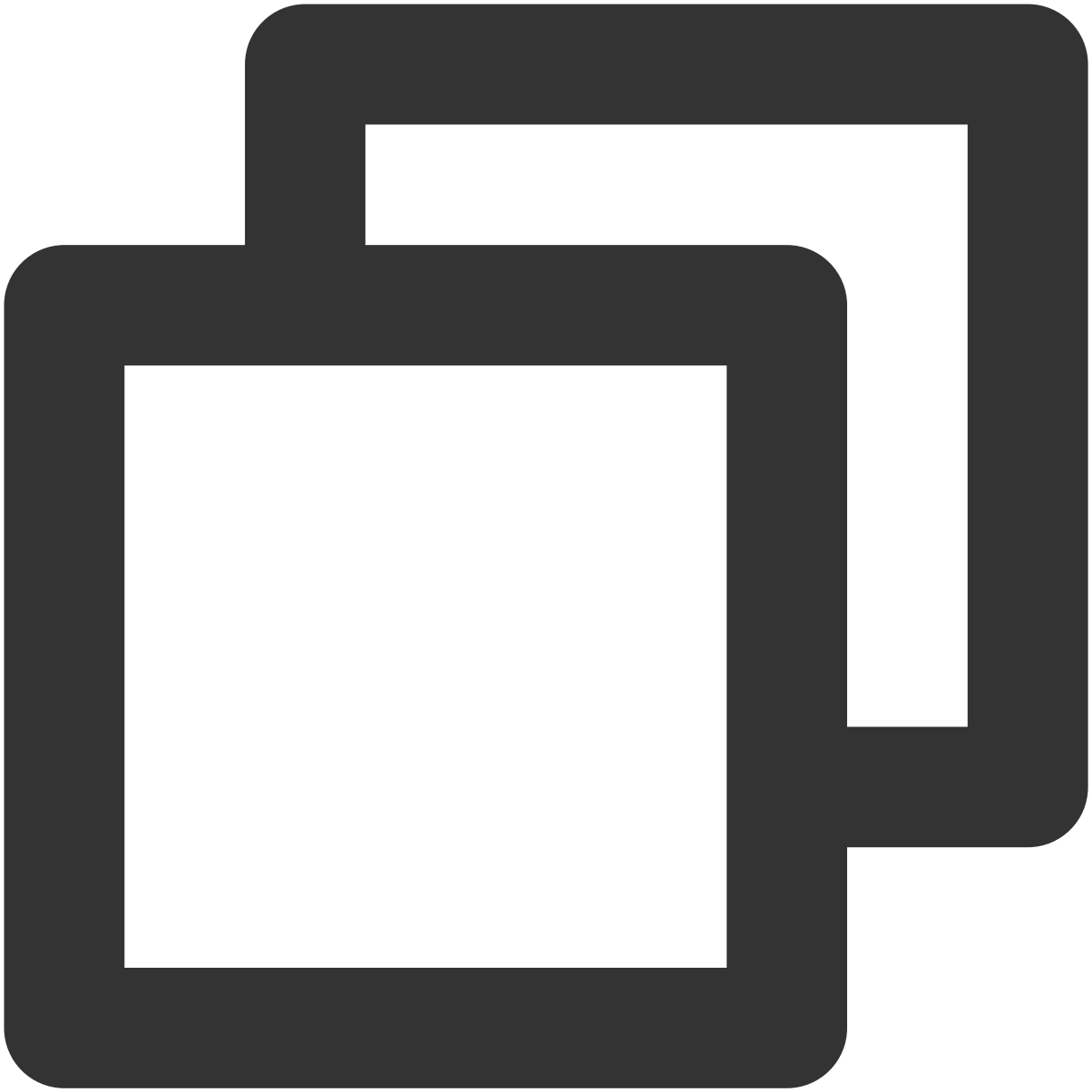
示例：



```
> SELECT kurtosis(col) FROM (VALUES (-10), (-20), (100), (1000)) AS tab(col);
 -0.7014368047529627
> SELECT kurtosis(col) FROM (VALUES (1), (10), (100), (10), (1)) as tab(col);
 0.19432323191699075s
```

LAST_VALUE

函数语法：



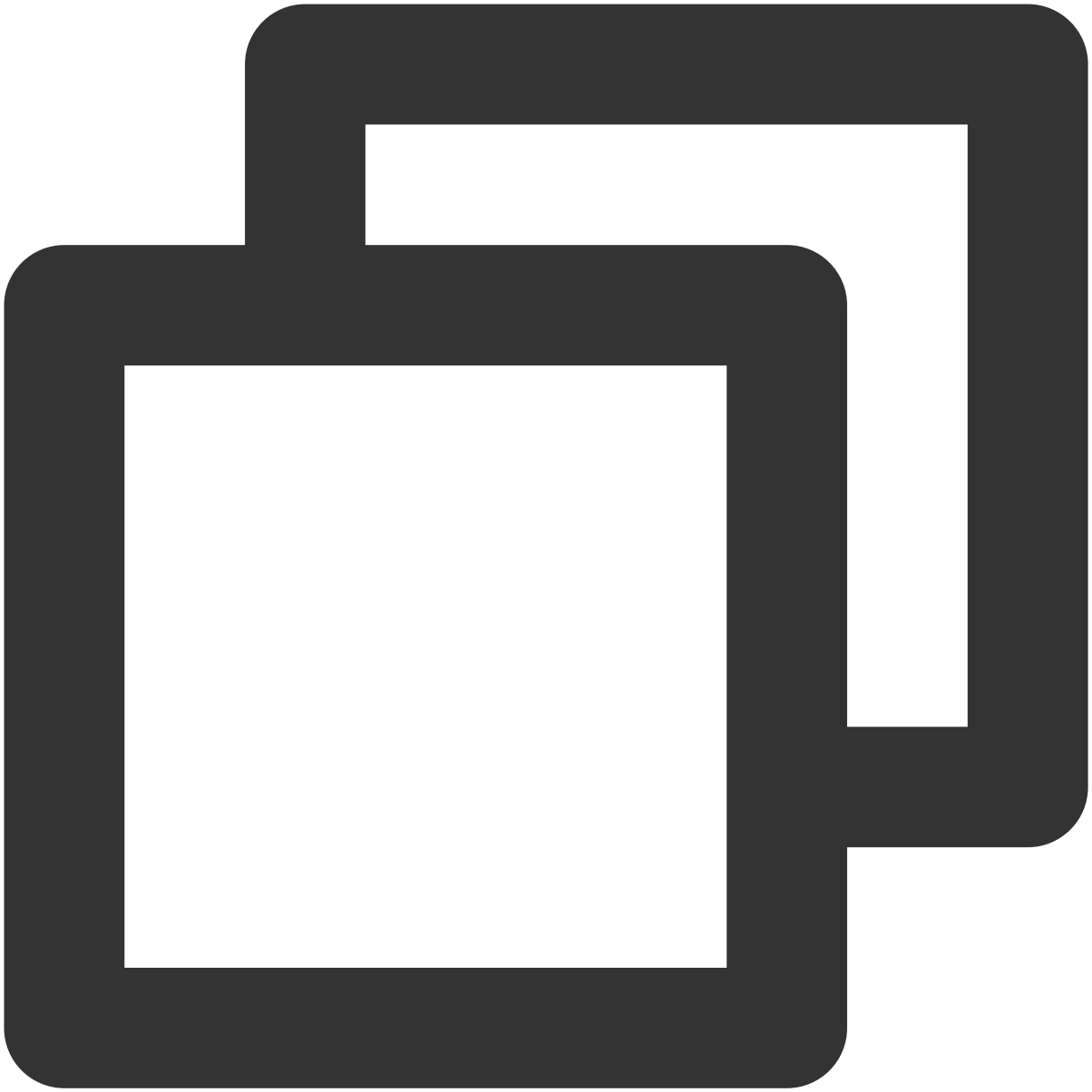
```
LAST_VALUE(<expr> T[, <isIgnoreNull> boolean])
```

支持引擎：SparkSQL、Presto

使用说明：返回一组行的expr的最后一个值。如果isIgnoreNull为true，则仅返回非null值。

返回类型：T

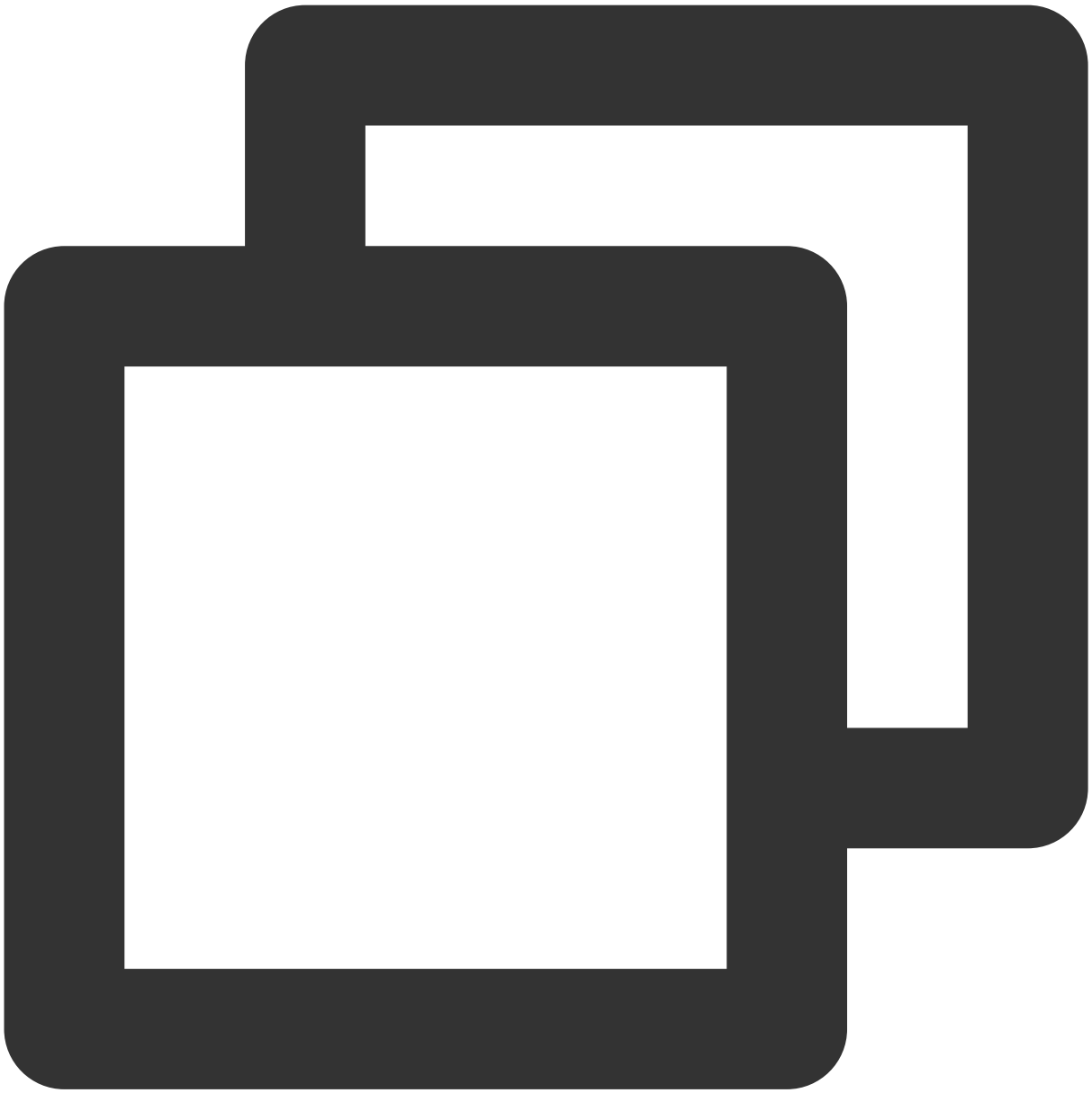
示例：



```
> SELECT last_value(col) FROM (VALUES (10), (5), (20)) AS tab(col);
 20
> SELECT last_value(col) FROM (VALUES (10), (5), (NULL)) AS tab(col);
 NULL
> SELECT last_value(col, true) FROM (VALUES (10), (5), (NULL)) AS tab(col);
 5
```

LAST

函数语法：



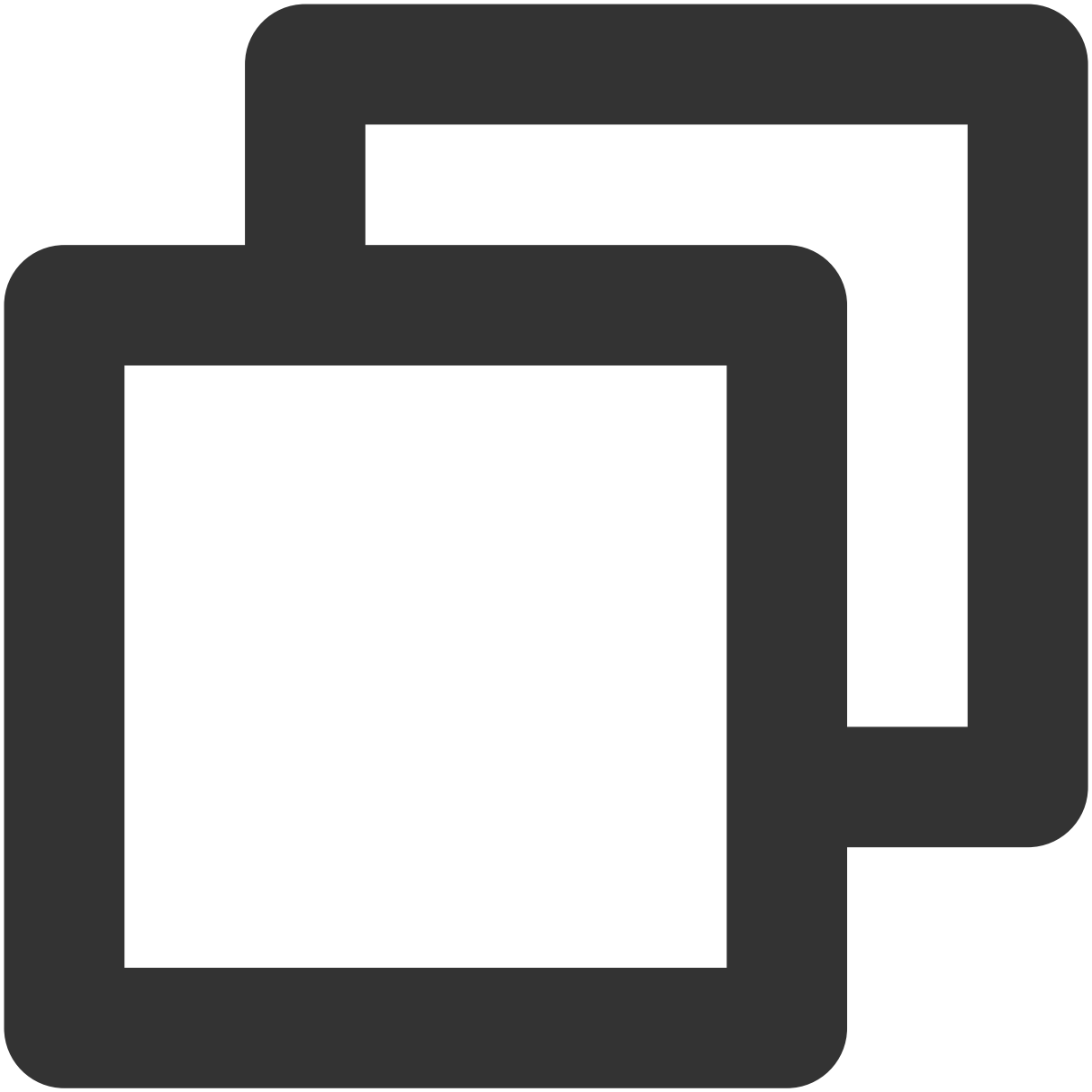
```
LAST(<expr> T[, <isIgnoreNull> boolean])
```

支持引擎：SparkSQL

使用说明：返回一组行的expr的最后一个值。如果isIgnoreNull为true，则仅返回非null值。

返回类型：T

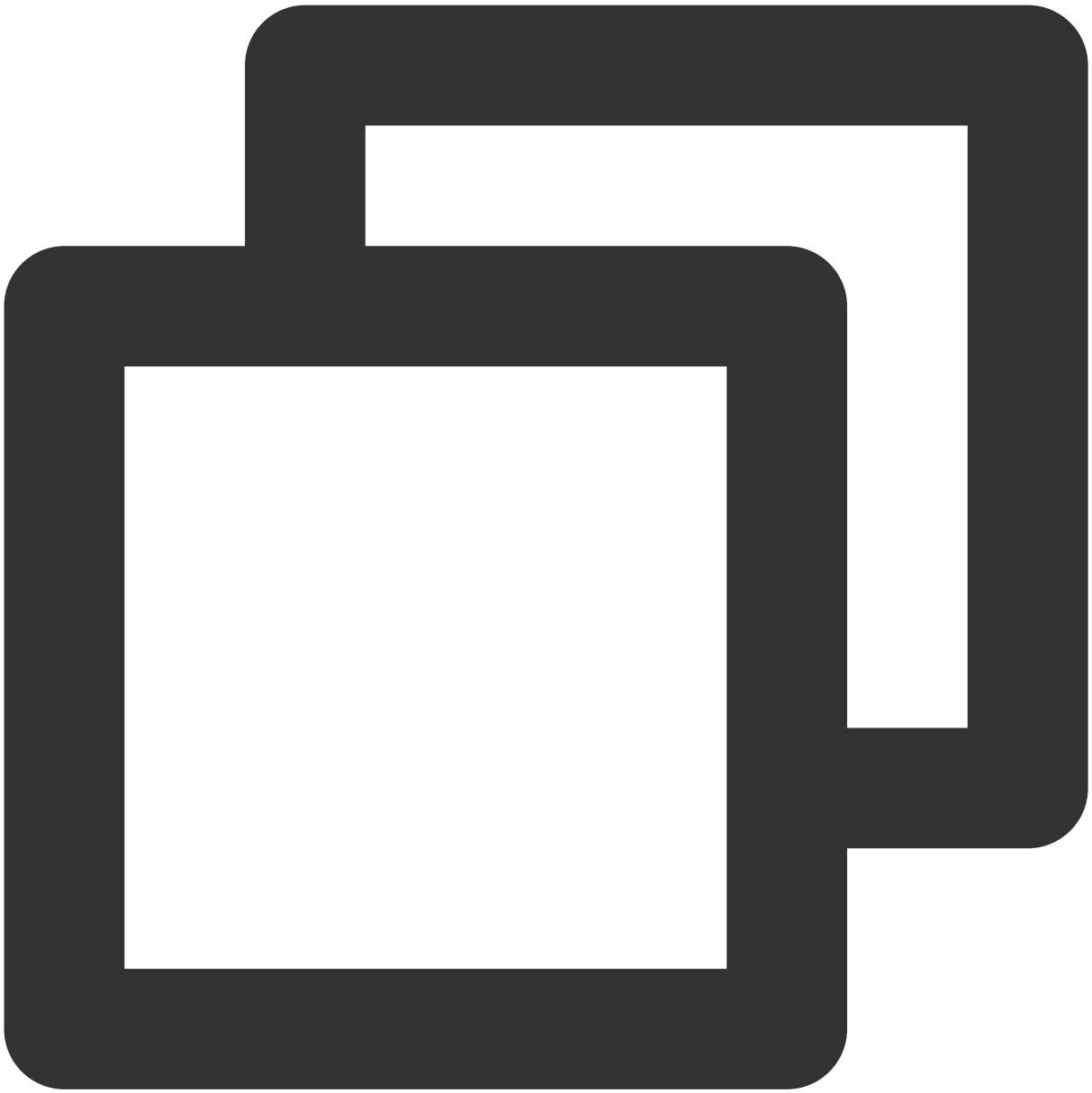
示例：



```
> SELECT last(col) FROM (VALUES (10), (5), (20)) AS tab(col);
 20
> SELECT last(col) FROM (VALUES (10), (5), (NULL)) AS tab(col);
 NULL
> SELECT last(col, true) FROM (VALUES (10), (5), (NULL)) AS tab(col);
 5
```

MEAN

函数语法：



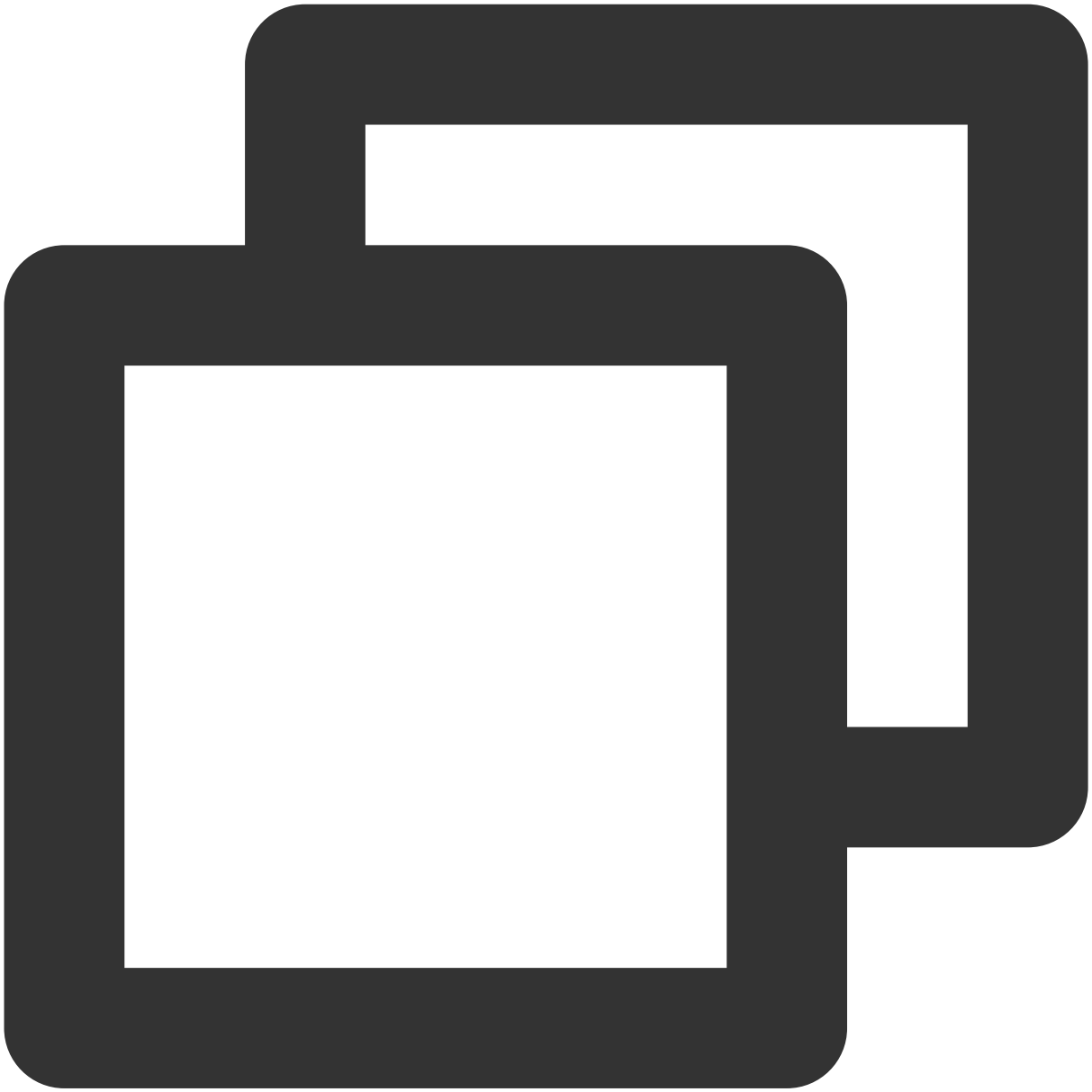
```
MEAN(<expr> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回根据组的值计算的平均值

返回类型：double

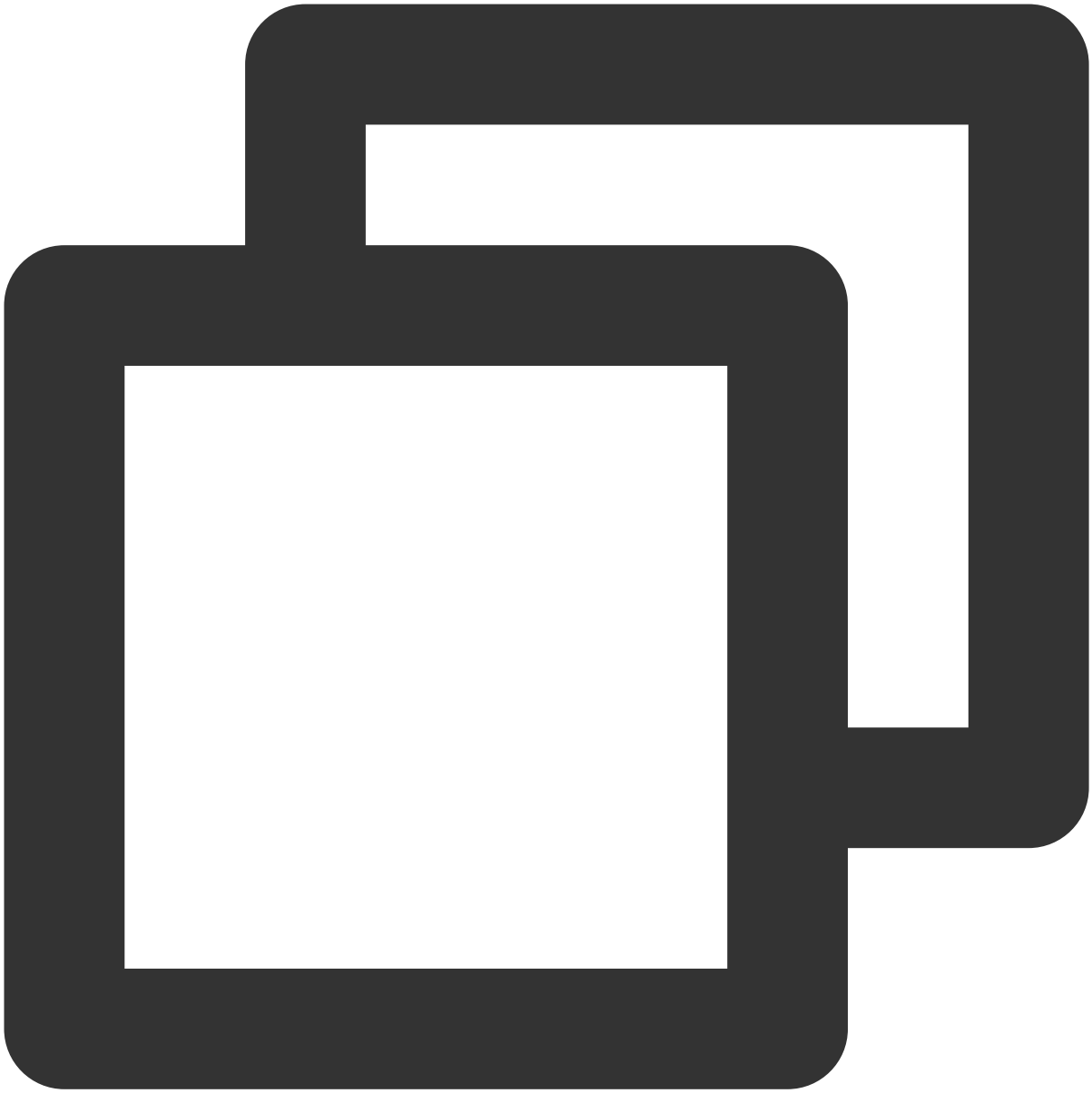
示例：



```
> SELECT mean(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 2.0
> SELECT mean(col) FROM (VALUES (1), (2), (NULL)) AS tab(col);
 1.5
```

PERCENTILE

函数语法：



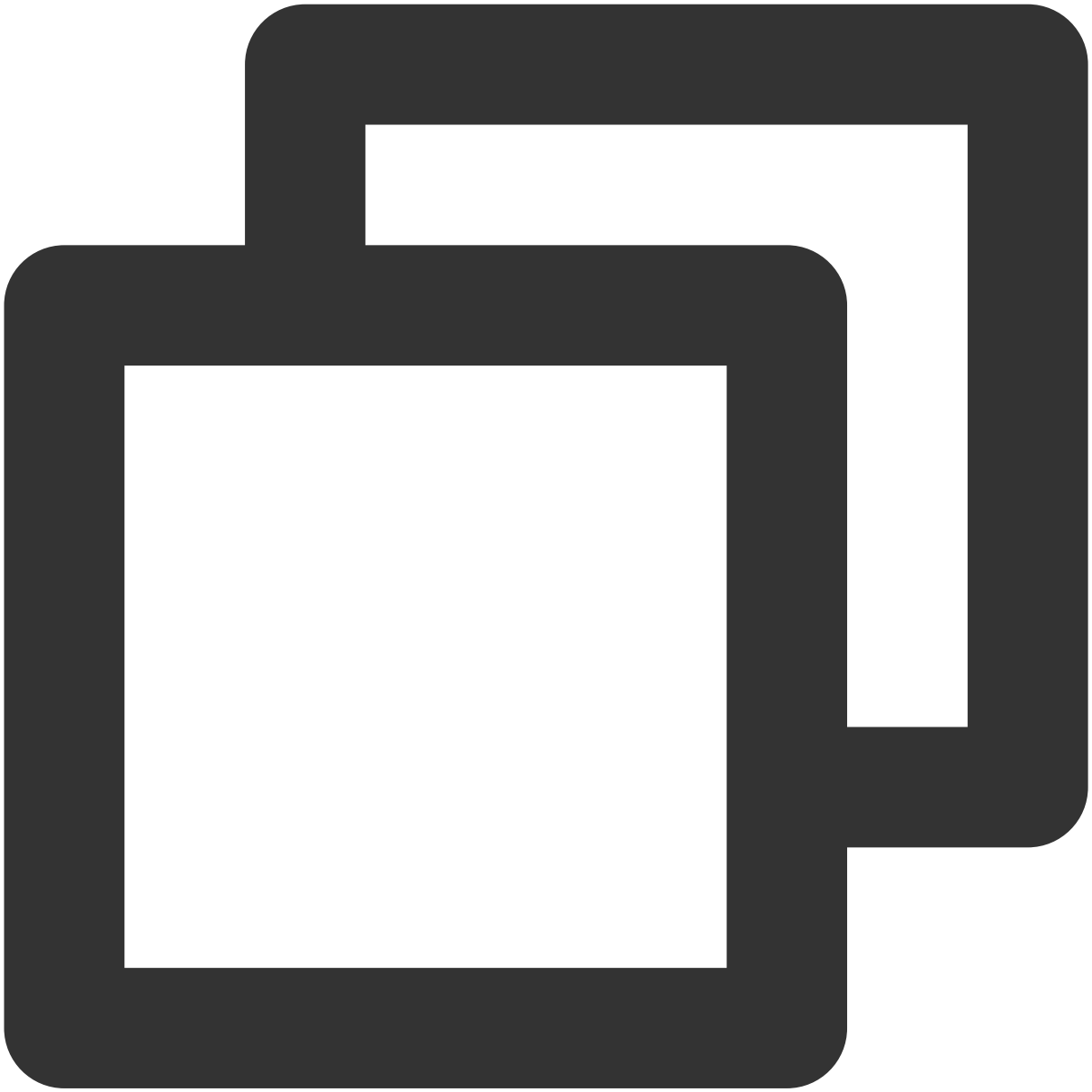
```
PERCENTILE(<col> ANY, <percentage> integer|double|decimal|array<double> [, <frequen
```

支持引擎：SparkSQL、Presto

使用说明：返回给定百分比下数值列col的精确百分位值。percentage值必须介于0.0和1.0之间。frequency应为正整数

返回类型：double

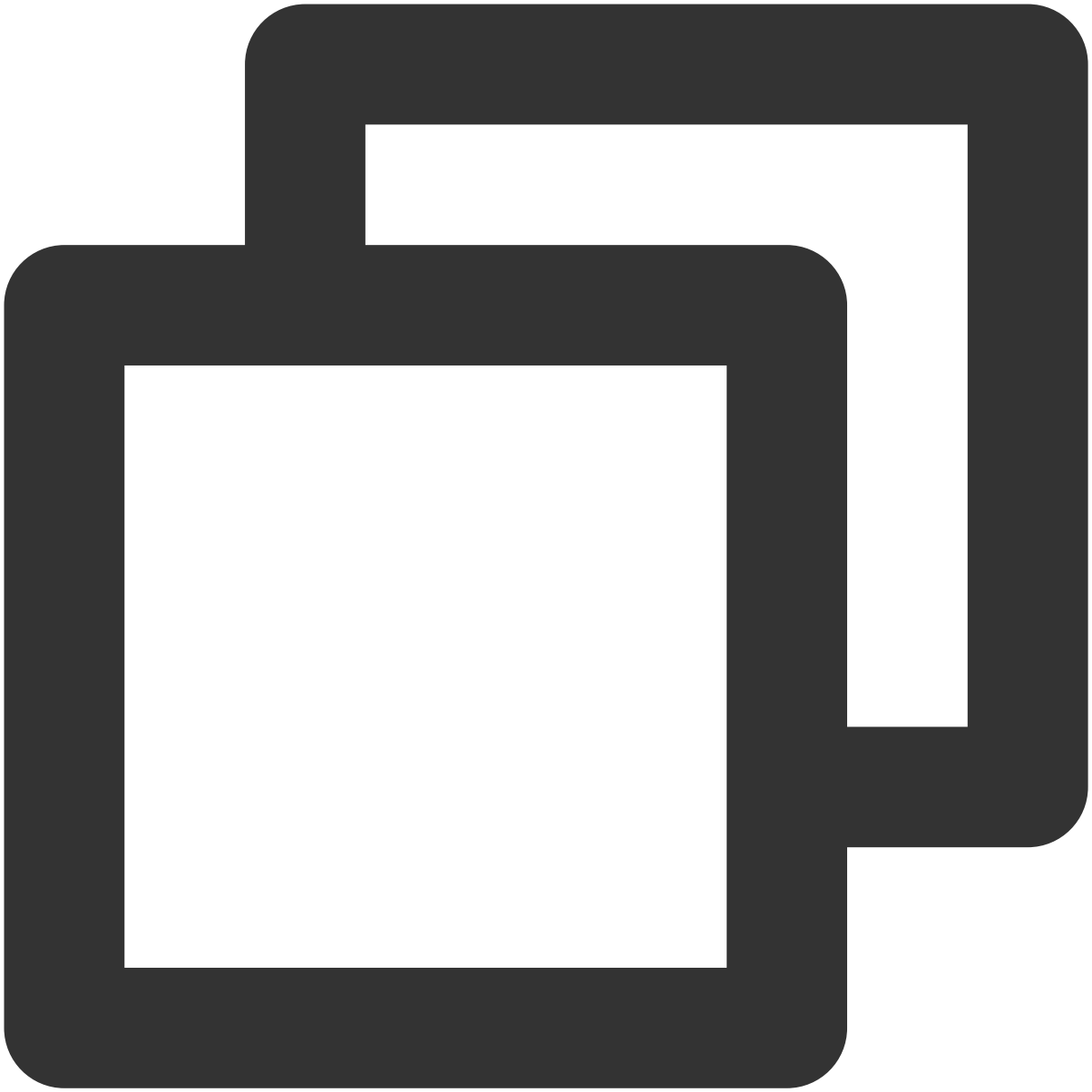
示例：



```
> SELECT percentile(col, 0.3) FROM (VALUES (0), (10)) AS tab(col);
 3.0
> SELECT percentile(col, array(0.25, 0.75)) FROM (VALUES (0), (10)) AS tab(col);
 [2.5,7.5]
```

SKEWNESS

函数语法：



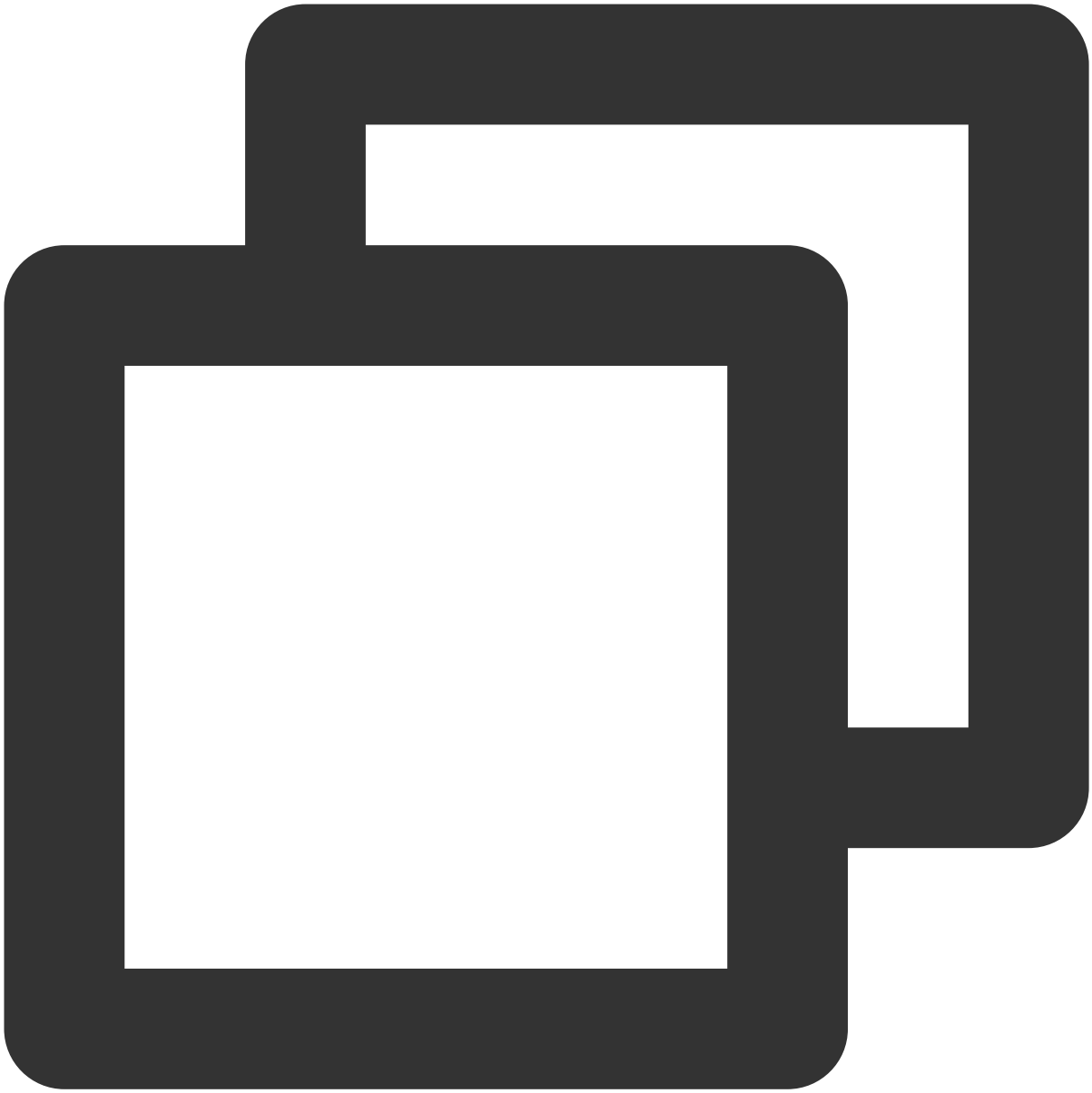
```
SKEWNESS(<col> integer|double|decimal)
```

支持引擎：SparkSQL

使用说明：返回根据组的值计算的偏度值

返回类型：double

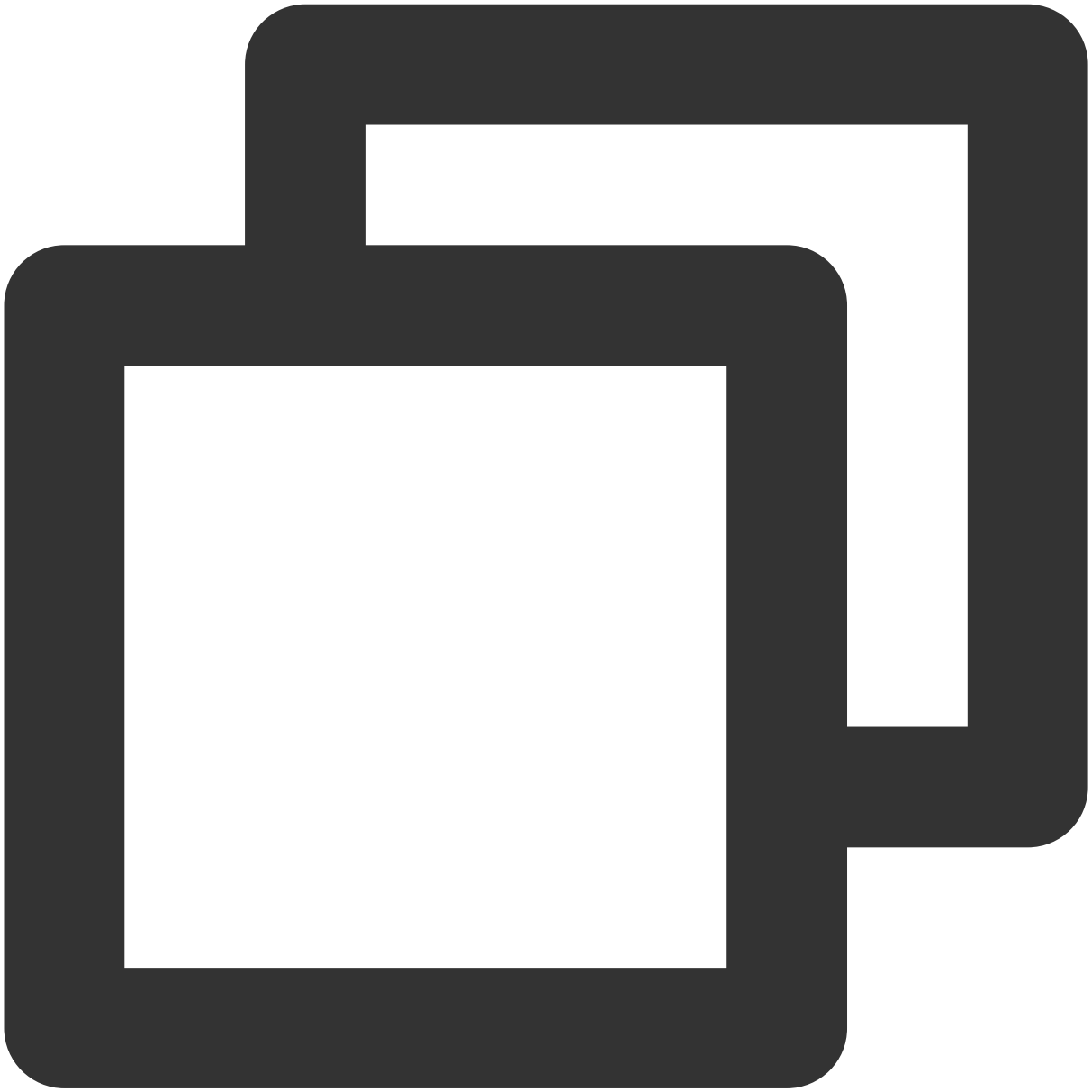
示例：



```
> SELECT skewness(col) FROM (VALUES (-10), (-20), (100), (1000)) AS tab(col);
 1.1135657469022011
> SELECT skewness(col) FROM (VALUES (-1000), (-100), (10), (20)) AS tab(col);
 -1.1135657469022011
```

PERCENTILE_APPROX

函数语法：



```
PERCENTILE_APPROX(<col> integer|double|decimal, <percentage> double|array<double>[,
```

支持引擎：SparkSQL、Presto

使用说明：返回数值列col的近似百分比，该数值是有序col值（从最小到最大排序）中的最小值，因此不超过col值的百分比小于或等于该值。percentage的值必须介于0.0和1.0之间。accuracy参数（默认值：10000）是一个正数值文字，它以内存为代价控制近似精度。精度值越高，精度越好，1.0/accuracy是近似的相对误差。当percentage是一个数组时，percentage数组的每个值必须介于0.0和1.0之间。在这种情况下，返回给定percentage数组中列col的近似百分比数组。

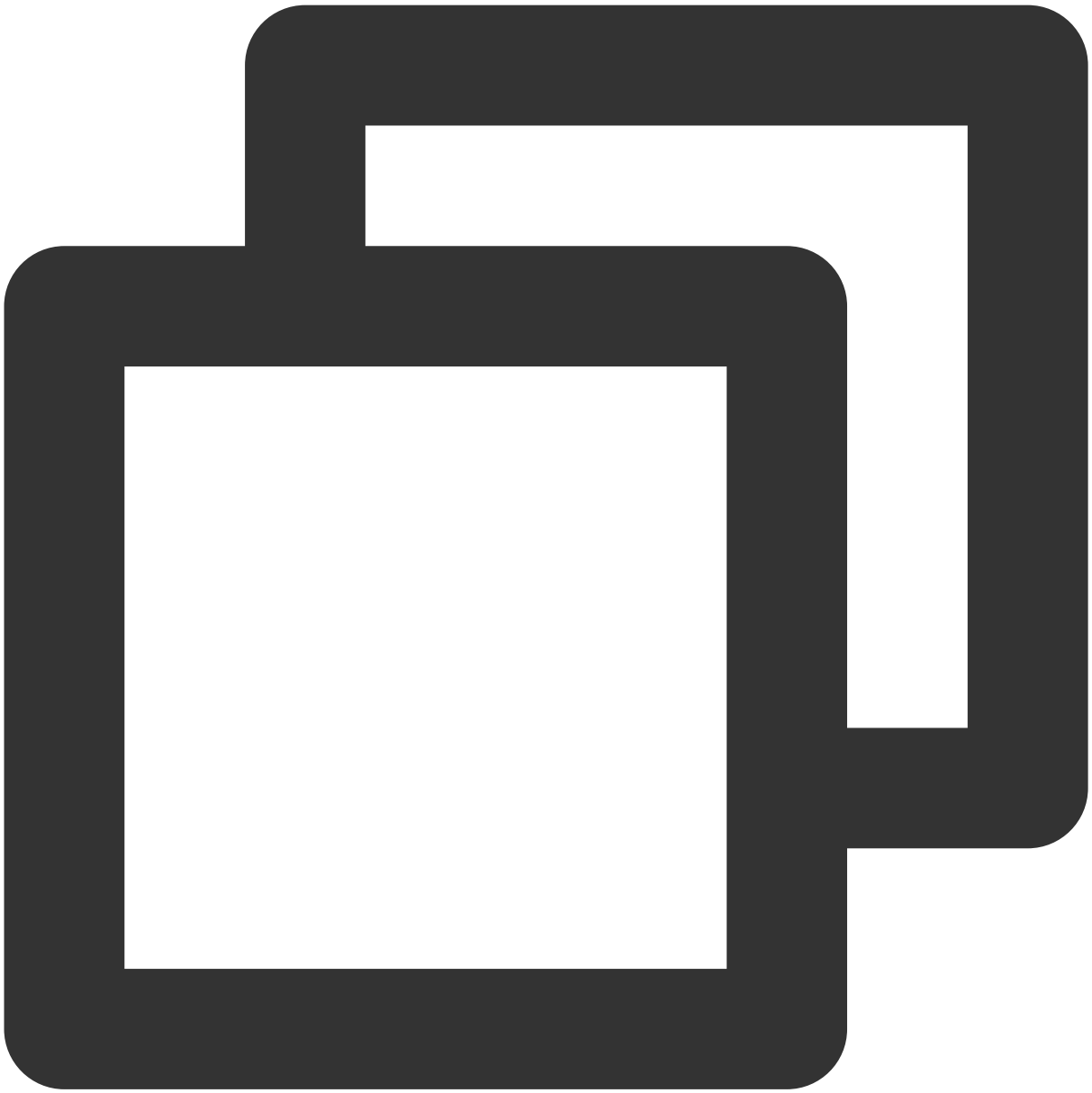
返回类型：integer |array<integer>

示例：

```
> SELECT percentile_approx(col, array(0.5, 0.4, 0.1), 100) FROM (VALUES (0), (1), (
 [1,1,0]
> SELECT percentile_approx(col, 0.5, 100) FROM (VALUES (0), (6), (7), (9), (10)) AS
 7
```

APPROX_PERCENTILE

函数语法：



```
APPROX_PERCENTILE(<col> integer|double|decimal, <percentage> double|array<double>[,
```

支持引擎：SparkSQL、Presto

使用说明：返回数值列col的近似百分比，该数值是有序col值（从最小到最大排序）中的最小值，因此不超过col值的百分比小于或等于该值。percentage的值必须介于0.0和1.0之间。accuracy参数（默认值：10000）是一个正数值文字，它以内存为代价控制近似精度。精度值越高，精度越好，1.0/accuracy是近似的相对误差。当percentage是一个数组时，percentage数组的每个值必须介于0.0和1.0之间。在这种情况下，返回给定percentage数组中列col的近似百分比数组。

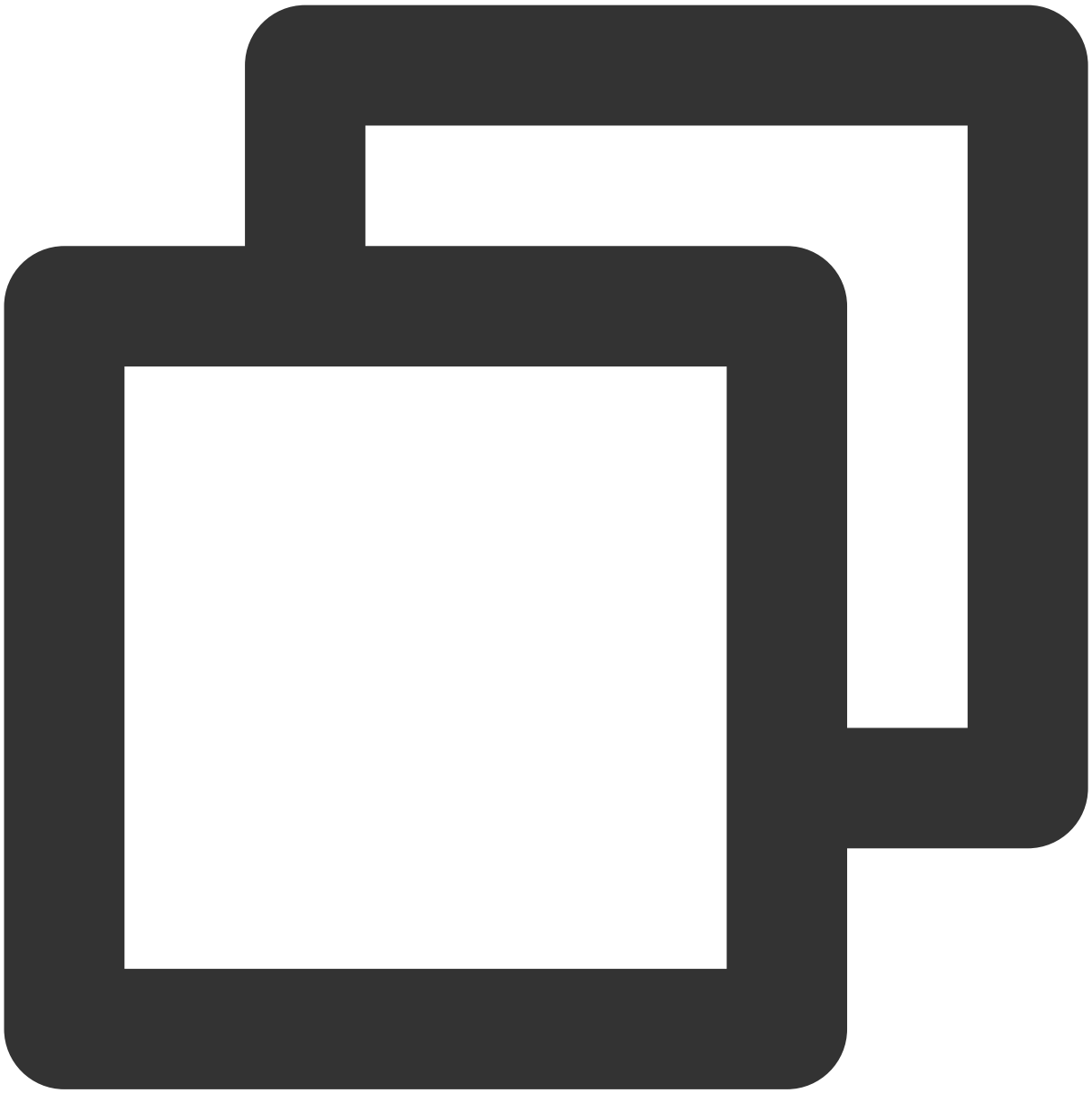
返回类型：integer |array<integer>

示例：

```
> SELECT APPROX_PERCENTILE(col, array(0.5, 0.4, 0.1), 100) FROM (VALUES (0), (1), (
 [1,1,0]
> SELECT APPROX_PERCENTILE(col, 0.5, 100) FROM (VALUES (0), (6), (7), (9), (10)) AS
 7
```

MAX

函数语法：



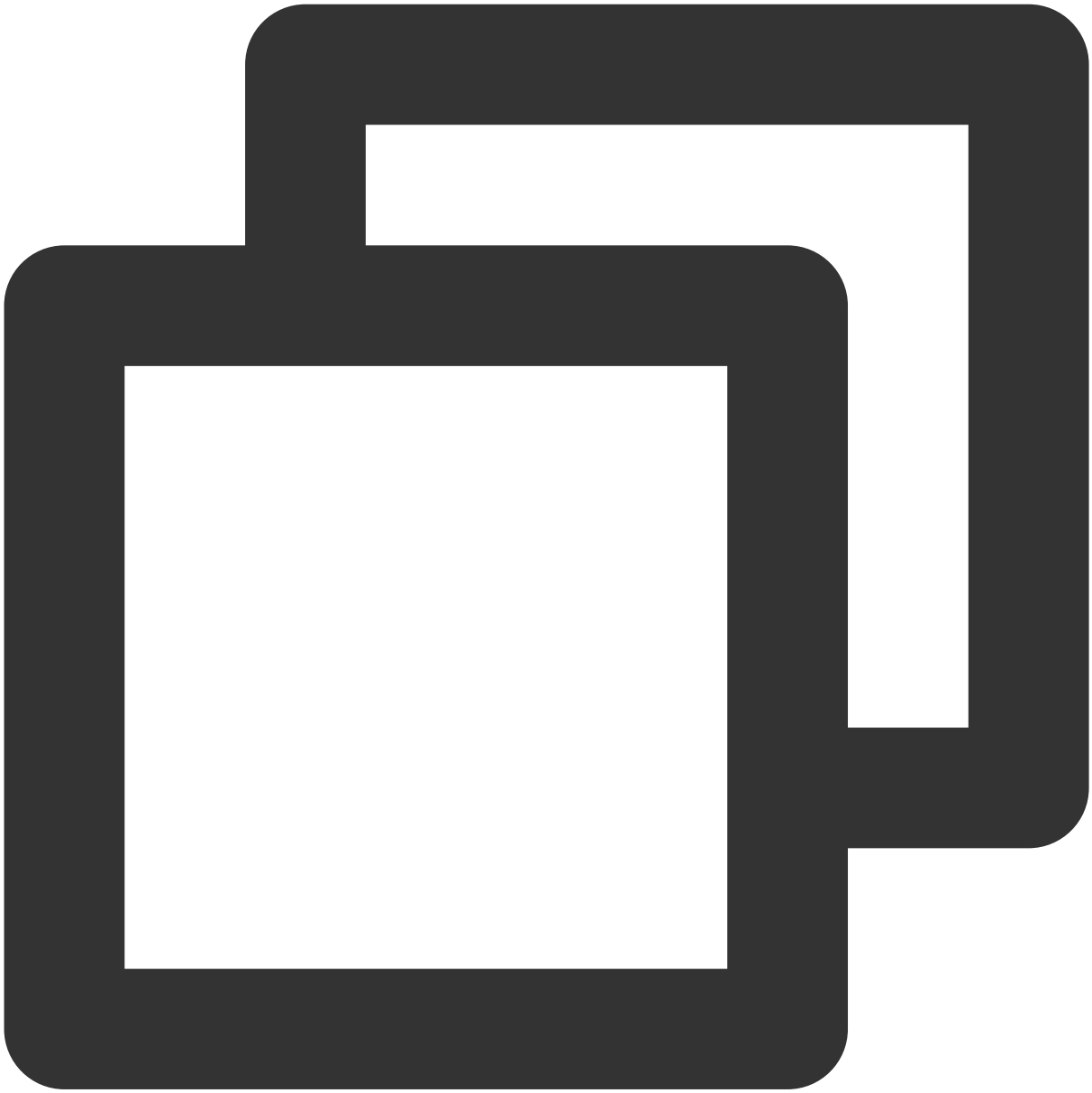
```
MAX(<col> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回col的最大值。

返回类型：与col一致

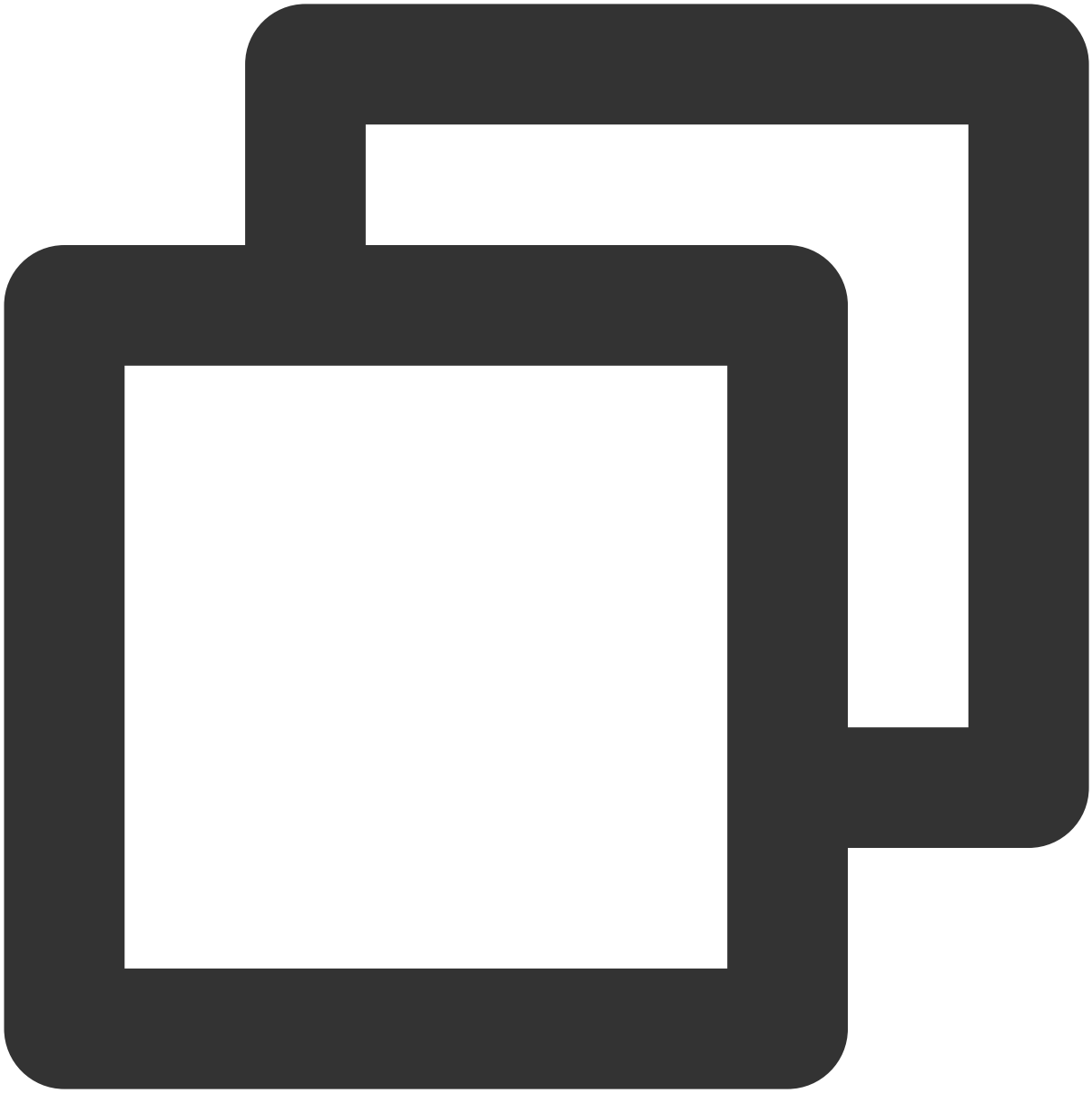
示例：



```
> SELECT max(col) FROM (VALUES (10), (50), (20)) AS tab(col);
 50
```

MAX_BY

函数语法：



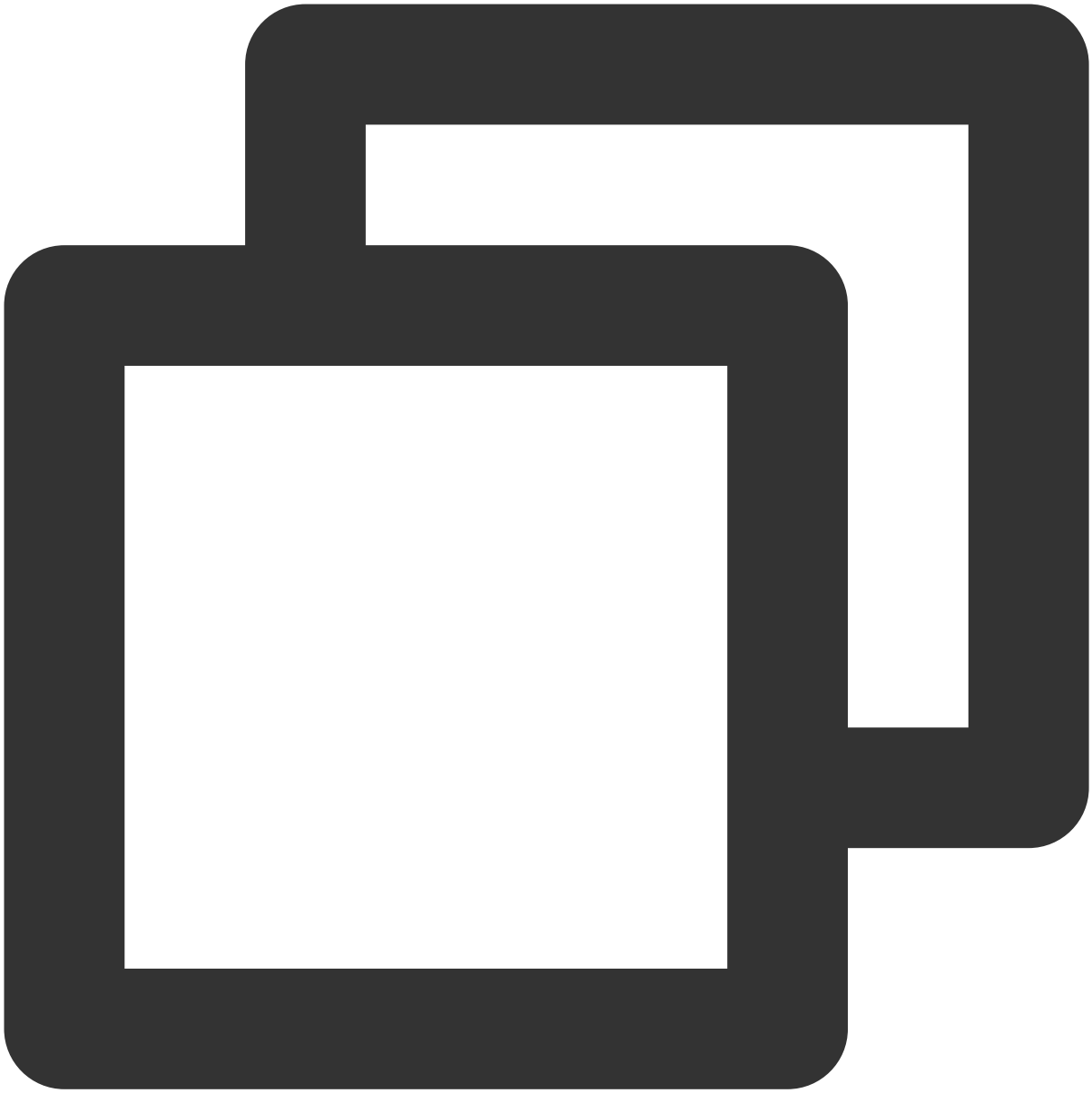
```
MAX_BY(<x> T, <y> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回与y的最大值关联的x值。

返回类型：T

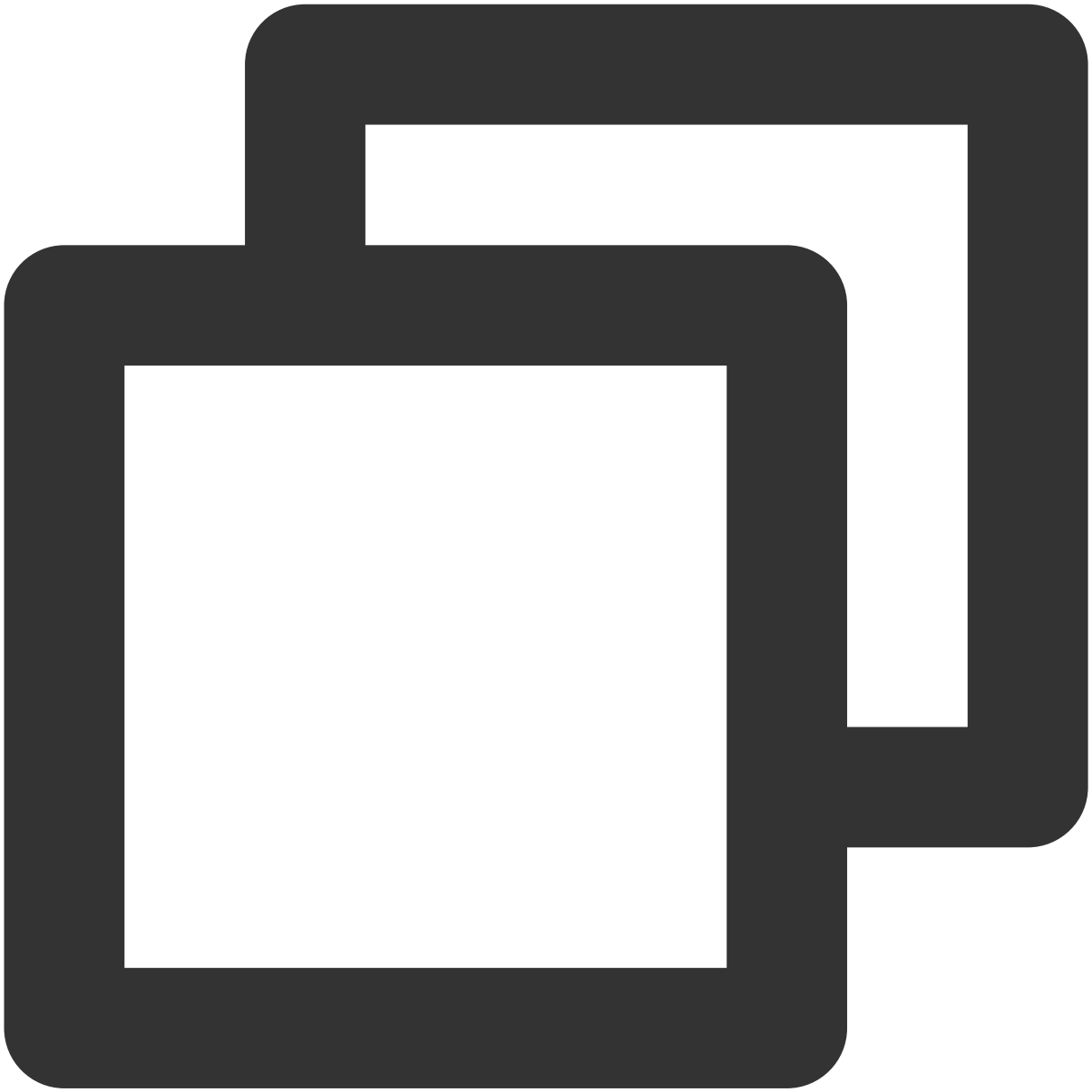
示例：



```
> SELECT max_by(x, y) FROM (VALUES (('a', 10)), (('b', 50)), (('c', 20))) AS tab(x,
 b
```

MIN

函数语法：



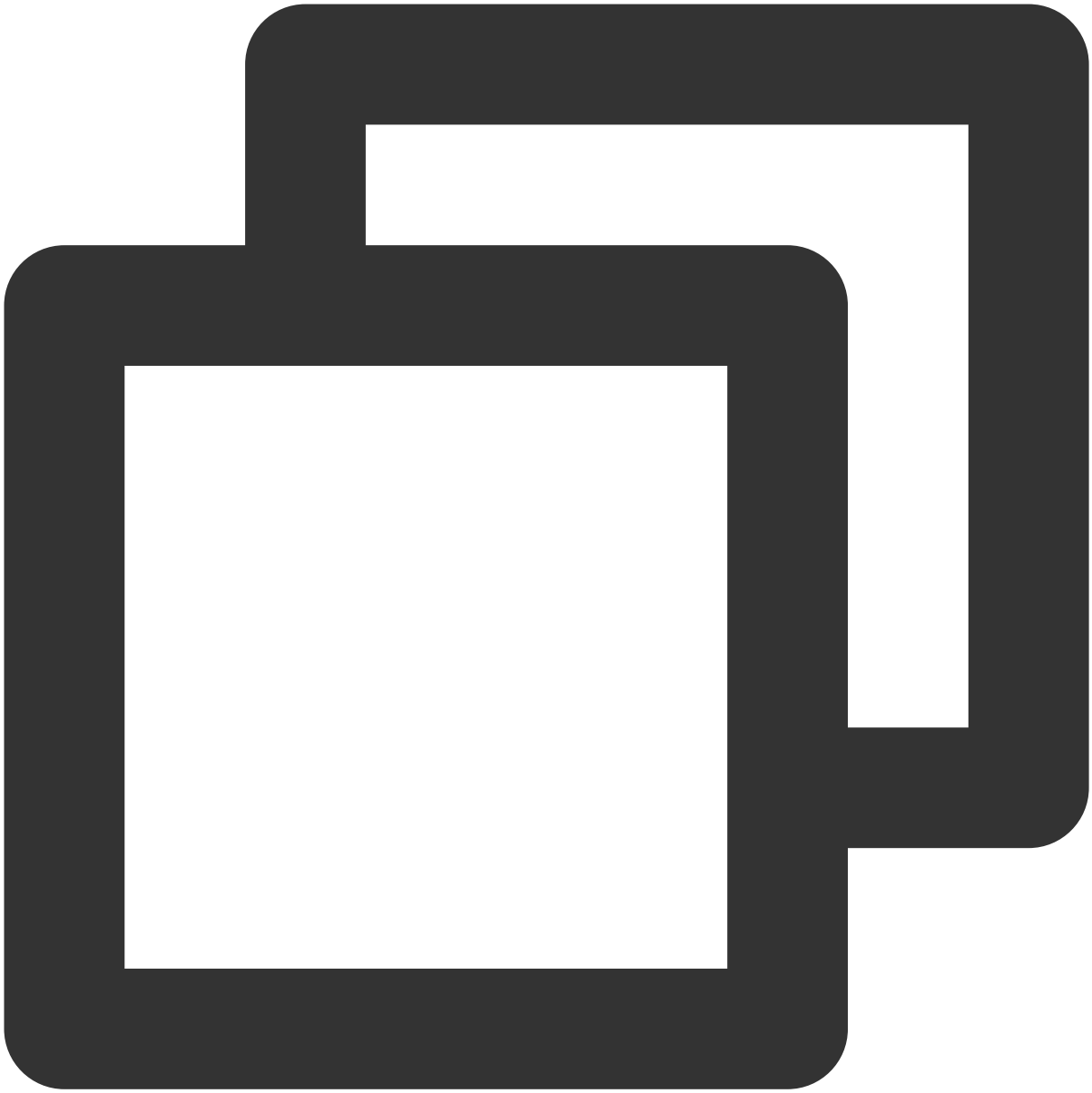
```
MIN(<col> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回col的最小值。

返回类型：与col一致

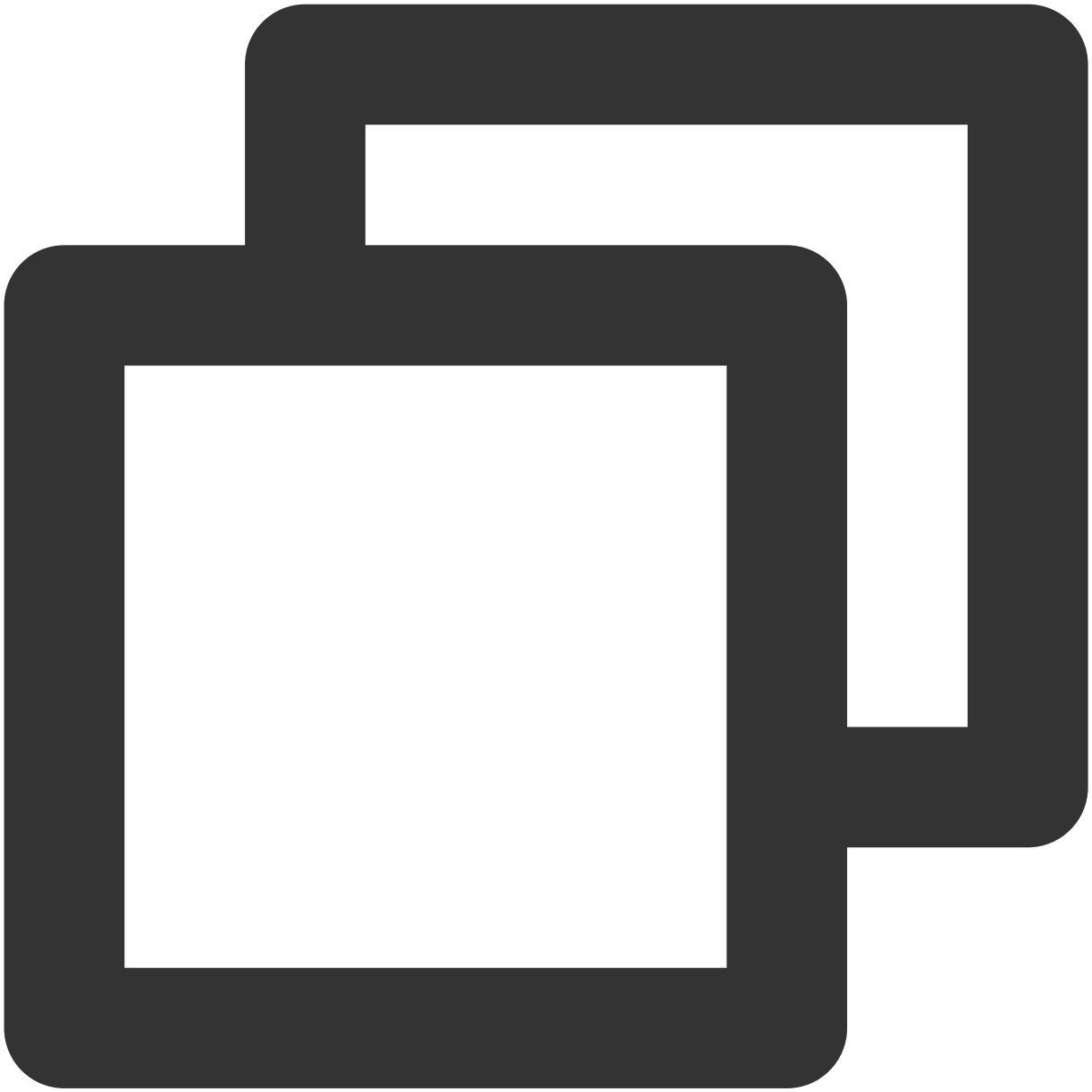
示例：



```
> SELECT min(col) FROM (VALUES (10), (50), (20)) AS tab(col);
 10
```

MIN_BY

函数语法：



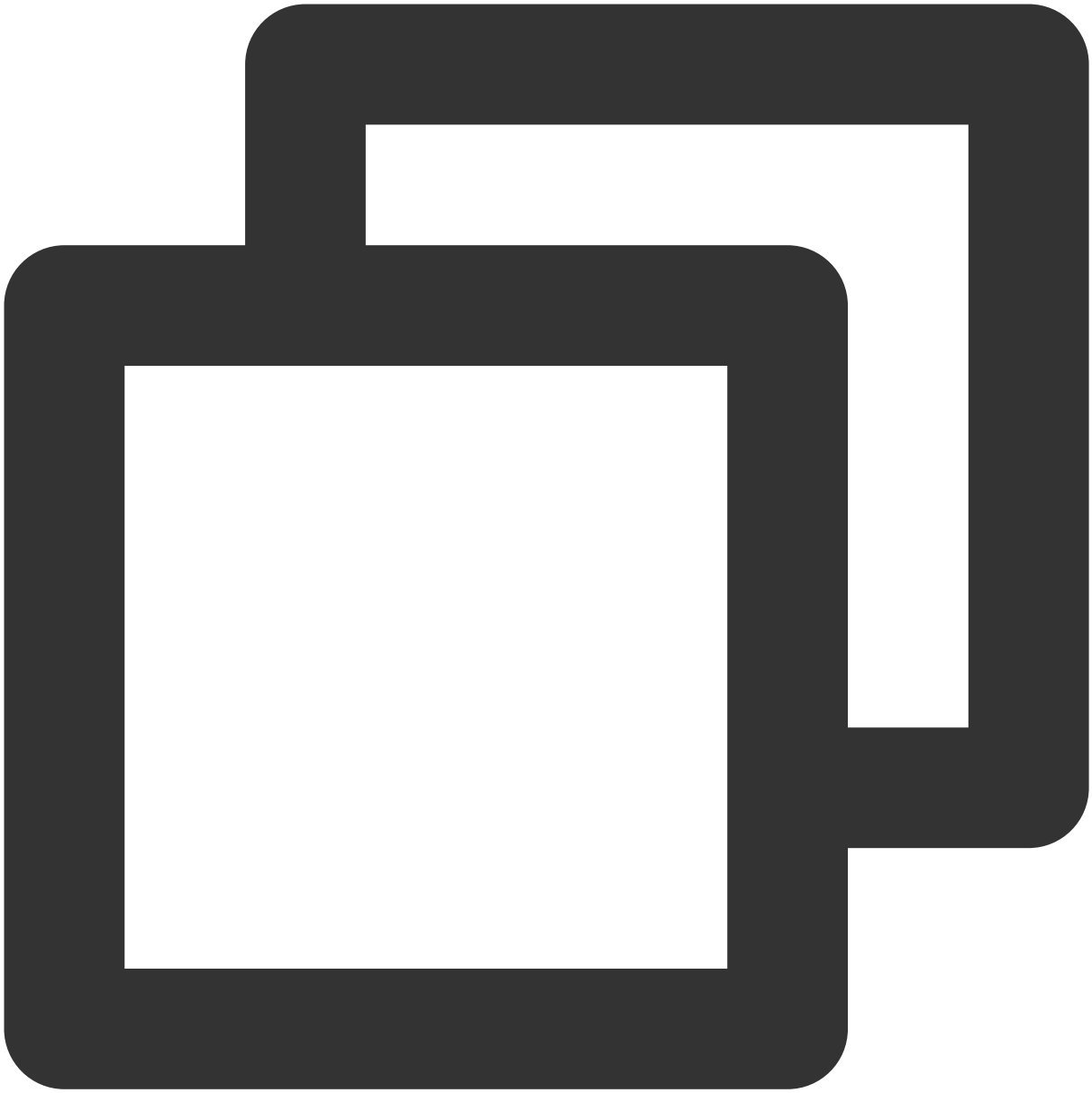
```
MIN_BY(<x> T, <y> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回与y的最小值关联的x值。

返回类型：T

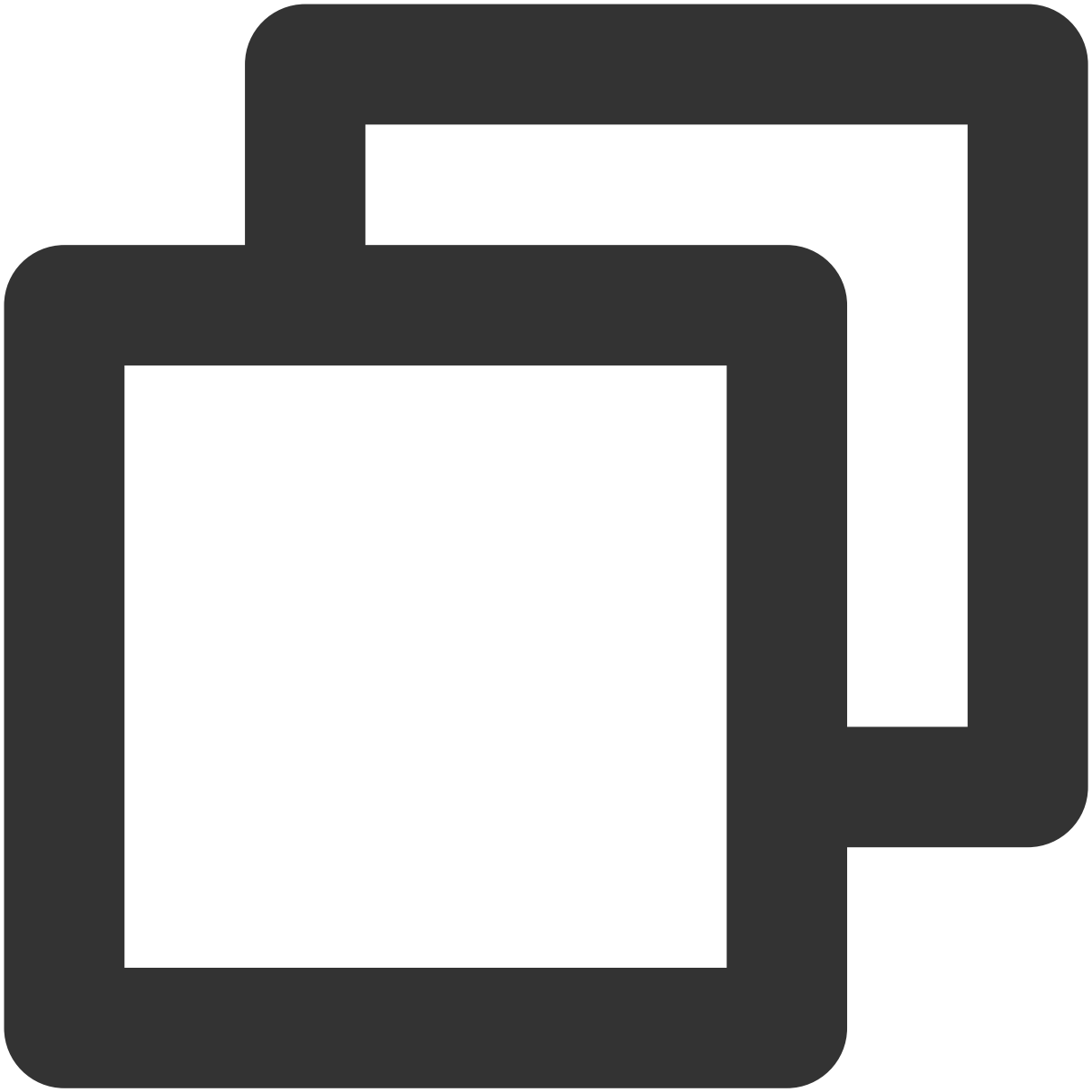
示例：



```
> SELECT min_by(x, y) FROM (VALUES (('a', 10)), (('b', 50)), (('c', 20))) AS tab(x,
 a
```

STD

函数语法：

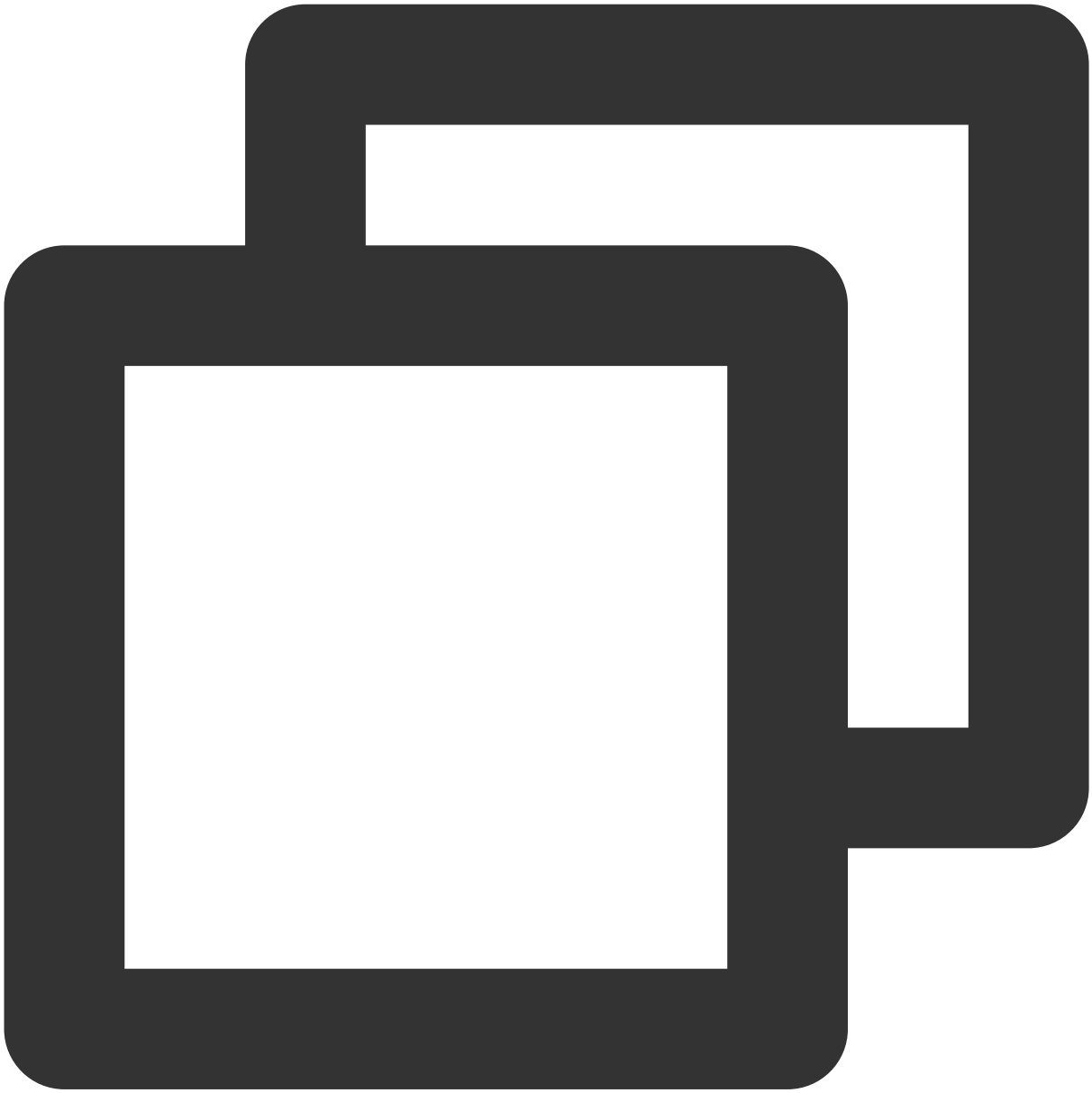


```
STD(<col> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明

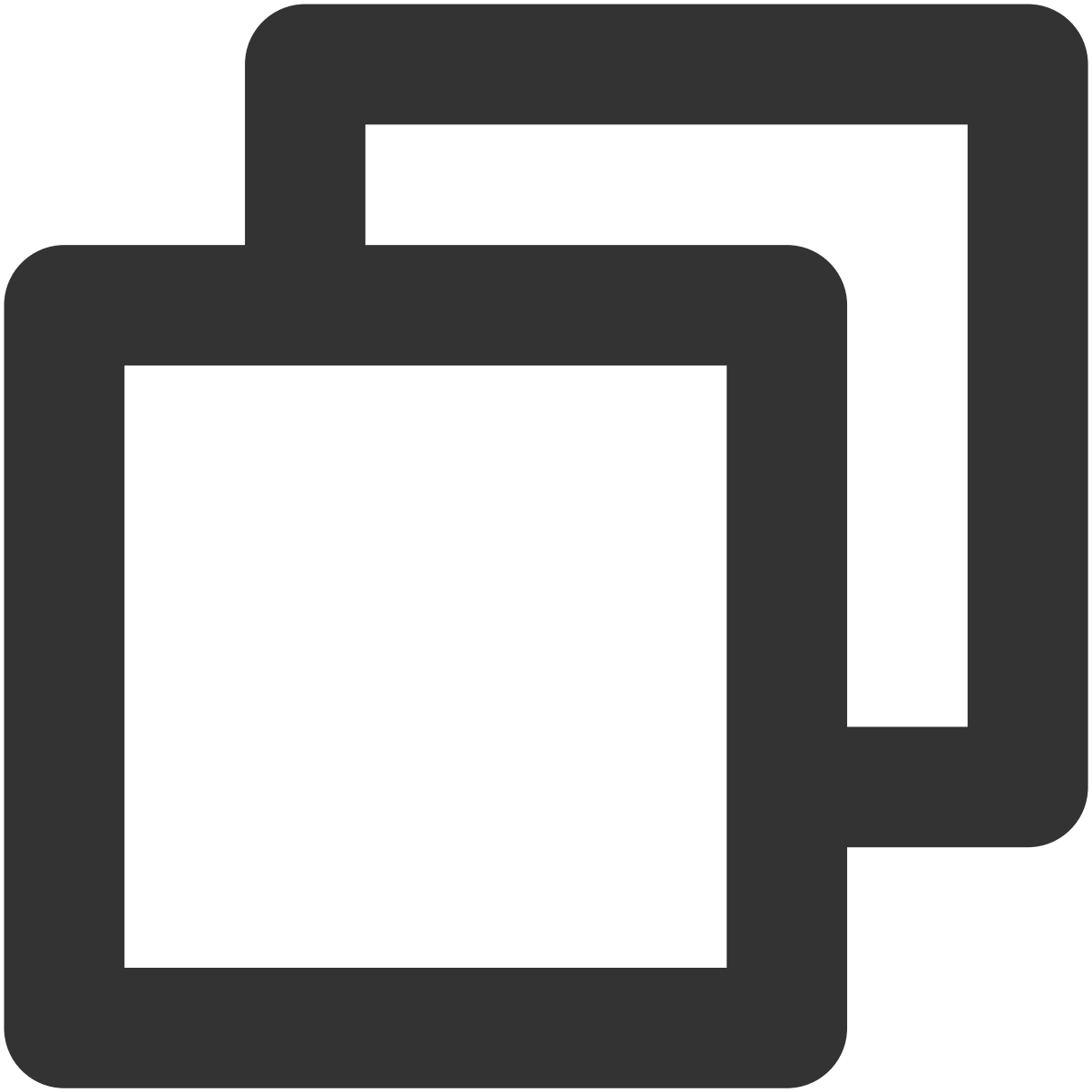
SparkSQL：返回根据组的值计算的样本标准偏差。
Presto：返回根据组的值计算的总体标准偏差。
返回类型：double
示例：



```
> SELECT std(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 1.0
```

STDDEV

函数语法：



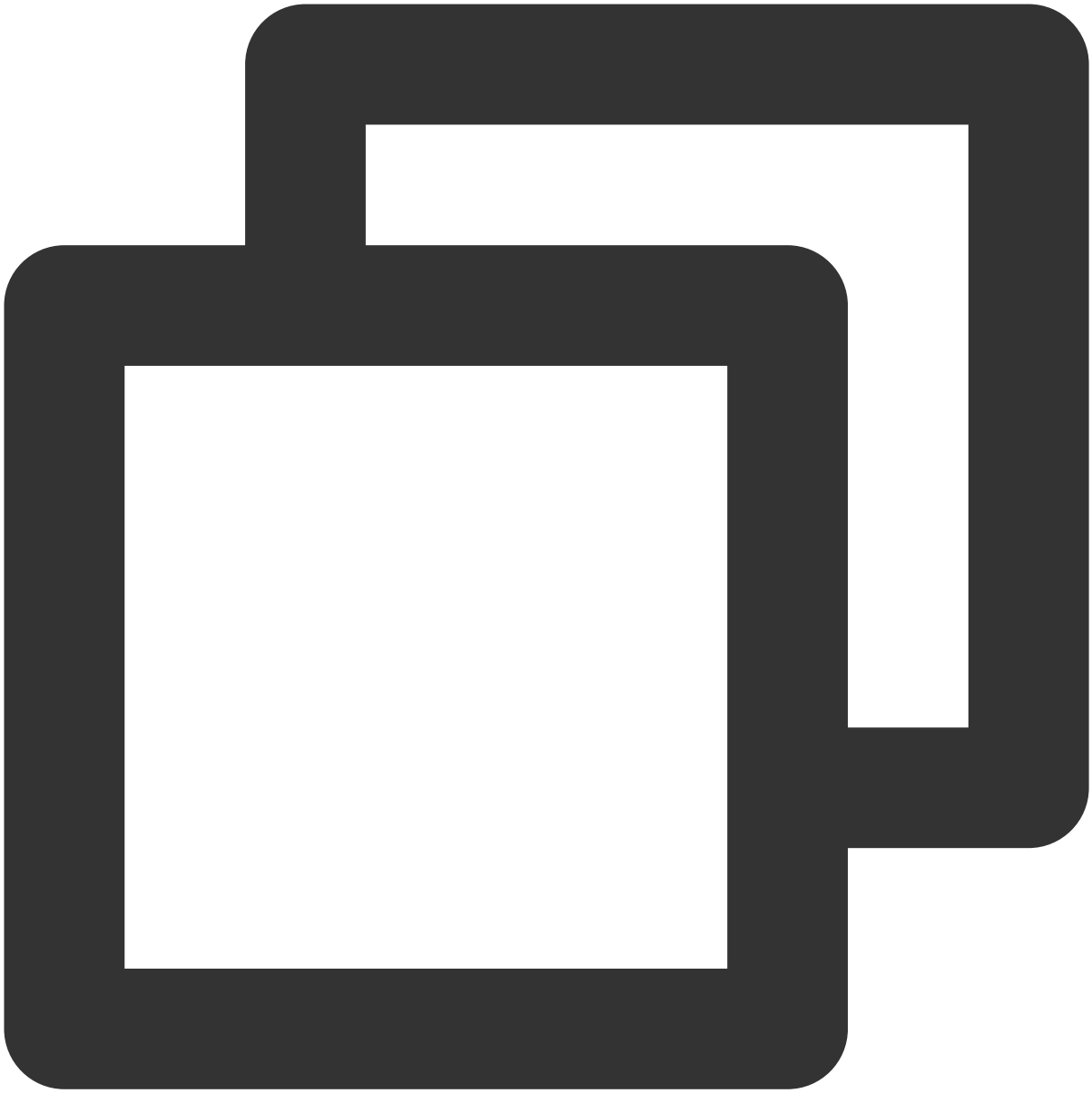
```
STDDEV(<col> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回根据组的值计算的样本标准偏差。

返回类型：double

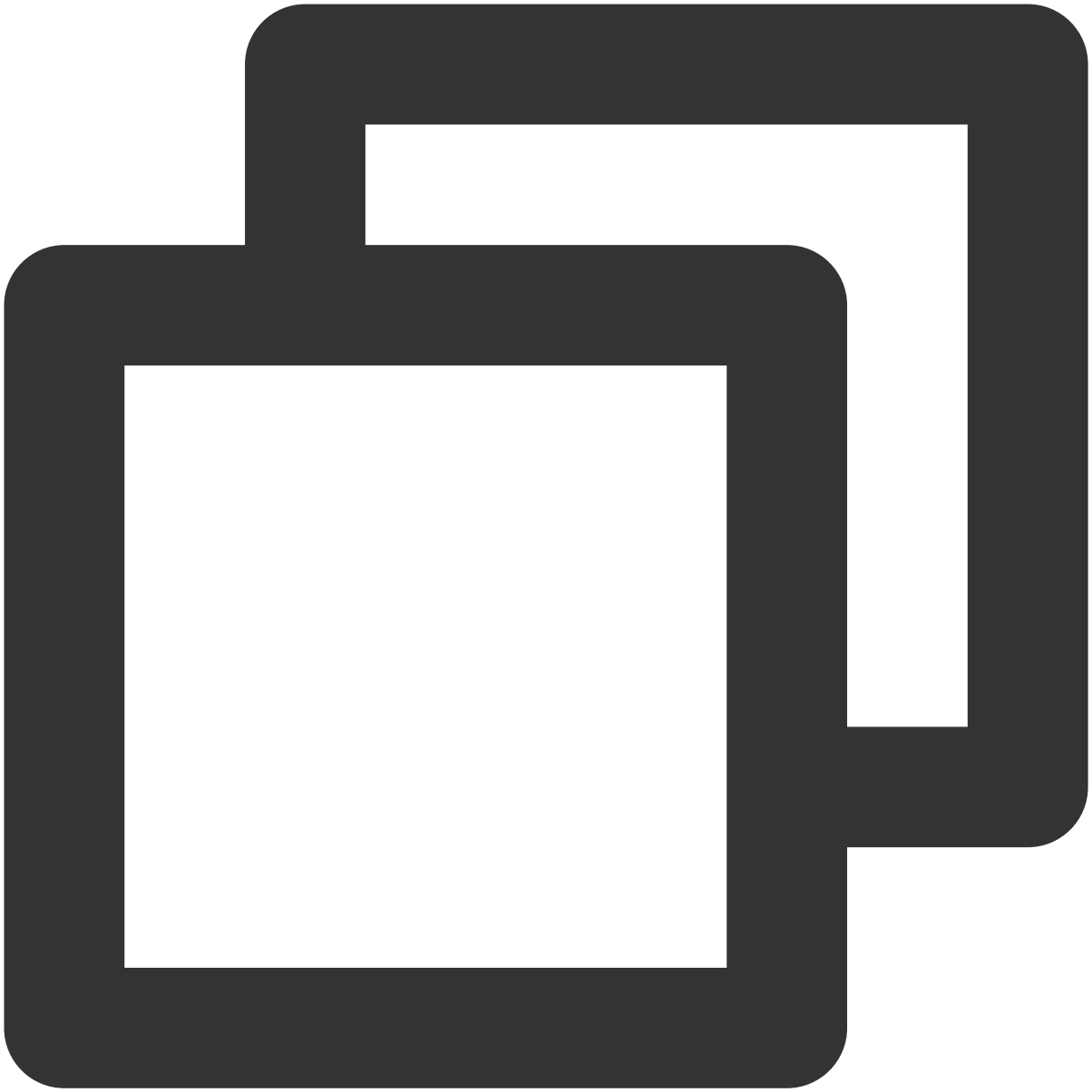
示例：



```
> SELECT stddev(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 1.0
```

STDDEV_POP

函数语法：



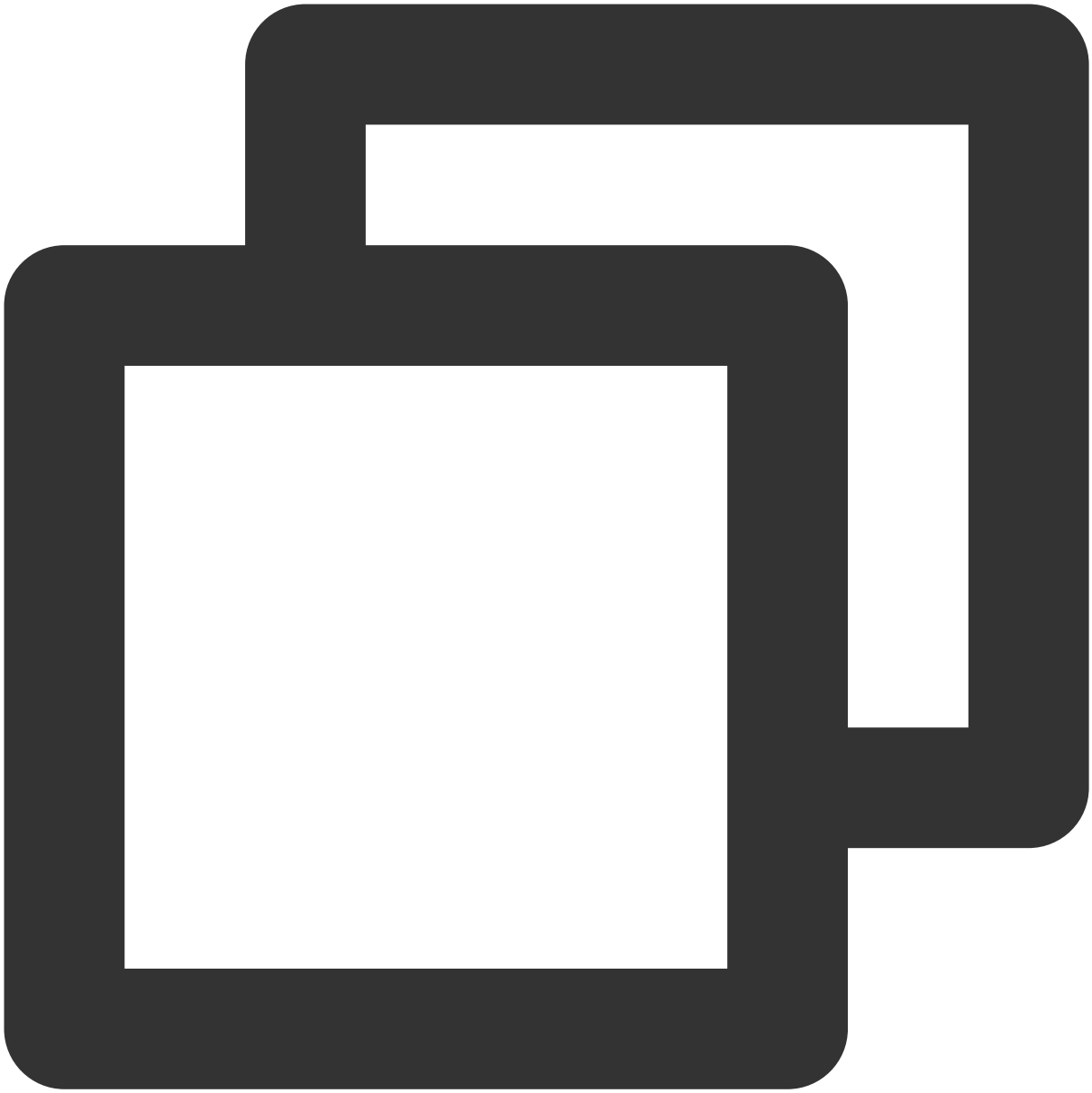
```
STDDEV_POP(<col> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回根据组的值计算的总体标准偏差。

返回类型：double

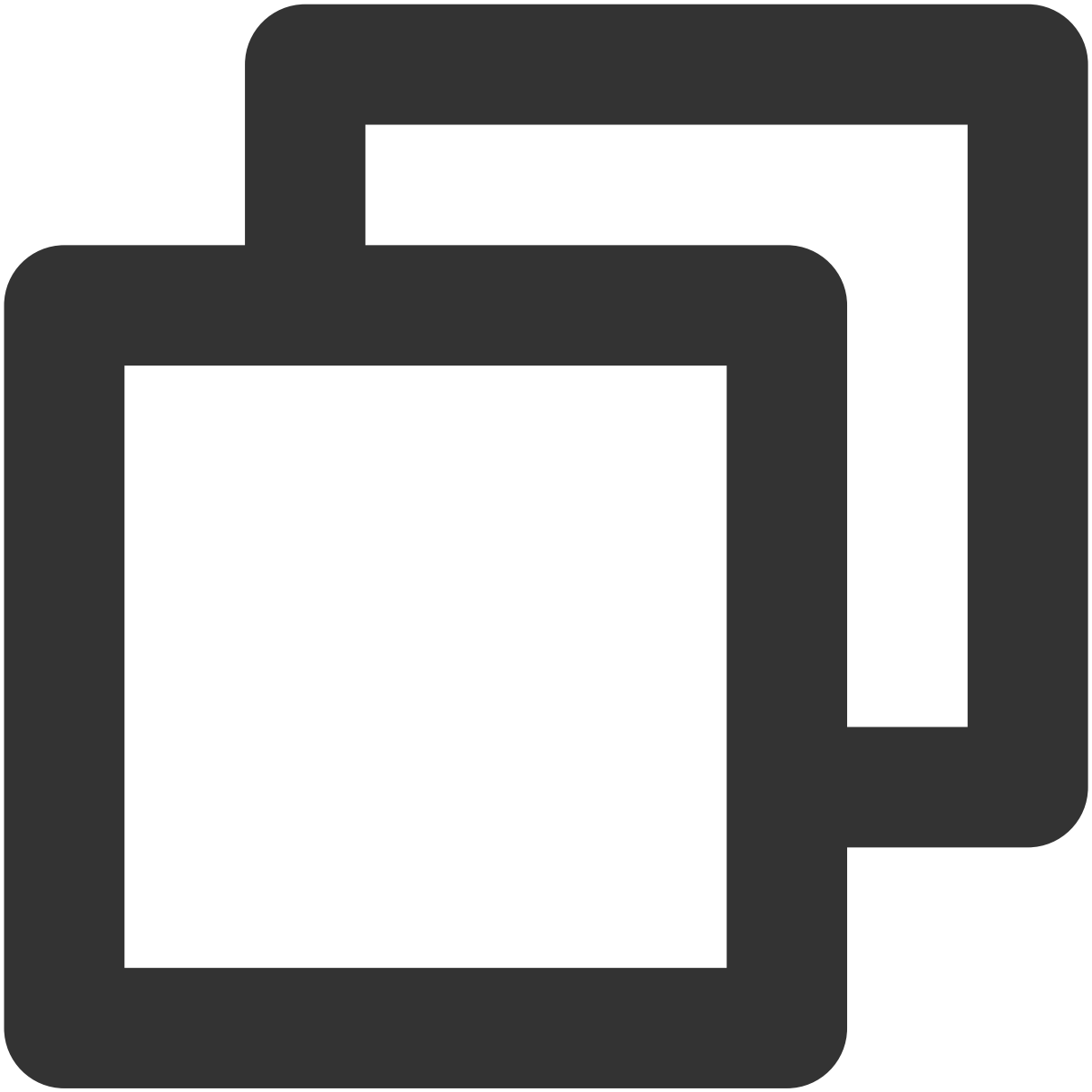
示例：



```
> SELECT stddev_pop(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 0.816496580927726
```

STDDEV_SAMP

函数语法：



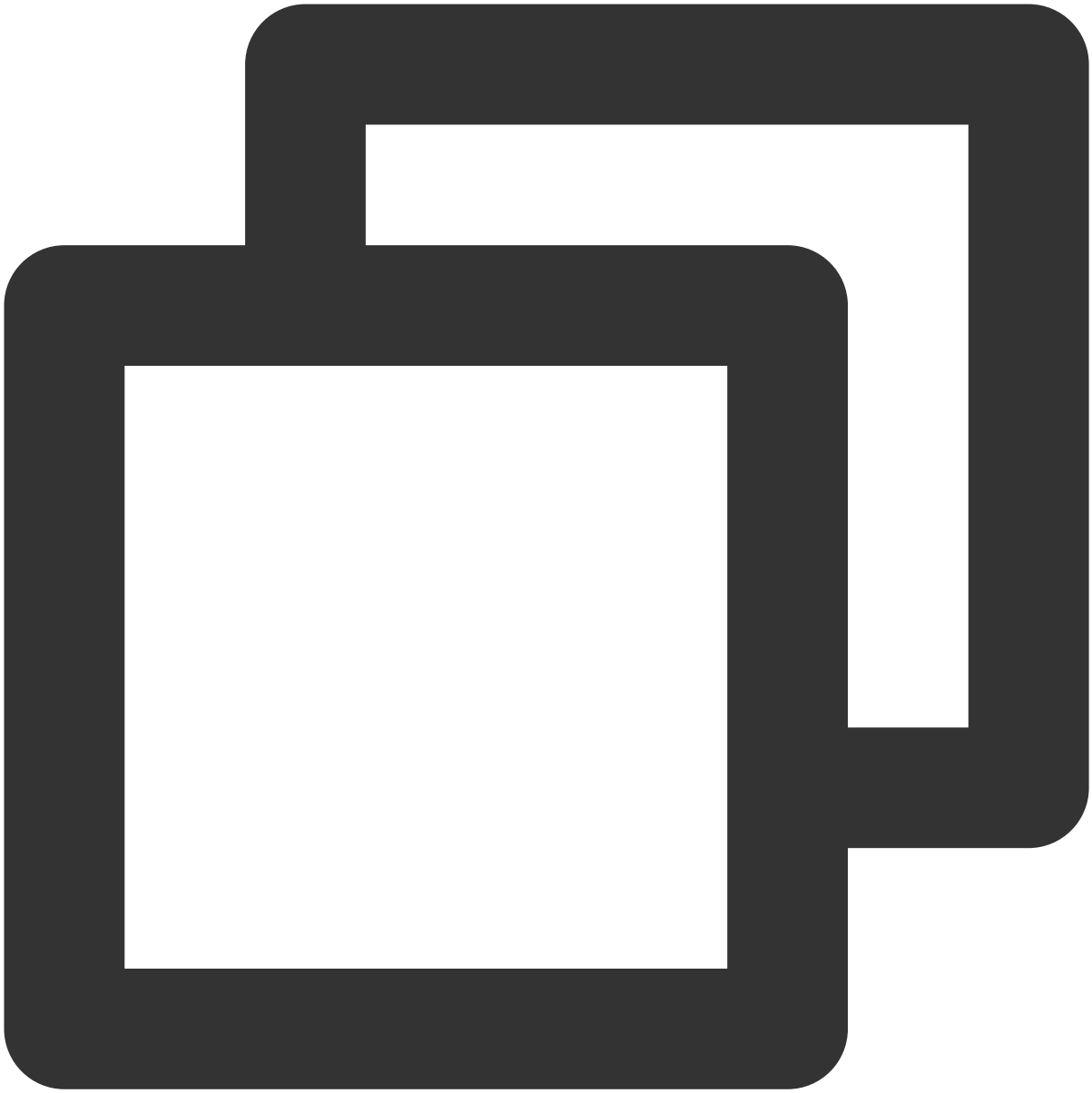
```
STDDEV_SAMP(<col> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回根据组的值计算的样本标准偏差。

返回类型：double

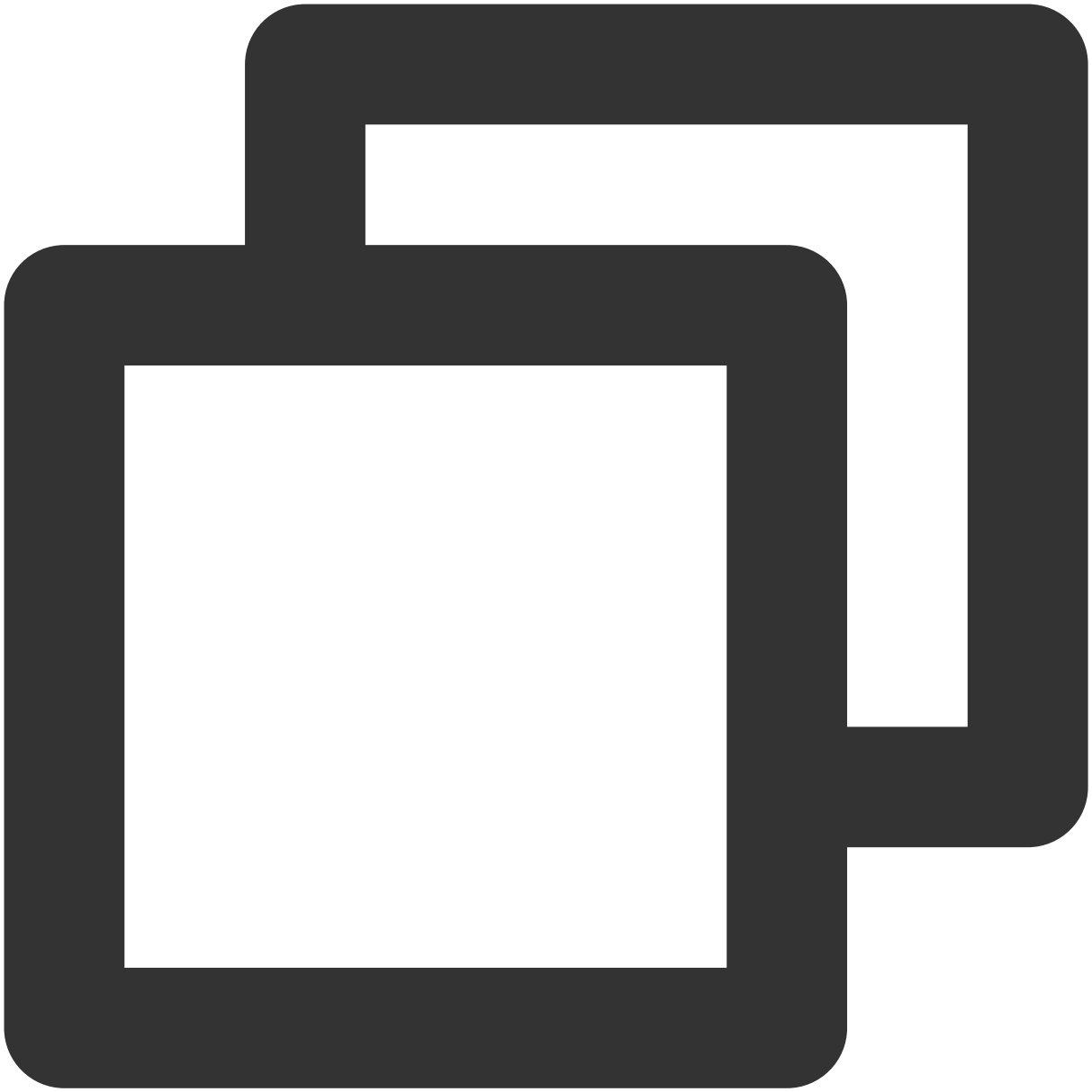
示例：



```
> SELECT stddev_samp(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 1.0
```

SUM

函数语法：



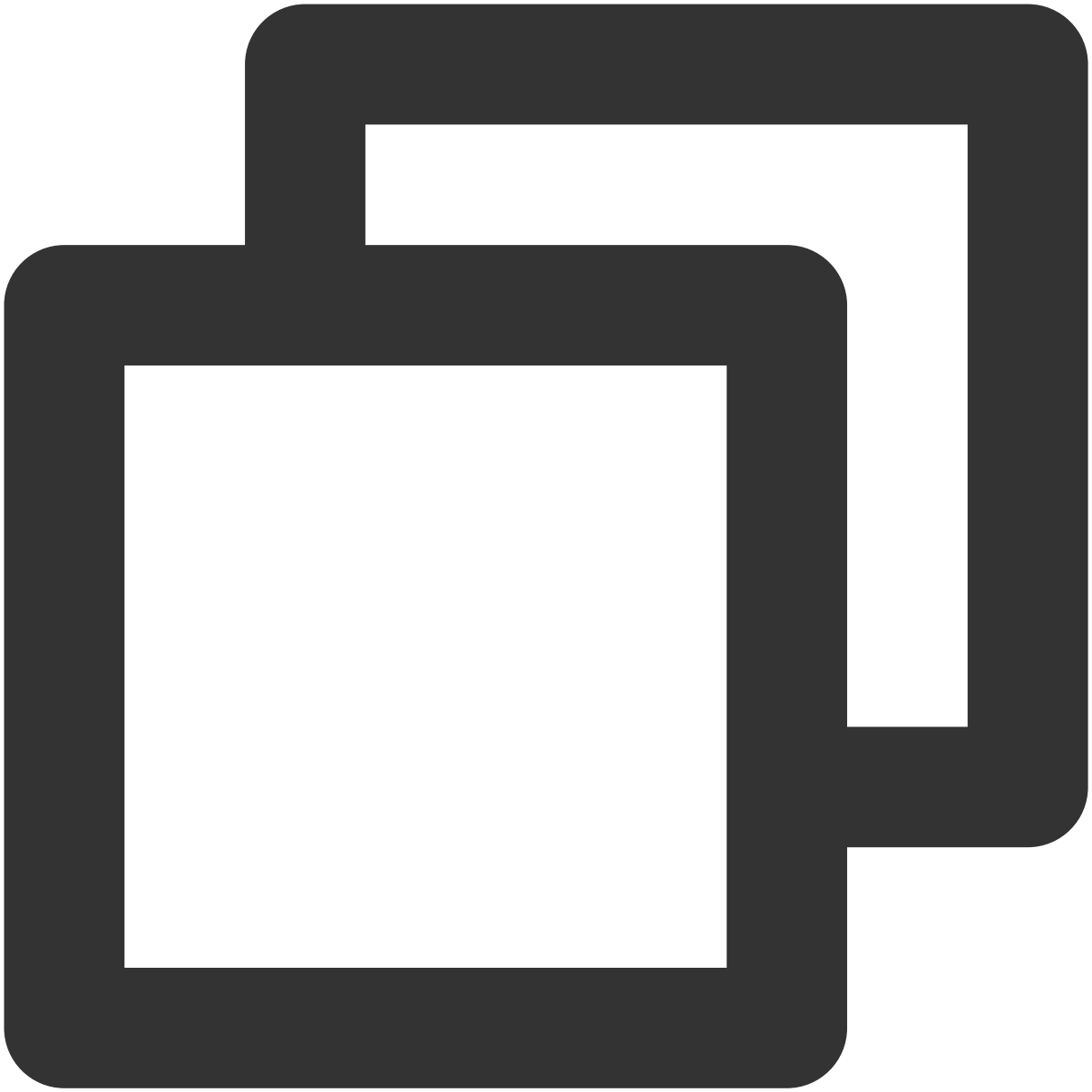
```
SUM(<col> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回根据组的值计算的总和。

返回类型：与col一致

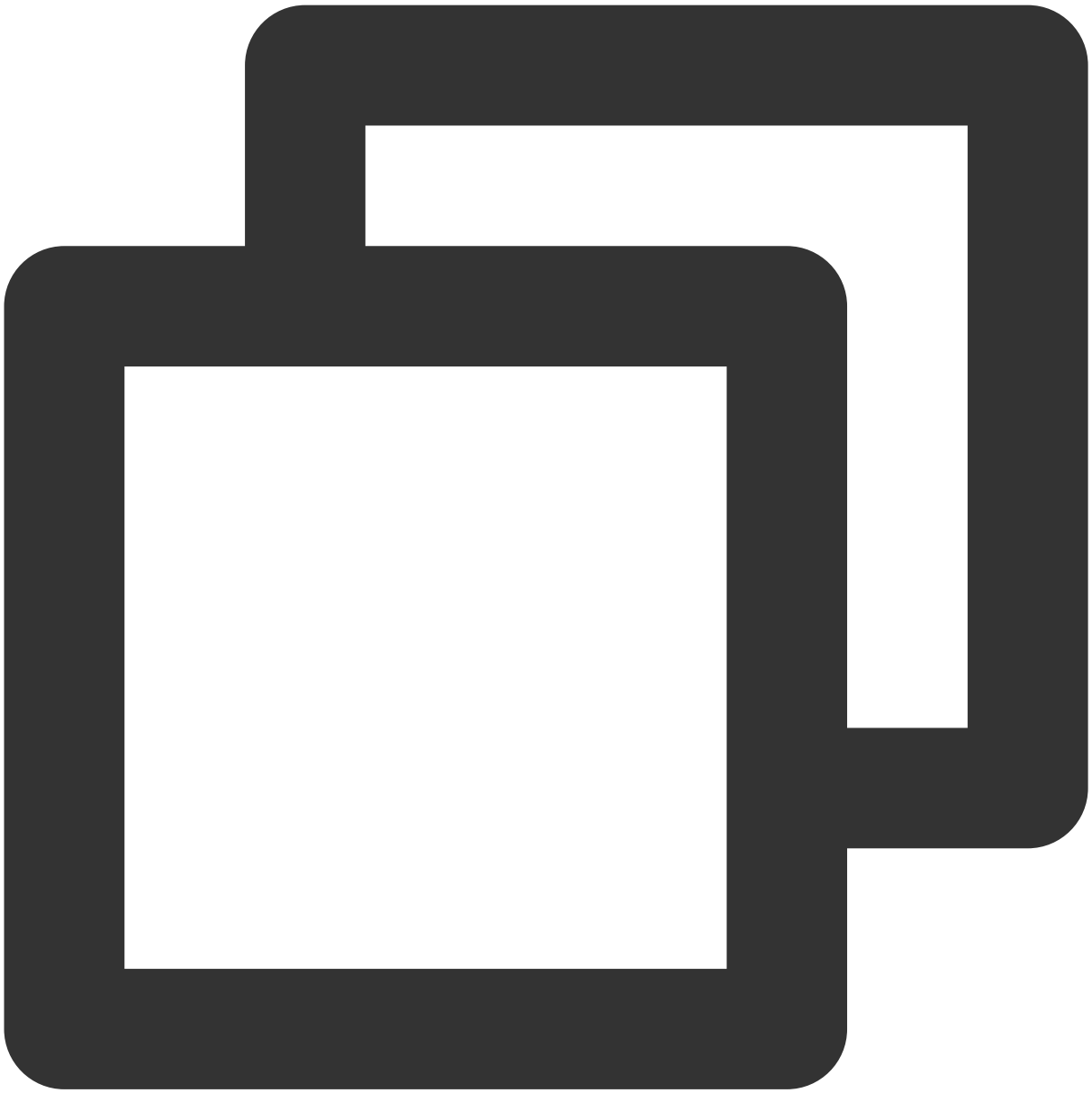
示例：



```
> SELECT sum(col) FROM (VALUES (5), (10), (15)) AS tab(col);
 30
> SELECT sum(col) FROM (VALUES (NULL), (10), (15)) AS tab(col);
 25
> SELECT sum(col) FROM (VALUES (NULL), (NULL)) AS tab(col);
 NULL
```

VARIANCE

函数语法：



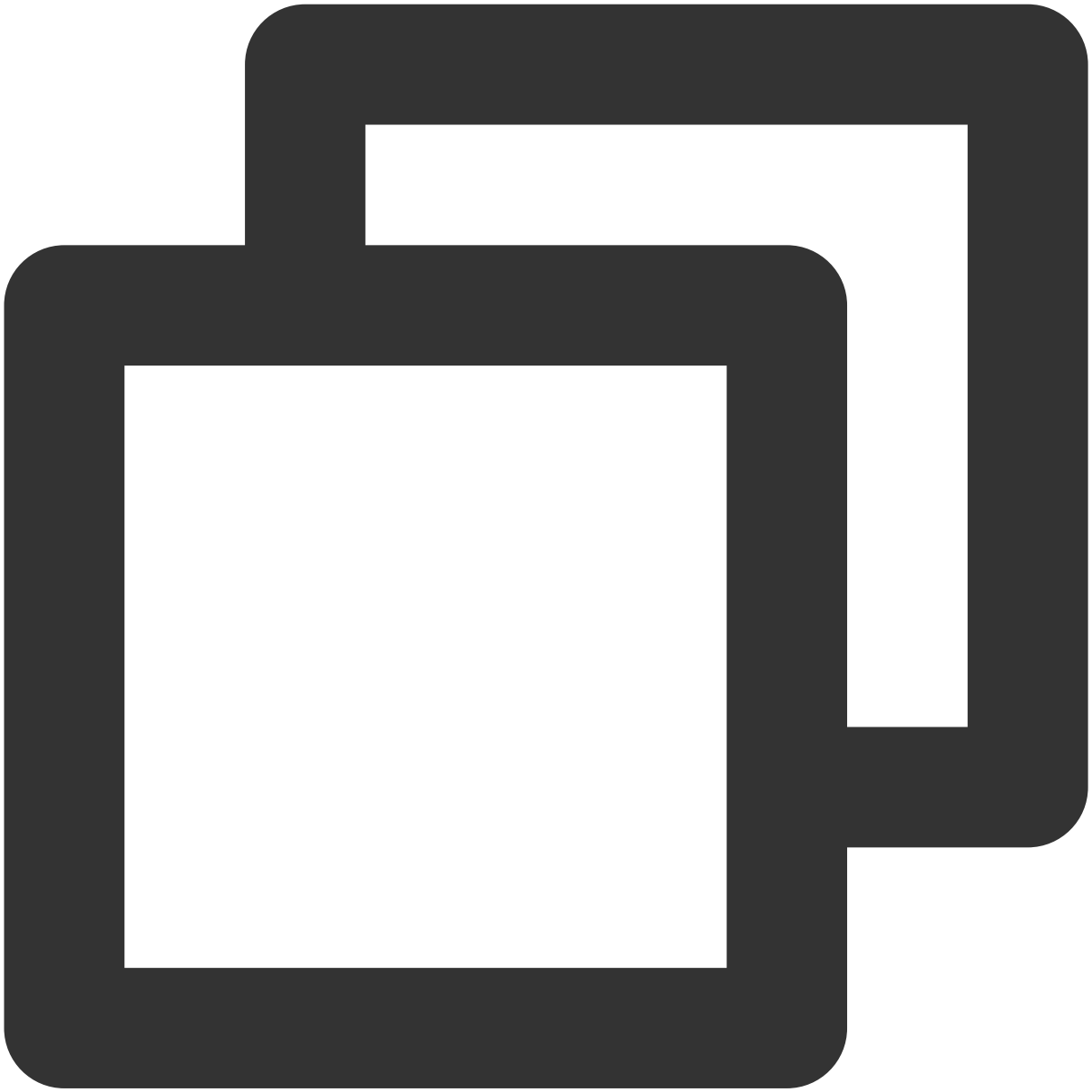
```
VARIANCE(<col> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回根据组的值计算的样本方差。

返回类型：double

示例：



```
> SELECT VARIANCE(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 1.0
```

VAR_POP

函数语法：

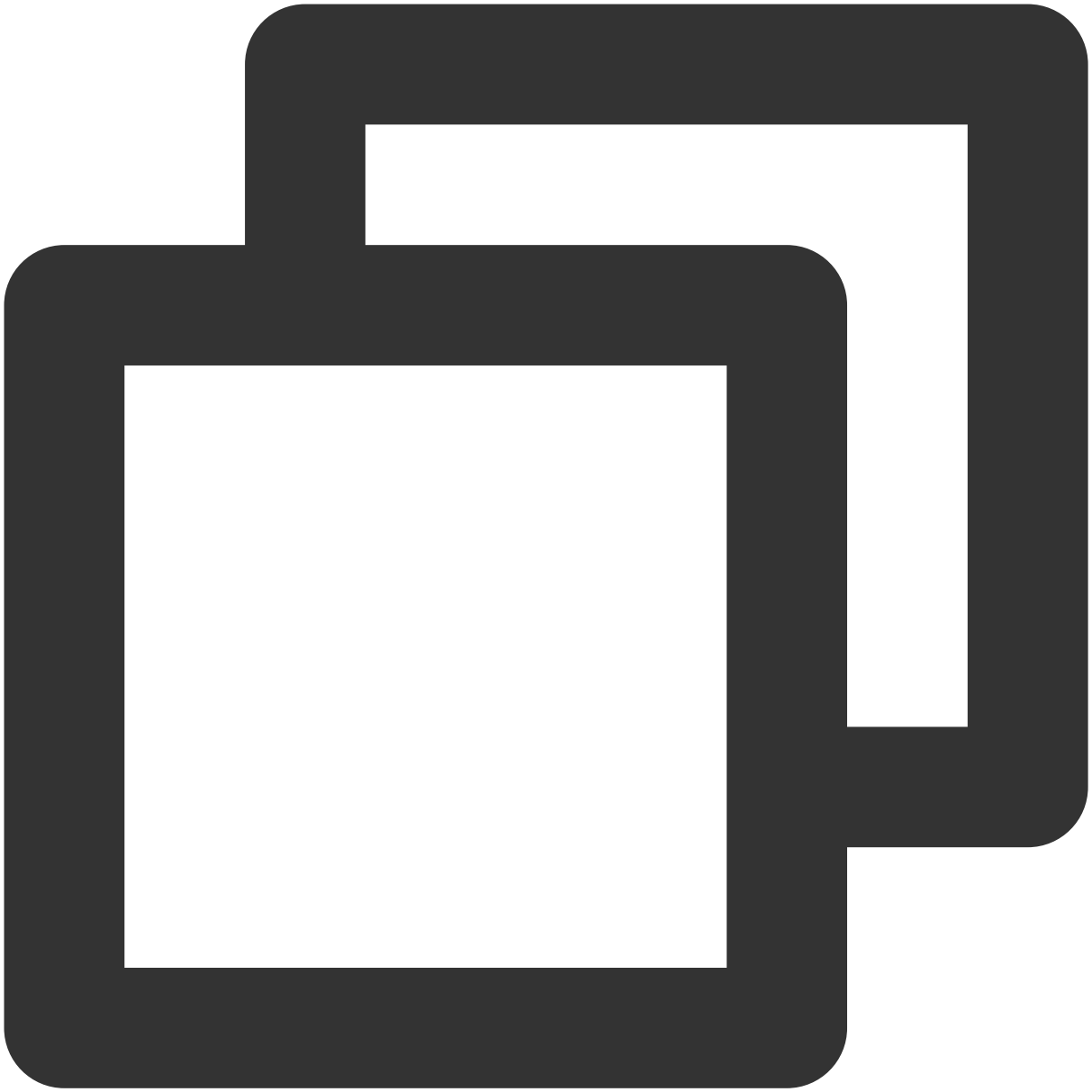
```
VAR_POP(<col> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回根据组的值计算的总体方差。

返回类型：double

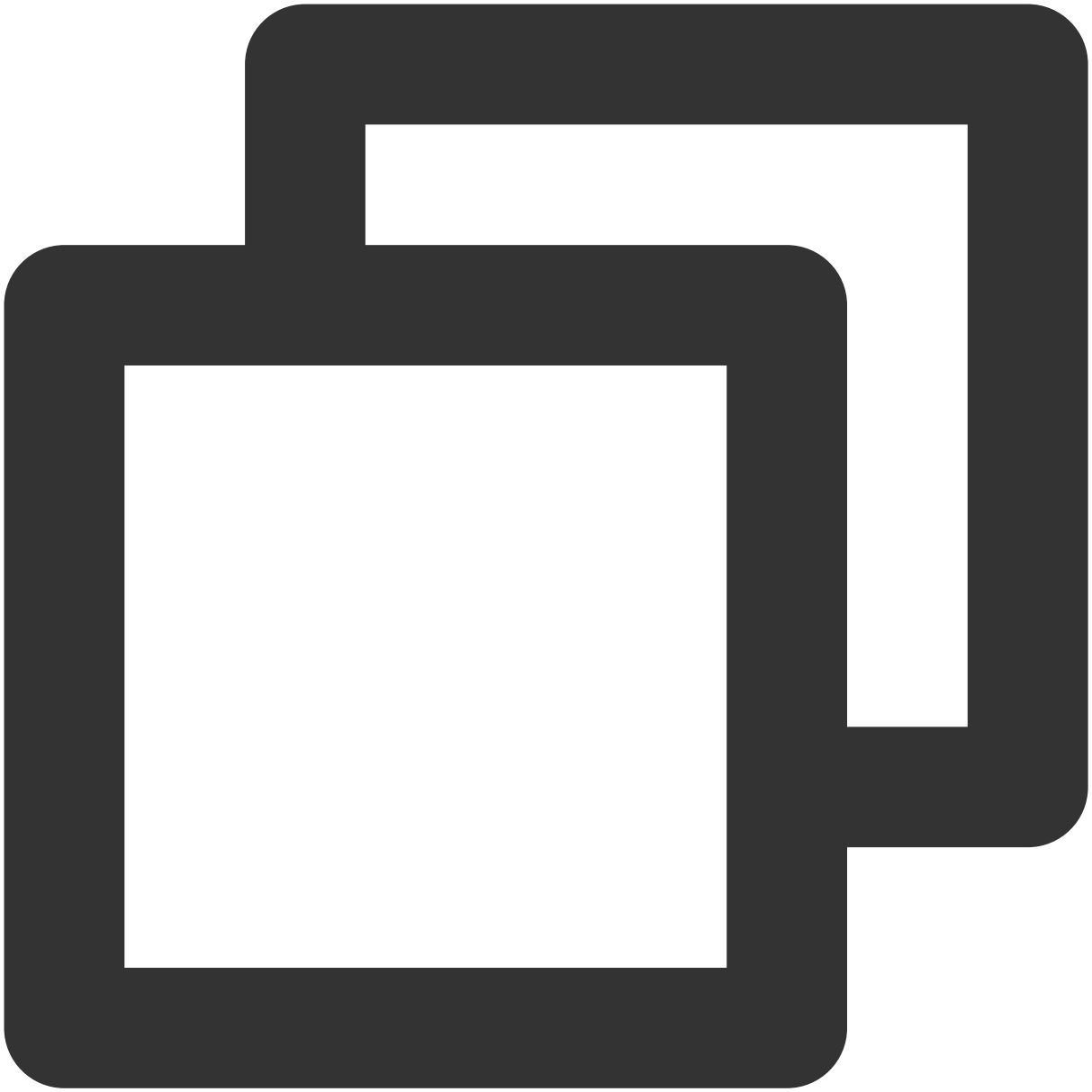
示例：



```
> SELECT var_pop(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 0.6666666666666666
```

VAR_SAMP

函数语法：



```
VAR_SAMP(<col> integer|double|decimal)
```

支持引擎：SparkSQL、Presto

使用说明：返回根据组的值计算的样本方差。

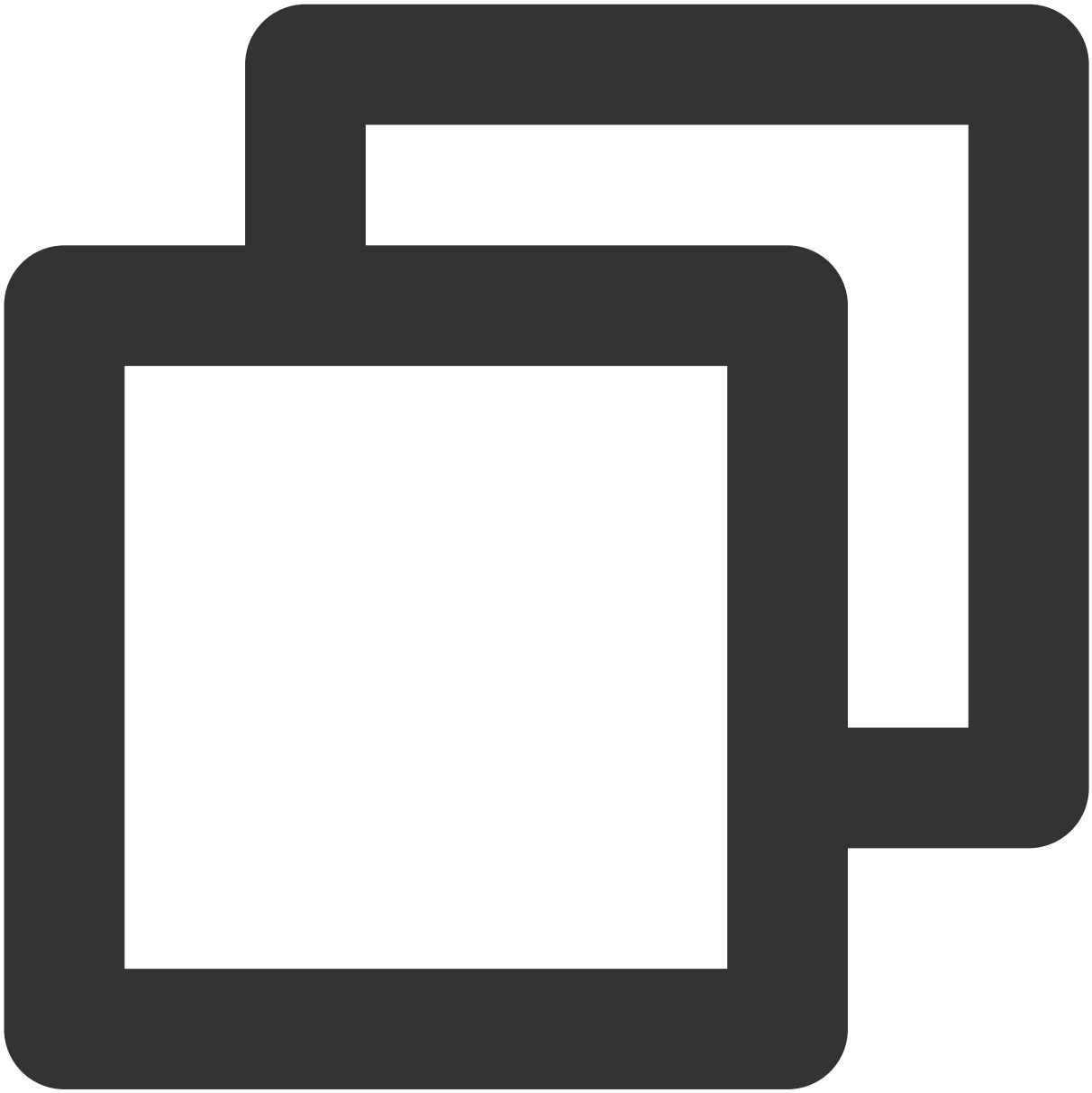
返回类型：double

示例：

```
> SELECT var_samp(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 1.0
```

HISTOGRAM_NUMERIC

函数语法：



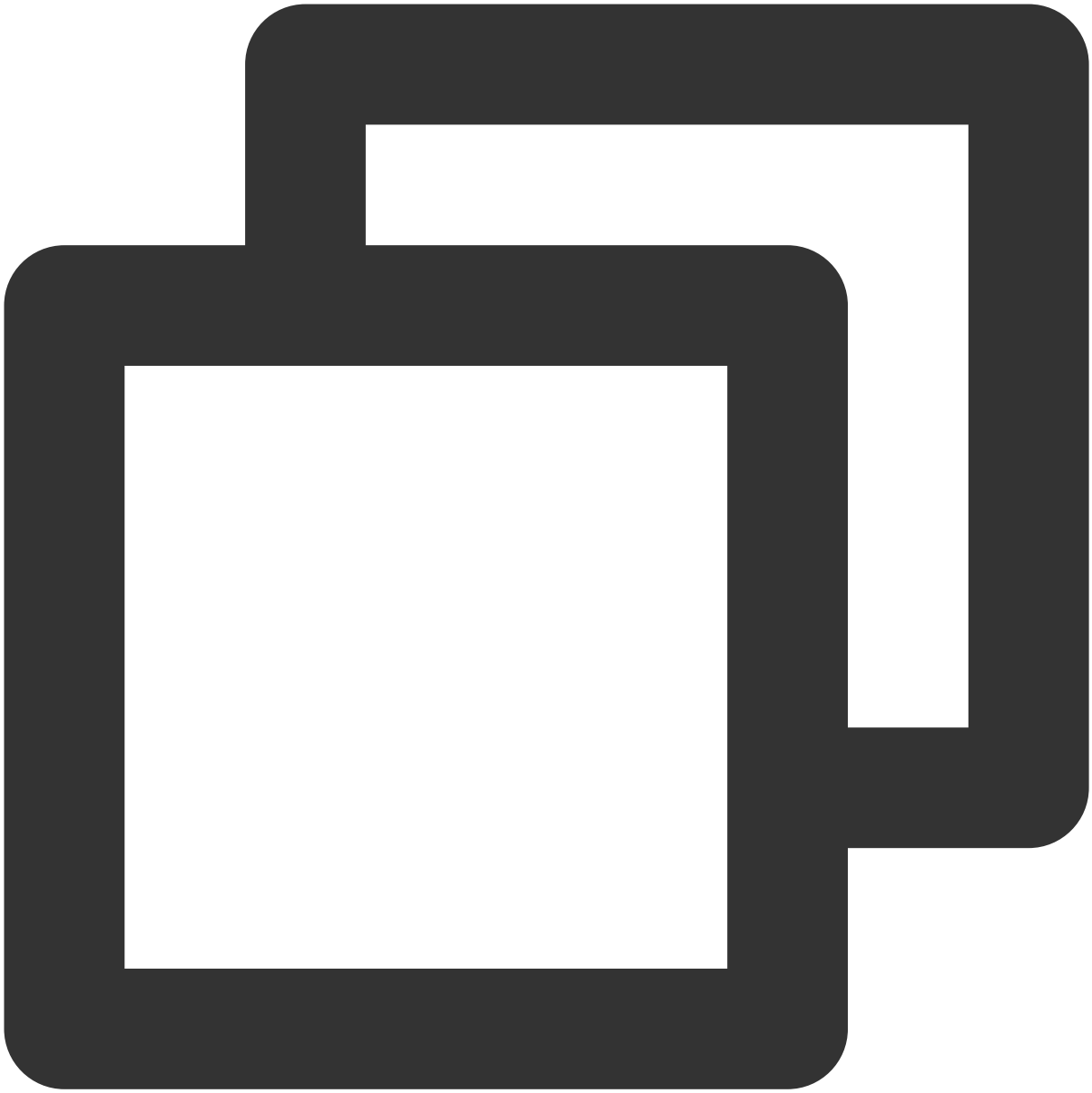
```
HISTOGRAM_NUMERIC(<col> integer, <nb> integer)
```

支持引擎：Presto

使用说明：使用nb个非均匀间隔的bin计算组中数字列的直方图。输出是一个大小为nb的（x，y）坐标数组，表示bin的中心和高度。

返回类型：array<struct {'x','y'}>

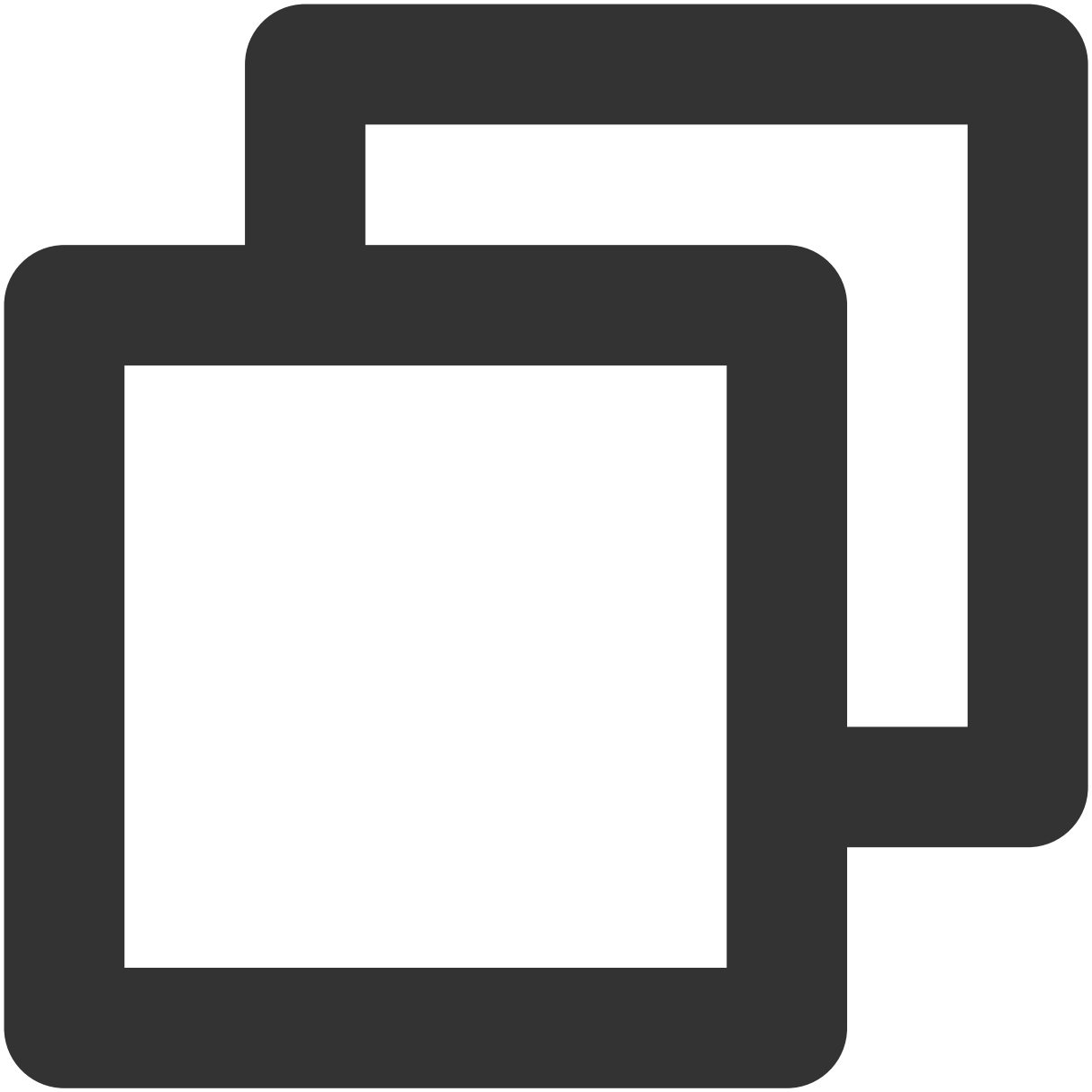
示例：



```
> SELECT histogram_numeric(col, 5) FROM (VALUES (0), (1), (2), (10)) AS tab(col);
 [{"x":0,"y":1.0},{"x":1,"y":1.0},{"x":2,"y":1.0},{"x":10,"y":1.0}]
```

COLLECT_LIST

函数语法：



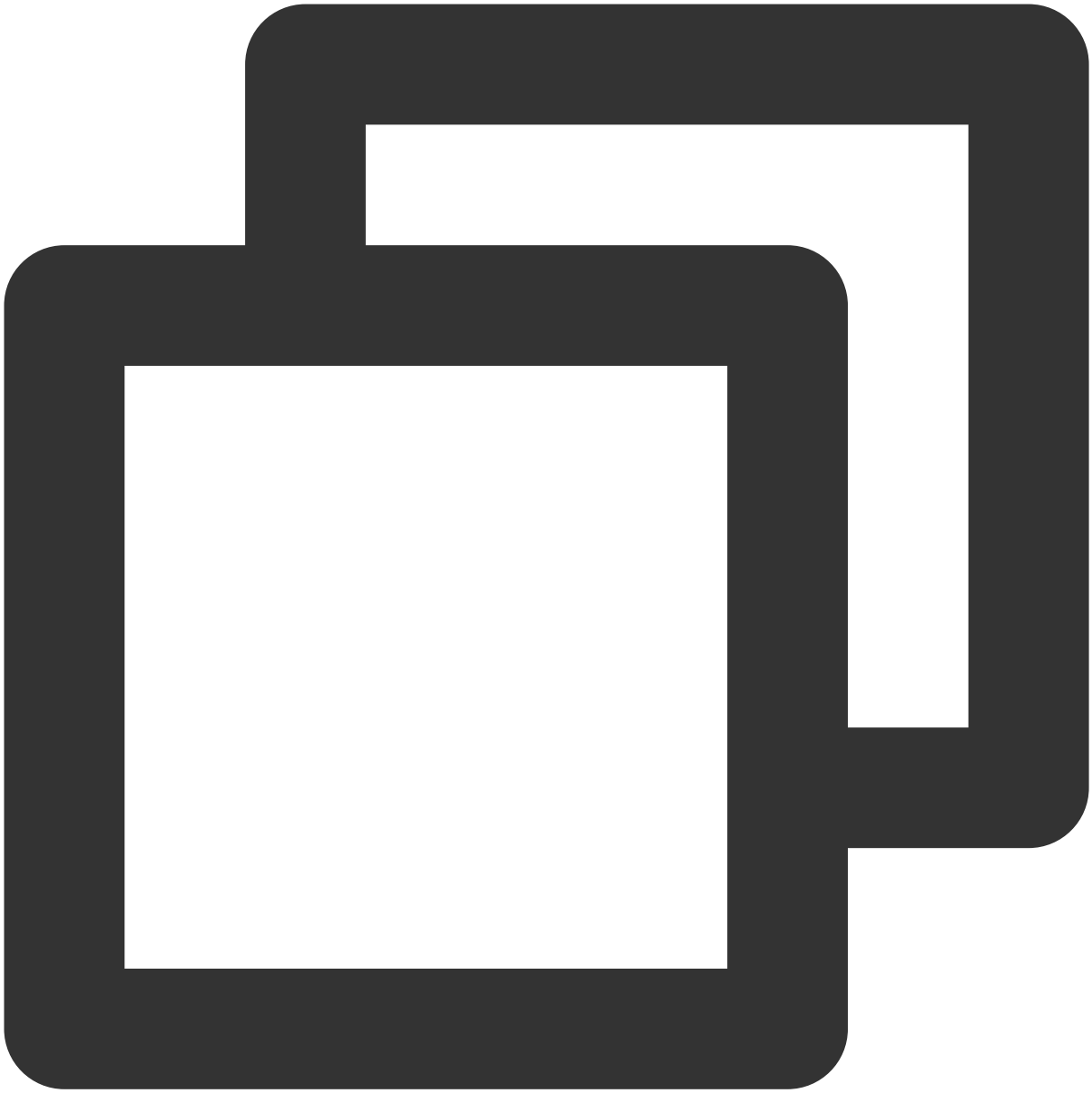
```
COLLECT_LIST(<col> T)
```

支持引擎：SparkSQL、Presto

使用说明：收集并返回非唯一元素的列表。

返回类型：array<T>

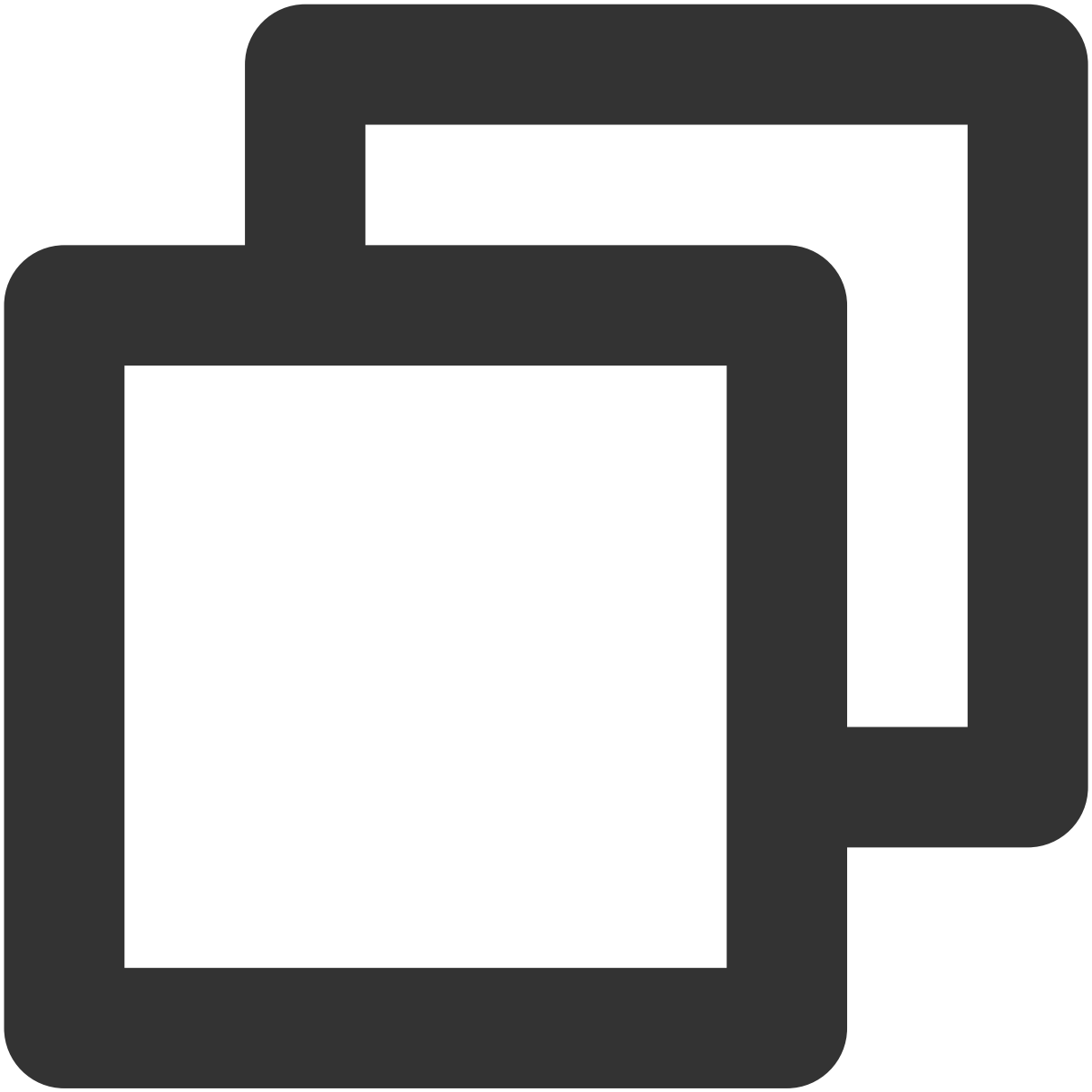
示例：



```
> SELECT collect_list(col) FROM (VALUES (1), (2), (1)) AS tab(col);
 [1,2,1]
```

COLLECT_SET

函数语法：



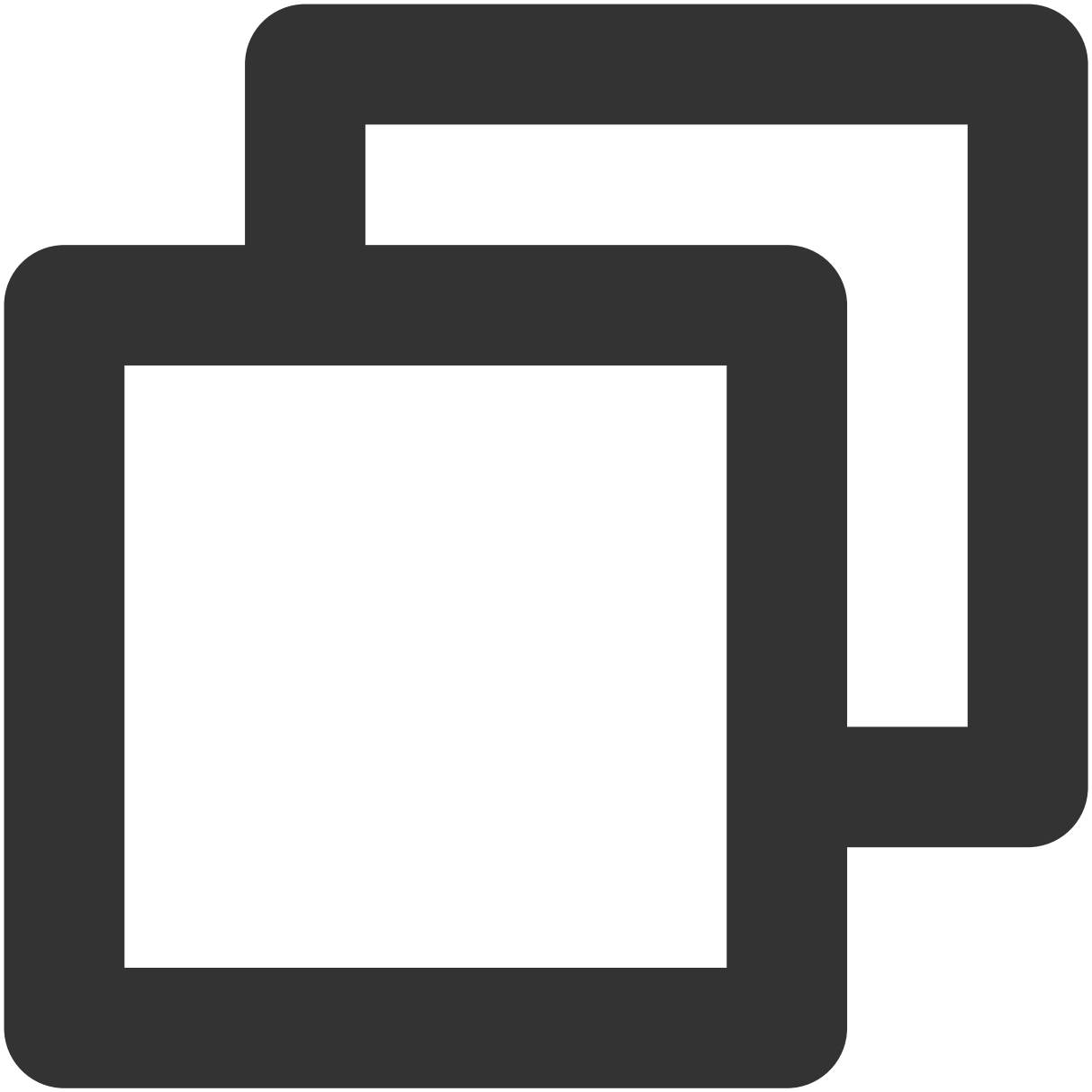
```
COLLECT_SET(<col> T)
```

支持引擎：SparkSQL、Presto

使用说明：收集并返回一组唯一的元素。

返回类型：array<T>

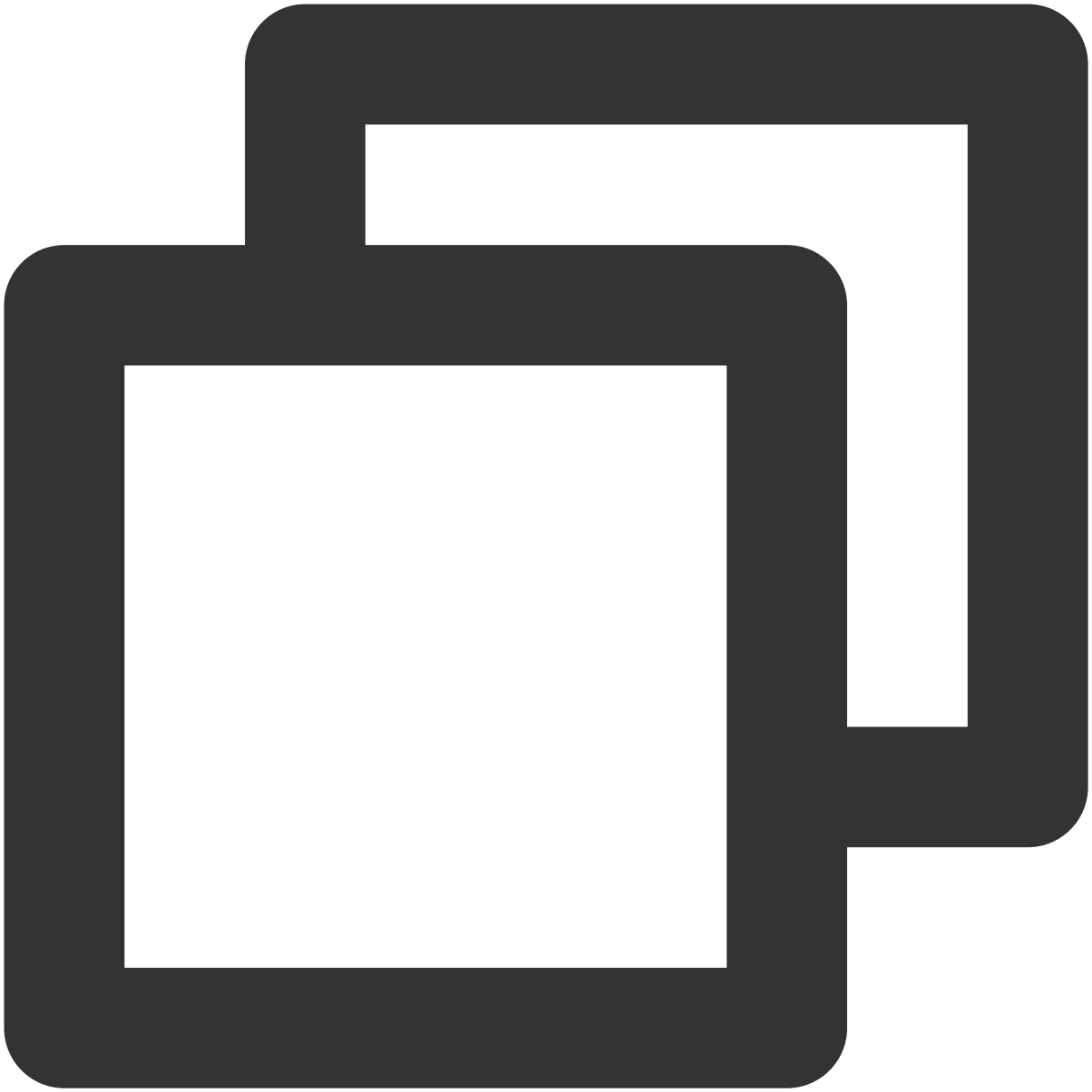
示例：



```
> SELECT collect_set(col) FROM (VALUES (1), (2), (1)) AS tab(col);
 [1,2]
```

COUNT_MIN_SKETCH

函数语法：



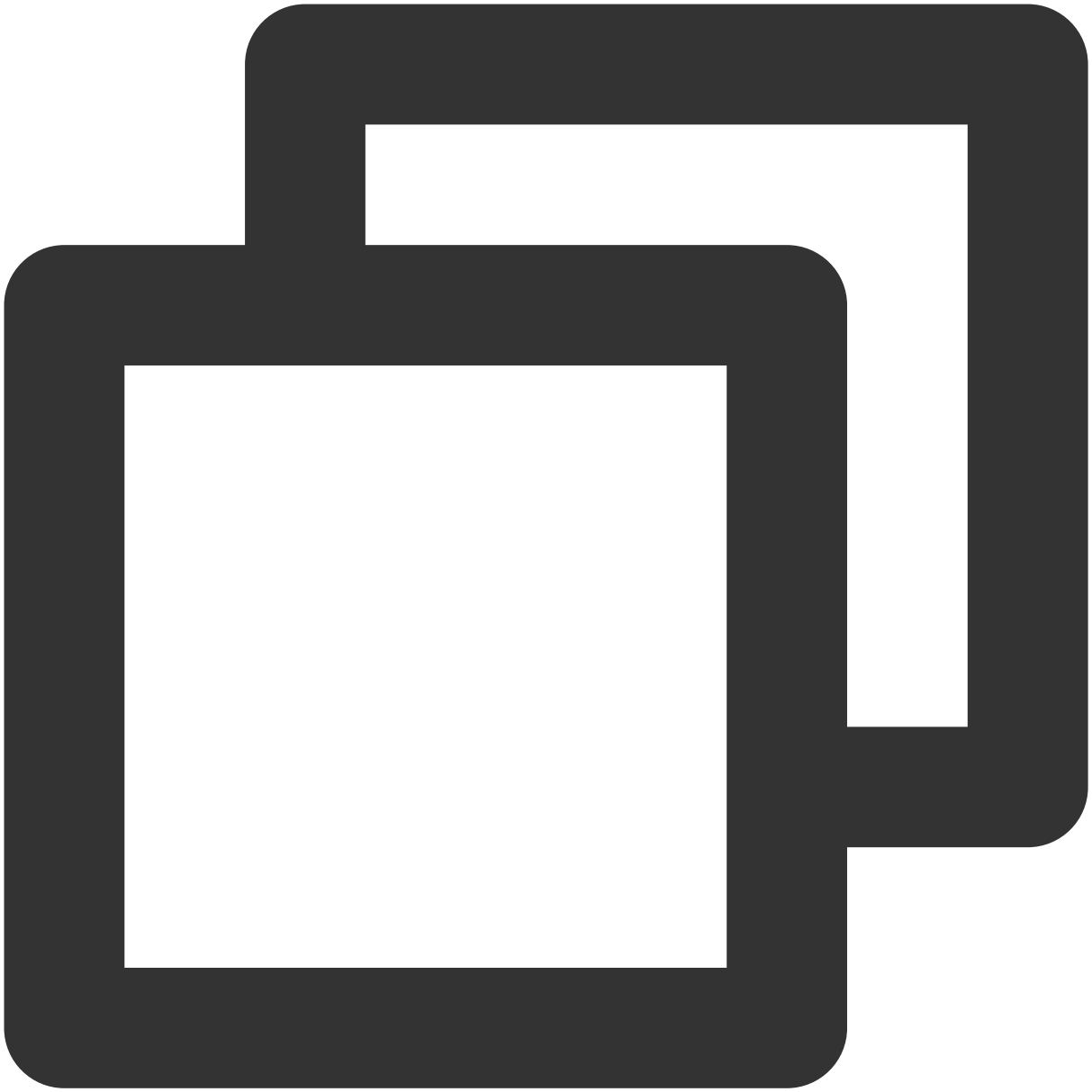
```
COUNT_MIN_SKETCH(<col> T, <eps> double, <confidence> double, <seed> integer)
```

支持引擎：SparkSQL

使用说明：返回具有给定esp、confidence和seed的列的count min sketch。结果是binary，可以在使用前反序列化为CountMinSketch

返回类型：binary

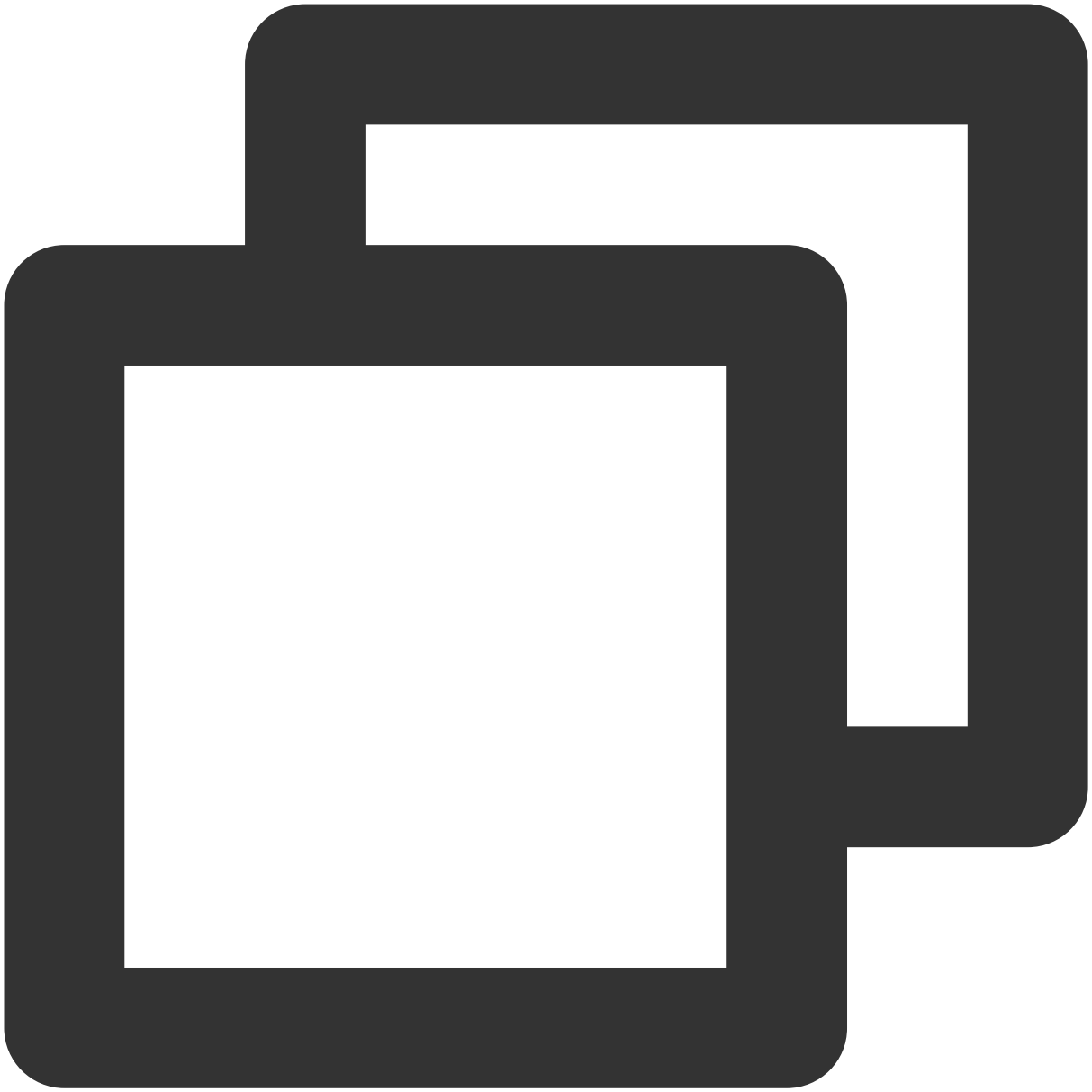
示例：



```
> SELECT hex(count_min_sketch(col, 0.5d, 0.5d, 1)) FROM (VALUES (1), (2), (1)) AS t
0000000100000000000000030000000100000004000000005D8D6AB90000000000000000000000000000
```

EVERY

函数语法：



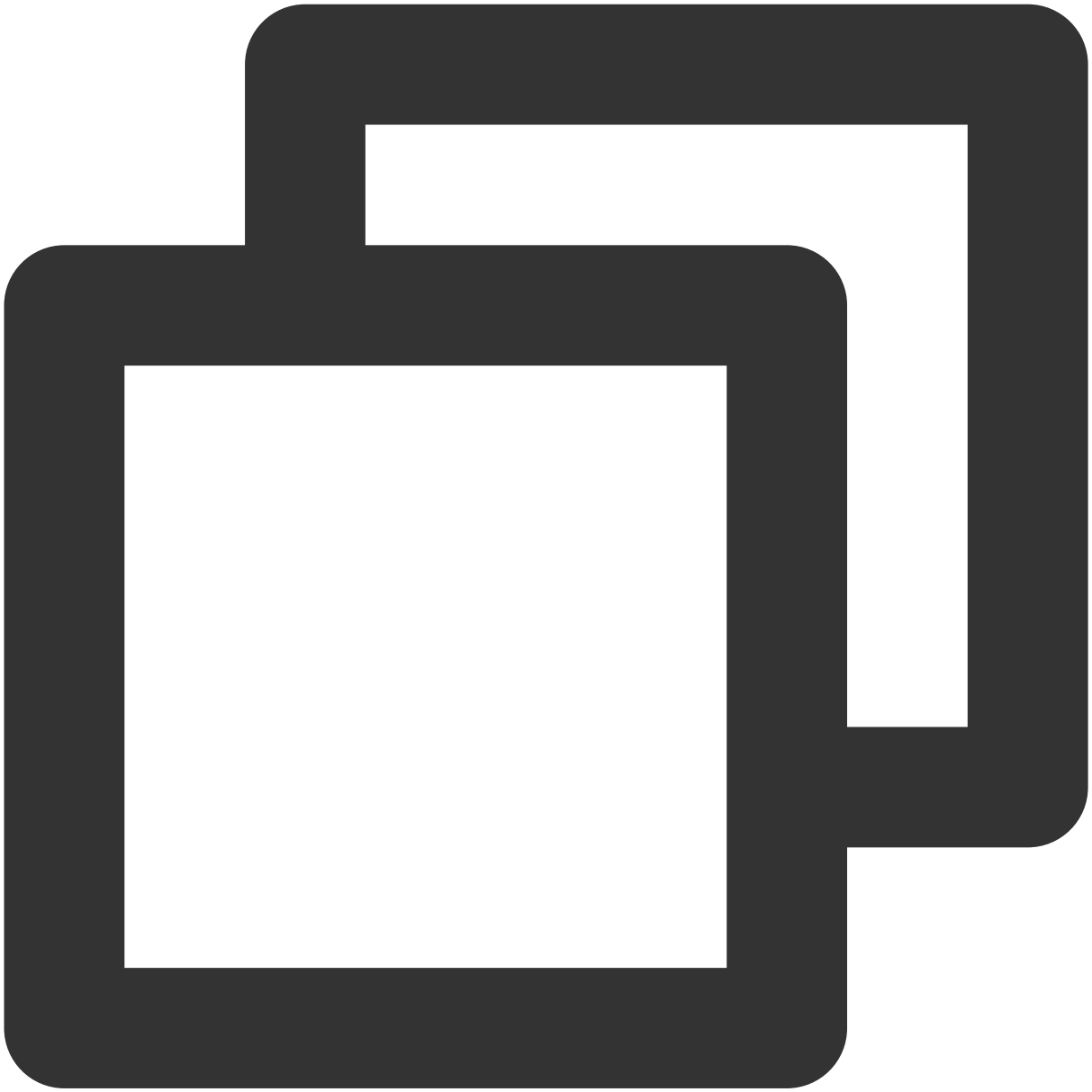
```
EVERY(<col> boolean)
```

支持引擎：SparkSQL、Presto

使用说明：如果col的所有值均为true，则返回true。

返回类型：boolean

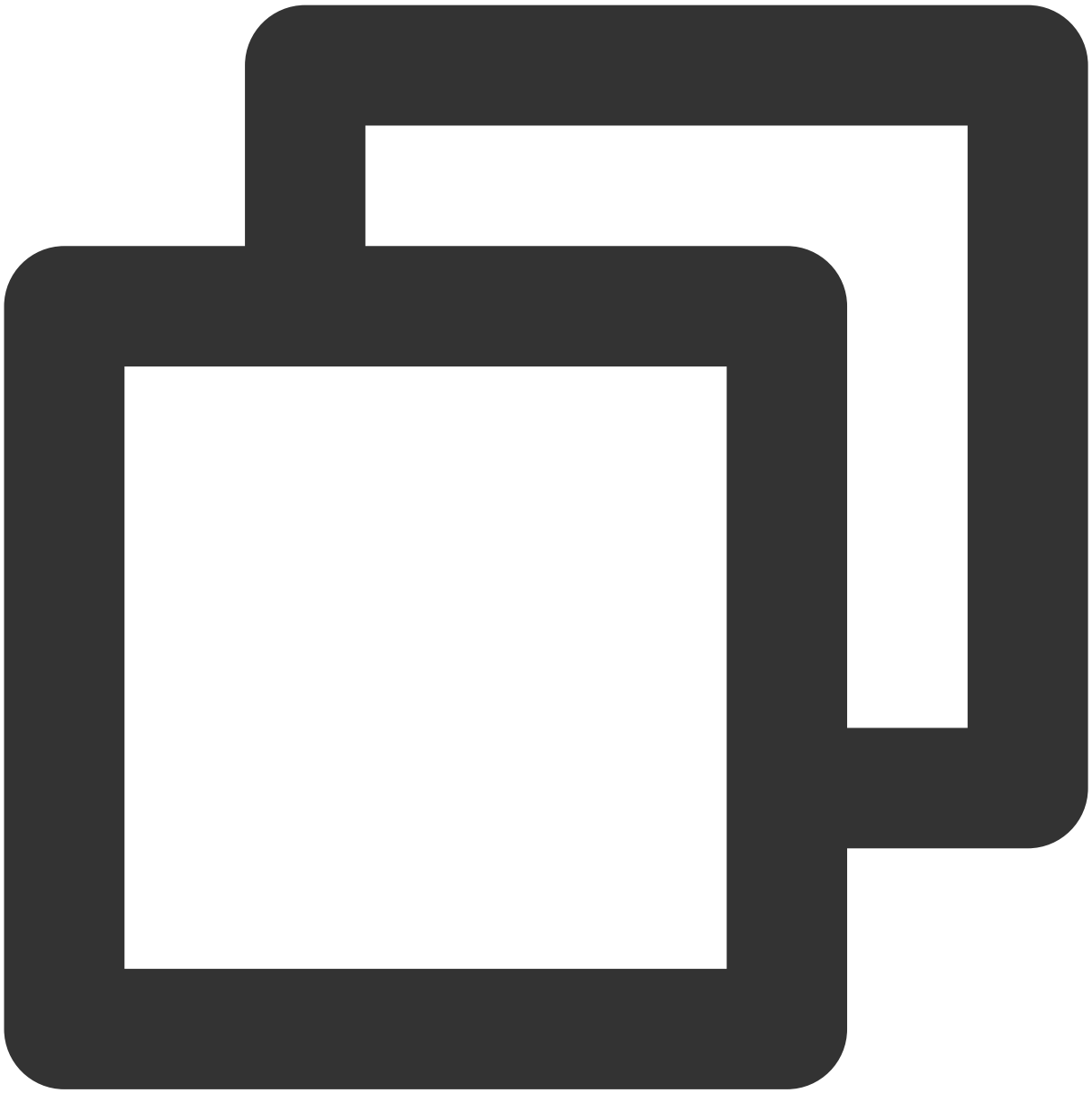
示例：



```
> SELECT every(col) FROM (VALUES (true), (true), (true)) AS tab(col);
 true
> SELECT every(col) FROM (VALUES (NULL), (true), (true)) AS tab(col);
 true
> SELECT every(col) FROM (VALUES (true), (false), (true)) AS tab(col);
 false
```

BOOL_AND

函数语法：



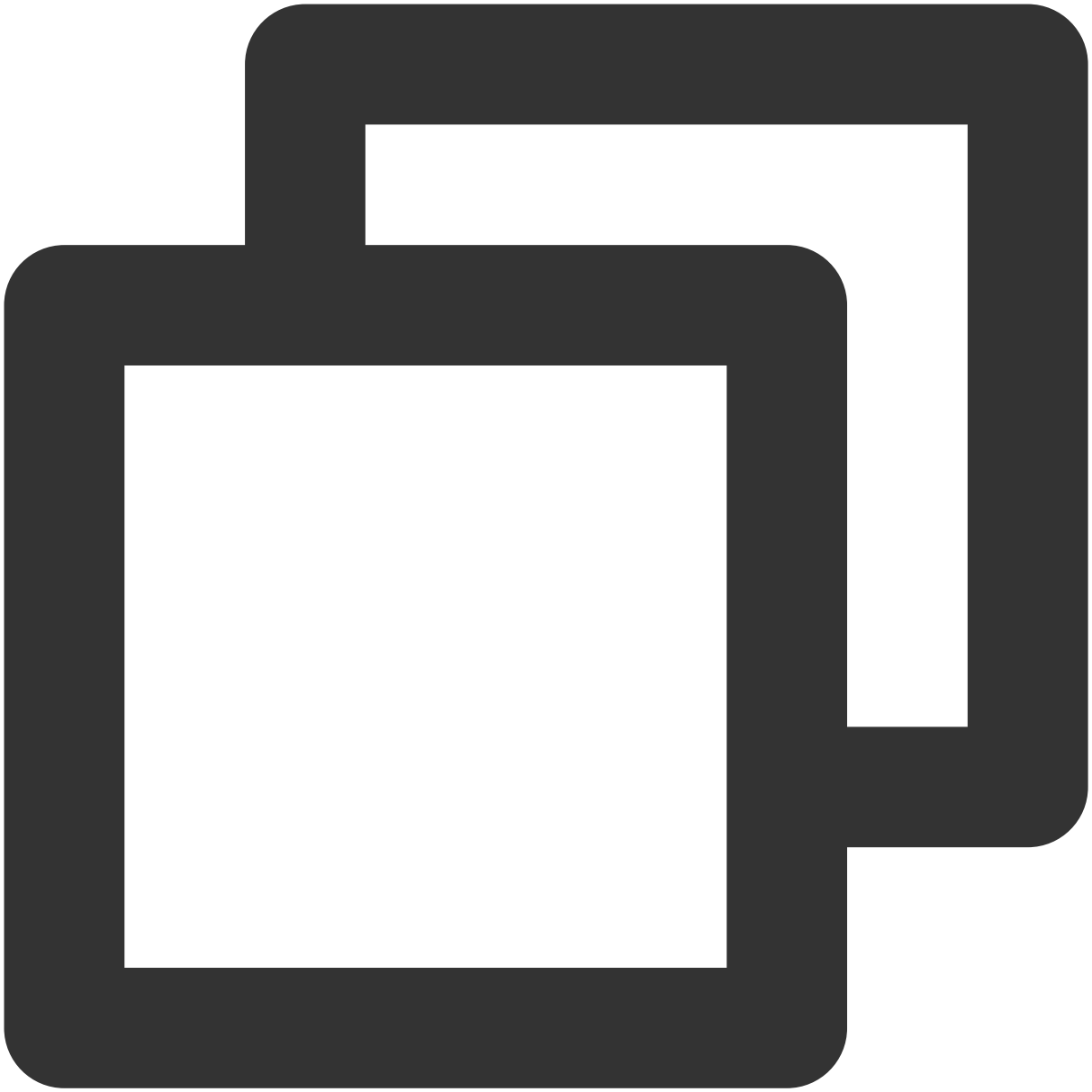
```
BOOL_AND(<col> boolean)
```

支持引擎：SparkSQL、Presto

使用说明：如果col的所有值均为true，则返回true。

返回类型：boolean

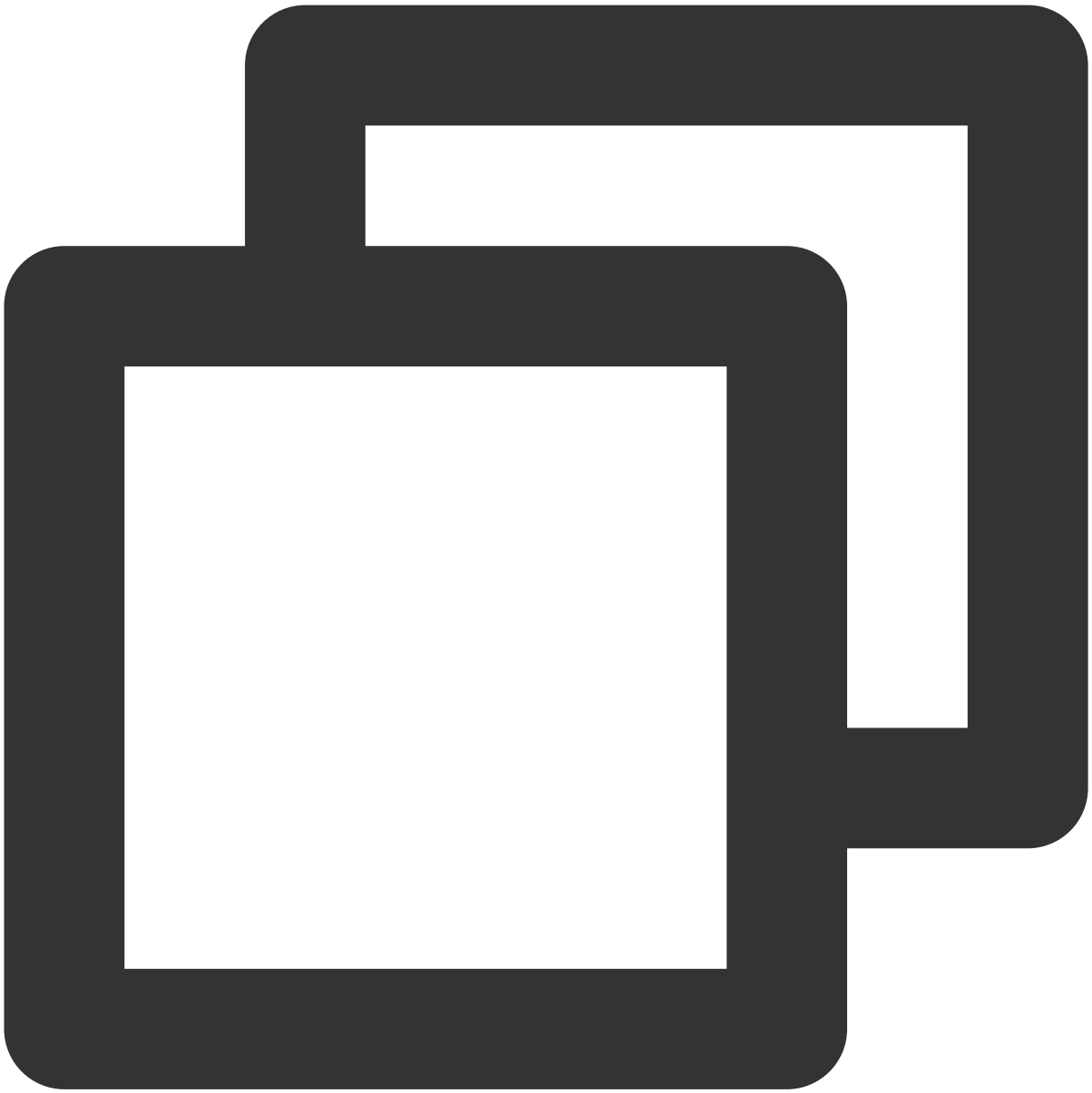
示例：



```
> SELECT bool_and(col) FROM (VALUES (true), (true), (true)) AS tab(col);
 true
> SELECT bool_and(col) FROM (VALUES (NULL), (true), (true)) AS tab(col);
 true
> SELECT bool_and(col) FROM (VALUES (true), (false), (true)) AS tab(col);
 false
```

AND

函数语法：



```
<expr1> AND <expr2>
```

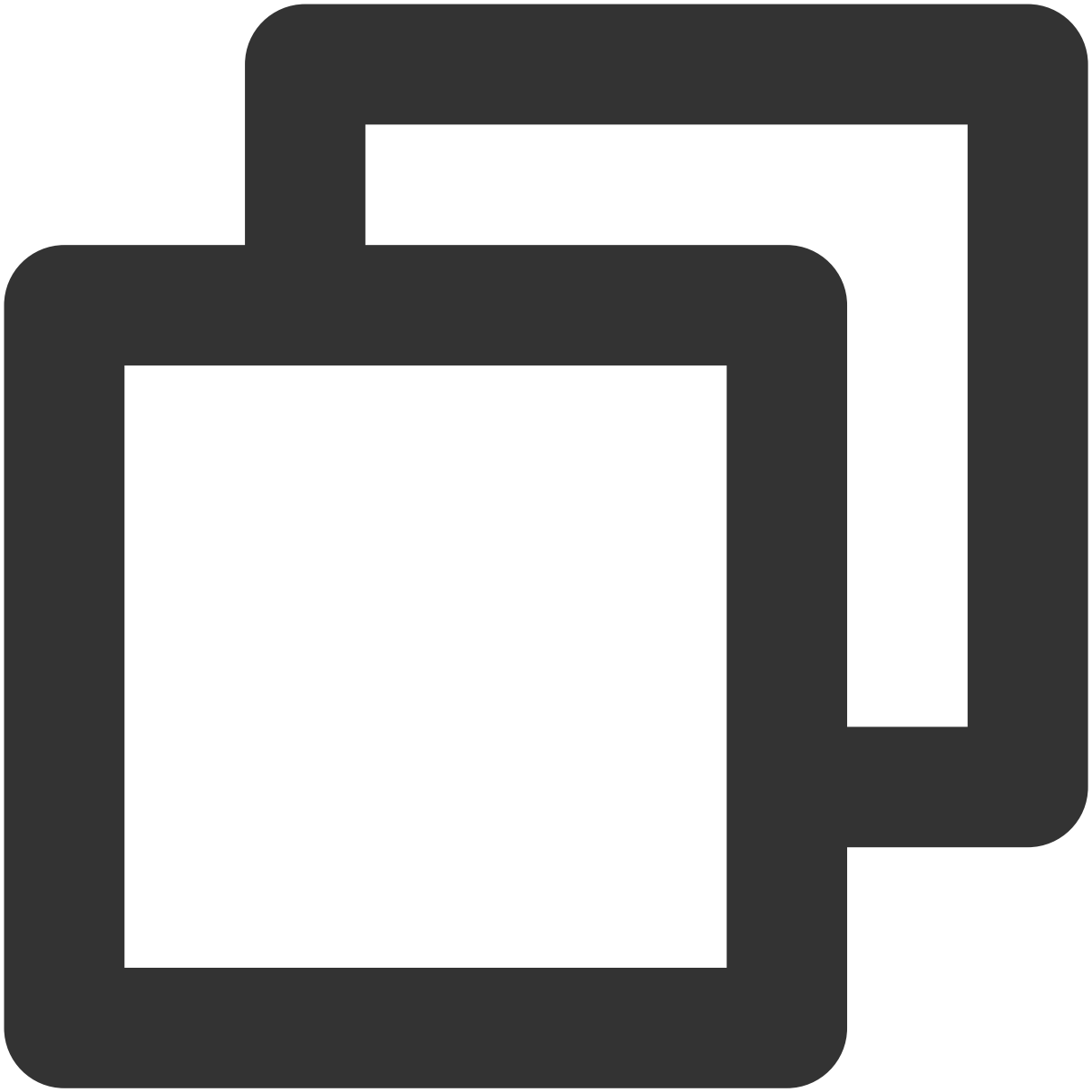
支持引擎：SparkSQL、Presto

使用说明

逻辑与

返回类型：boolean

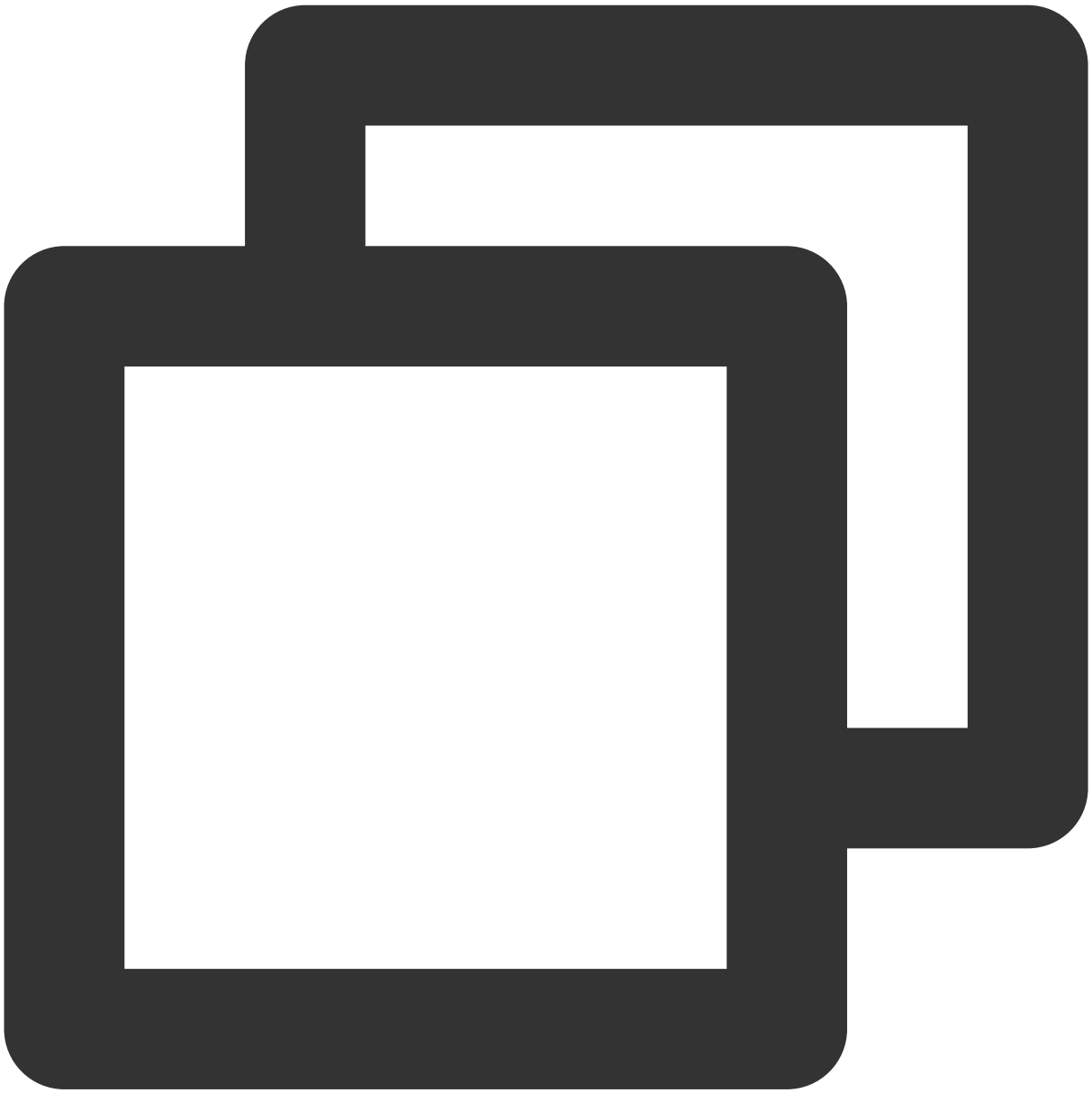
示例：



```
> SELECT true and true;
 true
> SELECT true and false;
 false
> SELECT true and NULL;
 NULL
> SELECT false and NULL;
 false
```

OR

函数语法：



```
<expr1> OR <expr2>
```

支持引擎：SparkSQL、Presto

使用说明：逻辑或

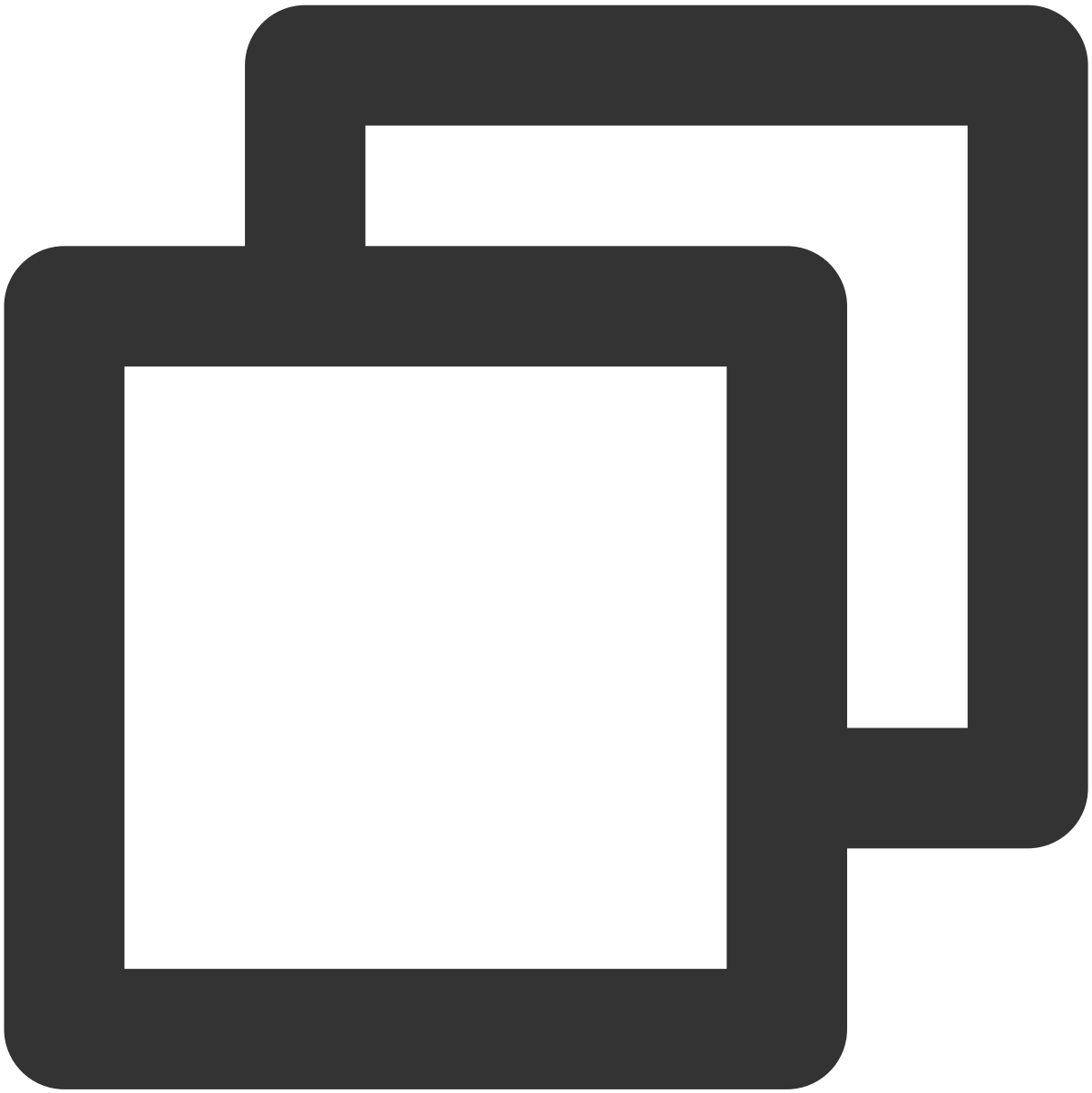
返回类型：boolean

示例：

```
> SELECT true or false;
 true
> SELECT false or false;
 false
> SELECT true or NULL;
 true
> SELECT false or NULL;
 NULL
```

ANY

函数语法：



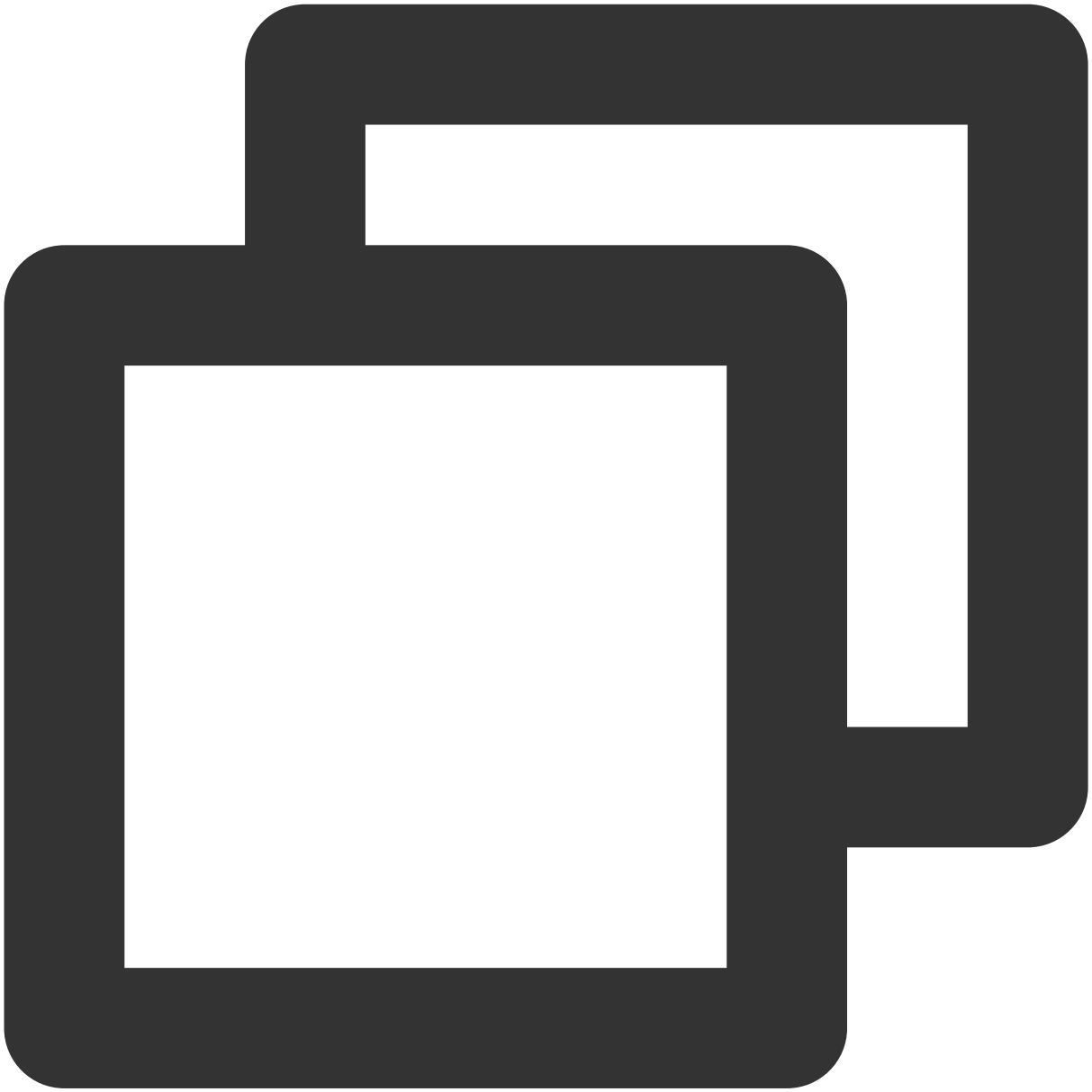
```
ANY(<col> boolean)
```

支持引擎：SparkSQL、Presto

使用说明：如果col的至少一个值为true，则返回true。

返回类型：boolean

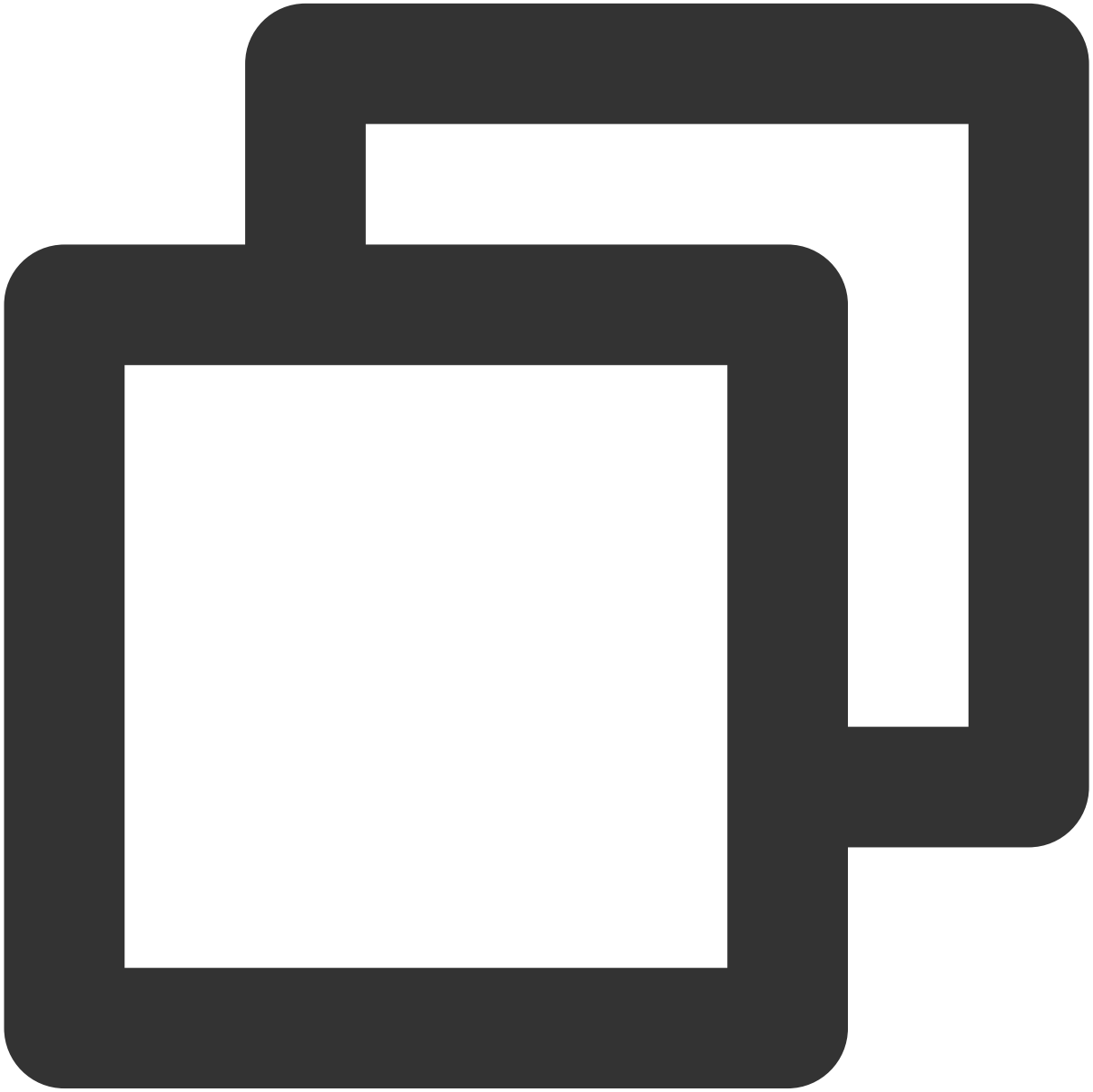
示例：



```
> SELECT any(col) FROM (VALUES (true), (false), (false)) AS tab(col);
 true
> SELECT any(col) FROM (VALUES (NULL), (true), (false)) AS tab(col);
 true
> SELECT any(col) FROM (VALUES (false), (false), (NULL)) AS tab(col);
 false
```

SOME

函数语法：



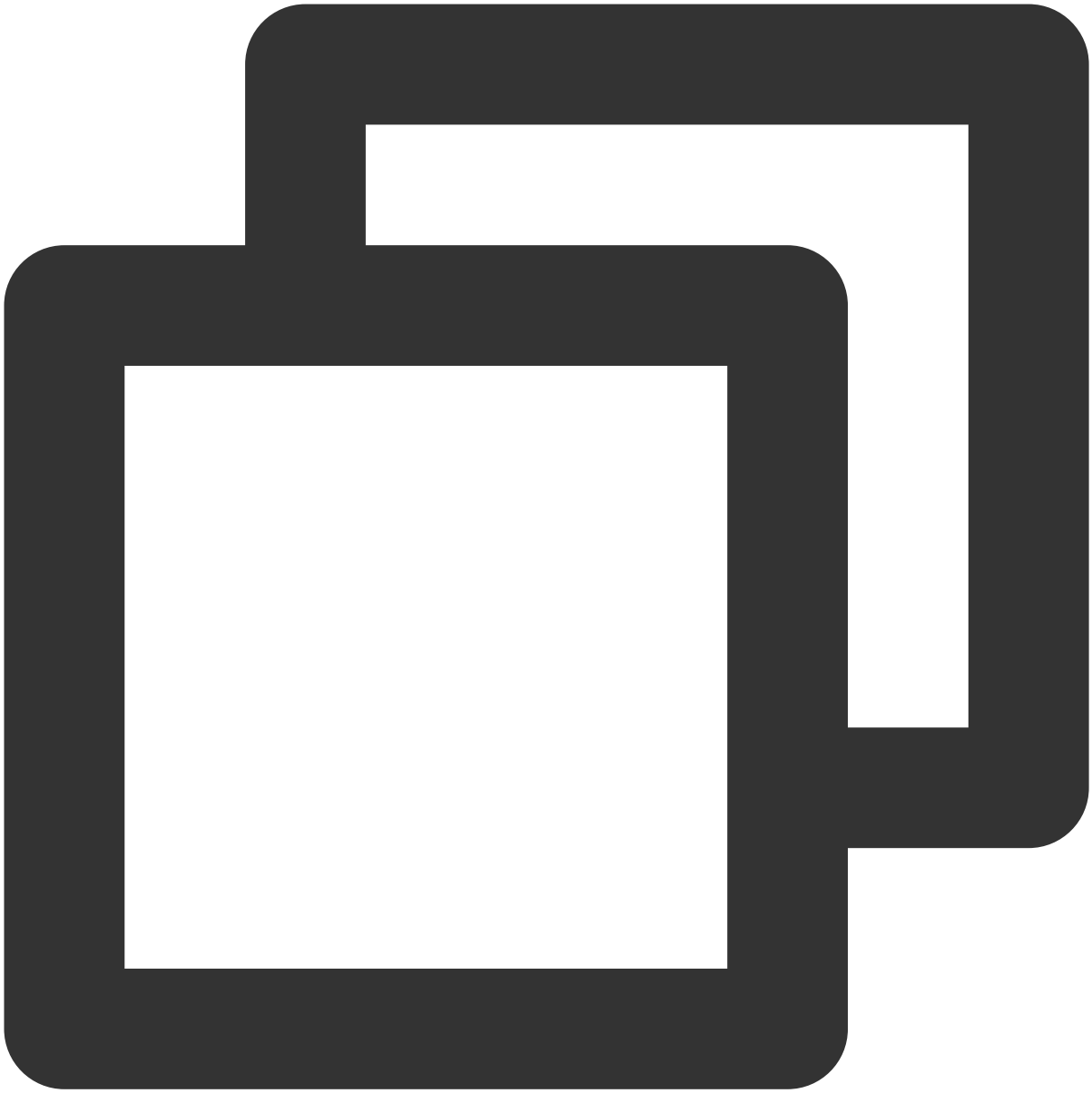
```
SOME(<col> boolean)
```

支持引擎：SparkSQL、Presto

使用说明：如果col的至少一个值为true，则返回true。

返回类型：boolean

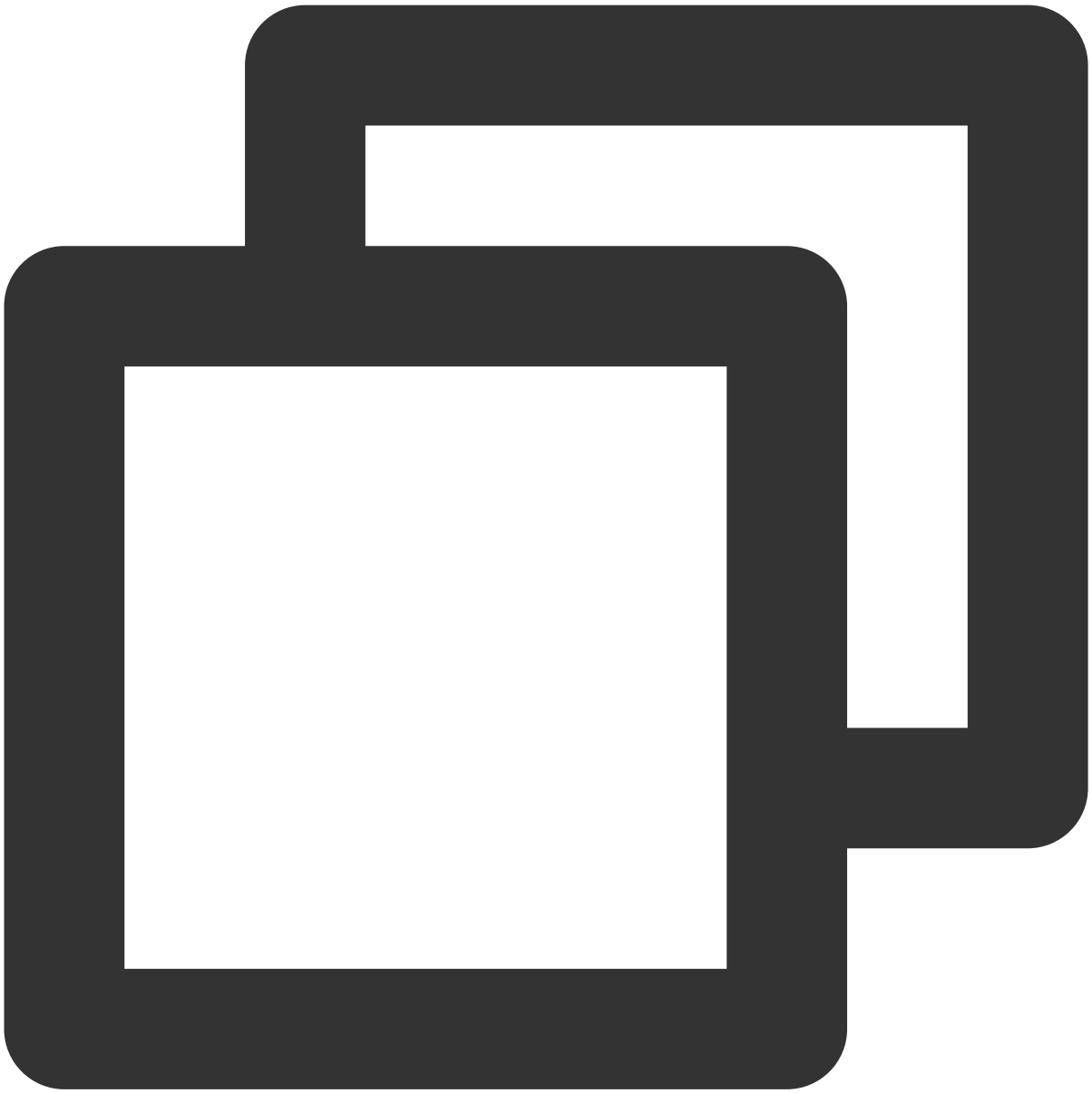
示例：



```
> SELECT some(col) FROM (VALUES (true), (false), (false)) AS tab(col);
 true
> SELECT some(col) FROM (VALUES (NULL), (true), (false)) AS tab(col);
 true
> SELECT some(col) FROM (VALUES (false), (false), (NULL)) AS tab(col);
 false
```

BOOL_OR

函数语法：



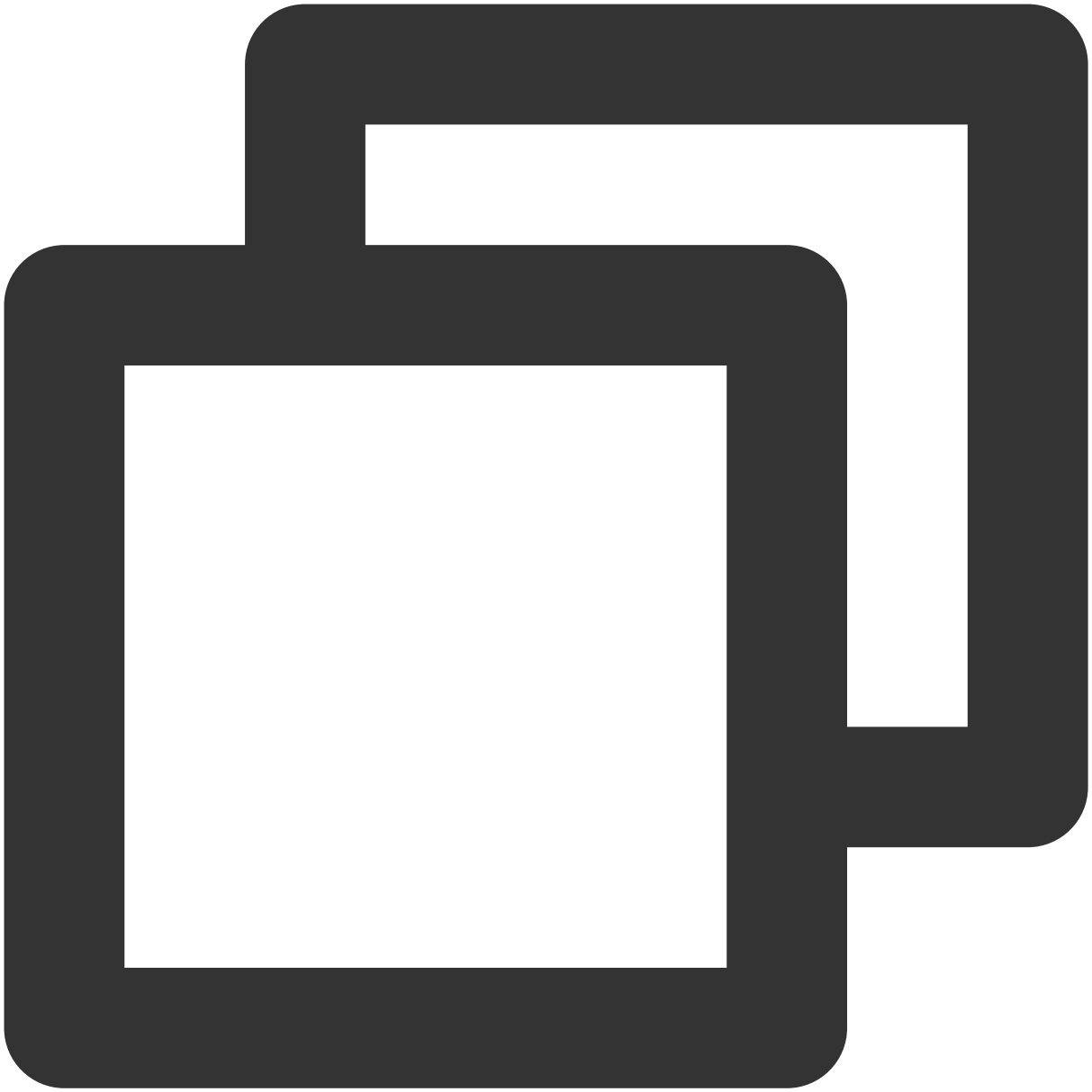
```
BOOL_OR(<col> boolean)
```

支持引擎：SparkSQL、Presto

使用说明：如果col的至少一个值为true，则返回true。

返回类型：boolean

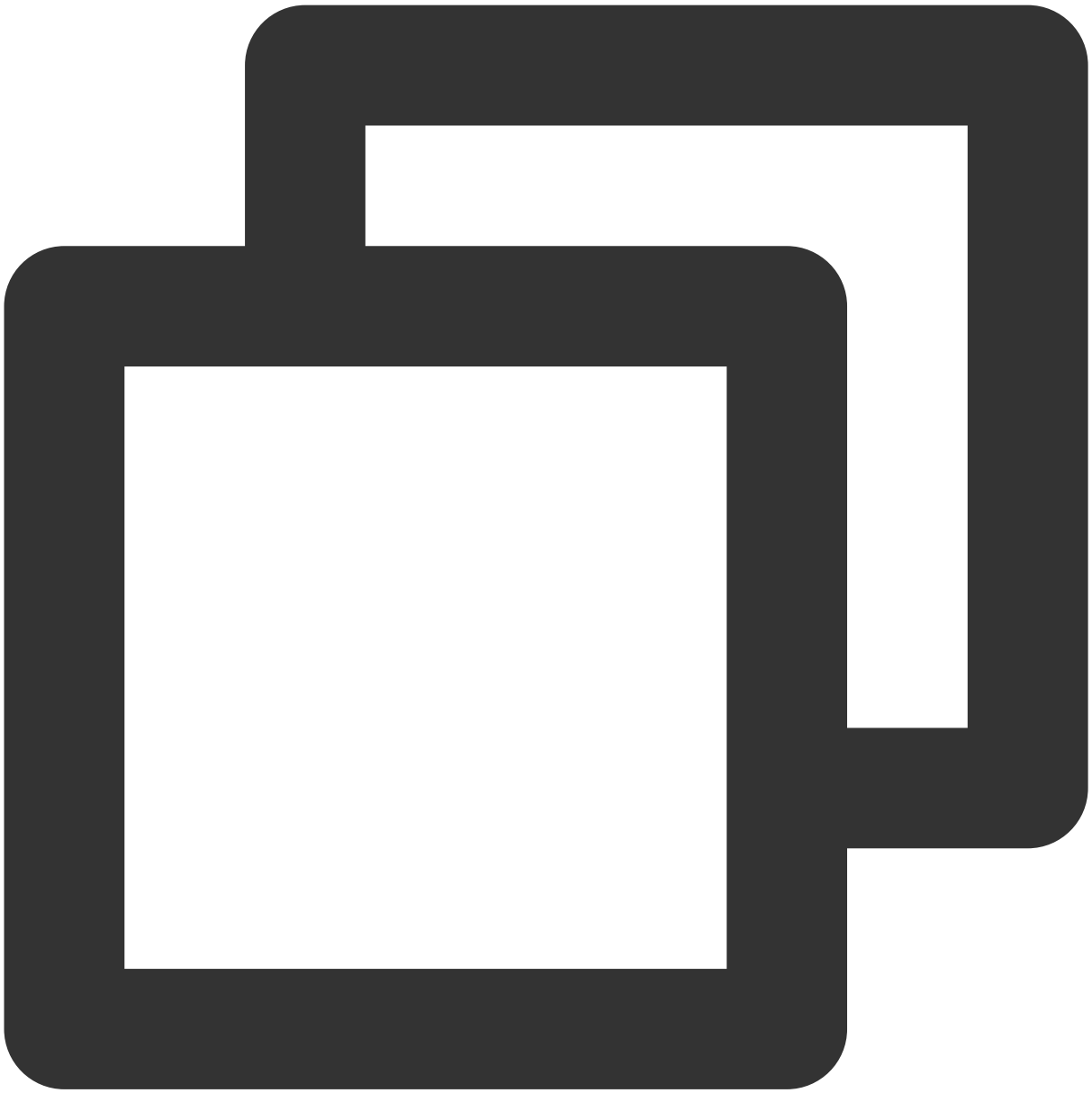
示例：



```
> SELECT BOOL_OR(col) FROM (VALUES (true), (false), (false)) AS tab(col);
 true
> SELECT BOOL_OR(col) FROM (VALUES (NULL), (true), (false)) AS tab(col);
 true
> SELECT BOOL_OR(col) FROM (VALUES (false), (false), (NULL)) AS tab(col);
 false
```

BIT_AND

函数语法：



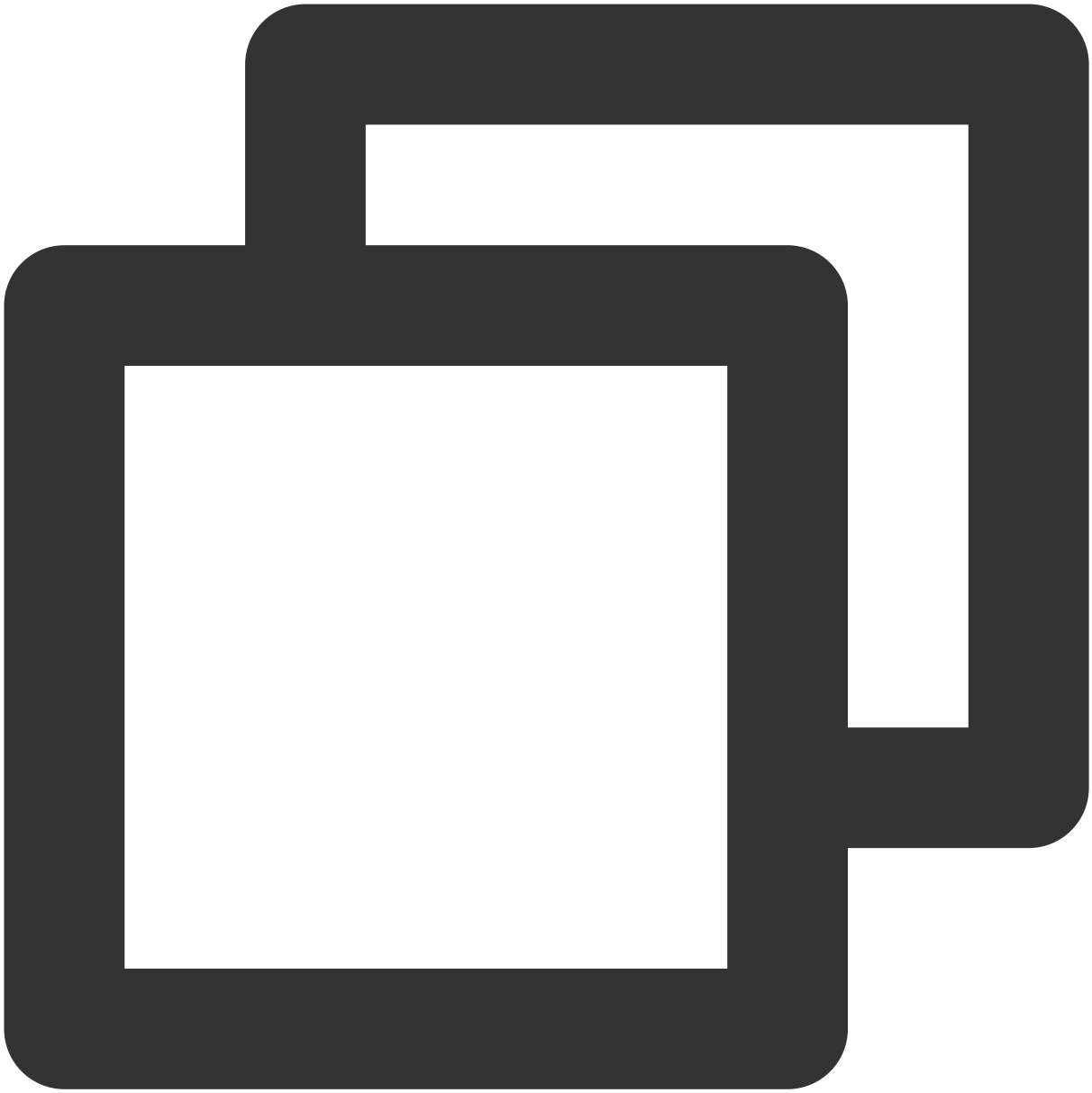
```
BIT_AND(<col> integer|bigint)
```

支持引擎：SparkSQL、Presto

使用说明：返回所有非空输入值的按位AND，如果没有，则返回null。

返回类型：与col一致

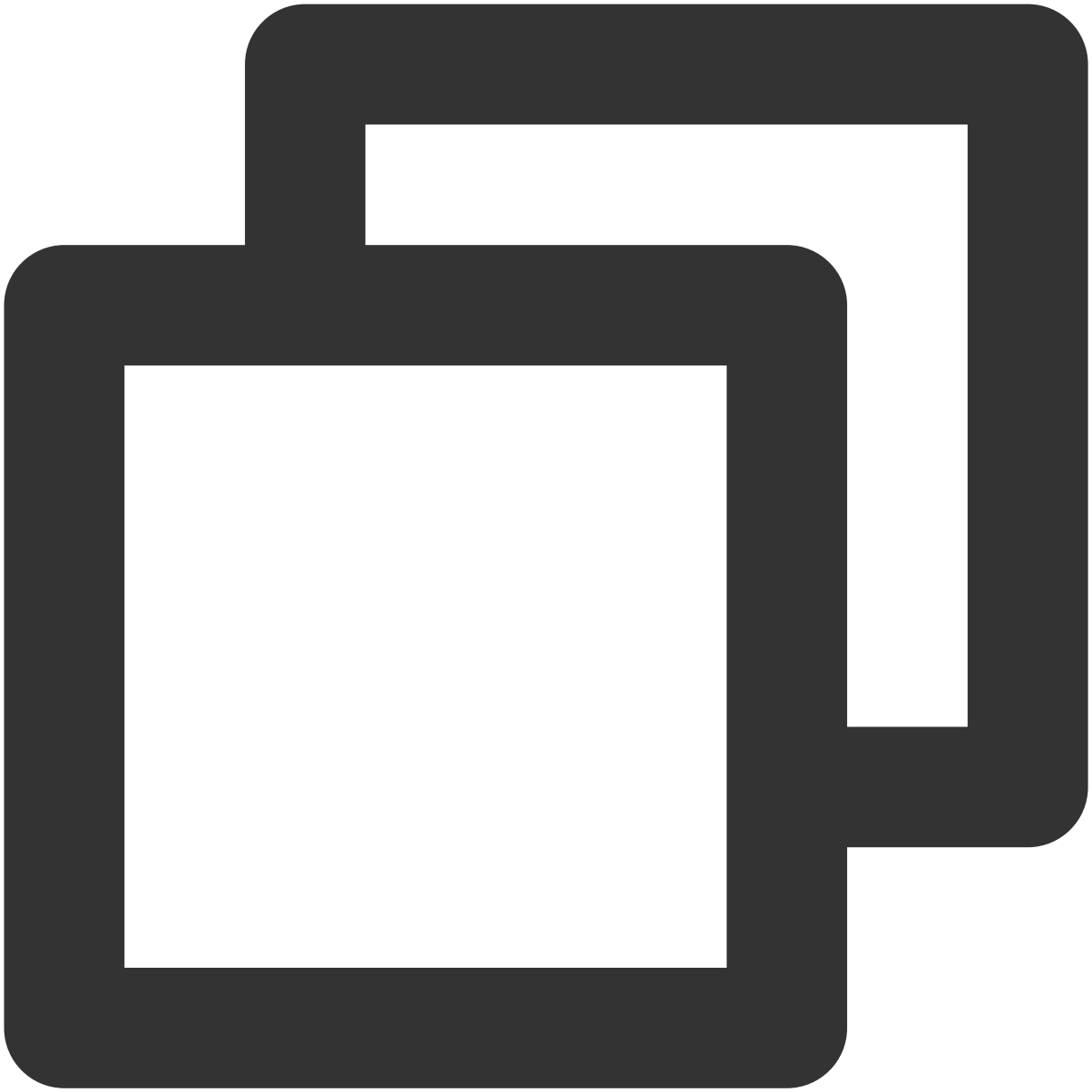
示例：



```
> SELECT bit_and(col) FROM (VALUES (3), (5)) AS tab(col);
 1
```

BIT_OR

函数语法：



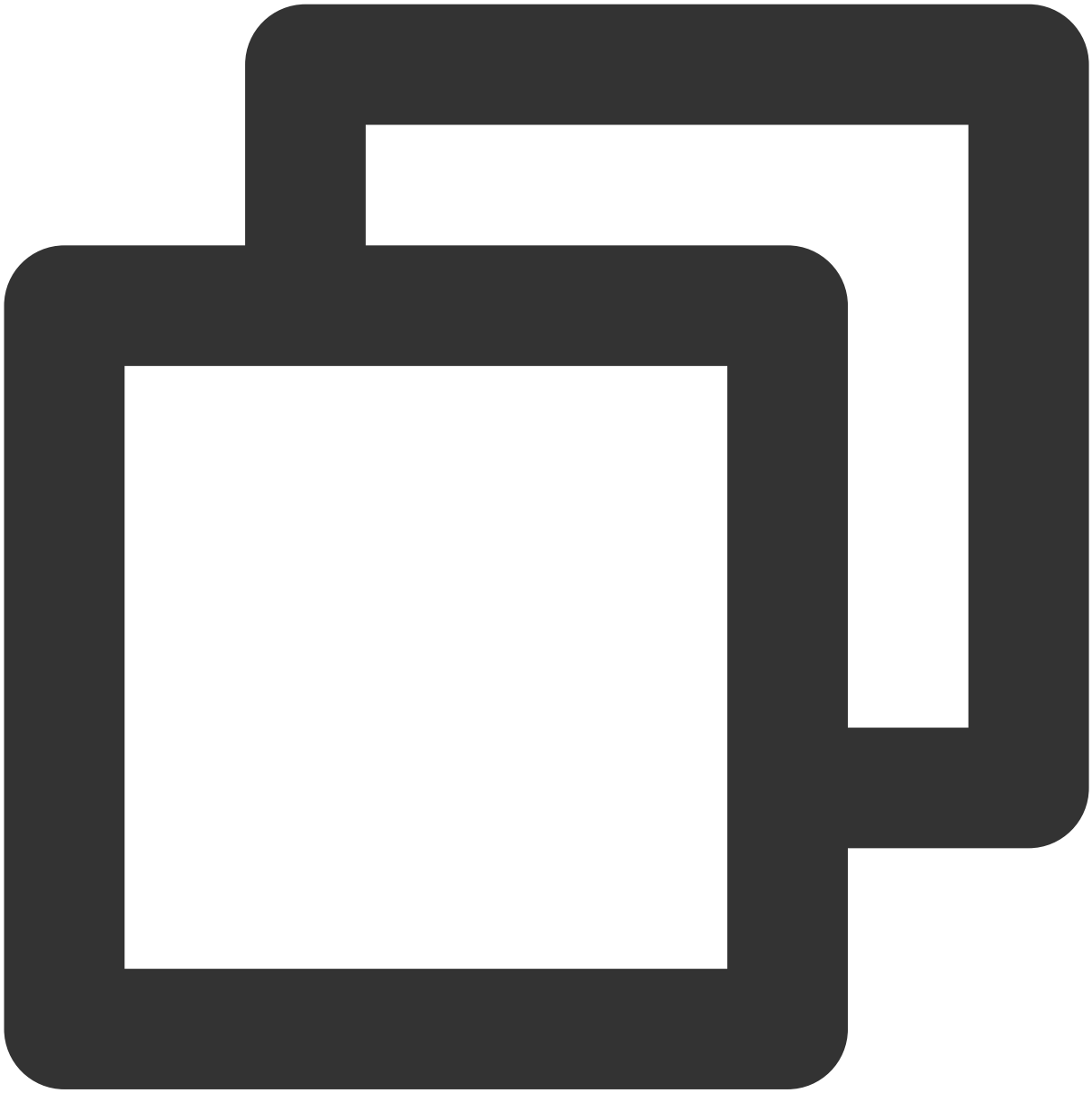
```
BIT_OR(<col> integer|bigint)
```

支持引擎：SparkSQL、Presto

使用说明：返回所有非空输入值的按位OR，如果没有，则返回null。

返回类型：与col一致

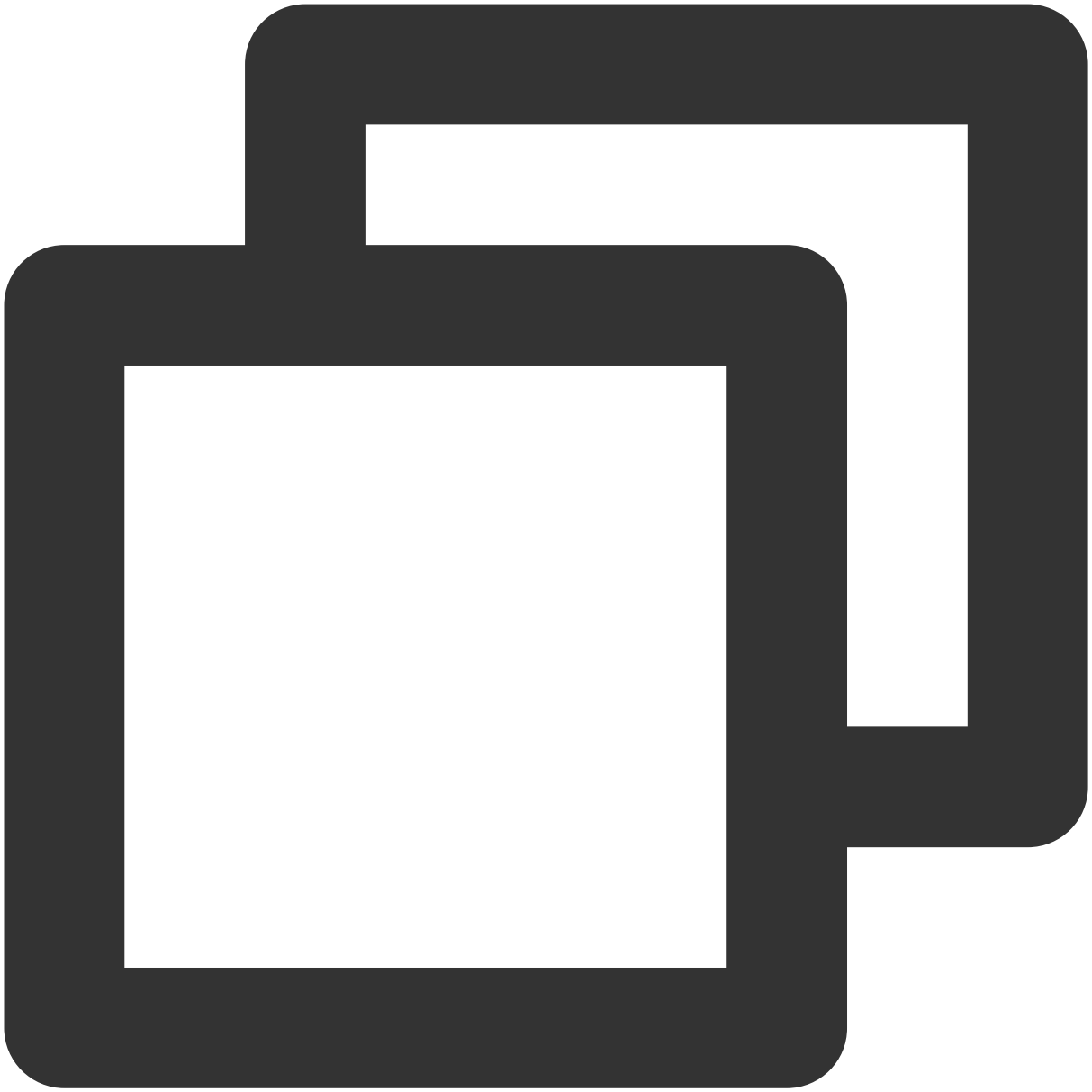
示例：



```
> SELECT bit_or(col) FROM (VALUES (3), (5)) AS tab(col);
 7
```

BIT_XOR

函数语法：



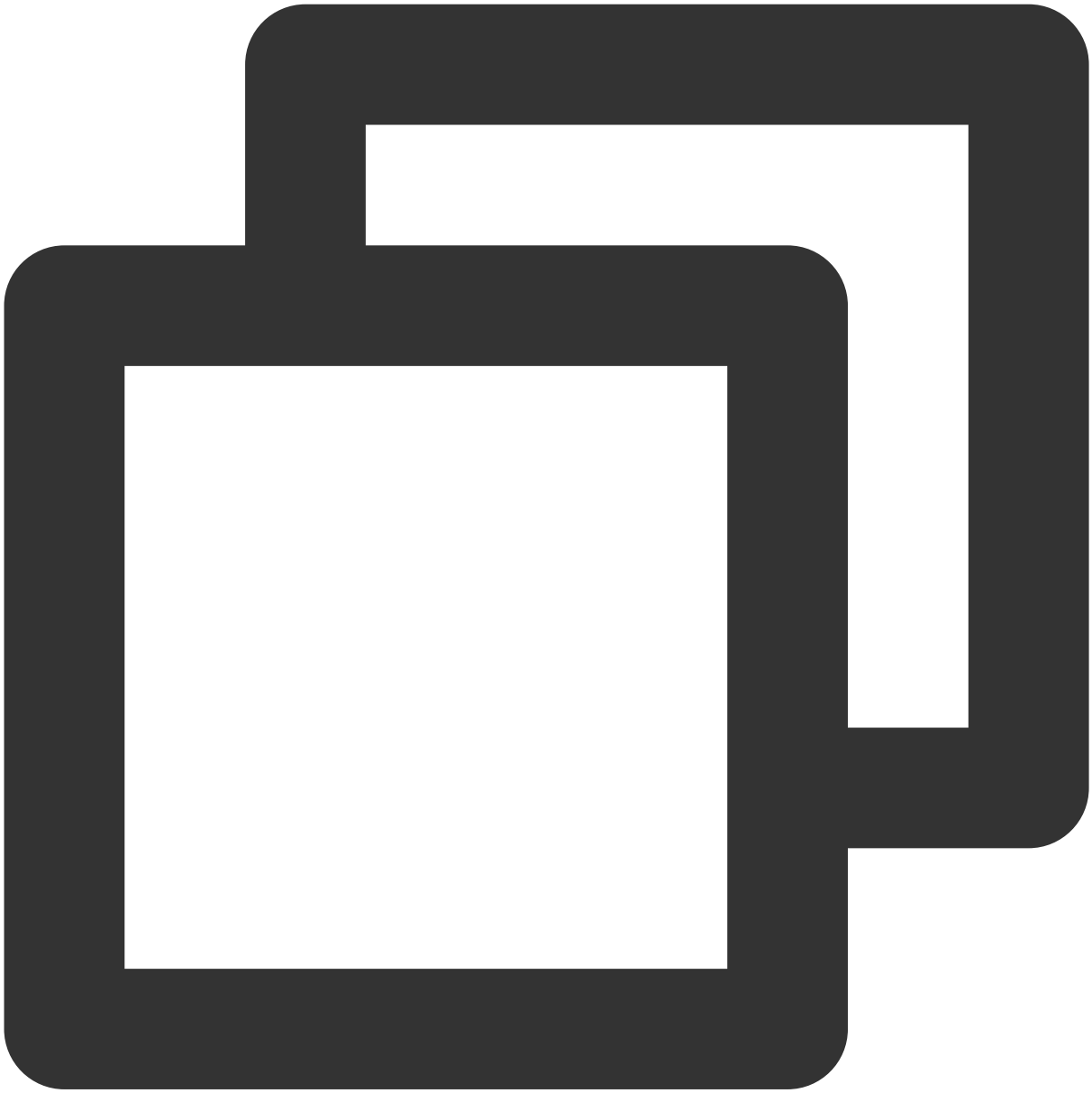
```
BIT_XOR(<col> integer|bigint)
```

支持引擎：SparkSQL、Presto

使用说明：返回所有非空输入值的按位XOR，如果没有，则返回null。

返回类型：与col一致

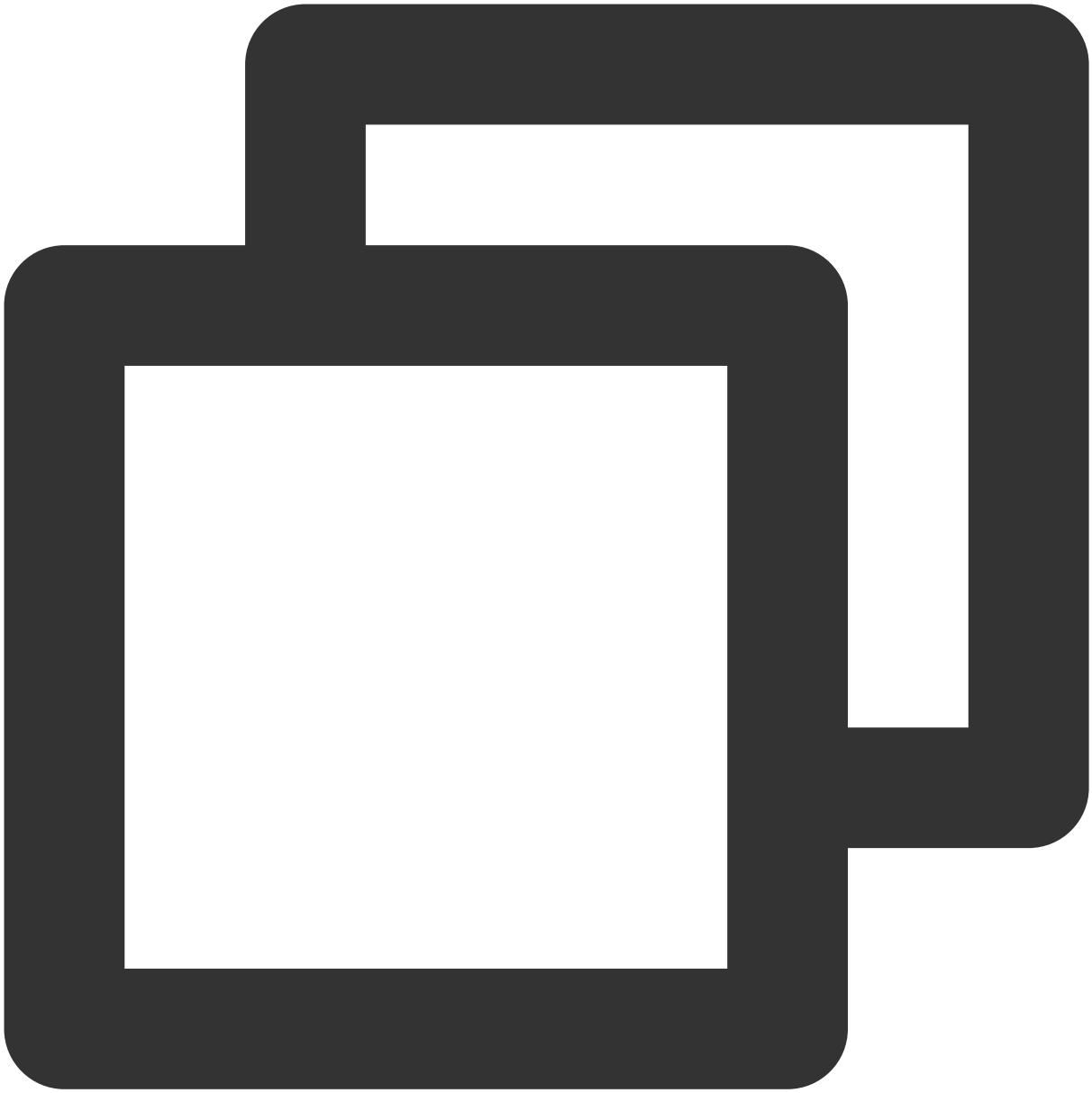
示例：



```
> SELECT bit_xor(col) FROM (VALUES (3), (5)) AS tab(col);
 6
```

ARG_MIN

函数语法：



```
ARG_MIN(<col1>, <col2> | expr(col2))
```

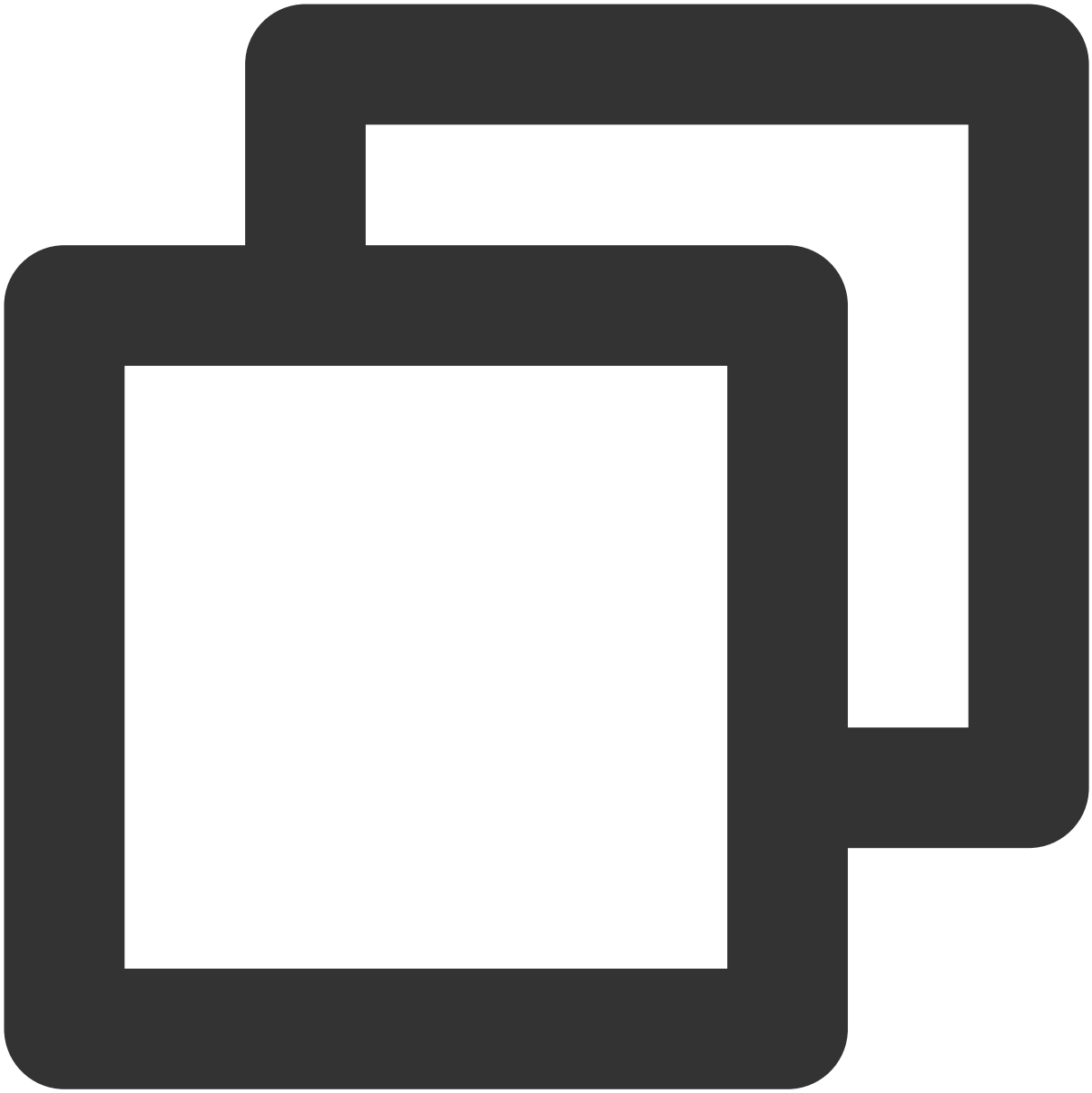
支持引擎：SparkSQ

使用说明：指定列col1最小的那一行，按指定的计算表达式返回。

col1: 可排序的列名，如果为常量，那么会返回任意值。如果有多个最小值，会随机返回其中一行。

返回类型：与col2或expr(col2)类型一致。

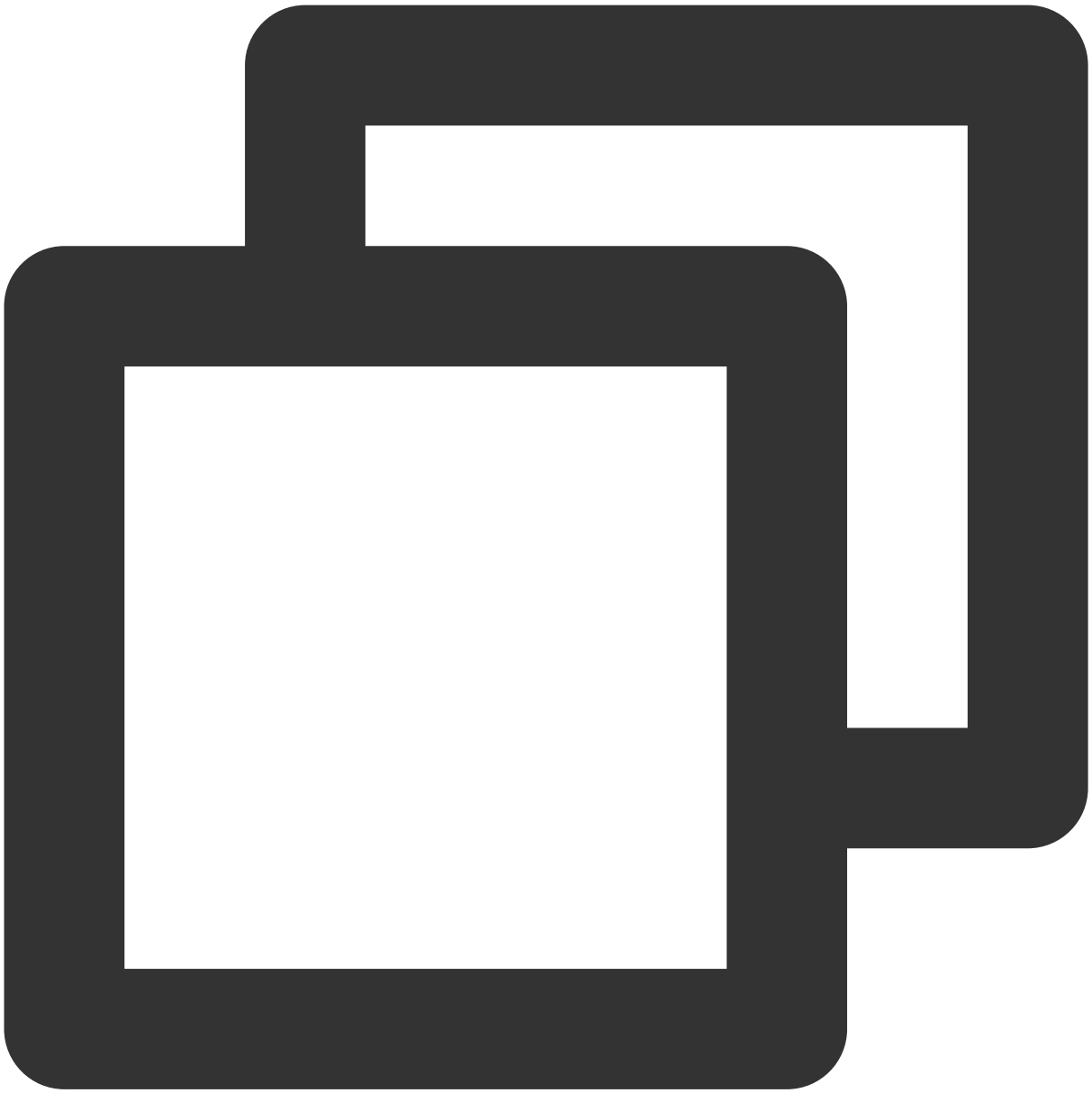
示例：



```
> SELECT arg_min(dt, uid) from values (1, 'm1'), (2, 't2'), (3, 'z3') as tab(dt, ui
m1
> SELECT arg_min(dt, upper(uid)) from values (1, 'm1', 1), (2, 't2', 1), (3, 'z3',
M1
Z3
```

ARG_MAX

函数语法：



```
ARG_MAX(<col1>, <col2> | expr(col2))
```

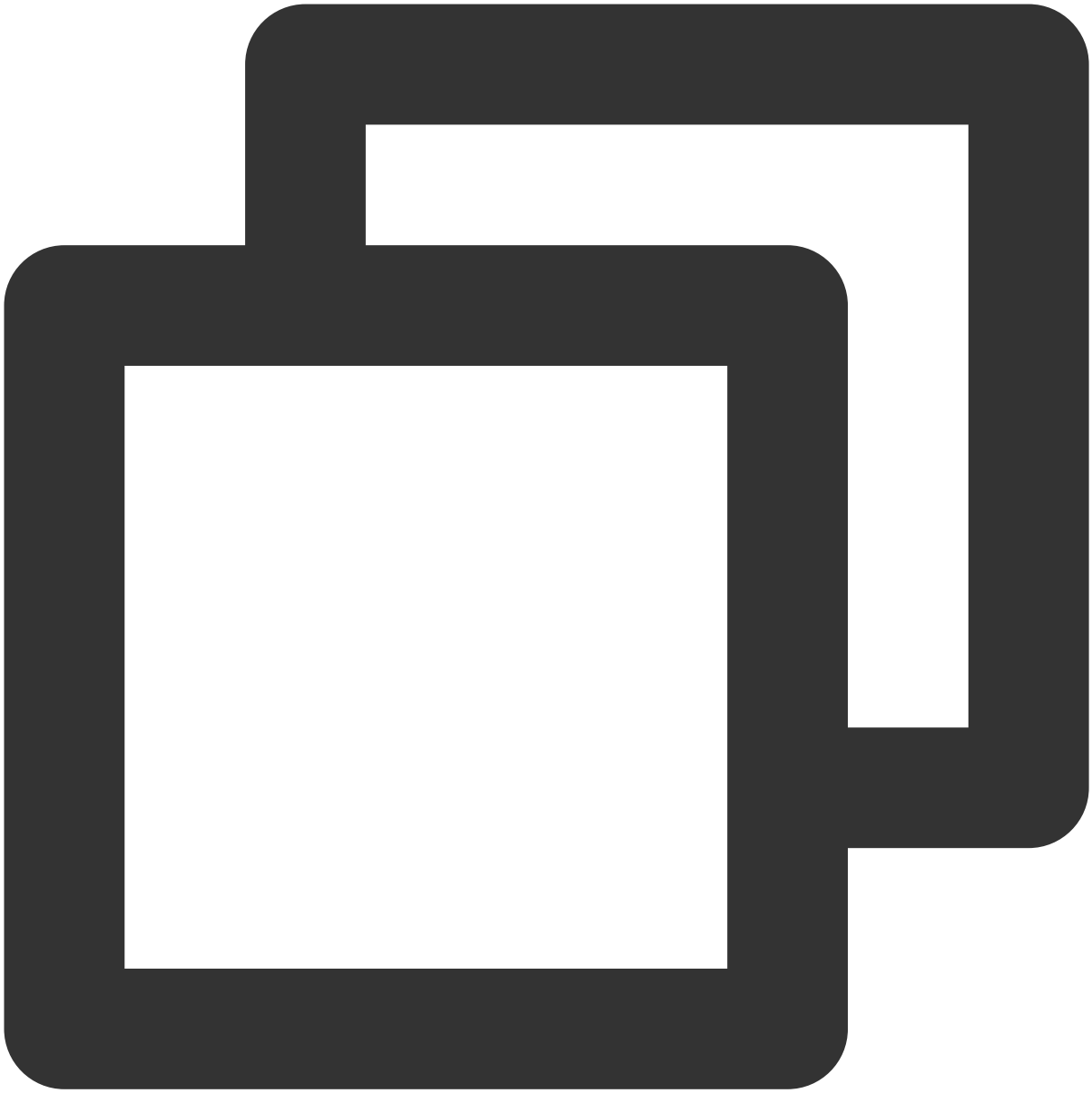
支持引擎：SparkSQL

使用说明：指定列col1最大的那一行，按指定的计算表达式返回。

col1: 可排序的列名，如果为常量，那么会返回任意值。如果有多个最大值，会随机返回其中一行。

返回类型：与col2或expr(col2)类型一致。

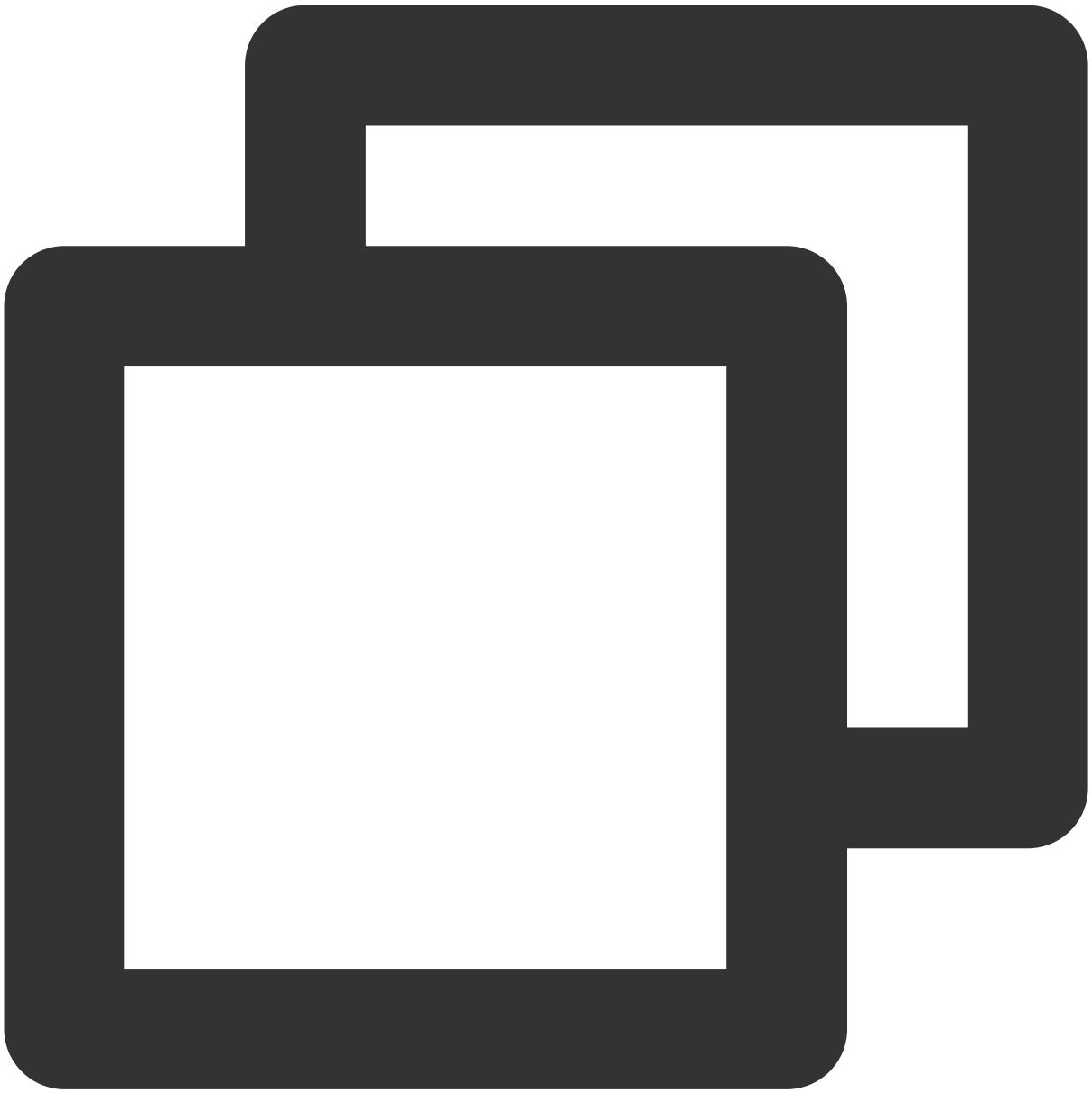
示例：



```
> SELECT arg_max(dt, uid) from values (1, 'm1'), (2, 't2'), (3, 'z3') as tab(dt, ui
z3
> SELECT arg_max(dt, upper(uid)) from values (1, 'm1', 1), (2, 't2', 1), (3, 'z3',
T2
Z3
```

MAP_UNION_SUM

函数语法：



```
MAP_UNION_SUM(map<k, v> input)
```

支持引擎：SparkSQL

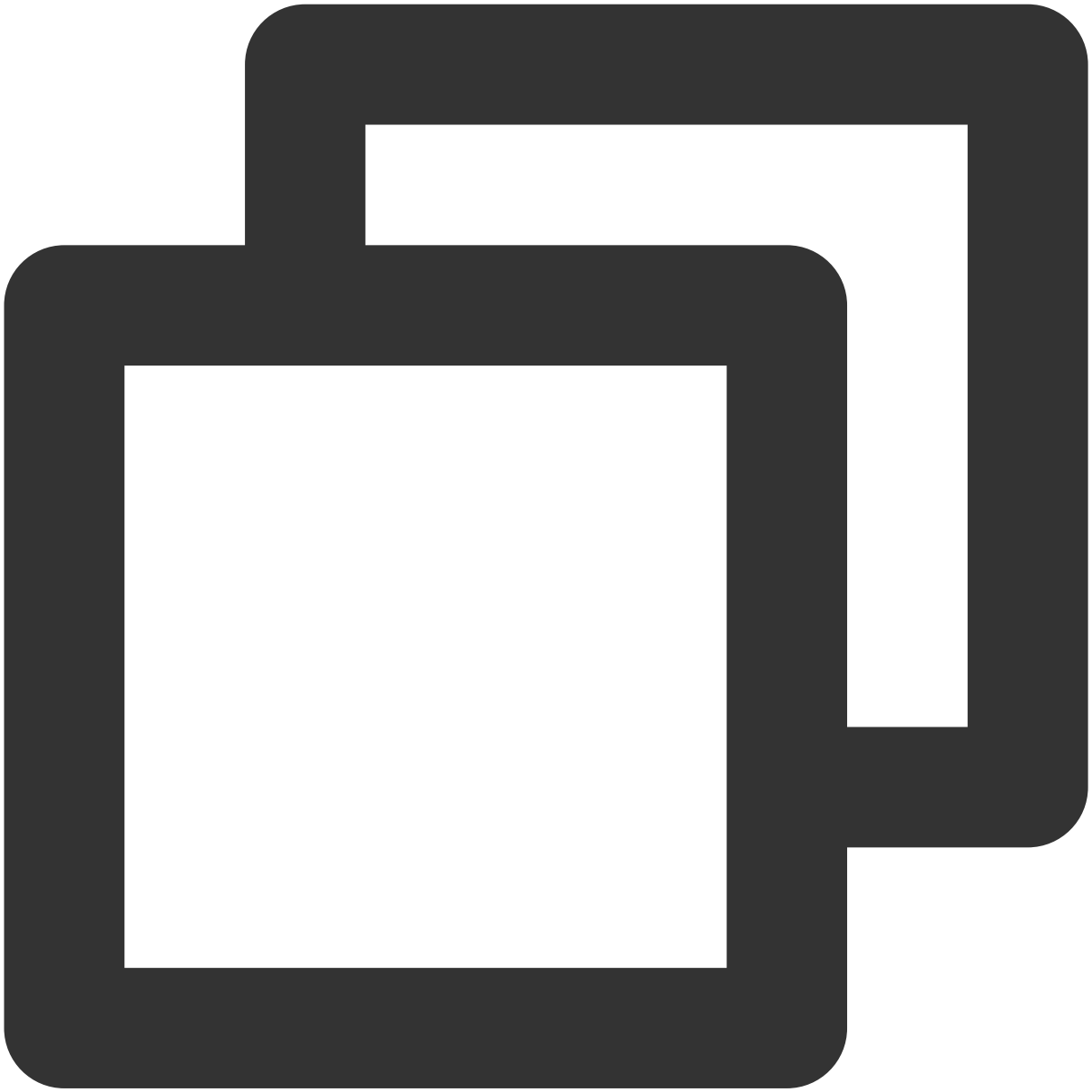
使用说明：指定列，将map按key值求和所有的value。

注意1：允许聚合的数据为null，全部为null时输出为null。

注意2：允许key的value为null，此时会转换为数字0。

返回类型：与输入类型一致。

示例：



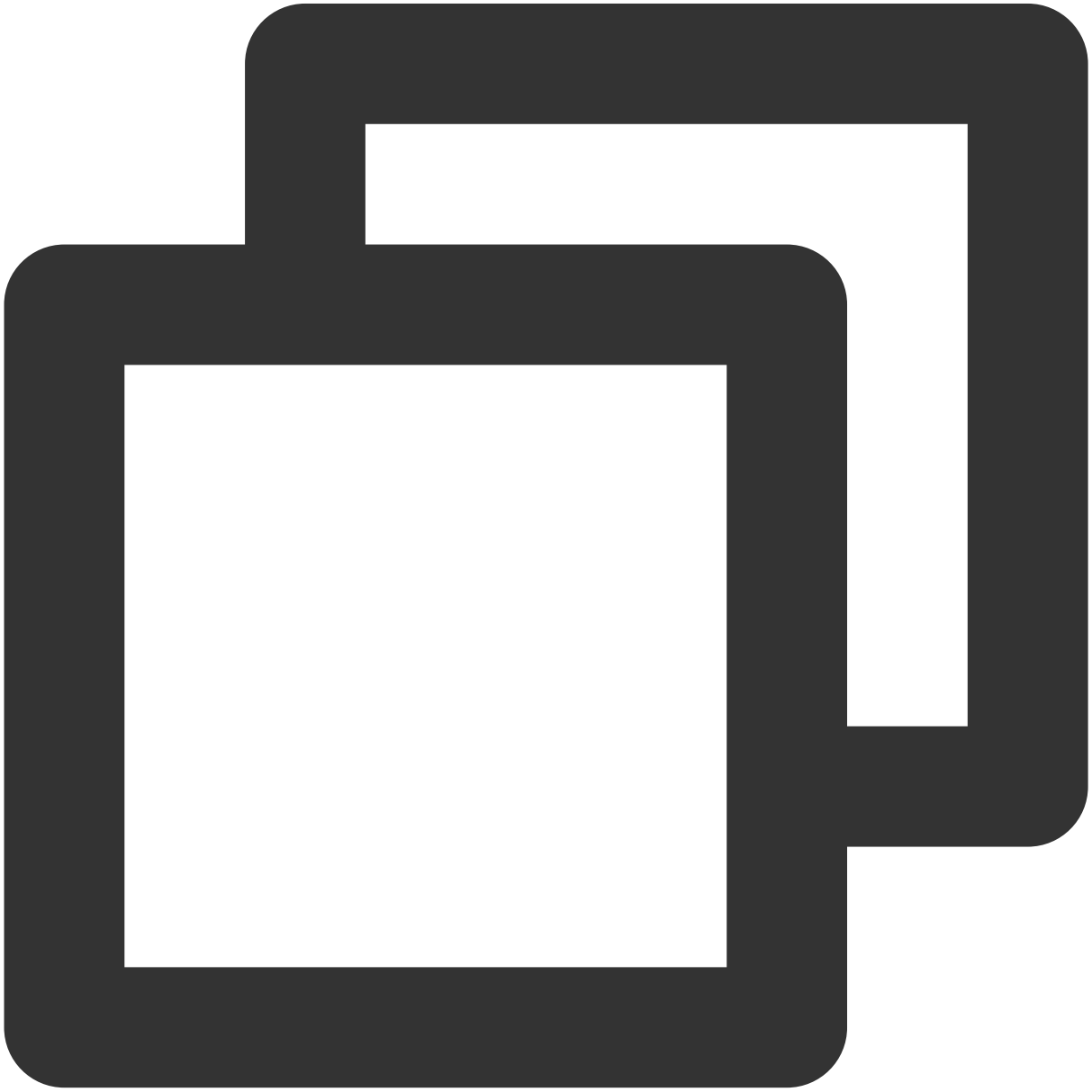
```
> SELECT map_union_sum(ids) from values (map(1, 1)), (map(1,2)), (map(2,1)) as tab(
{1:3,2:1}
> SELECT idx, map_union_sum(ids) from values (map(1, 1), 1), (map(1,2), 1), (map(2,
1 {1:3,2:1}
```

窗口函数

最近更新时间：2024-08-07 17:33:31

row_number

函数语法：



```
row_number()
```

支持引擎：SparkSQL、Presto

使用说明：为每一行分配一个唯一的连续编号

返回类型：int

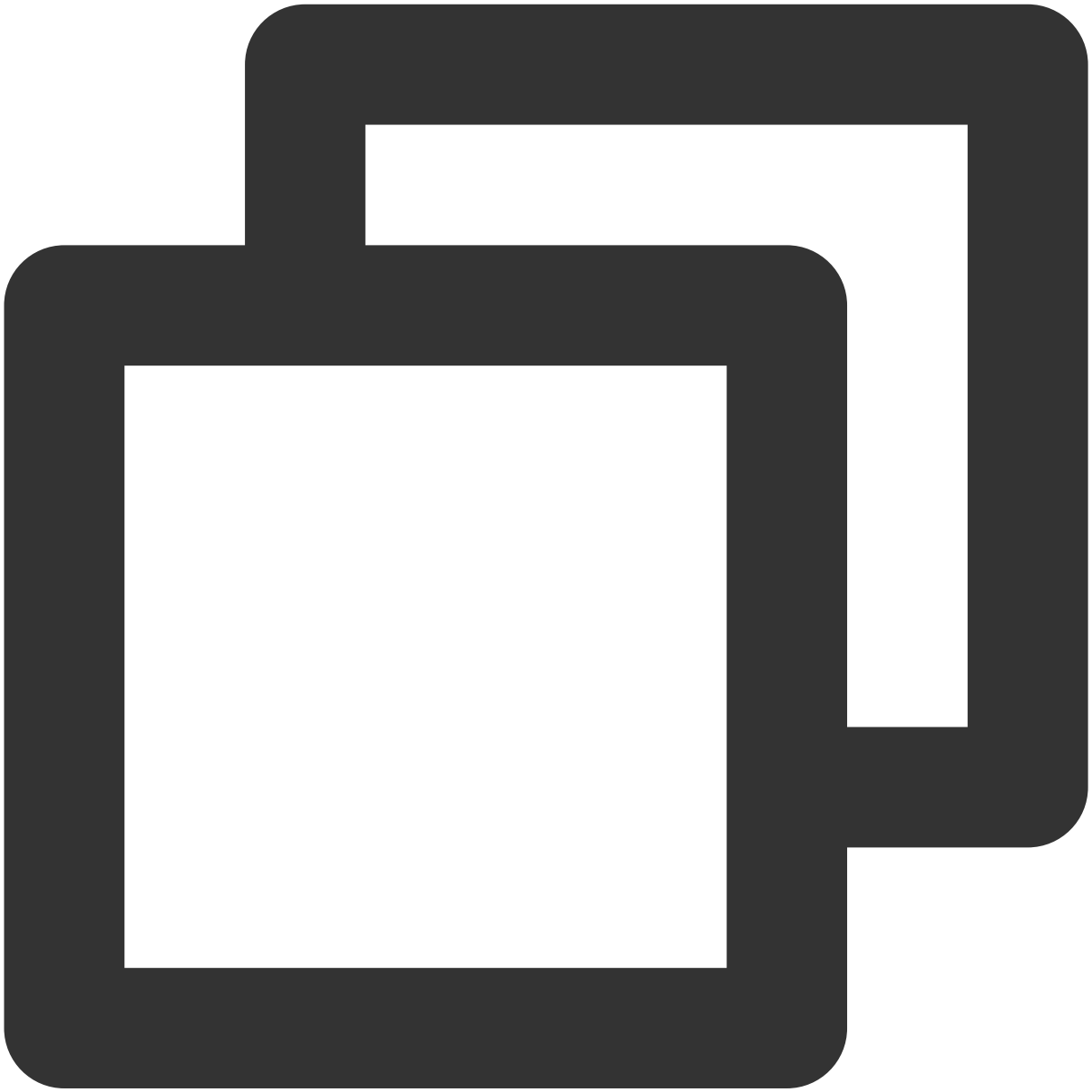
示例：

```
SELECT a, b, row_number() OVER (PARTITION BY a ORDER BY b) FROM VALUES ('A1', 2), (
A1       1        1
A1       1        2
A1       2        3
```

```
A2      3       1
```

rank

函数语法：



```
rank()
```

支持引擎：SparkSQL、Presto

使用说明：计算某个值在一组值中的排名。如果出现排名相等，则在排名序列中留出空位。

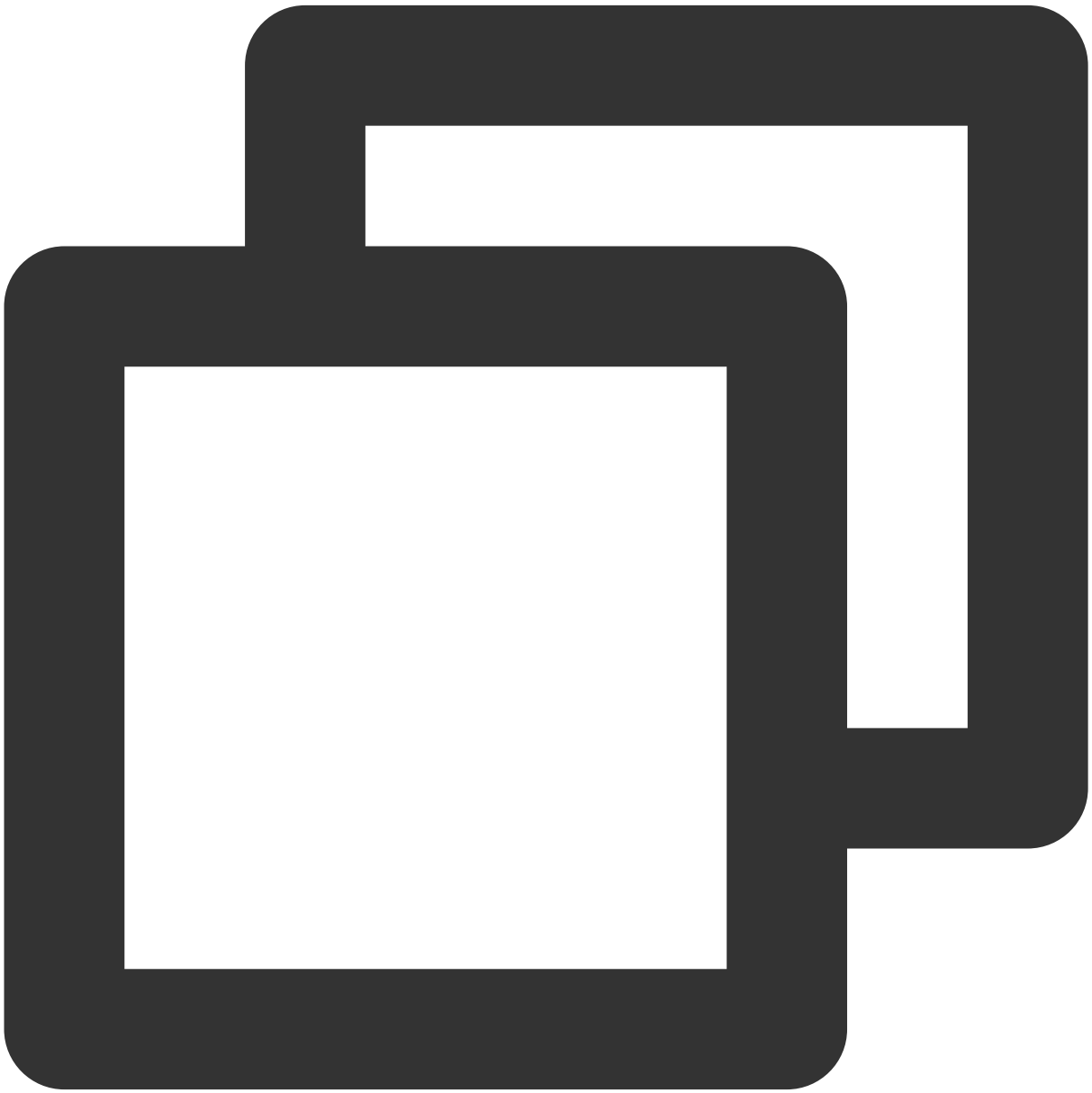
返回类型：int

示例：

```
SELECT a, b, rank() OVER (PARTITION BY a ORDER BY b) FROM VALUES ('A1', 2), ('A1',
A1      1       1
A1      1       1
A1      2       3
A2      3       1
```

dense_rank

函数语法：



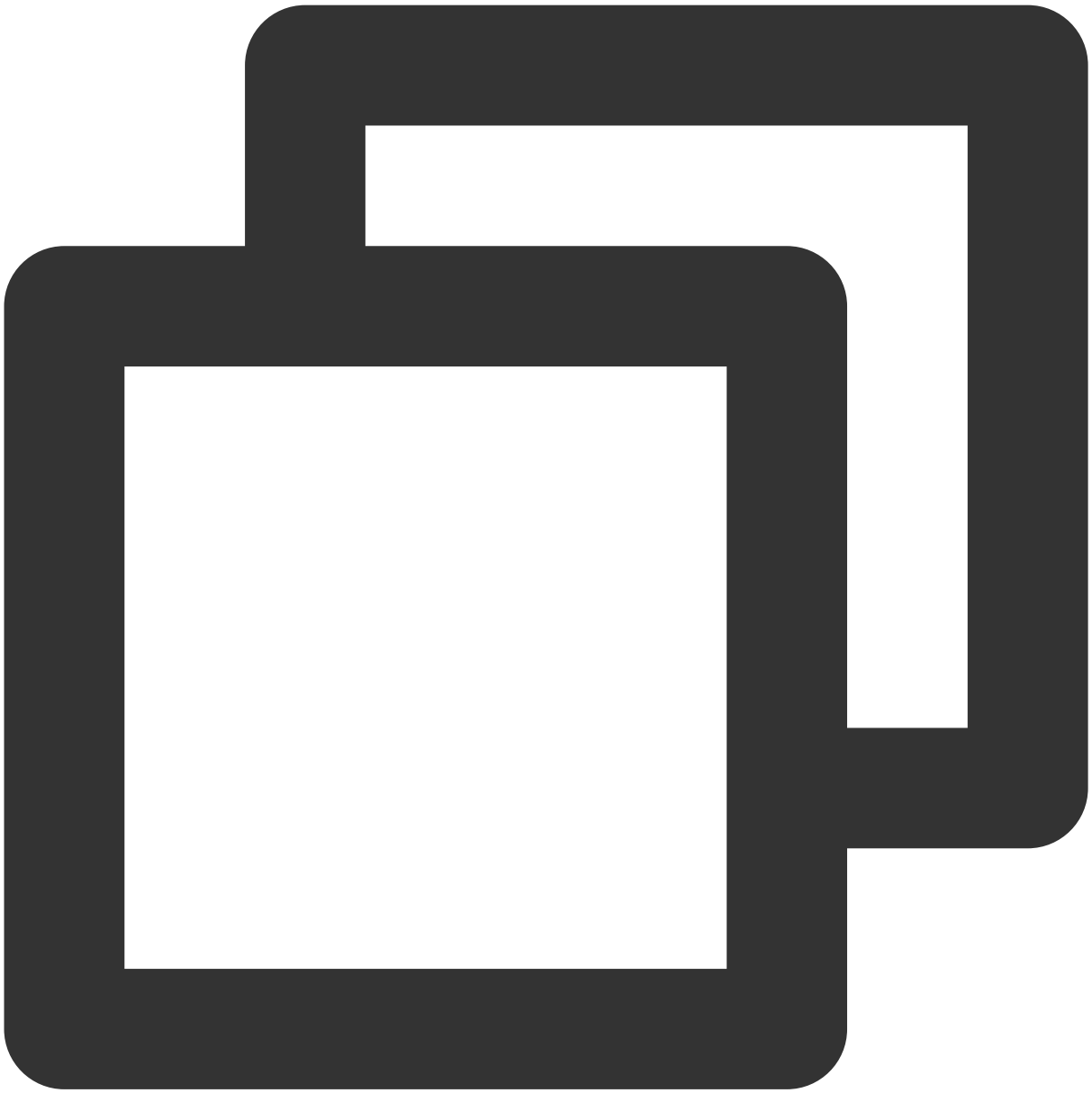
```
dense_rank()
```

支持引擎：SparkSQL、Presto

使用说明：计算某个值在一组值中的排名，如果出现排名相等，与函数rank不同，dense_rank不会在排名序列中产生空位。

返回类型：int

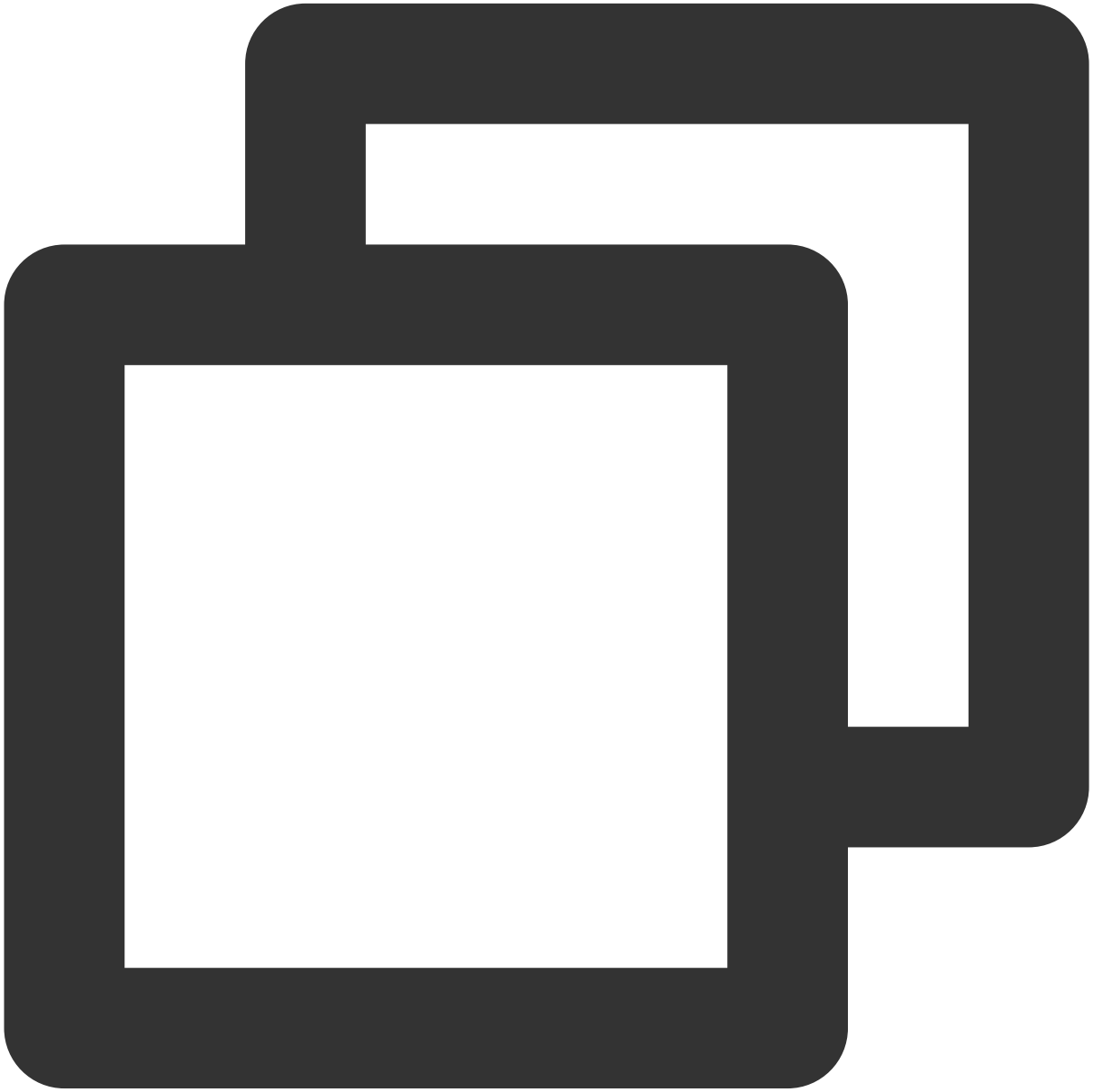
示例：



```
SELECT a, b, dense_rank() OVER (PARTITION BY a ORDER BY b) FROM VALUES ('A1', 2), (
A1      1       1
A1      1       1
A1      2       2
A2      3       1
```

percent_rank()

函数语法：



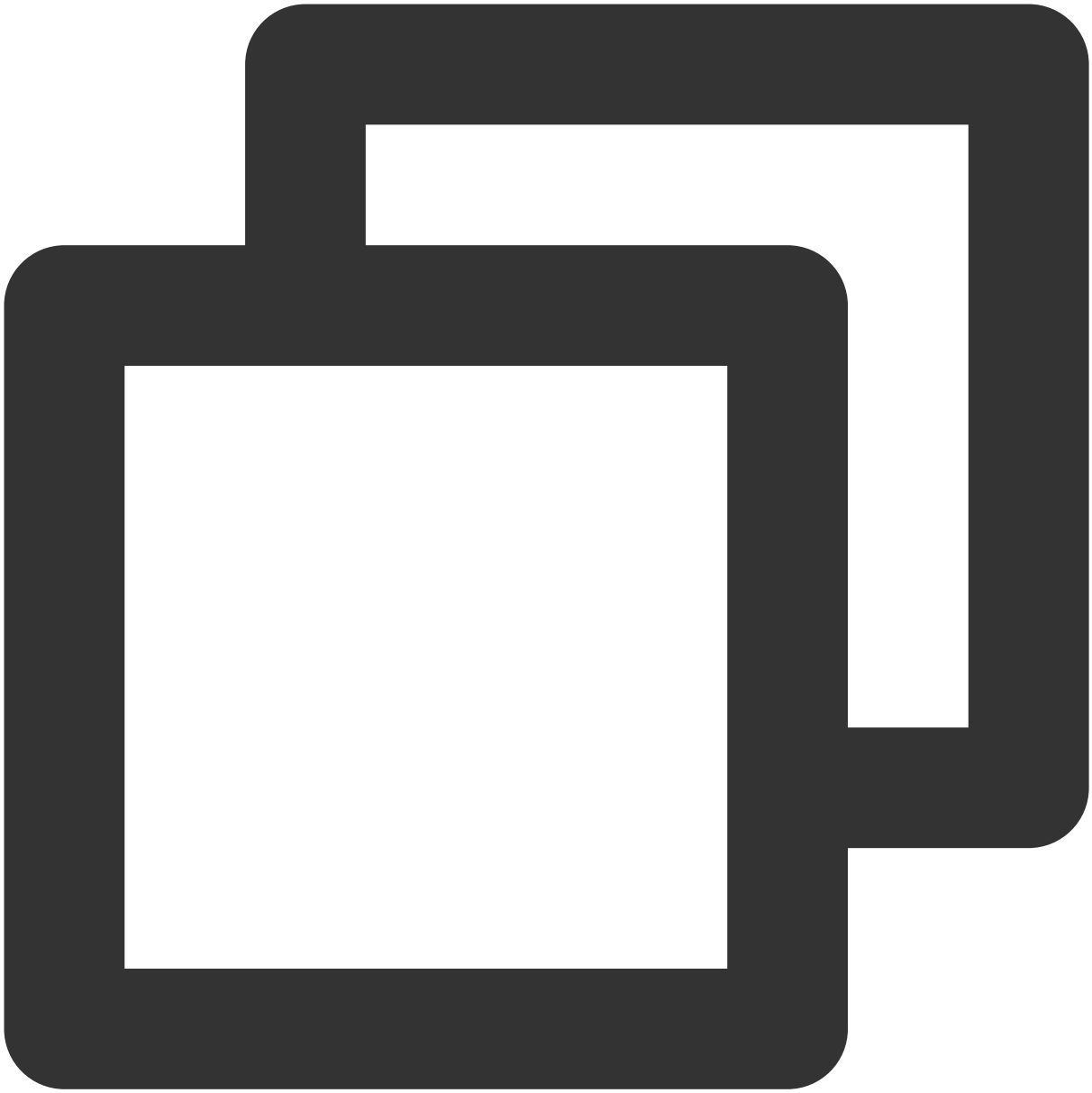
```
percent_rank()
```

支持引擎：SparkSQL、Presto

使用说明：计算某个值在一组值中的百分比排名。返回值以 0 到 1 之间的小数表示。

返回类型：double

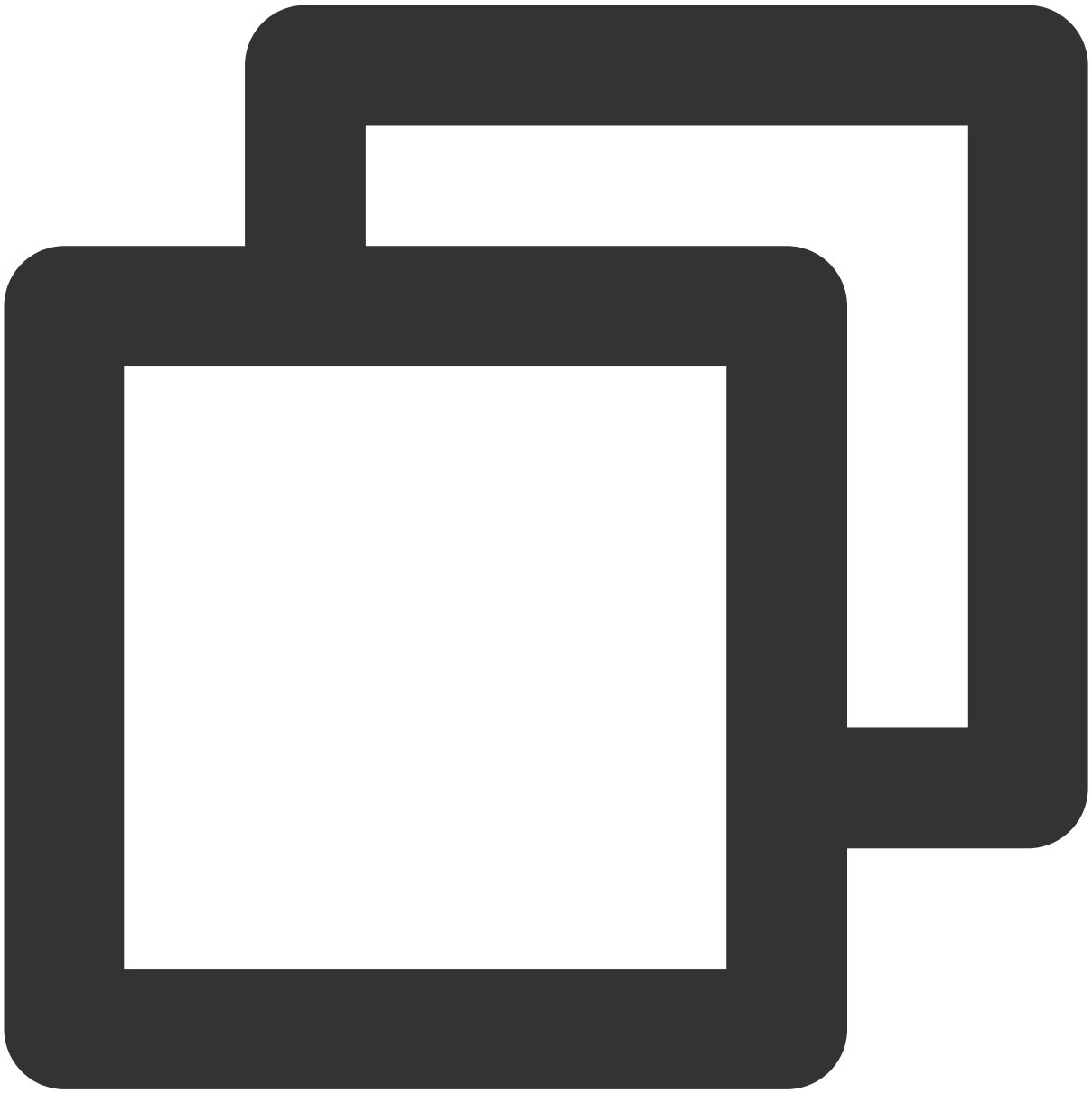
示例：



```
SELECT a, b, percent_rank() OVER (PARTITION BY a ORDER BY b) FROM VALUES ('A1', 2),
A1      1       0.0
A1      1       0.0
A1      2       1.0
A2      3       0.0
```

cume_dist

函数语法：



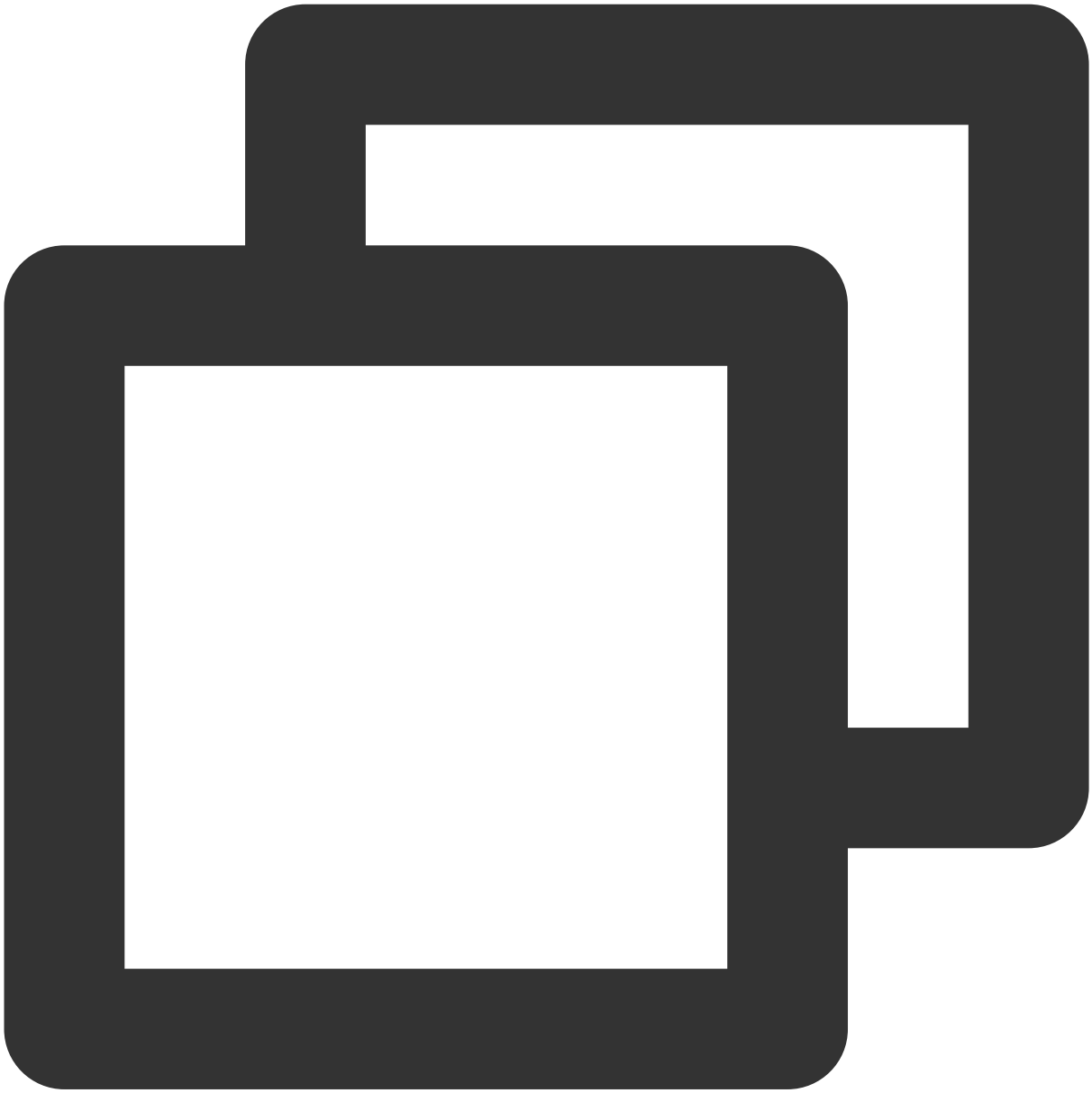
```
cume_dist()
```

支持引擎：SparkSQL、Presto

使用说明：计算某个值在分区中相对于所有值的位置。

返回类型：double

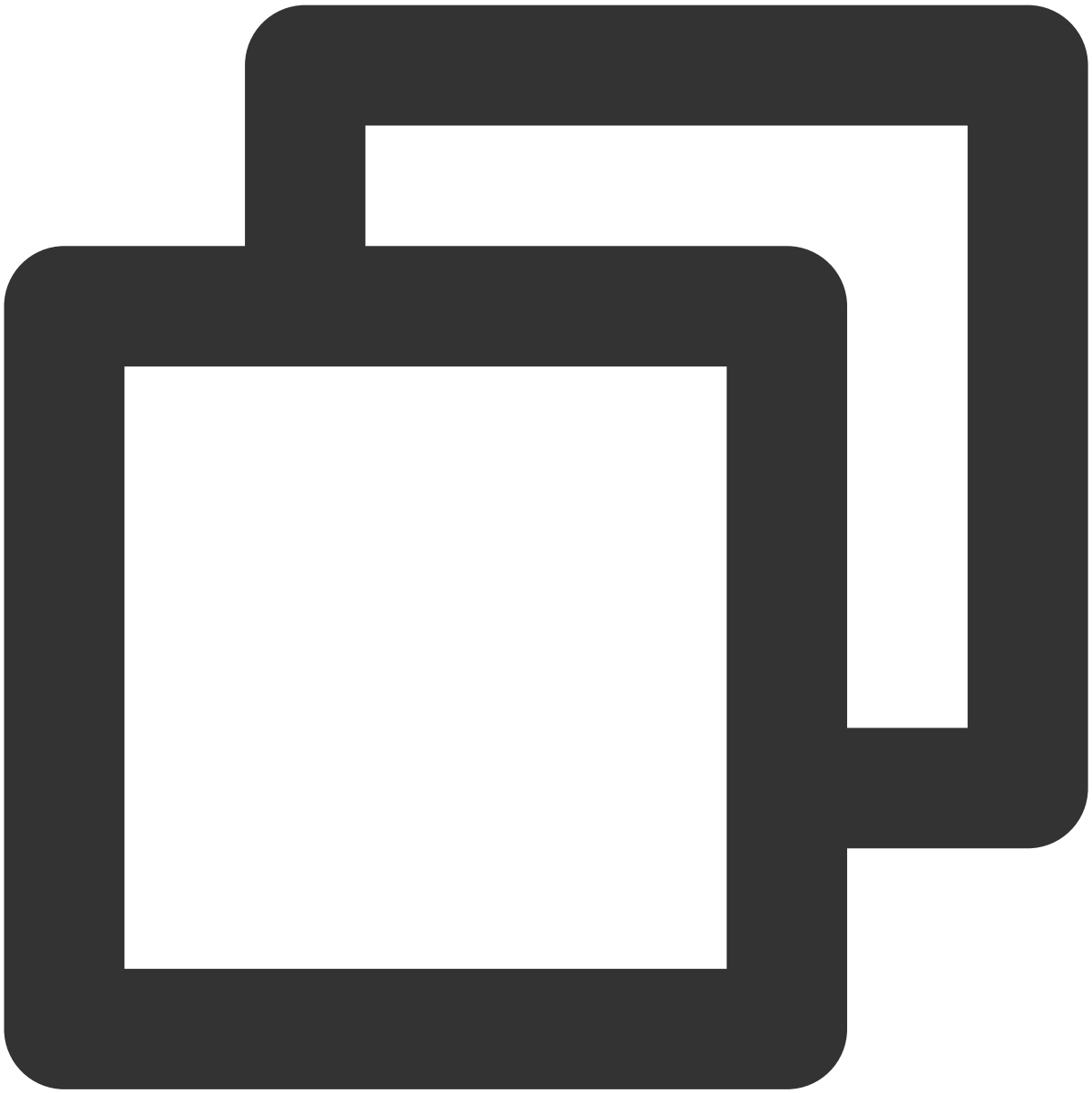
示例：



```
SELECT a, b, cume_dist() OVER (PARTITION BY a ORDER BY b)  FROM VALUES ('A1', 2), (
A1      1       0.6666666666666666
A1      1       0.6666666666666666
A1      2       1.0
A2      3       1.0
```

first_value

函数语法：



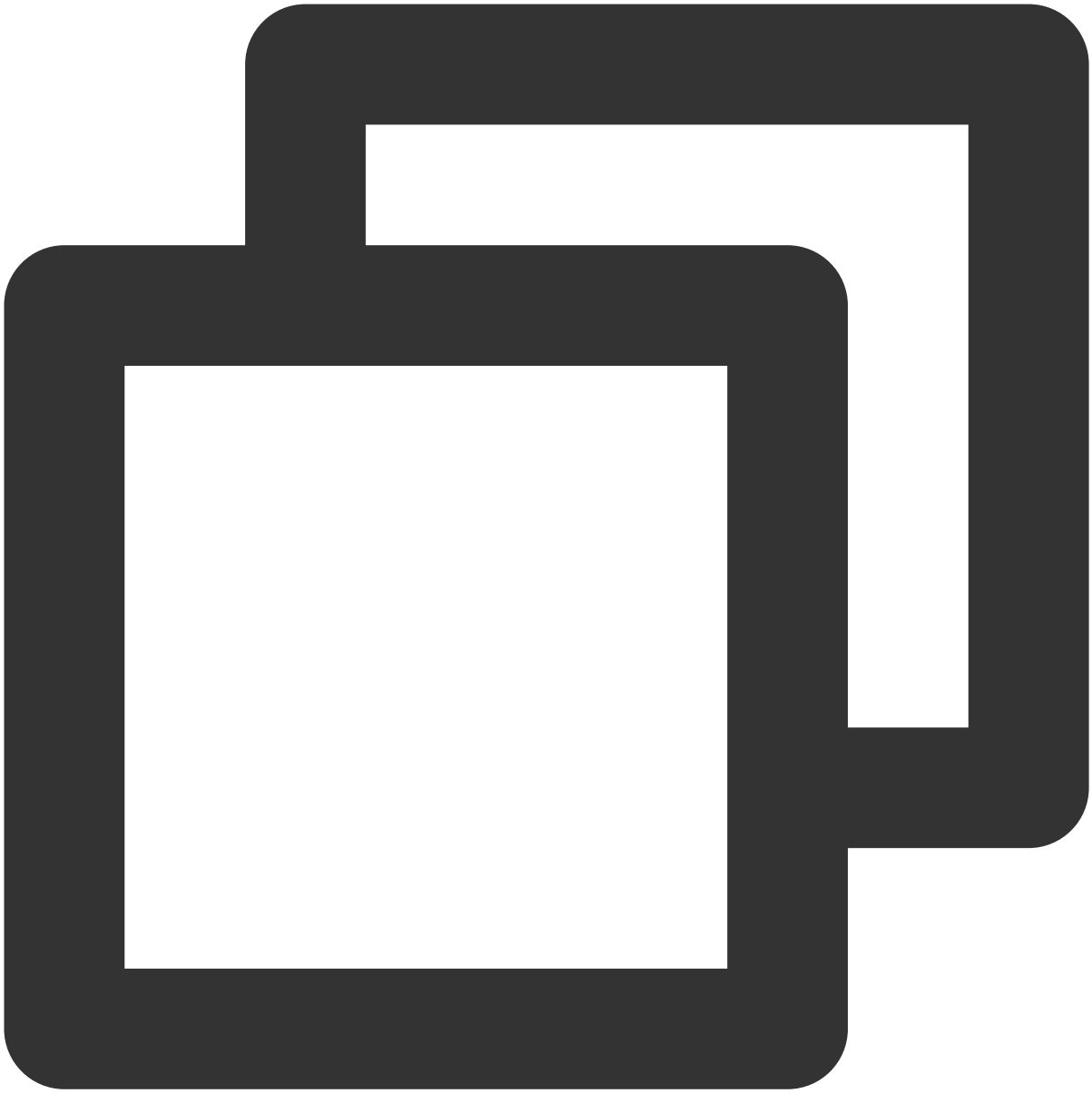
```
first_value(col)
```

支持引擎：SparkSQL、Presto

使用说明：返回分区中某列第一条数据的值

返回类型：col列的数据类型

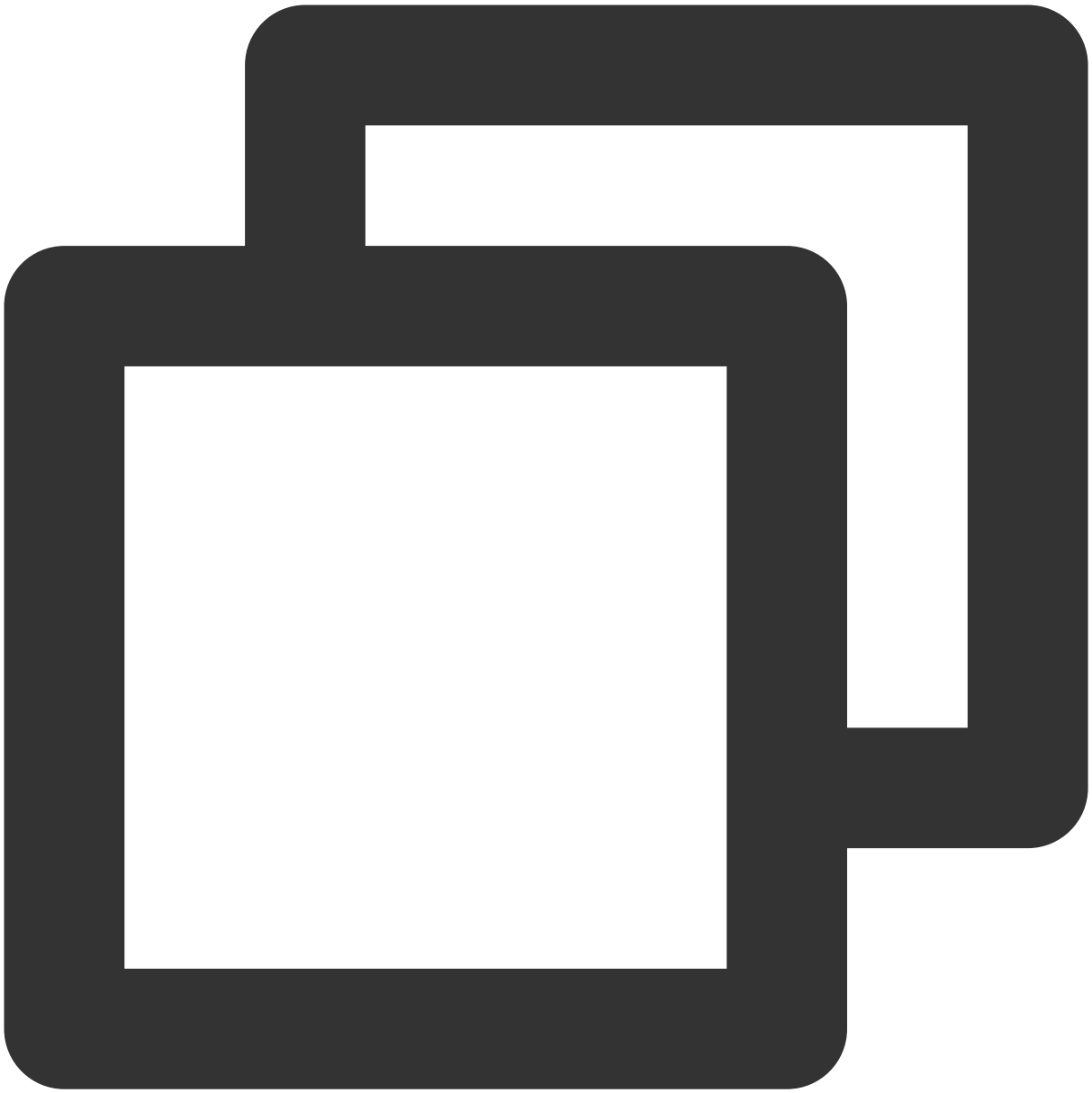
示例：



```
SELECT a, b, first_value(b) OVER (PARTITION BY a ORDER BY b) FROM VALUES ('A1', 2),
A1      1       1
A1      1       1
A1      2       1
A2      3       3
```

last_value

函数语法：



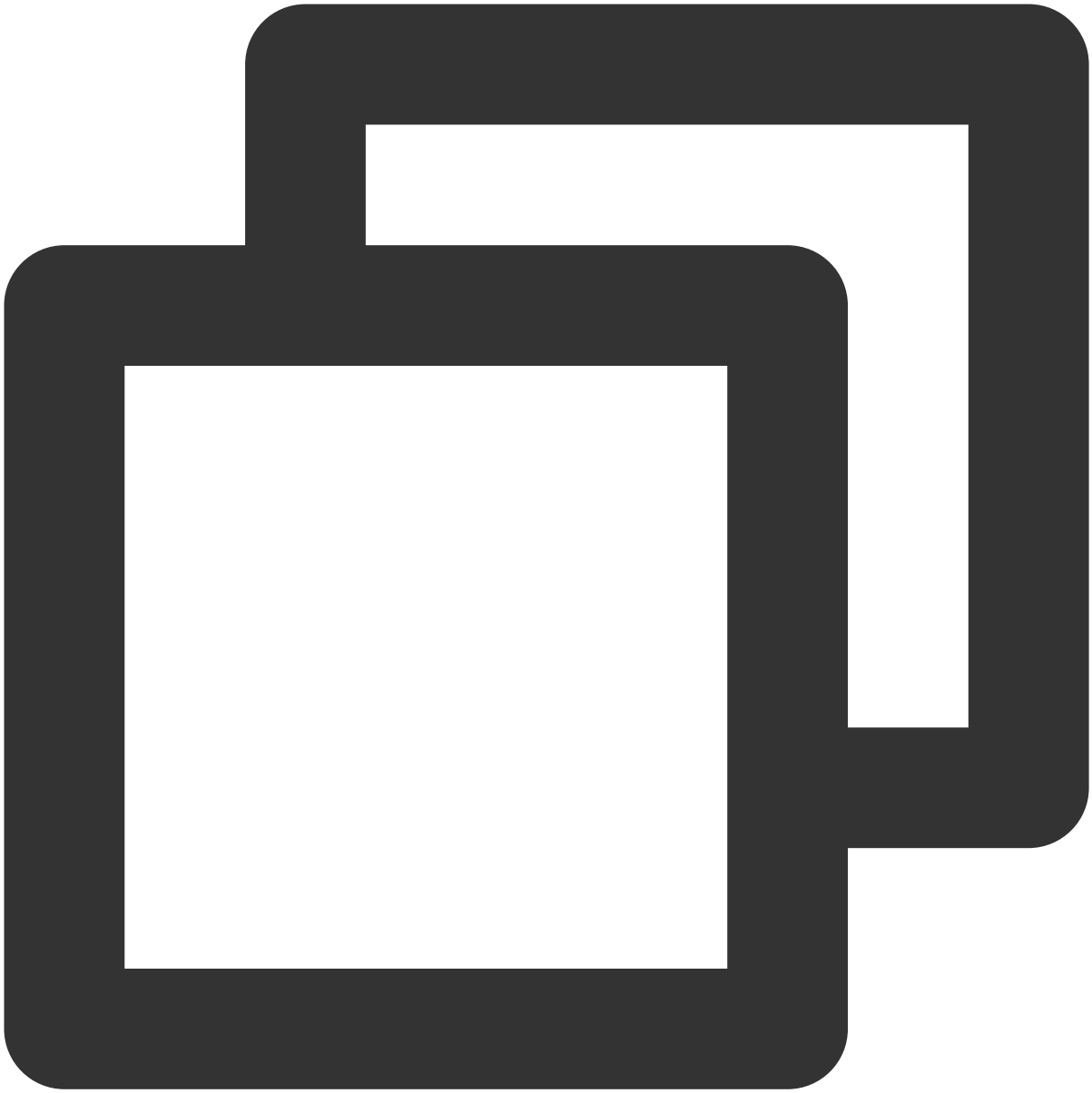
```
last_value(col)
```

支持引擎：SparkSQL、Presto

使用说明：返回分区中某列最后一条数据的值

返回类型：int

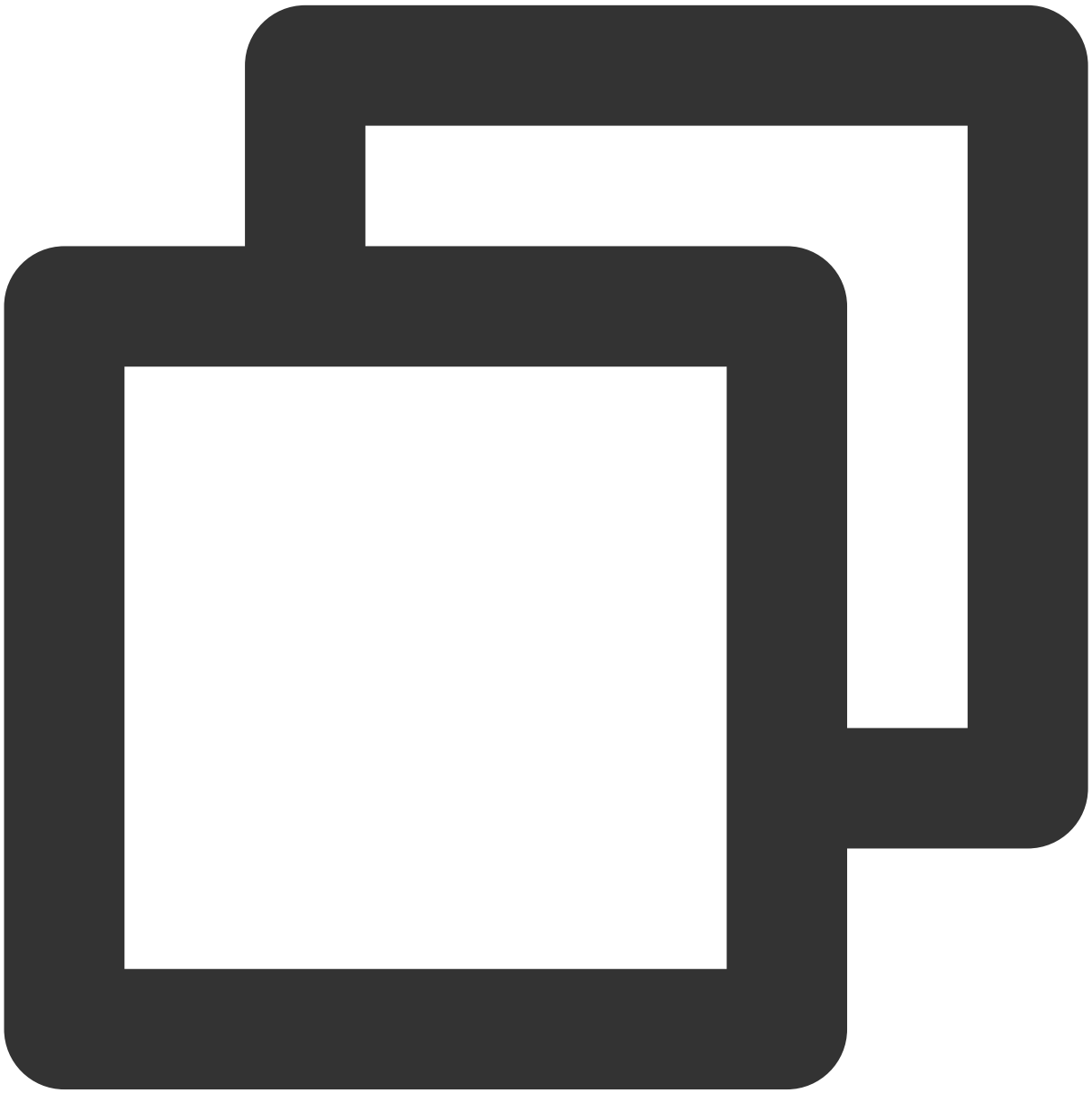
示例：



```
SELECT a, b, last_value(b) OVER (PARTITION BY a ORDER BY b) FROM VALUES ('A1', 2),
A1      1       1
A1      1       1
A1      2       2
A2      3       3
```

lag

函数语法：



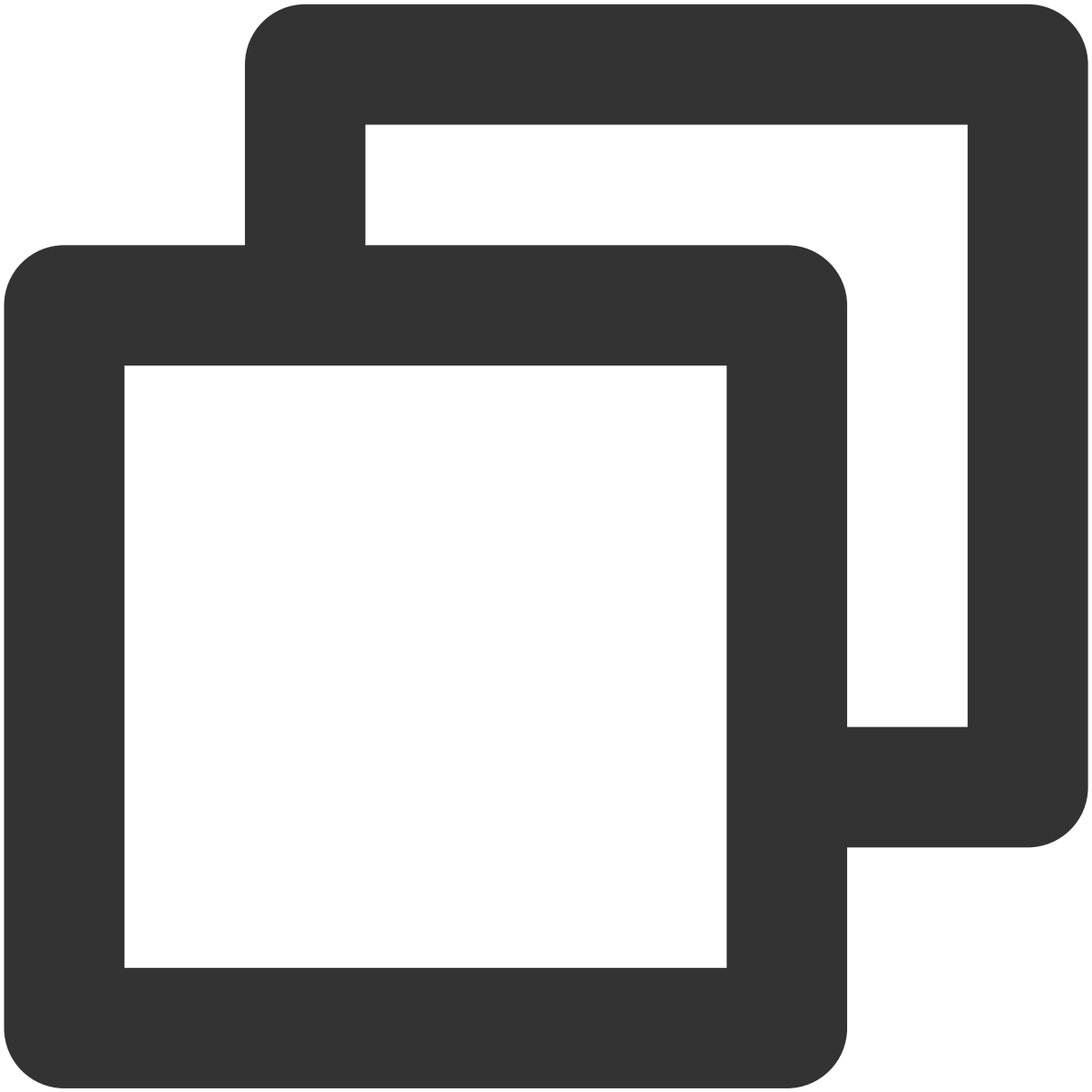
```
lag(col[, n [, default]])
```

支持引擎：SparkSQL、Presto

使用说明：返回窗口中当前行向上第 n 行的值。n 默认值为 1，default 默认值为 null。如果第 n 行的 值为 null，则返回 null。如果不存在这样的偏移行（例如，当偏移量为1时，窗口的第一行没有任何向上行），则返回 default 。第一个参数为列名，第二个参数为之前第 n 行，第三个参数为默认值。

返回类型：col列的数据类型

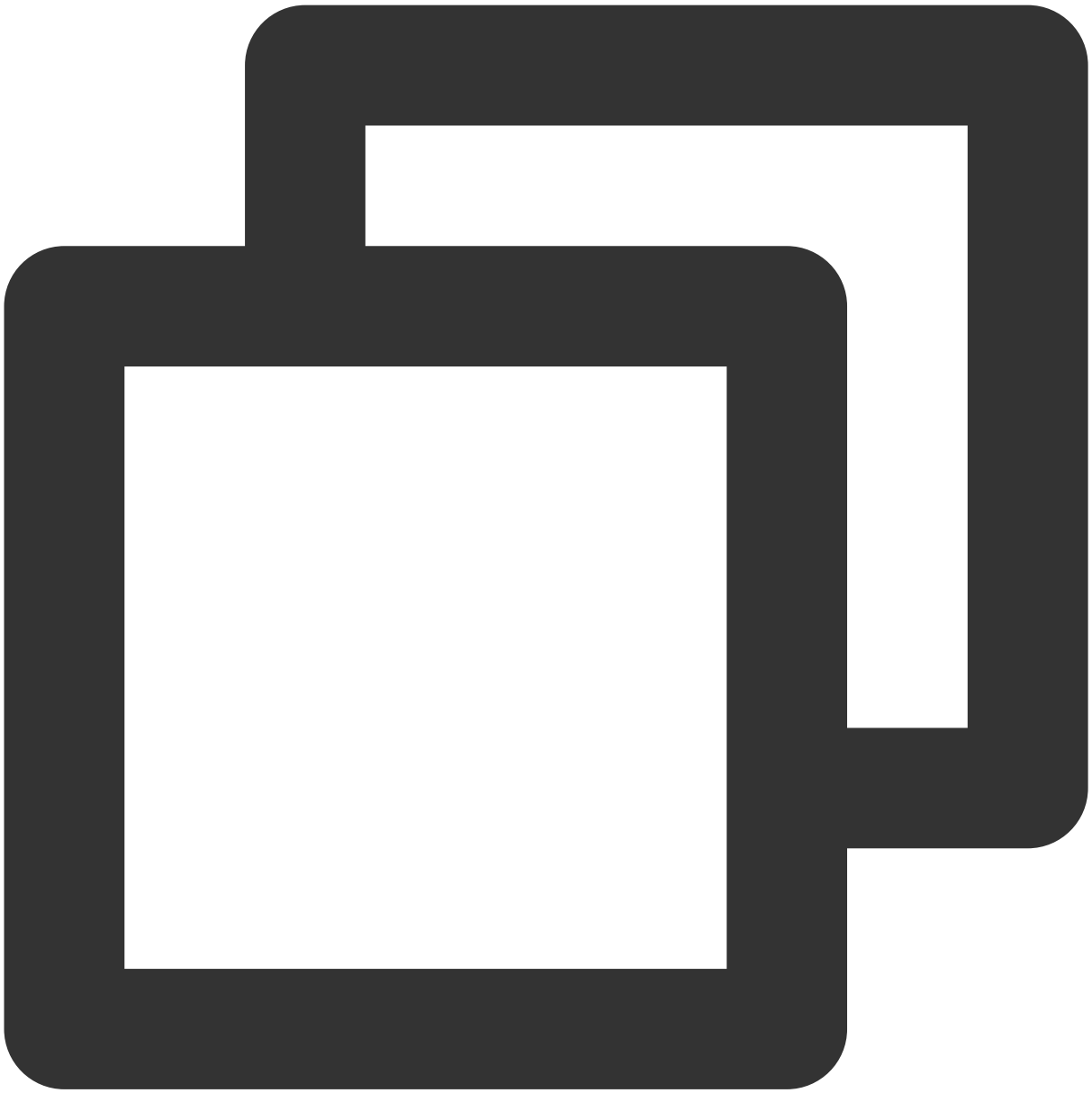
示例：



```
SELECT a, b, lag(b) OVER (PARTITION BY a ORDER BY b) FROM VALUES ('A1', 2), ('A1',
A1      1       NULL
A1      1       1
A1      2       1
A2      3       NULL
```

lead

函数语法：



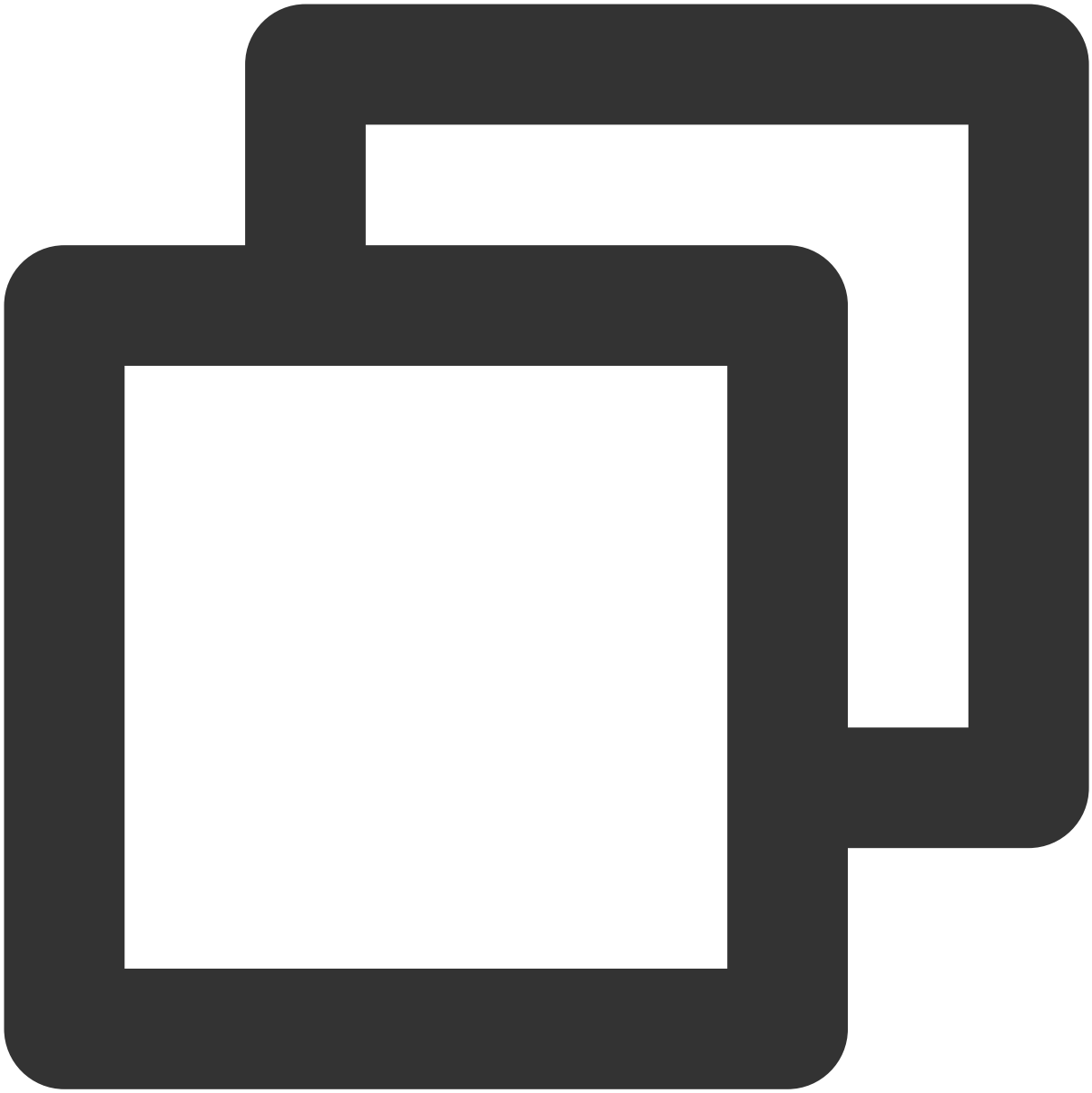
```
lead(col[, n[, default]])
```

支持引擎：SparkSQL、Presto

使用说明：返回窗口中当前行向下第 n 行的值。n 默认值为 1，default 默认值为 null。如果第 n 行的 值为 null，则返回 null。如果不存在这样的偏移行（例如，当偏移量为1时，窗口的最后一行没有任何向下行），则返回 default 。第一个参数为列名，第二个参数为之前第 n 行，第三个参数为默认值。

返回类型：col列的数据类型

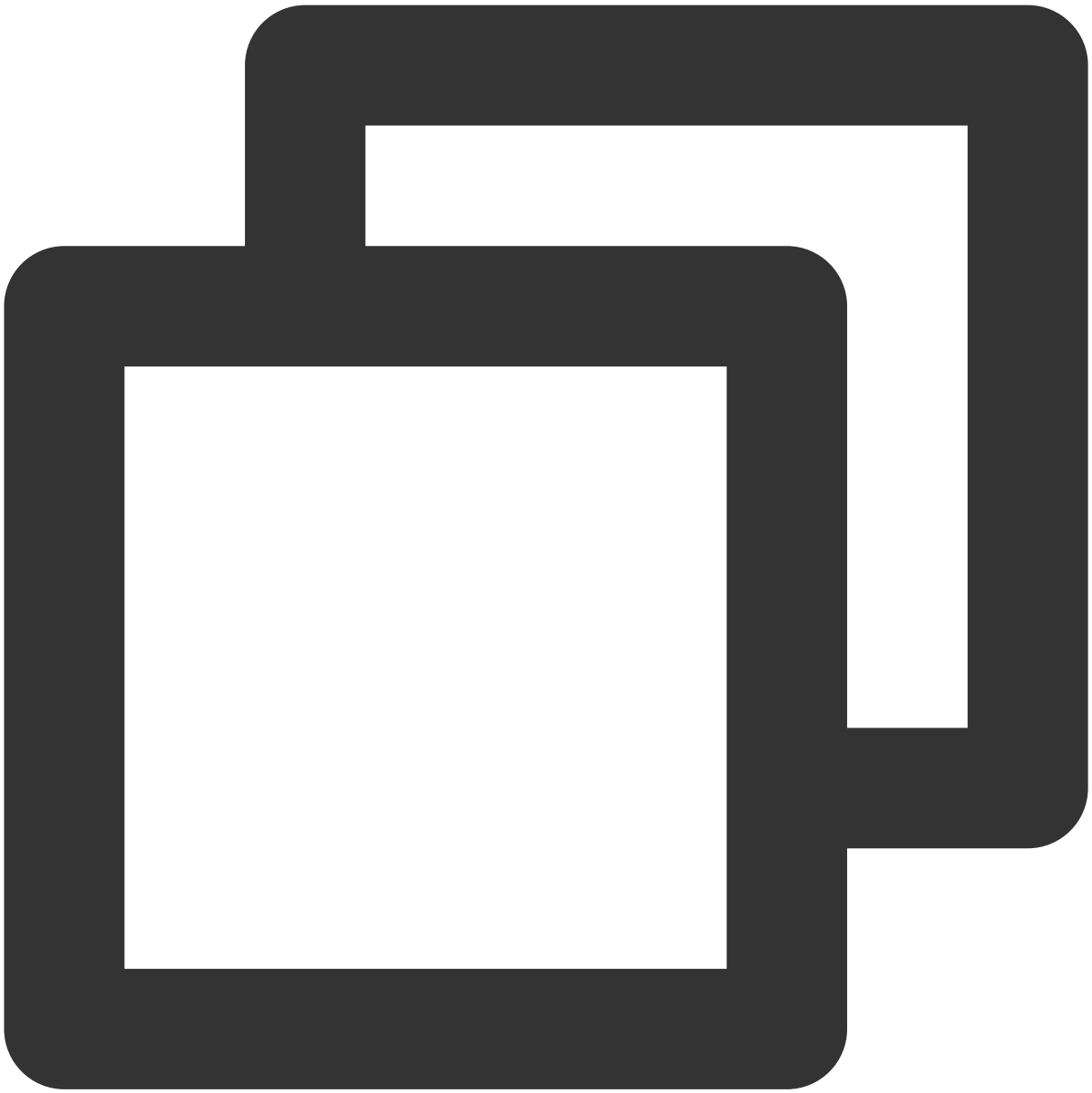
示例：



```
SELECT a, b, lead(b) OVER (PARTITION BY a ORDER BY b) FROM VALUES ('A1', 2), ('A1',
A1      1       1
A1      1       2
A1      2       NULL
A2      3       NULL
```

nth_value

函数语法：



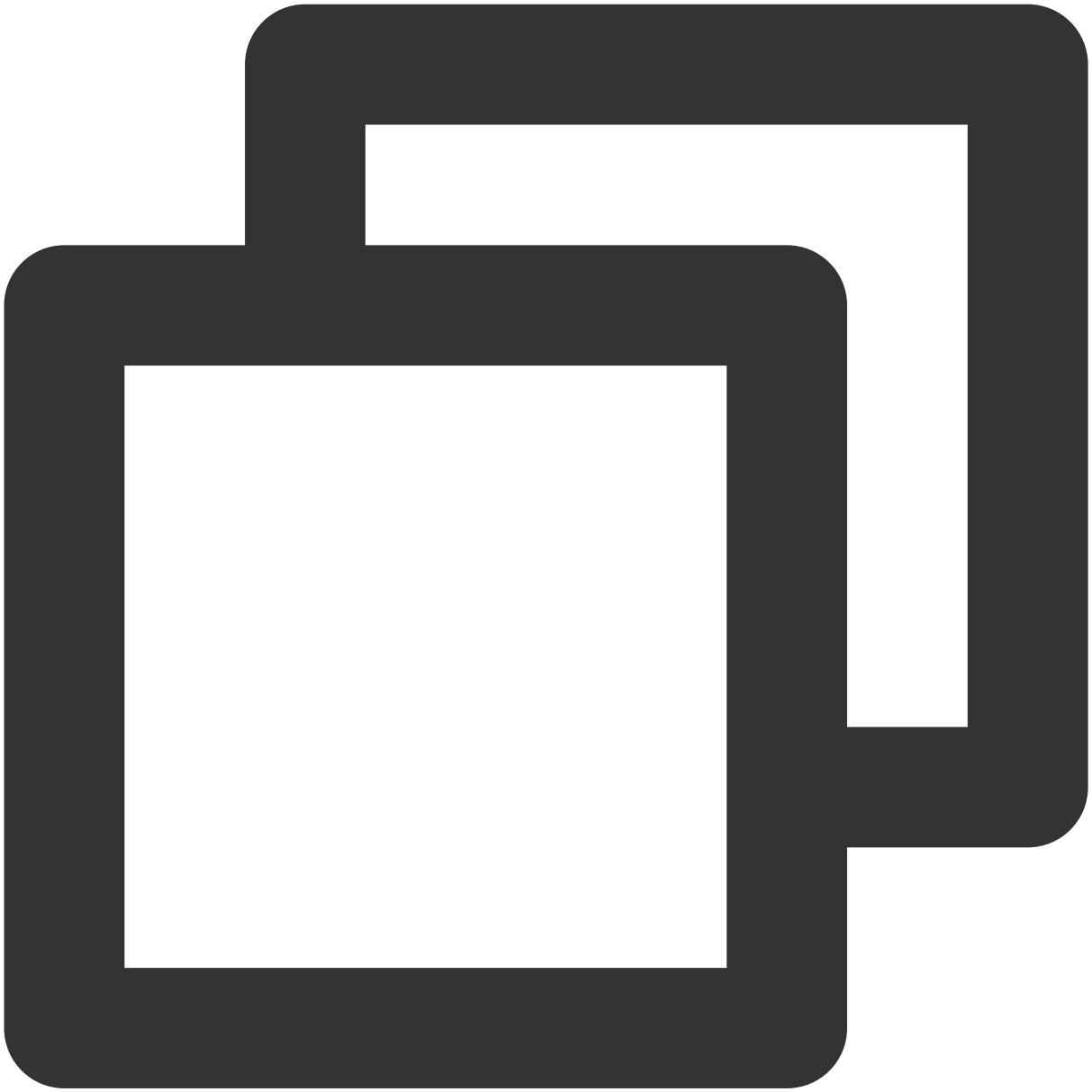
```
nth_value(col[, n])
```

支持引擎：SparkSQL、Presto

使用说明：返回距窗口头第 n 行的值。n 从 1 开始。如果ignoreNulls=true，查找第 n 行时将跳过 null。否则，每一行都计入 n。如果不存在这样的第 n 行（例如，当 n 为10时，窗口大小小于10），则返回null。第一个参数为列名，第二个参数为之前第 n 行。

返回类型：col列的数据类型

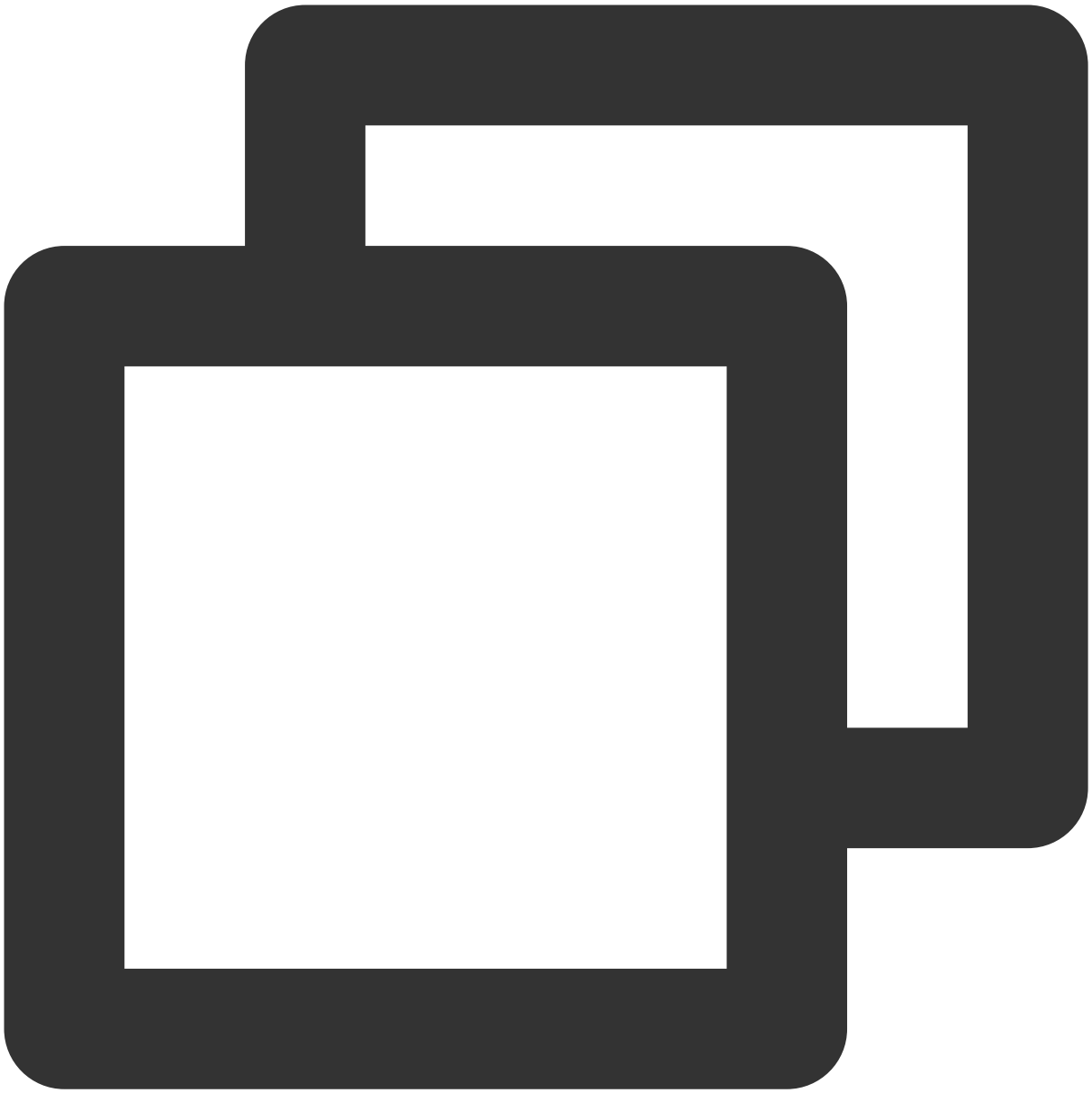
示例：



```
SELECT a, b, nth_value(b, 2) OVER (PARTITION BY a ORDER BY b) FROM VALUES ('A1', 2)
A1      1       1
A1      1       1
A1      2       1
A2      3       NULL
```

ntile

函数语法：



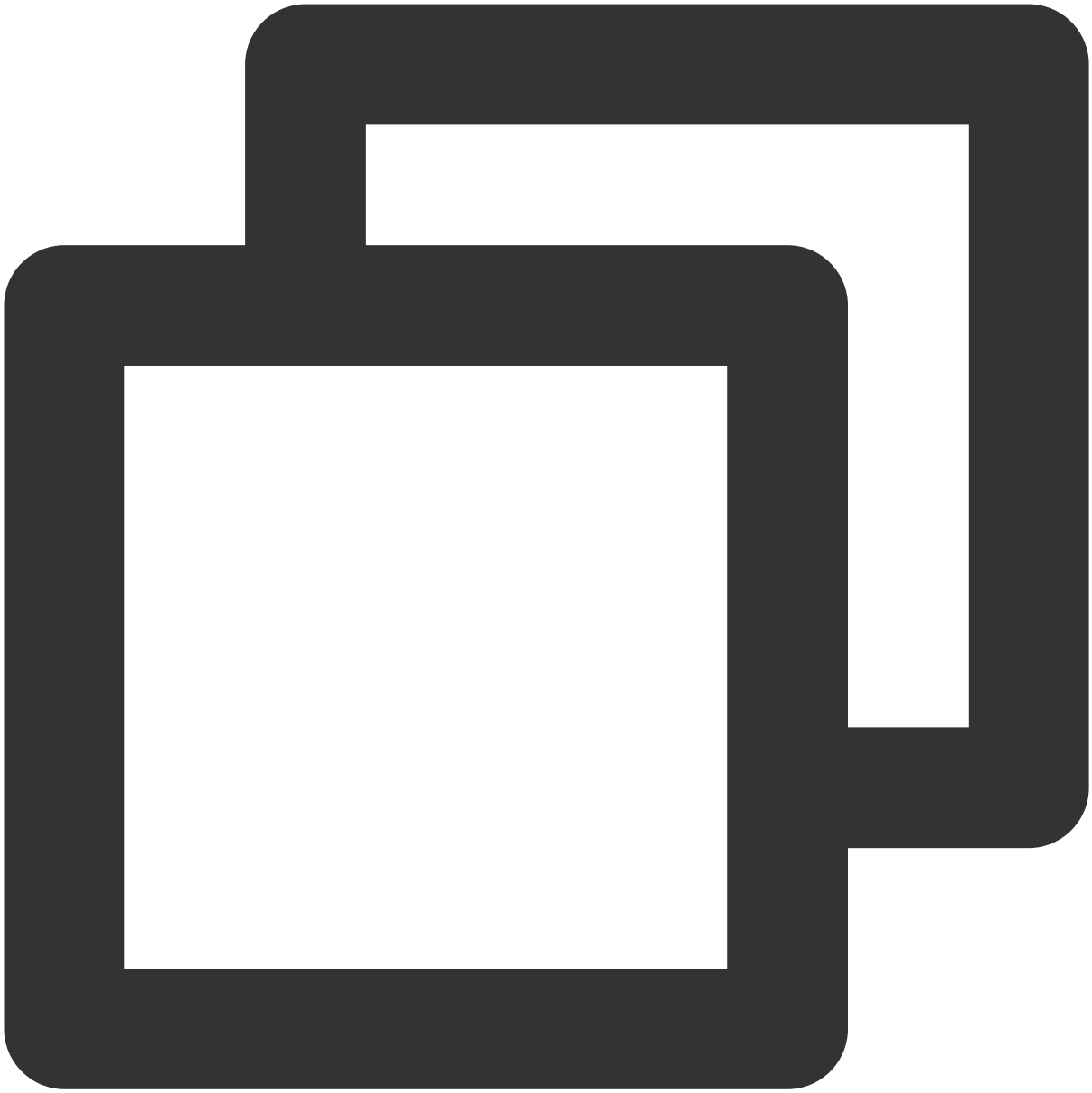
```
ntile(n)
```

支持引擎：SparkSQL、Presto

使用说明：将窗口分区的行划分为 n 个桶，返回行所在的桶数，范围从 1 到 n

返回类型：int

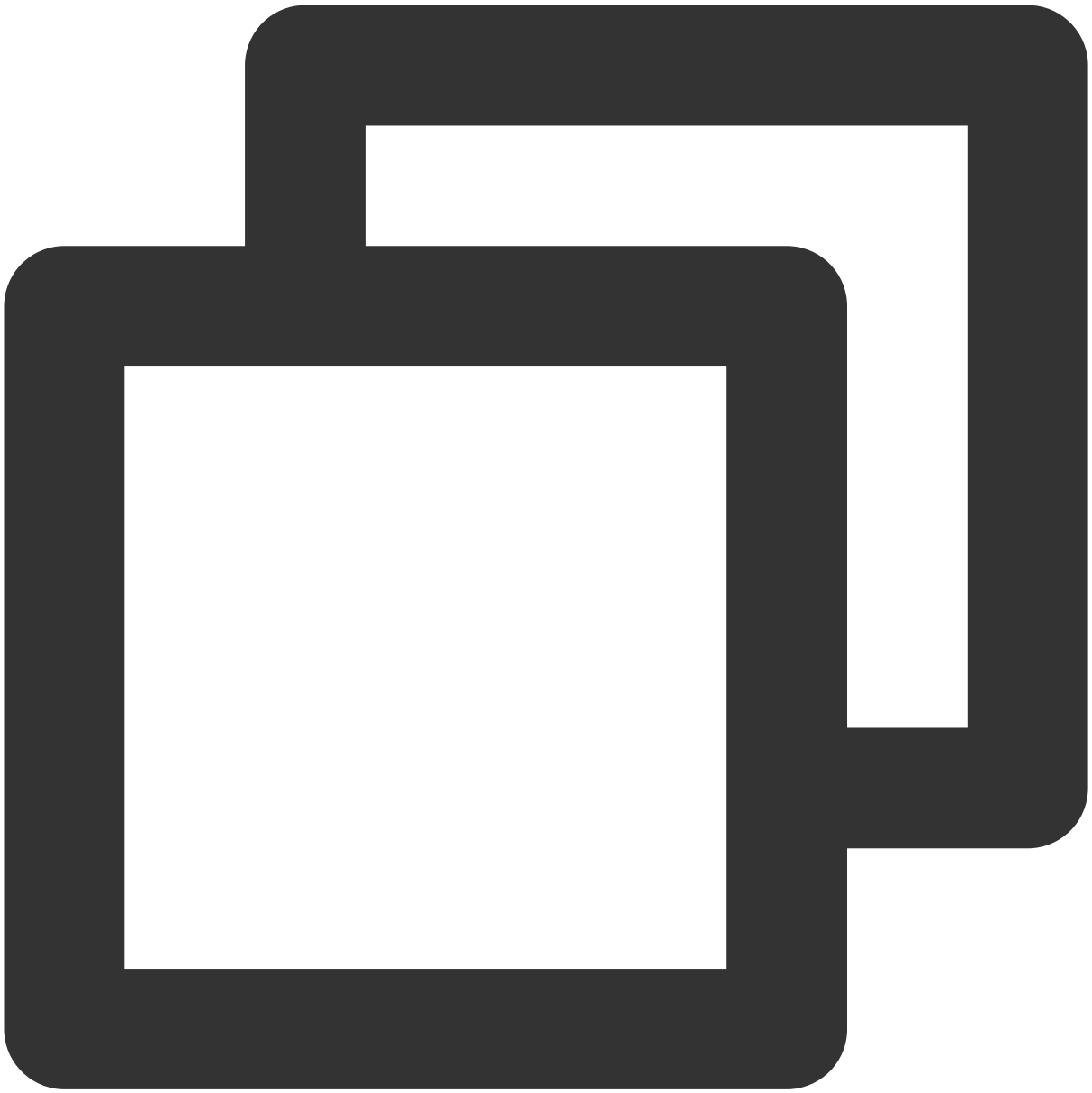
示例：



```
SELECT a, b, ntile(2) OVER (PARTITION BY a ORDER BY b) FROM VALUES ('A1', 2), ('A1'
A1      1       1
A1      1       1
A1      2       2
A2      3       1
```

CLUSTER_SAMPLE

函数语法：



```
CLUSTER_SAMPLE(<int> N[, <int> M]) over (PARTITION BY col1 ORDER by col2)
```

支持引擎：SparkSQL

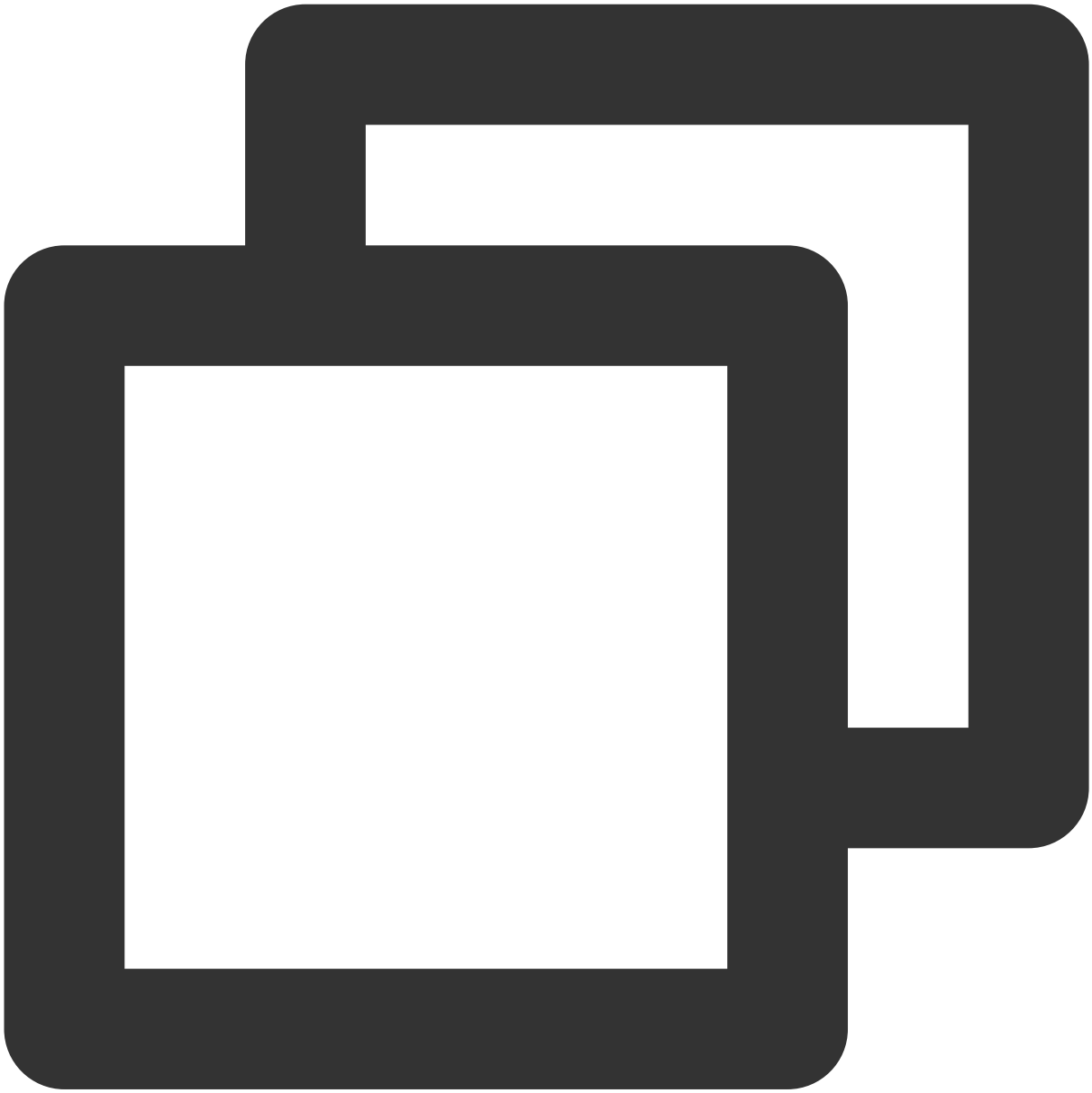
使用说明：在窗口内按指定的比例或数量采样。

N：必填，int 类型，当只有 N 时，表示采样出 N 条数据。采样结果接近 N 条，但不保证一定是 N 条。

M：可填，int 类型，当 M 被指定时，表示采样出 M/N * 窗口内总条数数量的数据。采样结果接近 M/N * 总条数。

返回类型：boolean 类型，true 表示被采样，false 表示未采样。

示例：



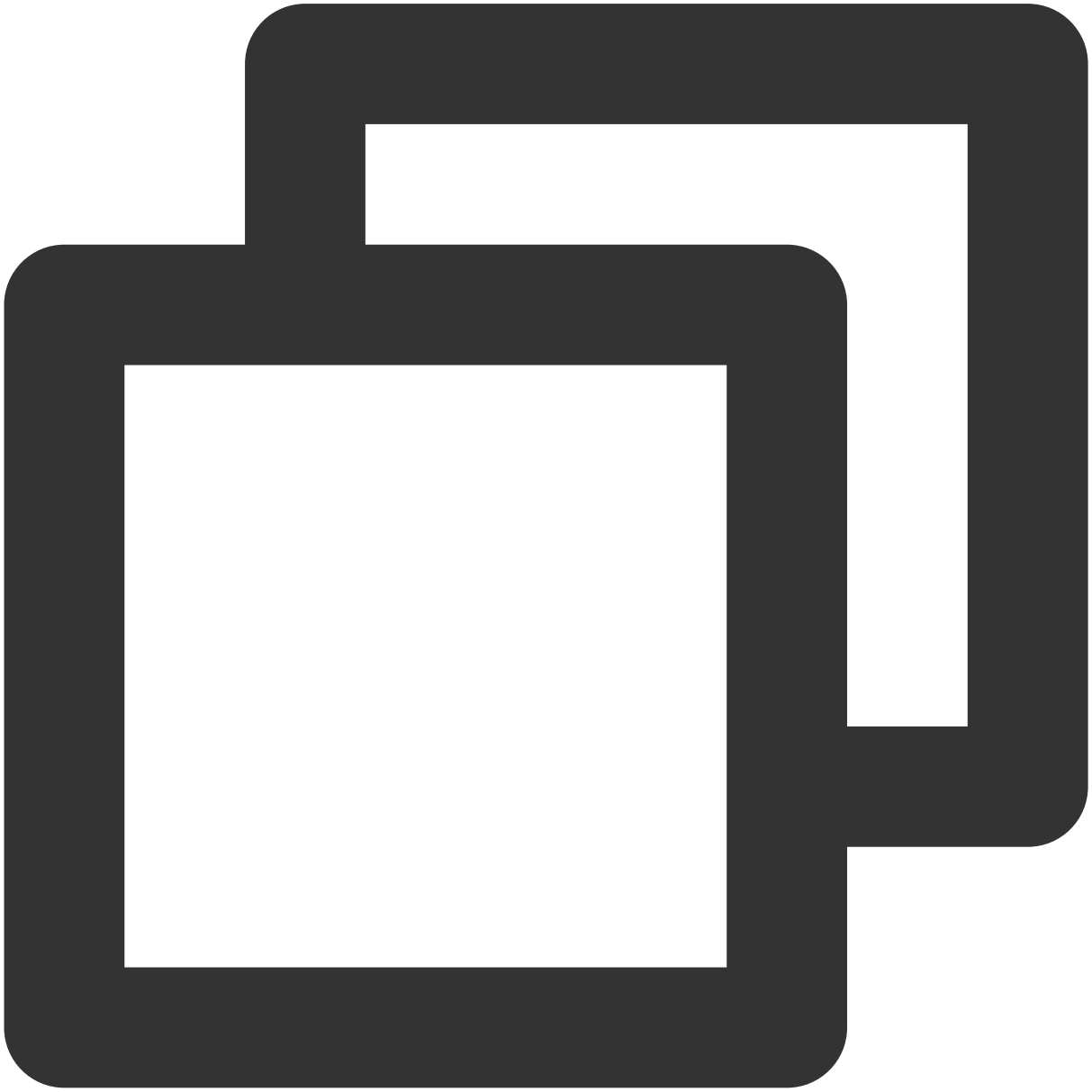
```
> SELECT a, b, cluster_sample(2) OVER (PARTITION BY a ORDER BY b) FROM VALUES ('A1'
A1 2 true
A1 1 true
A2 3 true
A1 1 false
> SELECT a, b from (select a, b, cluster_sample(2) OVER (PARTITION BY a ORDER BY b)
A1 2
A1 1
A2 3
```

其他函数

最近更新时间：2024-08-07 17:33:46

FIELD

函数语法：



```
FIELD(<val> T, <val1> T, <val2> T, ...)
```

支持引擎：Presto

使用说明：返回 val1、val2...列表中 val 的索引，如果未找到则返回0。

支持所有基元类型，使用 str.equals（x）比较参数。如果 val 为 NULL，则返回值为0。

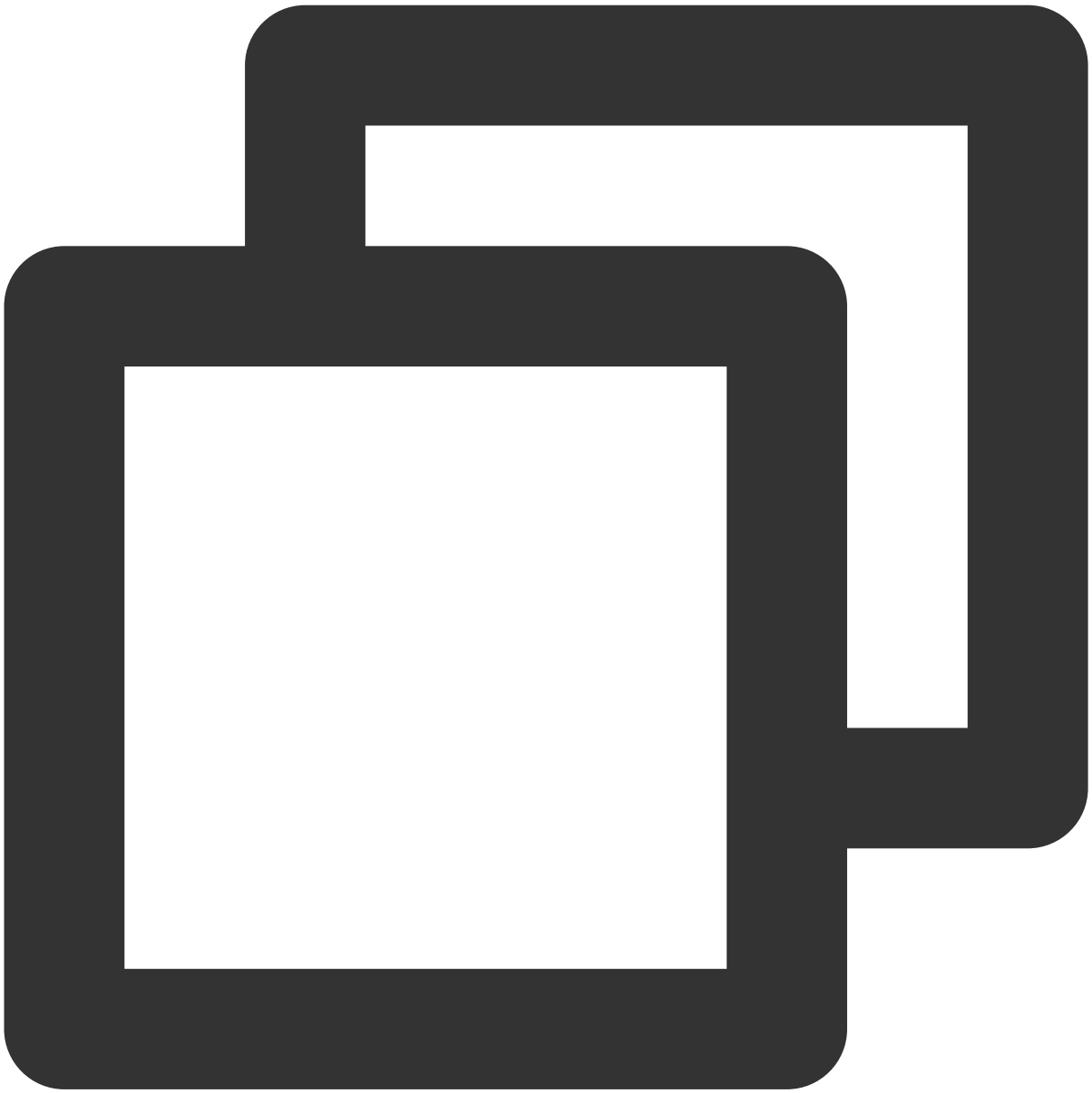
返回类型：integer

示例：

```
select field('world', 'say', 'hello', 'world');
 3
```

COALESCE

函数语法：



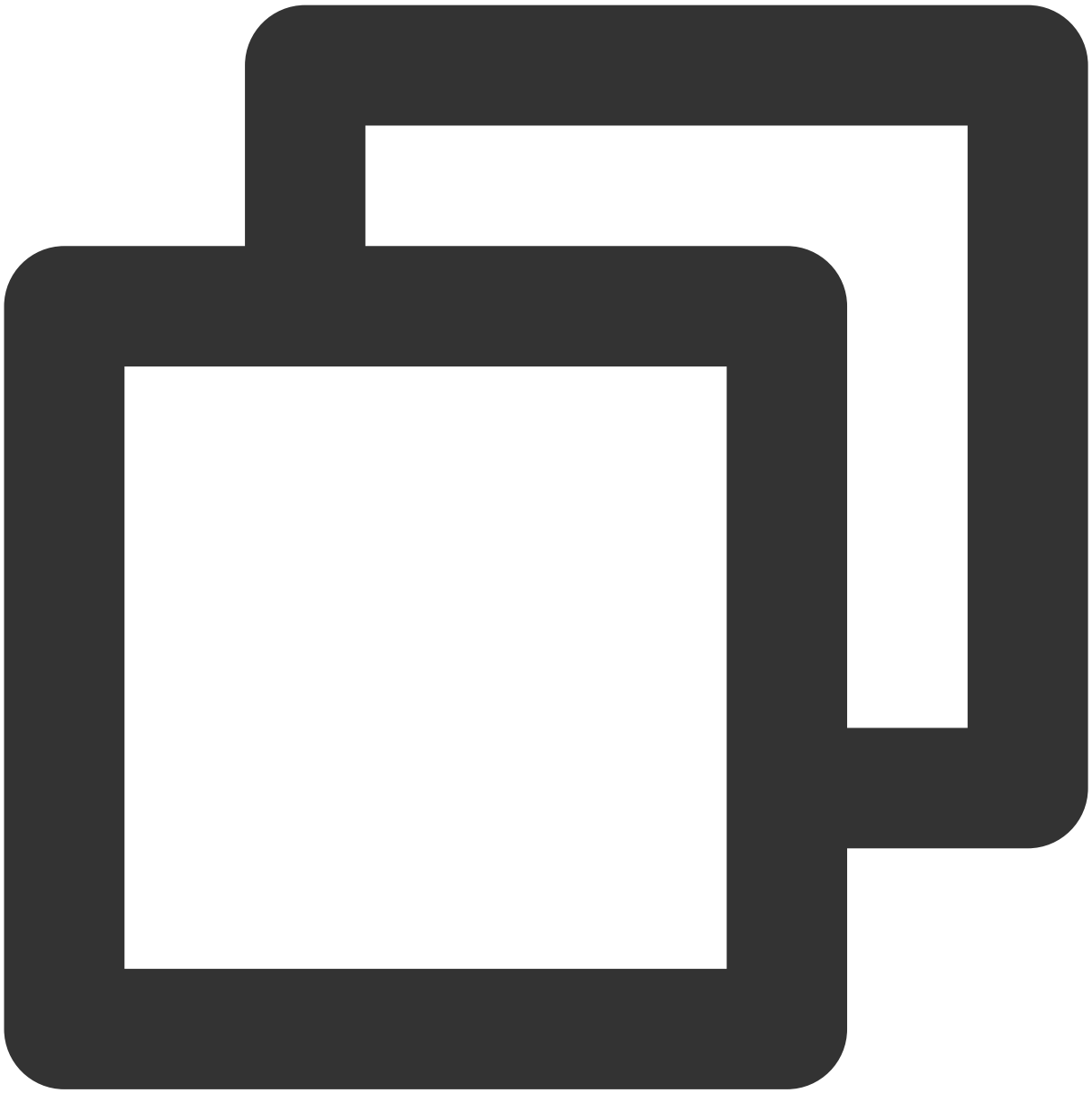
```
COALESCE(<expr1> T, <expr2> T)
```

支持引擎：SparkSQL、Presto

使用说明：如果存在，则返回第一个非空参数。否则返回 null。

返回类型：integer

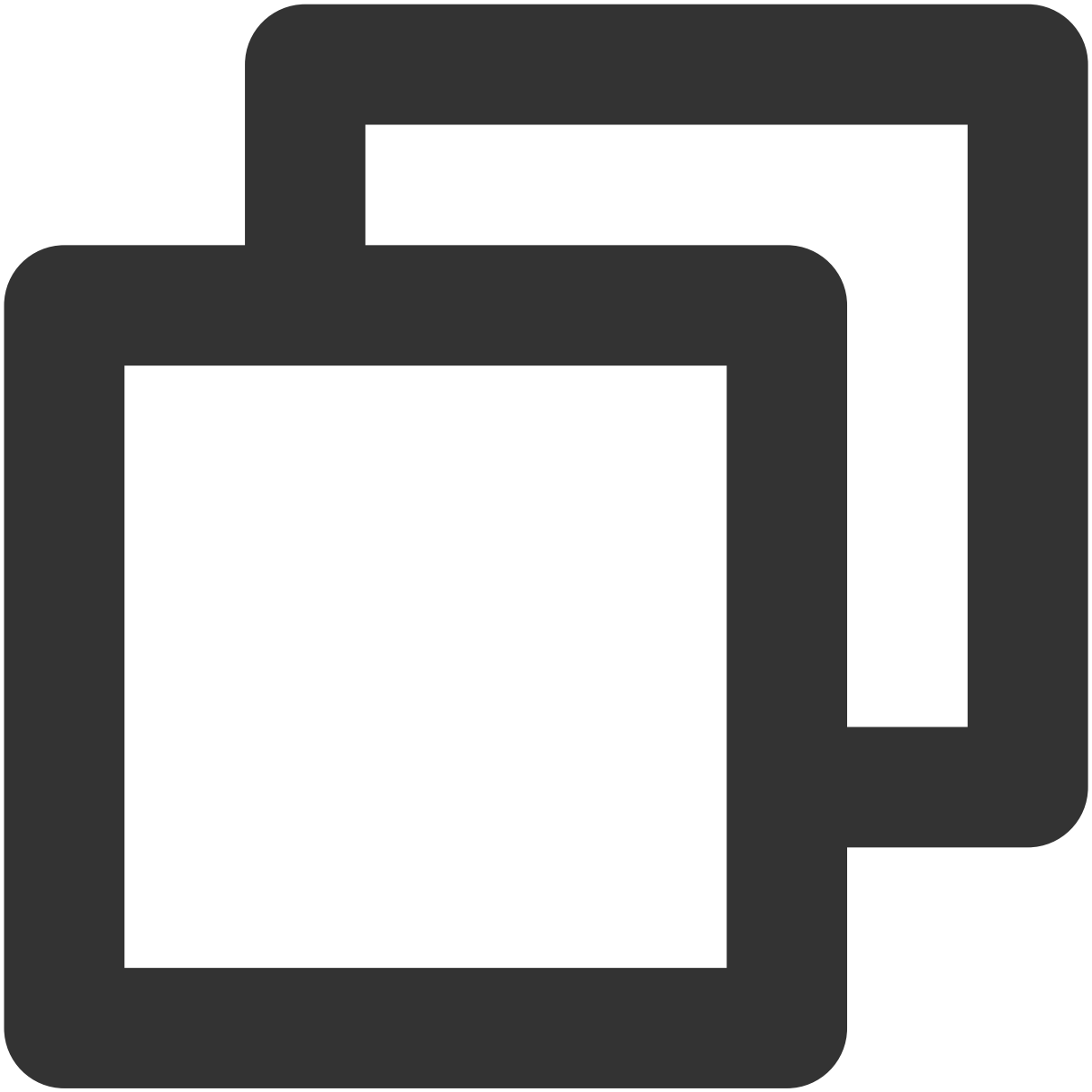
示例：



```
> SELECT coalesce(NULL, 1, NULL);
 1
```

EXPLODE

函数语法：



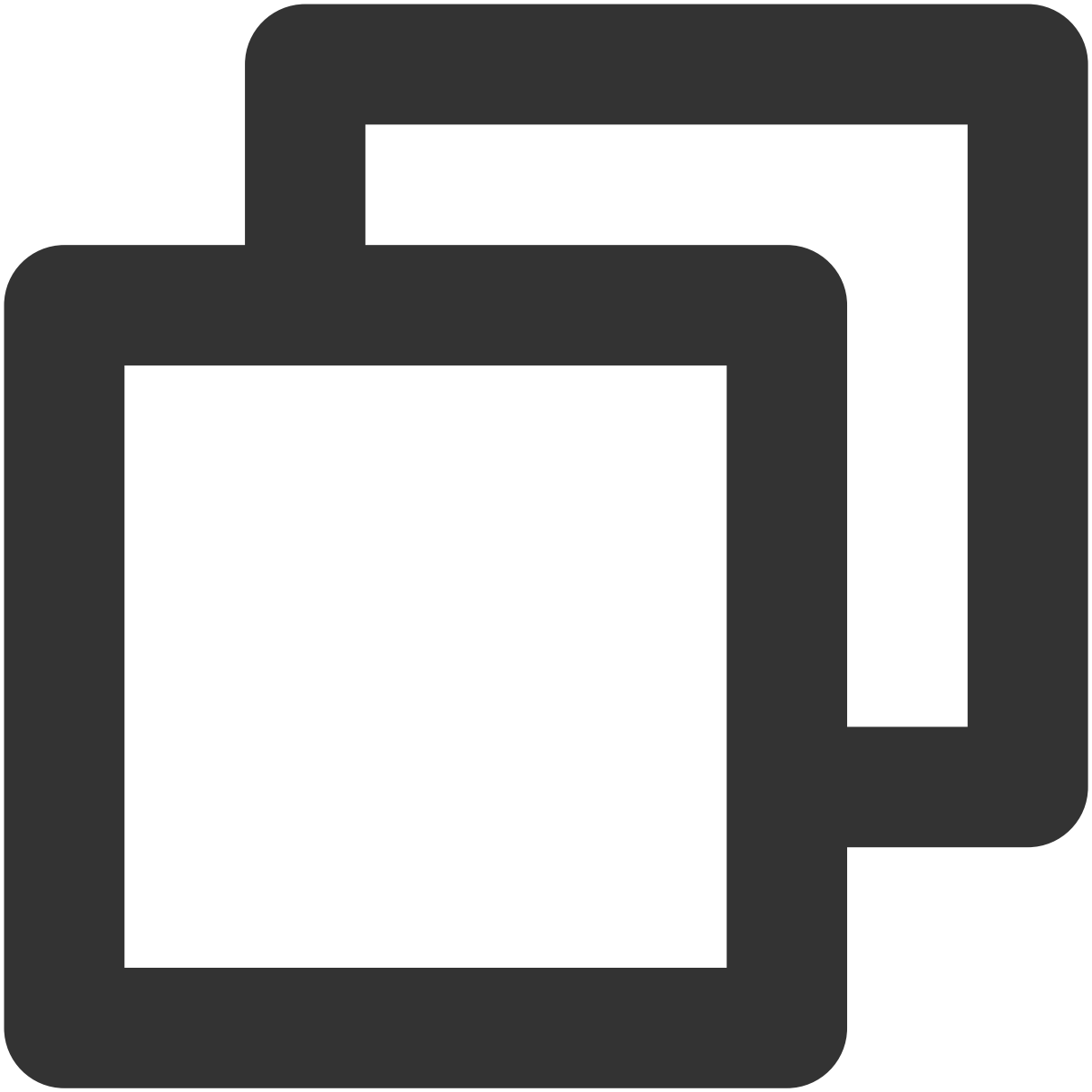
```
EXPLODE(<expr> array<T>|map<K, V>)
```

支持引擎：SparkSQL

使用说明：将 array 类型的 expr 的元素分隔为多行，或将 map 类型的 expr 分隔为多个行和列。对数组的元素使用默认列名 col，或对映射的元素使用 key 和 value。

返回类型：row(col T) | row(key K, value V)

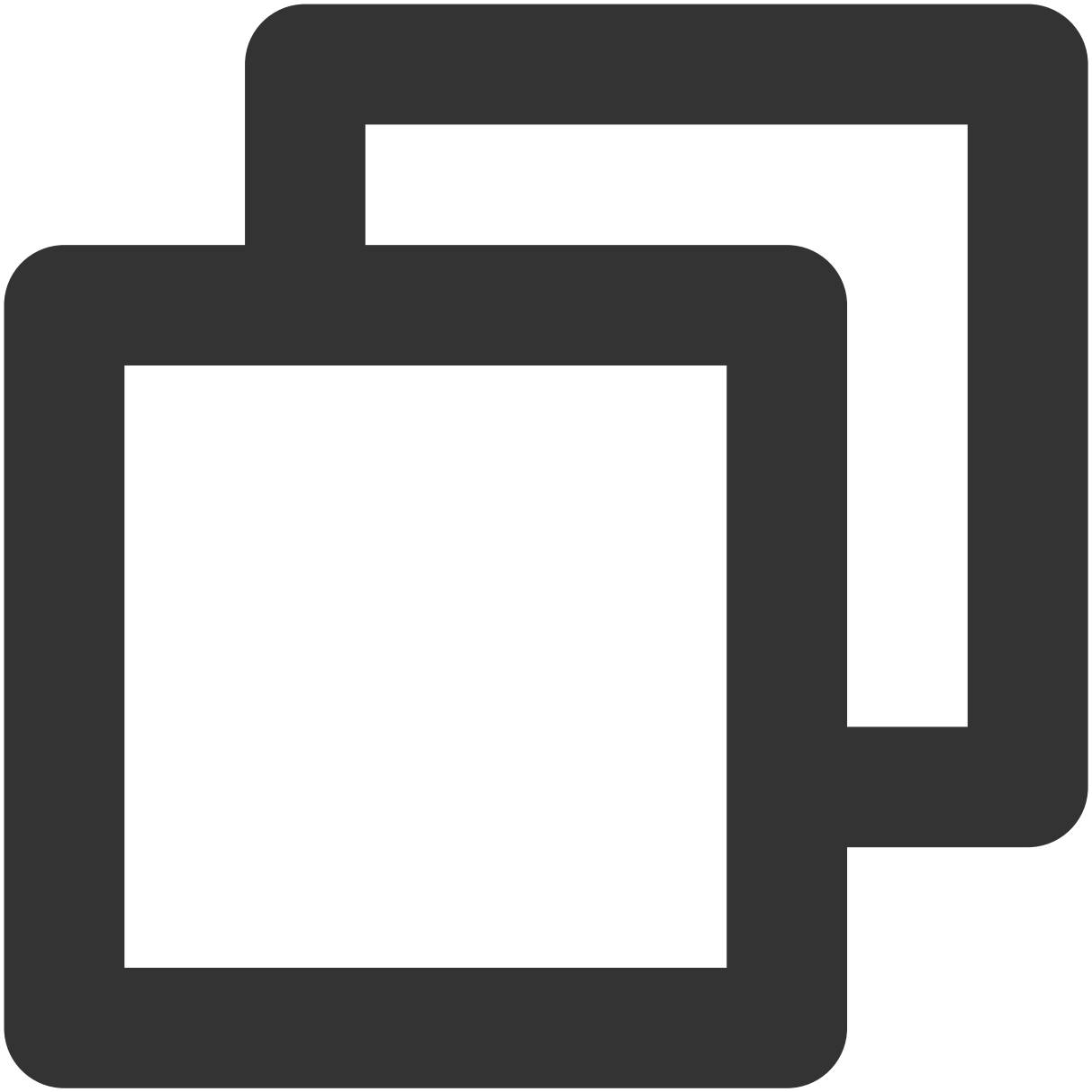
示例：



```
SELECT explode(array(10, 20));
 10
 20
```

EXPLODE_OUTER

函数语法：



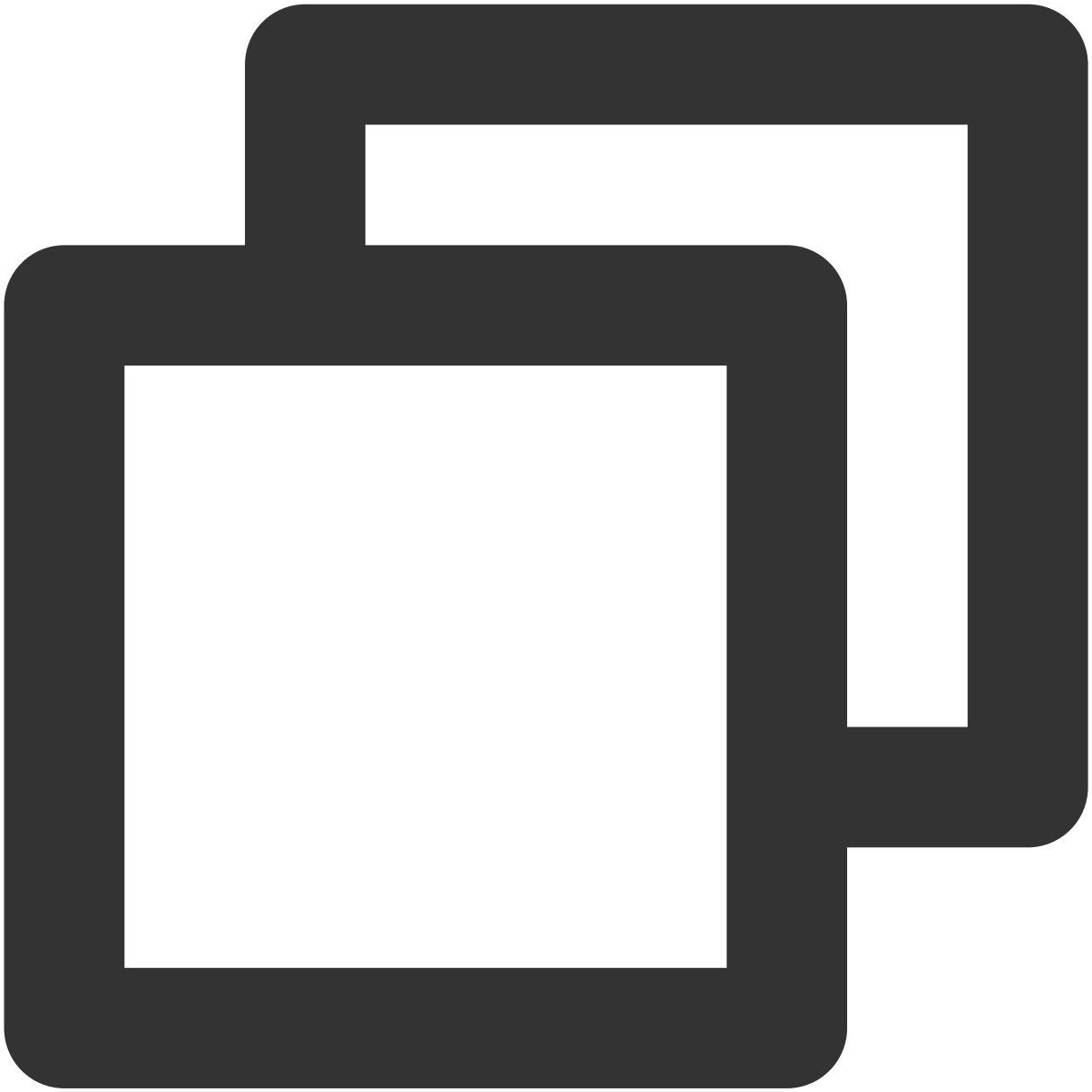
```
EXPLODE_OUTER(<expr> array<T>|map<K, V>)
```

支持引擎：SparkSQL

使用说明：将 array 类型的 expr 的元素分隔为多行，或将 map 类型的 expr 分隔为多个行和列。对数组的元素使用默认列名 col，或对映射的元素使用 key 和 value。

返回类型：row(col T) | row(key K, value V)

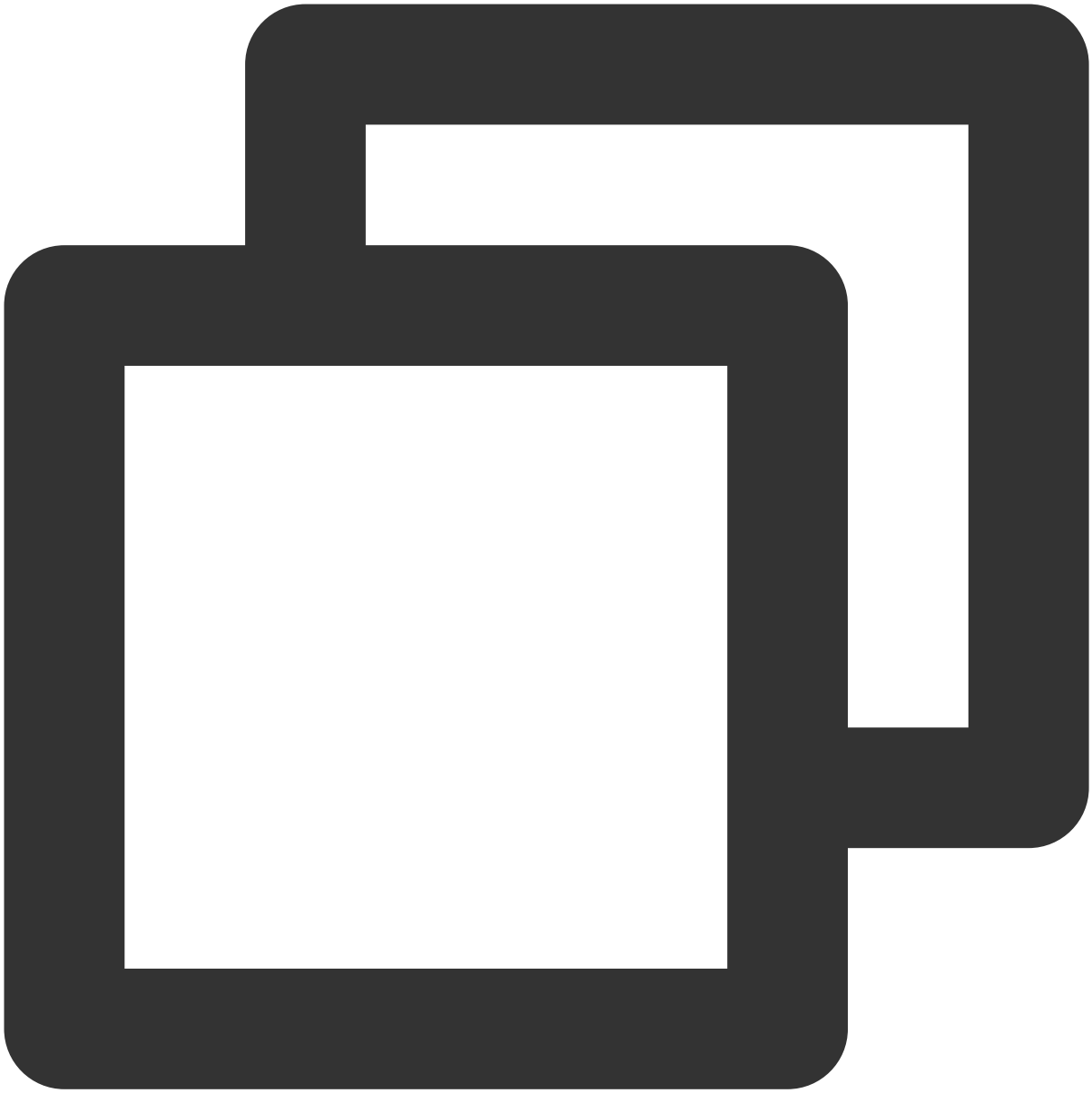
示例：



```
SELECT explode_outer(array(10, 20));
 10
 20
```

GREATEST

函数语法：



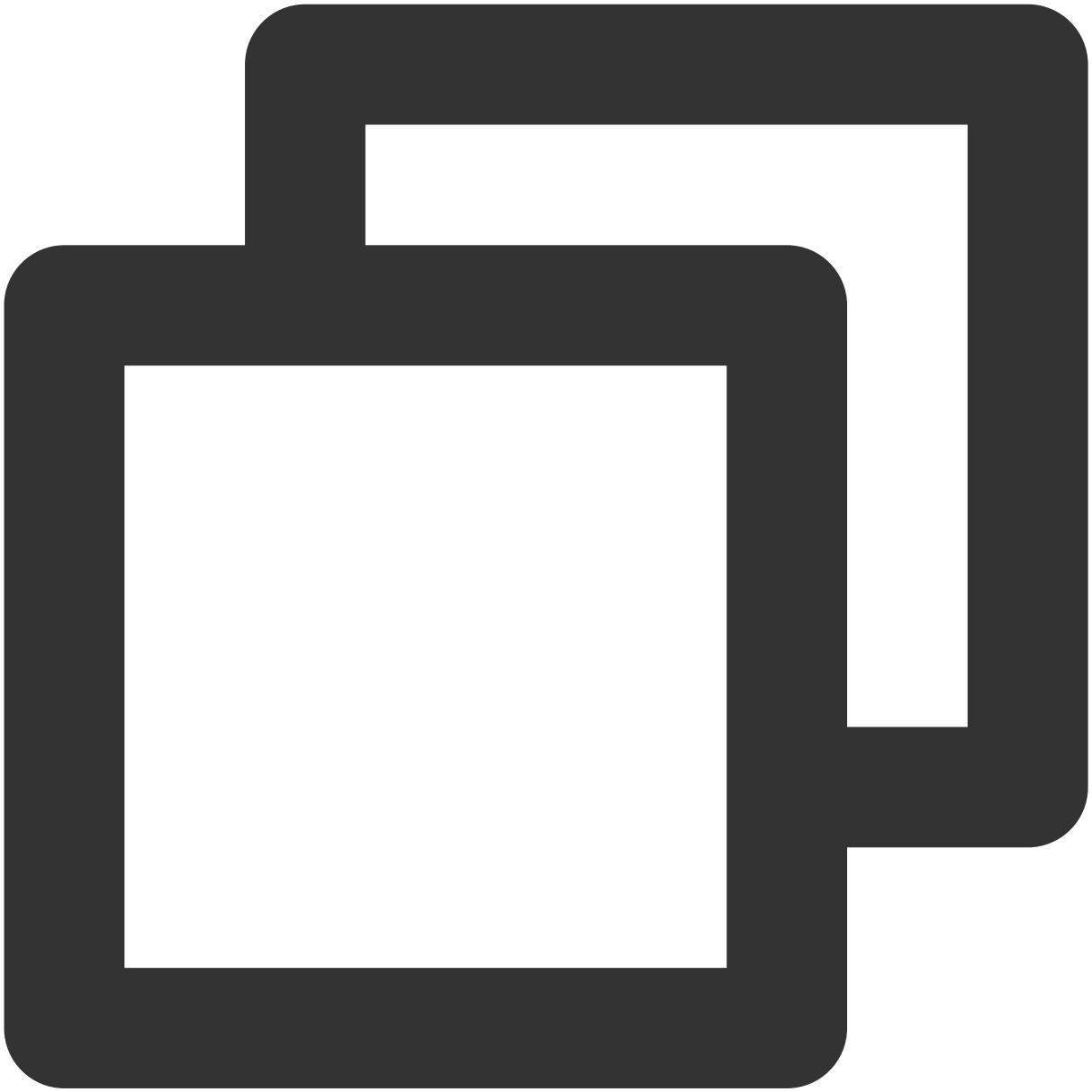
```
GREATEST(<expr1> T, <expr2> T, ...>)
```

支持引擎：SparkSQL、Presto

使用说明：返回所有参数中的最大值，跳过空值。

返回类型：T

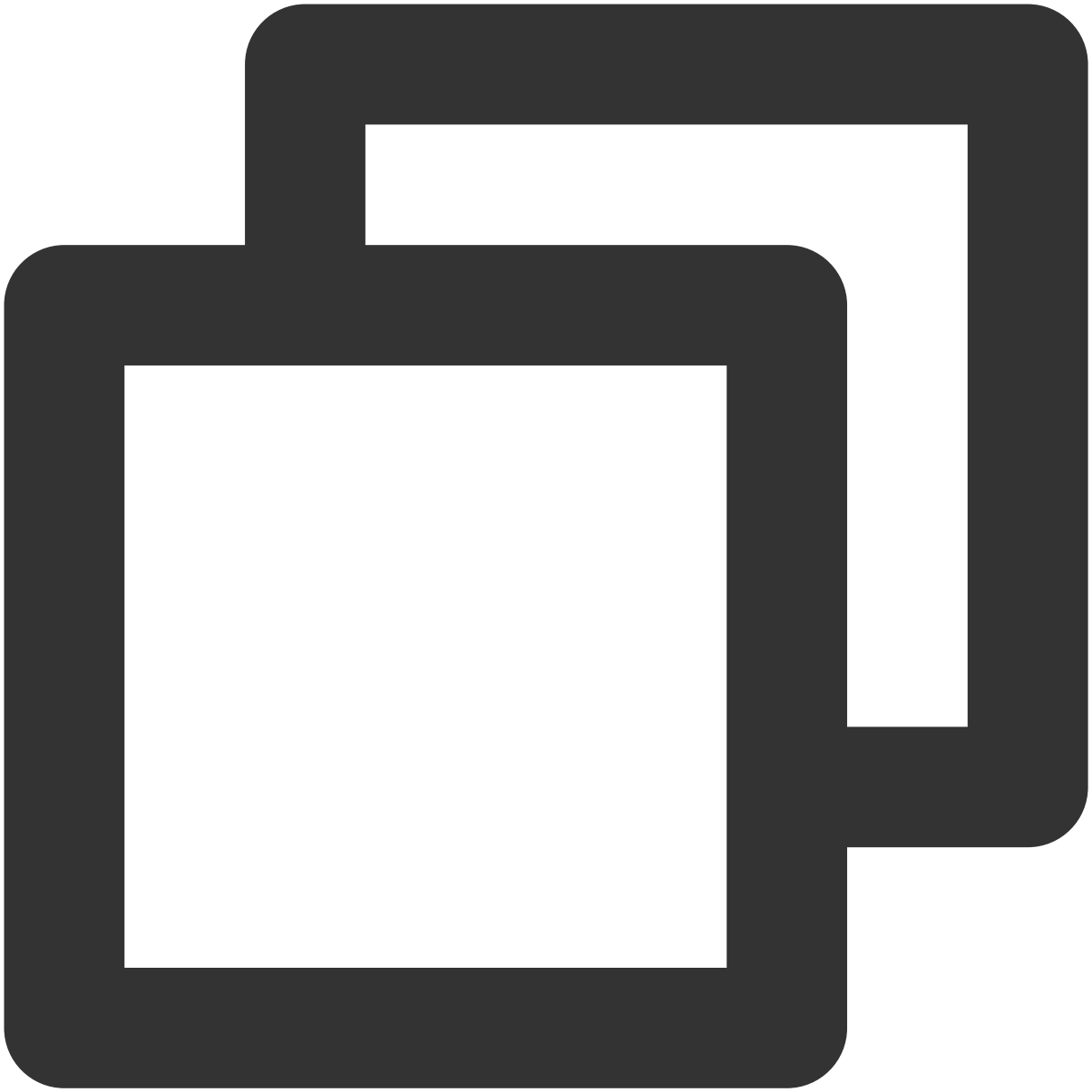
示例：



```
> SELECT greatest(10, 9, 2, 4, 3);
 10
```

IF

函数语法：



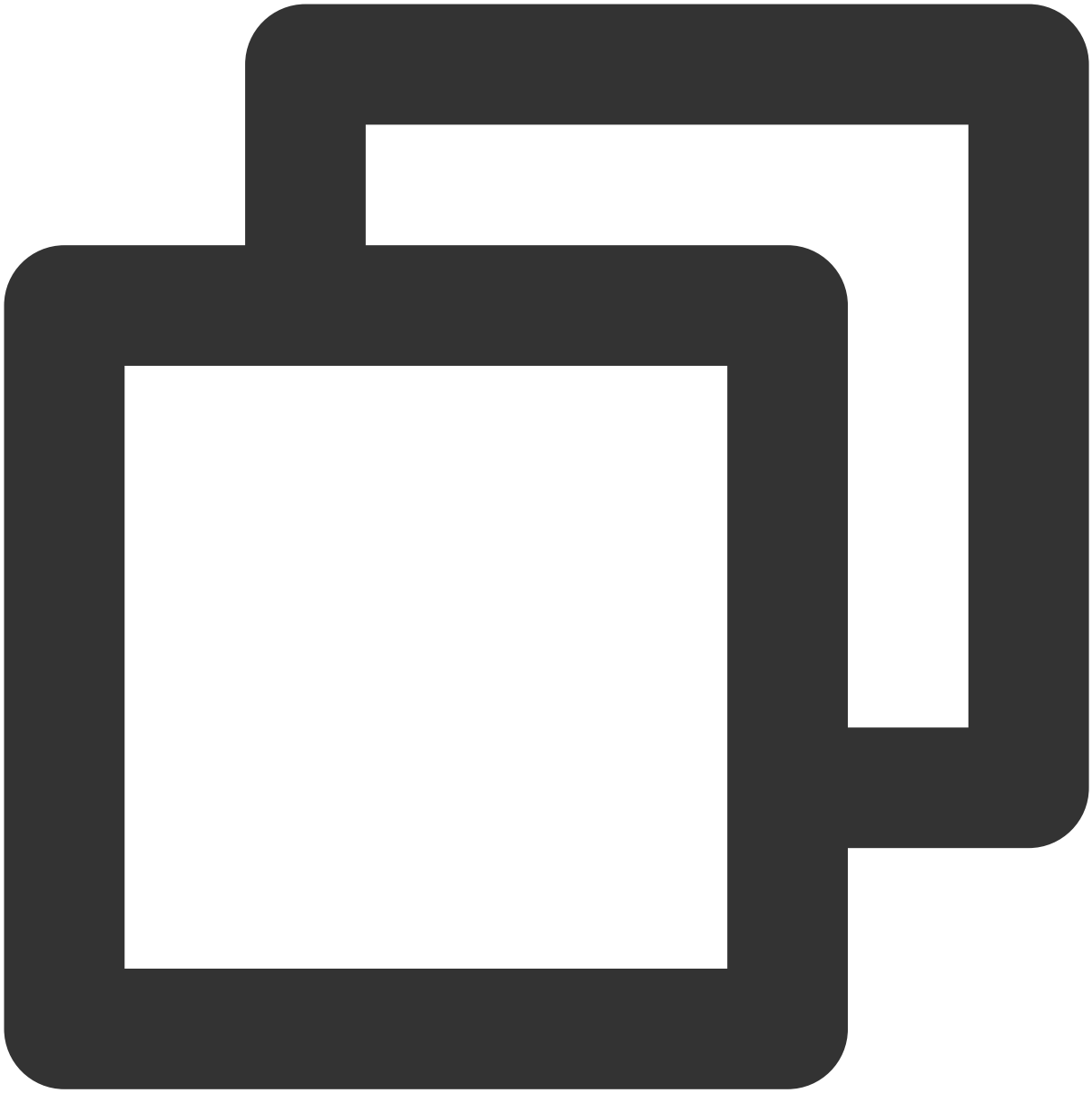
```
IF(<expr1> boolean, <expr2> T, <expr3> U)
```

支持引擎：SparkSQL、Presto

使用说明：如果expr1的计算结果为true，则返回expr2；否则返回expr3。

返回类型：T|U

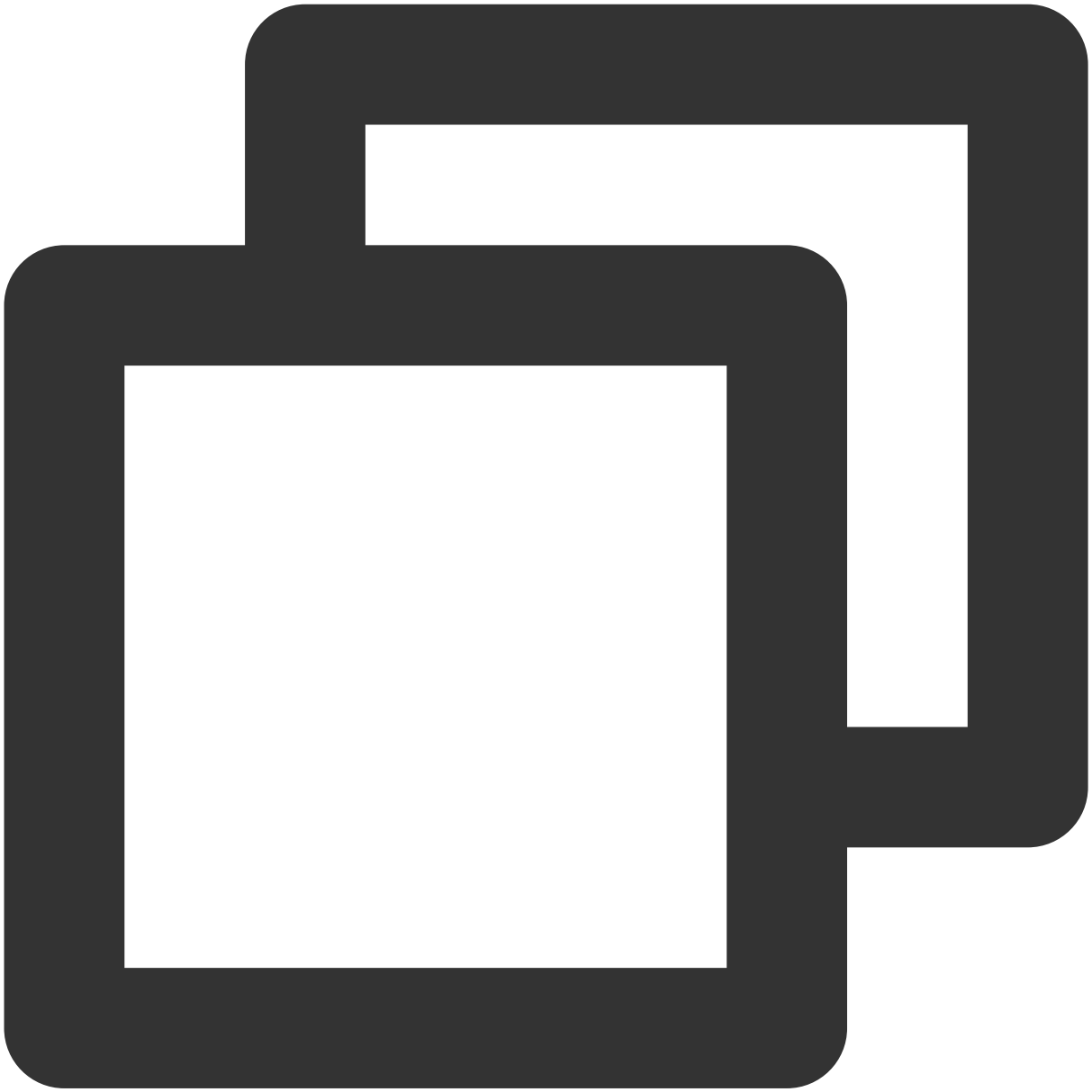
示例：



```
> SELECT if(1 < 2, 'a', 'b');
 a
```

INLINE

函数语法：



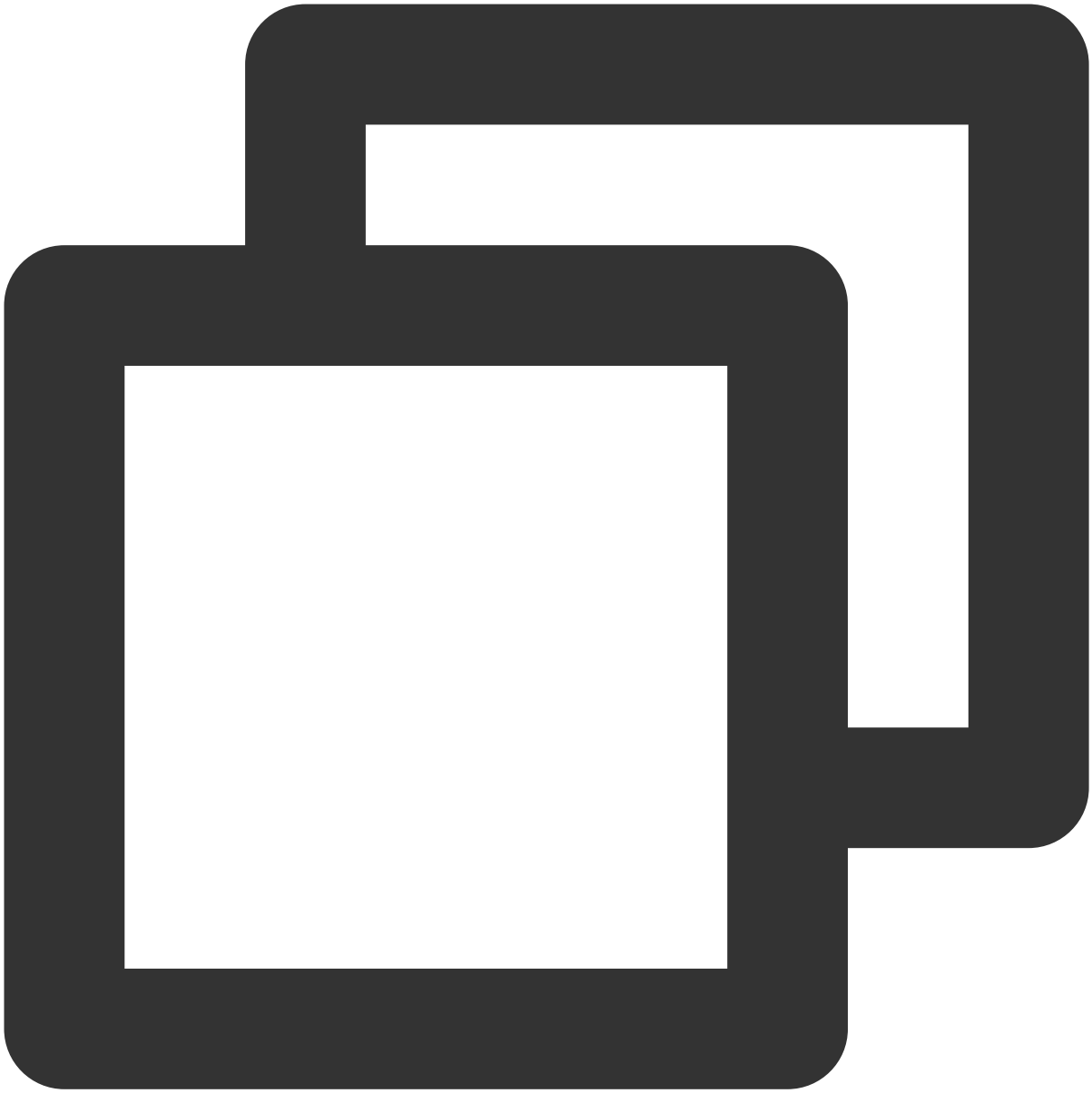
```
INLINE(a array<struct<f1:T1,...,fn:Tn>>)
```

支持引擎：SparkSQL

使用说明：将结构数组分解为表。默认情况下使用列名col1、col2等。

返回类型：row(T1, ..., Tn)

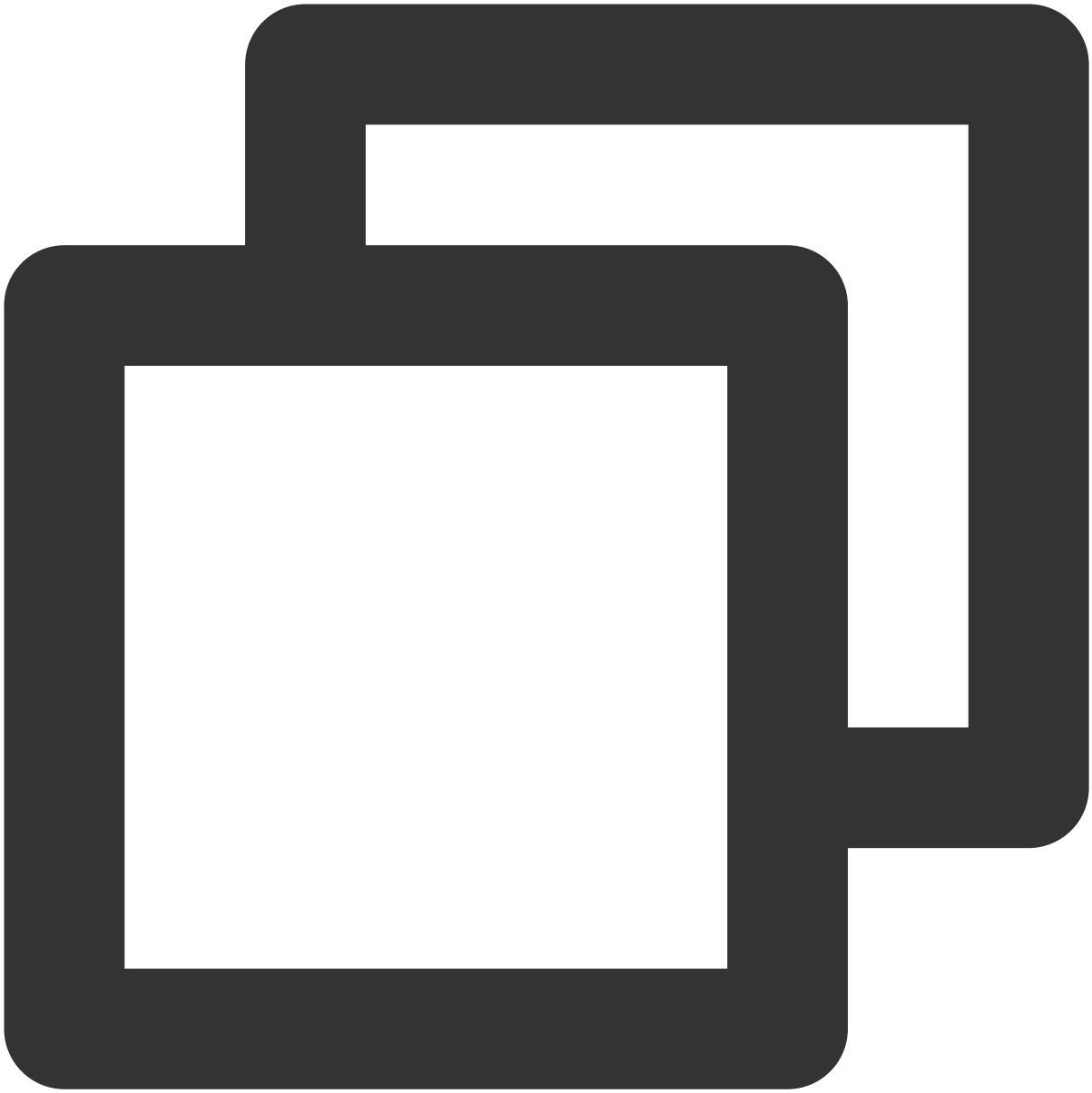
示例：



```
> SELECT inline(array(struct(1, 'a'), struct(2, 'b')));
 1  a
 2  b
```

INLINE_OUTER

函数语法：



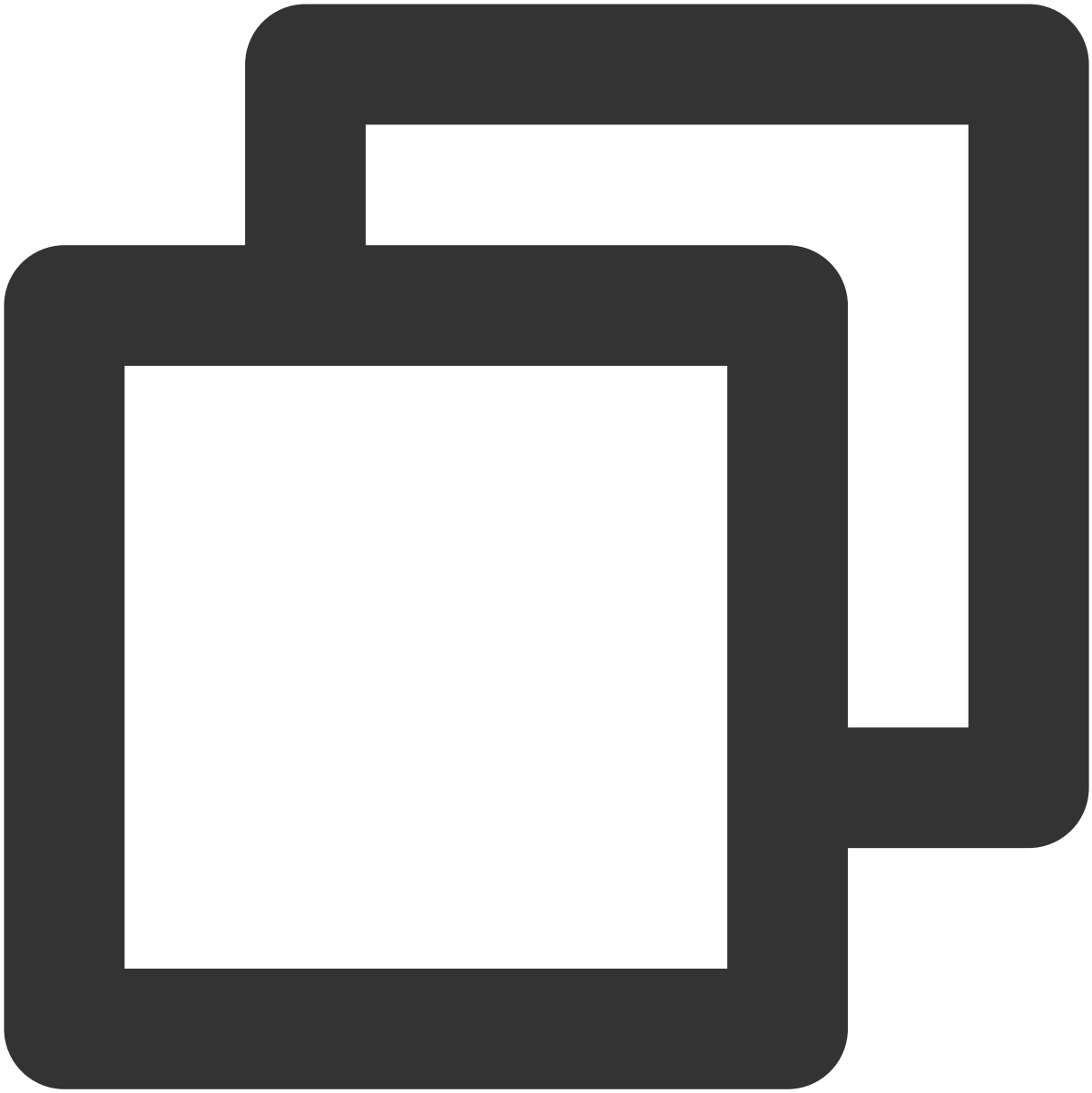
```
INLINE_OUTER(a array<struct<f1:T1,...,fn:Tn>>)
```

支持引擎：SparkSQL

使用说明：将结构数组分解为表。默认情况下使用列名col1、col2等。

返回类型：row(T1, ..., Tn)

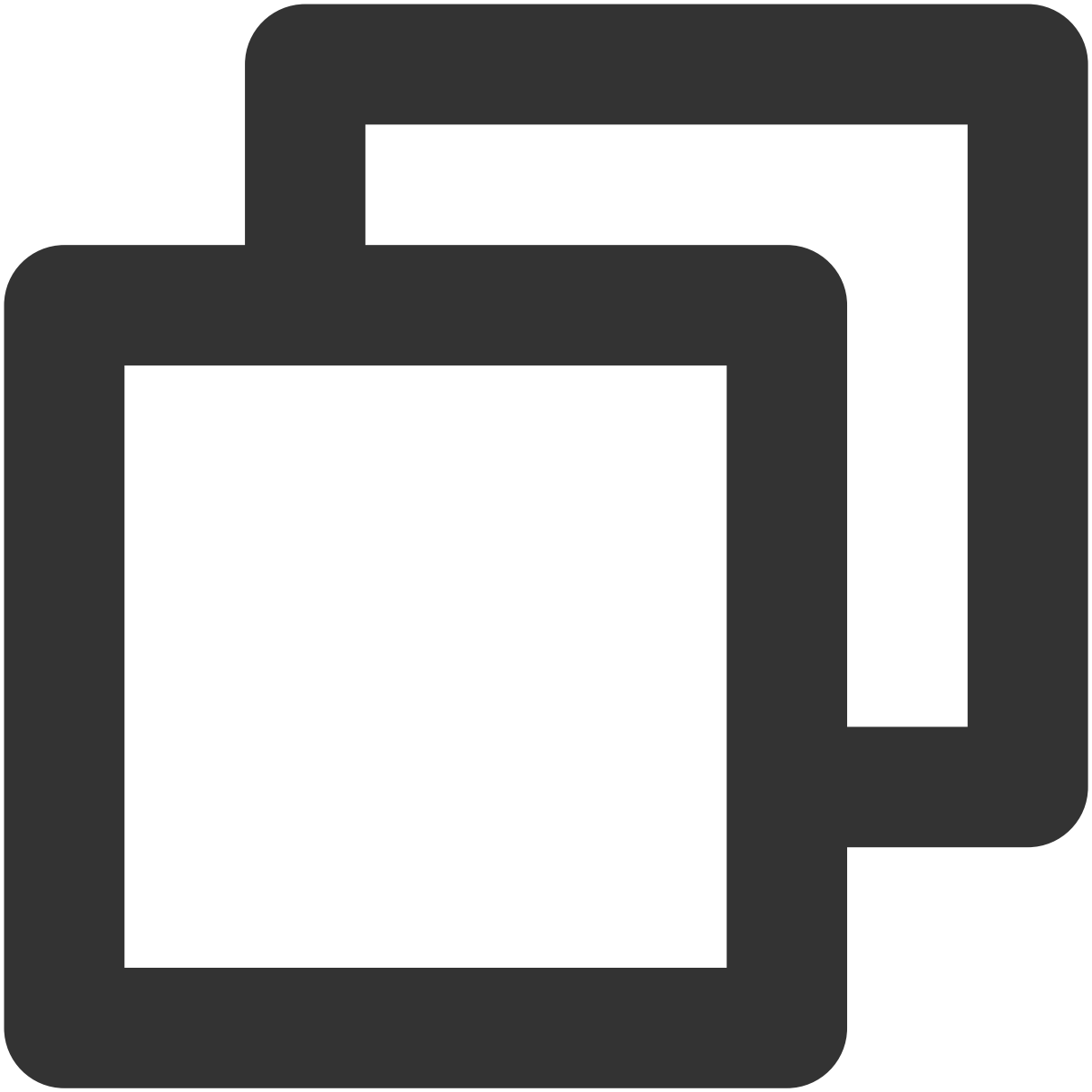
示例：



```
> SELECT inline(array(struct(1, 'a'), struct(2, 'b')));
 1  a
 2  b
```

IN

函数语法：



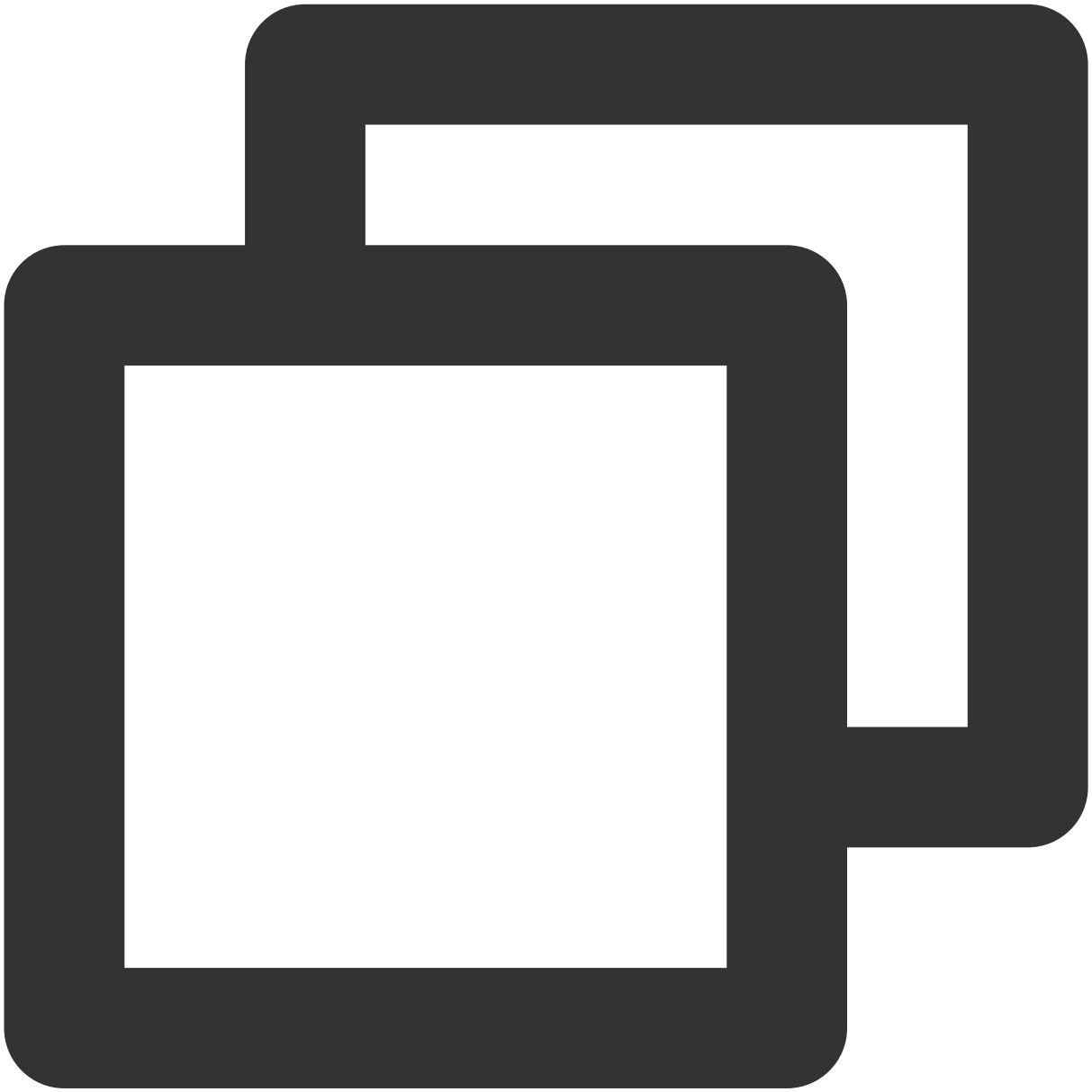
```
<expr1> IN(<expr2> T, <expr3> T, ...)
```

支持引擎：SparkSQL、Presto

使用说明：如果expr1等于任何exprn，则返回true。

返回类型：boolean

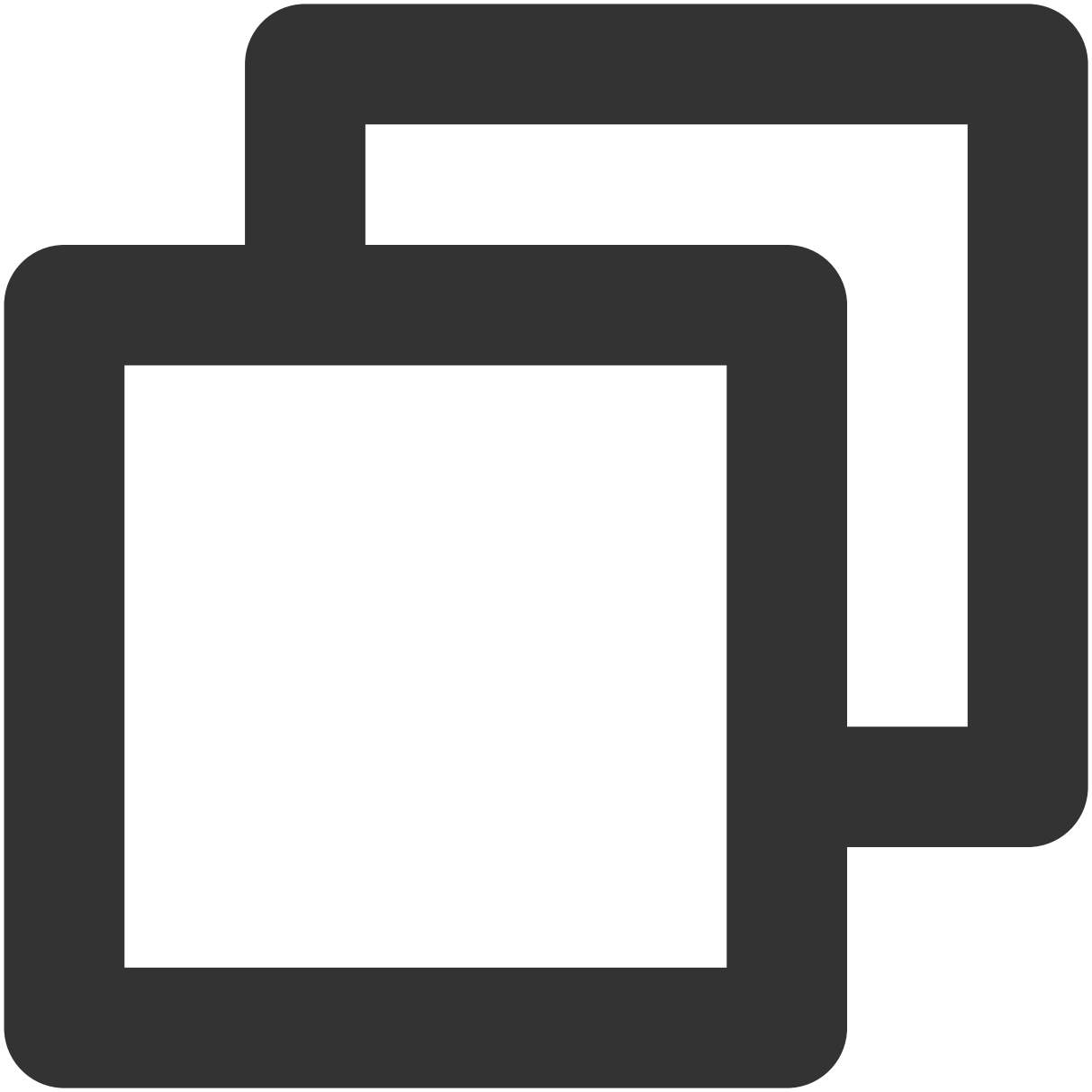
示例：



```
> SELECT 1 in(1, 2, 3);
 true
> SELECT 1 in(2, 3, 4);
 false
```

ISNAN

函数语法：



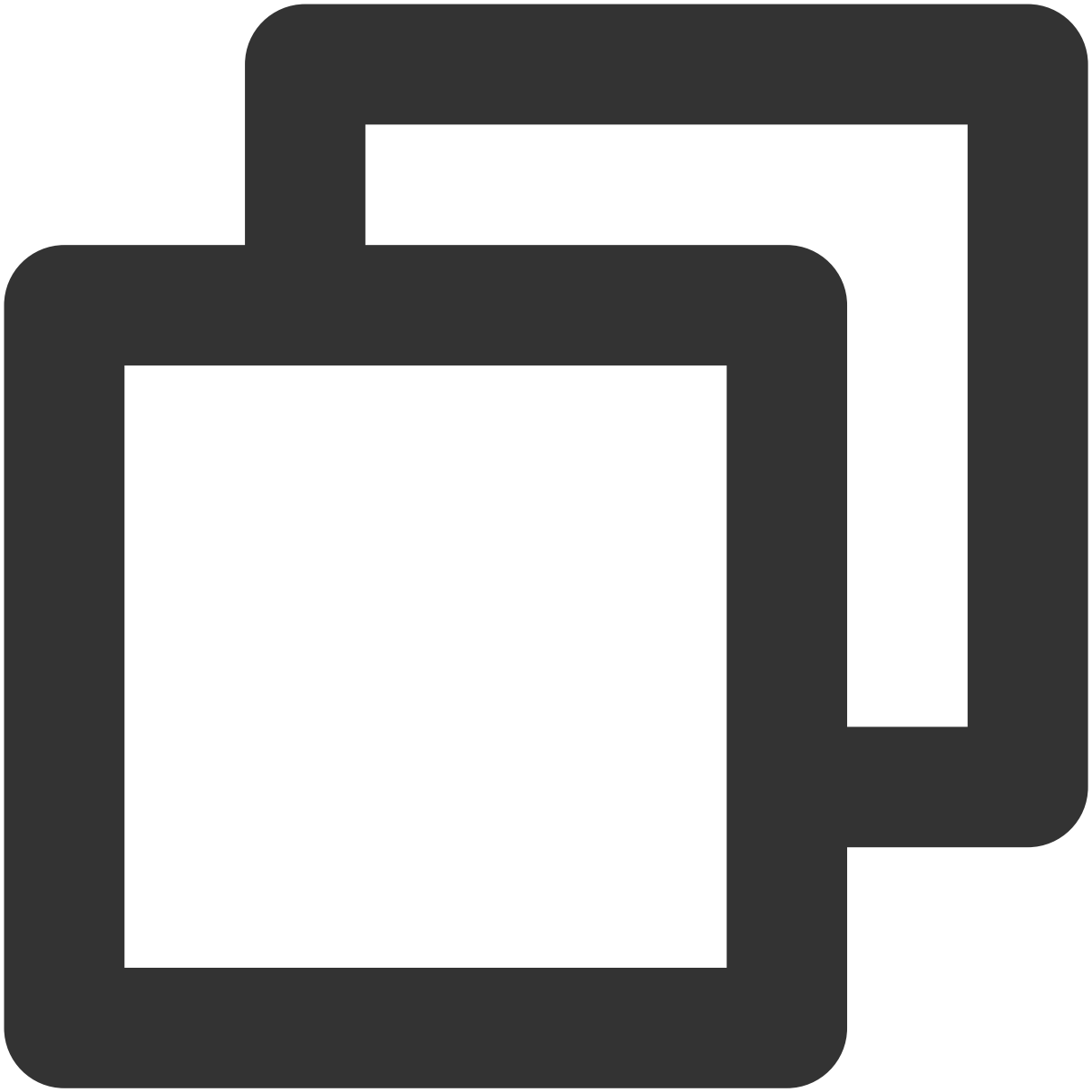
```
ISNAN(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：如果expr为NaN，则返回true，否则返回false。

返回类型：boolean

示例：



```
> SELECT isnan(cast('NaN' as double));
 true
```

IFNULL

函数语法：

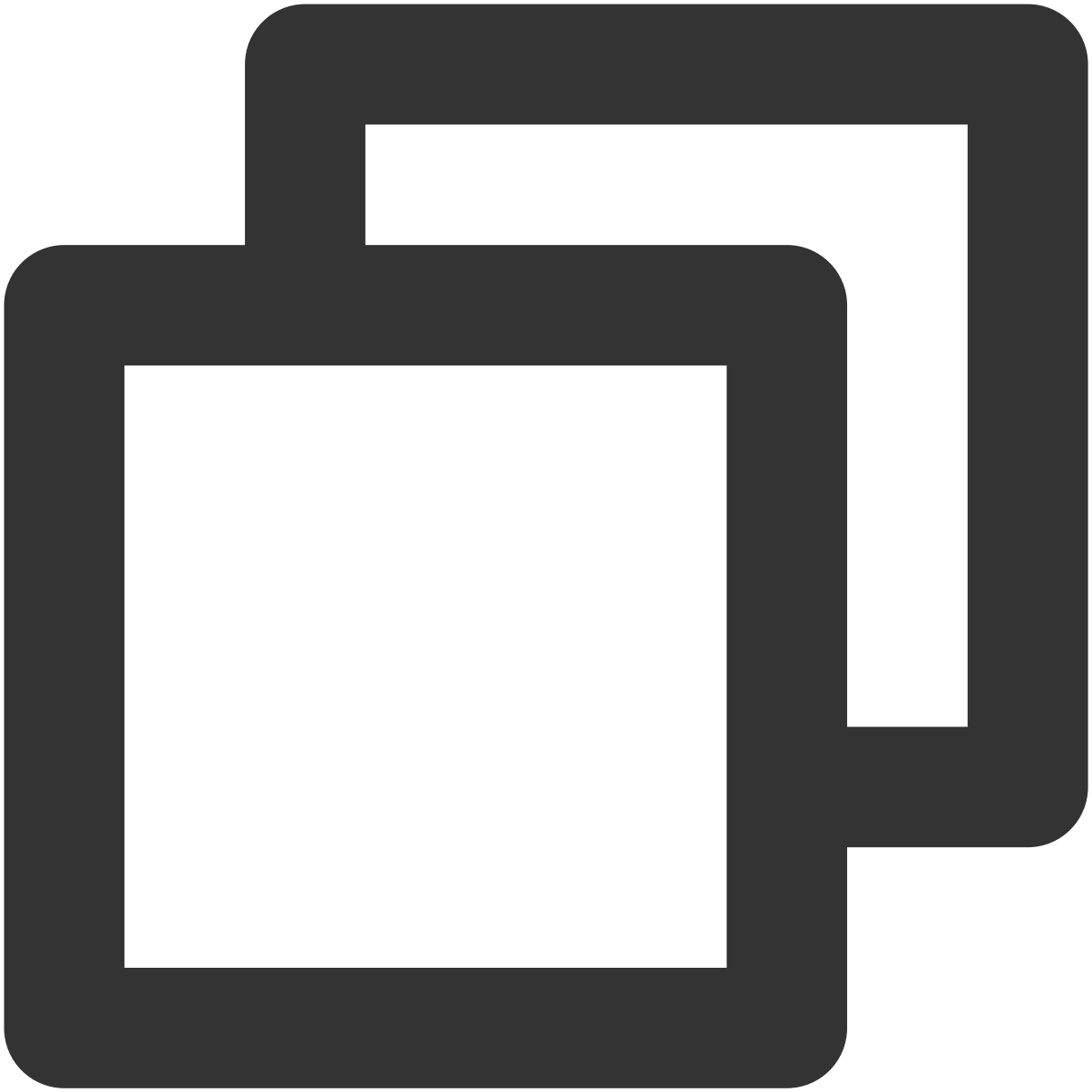
```
IFNULL(<expr1> T, <expr2> U)
```

支持引擎：SparkSQL

使用说明：如果expr1为null，则返回expr2，否则返回expr1。

返回类型：T|U

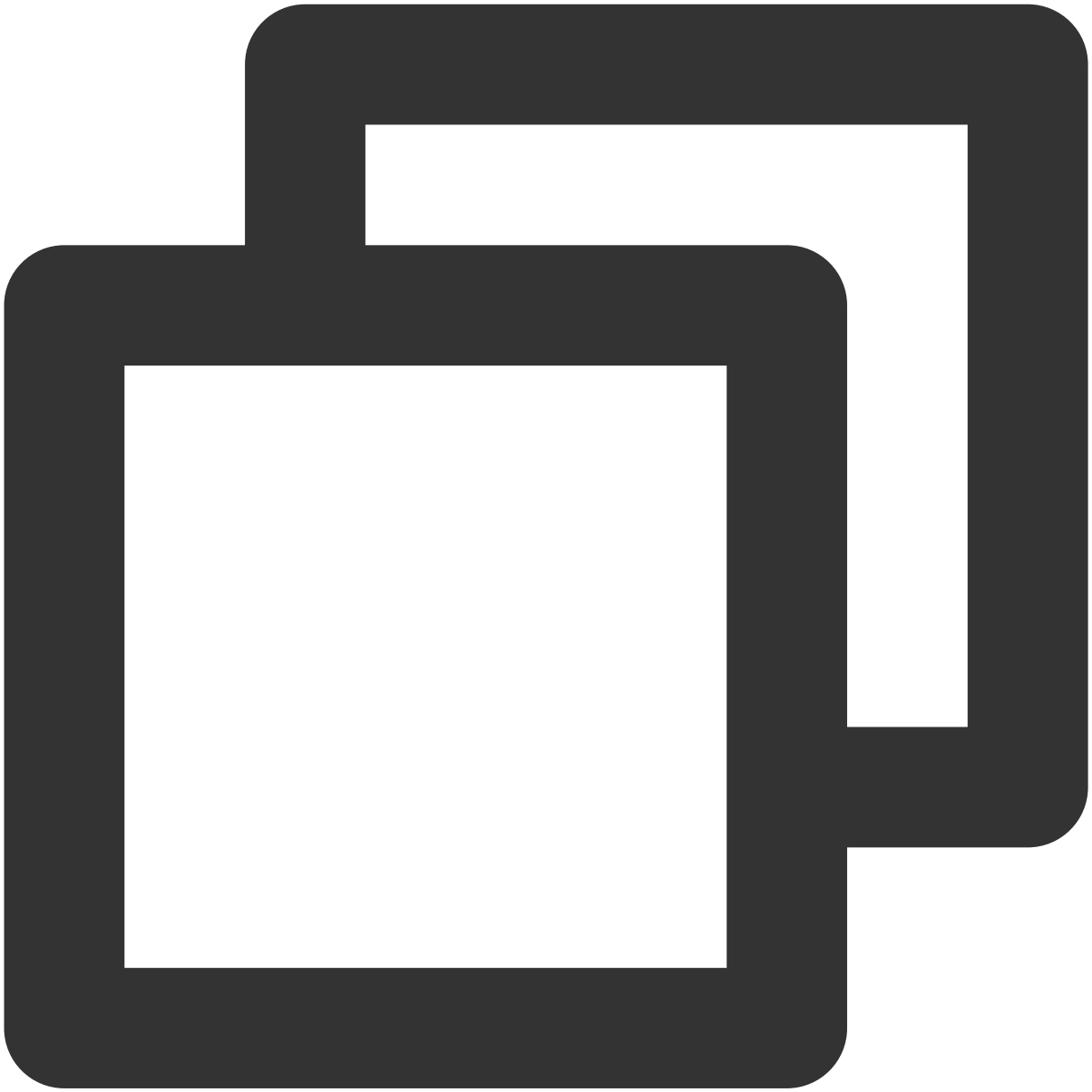
示例：



```
> SELECT ifnull(NULL, array('2'));
 ["2"]
```

ISNULL

函数语法：



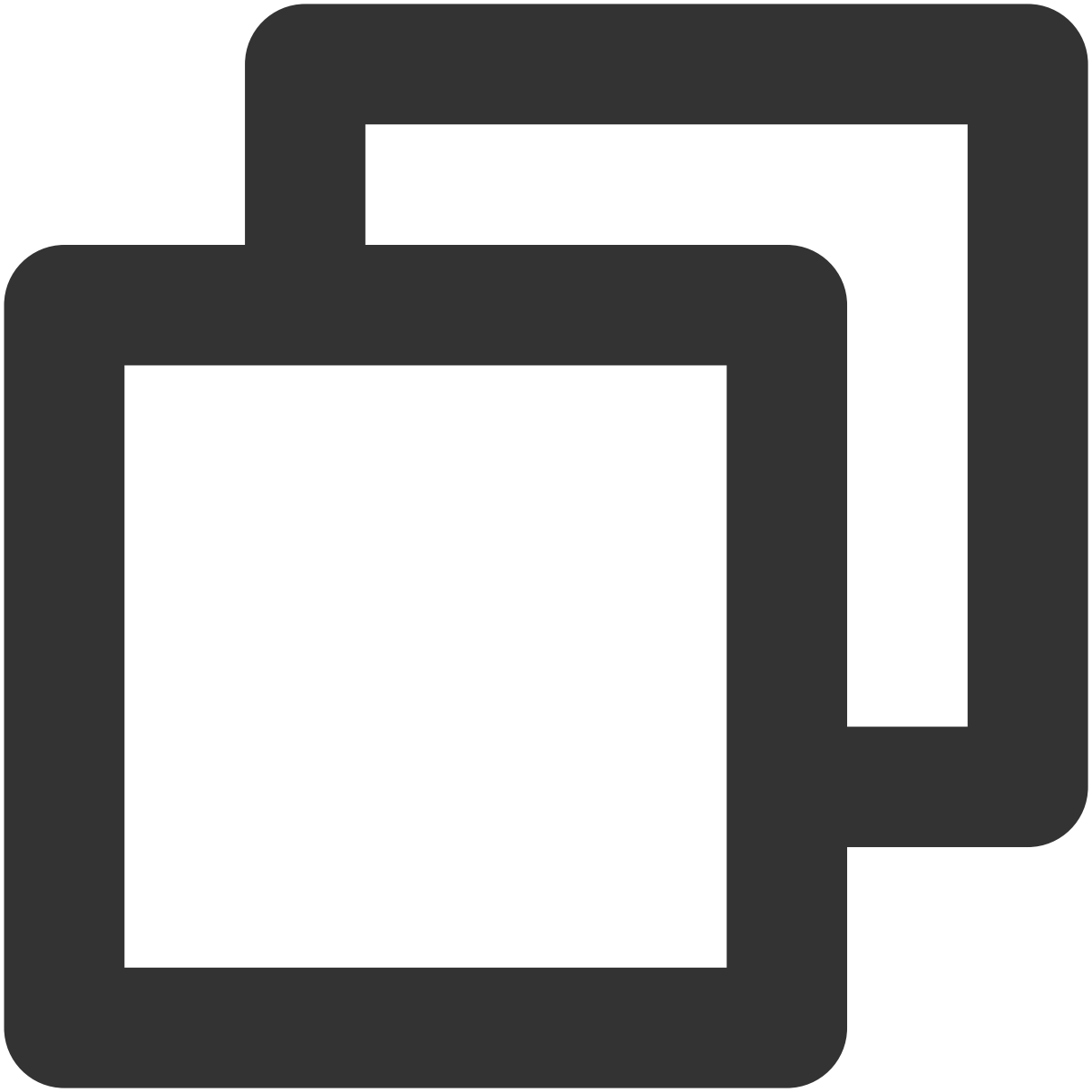
```
ISNULL(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：如果expr为null，则返回true，否则返回false。

返回类型：boolean

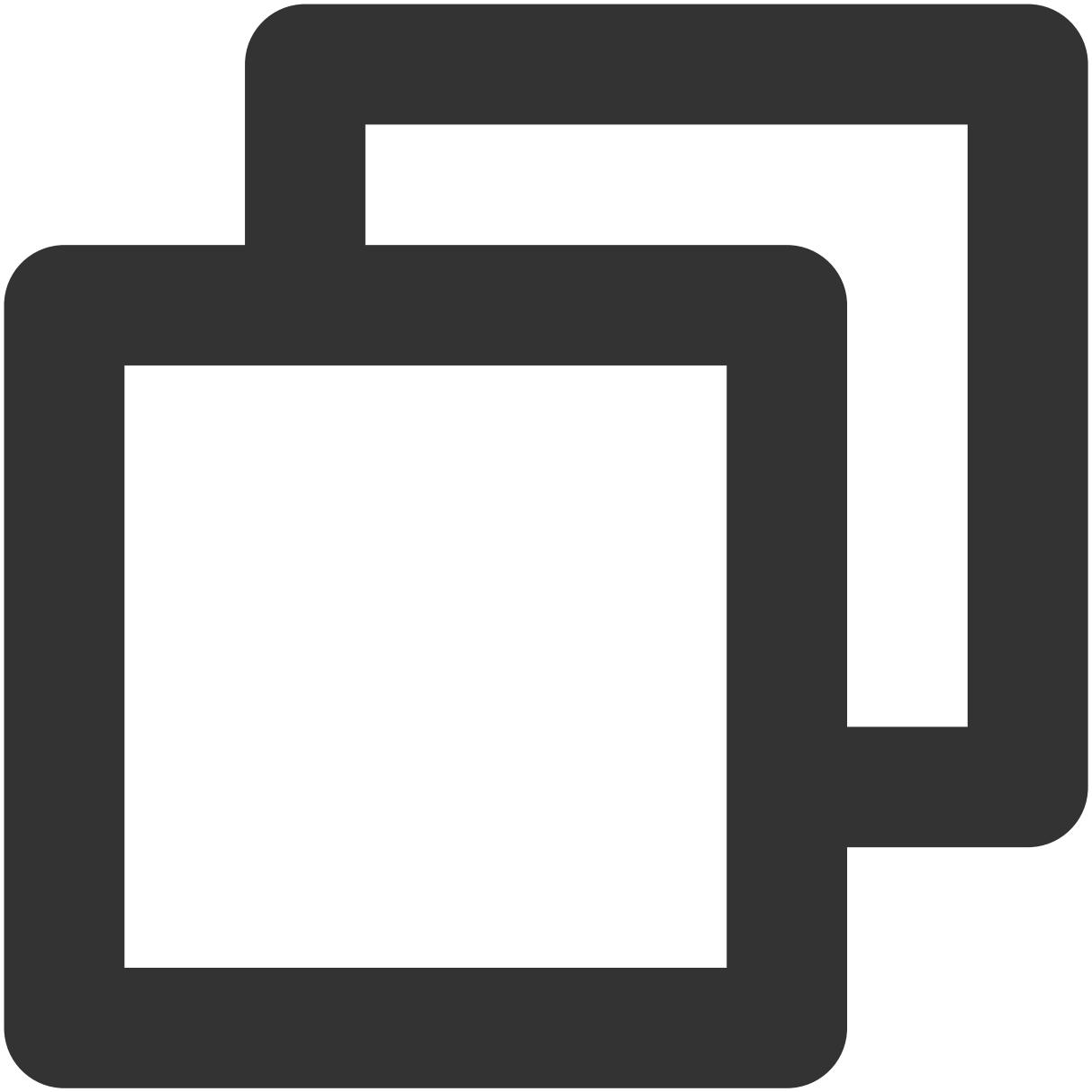
示例：



```
> SELECT isnull(1);
 false
```

ISNOTNULL

函数语法：



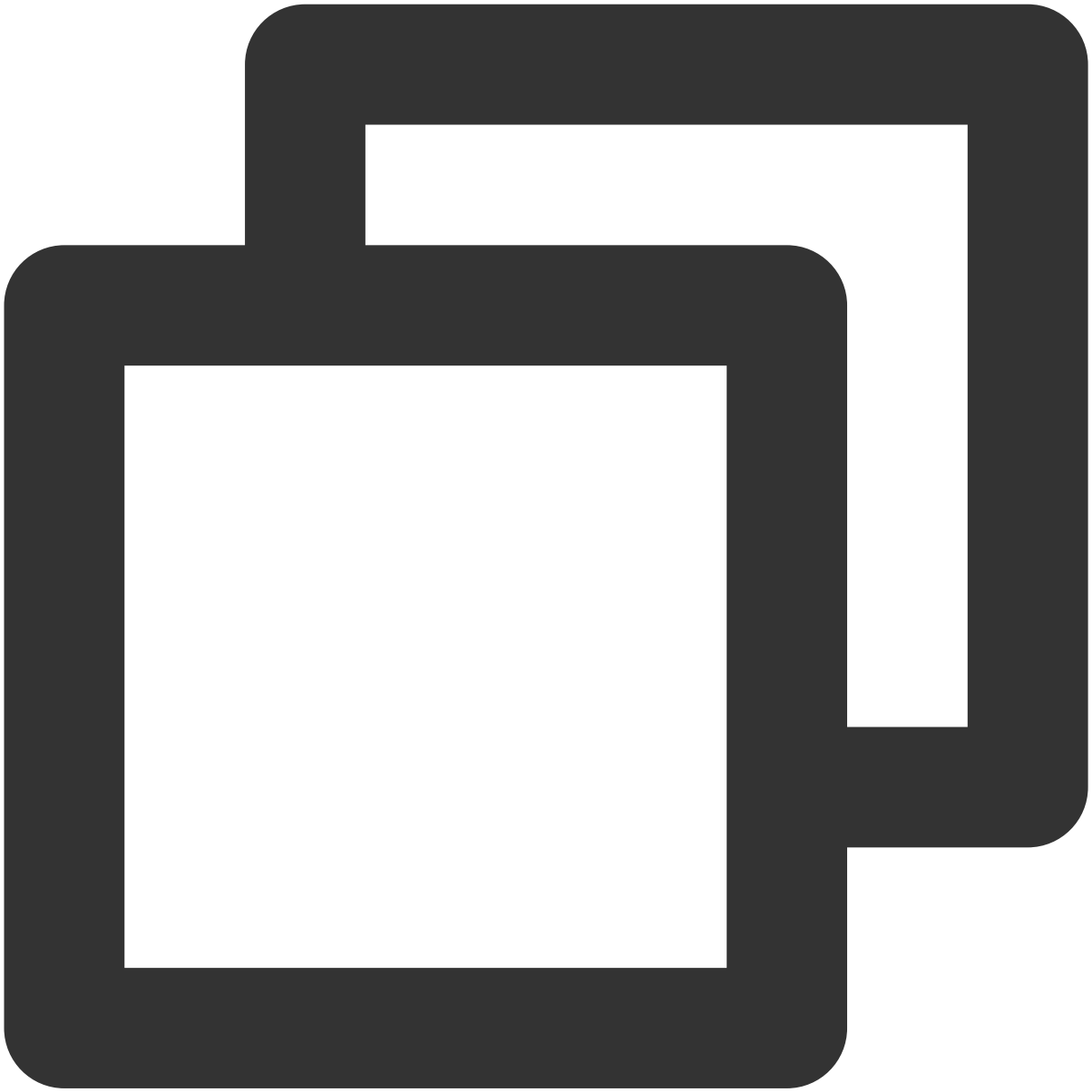
```
ISNOTNULL(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：如果expr不为null，则返回true，否则返回false。

返回类型：boolean

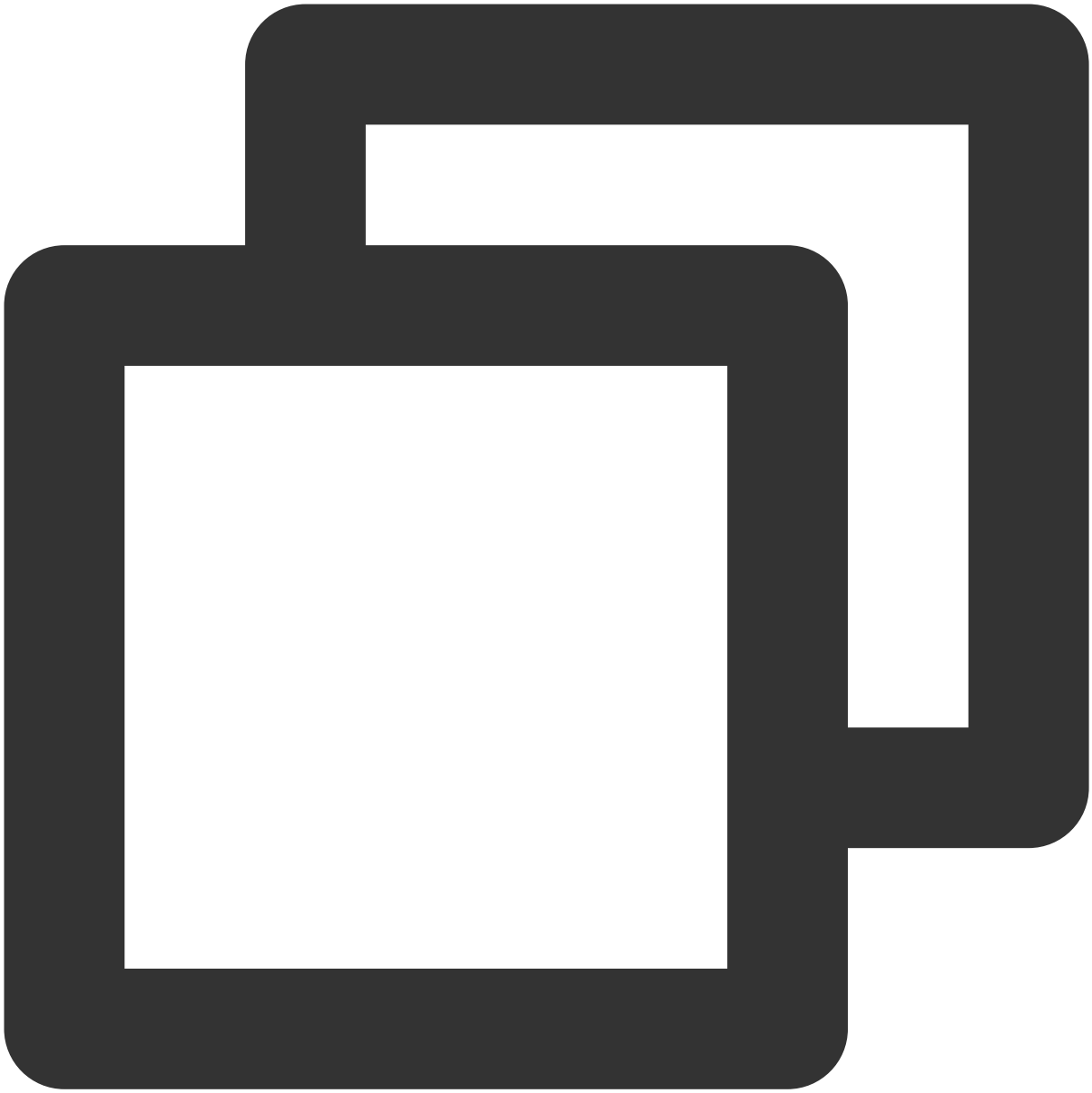
示例：



```
> SELECT isnotnull(1);
 true
```

LEAST

函数语法：



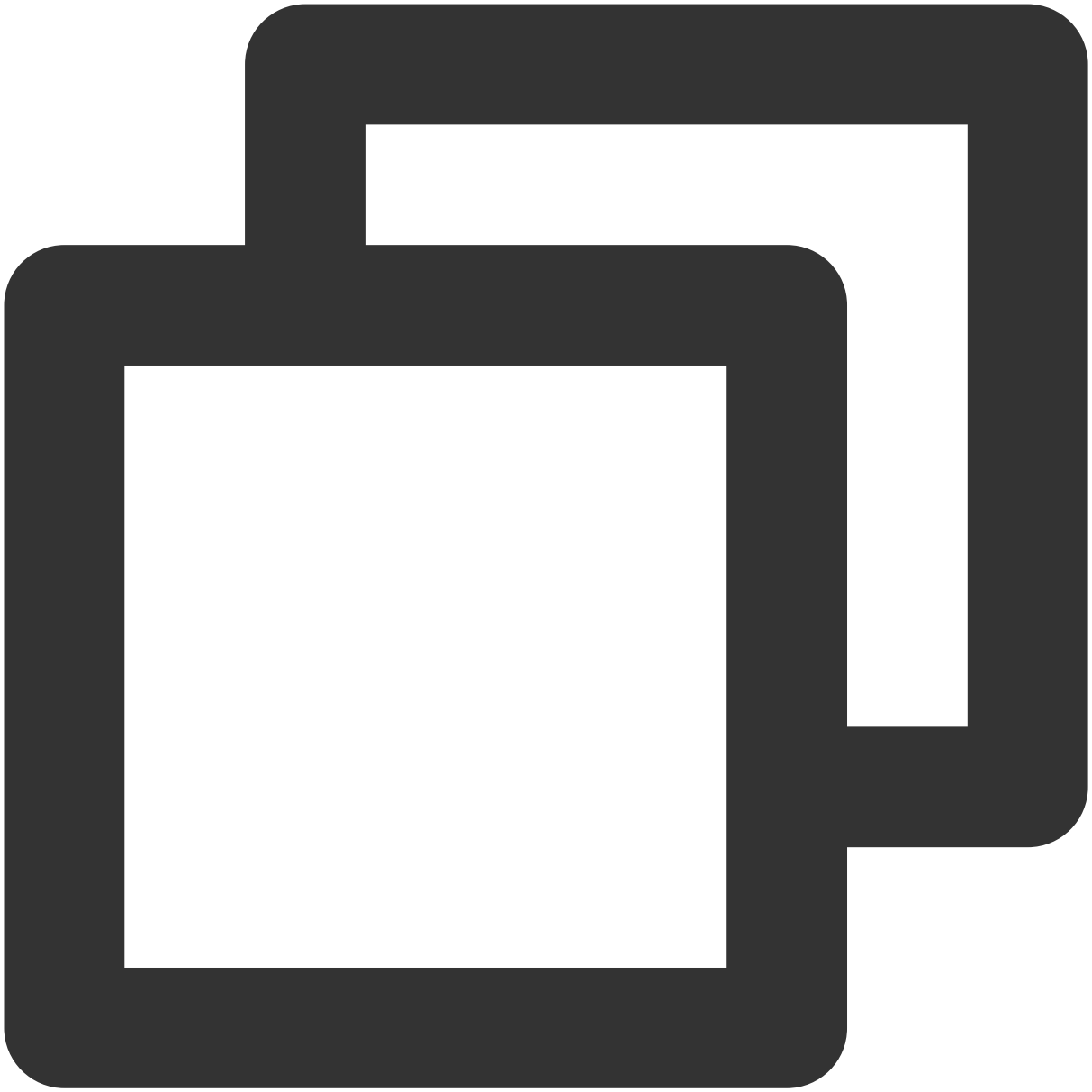
```
LEAST(<expr1> T, <expr2> T, ...)
```

支持引擎：SparkSQL、Presto

使用说明：返回所有参数中的最小值，跳过null。

返回类型：T

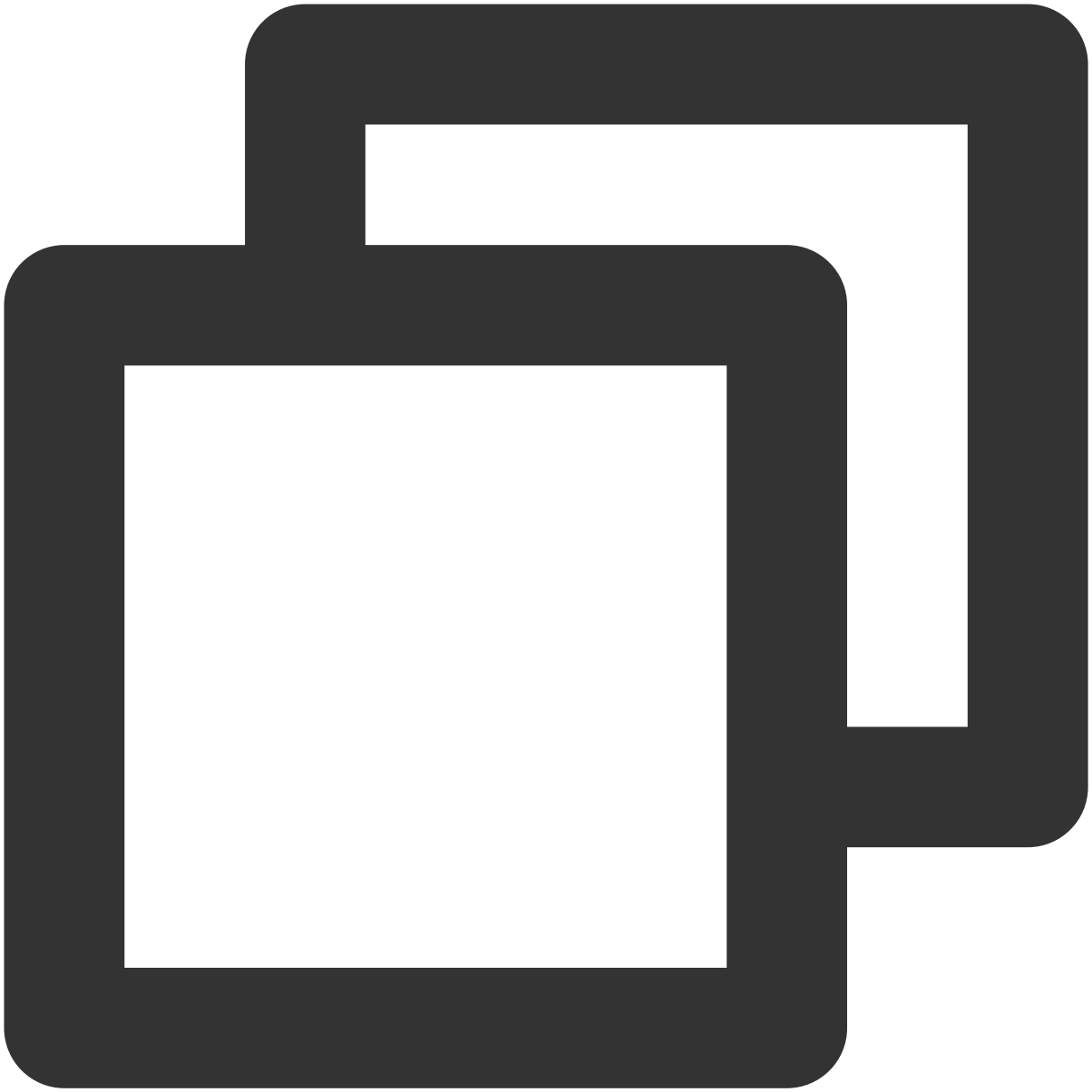
示例：



```
> SELECT least(10, 9, 2, 4, 3);
 2
```

NANVL

函数语法：



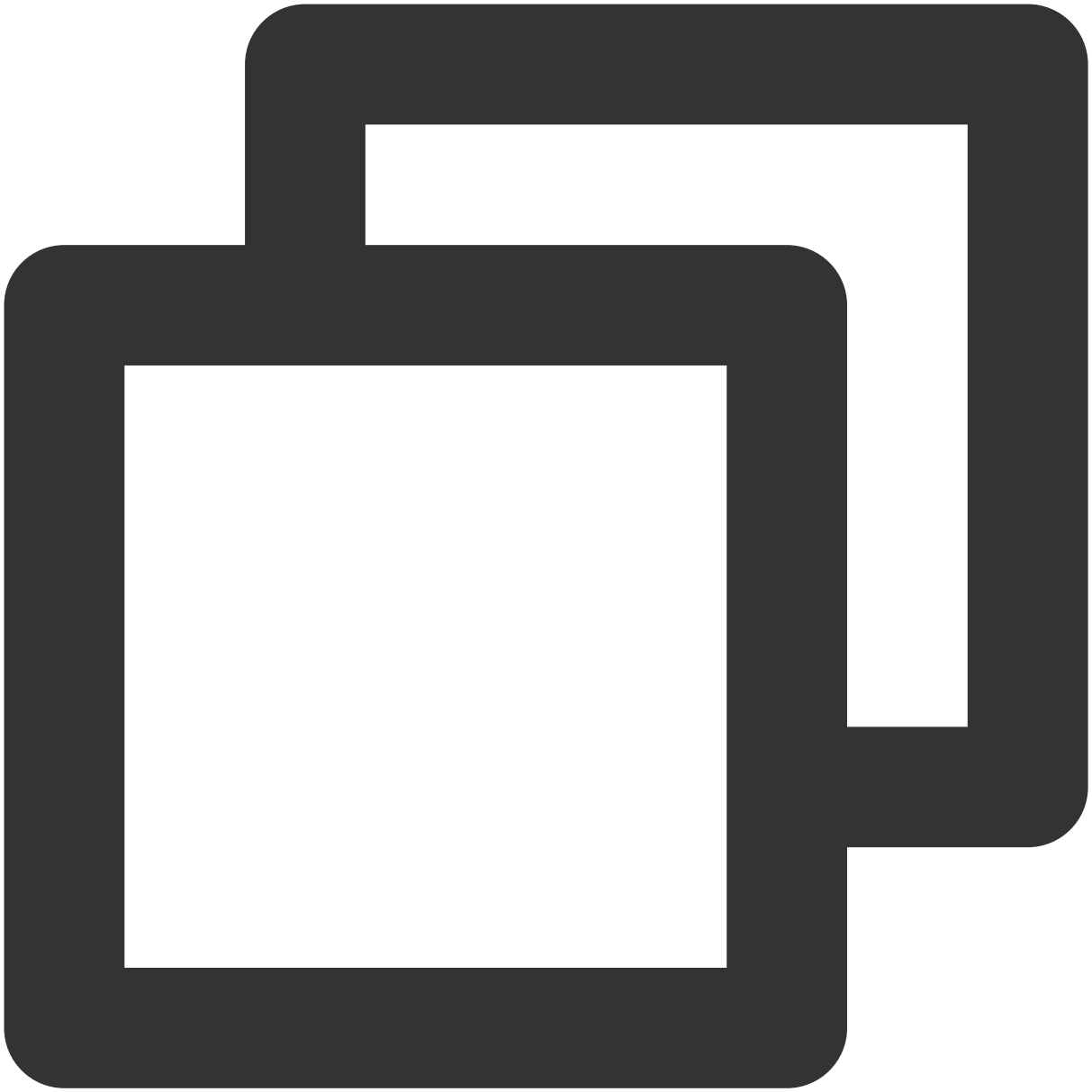
```
NANVL(<expr1> T, <expr2> U)
```

支持引擎：SparkSQL、Presto

使用说明：nanvl（expr1，expr2），如果expr1不是NaN，则返回expr1；否则返回expr2。

返回类型：T|U

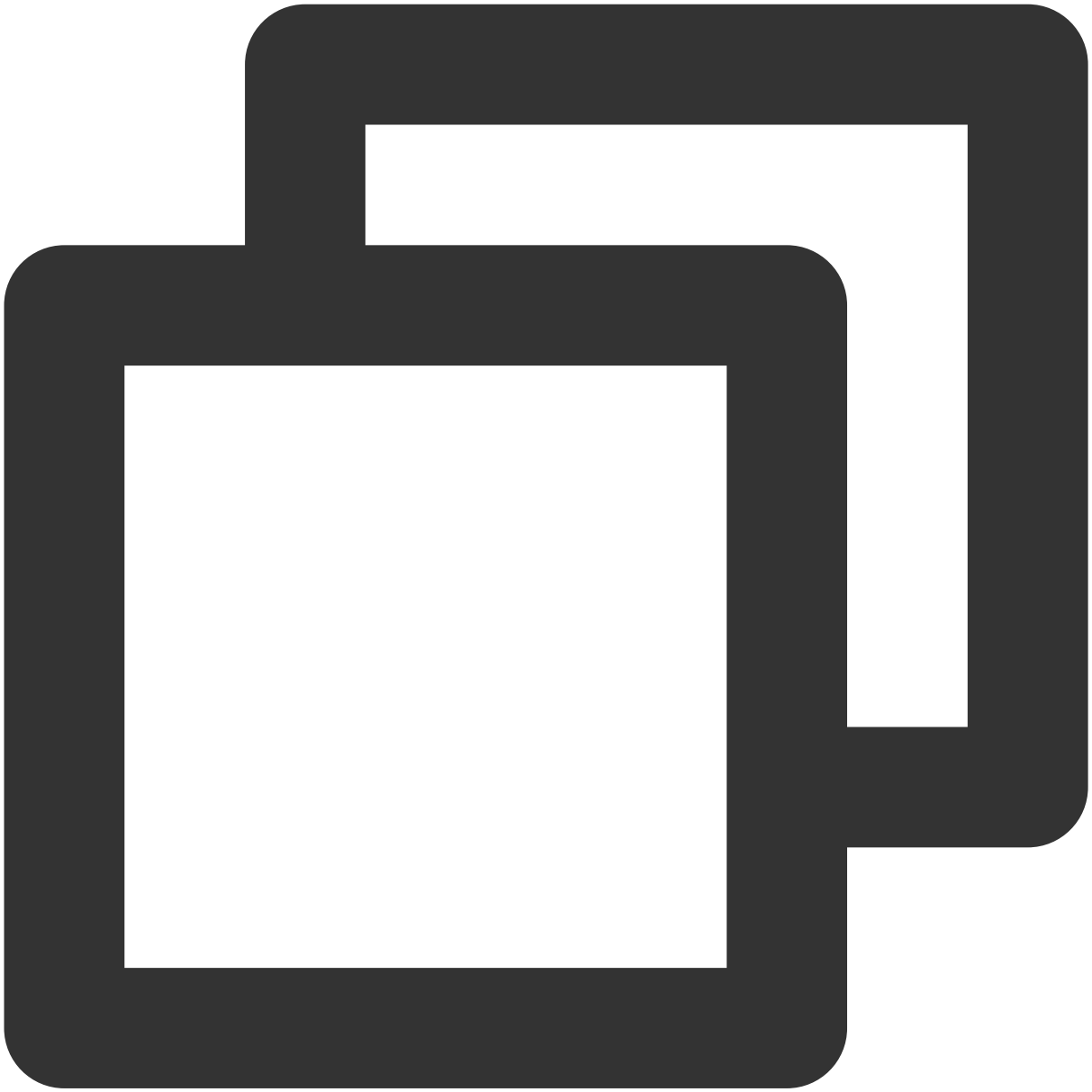
示例：



```
> SELECT nanvl(cast('NaN' as double), 123);
 123.0
```

NULLIF

函数语法：



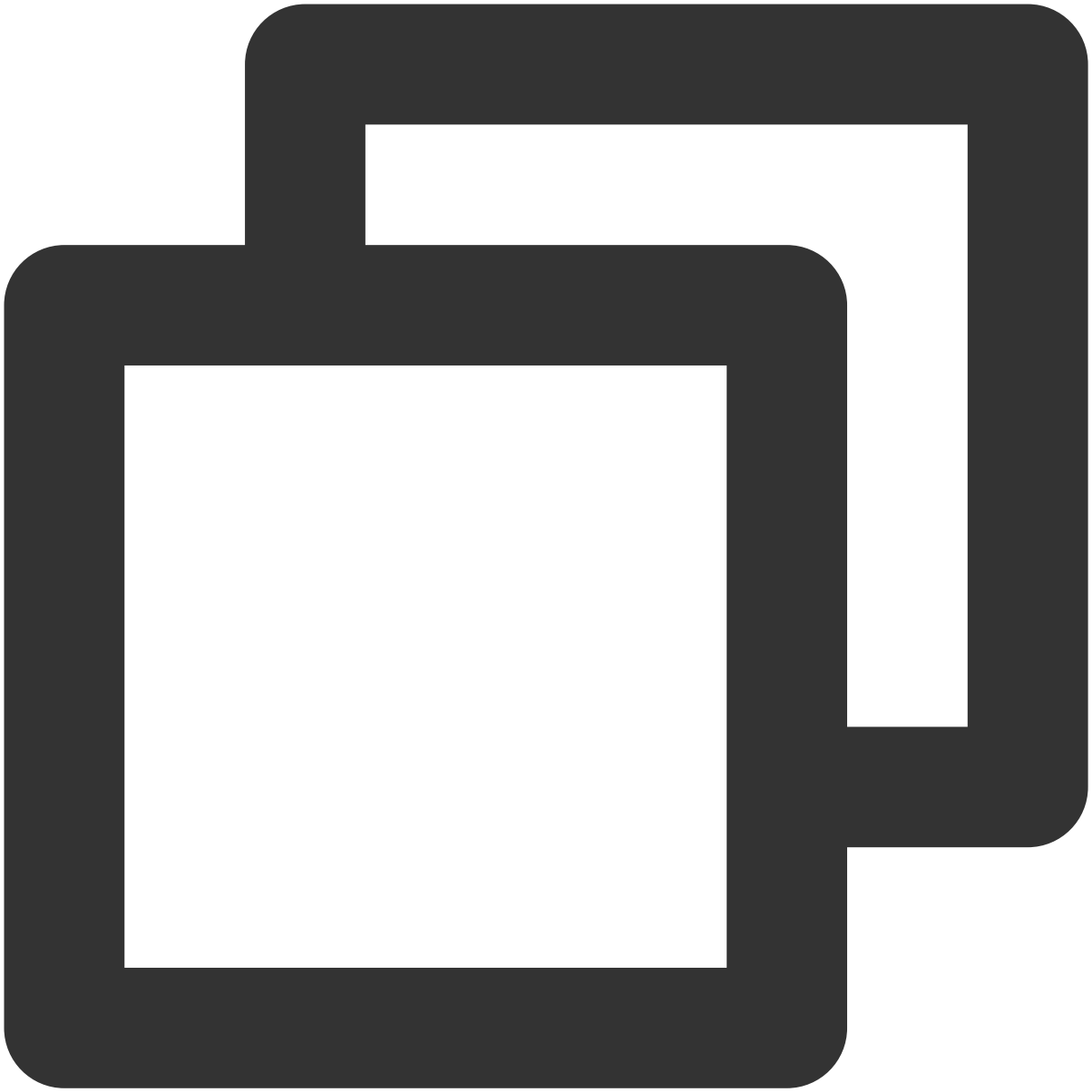
```
NULLIF(<expr1> T, <expr2> U)
```

支持引擎：SparkSQL、Presto

使用说明：如果expr1等于expr2，则返回null，否则返回expr1。

返回类型：T

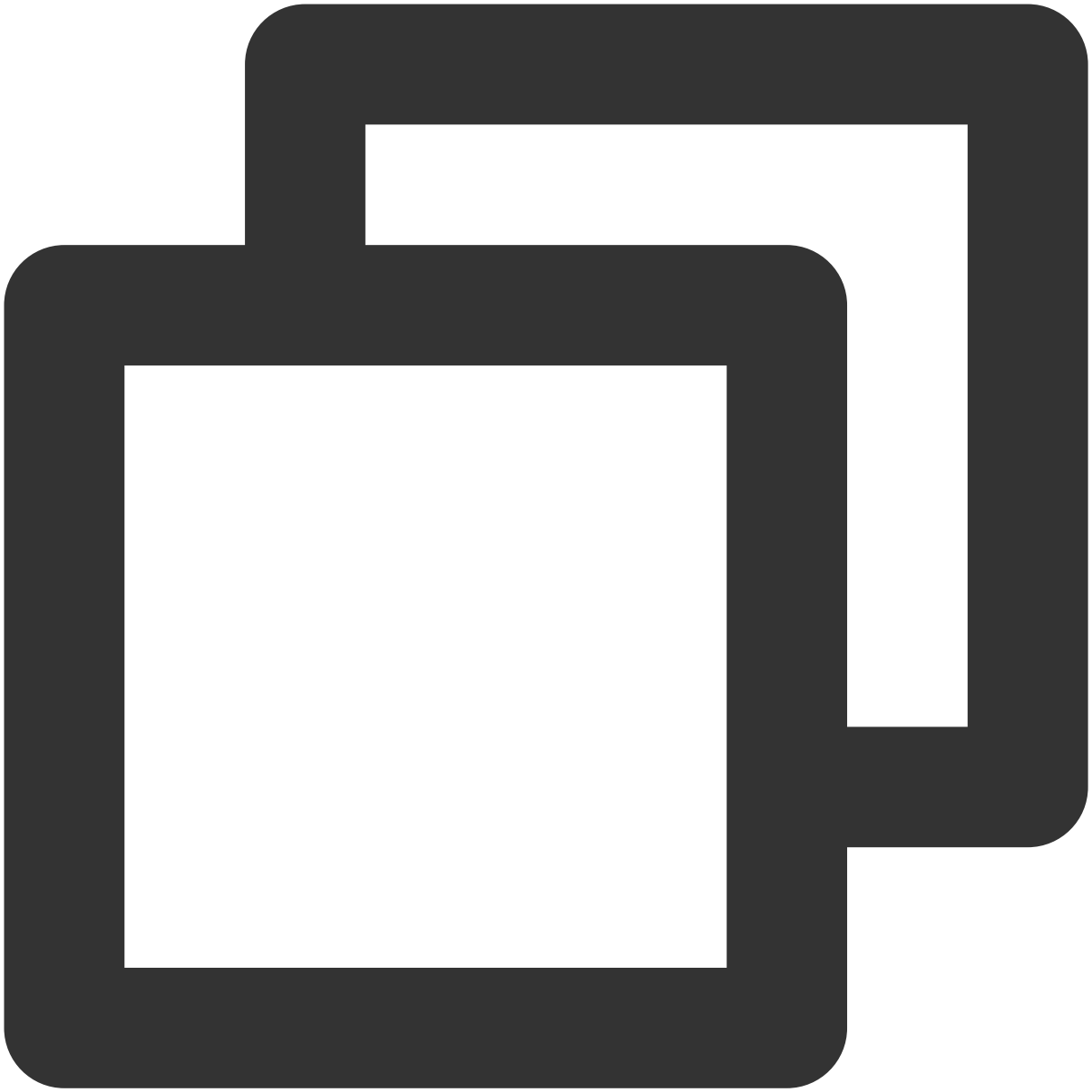
示例：



```
> SELECT nullif(2, 2);
 NULL
```

NVL

函数语法：



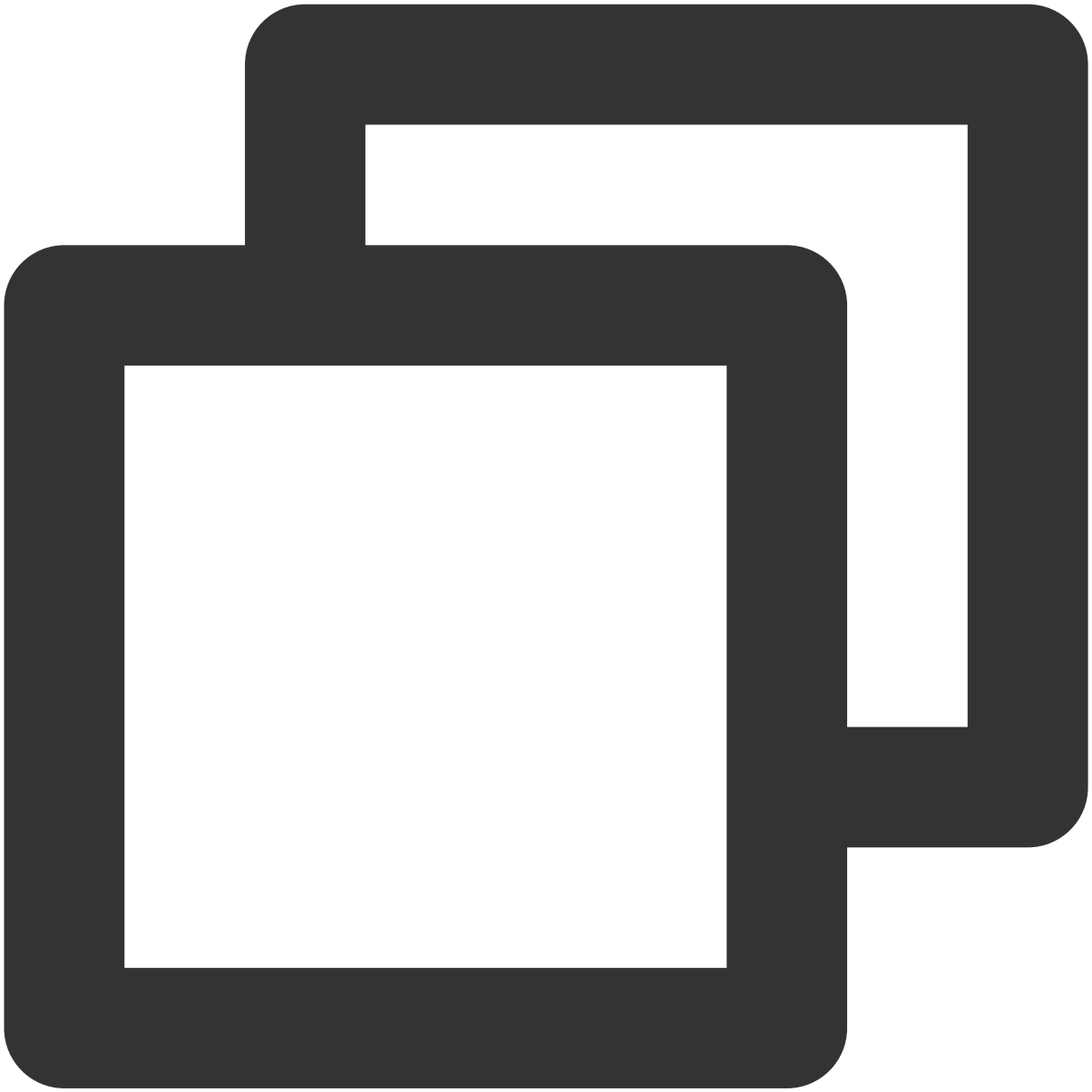
```
NVL(<expr1> T, <expr2> U)
```

支持引擎：SparkSQL、Presto

使用说明：如果expr1为null，则返回expr2，否则返回expr1。

返回类型：T|U

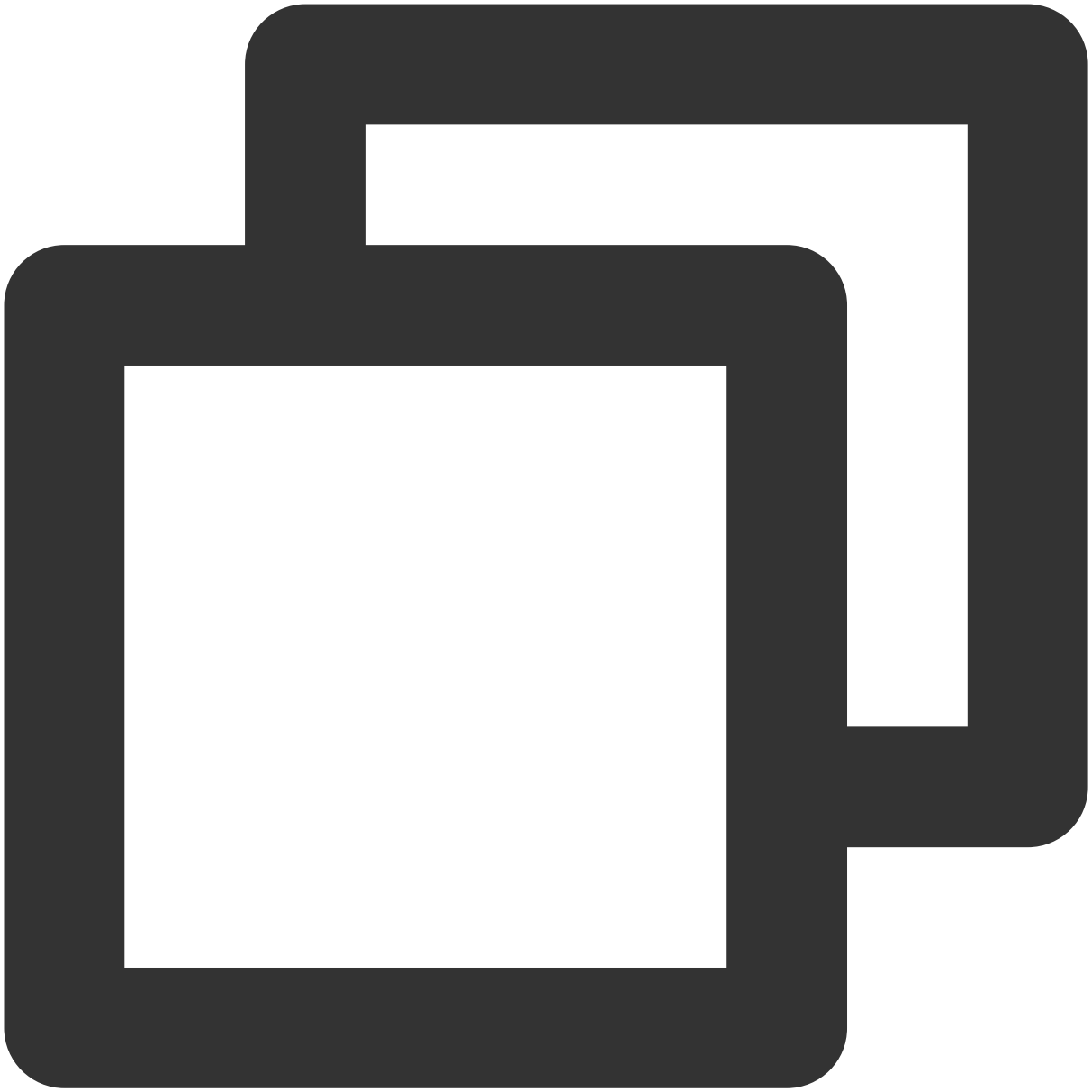
示例：



```
> SELECT nvl(NULL, array('2'));
 ["2"]
```

NVL2

函数语法：



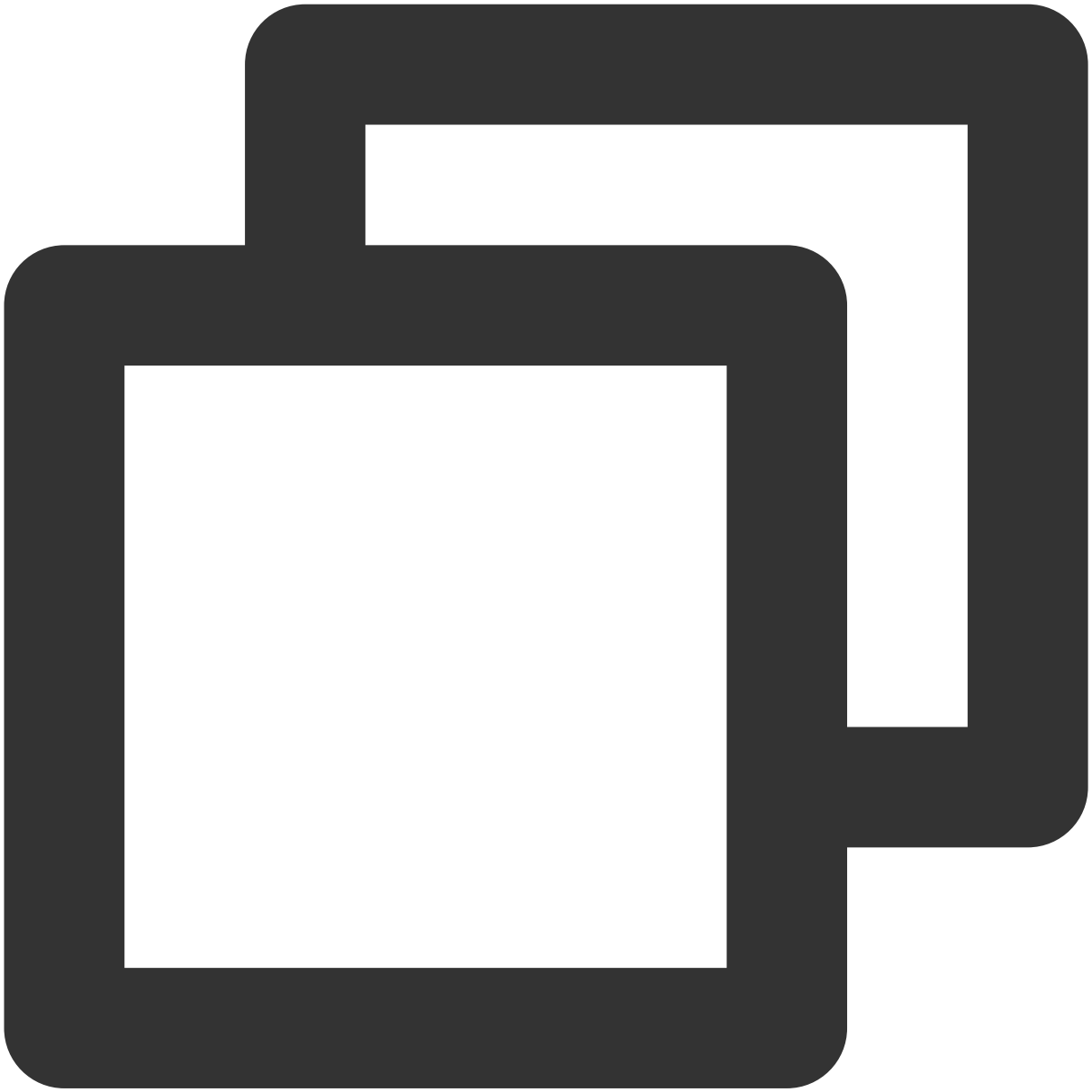
```
NVL2(<expr1> T1, <expr2> T2, <expr3> T3)
```

支持引擎：SparkSQL、Presto

使用说明：如果expr1不为null，则返回expr2，否则返回expr3。

返回类型：T2|T3

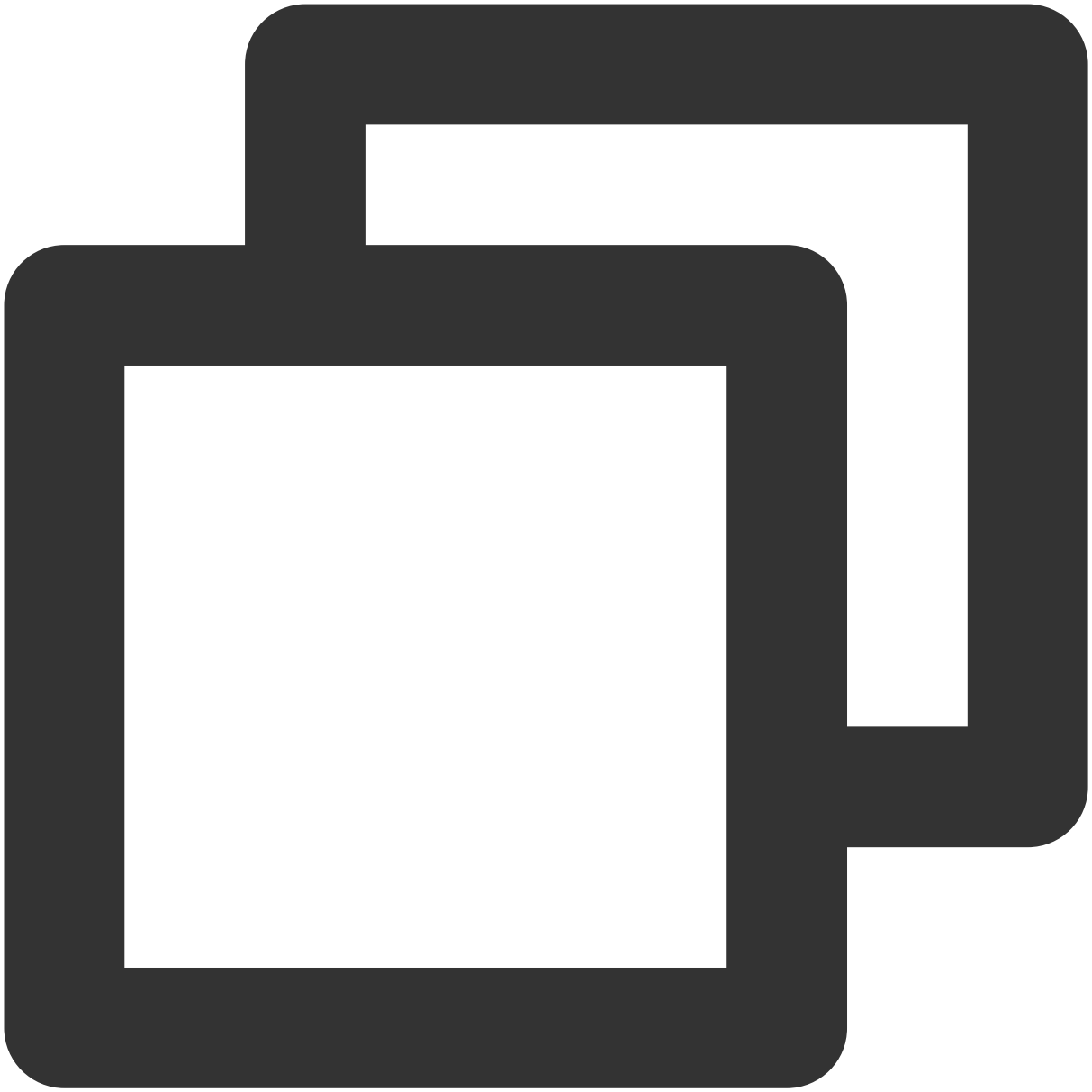
示例：



```
> SELECT nvl2(NULL, 2, 1);
 1
```

POSEXPLODE

函数语法：



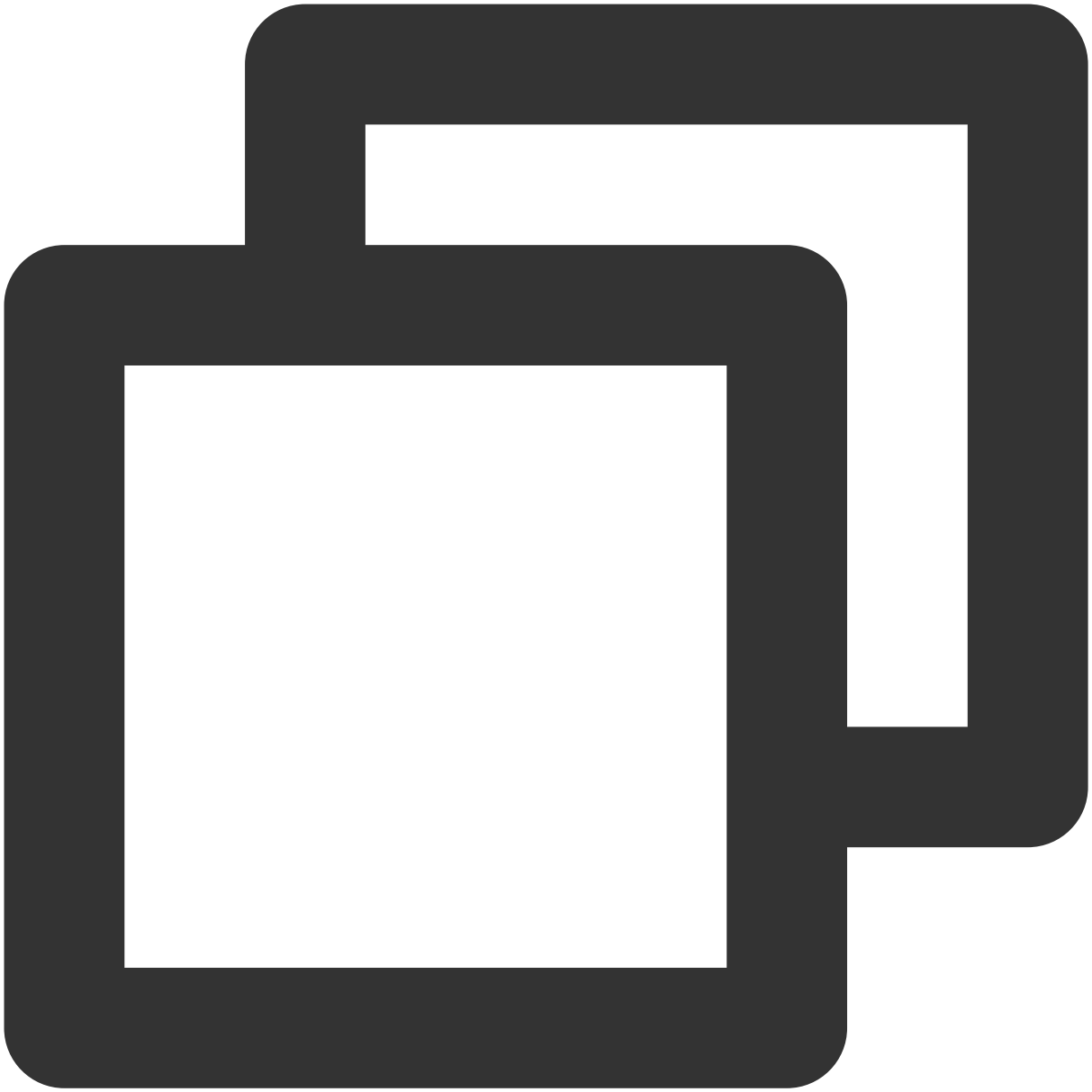
```
POSEXPLODE(<expr> array<T>|map<K, V>)
```

支持引擎：SparkSQL

使用说明：将array类型的expr的元素分隔为多行，或将map类型的expr分隔为多个行和列。使用列名pos表示位置,对数组的元素使用默认列名col，或对映射的元素使用key和value。

返回类型：row(pos integer, col T)|row(row integer, key K, value V)

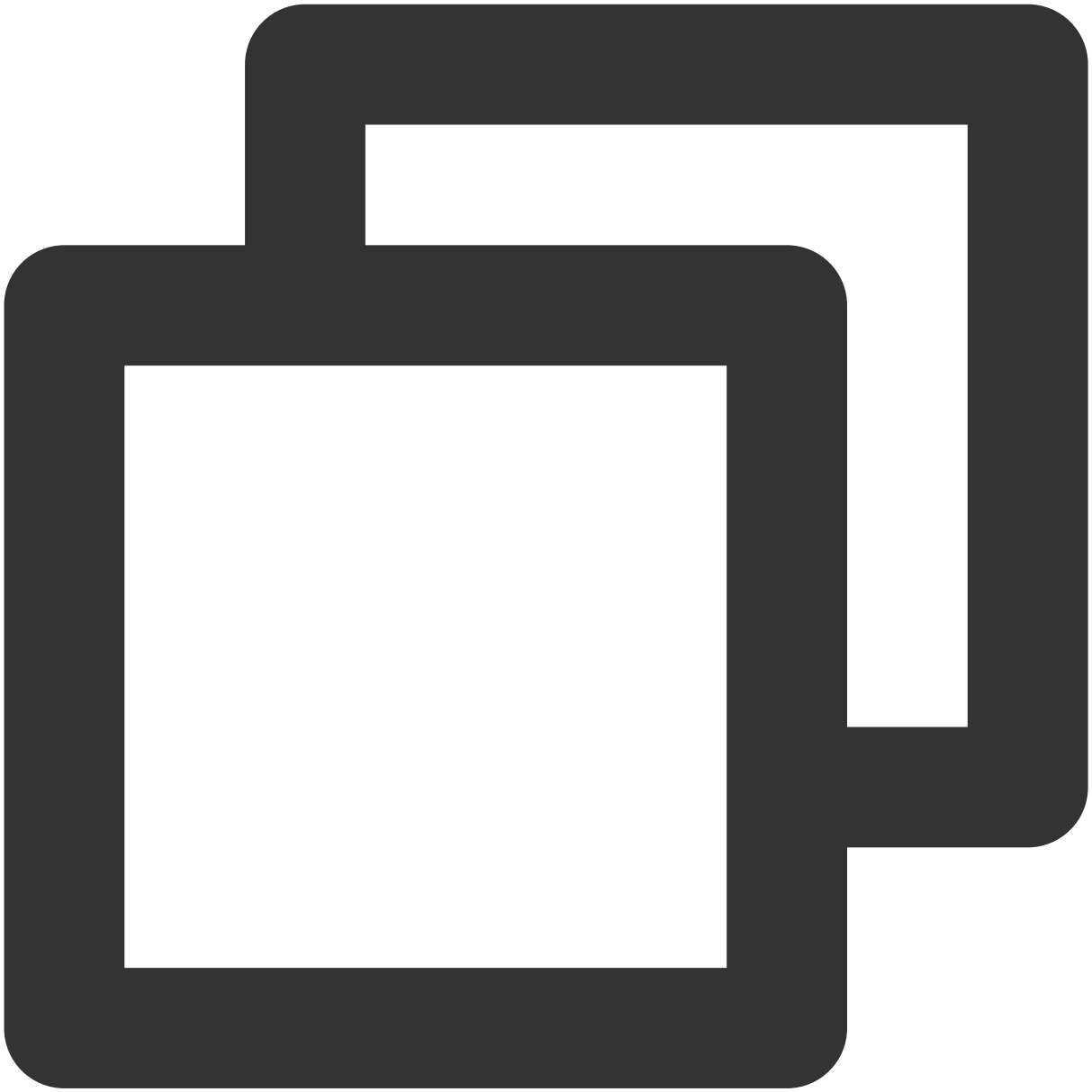
示例：



```
> SELECT posexplode(array(10,20));
 0  10
 1  20
```

POSEXPLODE_OUTER

函数语法：



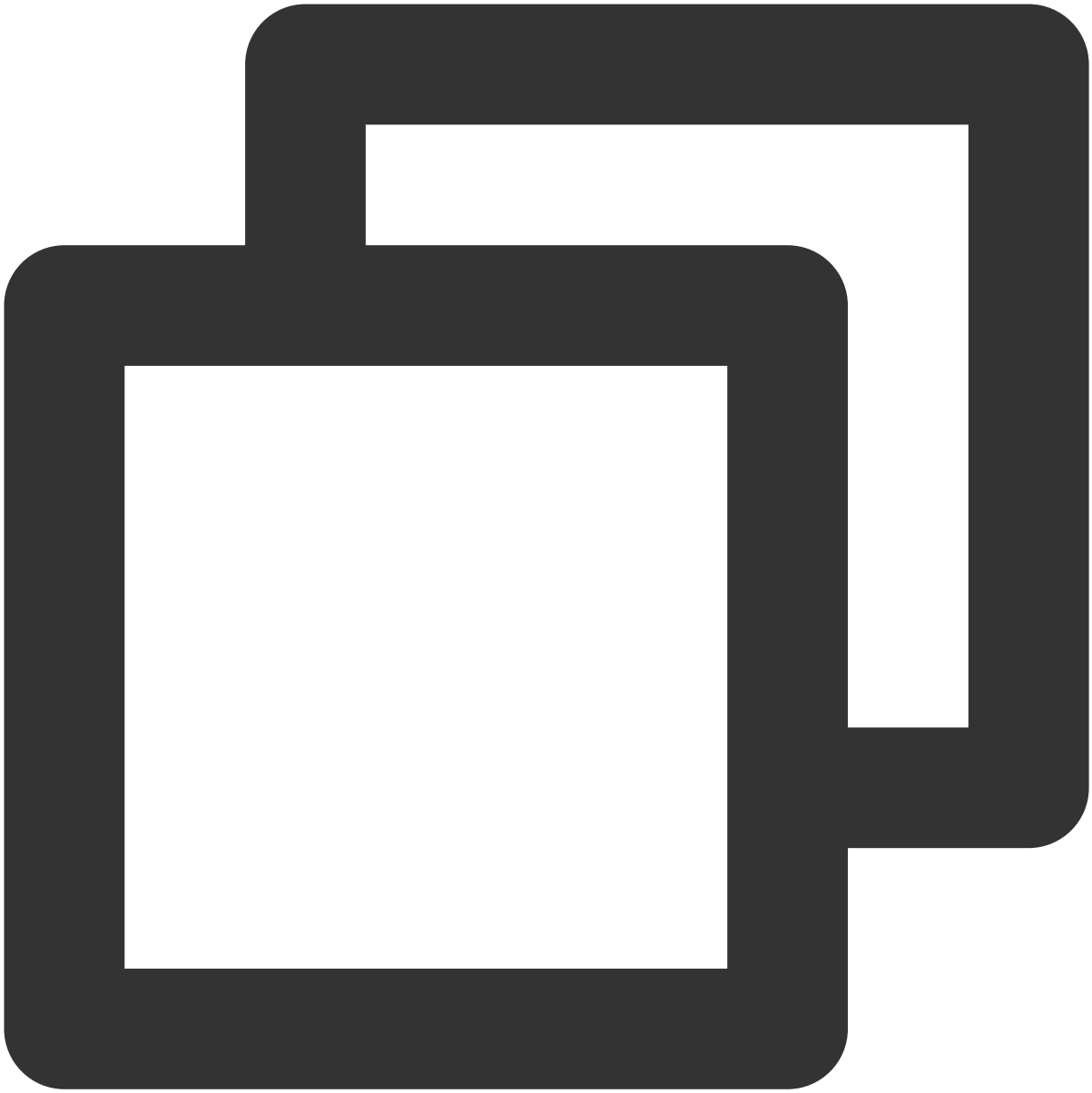
```
POSEXPLODE_OUTER(<expr> array<T>|map<K, V>)
```

支持引擎：SparkSQL

使用说明：将array类型的expr的元素分隔为多行，或将map类型的expr分隔为多个行和列。使用列名pos表示位置,对数组的元素使用默认列名col，或对映射的元素使用key和value。

返回类型：row(pos integer, col T)|row(row integer, key K, value V)

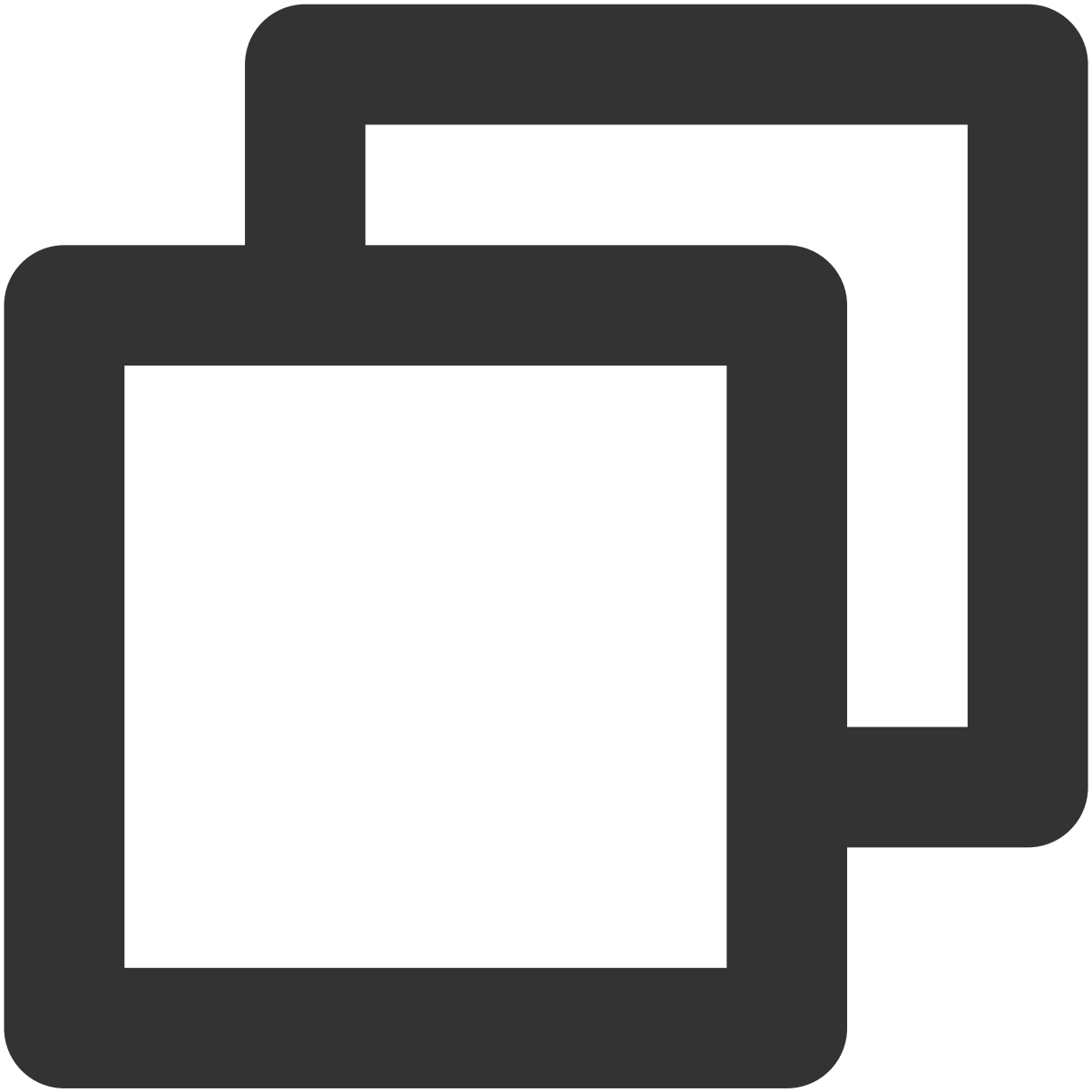
示例：



```
> SELECT posexplode_outer(array(10,20));
 0  10
 1  20
```

STACK

函数语法：



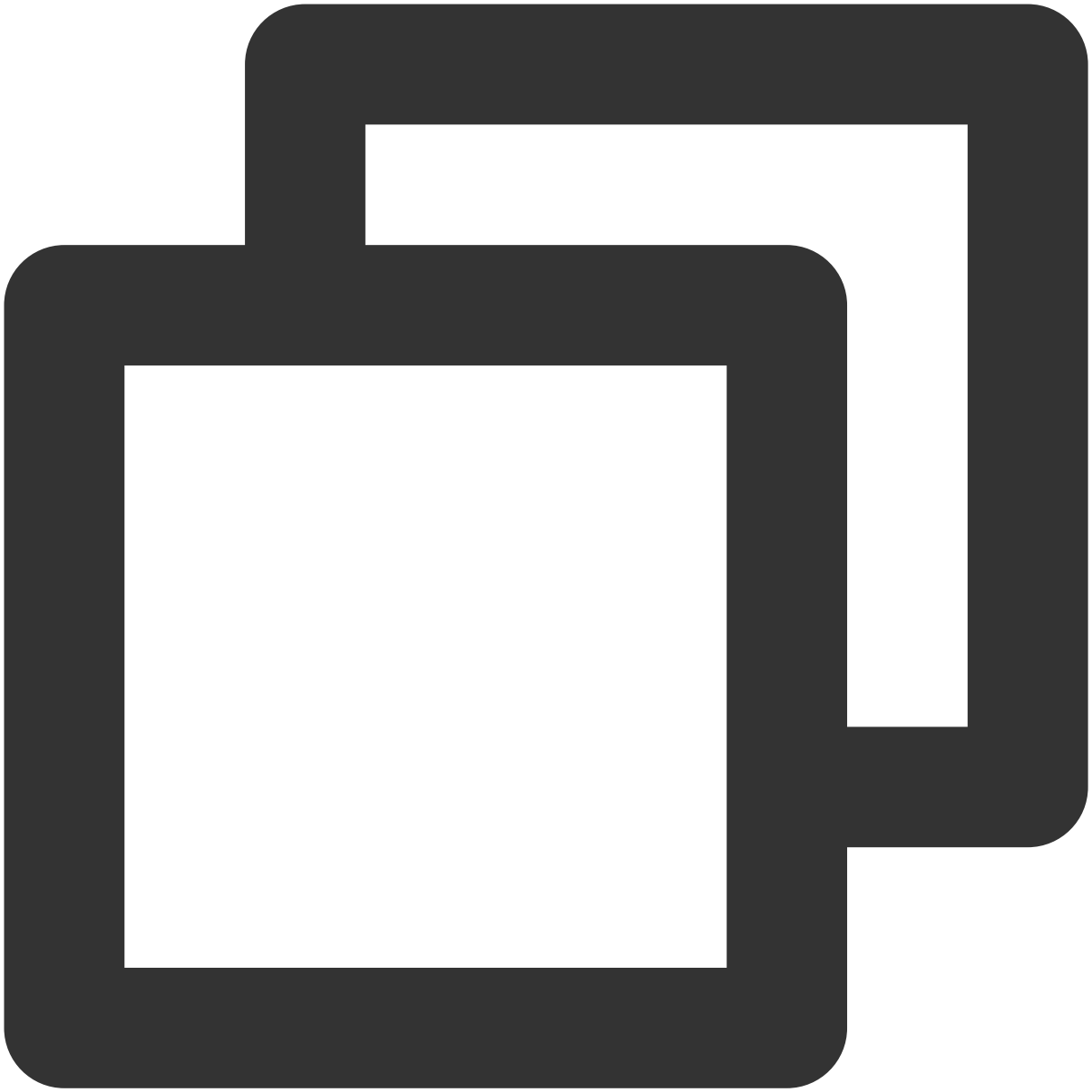
```
STACK(<n> integer, <expr0> T0, ..., <expr1> T1)
```

支持引擎：SparkSQL

使用说明：堆栈（n，expr1，...，exprk）-将expr1、...、exprk分隔为n行。默认情况下使用列名col0、col1等。

返回类型：row(col0 T0, ..., coln Tn)

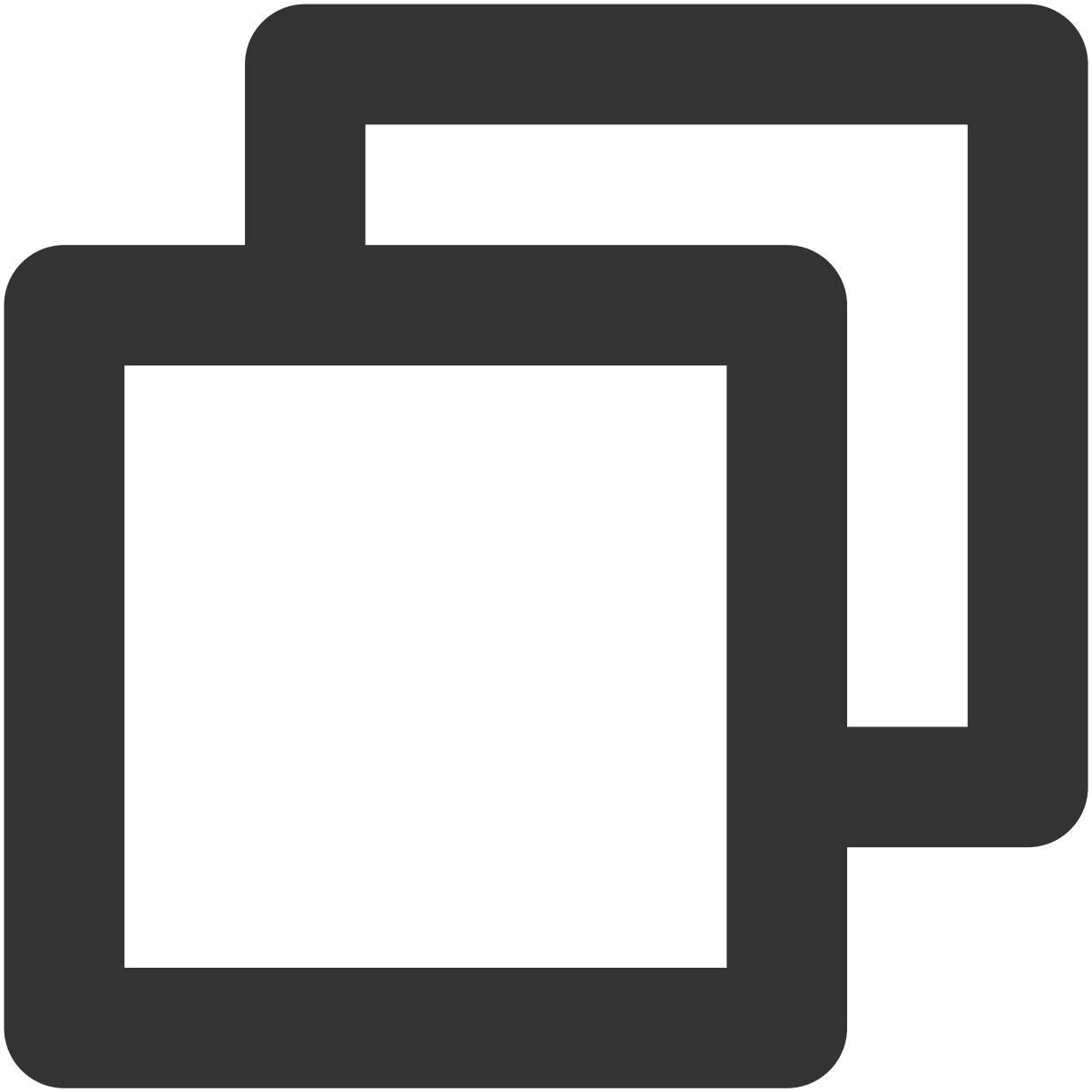
示例：



```
> SELECT stack(2, 1, 2, 3);
 1  2
 3  NULL
```

ASSERT_TRUE

函数语法：



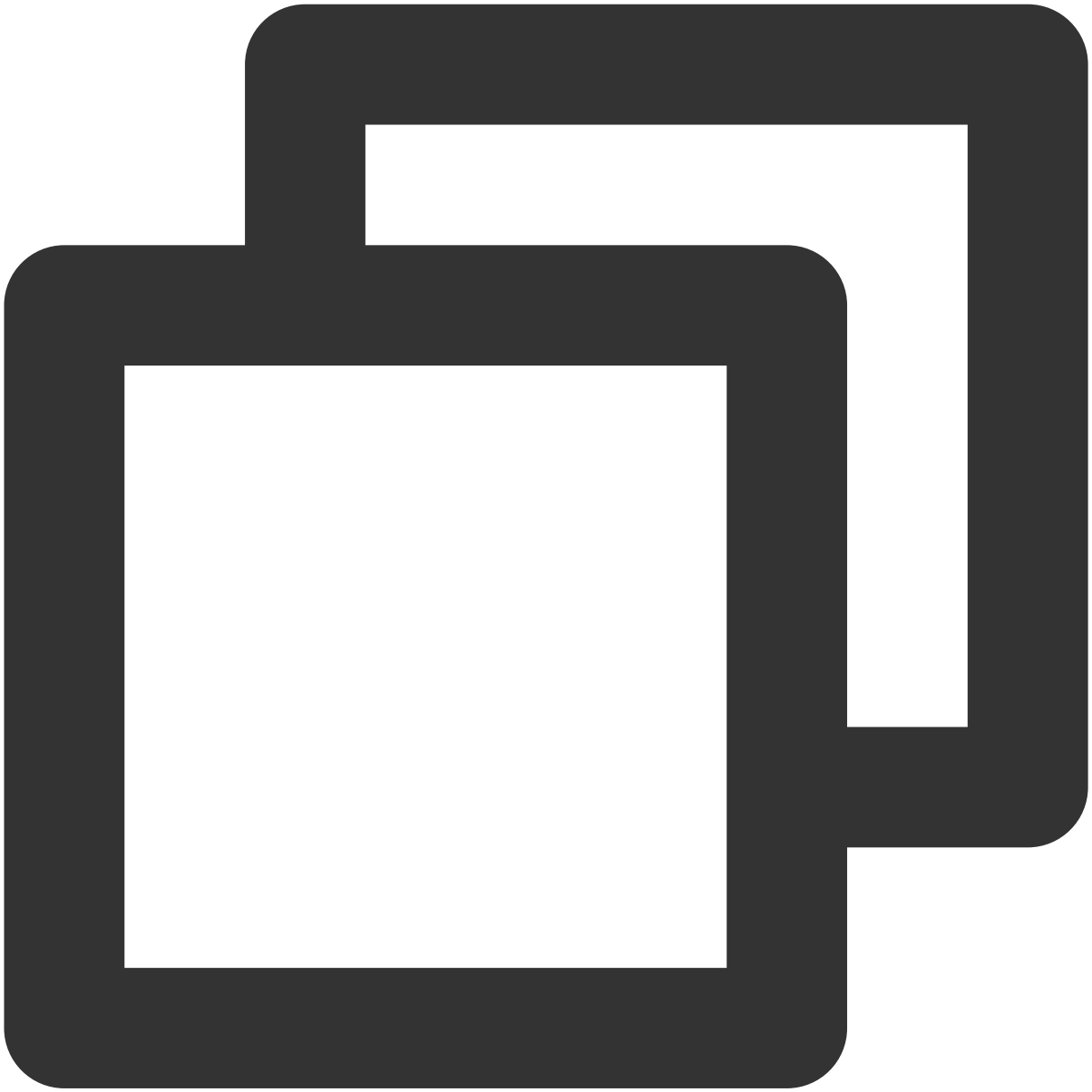
```
ASSERT_TRUE(<expr> boolean)
```

支持引擎：SparkSQL、Presto

使用说明：如果expr不为true，则抛出异常。

返回类型：boolean

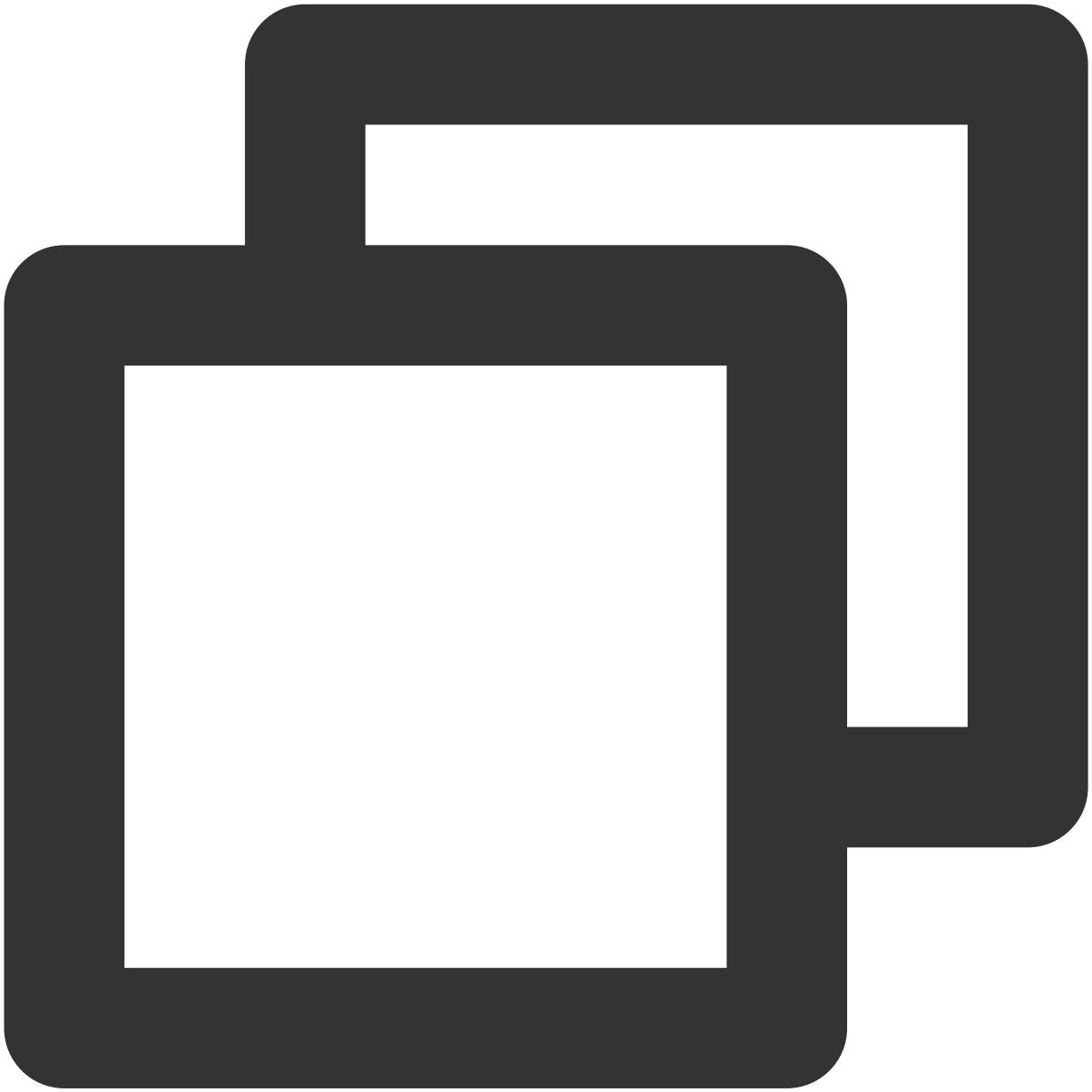
示例：



```
> SELECT assert_true(0 < 1);
 NULL
```

RAISE_ERROR

函数语法：



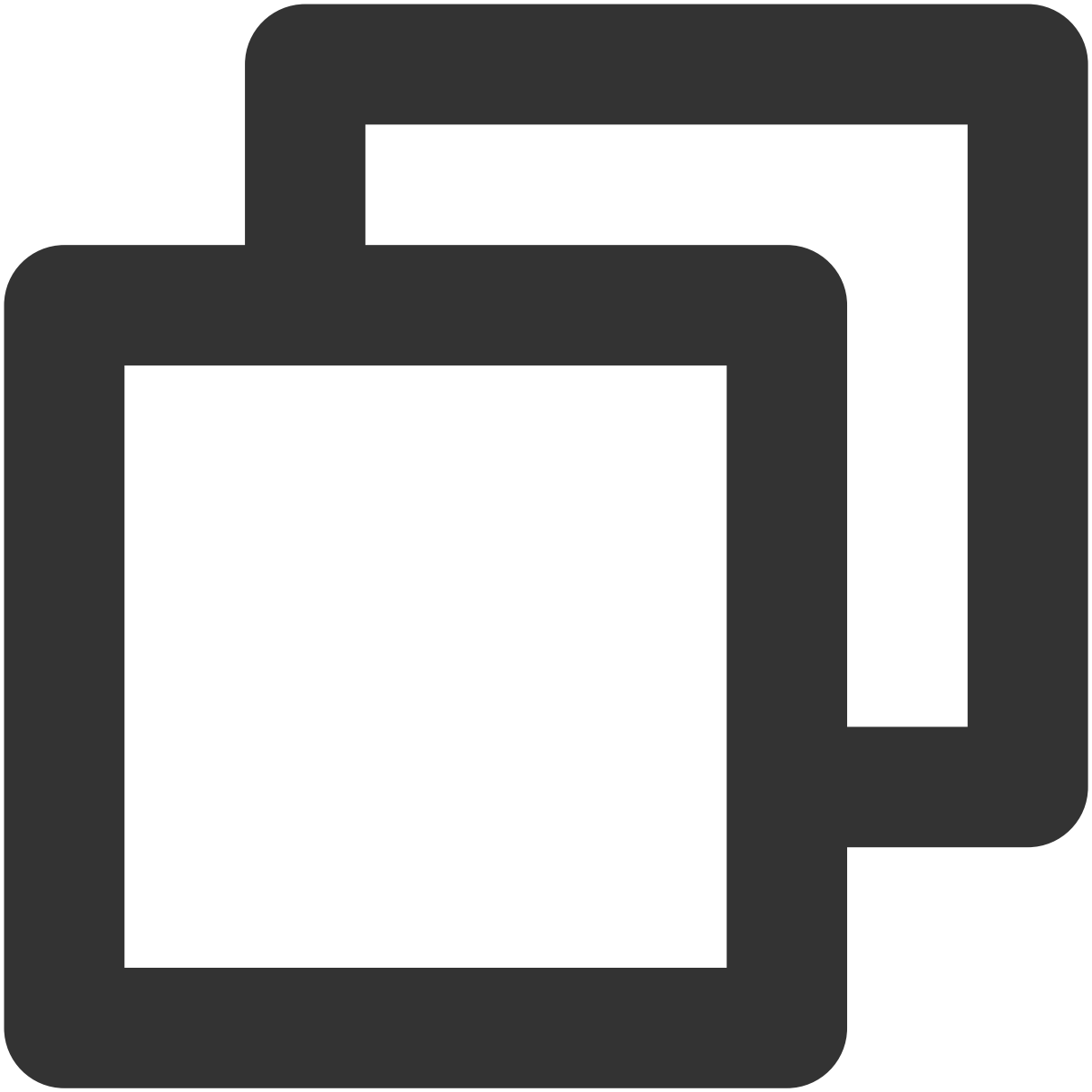
```
RAISE_ERROR(<error> string)
```

支持引擎：SparkSQL、Presto

使用说明：抛出expr异常。

返回类型：string

示例：



```
> SELECT raise_error('custom error message');
 java.lang.RuntimeException
 custom error message
```

SPARK_PARTITION_ID

函数语法：

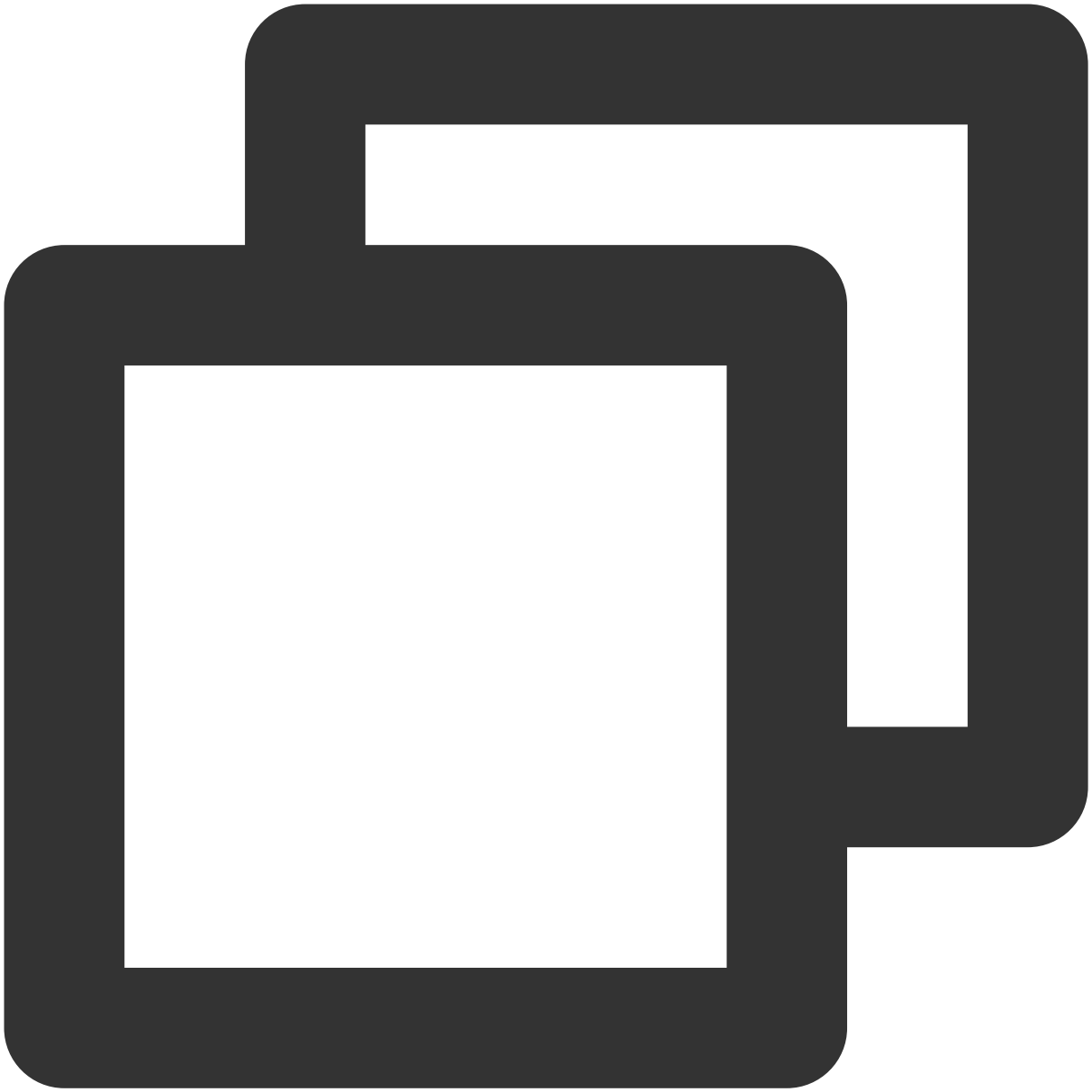
```
SPARK_PARTITION_ID()
```

支持引擎：SparkSQL

使用说明：返回当前分区id。

返回类型：integer

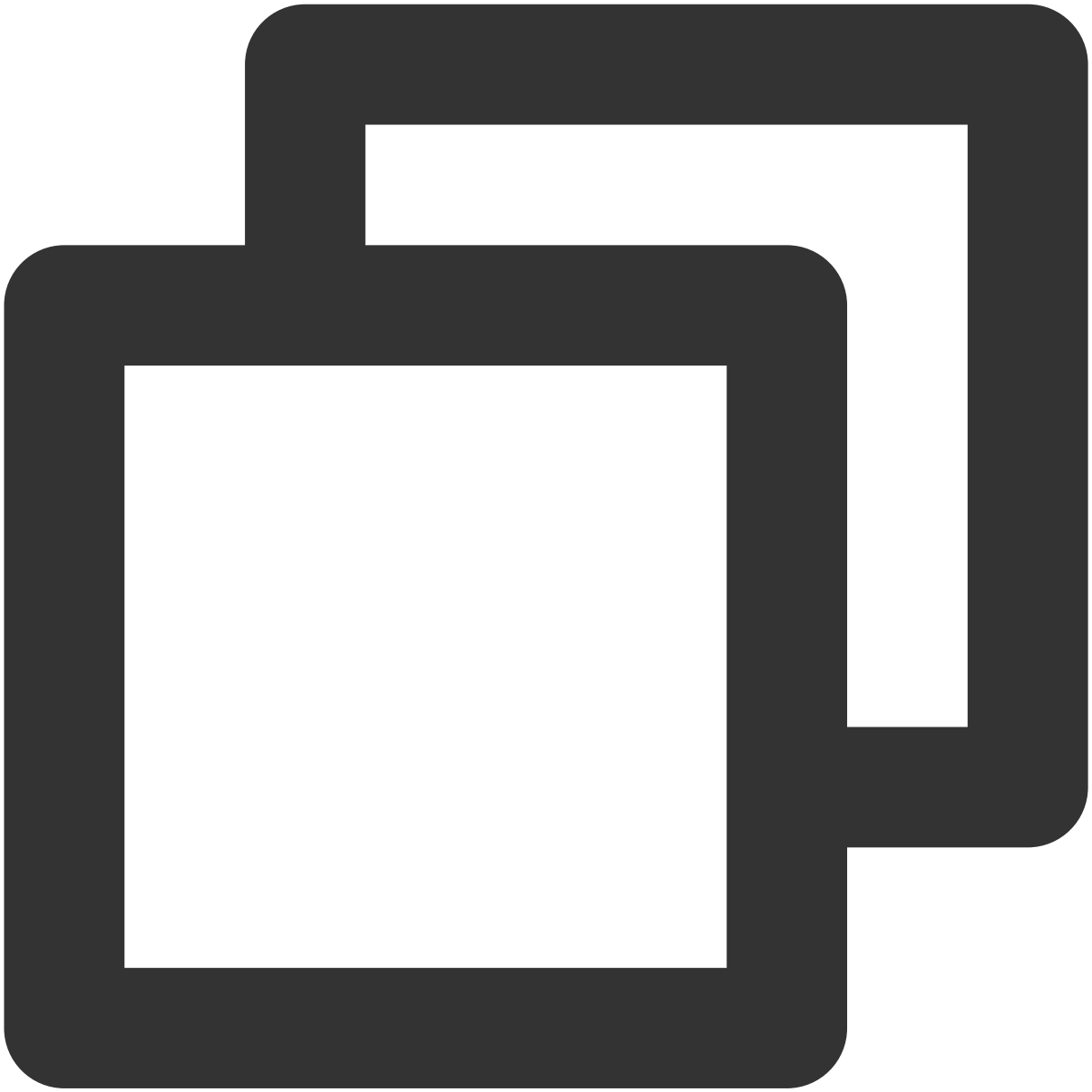
示例：



```
> SELECT spark_partition_id();
 0
```

INPUT_FILE_NAME

函数语法：



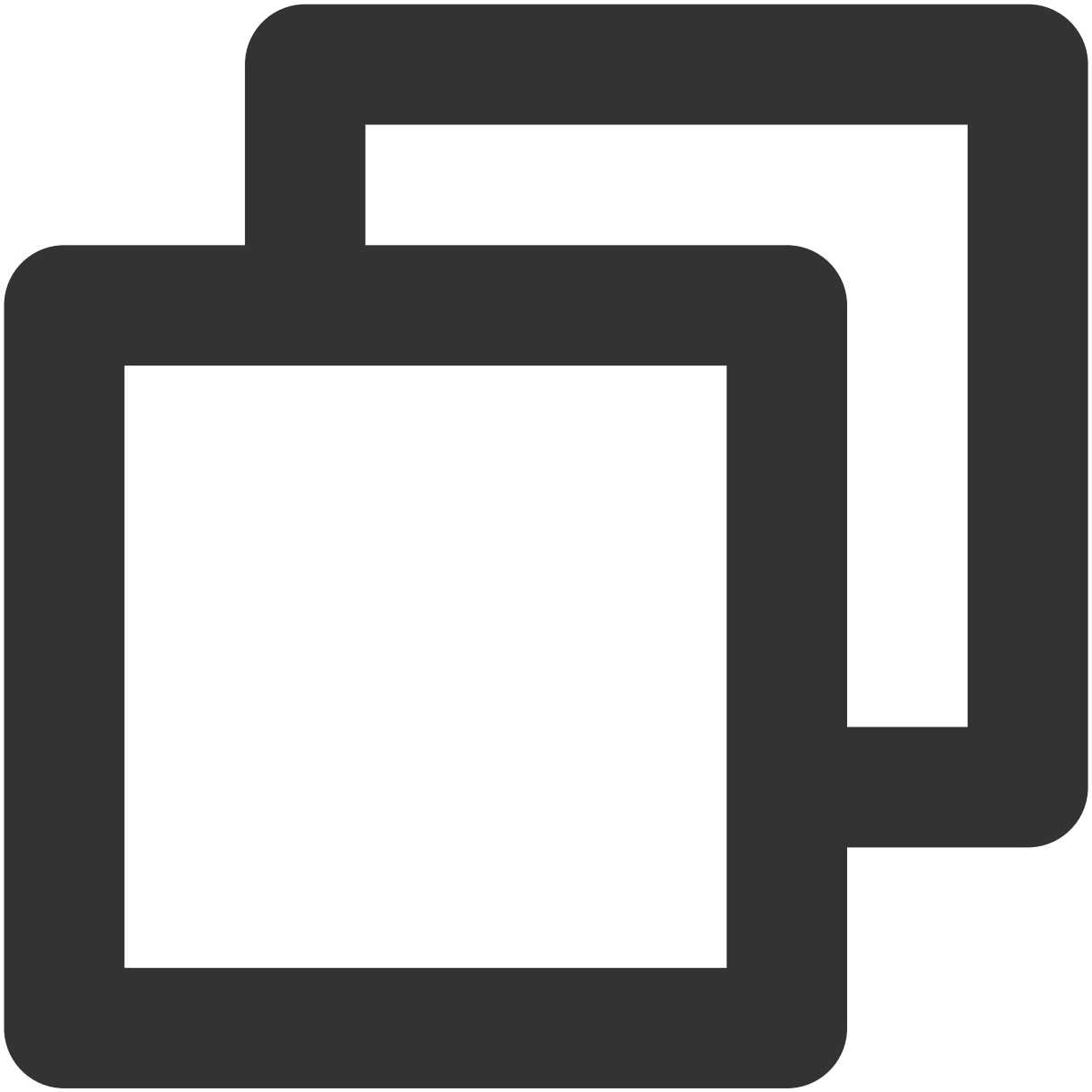
```
INPUT_FILE_NAME()
```

支持引擎：SparkSQL

使用说明：返回正在读取的文件的名称，如果不可用，则返回空字符串。

返回类型：string

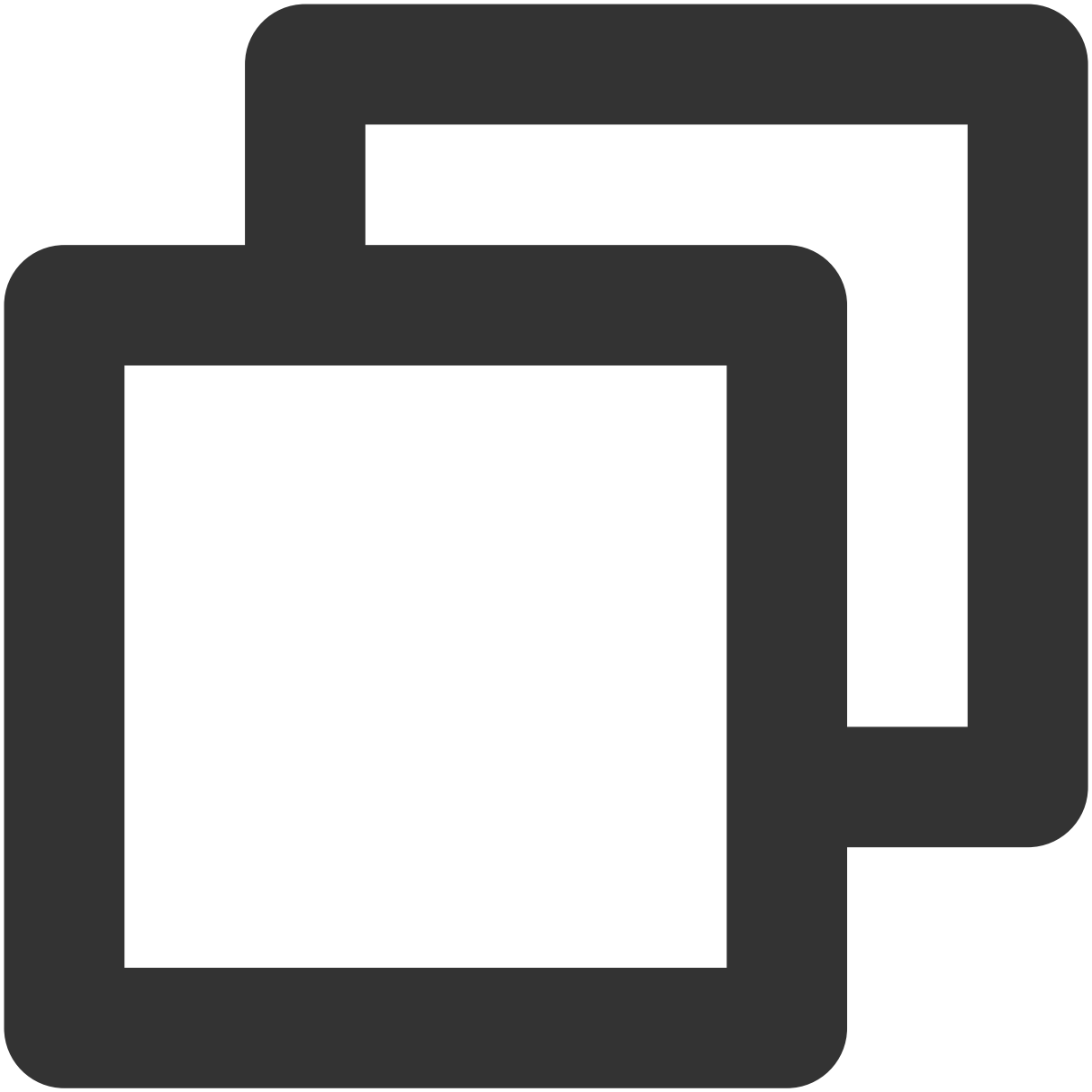
示例：



```
> SELECT input_file_name();
```

INPUT_FILE_BLOCK_START

函数语法：



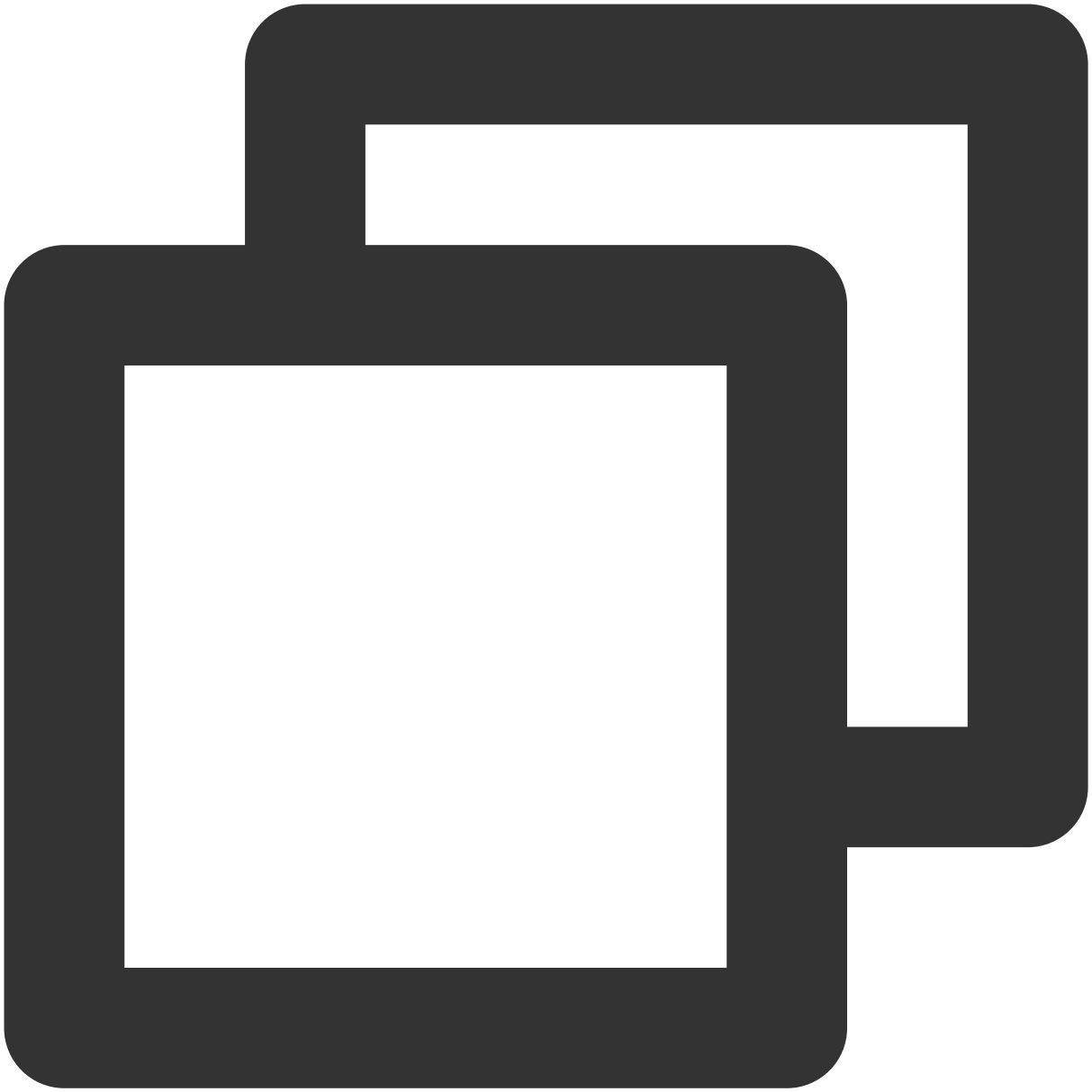
```
INPUT_FILE_BLOCK_START()
```

支持引擎：SparkSQL

使用说明：返回正在读取的块的开始偏移量，如果不可用，则返回-1。

返回类型：integer

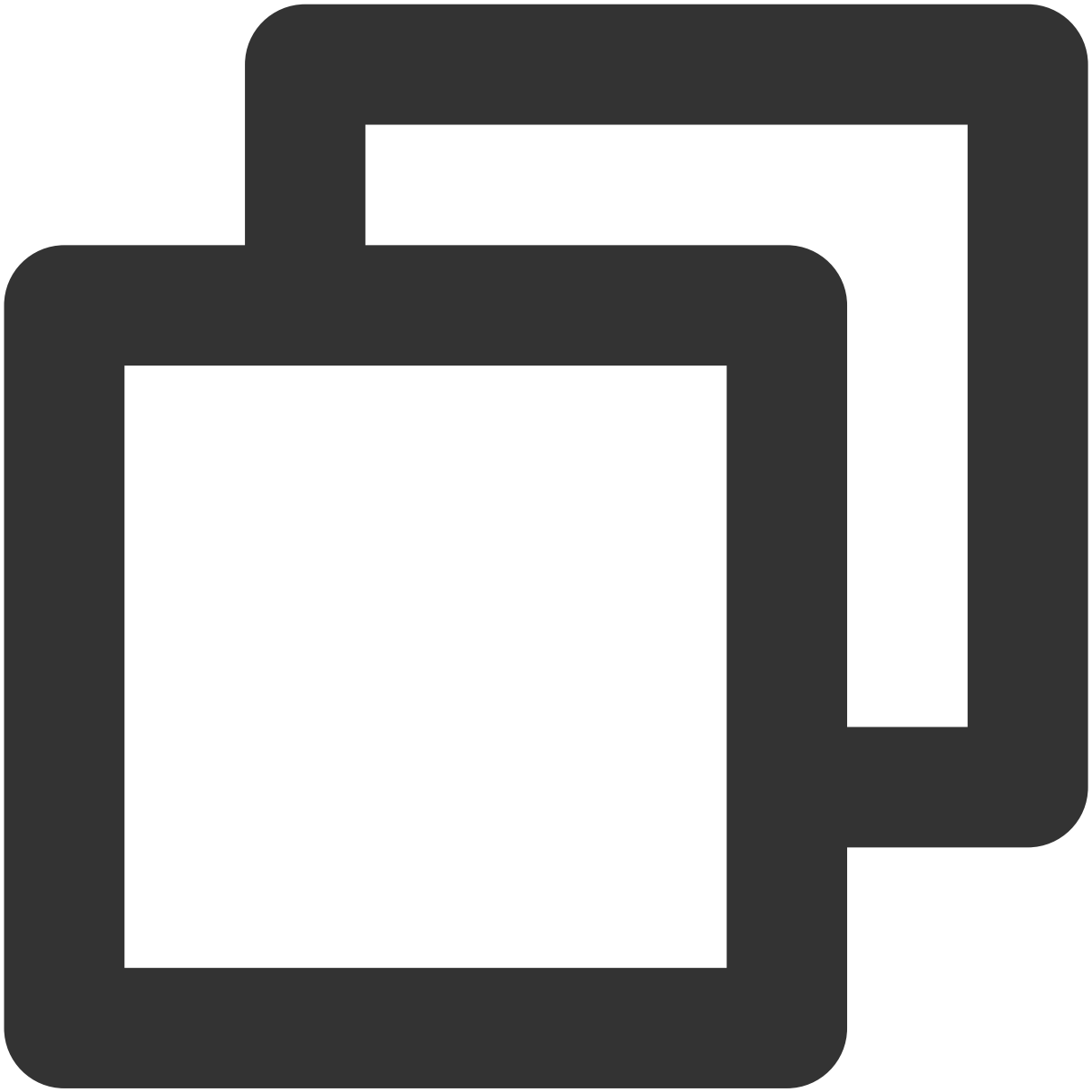
示例：



```
> SELECT input_file_block_start();
 -1
```

INPUT_FILE_BLOCK_LENGTH

函数语法：



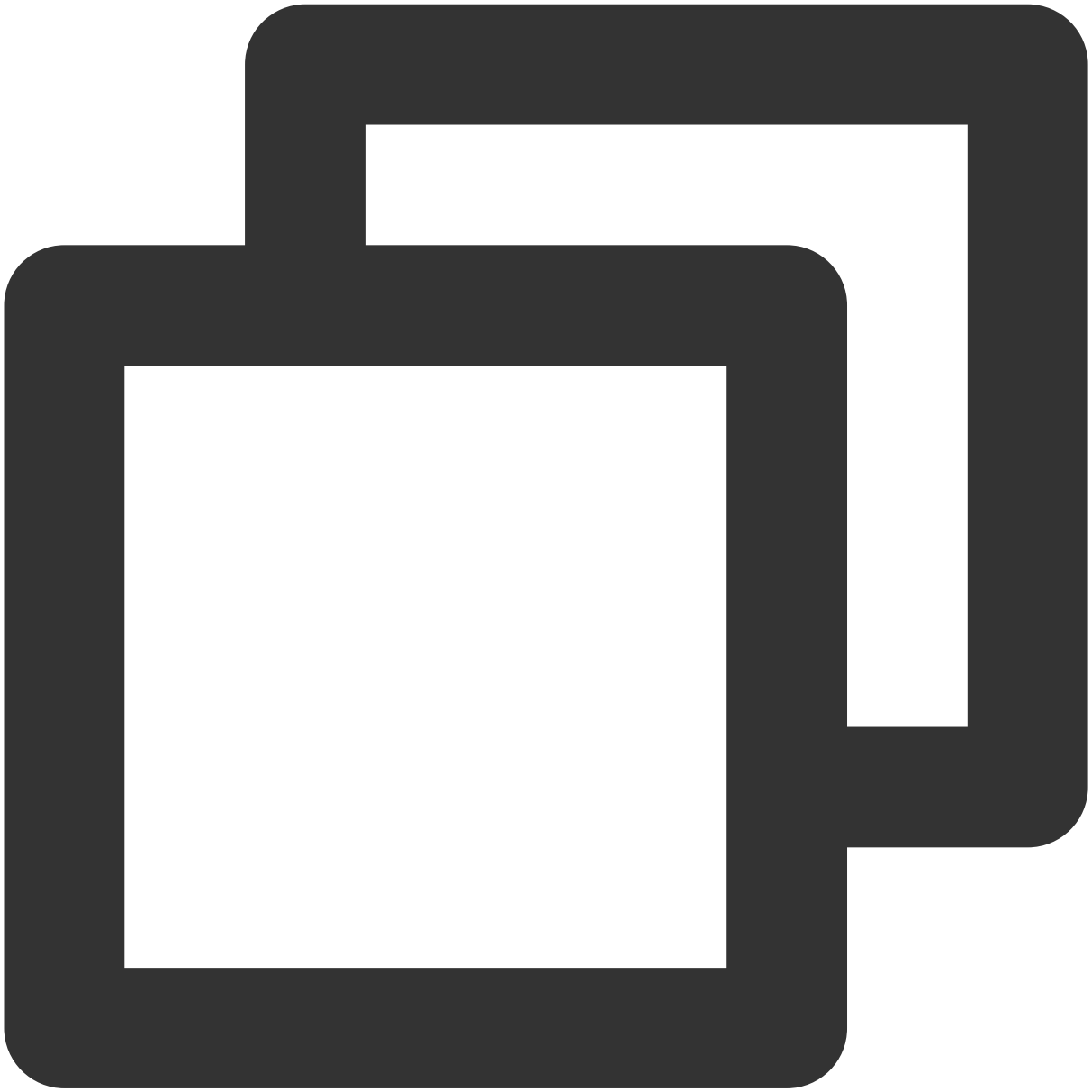
```
INPUT_FILE_BLOCK_LENGTH()
```

支持引擎：SparkSQL

使用说明：返回正在读取的块的长度，如果不可用，则返回-1。

返回类型：integer

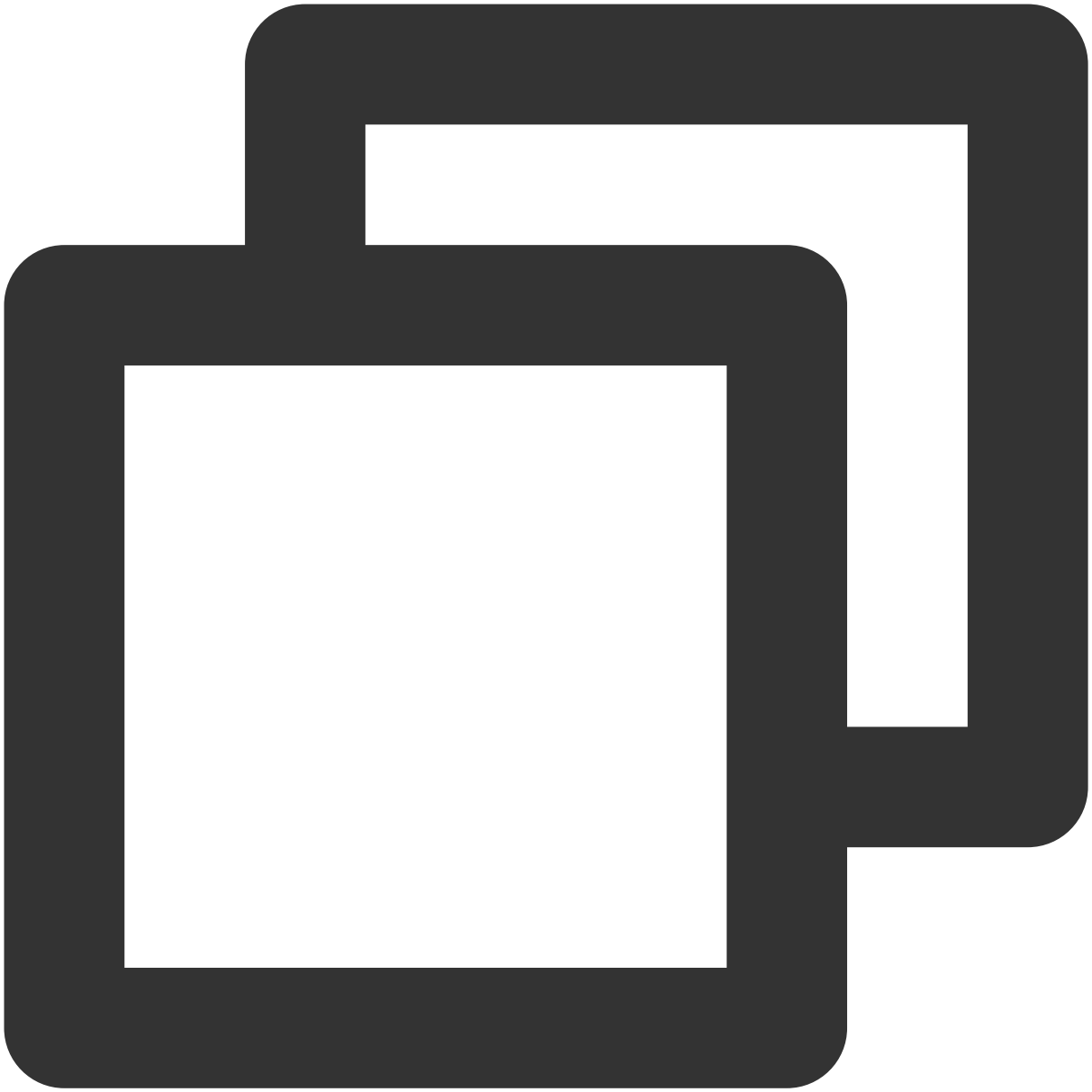
示例：



```
> SELECT input_file_block_length();
 -1
```

MONOTONICALLY_INCREASING_ID

函数语法：



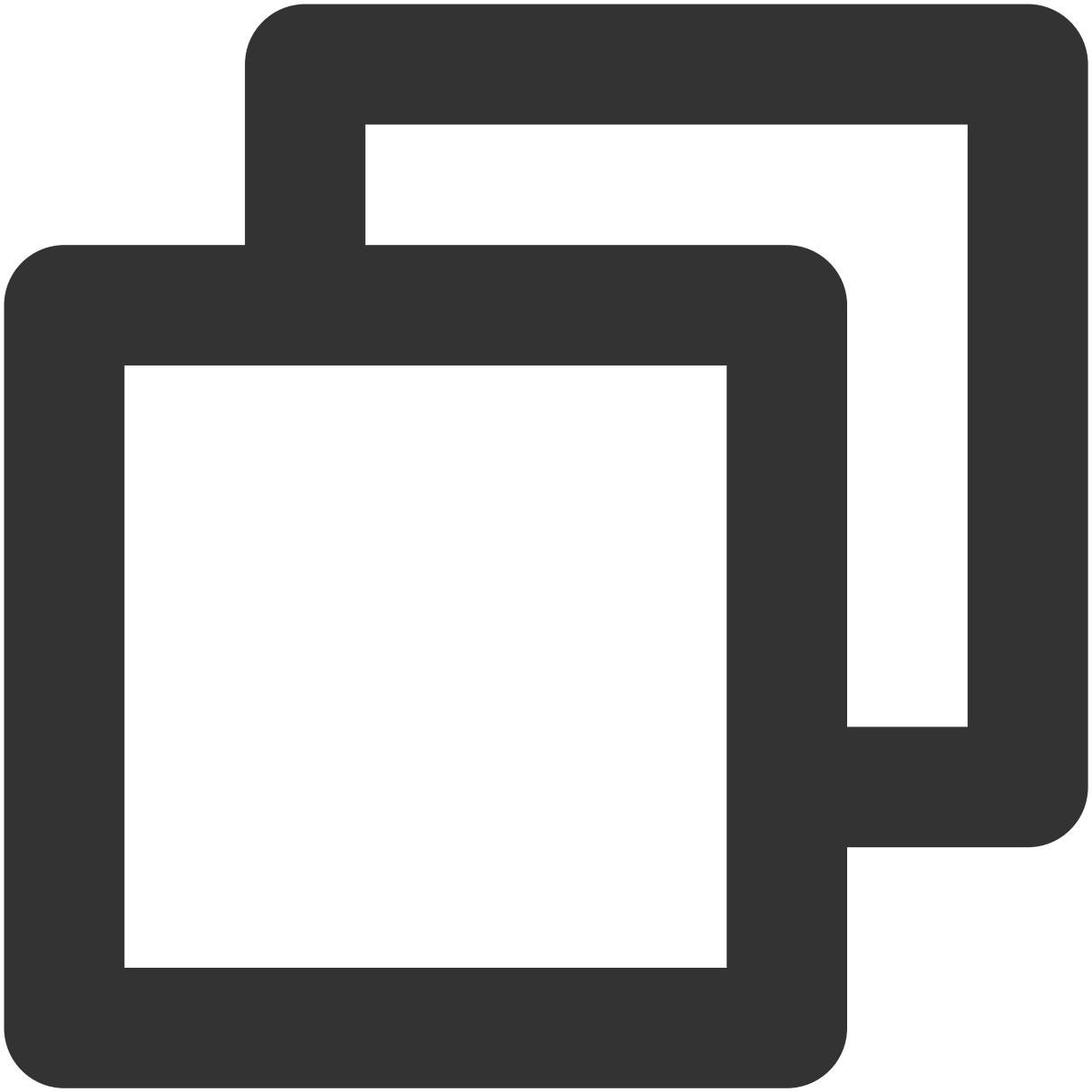
```
MONOTONICALLY_INCREASING_ID()
```

支持引擎：SparkSQL

使用说明：返回单调递增的64位整数。生成的ID保证单调递增且唯一，但不是连续的。当前实现将分区ID放在高31位，低33位表示每个分区内的记录号。假设数据帧的分区少于10亿，每个分区的记录少于80亿条。该函数是不确定的，因为其结果取决于分区ID。

返回类型：bigint

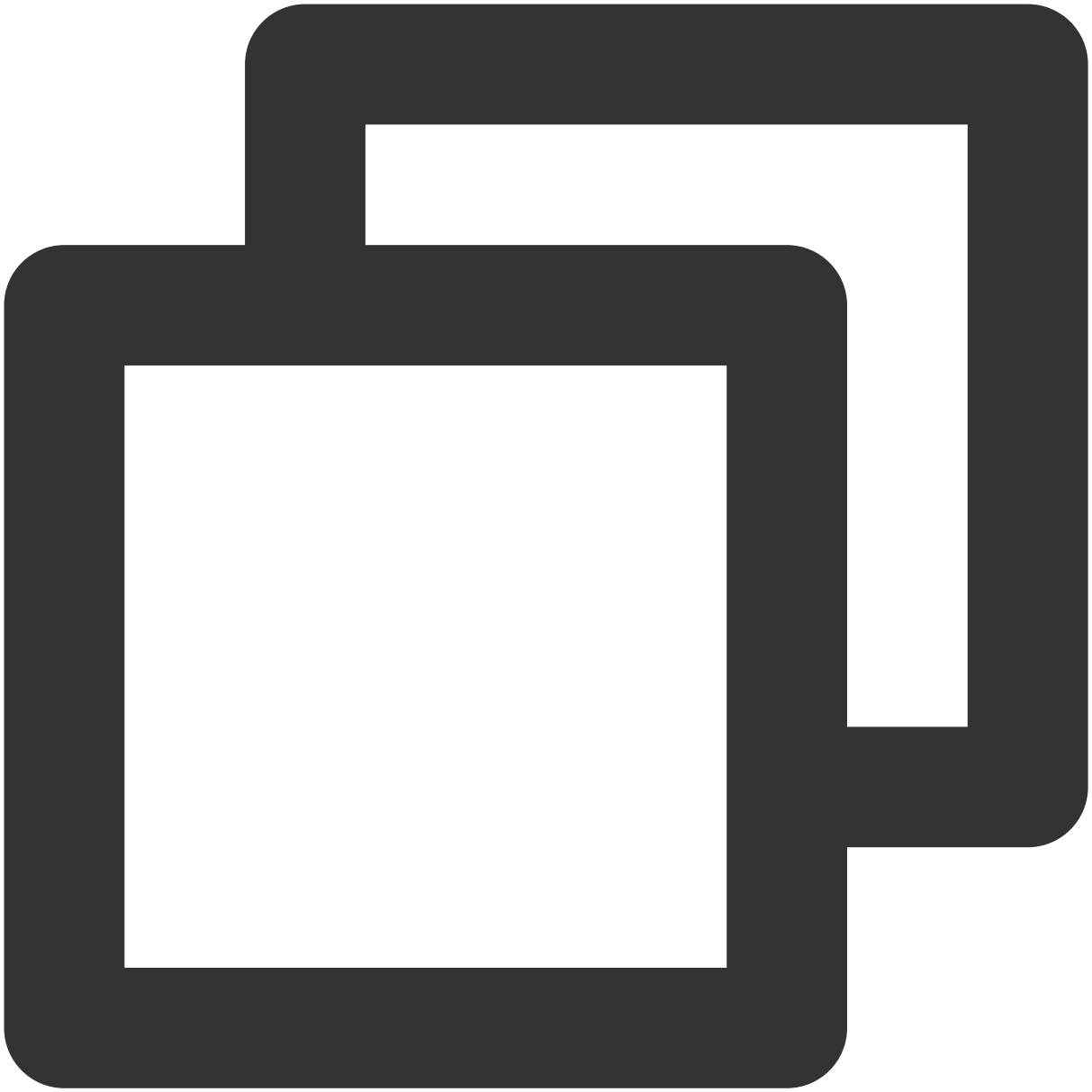
示例：



```
> SELECT monotonically_increasing_id();
 0
```

CURRENT_DATABASE

函数语法：



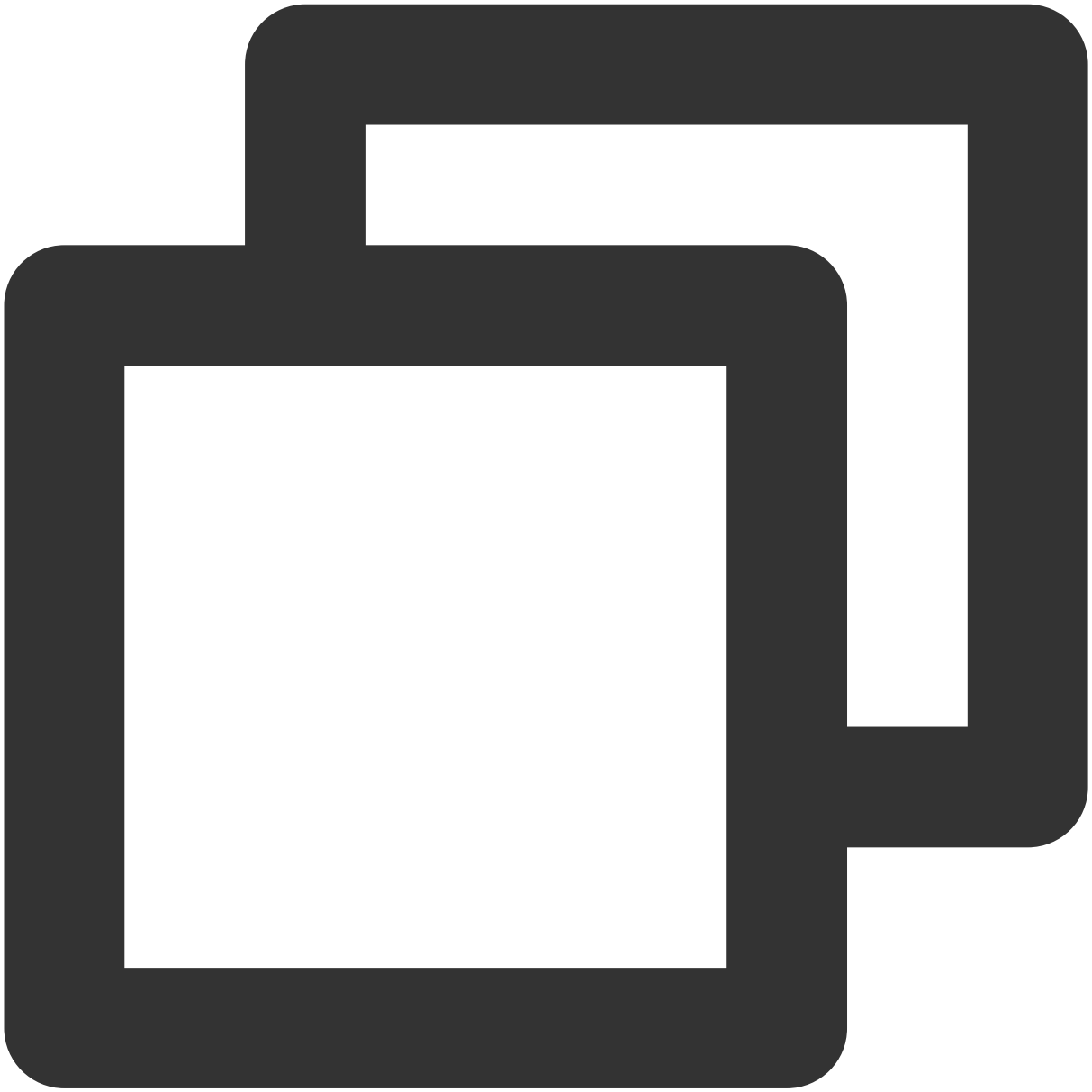
```
CURRENT_DATABASE()
```

支持引擎：SparkSQL

使用说明：返回当前database。

返回类型：string

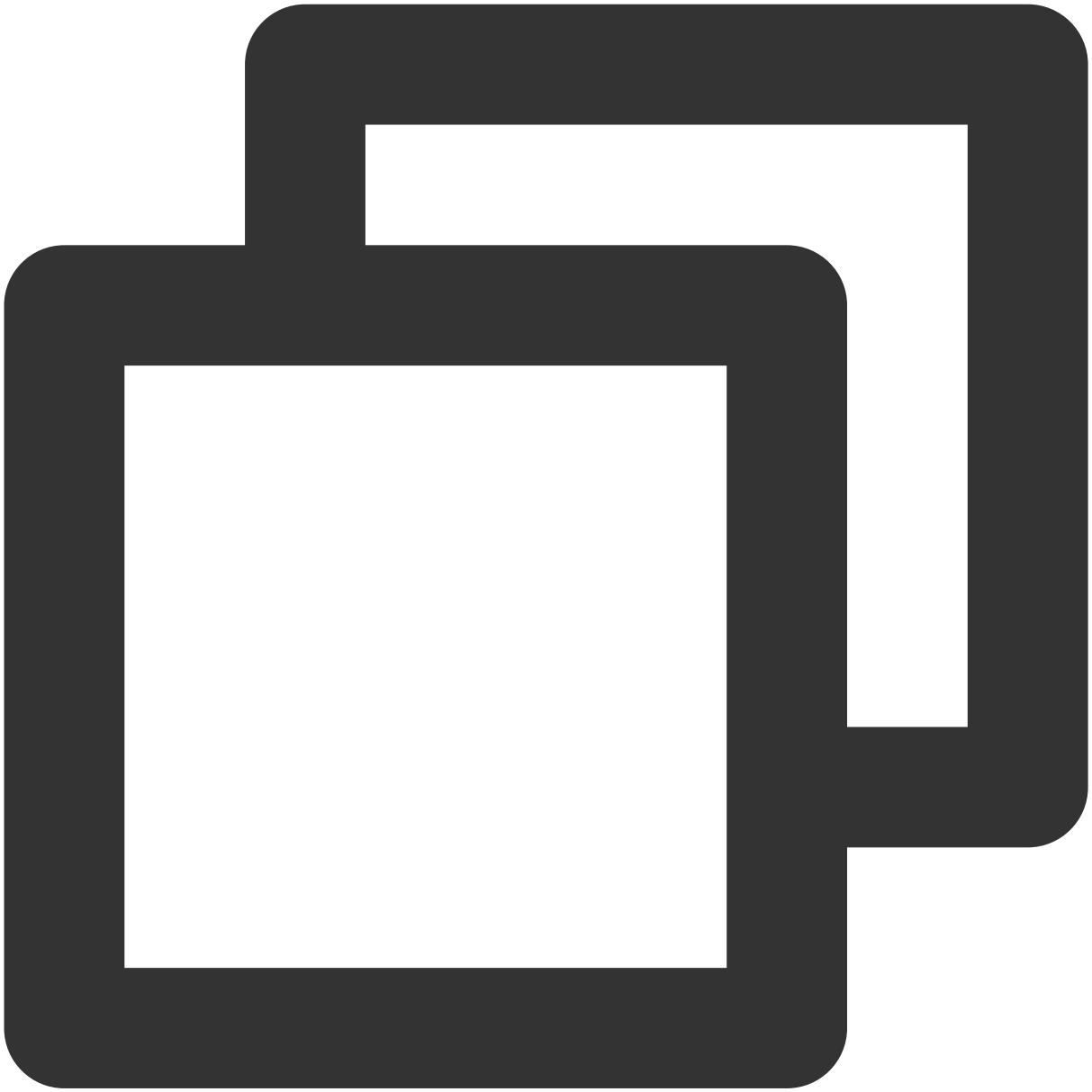
示例：



```
> SELECT current_database();
 default
```

CURRENT_CATALOG

函数语法：



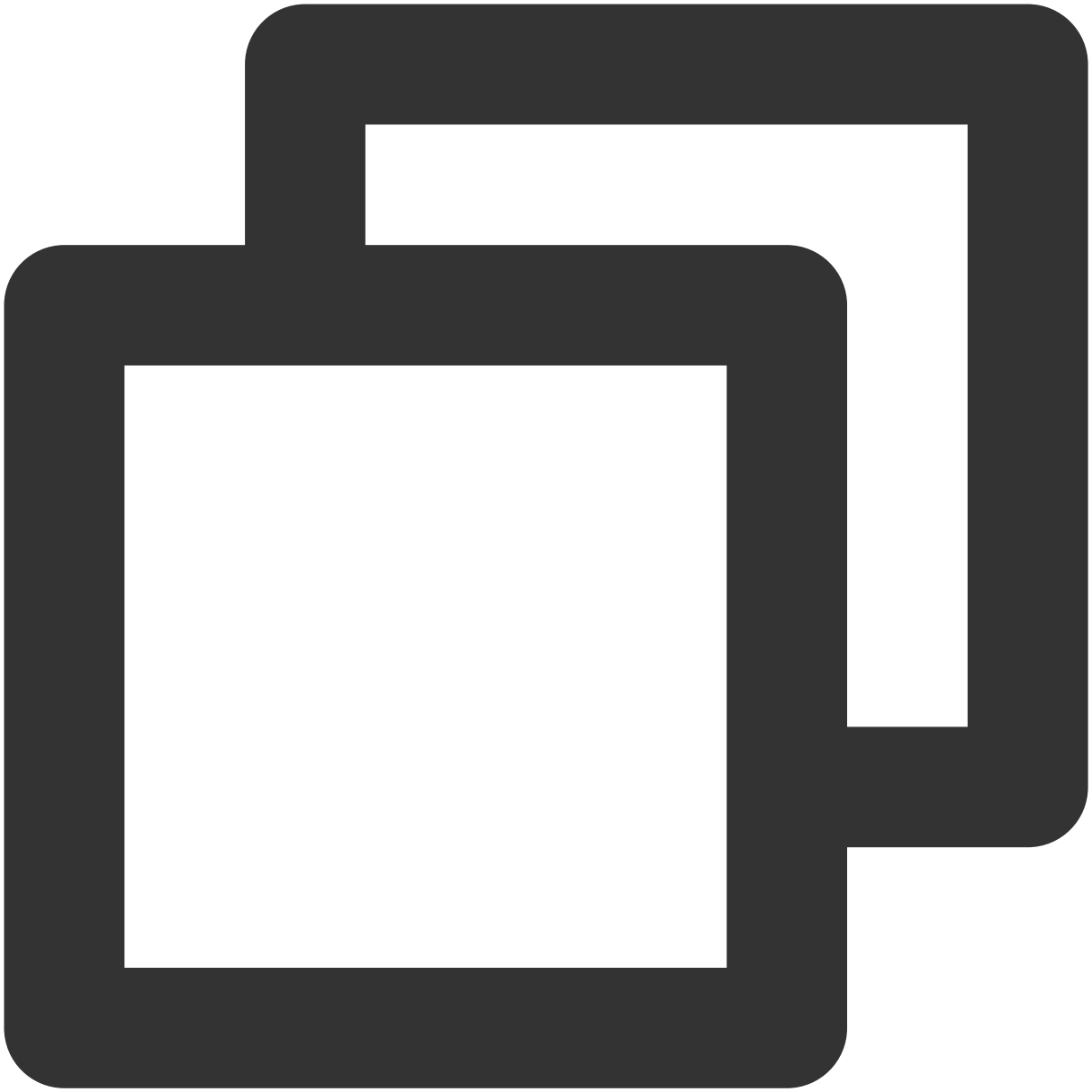
```
CURRENT_CATALOG()
```

支持引擎：SparkSQL

使用说明：返回当前catalog

返回类型：string

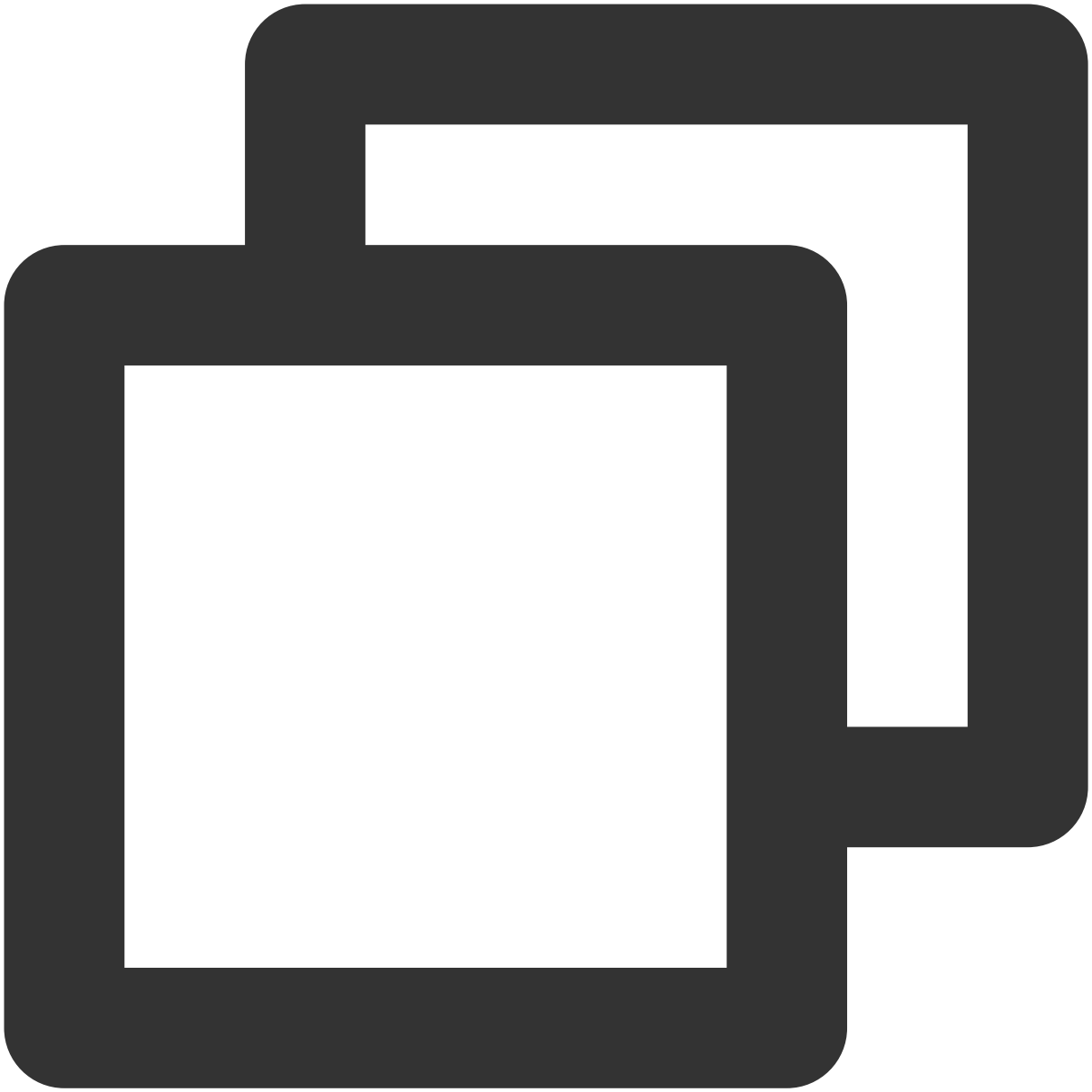
示例：



```
> SELECT current_catalog();
 spark_catalog
```

CURRENT_USER

函数语法：



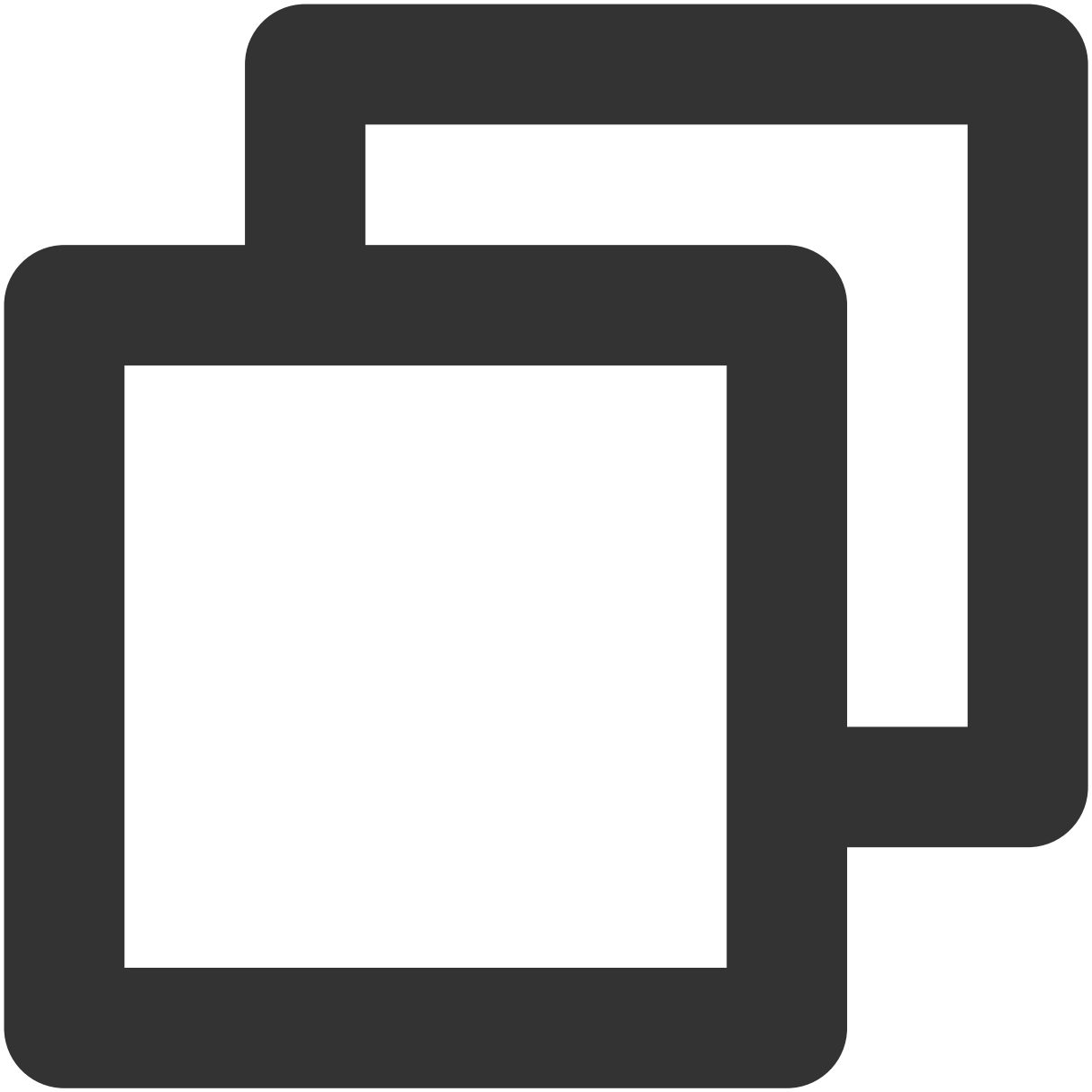
```
CURRENT_USER()
```

支持引擎：SparkSQL、Presto

使用说明：返回当前用户

返回类型：string

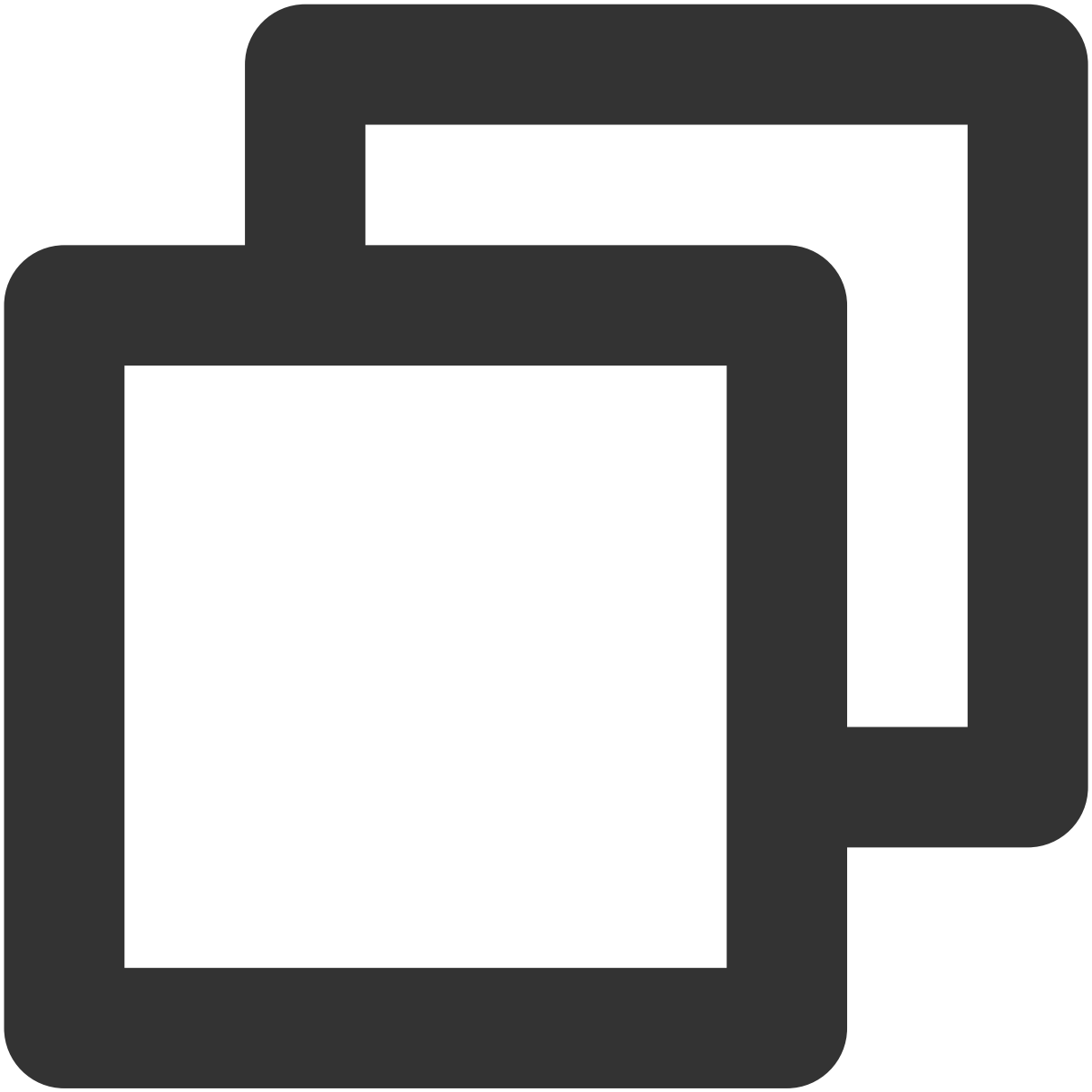
示例：



```
> SELECT current_user();
```

REFLECT

函数语法：



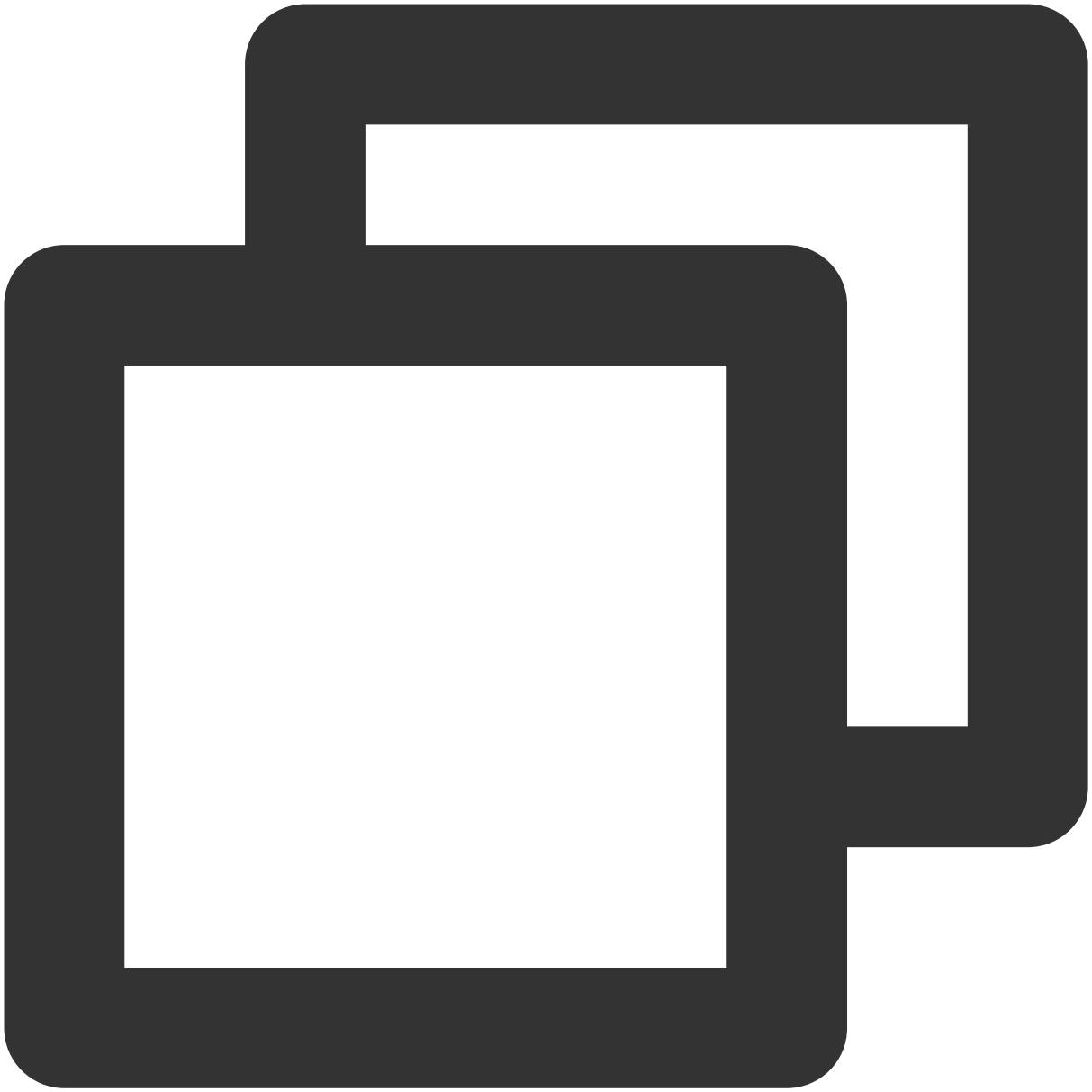
```
REFLECT(<class> string, <method> string[, <arg1> T1[, <arg2> T2, ...]])
```

支持引擎：SparkSQL、Presto

使用说明：调用具有反射的方法。

返回类型：string

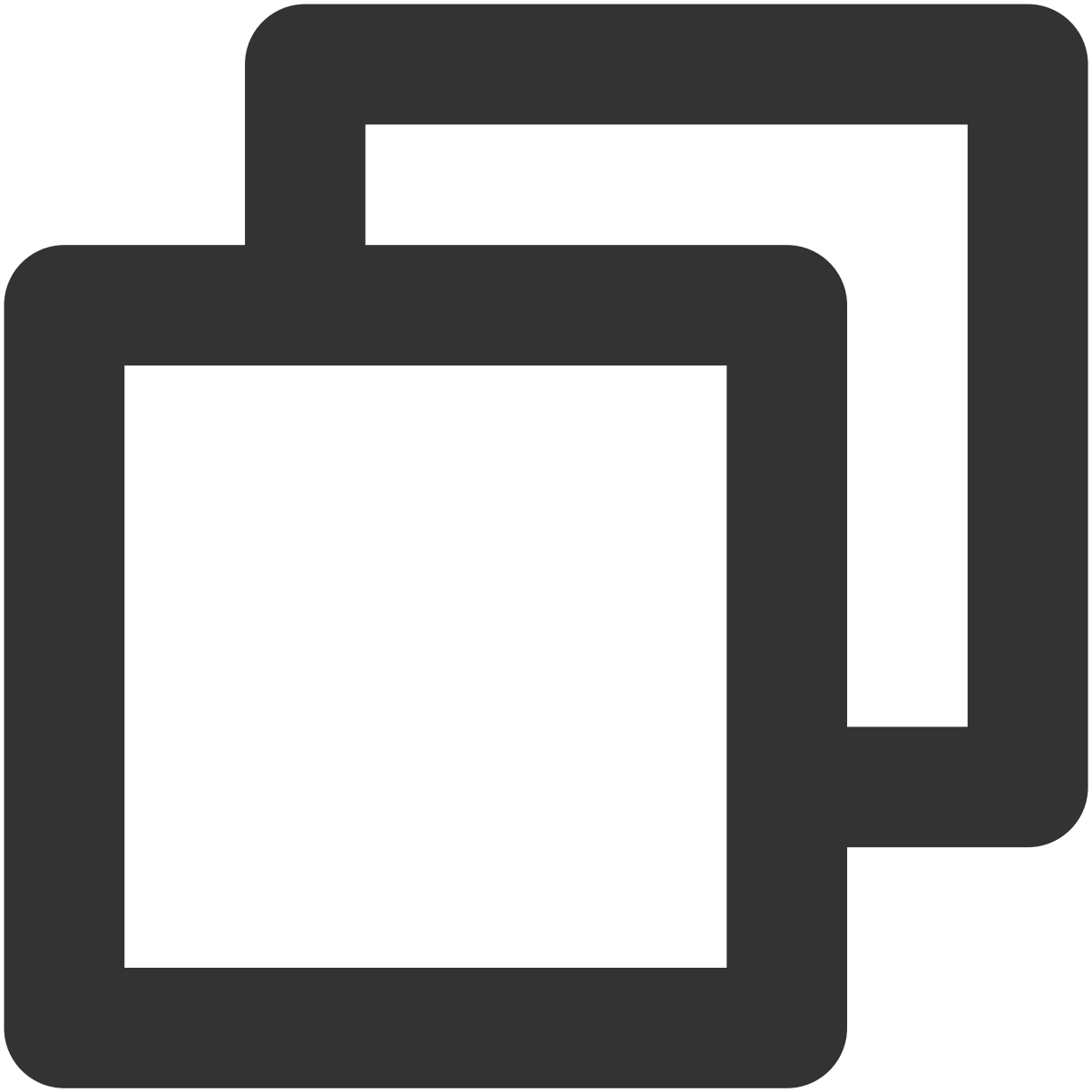
示例：



```
> select reflect('java.lang.Math', 'abs', -1);
 1
```

JAVA_METHOD

函数语法：



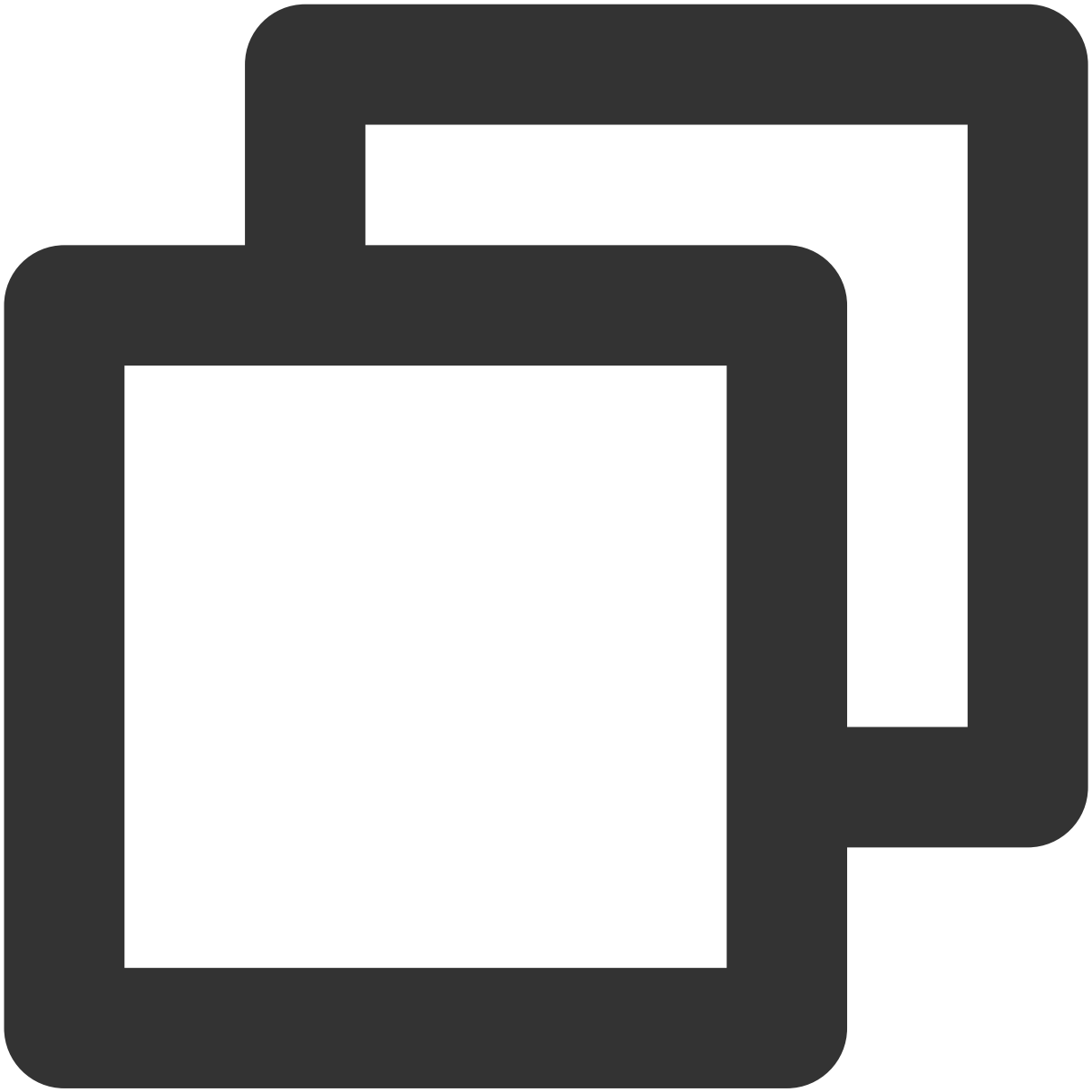
```
JAVA_METHOD(<class> string, <method> string[, <arg1> T1[, <arg2> T2, ...]])
```

支持引擎：SparkSQL、Presto

使用说明：调用具有反射的方法。

返回类型：string

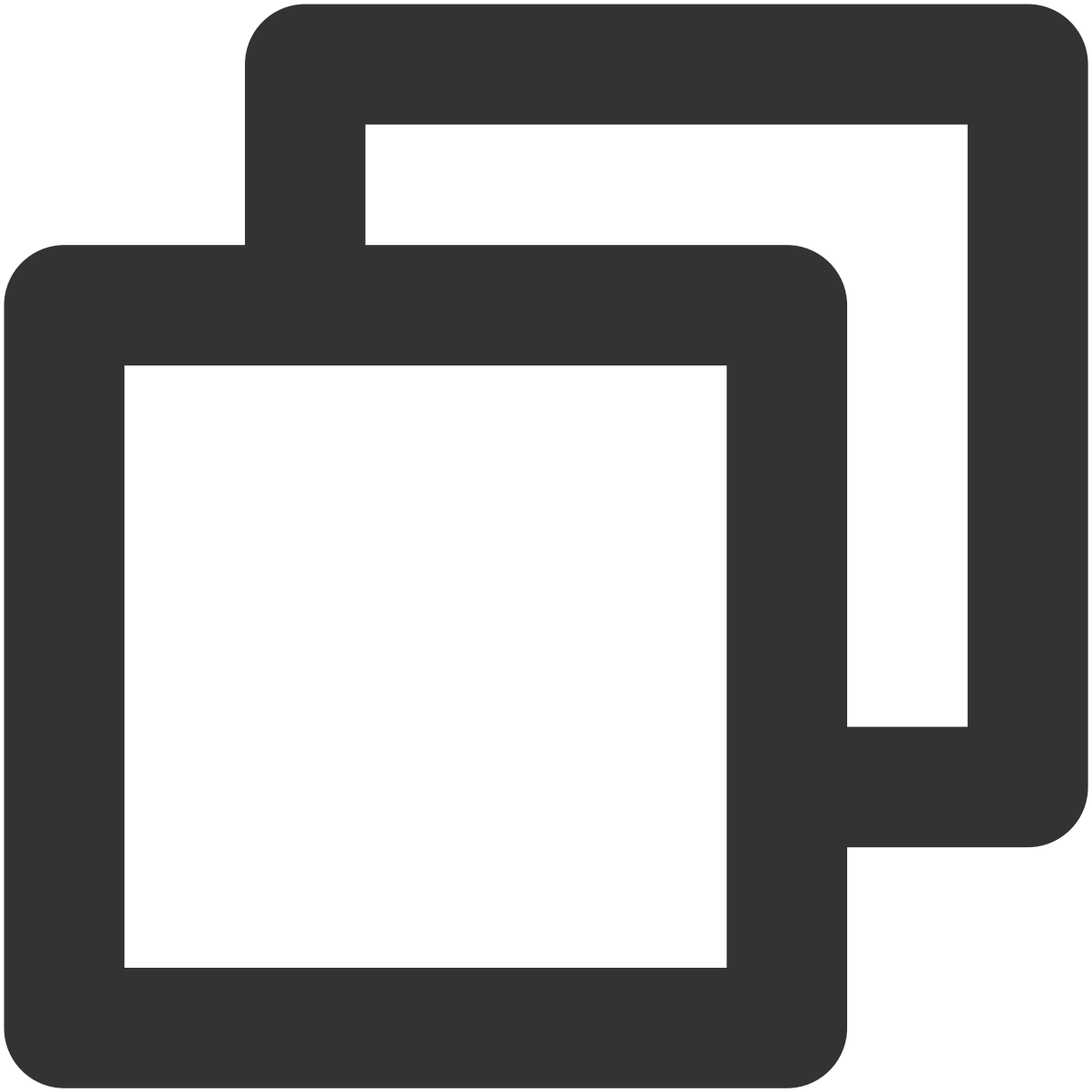
示例：



```
> select JAVA_METHOD('java.lang.Math', 'abs', -1);
 1
```

VERSION

函数语法：



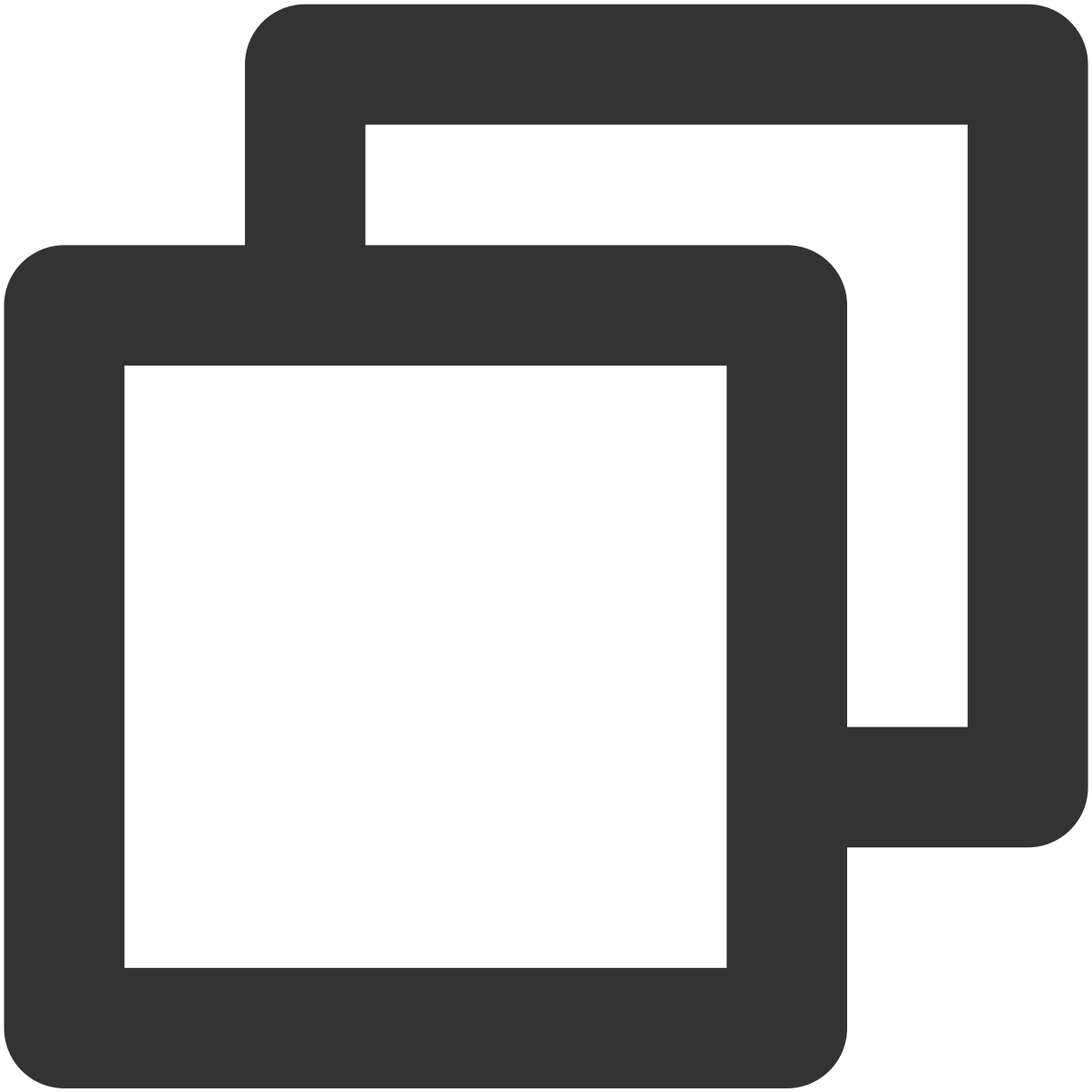
```
VERSION()
```

支持引擎：SparkSQL、Presto

使用说明：返回引擎版本。

返回类型：string

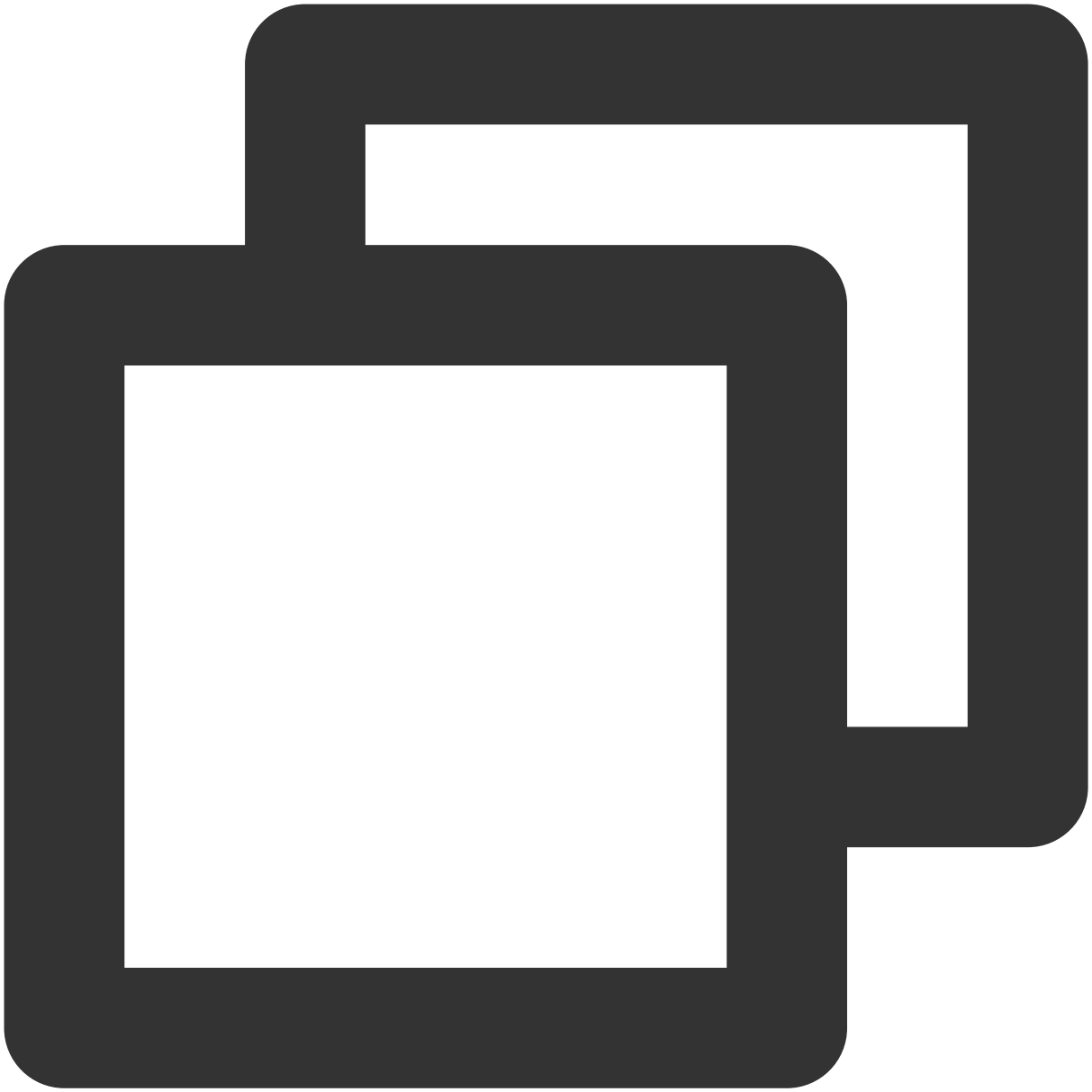
示例：



```
> select VERSION()
 3.0.0 rce61711a5fa54ab34fc74d86d521ecaeea6b072a
```

TYPEOF

函数语法：



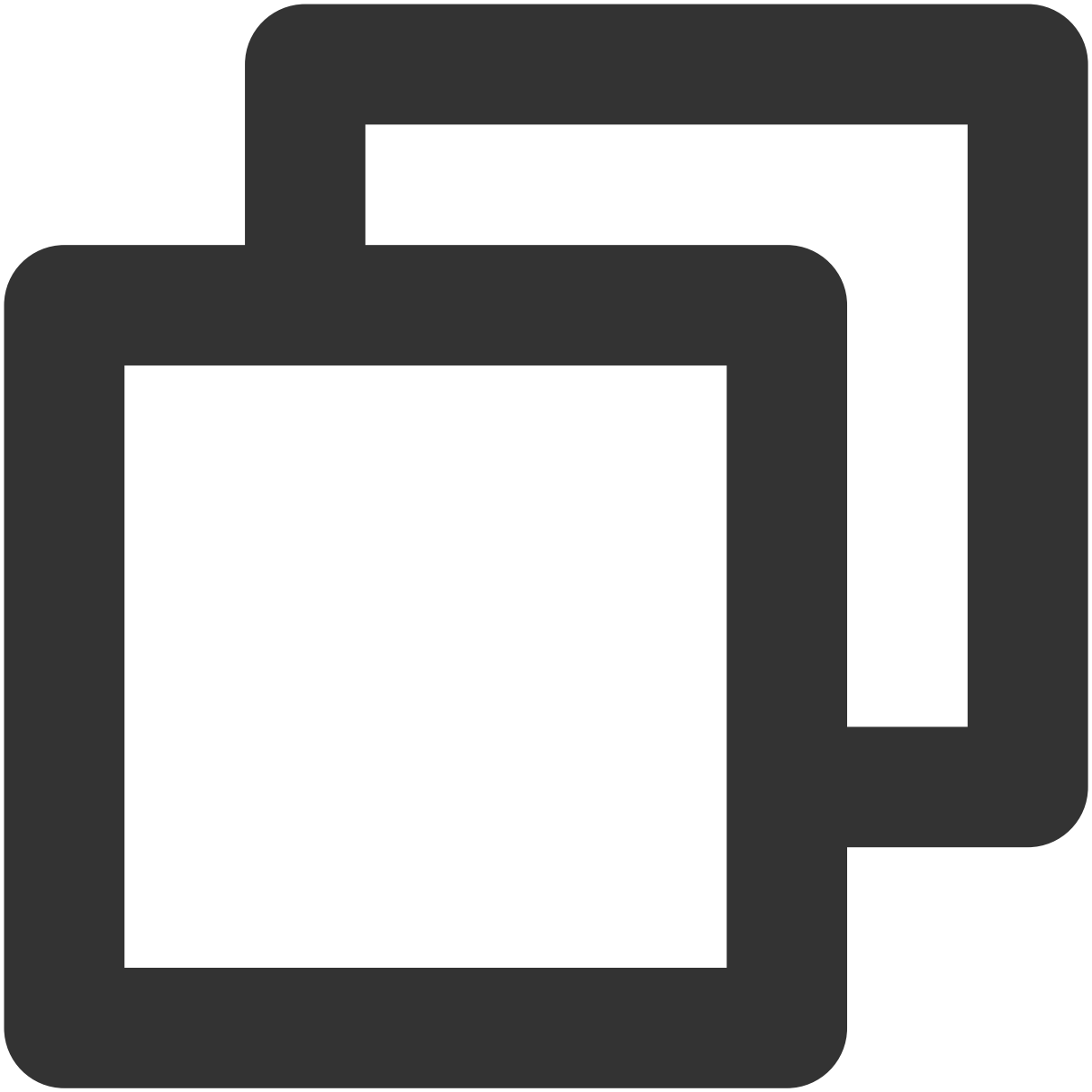
```
TYPEOF(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：返回expr的数据类型

返回类型：string

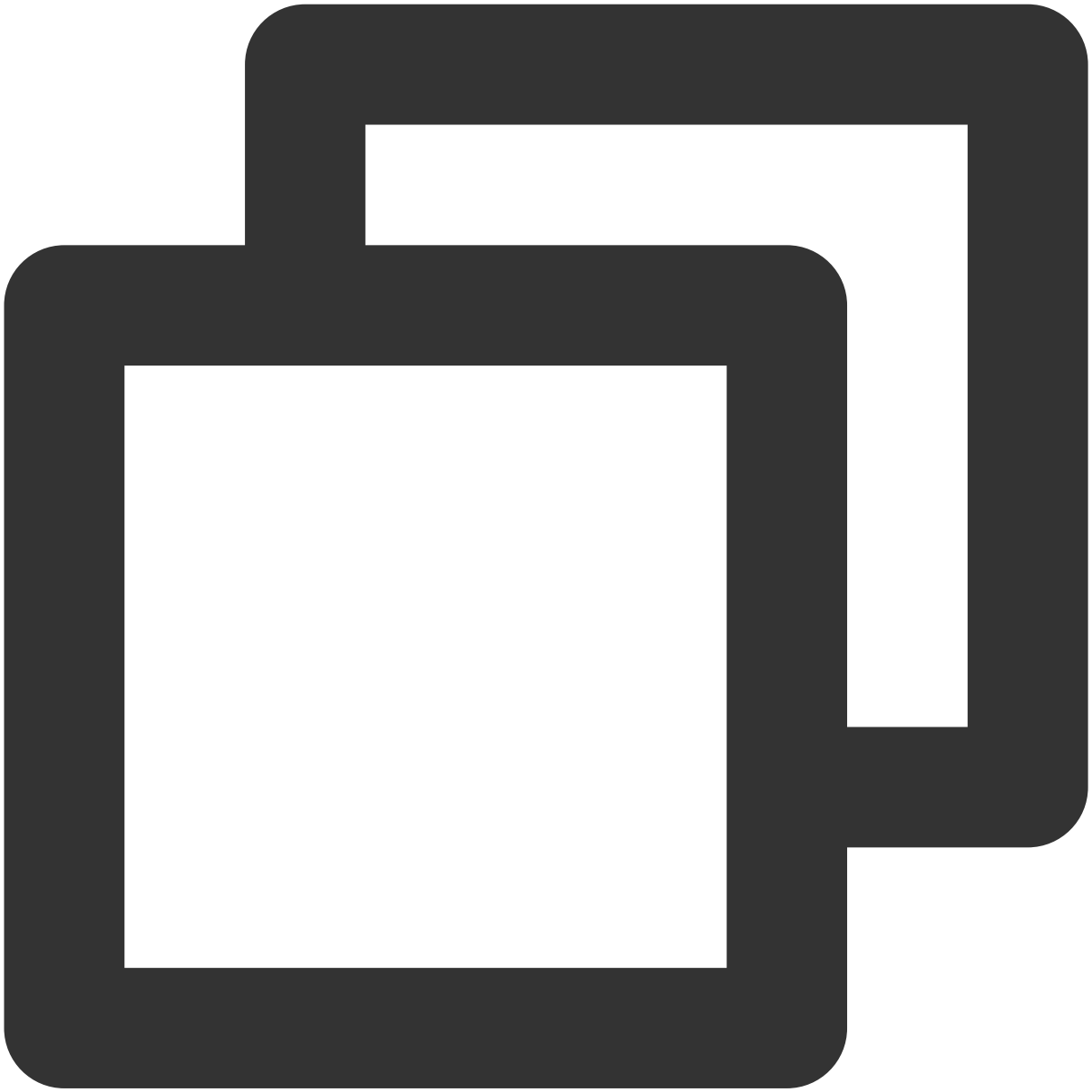
示例：



```
> SELECT typeof(1);
 int
> SELECT typeof(array(1));
 array<int>
```

CAST

函数语法：



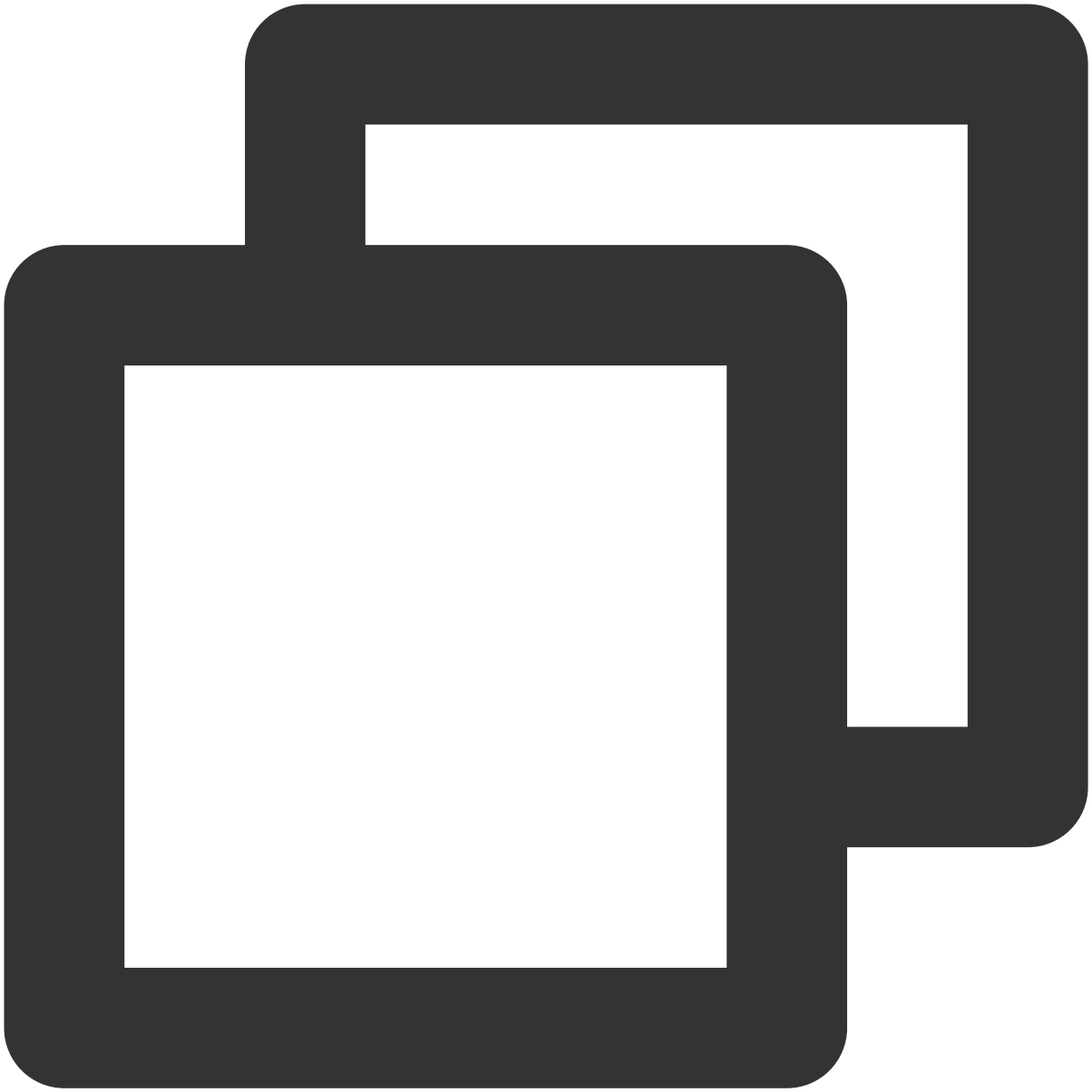
```
CAST(<expr> AS <type>)
```

支持引擎：SparkSQL、Presto

使用说明：将expr转换为type类型

返回类型：<type>

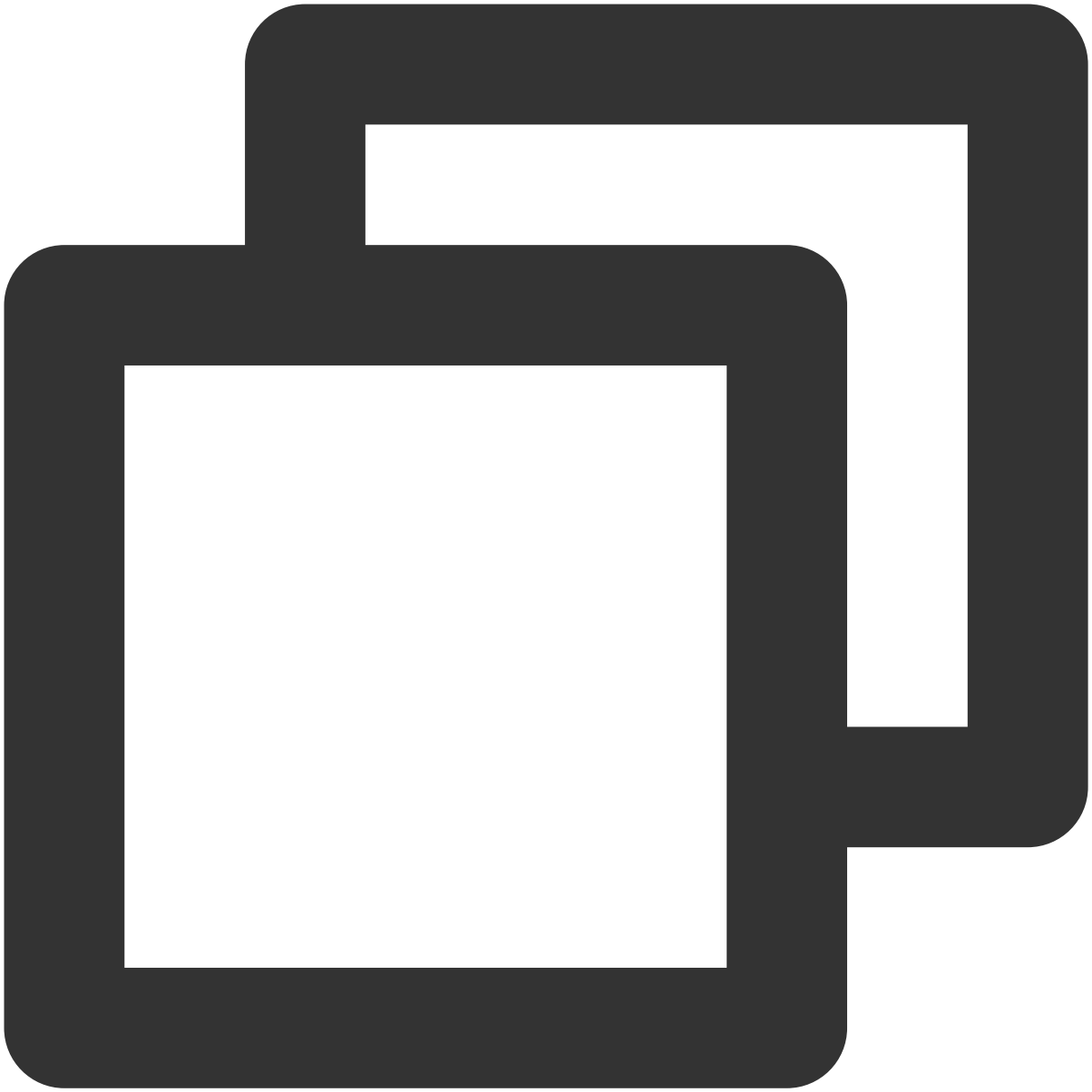
示例：



```
> SELECT cast('10' as int);
 10
```

BOOLEAN

函数语法：



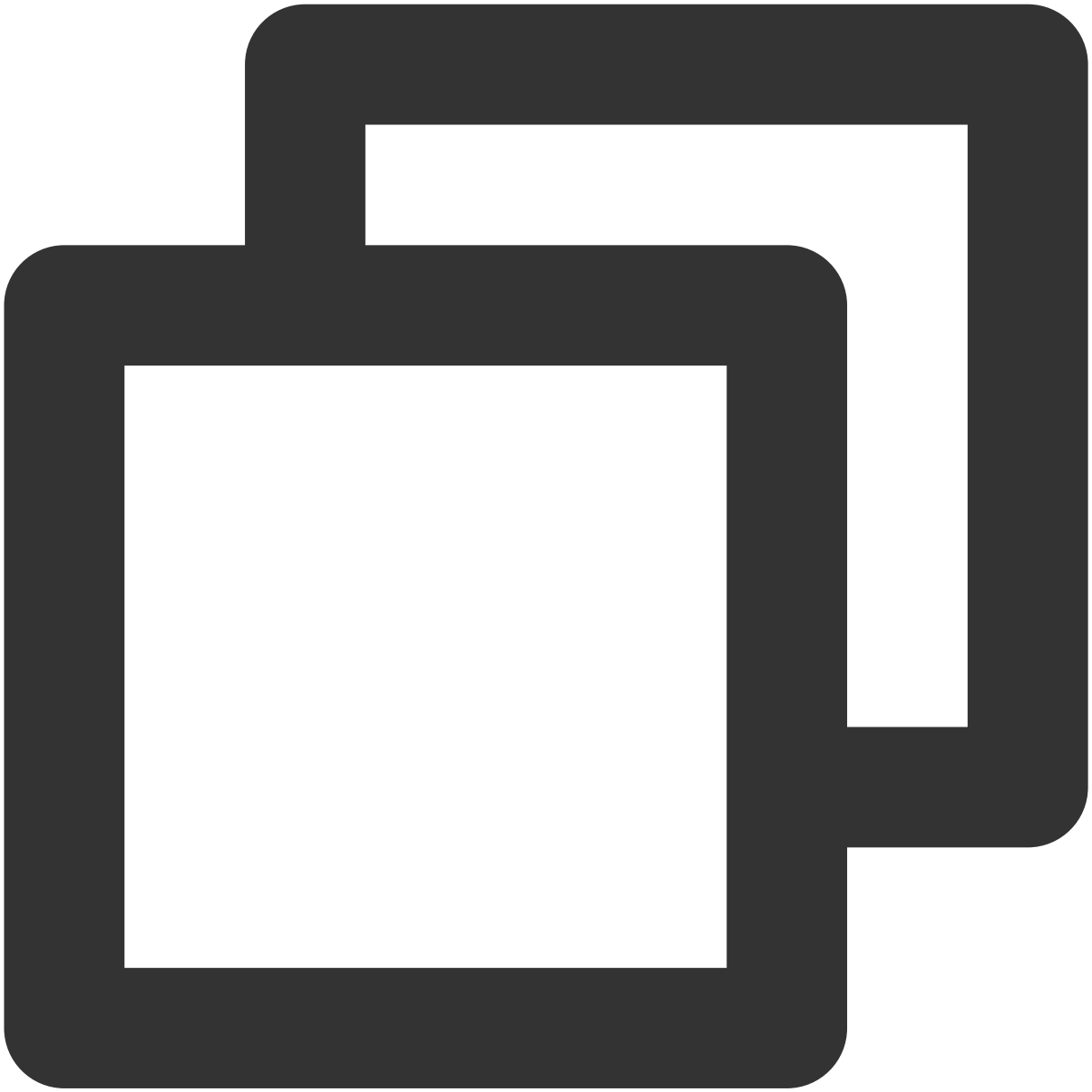
```
BOOLEAN(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：将expr转换为boolean类型

返回类型：boolean

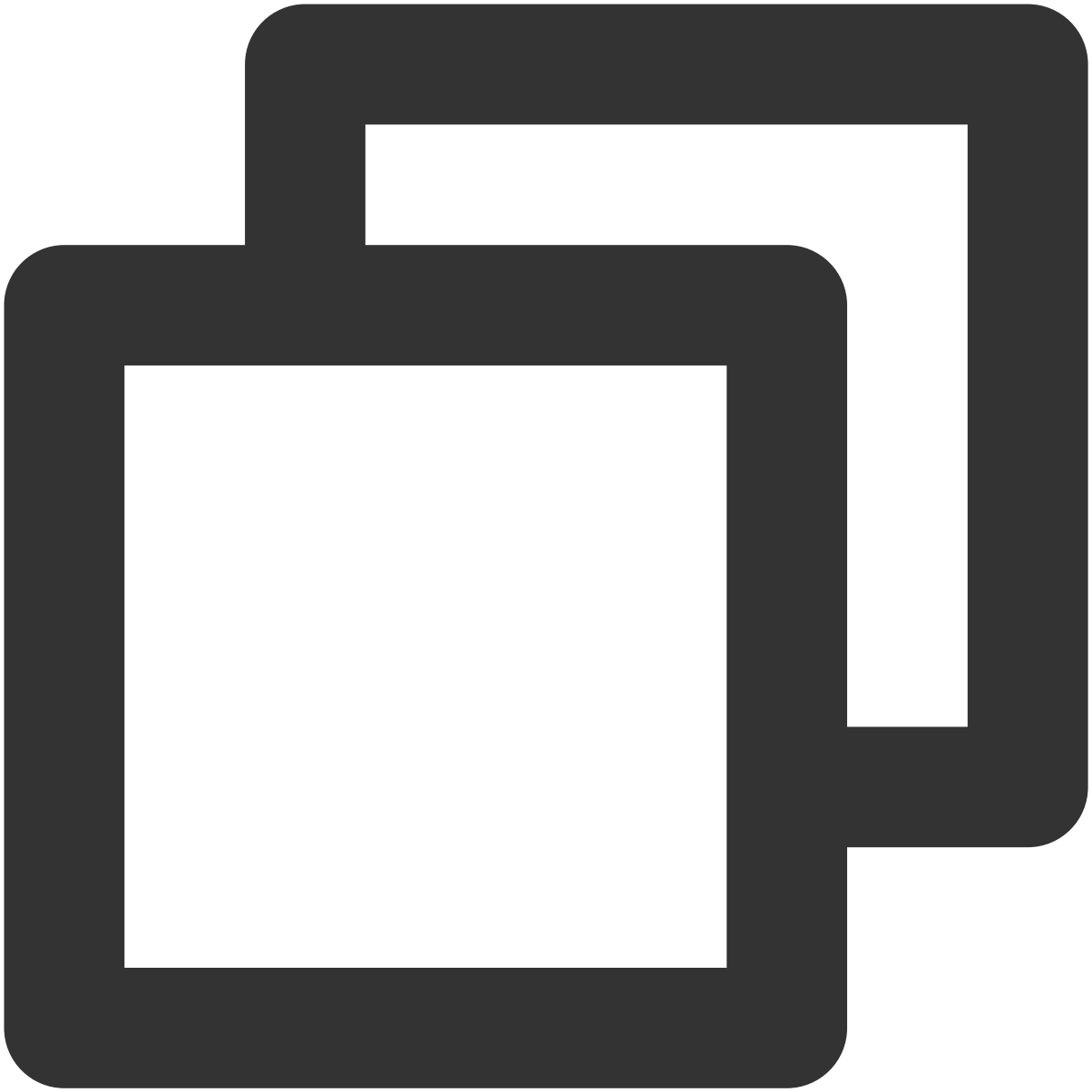
示例：



```
> SELECT boolean(1);
 true
```

BIGINT

函数语法：



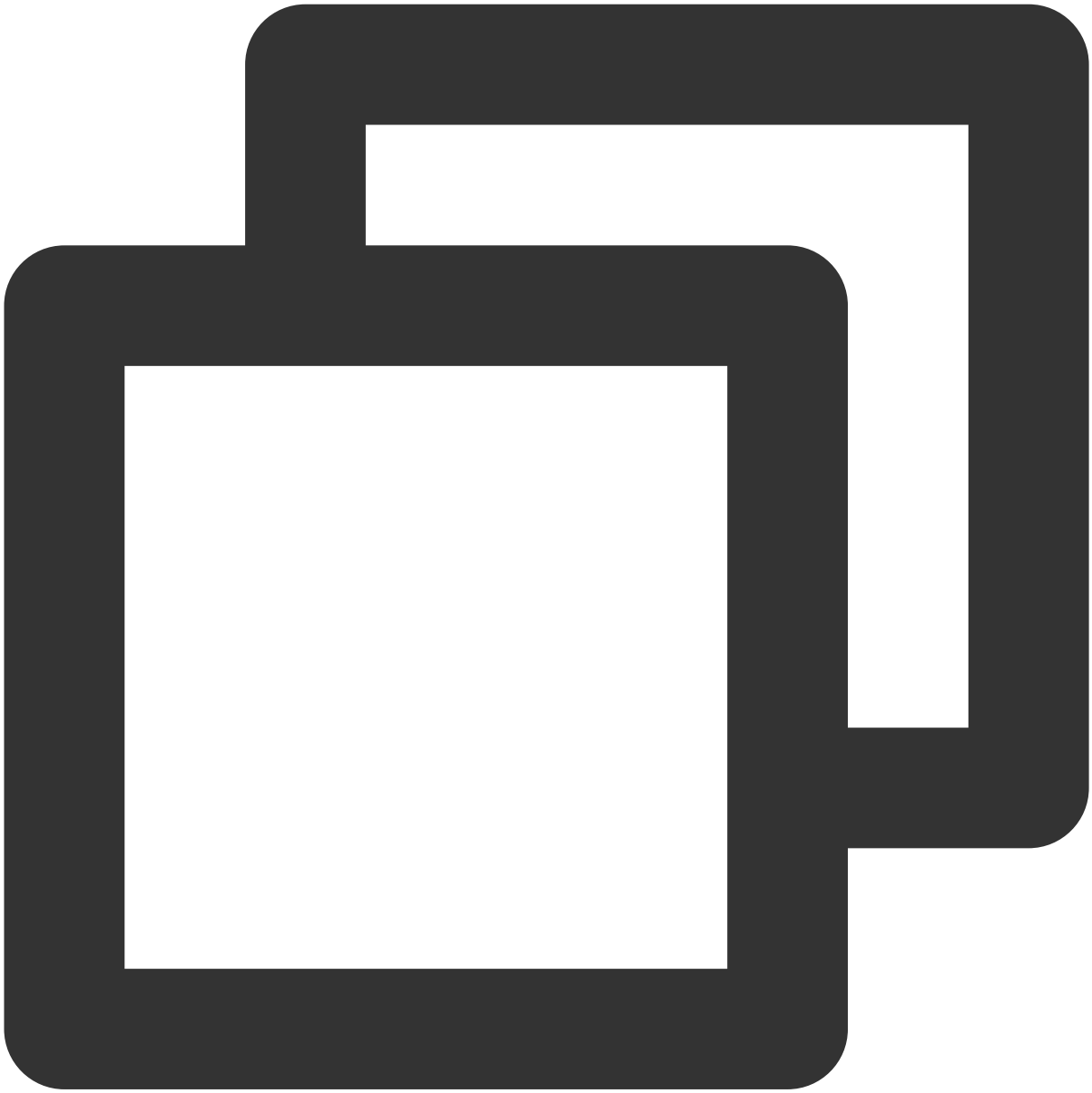
```
BIGINT(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：强制类型转换为bigint

返回类型：bigint

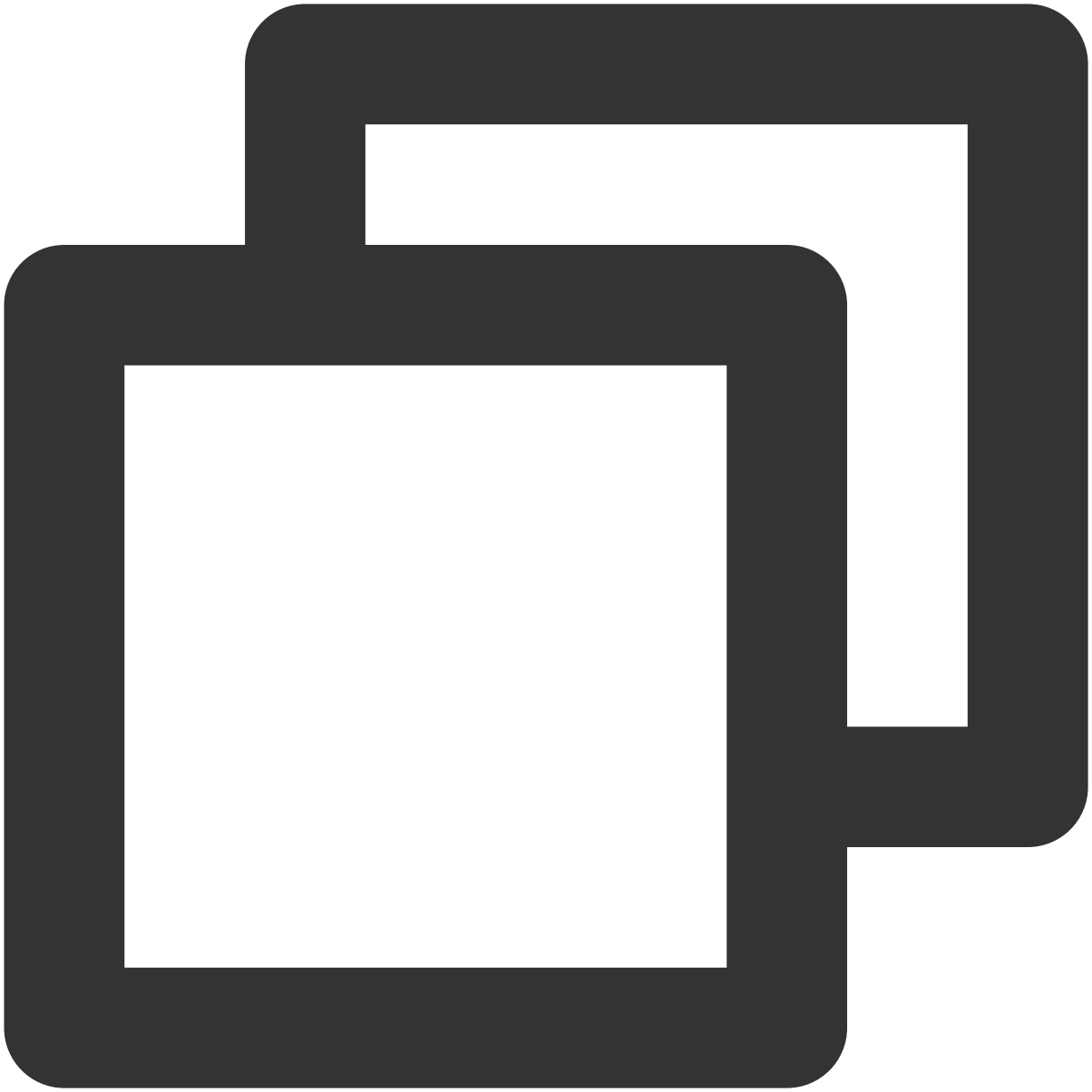
示例：



```
> select bigint(0);
 0
```

BINARY

函数语法：



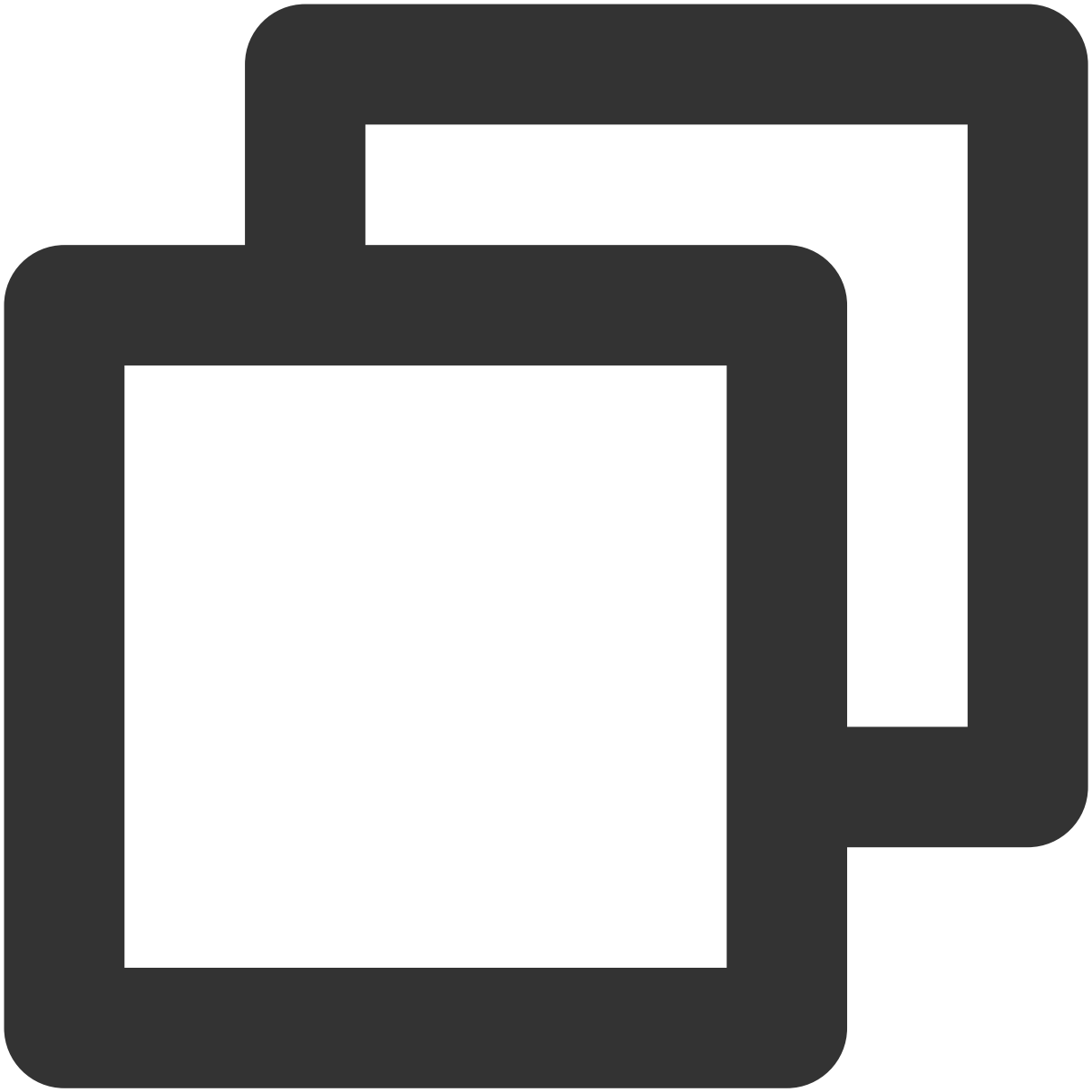
```
BINARY(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：强制类型转换为BINARY

返回类型：binary

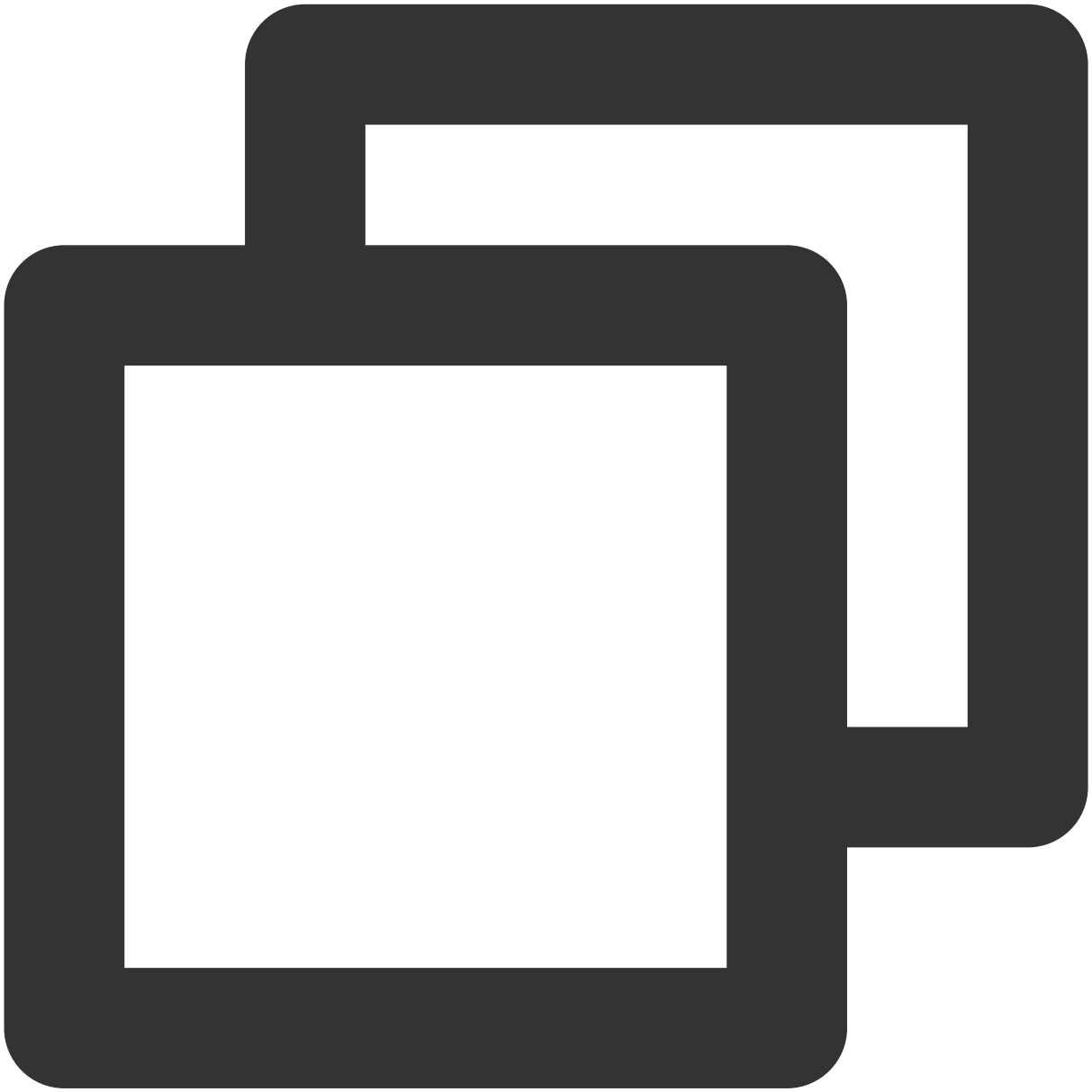
示例：



```
> select binary(65);
 A
```

DOUBLE

函数语法：



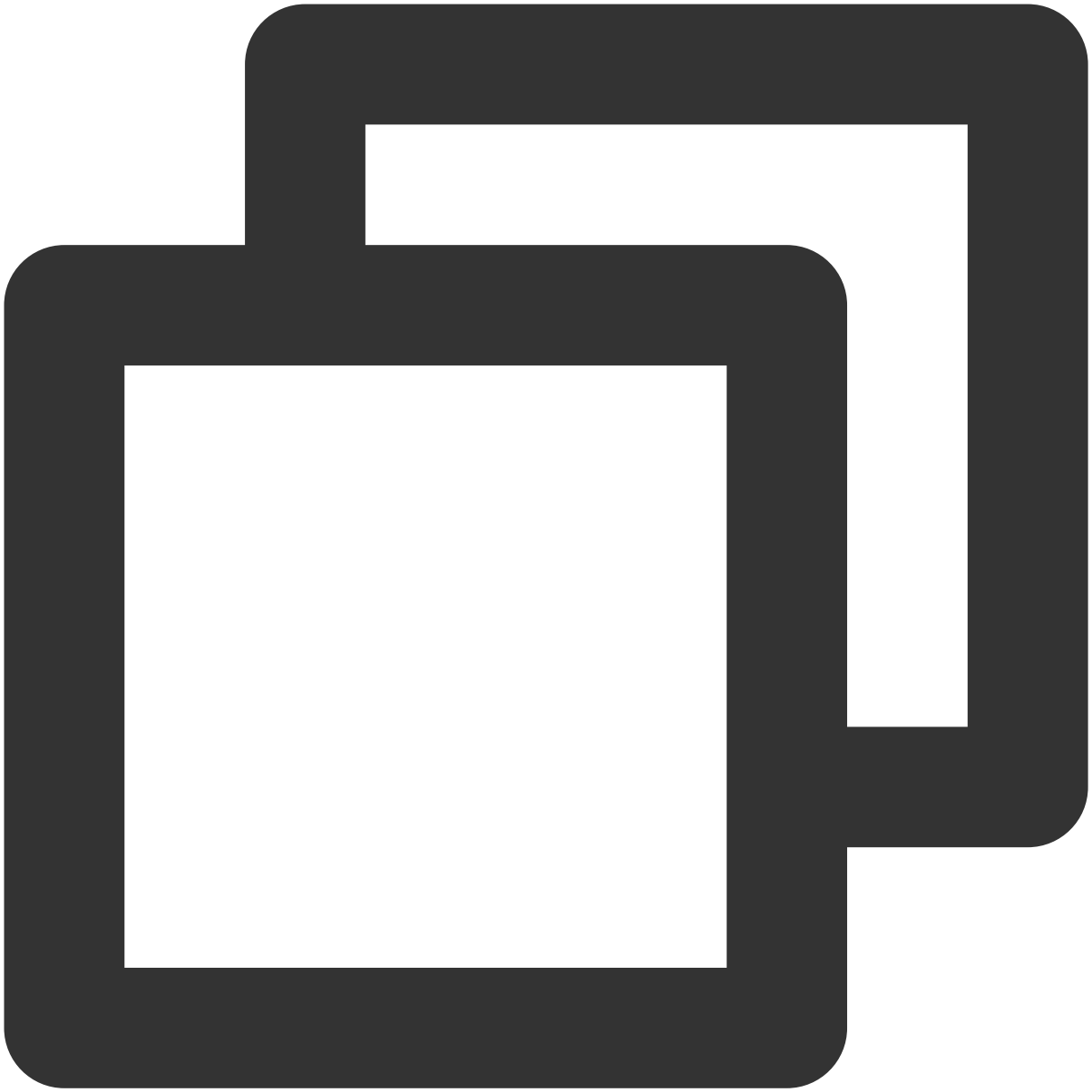
```
DOUBLE(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：强制类型转换为double

返回类型：double

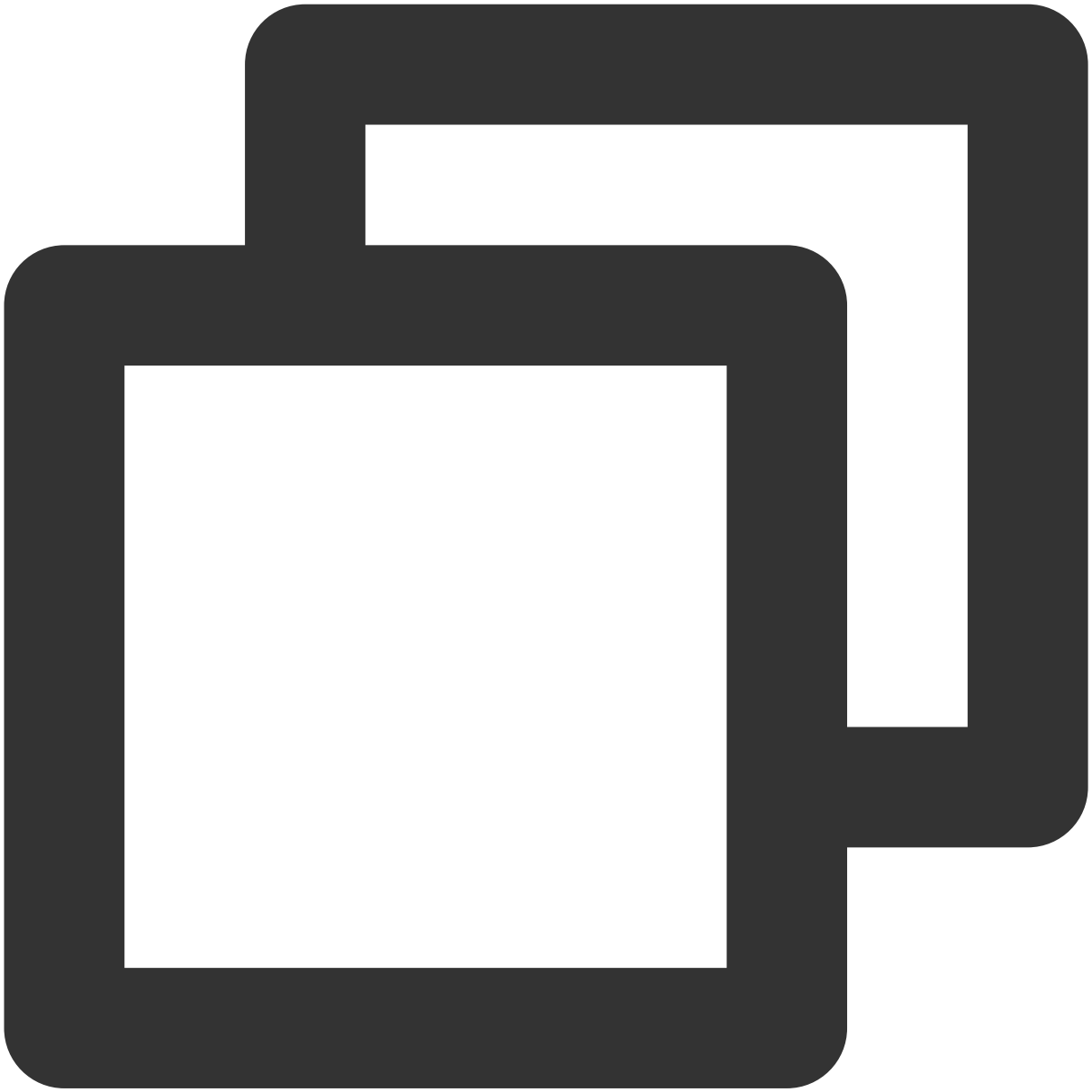
示例：



```
select double(1);
 1.0
```

FLOAT

函数语法：



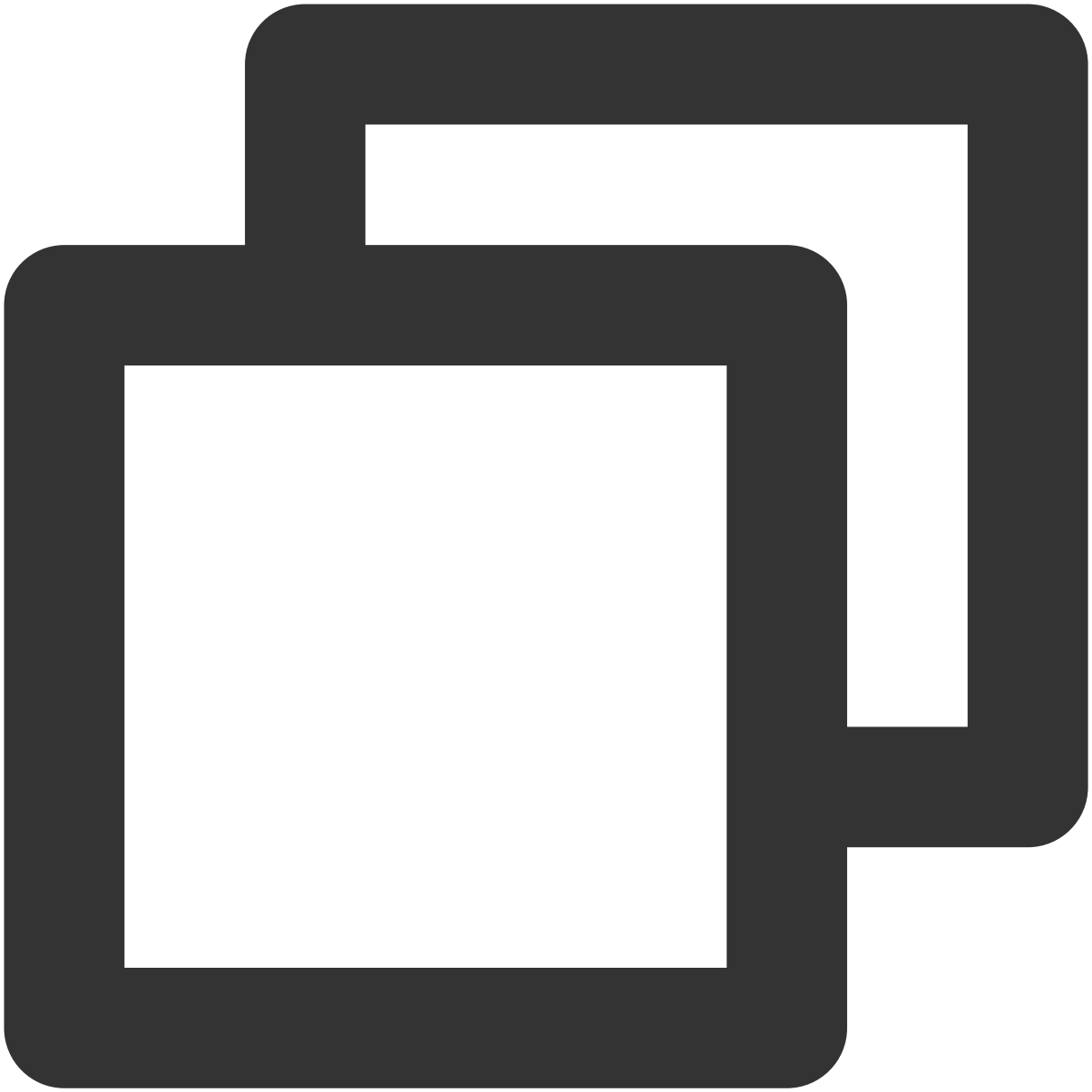
```
FLOAT(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：强制类型转换为float

返回类型：float

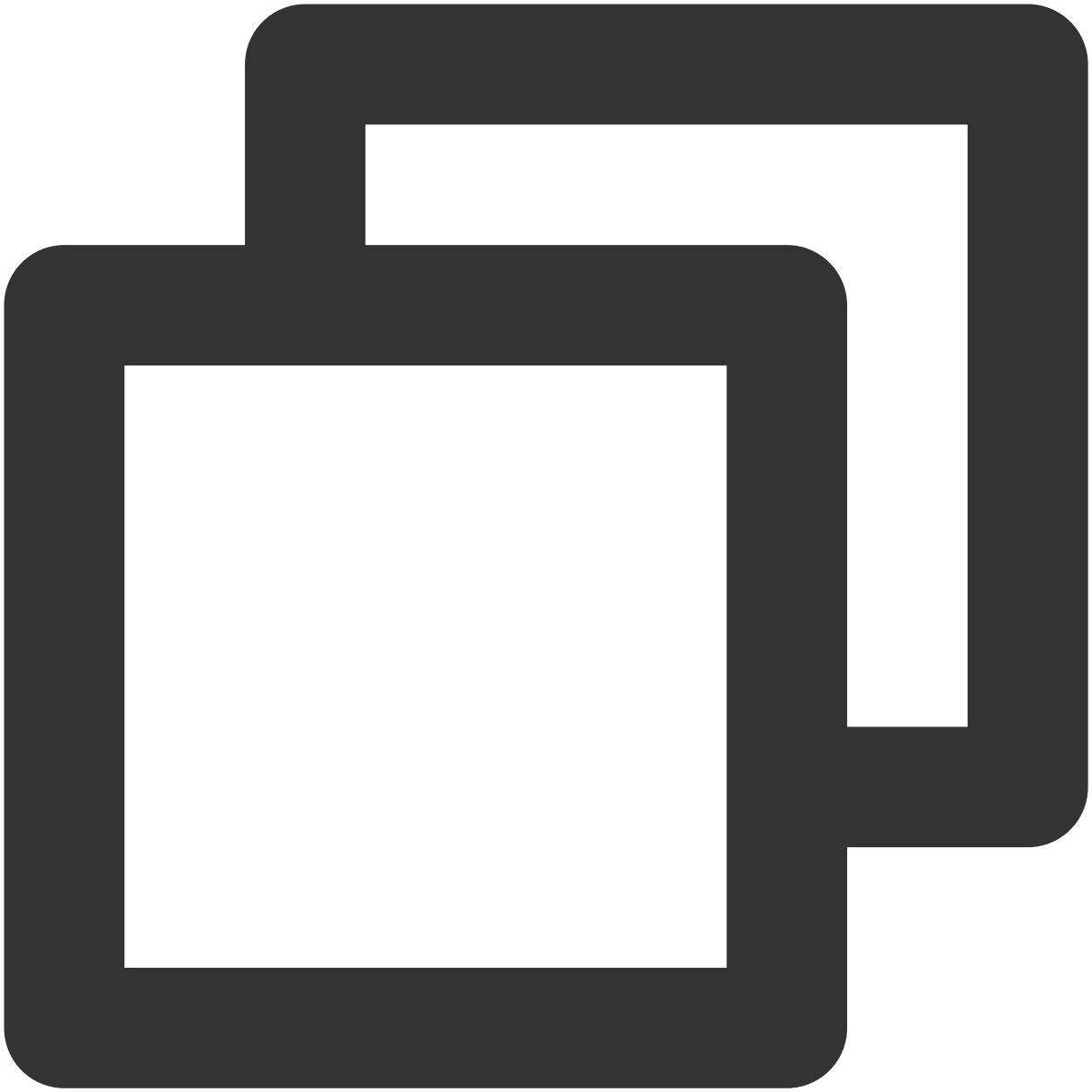
示例：



```
> select float(1);
 1.0
```

INT

函数语法：



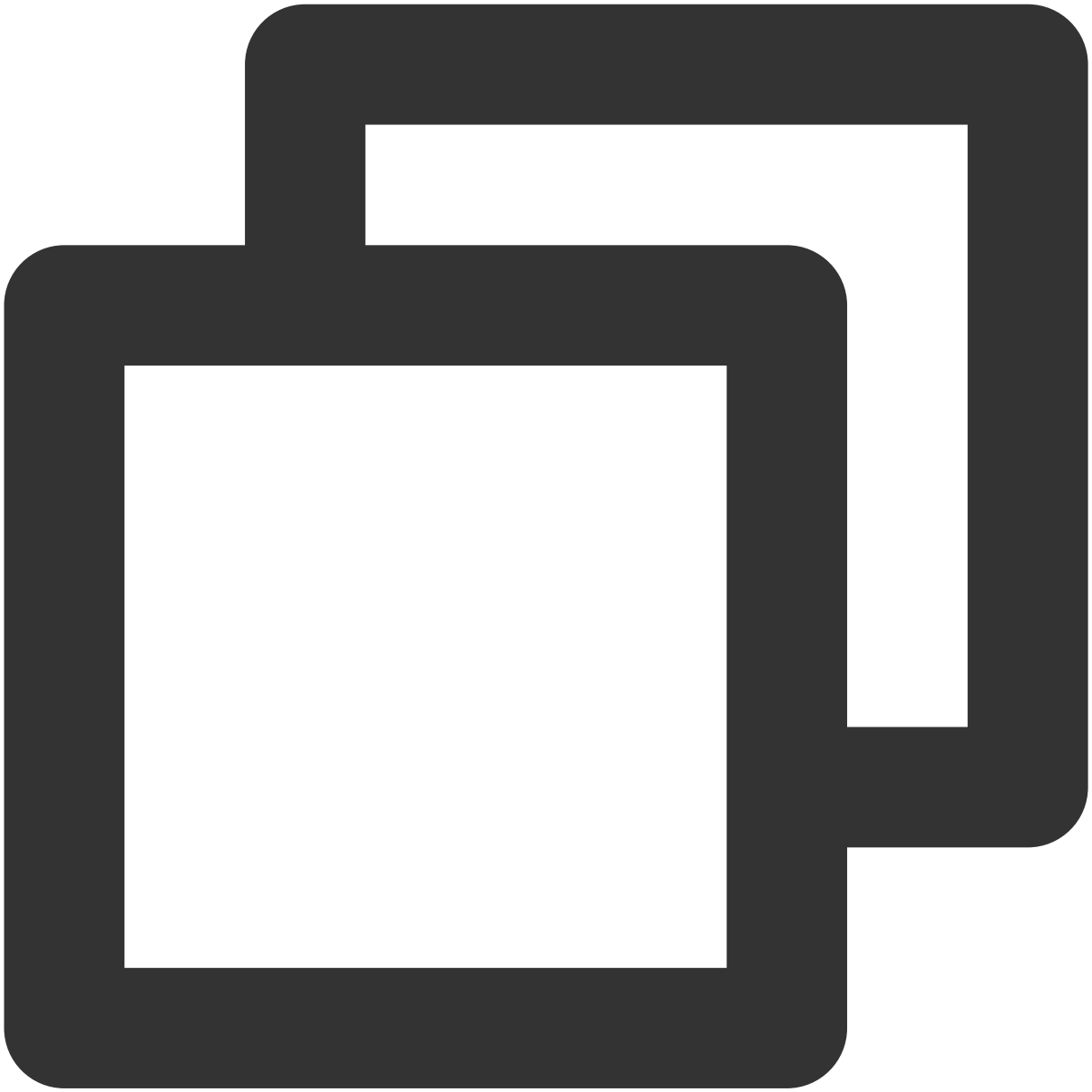
```
INT(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：强制类型转换为integer

返回类型：integer

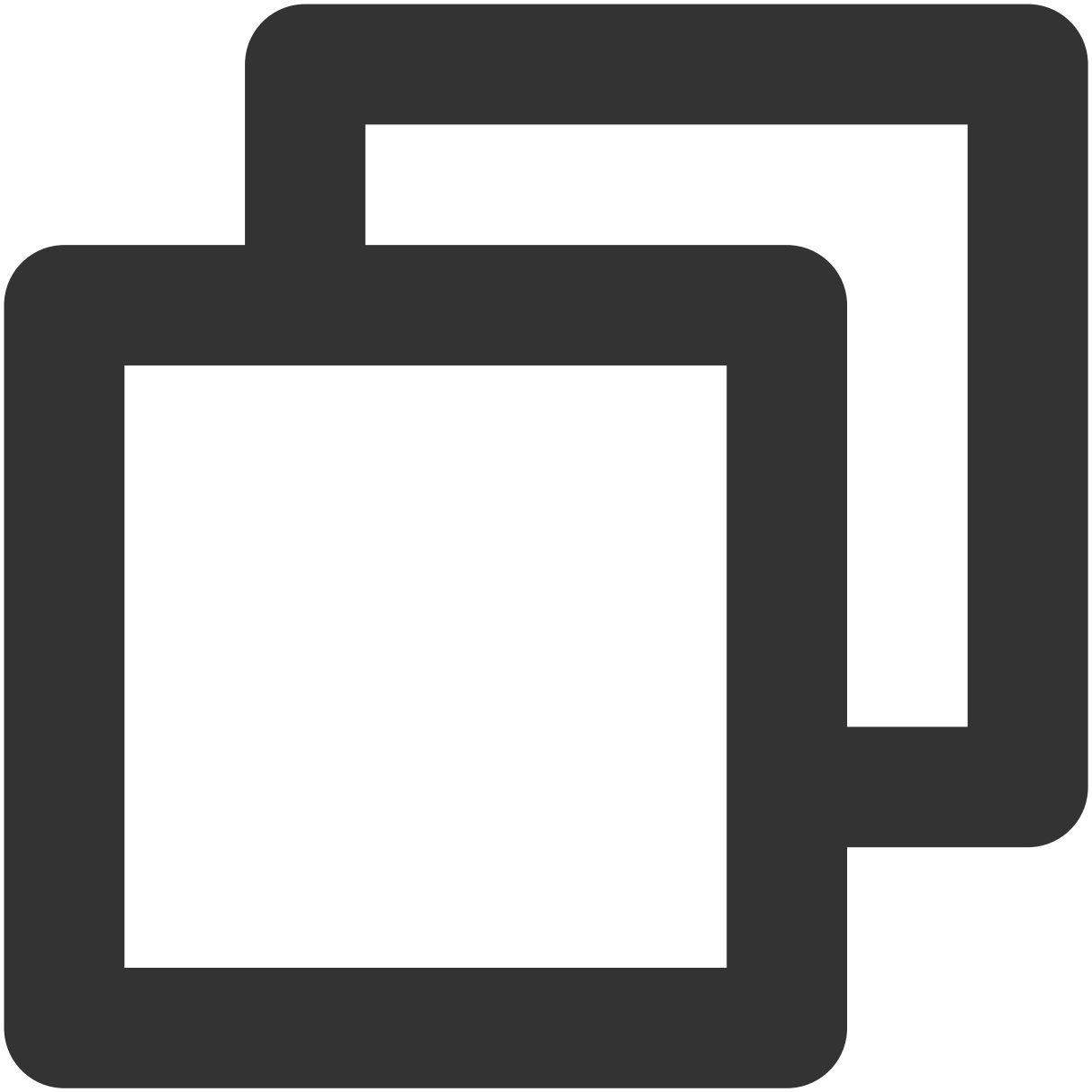
示例：



```
> select int(1.0);
 1
```

SMALLINT

函数语法：



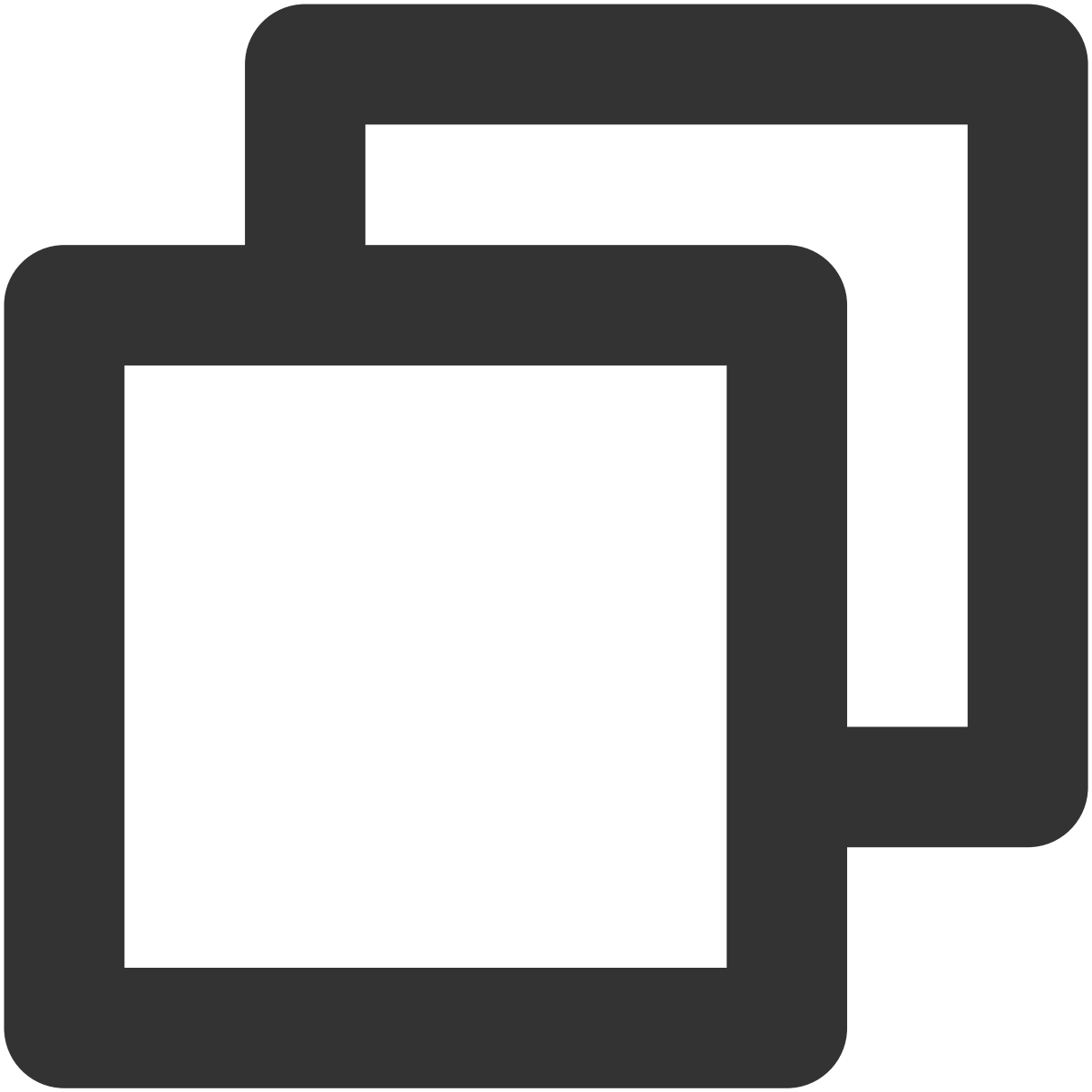
```
SMALLINT(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：强制转换为smallint类型

返回类型：smallint

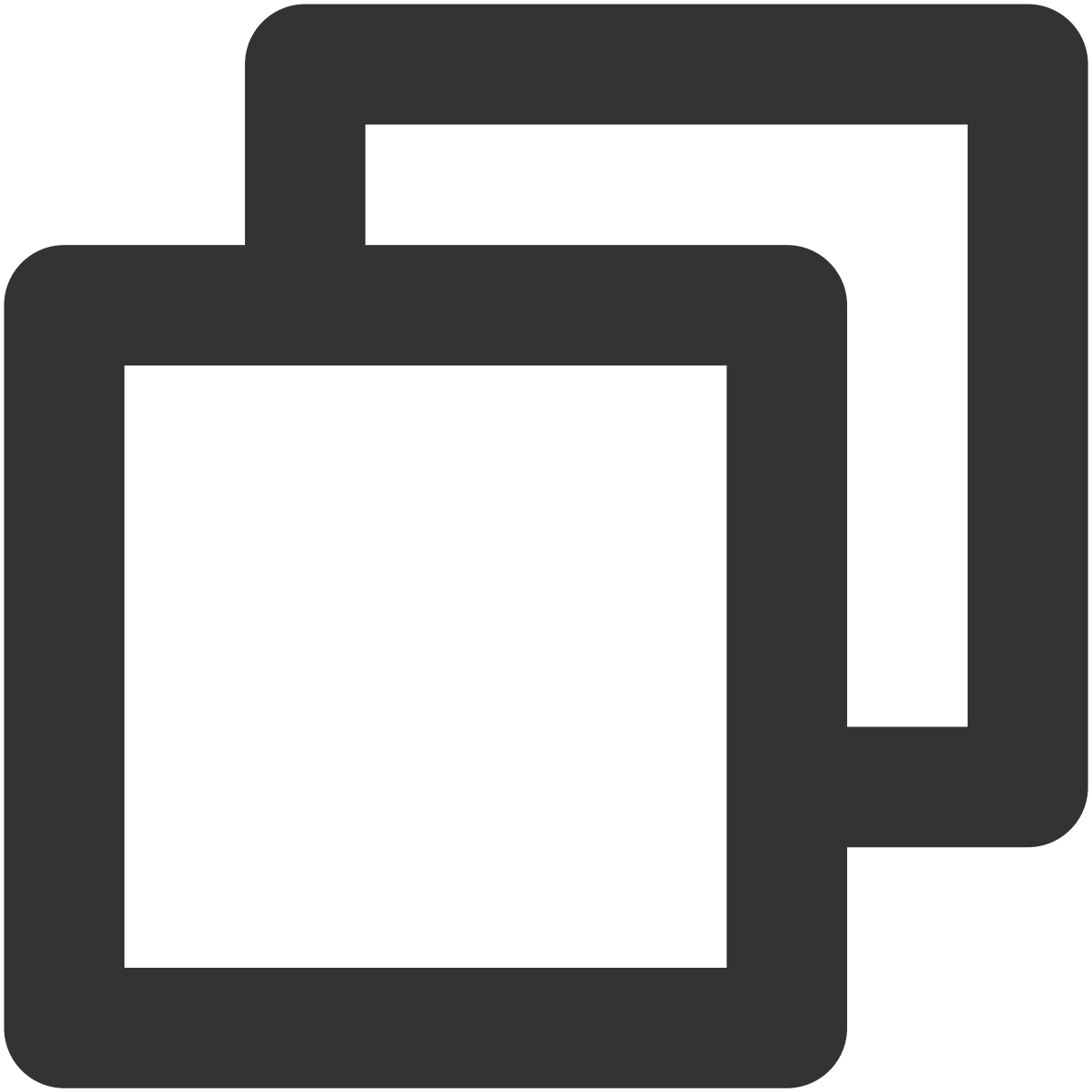
示例：



```
select typeof(smallint(1));
 smallint
```

STRING

函数语法：



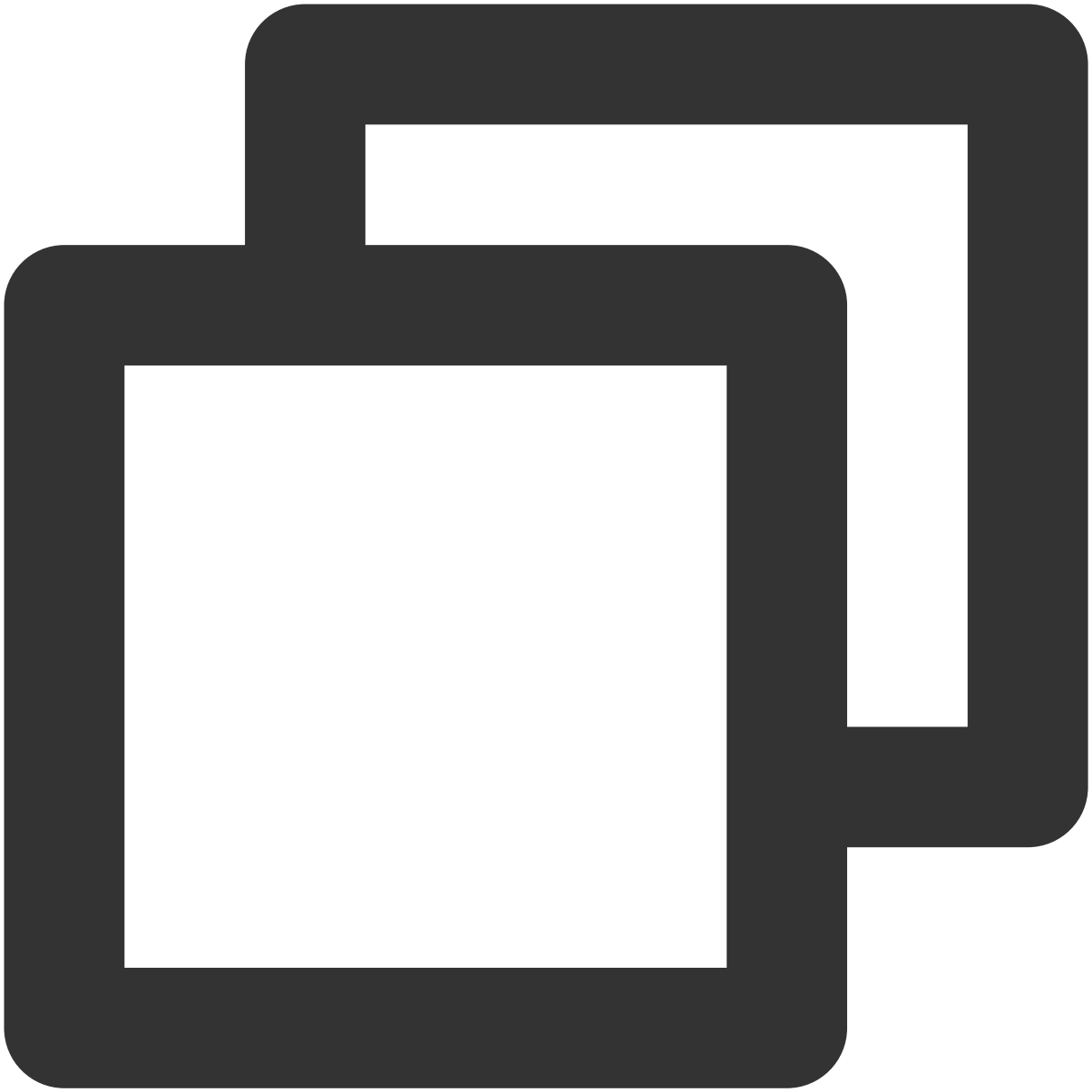
```
STRING(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：强制类型转换为string

返回类型：string

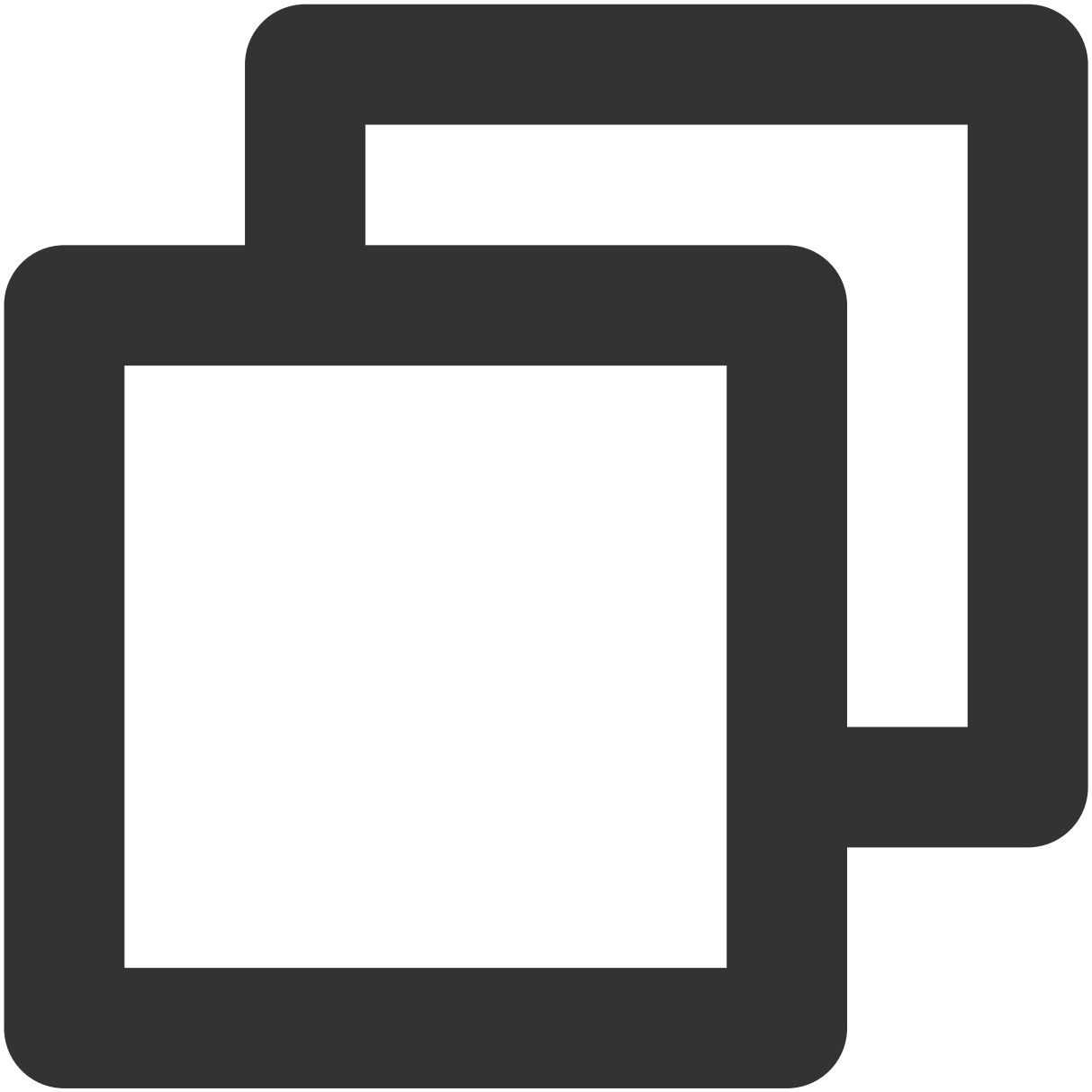
示例：



```
> select typeof(string(1));
 string
```

TINYINT

函数语法：



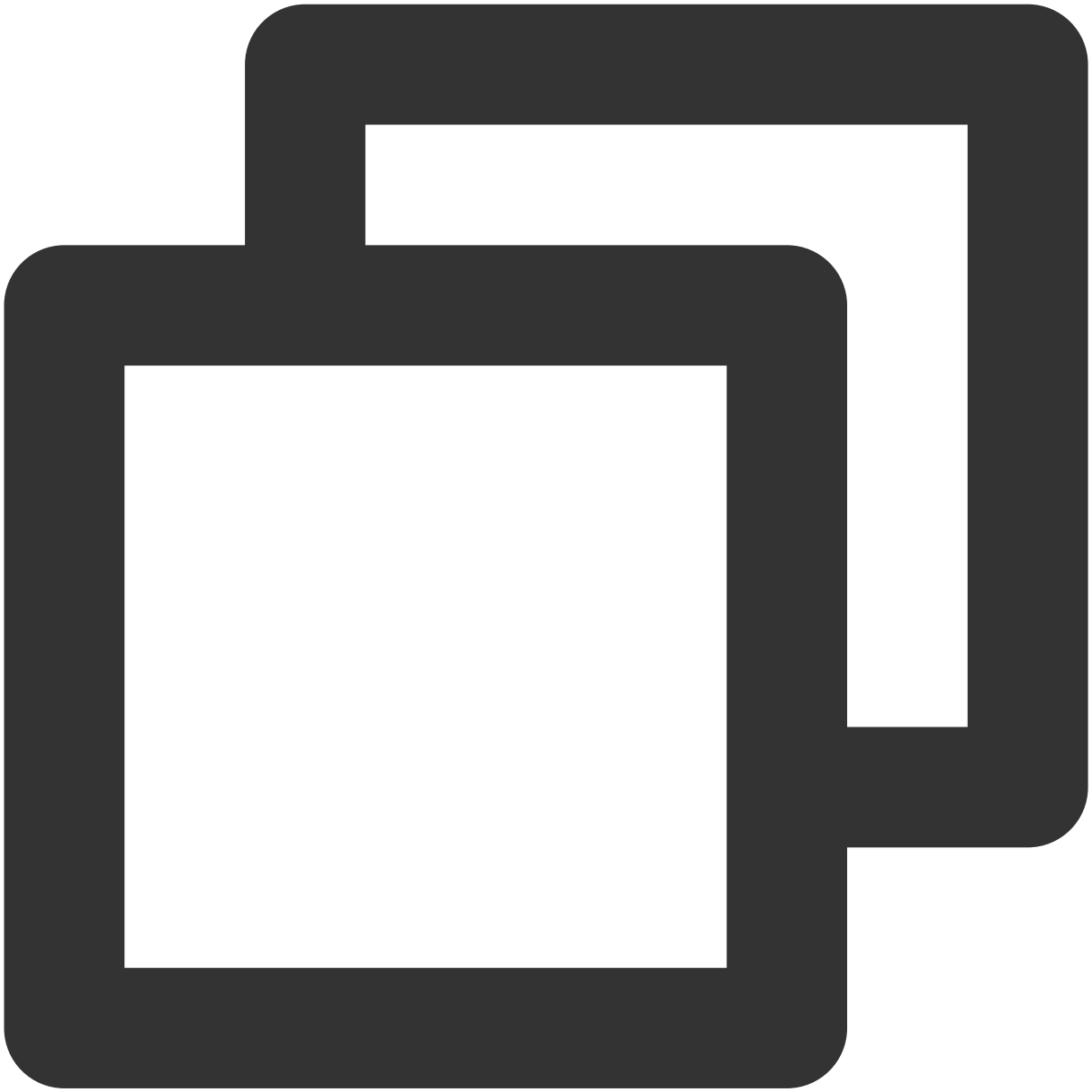
```
TINYINT(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：强制类型转换为tinyint

返回类型：tinyint

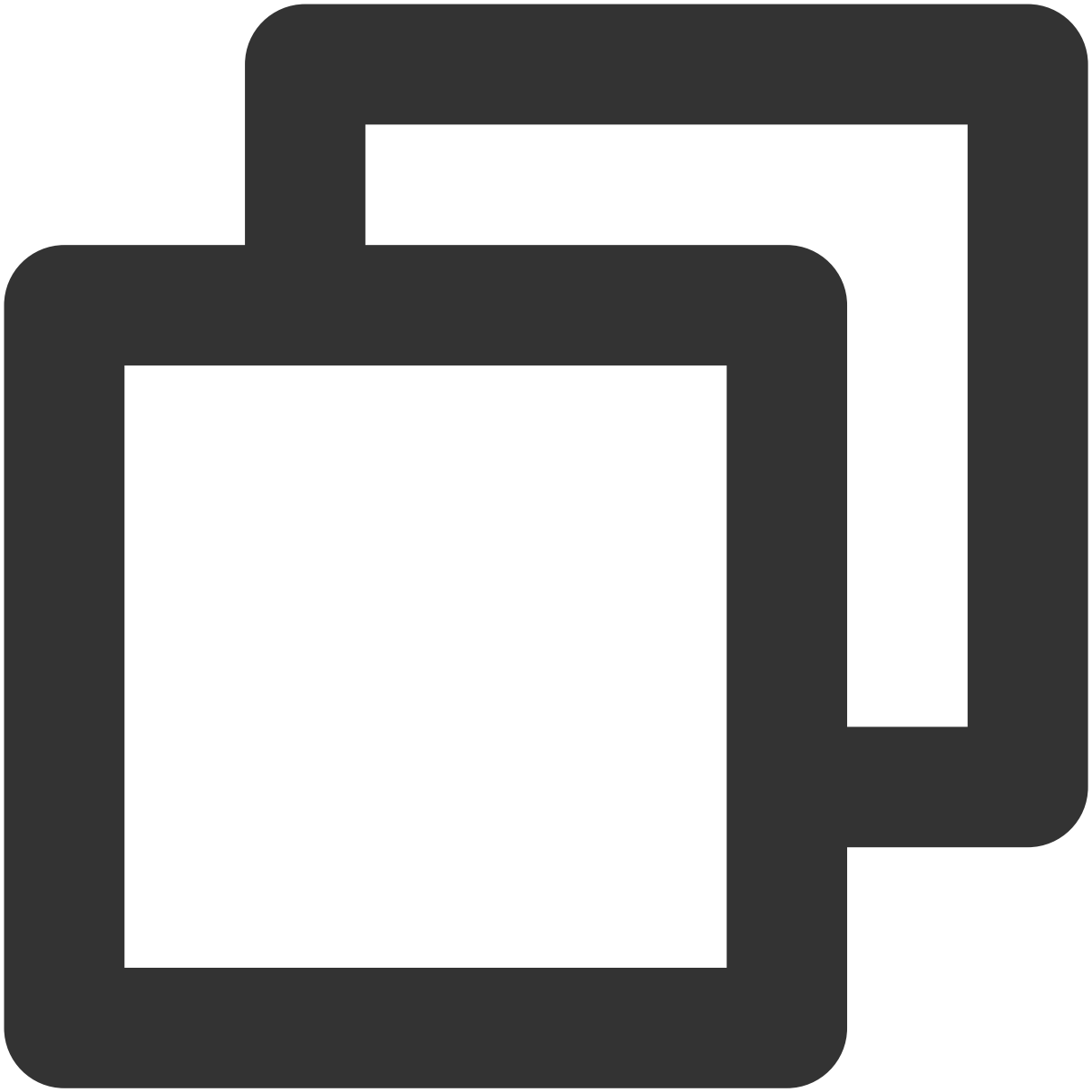
示例：



```
> select typeof(tinyint(1));
 tinyint
```

DECIMAL

函数语法：



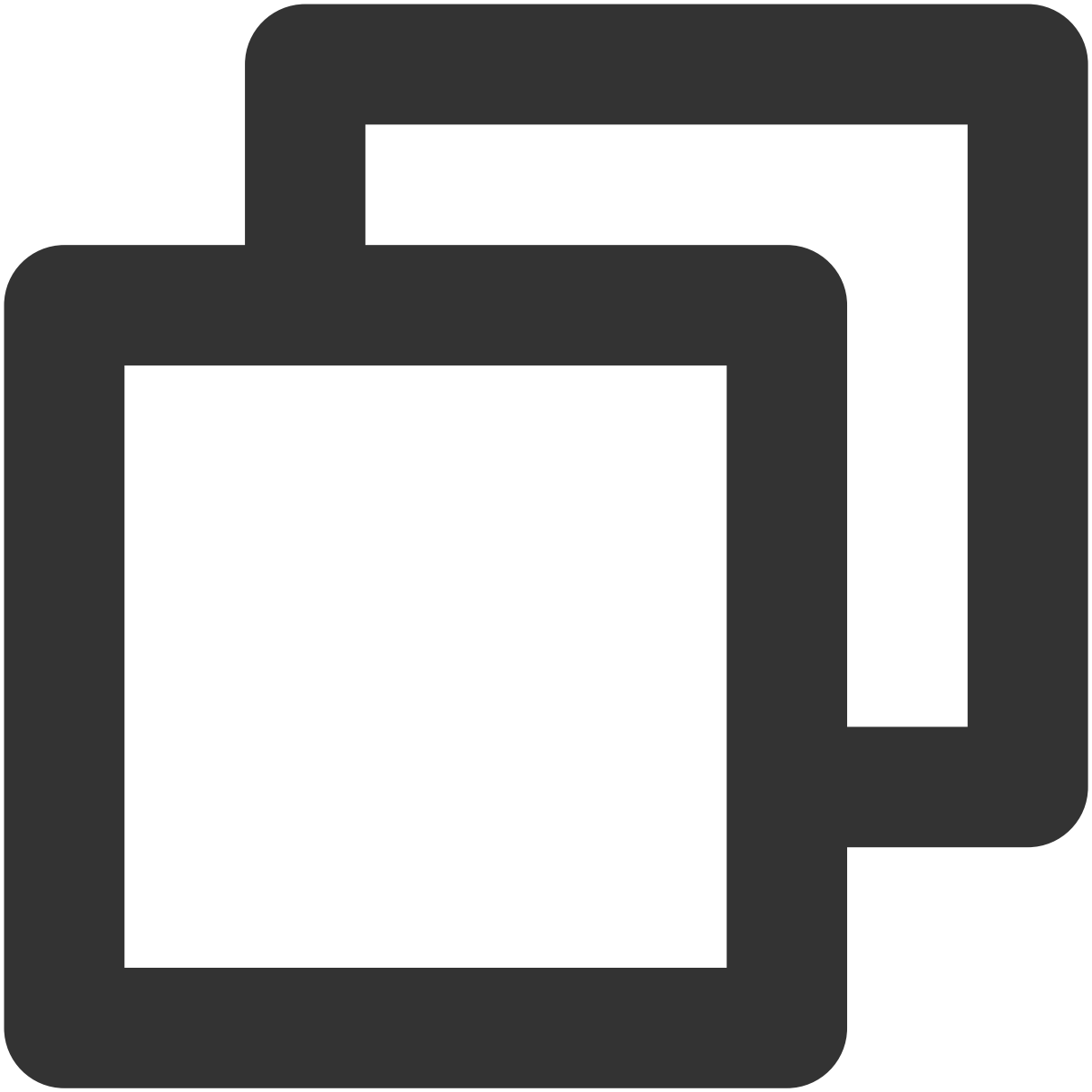
```
DECIMAL(<expr> T)
```

支持引擎：SparkSQL、Presto

使用说明：强制类型转换为decimal

返回类型：decimal

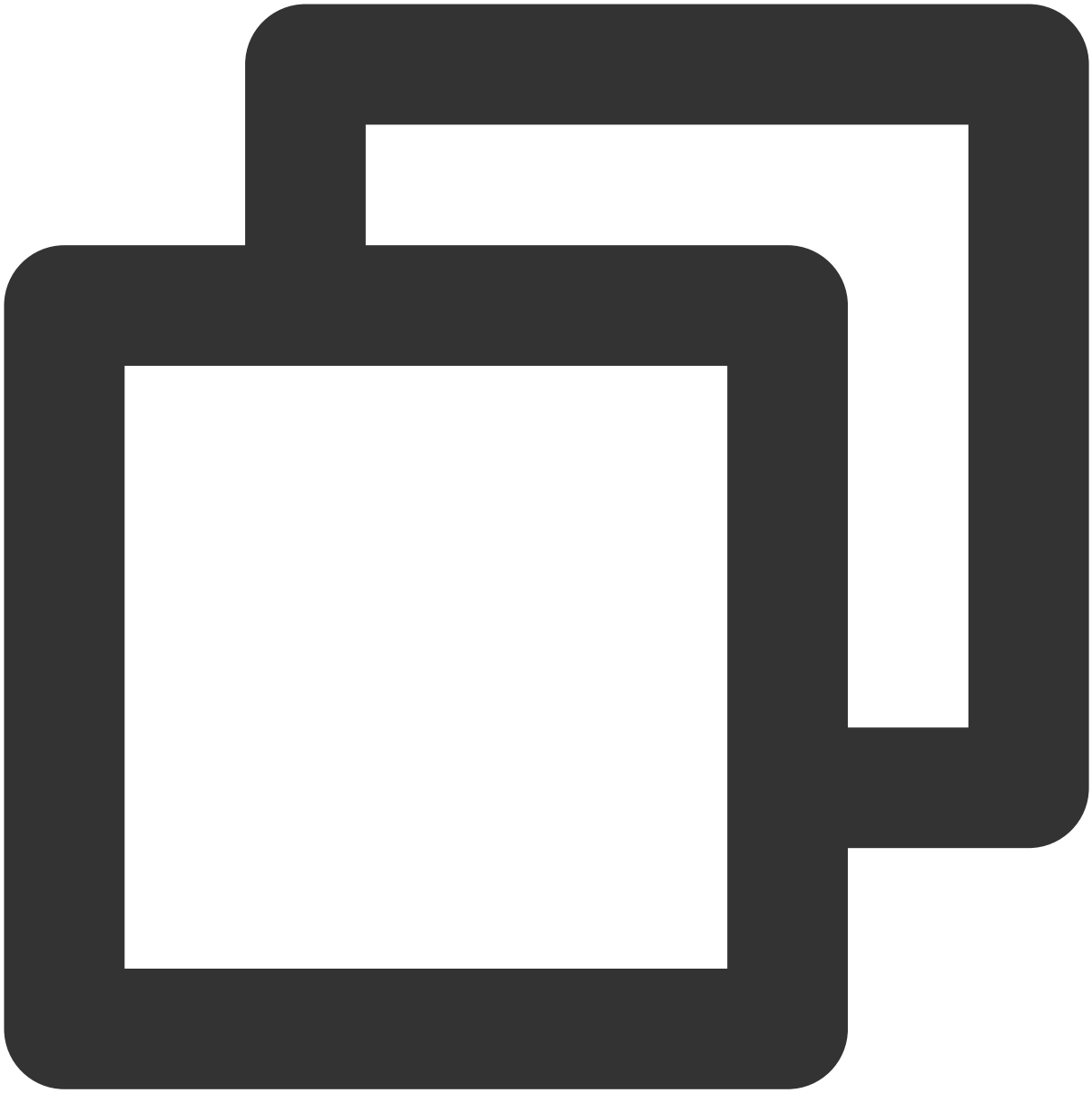
示例：



```
> select typeof(decimal(1));
 decimal(10, 0)
```

GET_IDCARD_BIRTHDAY

函数语法：



```
GET_IDCARD_BIRTHDAY(<string> idcardno)
```

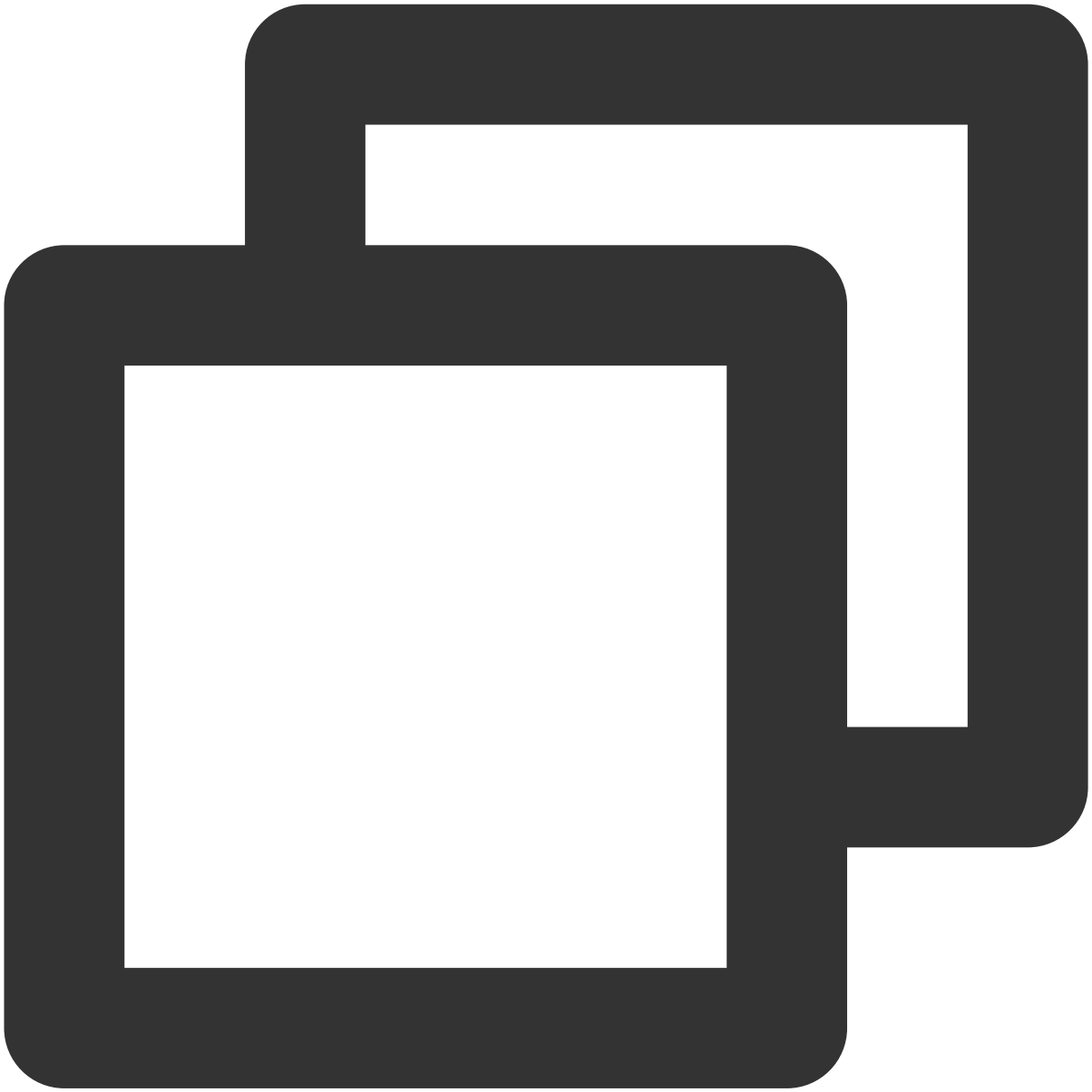
支持引擎：SparkSQL

使用说明：获取身份证号中的出生日期

idcardno：必填，string 类型，必须是15号或18位身份证号码，会校验身份证合理性。不允许其他非 null 型输入。输入为 null 时返回为 null。

返回类型：date

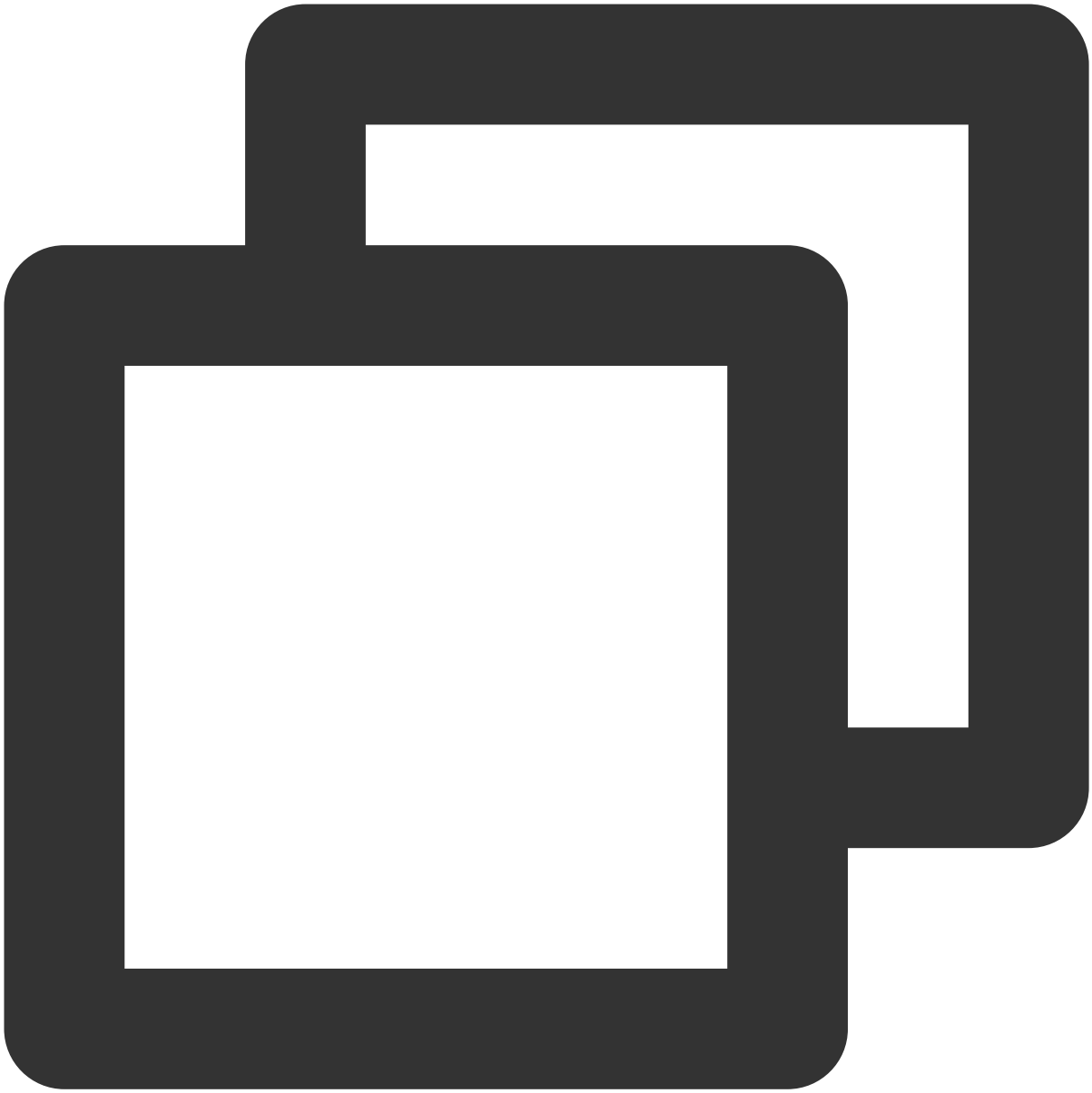
示例（以下测试身份证号为随机样例，不代表正确身份证号）



```
> SELECT get_idcard_birthday('421081199001011222');
1990-01-01
```

GET_IDCARD_SEX

函数语法：



```
GET_IDCARD_SEX(<string> idcardno)
```

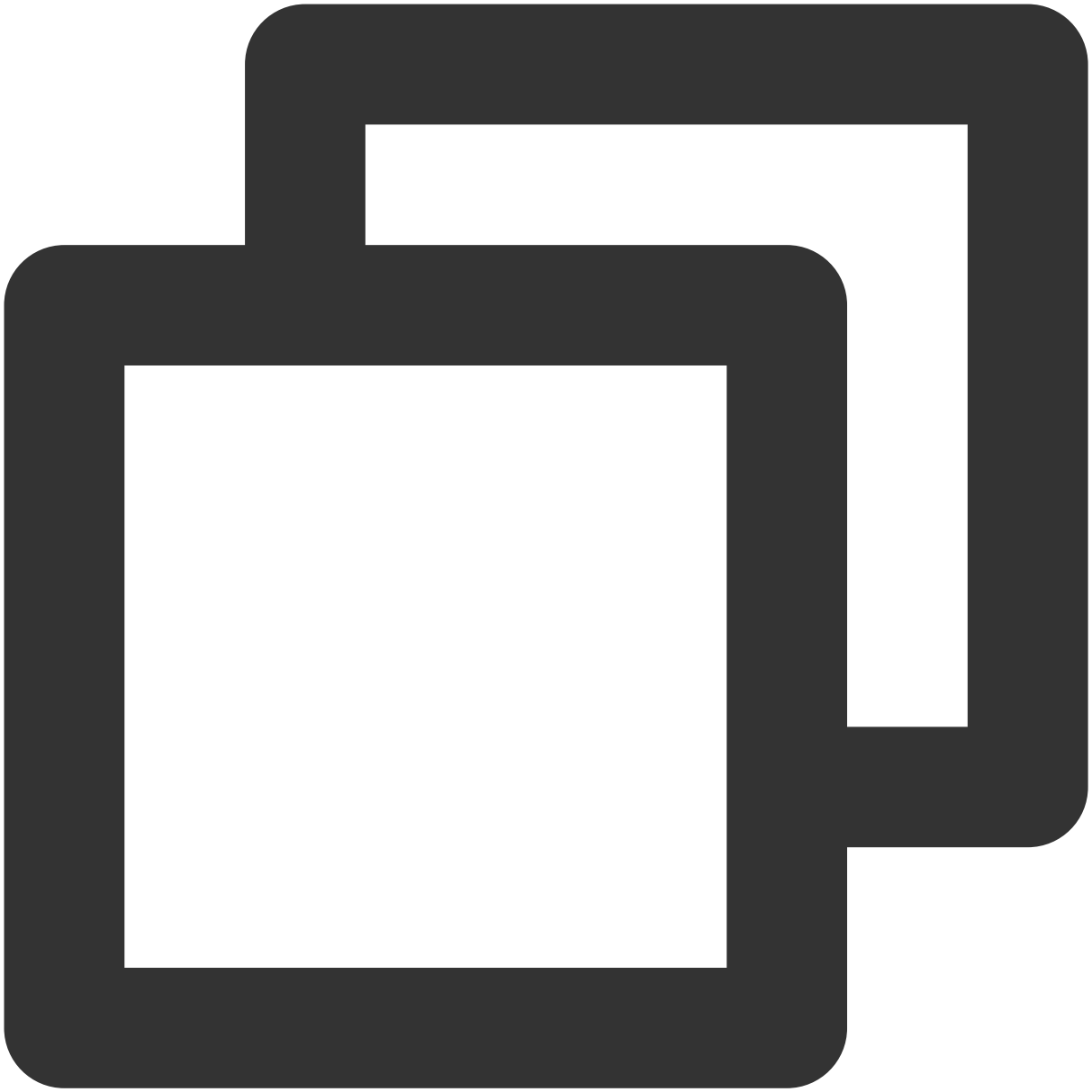
支持引擎：SparkSQL

使用说明：获取身份证号中的性别

idcardno：必填，string 类型，必须是15号或18位身份证号码，会校验身份证合理性。不允许其他非 null 型输入。输入为 null 时返回为 null。

返回类型：string

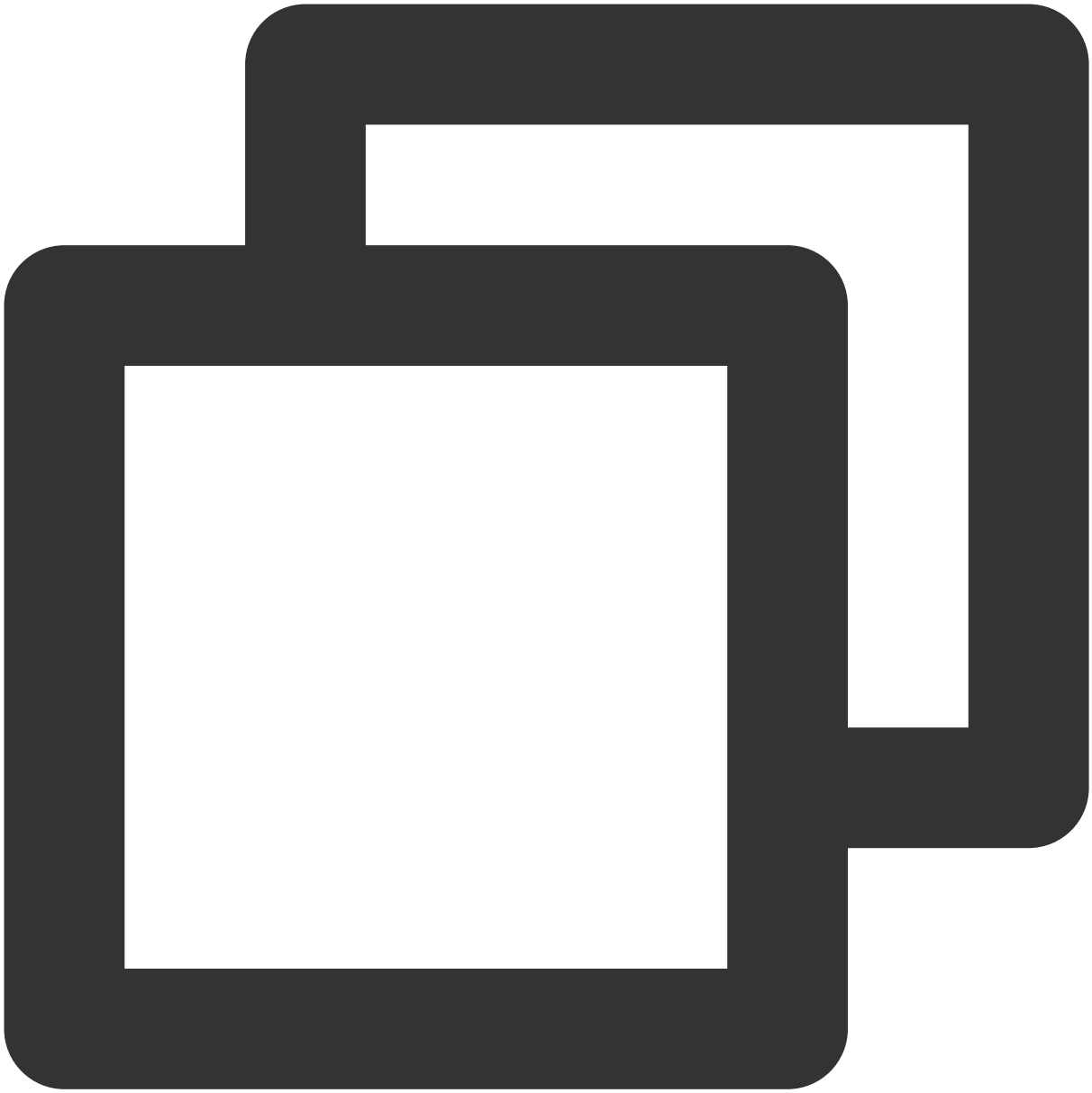
示例：（以下测试身份证号为随机样例，不代表正确身份证号）



```
> SELECT get_idcard_birthday('421081199001011222');
```

GET_IDCARD_AGE

函数语法：



```
GET_IDCARD_AGE(<string> idcardno)
```

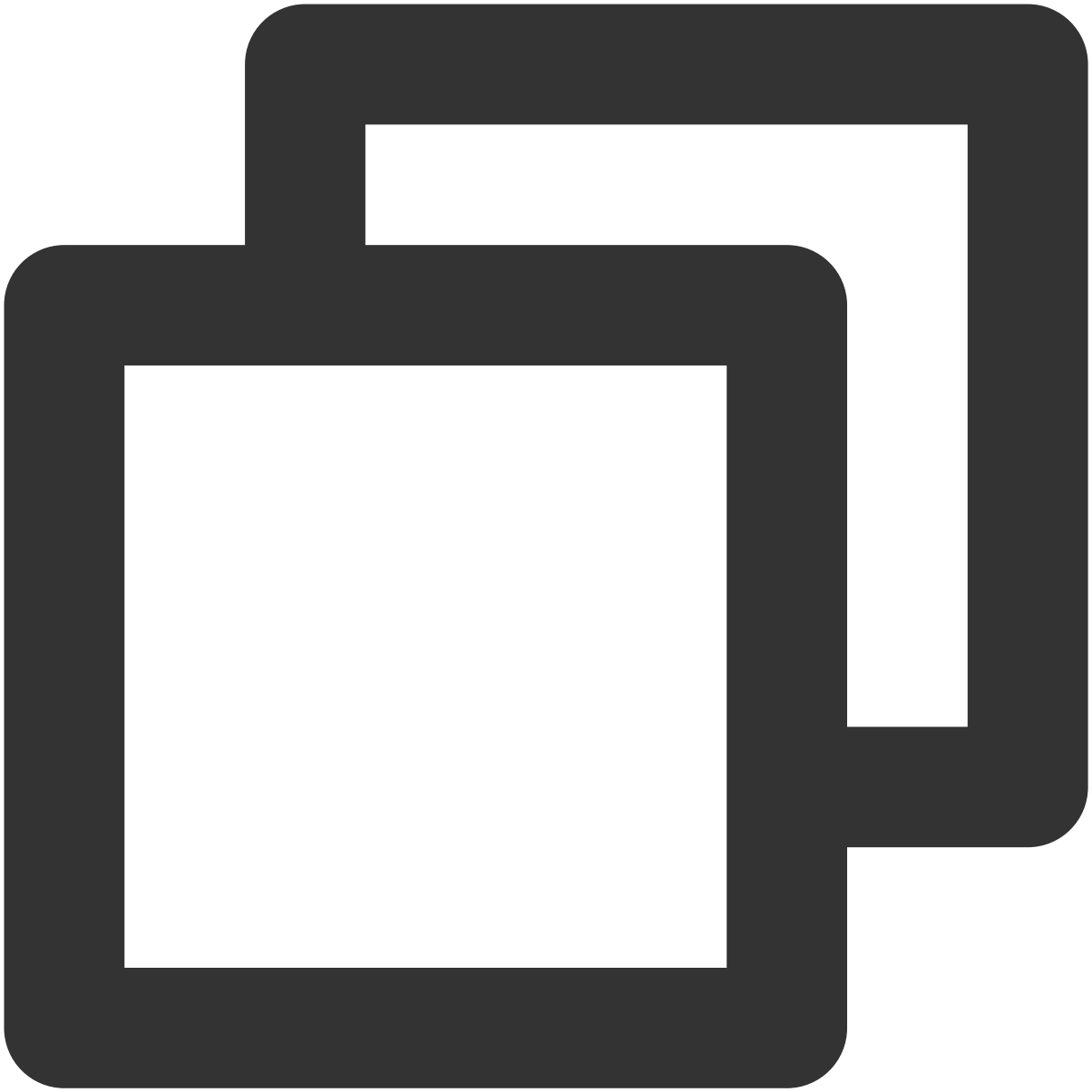
支持引擎：SparkSQL

使用说明：获取身份证号中的年龄

idcardno：必填，string 类型，必须是15号或18位身份证号码，会校验身份证合理性。不允许其他非 null 型输入。输入为 null 时返回为 null。

返回类型：int

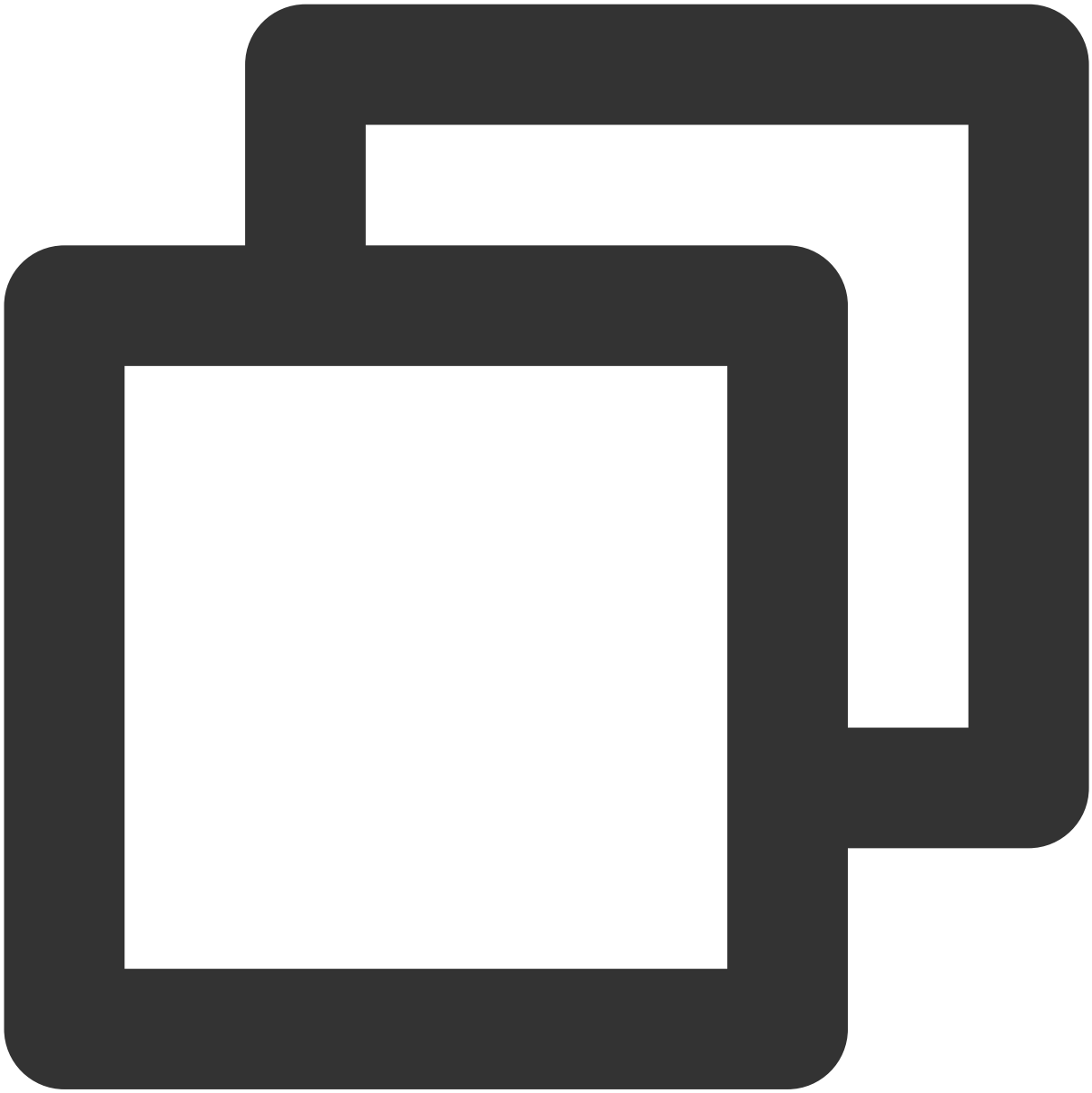
示例：（以下测试身份证号为随机样例，不代表正确身份证号）



```
> SELECT get_idcard_age('421081197001021233');
53
```

MAX_PT

函数语法：



```
MAX_PT(<const string> tableFullName)
```

支持引擎：SparkSQL

使用说明：获取指定分区表中的最大值

tableFullName：必填，string 类型，必须是常量值，否则会报错。tableFullName 由三段组成 catalog.db.table，其中 catalog 和 db 省略时会默认取当前 session 的配置，建议写全。

当输入的表不是分区表时，会报错。

该函数仅返回**有数据**的最大分区，由于是否有数据涉及到元数据中心，所以**对外部表，需要注意是否有执行 ANALYZE 语句**将分区的统计信息汇报到元数据中。

分区值按字典序排列取最大值。当有多级分区时，仅取**第一级分区**做排序。

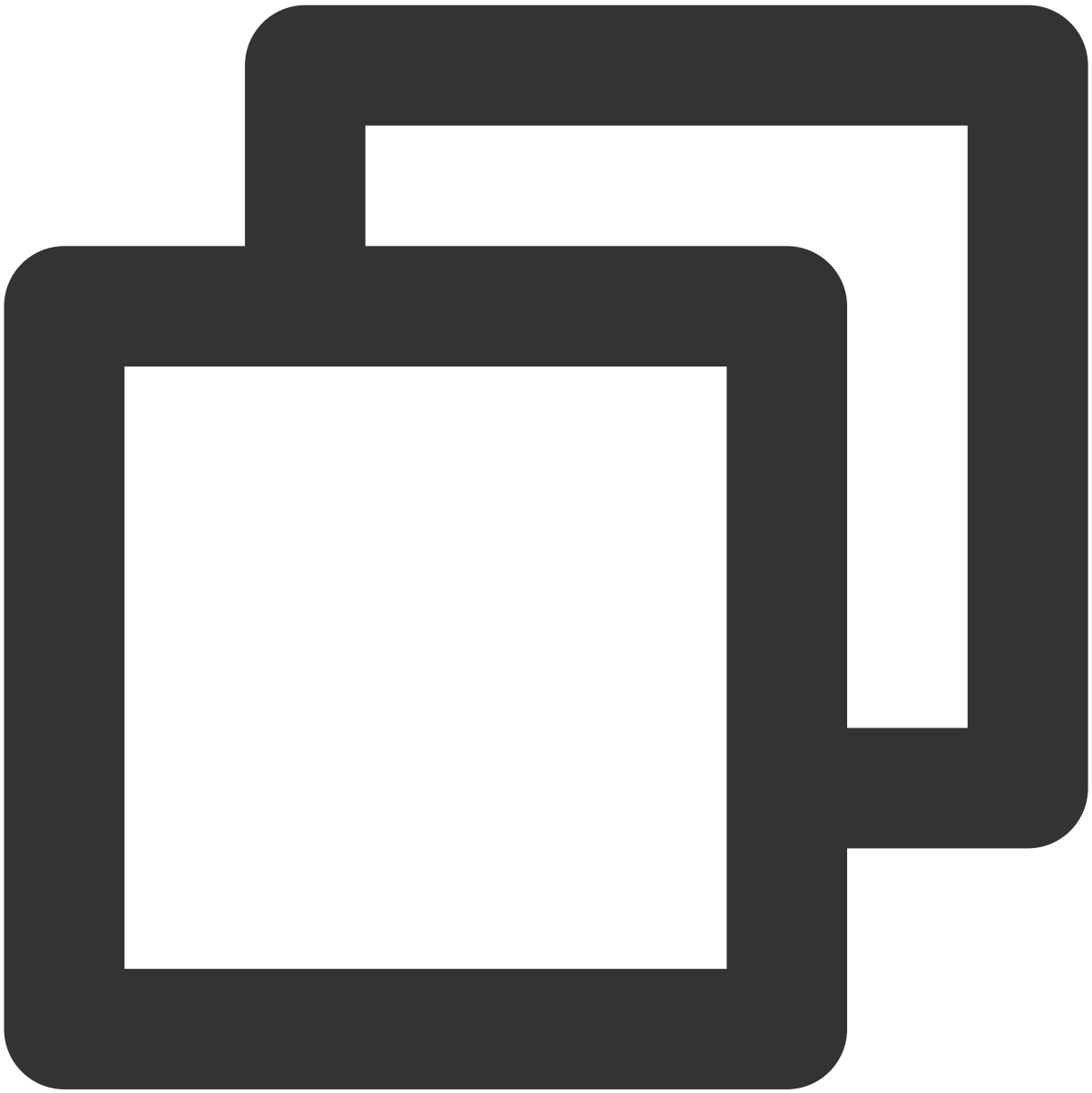
返回类型：string

示例：

```
> SELECT max_pt('test.tableName');
20231024
> select * from test.tableName where dt=max_pt('test.tableName');
等同于select * from test.tableName where dt='20231014';
```

TRANS_ARRAY

函数语法：



```
TRANS_ARRAY(<int> numKeys, <string> separator, <key1>, <key2>, ..., <col1>, <col2>,
```

支持引擎：SparkSQL

使用说明：将指定的多列 cols 拆分并转置成多行，同时支持指定部分列 keys 作为转置的 key。

numKeys：int 型，必填，表示作为转置的 keys 列的个数，必须大于等于0。

separator：string 类型，必填，当 cols 为字符串时，需要根据 separator 做拆分；当 cols 为数组时，该参数可随意填写。

keys：任意类型，个数由 numKeys 决定。

cols：string 类型或数组类型，指定的列中除去 keys 都视为 cols，即要拆分和转置的列。**所有 cols 的类型必须相同**，即全部为字符串，或者全部为数组，也可以特殊支持字符串和字符串数组两种类型组合的 cols。当没有 cols 时，只会输出一行。

注意1：keys 和 cols 两者的数量之和必须大于0。

注意2：当 cols 拆分后的长度不相等时，最终转置后的行数以最长的那一列为准，其他列补 NULL。

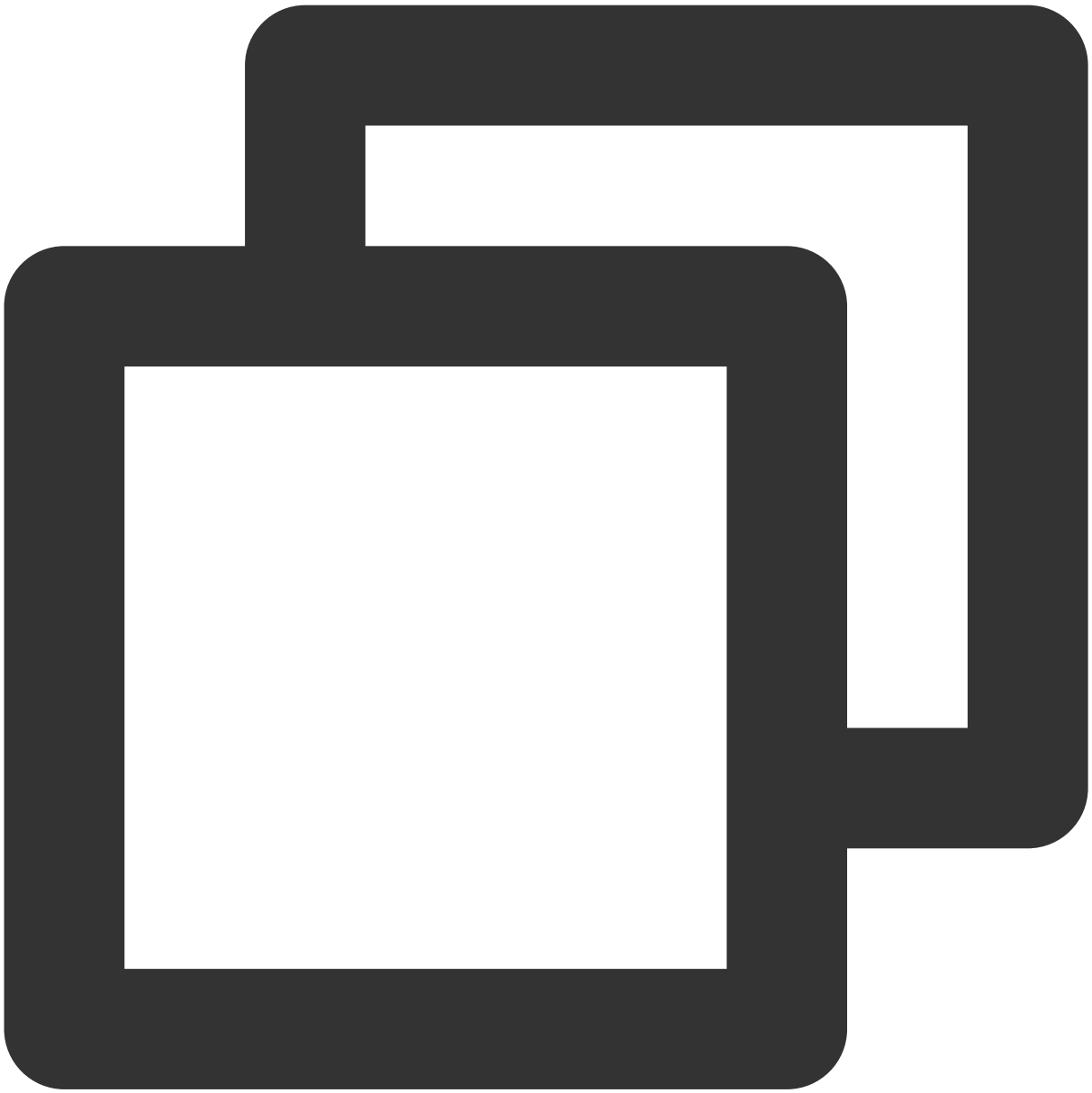
返回类型：任意类型。keys 的类型保持不变，cols 类型为字符串或数组的元素类型。

示例：

```
> SELECT trans_array(1, ',', key, trans1, trans2) as (key, col1, col2) from values
1 2 4
1 3 5
> SELECT trans_array(0, ',', key, trans1, trans2) as (key, col1, col2) from values
1 2 4
NULL 3 NULL
> SELECT trans_array(3, ',', key, trans1, trans2) as (key, col1, col2) from values
12,3 [4,5]
```

TRANS_COLS

函数语法：



```
TRANS_COLS(<int> numKeys, <key1>, <key2>, ..., <col1>, <col2>, ...)
```

支持引擎：SparkSQL

使用说明：将指定的多列cols转置成多行，同时支持指定部分列keys作为转置的key。

numKeys：int型，必填，表示作为转置的keys列的个数，必须大于等于0。

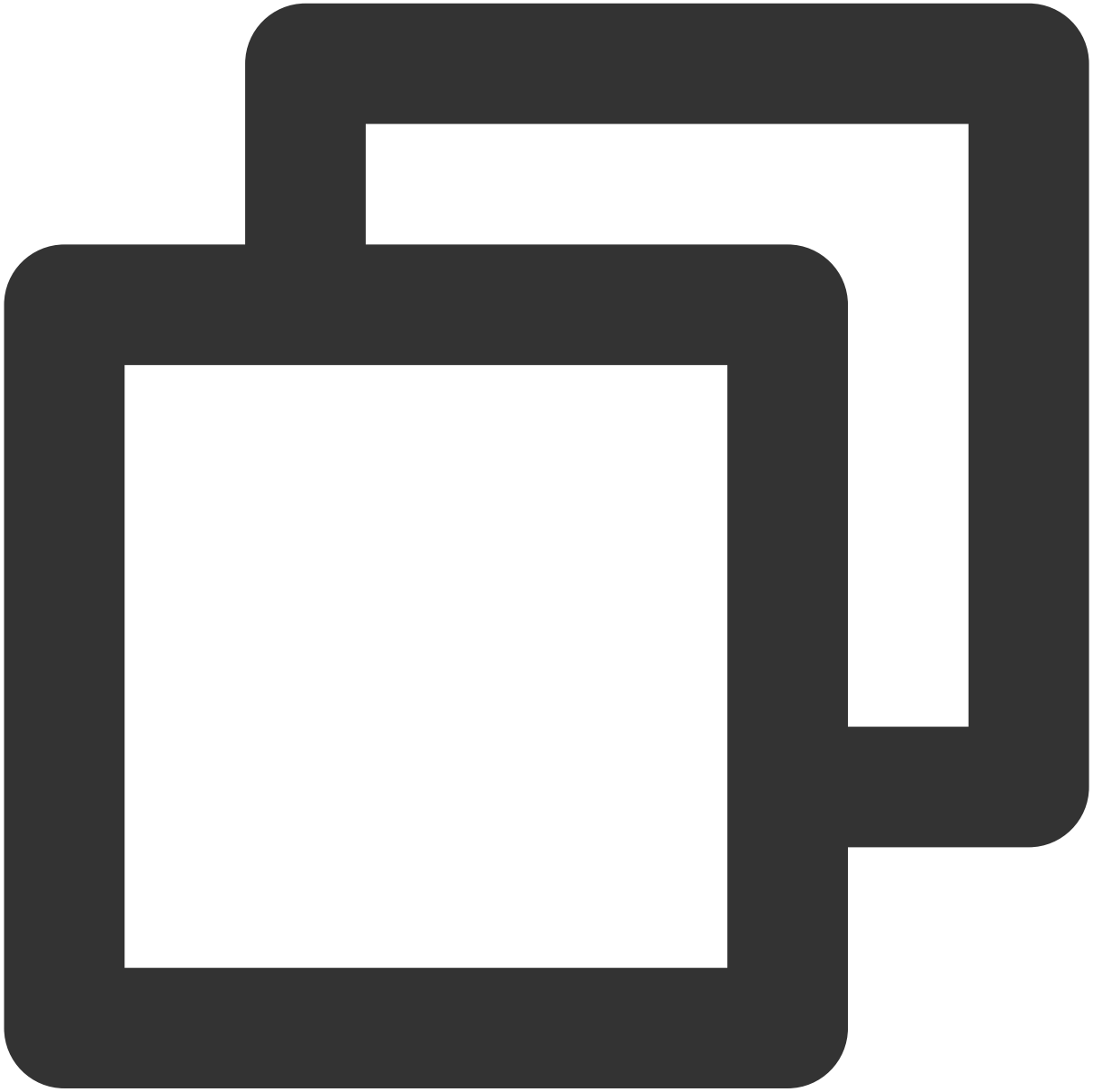
keys：任意类型，个数由numKeys决定。

cols：任意类型，指定的列中除去keys都视为cols，即要转置的列。**所有cols的类型必须相同**。当没有cols时，只会输出一行。

注意：keys和cols两者的数量之和必须大于0。

返回类型：任意类型。输出会包含一列idx，表示该行在转置的所有行中的序号，keys的类型保持不变，cols类型保持不变。

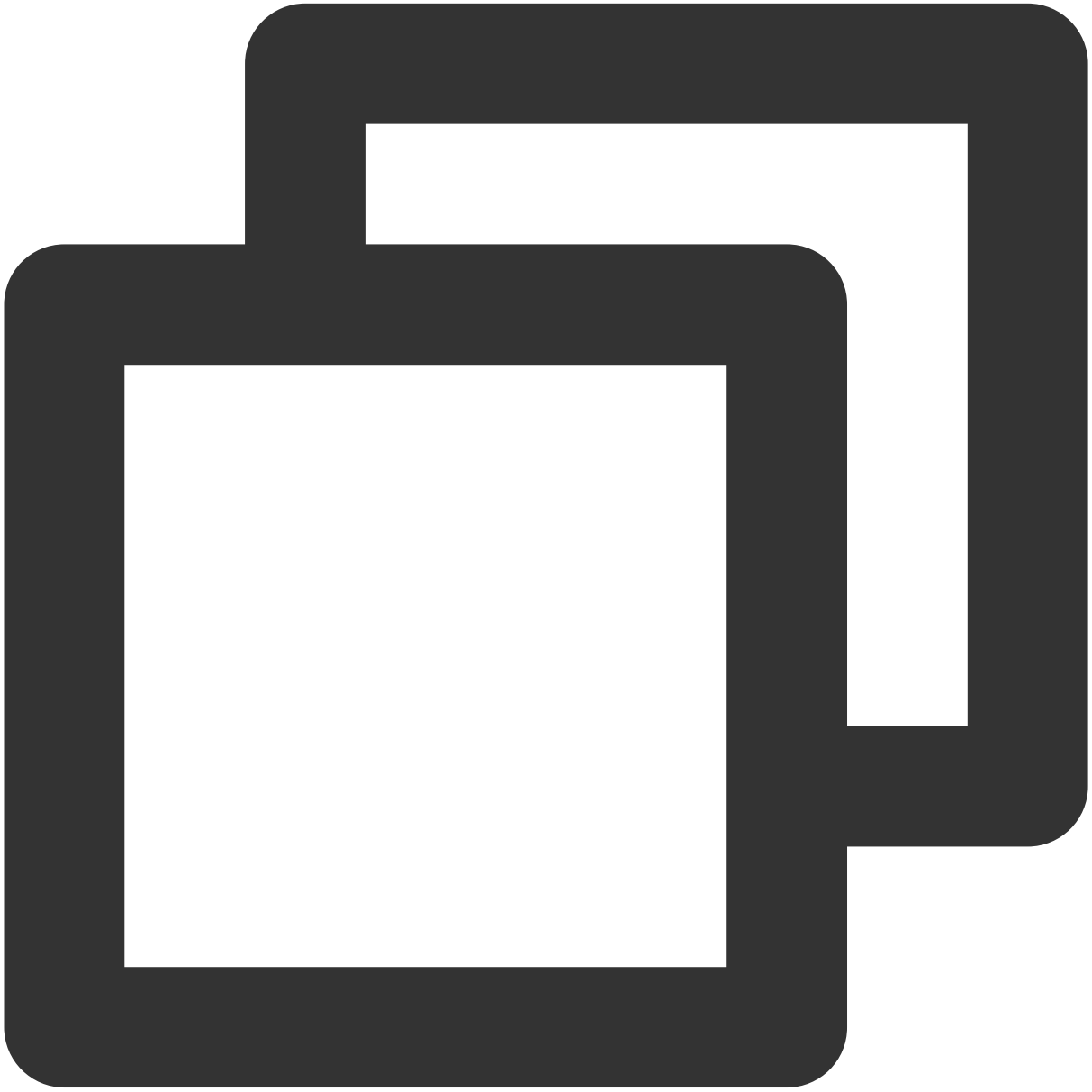
示例：



```
> SELECT trans_cols(1,sid,ip1,ip2) as (idx, sid, ip) from values ('s1','0.0.0.0','1
1 s1 0.0.0.0
2 s1 1.1.1.1
```

SAMPLE

函数语法：



```
SAMPLE(<int/long> totalParts[, <int/long> pointParts][, <col1>, <col2>, ...])
```

支持引擎：SparkSQL

使用说明：采样函数，对全局或按指定的列Hash值划分为totalParts份，并选择指定的第pointParts份结果返回。

totalParts：int/long型，必填，表示总共需要划分的分区数。

pointParts：int/long型，可填，表示要选择返回的第几份数据，默认值为1，必须小于等于totalParts。

cols：任意类型，可填，可以指定任意多列作为划分数据时的key，将按这些列的hash值来划分。

注意1：当cols不指定时，或者当前指定的cols全部为NULL时，为保证数据不倾斜，会**使用随机值替代hash值**做计算。而当cols指定时，只要有一列不为NULL，就会对hash值计算及分区结果有影响。

注意2：由于Hash值与随机值的种子时固定的，因此反复执行时只要数据顺序一致，单个partition内的采样数据也是一致的。对于存在小文件合并的场景，可能导致采样顺序变化，这个时候建议指定列做采样。

返回类型：boolean，为true表示被采样，为false表示未采样。

示例：

```
> select * from values (1, 'm1'), (2, 't2'), (3, 'z3') as tab(dt, uid) where sample
2 t2
→ select * from values (1, 'm1'), (2, 't2'), (3, 'z3') as tab(dt, uid) where sample
2 t2
3 z3
```

TO_CHAR

函数语法：

```
TO_CHAR(<boolean|int|long|double|decimal> data [, <string> format])
TO_CHAR(<date|timestamp> data, string format)
```

支持引擎：SparkSQL

使用说明1：按指定模板格式化数字。format可以省略，即直接将数字转为字符串。否则，format的定义如下：

'0' or '9': 数字的占位符，对字符串首部，如果0占位则没有数字时用0替代，如果9占位则没有数字时用空格代替；对字符串尾部，如果没有数字统一用0代替。

'.' or 'D': 小数点的占位符，只允许出现一次。

',' or 'G': 逗号，例如百、百万的占位符，左右必须是0-9数字。

'$': 美元符，只能出现一次。

'S' or 'MI': 负号和正号的占位符，只能出现在字符串首尾最多一次。

'PR': 只允许出现在字符串结尾一次，当输入的数据是符负号时会给数字求正并加上尖括号。

使用说明2：按指定模板格式化日期或时间。format不能省略，定义如下：

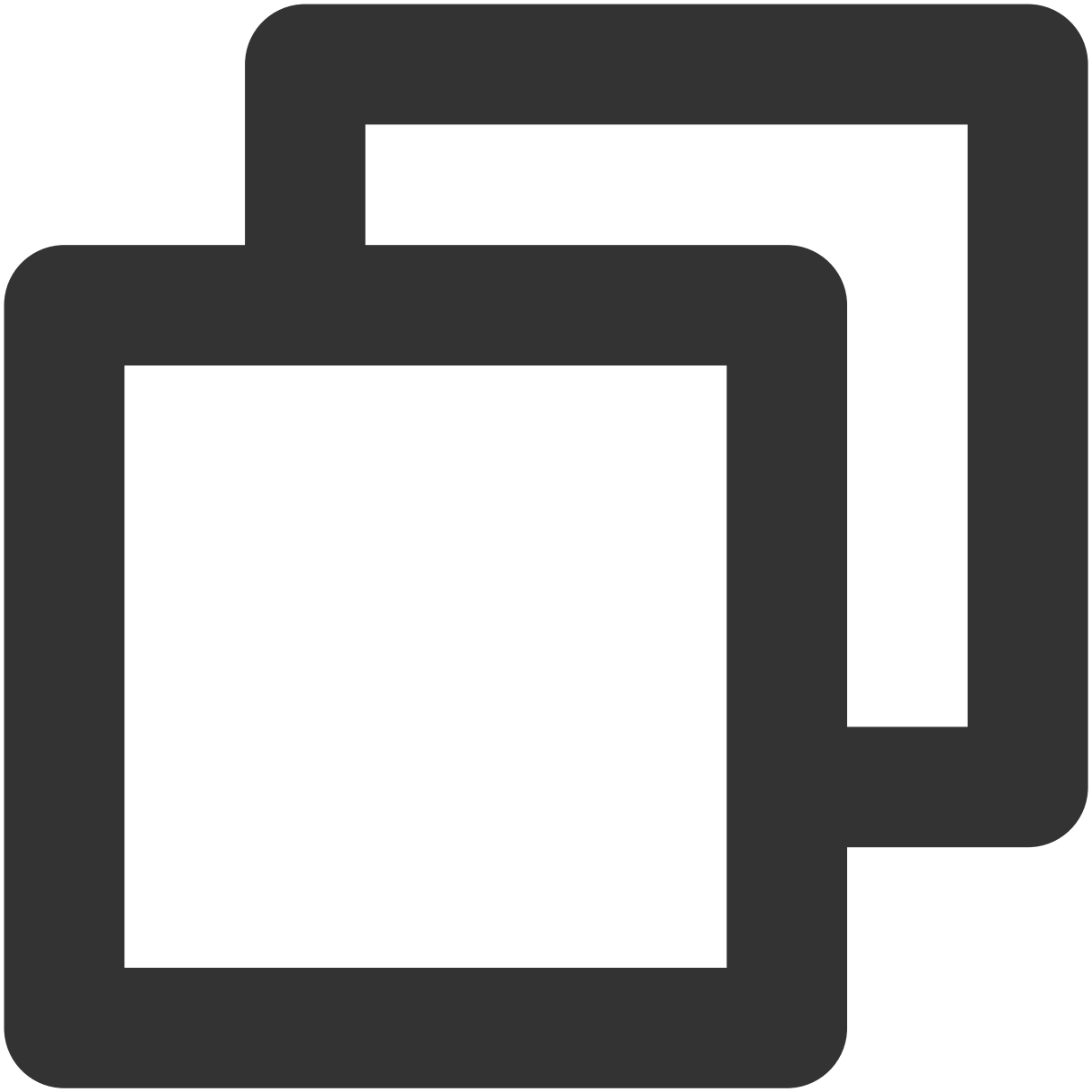
yyyyMMdd HH:mm:ss.SSS中的任意字符组成的模板。

详细参考: https://spark.apache.org/docs/latest/sql-ref-datetime-pattern.html

使用说明3：TO_CHAR支持隐式转换第一个参数的类型，当第一个参数为string类型时，函数会默认转为 timestamp 走模板格式化日期或时间的逻辑。

返回类型：string类型。

示例：



```
> select to_char(124.23);
124.23
> select to_char(124.23, '00999.9999');
<space><space>124.2300
> select to_char(4124.23, '9,999.99');
4,124.23
> select to_char(-124.23, '999.99S');
124.23-
> select to_char(-4124.23, '$9,999.99PR');
<$4,124.23>
> select to_char(date '2016-12-31 12:34:56', 'yyyyMMddHHmmss');
```

```
20161231123456
```

版权所有：腾讯云计算（北京）有限责任公司

Presto 内置函数

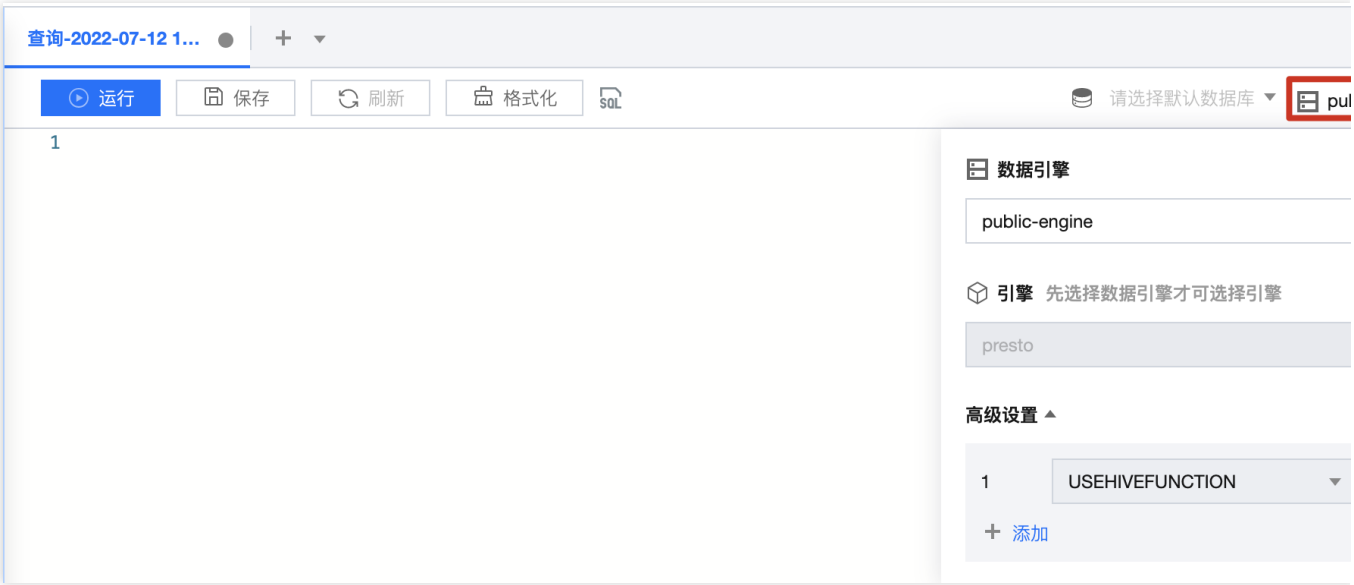
最近更新时间：2024-08-07 17:34:43

数据湖计算 DLC 在 统一函数 之外，同时支持 Presto 的内置函数。

如何开启 Presto 内置函数应用

途径一：在数据探索对数据引擎进行函数配置

1. 登录数据湖计算 DLC 控制台，选择服务地域。
2. 进入数据探索，选择数据引擎，当引擎内核为 Presto 时，可在高级配置中选择参数 **USEHIVEFUNCTION**，将该参数配置为 **false** 即可在使用该数据引擎进行 SQL 任务时使用 Presto 内置函数。



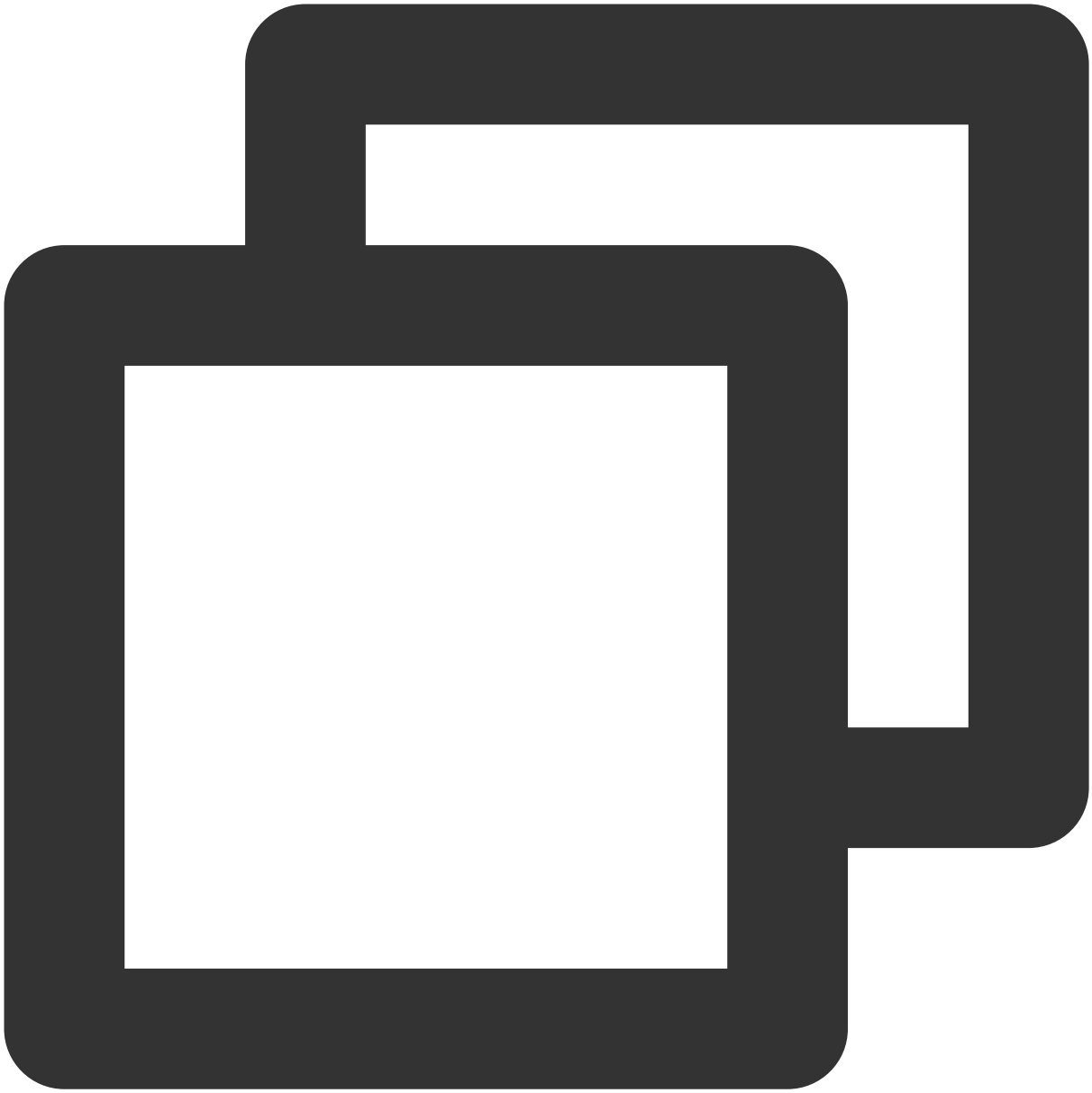
注意

在当前查询 session，所有使用该数据引擎的查询任务皆可使用 Presto 内置函数。

途径二：在 SQL 语句中添加参数

如您希望某个 SQL 任务调用 Presto 内置函数，可通过在 SQL 任务中添加配置信息实现。

示例：



```
SELECT/*+OPTIONS('useHiveFunction'='false')*/prestofunc(xx)
```

途径三：使用 API 时增加配置参数

在 task 结构体中 config 中设置 kv，useHiveFunction=false。

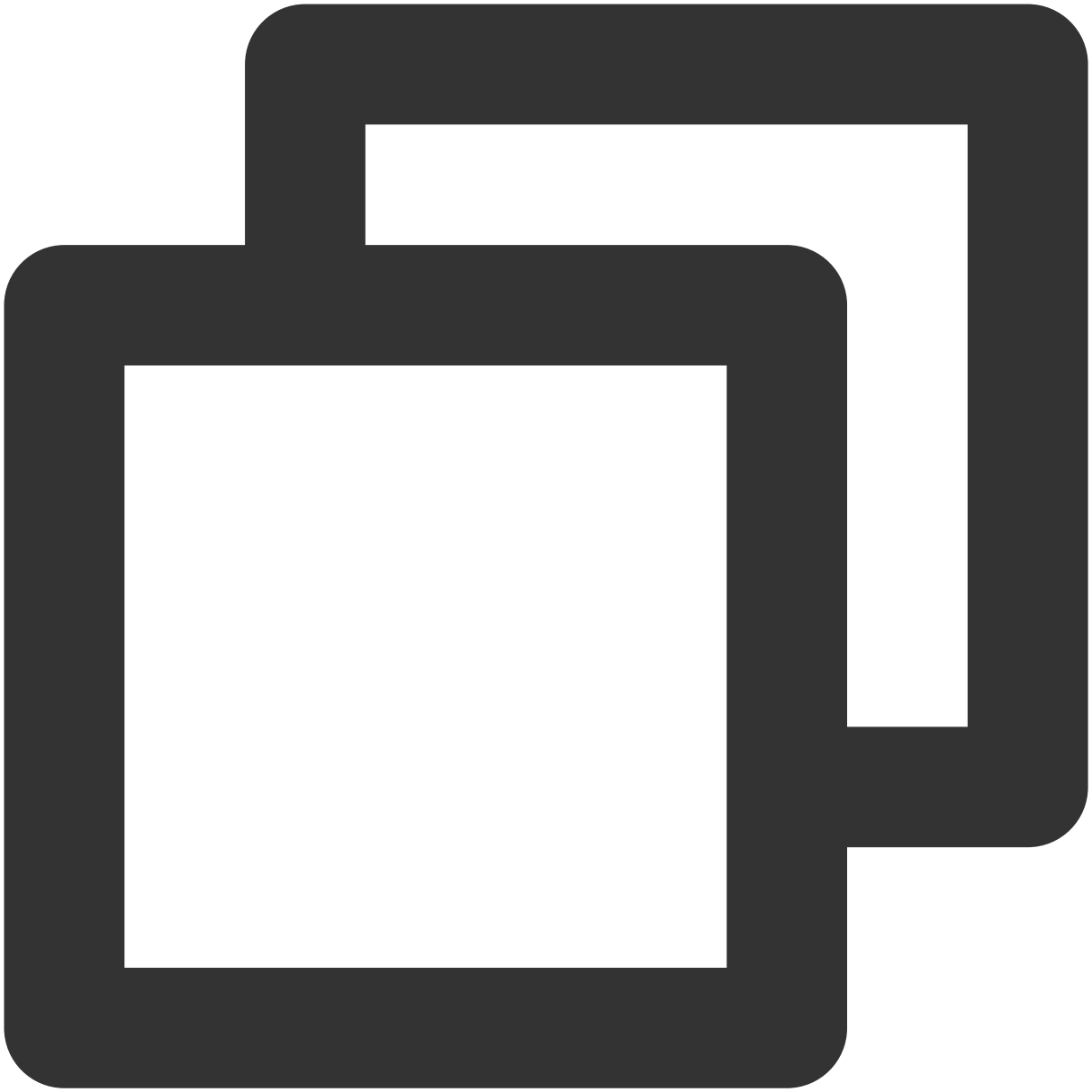
SQLTask

SQL查询任务

被如下接口引用：CreateTask。

名称	类型	必选	描述
SQL	String	是	base64加密后的SQL语句
Config	Array of KVPair	否	任务的配置信息

示例：



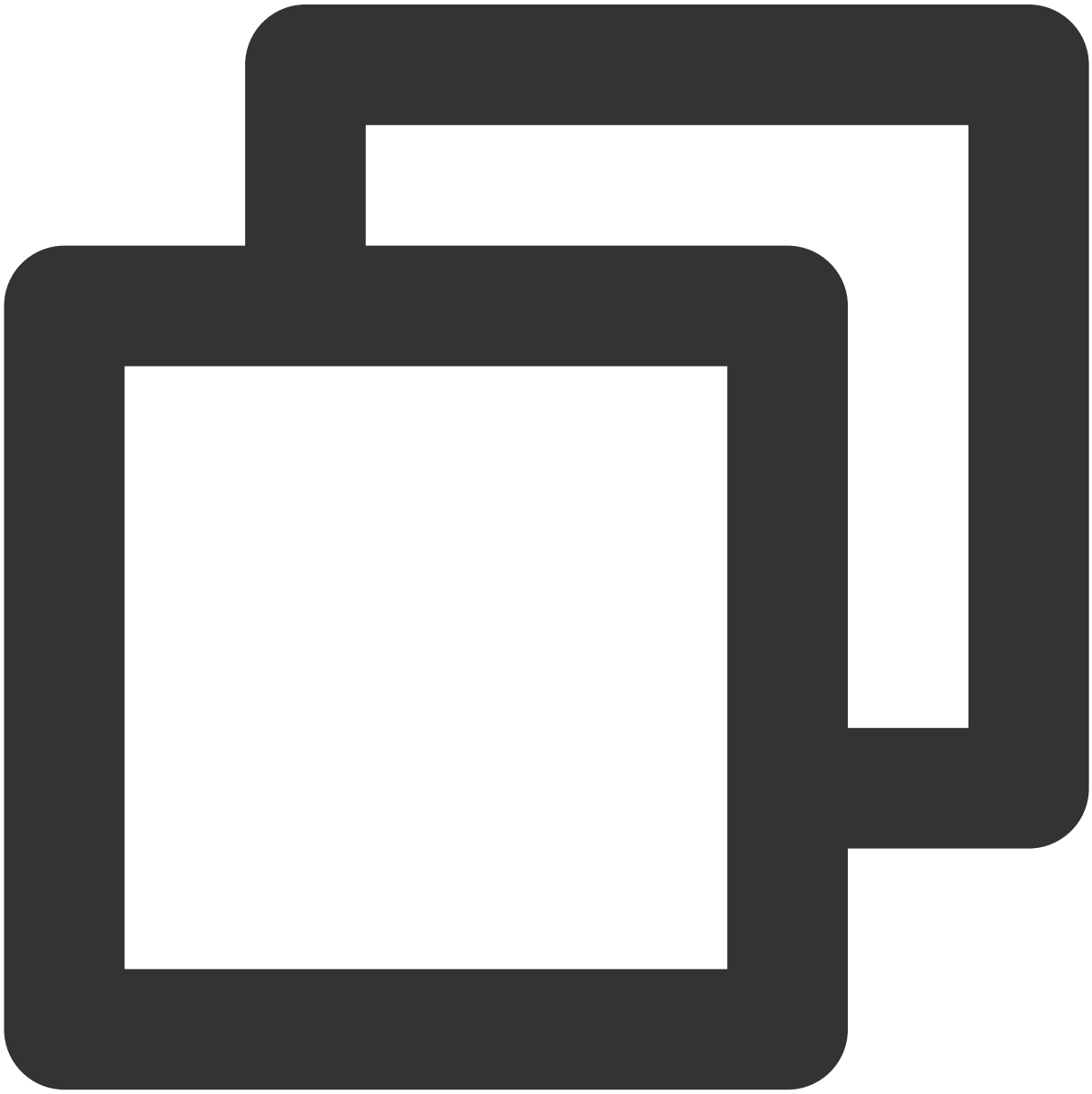
```
Stringstatement="SELECTdate_add('week',3,TIMESTAMP'2001-08-2203:04:05.321')";
TasksInfotask=newTasksInfo();
task.setTaskType("SQLTask");
task.setSQL(Base64.getEncoder().encodeToString(statement.getBytes()));
//添加以下参数配置
KVPairpair=newKVPair();
pair.setKey("useHiveFunction");
pair.setValue("false");
task.setConfig(newKVPair[]{pair});

CreateTasksRequestrequest=newCreateTasksRequest();
```

```
request.setDatabaseName("");
request.setDataEngineName(PRESTO_ENGINE);
request.setTasks(task);
```

参考 API 文档：任务创建。

途径四：使用 JDBC 进行任务创建时添加参数

在 JDBCURL 路径中添加 `&presto.USEHIVEFUNCTION=true` 参数或 `info.setProperty("presto.USEHIVEFUNCTION","false");` 。



```
Connectionconnection=
```

```
DriverManager.getConnection("jdbc:dlc:dlc.tencentcloudapi.com?task_type=SQLTask&x省
或者
info.setProperty("presto.USEHIVEFUNCTION","false");
info.setPropertv("user","");
info.setProperty("password","");
```

支持的 Presto 内置函数列表

数学函数

函数名称	返回值	函数功能描述
abs(x)	与 x 一致	返回x的绝对值
cbrt(x)	double	返回 x 的立方根
ceil(x) | ceiling(x)	与 x 一致	返回 x 向上最接近的整数。例如：ceil(2.2); -->3
cosine_similarity(x, y)	double	返回向量 x 和 y 的相似度 例如：SELECT cosine_similarity(MAP(ARRAY['a'], ARRAY[1.0]), MAP(ARRAY['a'], ARRAY[2.0])); -->1.0
degrees(x)	double	将弧度转换为角度
e()	double	返回常数 e
exp(x)	double	返回 e 的x次方
floor(x)	与 x 一致	返回 x 向下最接近的整数。例如：floor(2.2); -->2
ln(x)	double	返回x的自然对数
log2(x)	double	返回以 2 为底的 x 的对数
log10(x)	double	返回以 10 为底的 x 的对数
mod(n, m)	与 n 一致	返回 n 除以 m 的余数
pi()	double	返回常数Pi
pow(x, p) | power(x, p)	double	返回 x 的 p 次幂
radians(x)	double	将角度x(以度为单位)转换为弧度
rand() | random()	double	返回 0.0 <= x < 1.0 范围内的随机值。
random(n)	与 n 一致	返回介于 0 和 n (不包含n)之间的随机数

secure_rand() | secure_random()	double	返回0.0 <= x < 1.0范围内的加密随机值
secure_random(lower, upper)	与 lower 一致	返回范围为 lower <= x < upper 的加密随机值，其中lower < upper
round(x)	与 x 一致	返回x四舍五入到最接近的整数
round(x, d)	与 x 一致	返回x四舍五入到小数点后 d 位
sign(x)	与 x 一致	返回 x 所对应的正负号，x > 0 返回1；x< 0，返回-1；x=0， 返回 0
sqrt(x)	double	返回 x 的平方根
truncate(x)	bigint	返回截取小数点后的数 例如：truncate(4.9) -->4
truncate(x, n)	double	返回 x 截取小数点后n位。n<0时，从小数点左边截取 例如：truncate(12.333, -1) --> 10.0；truncate(12.333,0) --> 12.0；truncate(12.333,1) --> 12.3；

二进制函数

函数名称	返回值	函数功能描述
length(b)	bigint	以字节为单位，返回 b 的二进制长度
concat(b1, ..., bN)	varbinary	返回连接b1,...,bN的二进制值
substr(b, start)	varbinary	返回从 b 中 start 位开始截取二进制值，start > 0 时从开头截取，star t< 0 时从末尾开始截取
substr(b, start, length)	varbinary	返回从 b 中 start 位开始截取 length 长度 的二进制值，start > 0 时从开头截取，star t< 0 时从末尾开始截取
to_base64(b)	varchar	将二进制数据 b 转为 base64 字符串
from_base64(string)	varbinary	将 base64 编码的字符串转为二进制数据
to_base64url(b)	varchar	使用URL安全字符，将二进制数据 b 转为 base64 字符串
from_base64url(string)	varbinary	使用URL安全字符，将 base64 编码的字符串转为二进制数据
from_base32(string)	varbinary	将 base32 编码的字符串转为二进制数据

版权所有：腾讯云计算（北京）有限责任公司

to_base32(b)	varchar	将二进制数据 b 转为 base32 字符串
to_hex(b)	varchar	将二进制数据 b 转为16进制字符串
from_hex(string)	varbinary	将 16进制编码的字符串转为二进制数据
lpad(b, size, padb)	varbinary	返回在 b 的左边拼接 padb ,直到长度达到 size的二进制数
rpad(b, size, padb)	varbinary	返回在 b 的右边拼接 padb ,直到长度达到 size的二进制数
crc32(b)	bigint	使用CRC32算法计算表达式的循环冗余校验值
md5(b)	varbinary	计算二进制 b 的 md5 值
sha1(b)	varbinary	计算二进制 b 的 sha1 值
sha256(b)	varbinary	计算二进制 b 的 sha256 值
sha512(b)	varbinary	计算二进制 b 的 sha512 值
xxhash64(b)	varbinary	计算二进制 b 的 xxhash64 值

位运算函数

函数名称	返回值	函数功能描述
bit_count(x, bits)	bigint	返回 bit 在 x 中的位数
bitwise_and(x, y)	bigint	返回 x , y 的与
bitwise_not(x)	bigint	返回 x 的按位取反
bitwise_or(x, y)	bigint	返回 x, y的或
bitwise_xor(x, y)	bigint	返回 x, y的异或
bitwise_left_shift(value, shift)	与输入一致	返回 value 的左移动 shift 位的值 例如：bitwise_left_shift(TINYINT '7', 2) --> 28
bitwise_right_shift(value, shift)	与输入一致	返回 value 的右移动 shift 位的值 例如：bitwise_right_shift(TINYINT '7', 2) -->1

字符串函数

函数名称	返回值	函数功能描述

chr(n)	varchar	返回 n 的 Unicode 字符
codepoint(string)	integer	返回字符串的Unicode值 例如：codepoint('0') -->48
concat(string1, ..., stringN)	varchar	返回 string1,....,stringN的连接字符串
length(string)	bigint	返回字符串的长度
hamming_distance(string1, string2)	bigint	返回两个（相同长度）字符串对应位置的不同字符的数量 例如：hamming_distance(abc, art) -->2
levenshtein_distance(string1, string2)	bigint	返回两个字串之间，由一个转成另一个所需的最少编辑(插入、删除、替换)的次数 例如： levenshtein_distance('ab', 'abcde') -->3
lower(string)	varchar	返回string转为小写后的字符串
upper(string)	varchar	返回string转为大写后的字符串
ltrim(string)	varchar	从字符串中删除左边空格
rtrim(string)	varchar	返回从字符串中删除后面空格
trim(string)	varchar	返回删除字符串中的空格的字符串
replace(string, search)	varchar	从字符串中删除所有search字符
replace(string, search, replace)	varchar	用 replace 字符串替换 search 字符串 例如：replace('abcavc', ‘a', ‘c') -->'cbccvc'
reverse(string)	varchar	返回字符的反向排列 例如：reverse('abc') -->'cba'
lpad(string, size, padstring)	varchar	在 string 的左边拼接 padstring，直到字符串长度达到size。如果有size小于 string，则将string剪切为长度size的字符串。
rpad(string, size, padstring)	varchar	在 string 的右边拼接 padstring，直到字符串长度达到达到size。如果有size小于 string，则将string剪切为长度size的字符串。
split(string, delimiter)	array	返回 delimiter 分割字符串后的一个数组
split(string, delimiter, limit)	array	返回 delimiter 分割字符串后按 limit 大小限制的数组，limit > 0,数组最后一个元素包含字符串中的所有剩余内容

		例如：select split('ab-cd-efg', '-', 2) -->['ab','cd-efg']
split_part(string, delimiter, index)	varchar	返回 delimiter 分割字符串后的数组中，index 位置的字符串,字段索引从 1 开始。如果 index >数组长度，则返回 null 例如：select split_part('ab-cd-efg', '-', 2) -->'cd'
split_to_map(string, entryDelimiter, keyValueDelimiter)	map<varchar, varchar>	返回 entryDelimiter 和 keyValueDelimiter 拆分字符串后的map 例如：select split_to_map('a:1,b:2', ',', ':') --> {'a':'1','b':'2'}
split_to_map(string, entryDelimiter, keyValueDelimiter, function(K, V1, V2, R))	map<varchar, varchar>	返回 entryDelimiter 和 keyValueDelimiter 拆分字符串后的map,如果有重复key，则根据function指定的规则返回key对应的value 例如：split_to_map('a:1,b:2;a:3',‘,' , ':', (k, v1, v2) -> v1)-->{'a':'1','b':'2'}
substr(string, start)	varchar	返回字符串 string 中 start 位置之后的字符串。位置从1开始, start <0时，从字符串末尾开始计数
substr(string, start, length)	varchar	返回字符串 string 中 start 位置开始 length 长度的字符串。位置从1开始, start <0时，从字符串末尾开始计数
position(substring IN string)	bigint	返回string中substring的第一个出现的位置。位置从1开始。如果未找到，则返回0
strpos(string, substring)	bigint	返回string中substring的从左向右第一次出现的位置。位置从1开始。如果未找到，则返回0。 例如：strpos('abcdefg', 'a') -->1
strpos(string, substring, instance)	bigint	返回string中substring的从左向右第 instance 次出现的位置。instance >0 , 位置从1开始。如果未找到，则返回0。 例如：strpos('abcaefg', 'ab',2) -->0
strrpos(string, substring)	bigint	返回string中substring的从右向左第一次出现的位置。位置从1开始。如果未找到，则返回0 例如：strpos('abcdefg', 'a') -->7
strrpos(string, substring, instance)	bigint	返回string中substring的从右向左第 instance 次出现的位置。instance >0 , 位置从1开始。如果未找到，则返回0。 例如：strpos('abcaefg', 'a',2) -->0
to_utf8(string)	varbinary	将字符串转为utf8编码的二进制数

from_utf8(binary)	varchar	将二进制数转为utf8编码的字符串，无效字符用Unicode字符U+FFFD替换
from_utf8(binary, replace)	varchar	将二进制数转为utf8编码的字符串，无效字符用replace替换

日期时间函数

函数名称	返回值	函数功能描述
current_date	date	返回当前日期
current_time	time	返回当前带时区的时间
current_timestamp	timestamp	返回当前带时区的时间戳
current_timezone()	varchar	返回当前时区
date(x)	date	返回当前日期
last_day_of_month(x)	date	返回本月的最后一天
from_unixtime(unixtime)	timestamp	返回 unix 时间戳 例如：from_unixtime(1475996660) -->2016-10-09 15:04:20.000
to_unixtime(timestamp)	double	返回 timestamp 的 unix时间戳
from_unixtime(unixtime, string)	timestamp	返回 unixtime的时间戳，string指定时区 例如：from_unixtime(1617256617,'Asia/Shanghai') -->2021-04-01 13:56:57.000 Asia/Shanghai
from_iso8601_timestamp(string)	timestamp	以 iso8601 格式将 string 转为时间戳
from_iso8601_date(string)	date	以 iso8601 格式将 string 转为日期
localtime	time	返回当前时间
localtimestamp	timestamp	返回当前时间的时间戳
now()	timestamp	返回当前带时区的时间戳
to_iso8601(x)	varchar	返回 x 的iso8601 格式的字符串，x 可以为日期、时间戳或带时区的时间戳
date_trunc(unit, x)	与输入一致	返回 以unit为单位截取x的时间，unit 为 second、minute、hour、day、week、month、quarter、year.

		例如：date_trunc('hour', TIMESTAMP '2020-03-17 02:09:30') --> '2020-03-17 02:00:00'
date_add(unit, value, timestamp)	与输入一致	返回 加减相应单位的间隔时间 算出新的时间戳, value >0 表示加，value < 0 表示减. unit 为 millisecond、second、minute、hour、day、week、month、quarter、year. 例如：date_add('hour', 2, cast ('2023-07-20 20:22:22.022' as timestamp) -->'2023-07-20 22:22:22.022'
date_diff(unit, timestamp1, timestamp2)	bigint	返回两个时间戳相差（指定单位的）时间。unit 为 millisecond、second、minute、hour、day、week、month、quarter、year. 例如：date_diff('hour', cast ('2023-07-01 22:22:22' as timestamp) , cast ('2023-07-03 22:22:22' as timestamp) -->48
extract(field FROM x)	bigint	返回根据field 提取 x的时间，field可以为：year、quarter、month、week、day、dow、doy、yow、hour、minute、second、timezone_hour、timezone_minute、day_of_week、day_of_year、day_of_month 例如：extract(year from current_date) 2023
day(x)	bigint	返回 x 是月的第几天
day_of_month(x)	bigint	返回 x 是月的第几天
day_of_week(x)	bigint	返回 x 是一周的第几天，范围为1-7
day_of_year(x)	bigint	返回 x 是年的第几天
dow(x)	bigint	返回 x 是周的第几天。同 day_of_week(x)
doy(x)	bigint	返回 x 是年的第几天。同 day_of_year(x)
hour(x)	bigint	返回 x 是一天中的第几小时，范围 0-23
millisecond(x)	bigint	返回 x 是一秒中的第几毫秒
minute(x)	bigint	返回 x 是一小时中的第几分钟
month(x)	bigint	返回 x 是一年中的第几个月
quarter(x)	bigint	返回 x 是一年中的第几季度
second(x)	bigint	返回 x 是一分钟中的第几秒

week(x)	bigint	返回 x 是一年中的第几周
week_of_year(x)	bigint	返回 x 是一年中的第几周
year(x)	bigint	返回 x 的年份

数组函数

函数名称	返回值	函数功能描述
array_distinct(x)	array	从数组中删除重复的值x。
array_intersect(x, y)	array	返回 x 和 y 的交集中的不重复元素。
array_union(x, y)	array	返回 x 和 y 的并集中的不重复元素。
array_except(x, y)	array	返回属于 x 但不属于 y 的不重复元素。（差集）
array_join(x, delimiter, null_replacement)	varchar	使用delimiter连接 x 数组元素并用null_replacement来填充数组里面的空值。null_replacement为可选字符
array_max(x)	x	返回输入数组的最大值
array_min(x)	x	返回输入数组的最小值
array_position(x, element)	bigint	返回数组 x 中第一次出现 element 的位置（数字）（如果未找到，则返回0）
array_remove(x, element)	array	删除数组 x 中和element相同的所有元素
array_sort(x)	array	返回 x 的排序结果。x的元素必须是可排序的。空元素将放置在返回数组的末尾
array_sort(array(T), function(T, T, int))	array(T)	返回 array 根据比较函数function排序的结果 例如：array_sort(ARRAY [3, 2, 5, 1, 2], (x, y) -> IF(x < y, 1, IF(x = y, 0, -1))) -->[5, 3, 2, 2, 1]
arrays_overlap(x, y)	布尔值	判断数组 x 和 y 是否有任何共同的非空元素
cardinality(x)	bigint	返回数组的基数（大小）
concat(array1，array2，...，arrayN)	array	连接数组array1，array2，...，arrayN
contains(x, element)	布尔值	判断数组 x 是否包含 element
element_at（array(E), index）	E	返回 array数组中 index 位置的元素。如果index> 0，从左向右计数，index < 0, 从右向左计数

filter(array(T), funciton(T, boolean))	array(T)	返回用函数function返回true的元素构造的新数组 例如：filter(array[5, -6, NULL, 7], x -> x > 0); --> [5, 7]
repeat(element， count)	array	重复元素element的count次数
reverse(x)	array	返回一个具有相反数组顺序的数组x。
sequence（start, stop, n）	array（bigint ）	从start 根据步长 n 到 stop 生成一个整数序列，n 为可选项，不填为1,。如果start小于等于stop 每次自增1，否则自增 -1
shuffle(x)	array	打乱数组元素
slice(x, start, length)	array	返回将 x 数组从start位置开始，切片成长度为length的数组
transform(array(T), function(T, U))	array(U)	返回 array数组中的每个元素T经过function后生成U组成的数组 例如：transform(array [5, 6], x -> x + 1); -- [6, 7]

JSON函数

函数名称	返回值	函数功能描述
is_json_scalar(json)	boolean	判断 json是否为 json数字或 json字符串或 true 或 false 或 null 例如： is_json_scalar('[1, 2, 3]') -->false
json_array_contains(json, value)	boolean	判断 value 是否存在于 json（一个包含 json数组的字符串）中。 例如：json_array_contains('[1, 2, 3]', 2) -->true
json_array_length(json)	bigint	返回 json（一个包含 json数组的字符串）的数组长度。 例如：json_array_length('[1, 2, 3]') --> 3
json_extract(json, json_path)	json	在 json（一个包含 json的字符串）上计算类似 JSONPath 的表达式 json_path，并将结果作为 json返回。 例如：json_extract('{"log":{"file":{"path":"/etc/nginx/logs/access.log"},"offset":19991212}}', '$.log.file.path')--> "/etc/nginx/logs/access.log"
json_extract_scalar(json, json_path)	varchar	与 json_extract() 类似，但将结果值作为字符串返回,而不是 json串。json_path 所引用的值必须是布尔值、数字或字符串。 例如：json_extract_scalar('{"log":{"file":{"path":"/etc/nginx/logs/access.log"},"offset":19991212}}', '$.log.file.path') -->/etc/nginx/logs/access.log
json_format(json)	varchar	返回从输入的 json值序列化的 json文本。这是 json_parse() 的

		逆函数。 例如：json_format(JSON '[1, 2, 3]') -->[1,2,3]
json_parse(string)	json	返回从输入的 json文本反序列化的 json值。这是 json_format() 的逆函数。 例如：json_parse('[1, 2, 3]') -->[1,2,3]
json_size(json, json_path)	bigint	与 json_extract() 类似，但返回值的大小。对于对象或数组，大小是成员的数量，标量值的大小是零。 例如：json_size('{"x": [1, 2, 3]}', '$.x') -->3

聚合函数

函数名称	返回值	函数功能描述
arbitrary(x)	与输入一致	如果存在，则返回x的任意非空值
array_agg(x)	array[x]	返回由输入x元素创建的数组
avg(x)	double	返回所有输入值的平均值(算术平均值)
bool_and(boolean) | every(boolean)	boolean	输入的每个值都为true,则返回true,否则返回false
bool_or(boolean)	boolean	输入的值只要有一个为true,则返回true,否则返回false
checksum(x)	varbinary	返回输入值的校验和
count(*)	bigint	返回行数
count(x)	bigint	返回非空输入值的数量
count_if(x)	bigint	返回输入值中为true值的数量，等价于count(CASE WHEN x THEN 1 END)
geometric_mean(x)	double	返回几何平均值
max_by(x, y)	与输入 x 一致	返回 x 中与 y 的最大值相关连的值
max_by(x, y, n)	array[x]	按照y降序排列，返回 x 中 n 个 与 y 中 n 个最小值相关联的值
min_by(x, y)	与输入 x 一致	返回 x 中与 y 的最小值相关连的值
min_by(x, y, n)	array[x]	按照 y 升序排列，返回 x 中 n 个 与 y 中 n 个最小值相关联的值
max(x)	与输入一致	返回输入中的最大值

max(x, n)	与输入 x 一致	返回输入中最大的 n 个值
min(x)	与输入一致	返回输入中的最小值
min(x, n)	与输入 x 一致	返回输入中最小的 n 个值
set_agg(x)	与输入一致	返回一个不重复元素的数组
set_union(array(T))	array(T)	返回输入的每个数组中包含的所有不同值的数组 例如：select set_union(elements) FROM (VALUES ARRAY[1, 2, 3], ARRAY[2, 3, 4]) AS t(elements) -->ARRAY[1, 2, 3, 4]
sum(x)	与输入一致	返回和
bitwise_and_agg(x)	bigint	返回所有位的与
bitwise_or_agg(x)	bigint	返回所有位的或
histogram(x)		返回一个包含每个输入值出现次数的map
map_agg(key, value)	map(K, V)	返回key-value组成的map
map_union(x(K, V))	map(K, V)	返回所有输入映射的并集，如果一个key对应多个value，则结果集中key对应的value值为多个value中的任意一个value值
map_union_sum(x(K, V))	map(K, V)	返回相同key求和后的并集map,如果key对应的value为空，则计为0
multimap_agg(key, value)		返回 key-value组成的map,key可以对应多个值

窗口函数

函数名称	返回值	函数功能描述
row_number()	bigint	为每一行分配一个唯一的连续编号
rank()	bigint	计算某个值在一组值中的排名。如果出现排名相等，则在排名序列中留出空位。
percent_rank()	double	计算某个值在一组值中的百分比排名。返回值以 0 到 1 之间的小数表示。
dense_rank()	bigint	计算某个值在一组值中的排名，如果出现排名相等，与函数rank不同，dense_rank不会在排名序列中产生空位。
cume_dist()	bigint	计算某个值在分区中相对于所有值的位置。

ntile(n)	bigint	将窗口分区的行划分为 n 个桶，返回行所在的桶数，范围从 1 到 n
first_value(x)	与输入一致	返回分区中某列第一条数据的值
last_value(x)	与输入一致	返回分区中某列最后一条数据的值
nth_value(x, n)	与输入一致	返回距窗口头第 n 行的值。n 从 1 开始。 如果ignoreNulls=true，查找第 n 行时将跳过 null。否则，每一行都计入 n。 如果不存在这样的第 n 行（例如，当 n 为10时，窗口大小小于10），则返回null。第一个参数为列名，第二个参数为之前第 n 行。
lead(x[, n[, default_value]])	与输入一致	返回窗口中当前行向下第 n 行的值。n 默认值为 1，default 默认值为 null。 如果第 n 行的 值为 null，则返回 null。 如果不存在这样的偏移行（例如，当偏移量为1时，窗口的最后一行没有任何向下行），则返回 default 。第一个参数为列名，第二个参数为之前第 n 行，第三个参数为默认值。
lag(x[, n[, default_value]])	与输入一致	返回窗口中当前行向上第 n 行的值。n 默认值为 1，default 默认值为 null。 如果第 n 行的 值为 null，则返回 null。 如果不存在这样的偏移行（例如，当偏移量为1时，窗口的第一行没有任何向上行），则返回 default。第一个参数为列名，第二个参数为之前第 n 行，第三个参数为默认值。

网址函数

函数名称	返回值	函数功能描述
url_extract_host(url)	varchar	返回 ur 的主机
url_extract_parameter(url, name)	varchar	返回 url 中 name 的第一个值，按照RFC 1866#section-8.2.1查找
url_extract_path(url)	varchar	返回 url 的路径
url_extract_port(url)	bigint	返回 url 的端口
url_extract_protocol(url)	varchar	返回 url 的协议
url_extract_query(url)	varchar	返回 url 的查询字符串
url_encode(value)	varchar	对 value 进行 encode 编码

url_decode(value)	varchar	对 encode 编码 的url 解码

其他函数

函数名称	返回值	函数功能描述
uuid ( )	uuid	返回一个随机生成的UUID
cast(value AS type)	type	将 value 强制转为 type 类型
try_cast(value AS type)	type	将 value 转成 type 类型，如果失败返回 null
typeof(expr)	varchar	返回 expr 的类型名称 例如： typeof(cos(2) + 1.5) -->double
regexp_extract(string, pattern)	varchar	返回 string 中正则表达式 pattern 匹配的第一个子字符串 例如：regexp_extract('1a 2b 14m', '\\d+') ->1
regexp_replace(string, pattern)	varchar	从 string 中移除正则表达式 pattern 匹配的子字符串 例如：regexp_replace('1a 2b 14m', '\\d+[ab] ') --> '14m'
regexp_split(string, pattern)		使用正则表达式 pattern 拆分字符串并返回一个数组，保留后面的空字符串 例如：regexp_split('1a 2b 14m', '\\s*[a-z]+\\s*'); -- [1, 2, 14,]
regexp_like(string, pattern)	boolean	判断 string 中是否有 pattern 匹配的字符串

Hive 函数对照

最近更新时间：2024-08-07 17:34:57

DLC 对 Hive 函数有完善的支持，您可以轻松地从 Hive 升级到数据湖计算 DLC，使用更强大的数据湖特性。
DLC 统一函数与 Hive 函数存在部分细微差别，具体函数对照表如下。
Hive 函数详情可以参考 Apache Hive 官网LanguageManual UDF。

数学函数

Hive 数学函数可参考 Apache Hive 官网 MathematicalFunctions。
DLC 数学函数请参见腾讯云官网文档 数学函数。

Hive3.1函数名称	函数功能描述	DLC 函数名称	差异表述	使用参考
round	四舍五入取整	round	无差异	select round(1.23);
round	保留指定位数小数四舍五入	round	无差异	select round(1.234,2);
bround	HALF_EVEN 模式下的四舍五入	bround	DLC 函数中 bround 需要传入2个参数，其中第2个参数指定保留的小数位。将第二个参数置为0，效果等同 hive 中的 bround(double a)	select bround(1.237,0);
bround	HALF_EVEN 模式下的四舍五入，保留指定的小数位	bround	无差异	select bround(1.237,2);
floor	向下取整	floor	无差异	select floor(1.23);
ceil, ceiling	向上取整	ceil, ceiling	无差异	select ceil(1.93); select ceiling(1.13);
rand	返回一个0到1范围内的随机数。如果指定种子 seed，则会等到	rand	无差异	select rand(); select rand(5);

	一个稳定的随机数序列			
exp	e 的 a 次方	exp	无差异	select exp(1); select exp(2.4);
ln	自然对数函数	ln	无差异	select ln(2.4);
log10	以10为底的对数函数	log10	无差异	select log10(2.4);
log2	以2为底的对数函数	log2	无差异	select log2(2.4);
log	对数函数	log	无差异	select log(2,4);
pow, power	次方函数	pow, power	无差异	select pow(2,4);
sqrt	开方函数	sqrt	无差异	select sqrt(4);
bin	二进制函数	bin	无差异	select bin(4);
hex	十六进制函数	hex	无差异	select hex(10);
unhex	十六进制转string	unhex	DLC Spark 函数中需要对结果进行 decode	select decode(unhex('314 'UTF-8');
conv	进制转化	conv	无差异	select conv(2,10,2);
abs	取绝对值	abs	无差异	select abs(-1);
pmod	取模	pmod	无差异	select pmod(5,3);
sin	三角函数 sin，a 为弧度	sin	无差异	select sin(3.14);
asin	三角函数 arc sin，a 为弧度	asin	无差异	select asin(0.5);
cos	三角函数 cos，a 为弧度	cos	无差异	select cos(3.14);
acos	三角函数arc cos，a 为弧度	acos	无差异	select acos(1);
tan	三角函数 tan，a 为弧度	tan	无差异	select tan(3.14);
atan	三角函数 arc tan，a	atan	无差异	select atan(1);

	为弧度			
degrees	弧度转角度	degrees	无差异	select degrees(3.14);
radians	角度转弧度	radians	无差异	select radians(180);
positive	返回 a	positive	无差异	select positive(-1);
negative	返回 -a	negative	无差异	select negative(1);
sign	如果 a 为正数，则返回1.0；如果 a 为负数，则返回-1.0；如果 a 为0，则返回0.0	sign	无差异	select sign(1.12);
e	返回自然常数 e	e	无差异	select e();
pi	返回圆周率 pi	pi	无差异	select pi();
factorial	阶乘	factorial	无差异	select factorial(5);
cbrt	立方根	cbrt	无差异	select cbrt(27);
shiftleft	左移位	shiftleft	无差异	select shiftleft(3,1);
shiftright	右移位	shiftright	无差异	select shiftright(3,1);
shiftrightunsigned	无符号右移位	shiftrightunsigned	无差异	select shiftrightunsigned(
greatest	返回最大值	greatest	当参数中有 null 值时，DLC 函数对 null 值不做计算，返回除 null 之外最大值，Hive 函数会返回 null	select greatest(1,2,3.3);
least	返回最小值	least	当参数中有 null 值时，DLC 函数对 null 值不做计算，返回除 null 之外最小值，Hive 函数会返回 null	select least(1,2,3.3);

width_bucket	分桶值，按 min_value/max_value 创建 num_buckets+1 个相同大小的桶，返回当前值所在的桶编号	width_bucket	无差异	select width_bucket(10,1

集合函数

DLC 集合函数与 Hive 集合函数无差异。

Hive 集合函数可参考 Apache Hive 官网 CollectionFunctions。

DLC 集合函数请参见腾讯云官网文档 集合函数。

Hive3.1函数名称	函数功能描述	DLC 函数名称	差异表述	使用参考
size	返回 map / array 的大小	size	无差异	select size(str_to_map('a:1,b:2'));
map_keys	返回 map 的 key 值列表	map_keys	无差异	select map_keys(str_to_map('a:1,b:2'));
map_values	返回 map 的 value 值列表	map_values	无差异	select map_values(str_to_map('a:1,b:2'));
array_contains	array 中是否包含 value	array_contains	无差异	select array_contains(split('a,b',','),'a');
sort_array	升序排列 array 中的元素	sort_array	无差异	select sort_array(split('1,3,2',','));

类型转化函数

Hive3.1函数名称	函数功能描述	DLC 函数名称	差异表述	使用参考
binary	将输入的参数转换为二进	binary	Hive3.1入参仅支	select binary('testString')

	制数组		持 string 和 binary；DLC 支持 string、int、long 和 binary入参 Hive3.2的出参会转换为 string展示，而 DLC 的出参是二进制数组，不会转 string	select binary(1)（HIVE 不支持） select binary(inputCol) from inputTable（HIVE 不支持）
cast	将输入的参数转强制转换为指定 type 类型。 如果给定的表达式或值无法强制转换为指定类型则会报错，例如字符串转 long 错误：Cast function cannot convert value of type VARCHAR(65536) to type LONG	cast	Hive3.1对强制转换失败的会返回 NULL，而 DLC 会报错	select cast('10' as int)select cast(inputCol as int) from inputTable

时间函数

Hive3.1函数名称	函数功能描述	DLC 对应函数名称	差异表述	使用参考
from_unixtime	将数字化的 Unix 时间（表示从 1970-01-01 00:00:00 UTC 开始计数的秒数）转换为指定格式的字符串函数。 默认输出的时间格式为 yyyy-MM-dd HH:mm:ss，时区为当前系统定义的时区。	from_unixtime	无差异	select from_unixtime select from_u 'yyyyMMdd H
unix_timestamp	指定一个时间，或默认为当前时间，返回从1970-01-01 00:00:00 UTC 开始到指定时间的秒数。	unix_timestamp	无差异	select unix_ti select unix_timestar 12 00:00:00') select unix_timestar 12', 'yyyy-MM

to_date	指定一个时间，获取这个时间所在的日期。 例如2023-04-12 13:14:20所在的日期为2023-04-12。	to_date	HIVE 的 to_date 函数只能按指定格式 yyyy-MM-dd HH:mm:ss 输入，非指定格式的统一返回 NULL。 DLC 可以附加一个参数用于指定数据格式，从而使得输入的时间格式可以更灵活。	select to_date 19:41:23') select to_date 19:41', 'yyyy-l HH:mm')（HI select to_date 19:41', 'yyyyM HH:mm')（HI
year	指定一个时间，获取这个时间所在的年。 例如2023-04-12 13:14:20所在的天为2023	year	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。	select year('2 19:41:25') select year('2 （HIVE 不支
quarter	指定一个时间，获取这个时间所在的季度。 例如2023-04-12 13:14:20所在的季度为2	quarter	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。	select quarter 19:41') select quarter （HIVE 不支
month	指定一个时间，获取这个时间所在的月。 例如2023-04-12 13:14:20所在的月为4。	month	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持yyyy或yyyy-MM格式。 当时间格式月份超过12时，HIVE 会计算和最后一个月的差值并取余，而 DLC 返回 NULL。	select month(19:41') select month(（HIVE 不支
day/dayofmonth	指定一个时间，获取这个时间所在的天。 例如2023-04-12 13:14:20所在的月为当月第12天。	day/dayofmonth	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。 当时间格式中的天数超过了当月的天数，HIVE	select day('20 不支持） select dayofm 04-12 19:41') select day('20 19:41')

			会对当月天数取余，而 DLC 会返回 NULL。	
hour	指定一个时间，获取这个时间所在的小时。 例如2023-04-12 13:14:20所在的月为第13个小时。	hour	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。 当时间格式中的小时超过了23时，HIVE 会对23取余，而 DLC 会返回 NULL	select hour('2 19:41') select hour('2 （HIVE 不支
minute	指定一个时间，获取这个时间所在的分钟。 例如2023-04-12 13:14:20所在的月为第14分钟。	minute	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。 当时间格式中的分钟超过了59时，HIVE 会对60取余，而 DLC 会返回 NULL	select minute 00:41') select minute （HIVE 不支
second	指定一个时间，获取这个时间所在的秒。 例如2023-04-12 13:14:20所在的月为第20秒。	second	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持yyyy或yyyy-MM格式。 当时间格式中的秒超过了59时，HIVE 会对60取余，而 DLC 会返回 NULL	select seconc 00:41:24') select seconc （HIVE 不支
weekofyear	指定一个时间，返回指定时间在全年的第几周。 例如2023-04-12 13:14:20是2023年的第15周。 需要额外注意的是，当年的第一个周一，视为第一周的开始，如果日期	weekofyear	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。	select weekof 04-12 00:41:2 select weekof 04')（HIVE 不

	时间在当年的第一个周一之前，会被认为是上一年的第52周。例如2023-01-01是周日，则weekofyear('2023-01-01')返回的是52，即上一年的第52周，而weekofyear('2023-01-02')则返回1，即当年的第1周			
extract	extract(field from source)从输入的source 时间或时间间隔中抽取指定的 field 字段	extract	HIVE 和 DLC 版本目前不支持调整精度。 当 DATE 或 TIMESTAMP 标识符显式指定时，如果日期或时间格式不匹配，DLC 会报错，HIVE 会返回0。 HIVE 在计算 INTERNAL 的间隔时，如果 field 不在 source 中，会自动计算取余，DLC 则会报错。例如从间隔24个月中抽取年的字段，HIVE 会返回2年，DLC 则会报错字段不存在。	select extract DATE '2023-(select extract TIMESTAMP 00:41:25') select extract interval '23-1' MONTH) select extract interval '23' M select extract interval '23' MONTH) （D select extract interval '23 13 DAY TO SEC
datediff	计算从 startDate 到 endDate 所差的天数。如果 startDate 比 endDate 更迟，返回的是负数。 任意一个输入为 NULL，则返回 NULL	datediff	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持yyyy 或 yyyy-MM 格式。	select datedif '2022-10'）（H select datedif 13', '2022-04-
from_utc_timestamp	给定一个 UTC 标准时间 timestamp，将其转换为给定时区	from_utc_timestamp	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。	select from_utc_tim 04', 'Asia/Sec select from_utc_tim

	timeZone 的时间。 任意一个输入为 NULL，则返回 NULL。		DLC 则可以支持 yyyy 或 yyyy-MM 格式。 Hive 返回的是时间格式，字符串表示为 yyyy-MM-dd HH:mm:ss。 而 DLC 返回的是 UTC 标准表示，例如上海时区为 yyyy-MM-ddTHH:mm:ss+08:00。	04-12 15:00:0 'Asia/Shangh
to_utc_timestamp	给定一个时区 timeZone 的时间 timestamp，转换为 UTC 标准时间 timestamp。 任意一个输入为 NULL，则返回 NULL。	to_utc_timestamp	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。 Hive 返回的是时间格式，字符串表示为 yyyy-MM-dd HH:mm:ss。 而 DLC 返回的是 UTC 标准表示，例如上海时区为 yyyy-MM-ddTHH:mm:ss+08:00。	select to_utc_timest 04-12 15:00:0 'Asia/Shangh select to_utc_timest 04', 'Asia/Sha （HIVE 不支
date_add	日期加法，给定 startDate 日期，返回加上 numDays 天后的日期。 numDays 可以是负数。 任意一个输入为 NULL，则返回 NULL。	date_add	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 可以支持 yyyy 或 yyyy-MM 格式。	select date_a 12 15:00:00', select date_a -1)（HIVE 不
date_sub	日期减法，给定 startDate 日期，返回减去 numDays 天后的日期。 numDays 可以是负数。 任意一个输入为 NULL，则返回	date_sub	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。	select date_s 12 15:00:00', select date_s -1)（HIVE 不

	NULL。			
current_date	获取当前日期	current_date	无差异	select current
current_timestamp	获取当前时间	current_timestamp	无差异	select current
add_months	计算指定的日期（或时间）加上指定月数后的日期。输入为 NULL，则返回 NULL。	add_months	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。	select add_m 04-12 15:00:0 select add_m 04', -1)（HIV
last_day	计算指定的日期所在月的最后一天。输入为 NULL，则返回 NULL。	last_day	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。	select last_da 12 15:00:00') select last_da （HIVE 不支
next_day	计算指定日期后第一个 day_of_week 的日期，day_of_week 表示周几，可选值从周一到周日。 任意一个输入为 NULL，则返回 NULL。	next_day	HIVE 的输入不支持 yyyy 或 yyyy-MM格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。	select next_d 12 15:00:00') select next_d （HIVE 不支
trunc	计算指定日期按 format（例如年、季度、月、周等）缩短后的第一天的日期，例如所在季度的第一天，所在月份的第一天，所在周的第一天等。	trunc	HIVE 的输入不支持 yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。 HIVE 不支持 WEEK 周。	select trunc('2 15:00:00', 'M select trunc('2 'YEAR')（HIV
-	计算指定日期按 format（例如年、季度、月、周等）缩短后的第一天的时间，例如所在季度的第一天的零点。和 trunc 方法的不同在于	date_trunc	HIVE 不支持该函数，可参考 trunc 函数。	select date_tr '2023-04-12 1 select date_trunc('S '2023-04-12 1

	date_trunc 返回的是时间。			
months_between	计算 date1到date2相差的月数。如果date1<date2，则返回负数。 需要注意的是，计算时，每个月是按31天估算的，即分母为31。分子则为两个时间实际相差的精确到毫秒级别的时间差。最终计算的结果为小数。 任意一个入参为NULL 时返回NULL。	months_between	HIVE 的输入不支持yyyy 或 yyyy-MM 格式，其他时间或日期格式都支持。 DLC 则可以支持 yyyy 或 yyyy-MM 格式。	select months_betw 12 15:00:00', 15:00:00') select months_betw 04', '2023-04- （HIVE 不支
date_format	按照指定的格式化模板 fmt（参考Datetime Patterns for Formatting and Parsing），将输入的日期格式化。 任意一个入参为NULL 时返回NULL。	date_format	当 date 时间或日期格式不对，或者 fmt 模板格式不对时，HIVE 返回NULL，而 DLC 会报错。	select date_fo 04-12 15:00:0 select date_fo 04-12', 'yyyy- HH:mm')

条件函数

Hive3.1函数名称	函数功能描述	DLC 函数名称	差异表述	使用参考
if	当条件为真时返回valueTrue，否则返回 valueFalseOrNull	if	判断条件为 NULL 时，HIVE 认为判断条件为假，而 DLC 会报错	select if(1<2, 12, 'false')
isnull	如果 a 为 NULL，则返回 true，否则返回false	isnull	无差异	select isnull('false')

isnotnull	如果 a 不是 NULL，则返回 true，否则返回 false	isnotnull	无差异	select isnotnull('false')
nvl	如果值为 NULL，则返回第二个参数	nvl	无差异	select nvl(NULL, 'false') select nvl(1, 'false')
coalesce	返回第一个非 NULL 的参数	coalesce	无差异	select coalesce(NULL, 'false')
case/when	匹配判断	case/when	匹配选择的类型不相同时，HIVE 仅支持整型和字符串类型的隐式转换，而DLC 还可以支持 boolean 类型等的隐式转换。	select CASE 'c' WHEN 'a' THEN 1 WHEN 'b' THEN 2 ELSE 0 END; select CASE WHEN 1 > 0 THEN 1 WHEN 2 > 0 THEN 2.0 ELSE 1.2 END;

nullif	如果参数 a=b，则返回 null，否则返回 a	nullif	HIVE 不支持第一个参数为 NULL，而 DLC 可以支持任一参数为 NULL	select nullif(1, 'test');
				select nullif(NULL, 'test');
				（HIVE不支持）

assert_true	如果“条件”不为真，则抛出异常，否则返回 null	assert_true	无差异	select assert_true(1>0)

字符串函数

Hive3.1函数名称	函数功能描述	DLC 对应函数名称	差异表述	使用参考
ascii	返回 str 的第一个字符串的数值	ascii	无差异	select ascii('222');
base64	将参数从二进制转换为base64字符串	base64	无差异	select base64('tencen
character_length	返回 str 中包含的 UTF-8字符数	character_length	无差异	select char_length(bir
chr	返回具有等效于 A 的二进制字符	chr	无差异	select chr(65);
concat	返回按顺序串联作为参数传递	concat	无差异	select concat('Spark',

	的字符串或字节所产生的字符串或字节			
context_ngrams	给定一个 contextual N-grams，从一组标记化的句子中返回前 k 个上下文 N-gram	context_ngrams	无差异	-
concat_ws	返回由 sep 分隔的字符串	concat_ws	无差异	select concat_ws(' ', 't
decode	使用提供的字符集（“US-ASCII”、“ISO-8859-1”、“UTF-8”、“UTF-16BE”、“UTF-16LE”、“UTF-16”之一）将第一个参数解码为字符串。如果任一参数为空，结果也将为空。	decode	无差异	select decode(encode 8');
elt	返回索引号的字符串。例如，elt(2,'hello','world')返回'world'。如果 N 小于1或大于参数，则返回NULL。	elt	无差异	select elt(1, 'scala', 'ja
encode	使用提供的字符集（“US-ASCII”、“ISO-8859-1”、“UTF-8”、“UTF-16BE”、“UTF-16LE”、“UTF-16”之一）将第一个参数编码为二进制。如果任一参数为 Null，结果也将为 Null	encode	无差异	select encode('abc', 'u
field	field(val T,val1 T,val2 T,val3 T,...) 该函数返回在参数列表 val1,val2,val3,...中 val 的索引位置，如果未找到则返回0。例如，field('world','say','hello','world')返回3。 该函数支持所有基本数据类型，参数使用 str.equals(x)进行比较。 如果 val 为 NULL，则返回值为0。	field	DLC 函数中，仅 presto 引擎支持	select field('world', 'sa
find_in_set	返回逗号分隔列表 atr_array 中给定字符串 str 的索引（从1开始计数）。如果未找到字	find_in_set	无差异	select find_in_set('ab'

	符串或 str 包含逗号，则返回 0。			
format_number	将输入的参数格式化为'#,###,###.##'，四舍五入到 D 小数位，并以字符串形式返回结果。	format_number	无差异	select format_number
get_json_object	提取 json 对象	get_json_object	无差异	select get_json_objec
in_file	in_file(string str, string filename) 如果字符串 str 在文件 filename 中作为整行出现，则返回 true。	不支持	-	-
instr	返回 str 中第一次出现 substr 的索引（从1开始计数）	instr	无差异	select instr('SparkSQ
length	返回字符串的长度	length	无差异	select length('Spark S
locate	返回位置pos之后str中substr首次出现的位置。	locate	无差异	select locate('bar', 'foo
lower	返回将B的所有字符转换为小写产生的字符串	lower	无差异	select lower('TENCEN
lpad	lpad(string str, int len, string pad) 将字符串 str 左侧填充字符 pad，使其长度达到 len。如果 str 的长度大于 len，则返回值将被截断为 len 个字符。如果填充字符 pad 为空，则返回值为 null。	lpad	DLC sparksql 引擎中 pad 为可选参数，pad 默认为空格	select lpad('hi', 5, '??'
ltrim	返回从输入字符串的开头（左侧）修剪空格产生的字符串	ltrim	无差异	select ltrim(' SparkSQ
ngrams	返回一组标记化句子中的前 k 个 N-grams。	ngrams	无差异	-
octet_length	返回字符串数据的字节长度或二进制数据的字节数。	octet_length	无差异	select octet_length('S
parse_url	从 url 中提取 path	parse_url	无差异	select parse_url('http://spark query=1', 'HOST');

printf	返回 printf 样式格式字符串中的格式化字符串	printf	无差异	select printf("Hello Wo %s", 100, "days");
quote	返回引号字符串	不支持	-	-
regexp_extract	提取str中与regexp表达式匹配并对应于regex组索引idx的第一个字符串	regexp_extract	无差异	select regexp_extract (\\\\d+)', 1);
regexp_replace	regexp_replace(string INITIAL_STRING, string PATTERN, string REPLACEMENT) 返回将 INITIAL_STRING 中与 PATTERN 中定义的 Java 正则表达式语法匹配的所有子字符串替换为 REPLACEMENT 实例后得到的字符串。	regexp_replace	DLC sparksql 引擎中还可以加入第四个自选参数 position，position 是一个正整数字面量，表示在字符串中开始搜索的位置。默认值为1。	select regexp_replace 'num');
repeat	返回将给定字符串重复 n 次的字符串	repeat	无差异	select repeat('123', 2)
replace	replace(string A, string OLD, string NEW) 返回将字符串A中所有不重叠的 OLD 出现替换为NEW后得到的字符串。	replace	sparksql 引擎中第三个参数 NEW 为可选参数，默认为空	select replace('ABCab
reverse	reverse(string A) 返回反转后的字符串。	reverse	sparksql 引擎中参数可以是 array	select reverse('Spark
rpad	rpad(string str, int len, string pad) 将字符串 str 右侧填充字符 pad，使其长度达到 len。如果 str 的长度大于 len，则返回	rpad	sparksql 引擎中 pad 为可选参数，	select rpad('hi', 5, '??'

	值将被截断为 len 个字符。如果填充字符 pad 为空，则返回值为 null。		pad 默认为空格	
rtrim	返回从输入字符串的末端（右侧）修剪空格产生的字符串	rtrim	无差异	select rtrim(' SparkSC
sentences	将输入字符串 str 拆分为一个单词数组	sentences	无差异	select sentences('Hi tl morning.');
space	返回一个由 n 个空格组成的字符串	space	无差异	select concat(space(2
split	split(string str, string pat) 围绕正则表达式 pat（pat 是一个正则表达式）拆分字符串 str。	split	DLC sparksql 引擎中有第三个可选参数 limit。limit 可以用以限制返回数组的长度。	select split('oneAtwoE
str_to_map	使用分隔符将输入参数拆分为键/值对后创建 map。	str_to_map	无差异	select str_to_map('a:1
substr	返回从 pos 开始且长度为 len 的 str 子字符串，或从 pos 开始且长度为 len 的字节数组切片	substr	无差异	select substr('Spark S
substring	返回从 pos 开始且长度为 len 的 str 子字符串，或从 pos 开始且长度为 len 的字节数组切片。	substring	无差异	select substring('Spar
substring_index	在 delim 的出现 count 之前，从 str 返回子字符串。如果 count 为正数，则返回最后定界符左侧的所有内容（从左侧计数）。如果计数为负数，则返回最终定界符右侧的所有内容（从右侧计数）。该函数在匹配 delim 时区分大小写。	substring_index	无差异	select substring_index('cloud
translate	通过将 from 字符串中的字符	translate	无差异	select translate('AaBb

	替换为 to 字符串中的相应字符来转换 input 字符串			
trim	返回从输入字符串的两端修剪空格产生的字符串	trim	DLC SparkSQL 引擎支持更复杂的修剪表达式	select trim(' SparkSQ
unbase64	将 str 从 base64字符串转换为二进制	unbase64	无差异	select unbase64('U3E
upper	返回所有字符都改为大写的 str	upper	无差异	select upper('tencent'
ucase	返回所有字符都改为大写的 str	ucase	无差异	select ucase('SparkS
initcap	每个单词的第一个字母都改为大写，所有其他字母均为小写。	initcap	无差异	select initcap('sPark s
levenshtein(string A, string B)	返回两个给定字符串之间的 Levenshtein 距离	levenshtein	无差异	select levenshtein('kit
soundex(string A)	返回字符串的 Soundex 编码	soundex	无差异	select soundex('Miller

数据屏蔽函数

Hive3.1函数名称	函数功能描述	DLC 对应函数名称	差异表述	使用参考
mask	mask(string str[, string upper[, string lower[, string number]]]) 返回 str 的掩码版本。	不支持	-	-
mask_first_n	mask_first_n(string str[, int n]) 返回 str 的掩码版本，其中前 n 个值被掩码替换。	不支持	-	-
mask_last_n	mask_last_n(string str[, int n]) 返回 str 的掩码版本，其中最后 n 个值被掩码替换。	不支持	-	-

mask_show_first_n	mask_show_first_n(string str[, int n]) 返回 str 的掩码版本，其中显示前 n 个字符未掩码替换。	不支持	-	-
mask_show_last_n	mask_show_last_n(string str[, int n]) 返回 str 的掩码版本，其中显示前 n 个字符未掩码替换。	不支持	-	-
mask_hash	mask_hash(string|char|varchar str) 基于 str 返回一个哈希值。	不支持	-	-

其他函数

Hive3.1函数名称	函数功能描述	DLC 对应函数名称	差异表述	使用参考
java_method	reflect 的同义词	不支持	-	-
reflect	使用反射匹配参数签名来调用 Java 方法	不支持	-	-
hash	返回参数的散列值	hash	不同引擎计算方式不一致，可能得到不同结果	select hash('tencent', array(1
current_user	返回当前用户	current_user	无差异	select current_user();
logged_in_user	从会话状态返回当前用户名。这是连接到 Hive 时提供的用户名。	不支持	-	-
current_database	返回当前数据库名称	不支持	-	-
md5	以十六进制字符串形式返回 MD5 128位校验和	md5	无差异	select md5('tencent');
sha1	以十六进制字符串形式返回输入参数的 sha1哈希值	sha1	无差异	select sha1('tencent');
sha	以十六进制字符串形式返回输入参数的 sha1哈希值	sha	无差异	select sha('tencent');

版权所有：腾讯云计算（北京）有限责任公司

crc32	使用 CRC32算法计算表达式的循环冗余校验值	crc32	无差异	select crc32('tencent');
sha2	以十六进制字符串形式返回 expr 的 SHA-2族的校验和。支持 SHA-224、SHA-256、SHA-384和 SHA-512。位长度0等于256	sha2	无差异	select sha2('tencent', 256);
aes_encrypt	aes_encrypt(input string/binary, key string/binary) 使用 aes 加密	aes_encrypt	DLC 中 sparksql 引擎不支持该函数	select hex(aes_encrypt('tenc
aes_decrypt	aes_decrypt(input binary, key string/binary) 使用 AES 解密	aes_decrypt	DLC 中 sparksql 引擎不支持该函数	select aes_decrypt(unhex('B99B99 '0000111122223333');
version	version() 返回引擎版本	version	sparksql 返回 spark 版本, presto 返回 hive版本	select version();
surrogate_key	surrogate_key([write_id_bits, task_id_bits]) 在向表中插入数据时自动生成数字 ID。只能用作 ACID 或仅插入表的默认值。	不支持	-	-

标准 Spark 语法概览

最近更新时间：2024-08-07 17:35:24

标准 Spark 引擎的内核基于 Spark 3.2 版本自研，兼容 Spark 原生语法和行为。详细的语法请参考 Spark 语法文档。

DDL 语法

数据库相关语法

用途	语法
新建数据库	CREATE DATABASE
展示在该元数据中定义的所有数据库	SHOW DATABASES
查看数据库属性	DESCRIBE DATABASE
数据库属性变更	ALTER DATABASE SET DBPROPERTIES
删除数据库	DROP DATABASE

数据表相关语法

用途	语法
新建数据表	CREATE TABLE
查询数据表建表信息	SHOW CREATE TABLE
查询表属性	SHOW TBLPROPERTIES
查询数据库中的所有表	SHOW TABLES
查看数据表列信息及元数据信息	DESCRIBE TABLE
向数据表添加列	ALTER TABLE ADD COLUMNS
对数据表新增列	ALTER TABLE ADD COLUMN AFTER/FIRST
变更字段名称	ALTER TABLE ... RENAME COLUMN
删除数据表某个字段	ALTER TABLE DROP COLUMN

版权所有：腾讯云计算（北京）有限责任公司

对数据表新增分区信息	ALTER TABLE ADD PARTATION
列出表分区	SHOW PARTITIONS
删除数据表分区信息	ALTER TABLE DROP PARTITION
对 Iceberg 表添加分区字段	ALTER TABLE ADD PARTITION FIELD
删除 Iceberg 表分区字段	ALTER TABLE DROP PARTITION FIELD
数据表属性变更	ALTER TABLE SET TBLPROPERTIES
数据表存储位置变更	ALTER TABLE SET LOCATION
修改表数据的排序方式	ALTER TABLE ... WRITE ORDERED BY
修改分区表的分配策略	ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION
更新分区信息	MSCK REPAIR TABLE
删除元数据表	DROP TABLE
展示执行 sql 的逻辑或物理计划	EXPLAIN
调用表存储过程	CALL STATEMENT

视图相关语法

用途	语法
将 select 结果创建为视图	CREATE VIEW AS
查询数据库中的视图	SHOW VIEWS
查看视图的列信息	DESCRIBE VIEW
展示视图创建语句	SHOW CREATE VIEW
查看视图列信息	SHOW COLUMNS IN VIEW
修改视图名称	ALTER VIEW RENAME TO
修改视图属性	ALTER VIEW SET TBLPROPERTIES
删除视图	DROP VIEW

函数相关语法

用途	
创建函数	CREATE FUNCTION
查看创建函数语法	SHOW FUNCTION
删除函数	DROP FUNCTION

DML 语法

用途	
插入一行数据	INSERT STATEMENT
替换一行数据	INSERT OVERWRITE
行级数据更新操作，可用于替换 INSERT OVERWRITE 操作	MERGE INTO
Iceberg 表元数据查询	TABLE METADATA
将查询结果插入数据表	INSERT INTO
删除 Iceberg 表的数据	DELETE STATEMENT
更新指定行	UPDATE

DQL 语法

用途	
数据查询	SELECT STATEMENT

标准 Presto 语法概览

最近更新时间：2024-08-07 17:35:37

标准 Presto 引擎的内核基于 Presto 0.242 自研，兼容 Presto 原生语法和行为，适用于交互式查询分析。详细的语法请参考 Presto 语法文档。

用途	语法	是否支持	备注
修改函数定义	ALTER FUNCTION	否	
重命名 Schema	ALTER SCHEMA	是	
修改数据表	ALTER TABLE	是	只支持 iceberg 表。 执行时需要通过三段式指定目标表的 iceberg catalog 或者先执行 use iceberg.dbname
对数据表进行统计	ANALYZE	是	只支持 hive 表
提交当前事务	COMMIT	否	
创建函数	CREATE FUNCTION	否	
创建角色	CREATE ROLE	否	
创建 SCHEMA	CREATE SCHEMA	否	
创建数据表	CREATE TABLE	否	
使用 SELECT 建表	CREATE TABLE AS	否	
创建视图	CREATE VIEW	否	
删除 PREPARE	DEALLOCATE PREPARE	否	
删除数据	DELETE	否	
展示列信息	DESCRIBE	是	
展示 PREPARE 输入信息	DESCRIBE INPUT	否	
展示 PREPARE 输出信息	DESCRIBE OUTPUT	否	

版权所有：腾讯云计算（北京）有限责任公司

删除函数	DROP FUNCTION	否	
删除角色	DROP ROLE	否	
删除 SCHEMA	DROP SCHEMA	否	
删除数据表	DROP TABLE	否	
删除视图	DROP VIEW	否	
执行 PREPARE	EXECUTE	是	
展示执行 SQL 的逻辑或物理计划	EXPLAIN	是	
执行 SQL 并展示执行计划	EXPLAIN ANALYZE	是	
授权	GRANT	否	
将角色授权给指定对象	GRANT ROLE	否	
插入数据	INSERT	是	如果是 iceberg 表，执行时需要通过三段式指定目标表的 iceberg catalog 或者先执行 use iceberg.dbname
创建 PREPARE	PREPARE	是	
指定 SESSION 恢复默认值	RESET SESSION	是	
取消授权	REVOKE	否	
取消授予的角色	REVOKE ROLES	否	
回滚事务	ROLLBACK	否	
查询数据	SELECT	是	
设置角色	SET ROLE	否	
设置指定 SESSION 的值	SET SESSION	是	
展示 CATALOG 列表	SHOW CATALOGS	是	
展示表的列信息	SHOW COLUMNS	是	

版权所有：腾讯云计算（北京）有限责任公司

展示函数信息	SHOW CREATE FUNCTION	否	
展示建表信息	SHOW CREATE TABLE	是	
展示创建视图信息	SHOW CREATE VIEW	否	
展示函数列表	SHOW FUNCTIONS	是	
展示指定用户的权限	SHOW GRANTS	否	
展示授权的角色列表	SHOW ROLE GRANTS	否	
展示角色列表	SHOW ROLES	否	
展示 SCHEMA 列表	SHOW SCHEMAS	是	
展示 SESSION 列表	SHOW SESSION	是	
展示表的统计信息	SHOW STATS	是	
展示表列表	SHOW TABLES	是	
开始事务	START TRANSACTION	否	
删除表的所有内容	TRUNCATE	否	
更新表的内容	UPDATE	否	
指定默认 SCHEMA 或数据库	USE	是	
定义内联表	VALUES	是	

版权所有：腾讯云计算（北京）有限责任公司

保留字

最近更新时间：2024-08-07 17:35:50

注意事项

如果这几种标识符是下面的保留字作为数据库名、表名、列名、函数名、视图名等标识符的时候，需要加上默认的转义符：

```
标识符：` 反引号

例如: hour -> `hour`

create table `hour` (
id string,
`asc` int
)

hour、asc 为关键字，此时如果需要创建这样的列就必须带上反引号进行创建。
```

关键保留字

A

ALL
ALTER
AND
ANY
AS
AUTHORIZATION

B

BETWEEN
BOTH
BY

C

CALL
CASE
CAST
CHECK
CLUSTER
COLLATE
COLUMN
CONSTRAINT
CREATE
CROSS

CUBE
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR

D

DEALLOCATE
DEFAULT
DELETE
DESCRIBE
DISTINCT
DISTRIBUTE
DROP

E

ELSE
END
ESCAPE
EXCEPT
EXECUTE
EXISTS
EXPLAIN
EXTRACT

F

FETCH
FILTER
FOR
FOREIGN
FROM
FULL
FALSE

G

GRANT
GROUP

GROUPING

H

HAVING

I

IN
INNER
INSERT
INTERSECT
INTERVAL
INTO
IS

J

JOIN

L

LATERAL
LEADING
LEFT
LIKE
LIMIT
LOCALTIME
LOCALTIMESTAMP

M

MERGE
MINUS

N

NATURAL
NEW
NEXT
NORMALIZE
NOT
NULL

版权所有：腾讯云计算（北京）有限责任公司

O

OFFSET
ON
ONLY
OR
ORDER
OUTER
OVER
OVERLAPS

P

PARTITION
PATTERN
PERCENTILE_CONT
PERCENTILE_DISC
PERMUTE
PREPARE
PRIMARY

R

RANGE
RECURSIVE
REFERENCES
RIGHT
ROLLUP
ROW
ROWS

S

SELECT
SEMI
SESSION_USER
SET
SOME

T

TABLE
THEN

版权所有：腾讯云计算（北京）有限责任公司

TIME
TO
TRAILING
TRUE

U

UESCAPE
UNION
UNIQUE
UNKNOWN
UNNEST
UPDATE
USER
USING

V

VALUES

W

WHEN
WHERE
WINDOW
WITH
WITHIN

版权所有：腾讯云计算（北京）有限责任公司