

オープンソースのIoT向けデータベースGridDB ～最新の強化ポイントについて～



GridDB

TOSHIBA

東芝デジタルソリューションズ株式会社 GridDBコミュニティ版担当

野々村 克彦

2025.10.17

プロフィール



Katsuhiko Nonomura
knonomura

名前：野々村 克彦（ののむら かつひこ）

所属：

- 東芝デジタルソリューションズ株式会社
デジタルエンジニアリングセンター ソフトウェア開発部

経歴：

- 1993年 東芝 研究開発センター入所
ルールベースの不良解析システム開発に従事
- 1998年～ XMLデータベース開発に従事
- 2012年～ IoT向けデータベースGridDB開発に従事
- 2015年～ GridDBのオープンソースPJ開始
- 現在 GridDB保守、製品版開発（PythonAPIなど）
GridDB OSS版開発、OSS活動

Contents

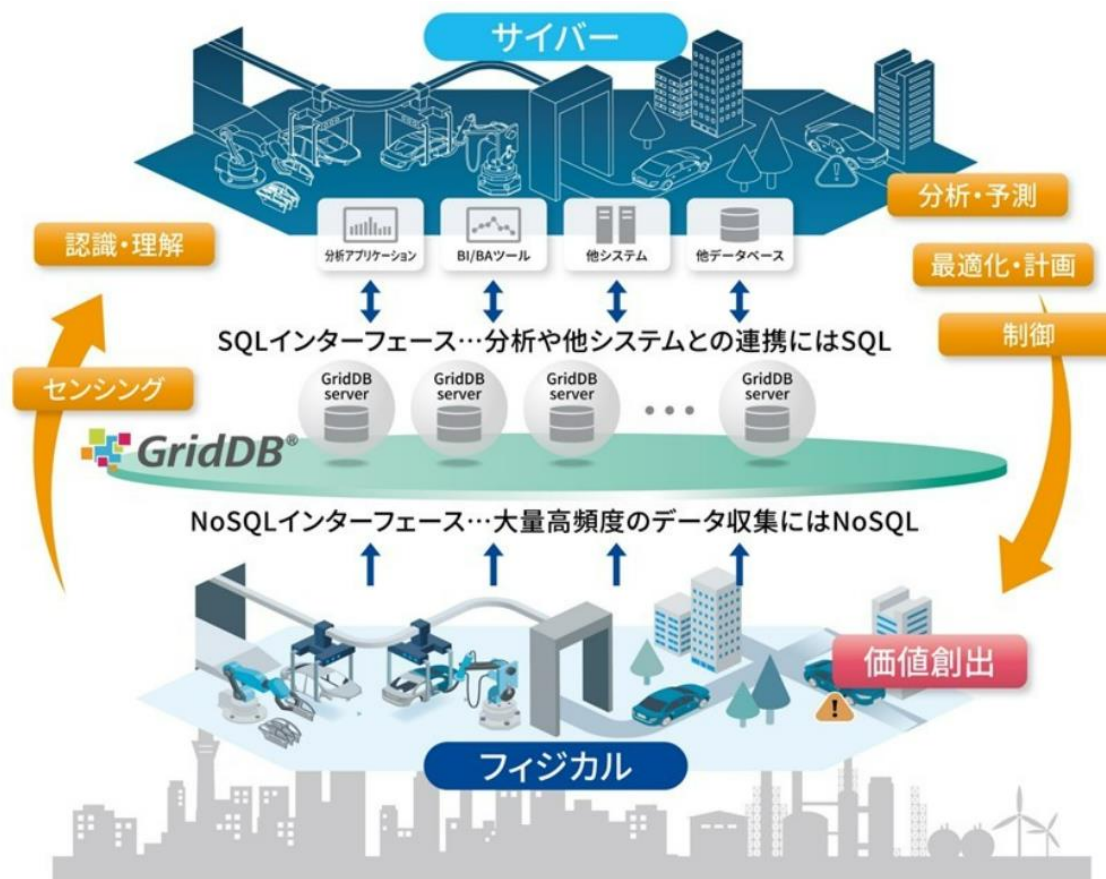
- 01 GridDBの概要
- 02 GridDBの最新の強化ポイント
- 03 GridDBの使い方
- 04 まとめ

01

GridDBの概要

GridDBについて

- GridDBは、東芝デジタルソリューションズが開発した日本発のNoSQL型のDBMSです。
- ビッグデータやIoTシステム向けに特化して作られたDBMSです。



・電力会社 低圧託送業務システム

スマートメータから収集される電力使用量を集計し、需要量と発電量のバランスを調整

<https://www.global.toshiba/jp/company/digitalsolution/articles/tsoul/22/004.html>

・HDD製造会社 品質管理システム

製造装置のセンサーデータを長期に渡って蓄積・分析し、品質分析・改善に適用

https://www.global.toshiba/content/dam/toshiba/jp/products-solutions/ai-iot/griddb/event/2024/%E8%B3%87%E6%96%99_A20_session_TDSL_GridDB.pdf

・ものづくりIoTソリューション「Meister Factoryシリーズ」

IoT データの蓄積を担う「Meister DigitalTwin」のデータ基盤として利用

<https://www.global.toshiba/jp/products-solutions/manufacturing-ict/meister-factory.html>

社会インフラ、製造業を中心に、高い信頼性・可用性が求められるシステムに適用されています。



GridDB Community Edition

高頻度・大量に発生するデータの蓄積とリアルタイムな活用をスムーズに実現する次世代のオープンソースデータベース



GridDB Enterprise Edition

高頻度・大量に発生する時系列データの蓄積とリアルタイムな活用をスムーズに実現し、ビジネスを大きく成長させるために最適化された次世代のデータベース



GridDB Cloud

高頻度・大量に発生する時系列データの蓄積とリアルタイムな活用をスムーズに実現するクラウドデータベースサービス

- **GitHub上にソース公開**



- <https://github.com/griddb/griddb>

- **オープンソース化の目的**

- ビッグデータ技術の普及促進

- 多くの人に知ってもらいたい、使ってもらいたい。
 - いろんなニーズをつかみたい。

- 他のオープンソースソフトウェア、システムとの連携強化

- V2.8 (2016年)
NoSQL機能をGitHub上にソース公開
- V4.5 (2020年)
SQL機能もソース公開
- 最新版 V5.8 (2025年6月)

GridDB CEの特徴

時系列データ指向

- データモデルはキー・コンテナ。コンテナ内でのデータ一貫性を保証
- 巨大テーブルに対するインターバル（ハッシュ）パーティショニング
- パーティショニング期限解放、分析関数(SQL)

開発の俊敏性と使いやすさ

- NoSQL（キーバリュー型）インタフェースだけではなく、SQLインタフェースを提供（デュアルインタフェース）
- （SQLインタフェース）ジョインなど複数テーブルに対するSQL

高い処理能力

- メモリを主、ストレージを従としたハイブリッド型インメモリDB
- （SQLインタフェース）SQLにおける分散並列処理
- （NoSQLインタフェース）バッチ処理 MultiPut/MultiGet/MultiQuery

拡張性

- ペタバイト級の大規模データへの対応
- コアスケールへの対応

※ チェックポイント、Redoログによる耐障害性にも対応しています

NoSQL DB (Key Value Store(KVS))とキー・コンテナモデル

<キー、バリュー>



ハッシュテーブル

--	--	--	--	--	--	--	--

キーバリュー型 (例 : Redis)

キー	値
キー	値
キー	値
キー	値
キー	値

カラム指向型 (例 : Cassandra)

キー	値	値	値	値	値
キー	値			値	値
キー	値	値	値	値	値
キー	値	値	値		値
キー	値		値	値	値

ドキュメント指向型 (例 : MongoDB)

キー	ドキュメント
キー	ドキュメント
キー	ドキュメント
キー	ドキュメント
キー	ドキュメント

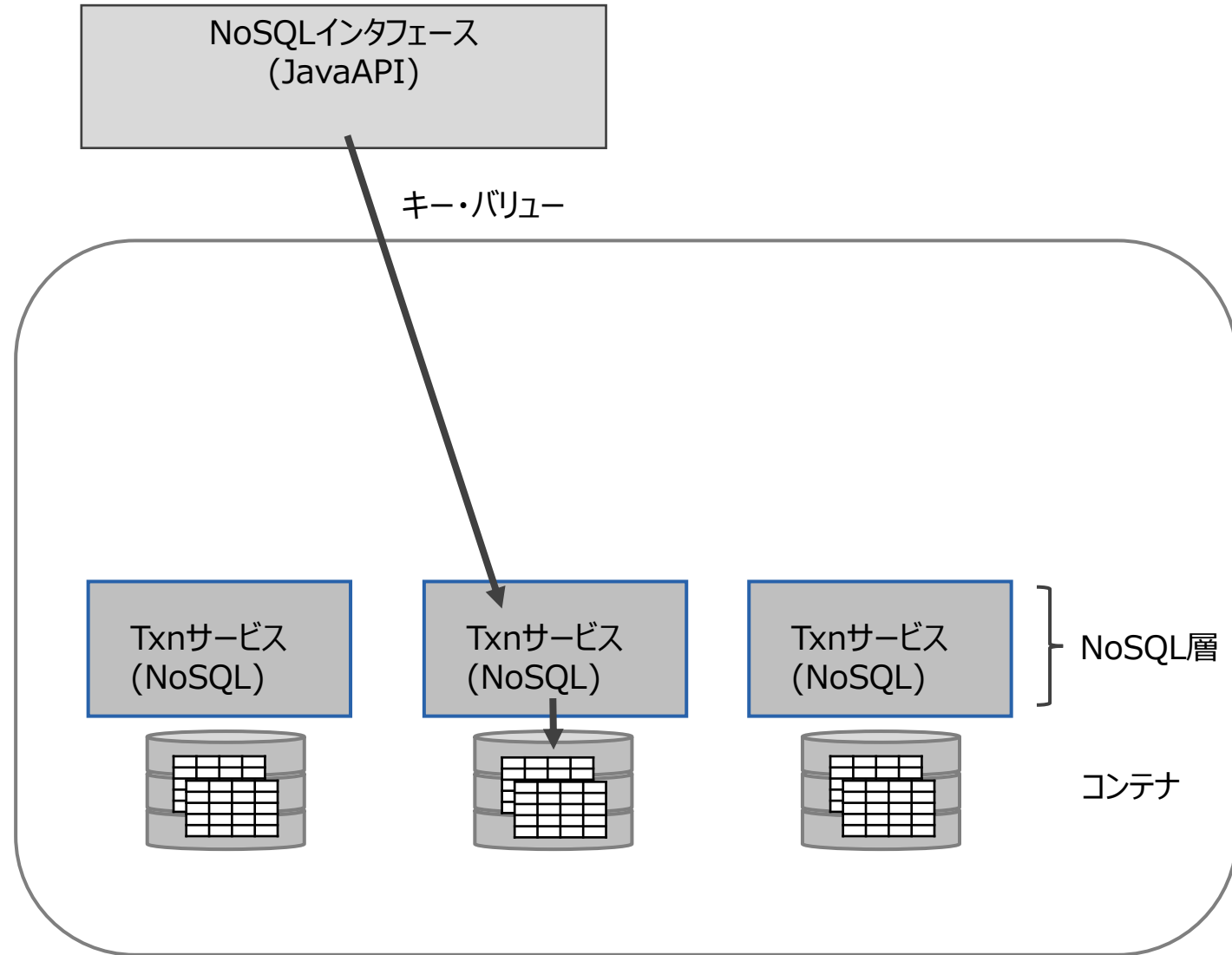
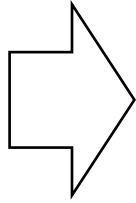
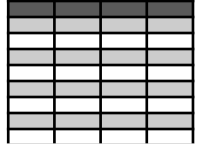
キーコンテナ型 GridDB

キー			
キー			
キー			
キー			
キー			

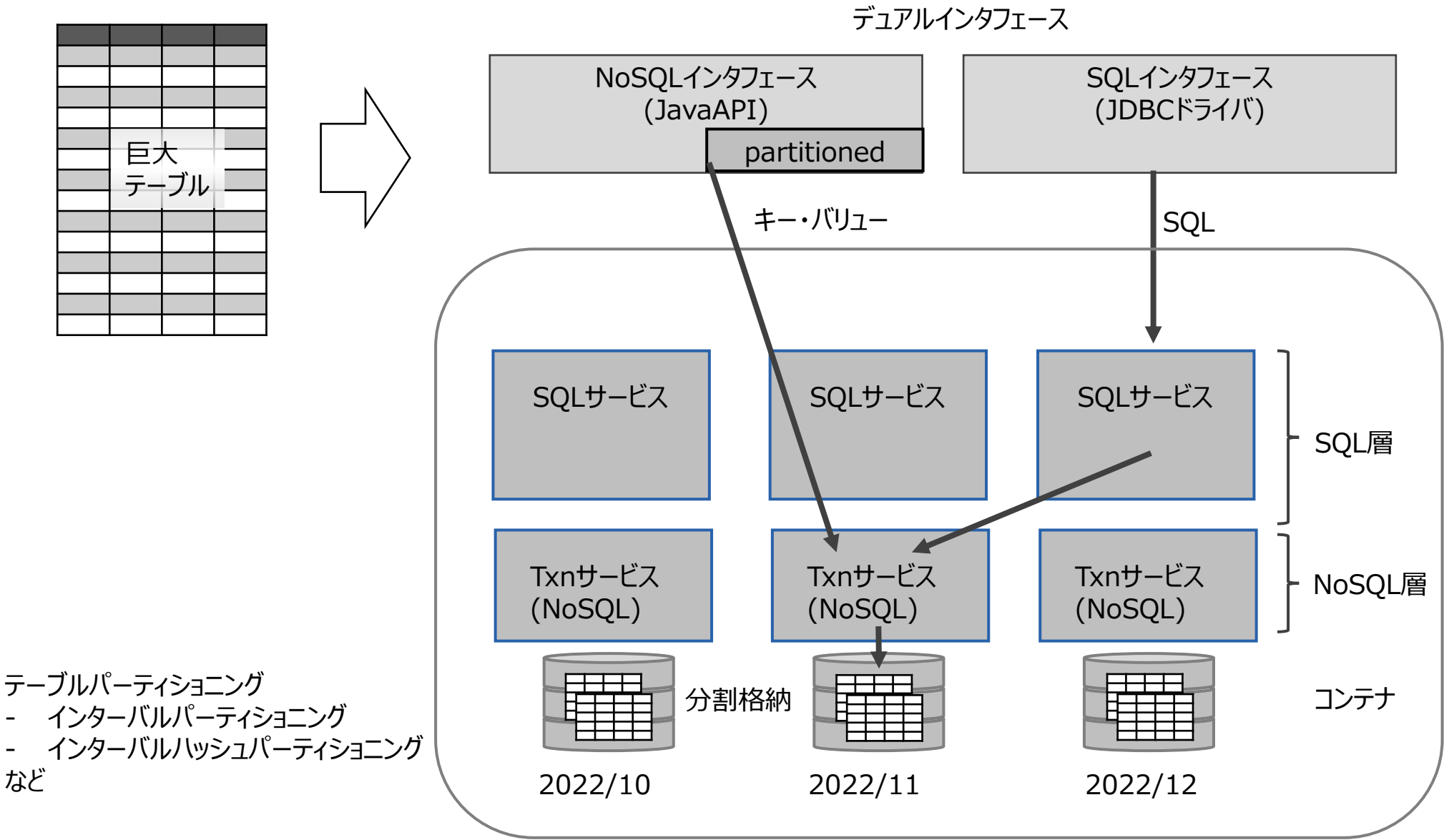
※コンテナ（テーブル）名がキーになる
※索引、検索言語TQL、トランザクションをサポート

NoSQLインタフェース

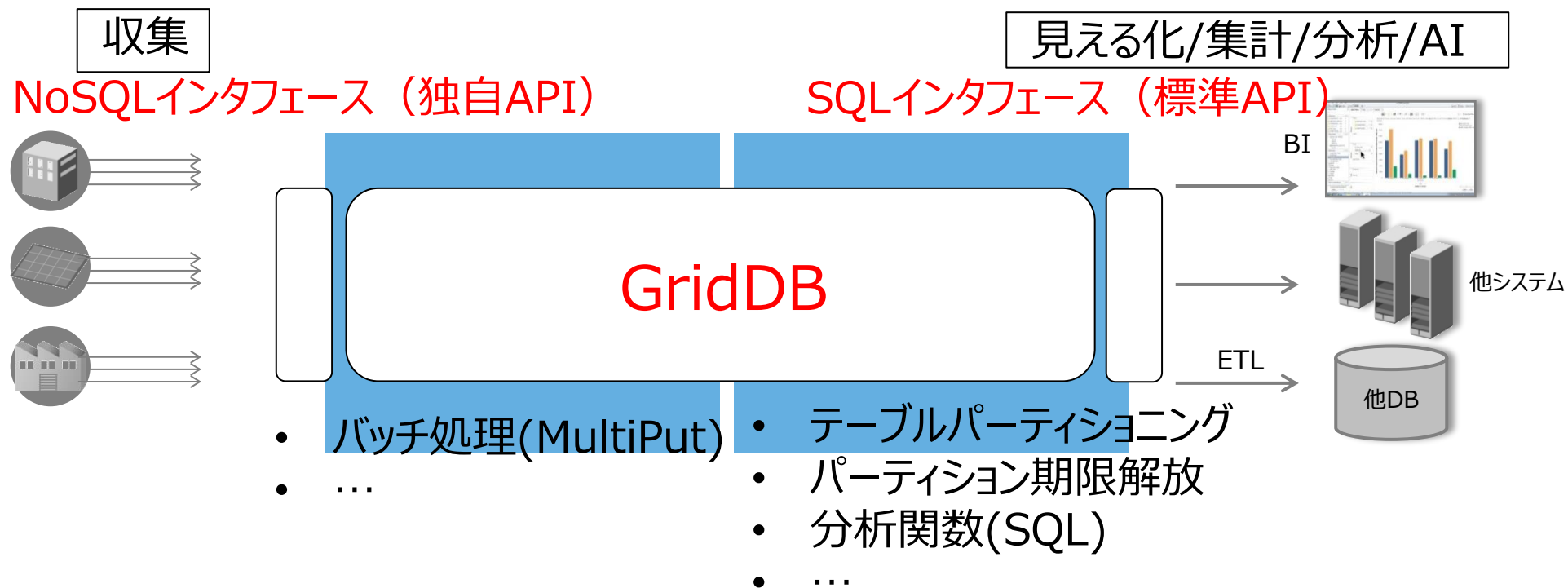
テーブル
(コンテナ)



NoSQL/SQLデュアルインタフェースとテーブルパーティショニング



NoSQL/SQLデュアルインタフェースによるシステム化



- NoSQL + SQLによる高速処理
- SQLインタフェースによる他システム連携強化

02

GridDBの最新の強化ポイント

最新の強化ポイントの一覧

- 2024/11 GridDB V5.7CE
- 2025/06 GridDB V5.8CE

	V5.7CE	V5.8CE
機能強化（主に時系列機能）	SQLメモリ使用量監視機能	SQL分析関数機能強化(移動平均演算)
性能強化（主に大規模対応）		コストベースによるジョイン方向（駆動表・内部表）の最適化
API強化	WebAPIの強化 PythonAPIの強化(SQLインタフェースへの対応)	PythonAPIの強化(NoSQLインタフェースへの対応)
その他	データベースリカバリ時間の短縮	

最新の強化ポイントの一覧

- 2024/11 GridDB V5.7CE
- 2025/06 GridDB V5.8CE

	①SQL最適化の強化 (コストベースへの対応)	ジョイン順序の最適化(V5.5) ジョイン方向（駆動表・内部表）の最適化(V5.8) 索引スキンの最適化(V5.9予定)	
機能強化（主）			章）
性能強化（主）			長・内部表）の最適化
API強化	②SQL・時系列機能の強化	SQLメモリ使用量監視機能(V5.7) SQL分析関数機能強化(移動平均演算)(V5.8)	スへの対応)
その他	③PythonAPIの強化	SQLインタフェースへの対応(V5.7) NoSQLインタフェースの強化(V5.8)	

① SQL最適化の強化（コストベースへの対応）

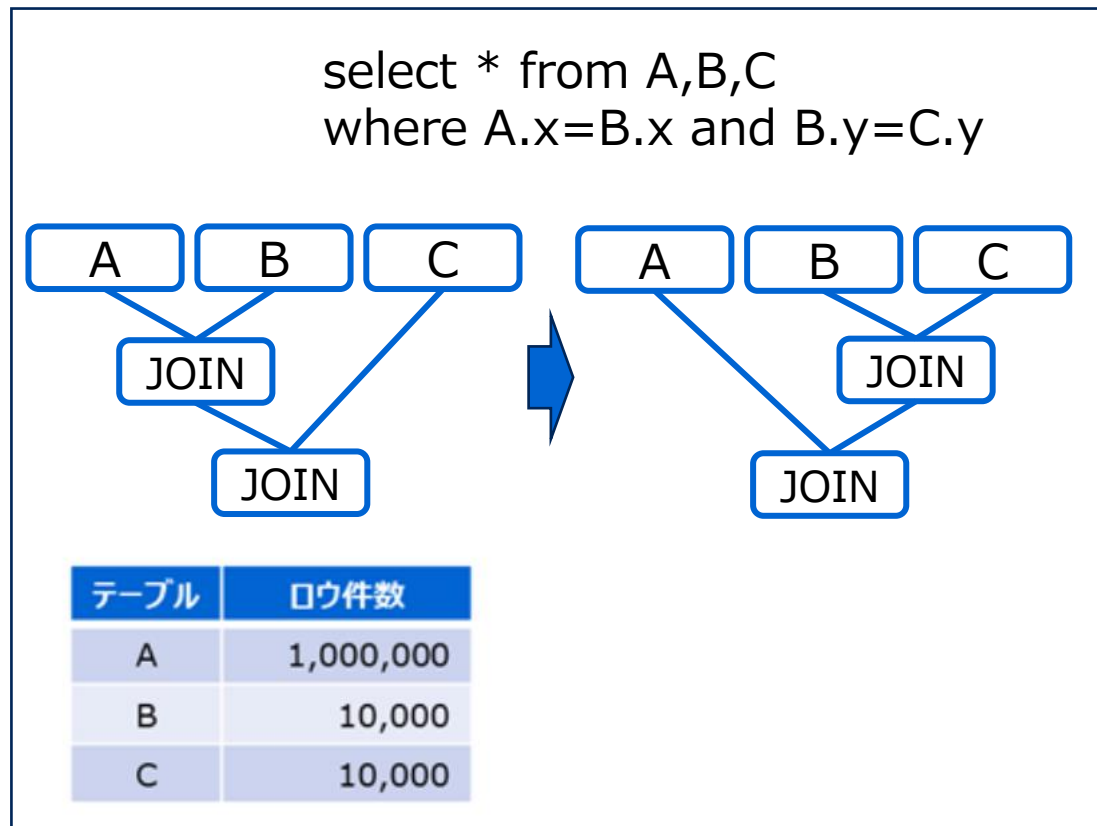
SQL最適化の手法

- ルールベース
 - SQLの記述からルールに従いSQL実行プランを生成します。
 - ルールは、SQL内に記述された演算子、カラムのデータ型・索引の有無などに関するものがあります。
 - 記述順に実行される傾向があるため、SQL書き換えによるチューニングが可能です。
- コストベース
 - 格納されているデータの統計値（各テーブルの行数など）も使って見積もった処理コストを基にSQL実行プランを生成します。
 - SQL書き換えのチューニングの手間が軽減されます。

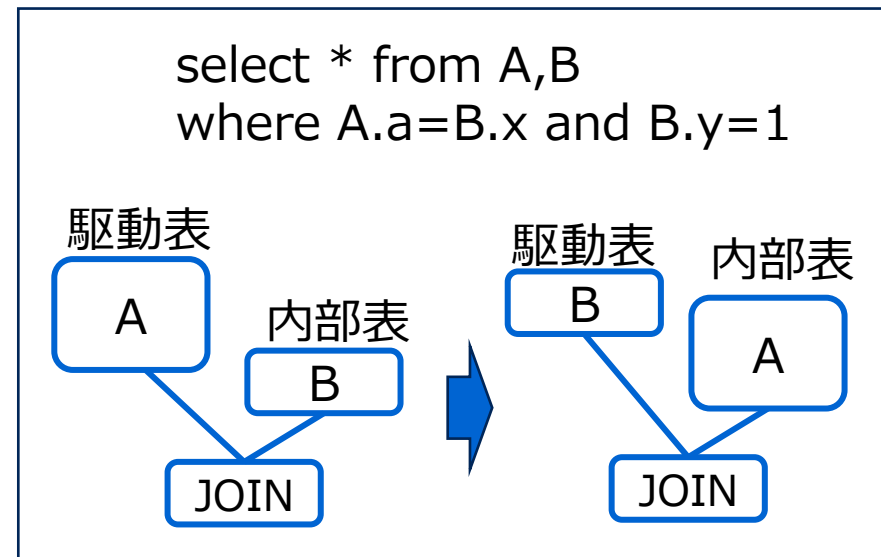
① SQL最適化の強化（コストベースへの対応）

1. ジョイン順序の最適化(V5.5)
2. ジョイン方向（駆動表・内部表）の最適化(V5.8)
3. 索引スキャンの最適化(V5.9予定)

ジョイン順序の最適化の例



ジョイン方向の最適化の例



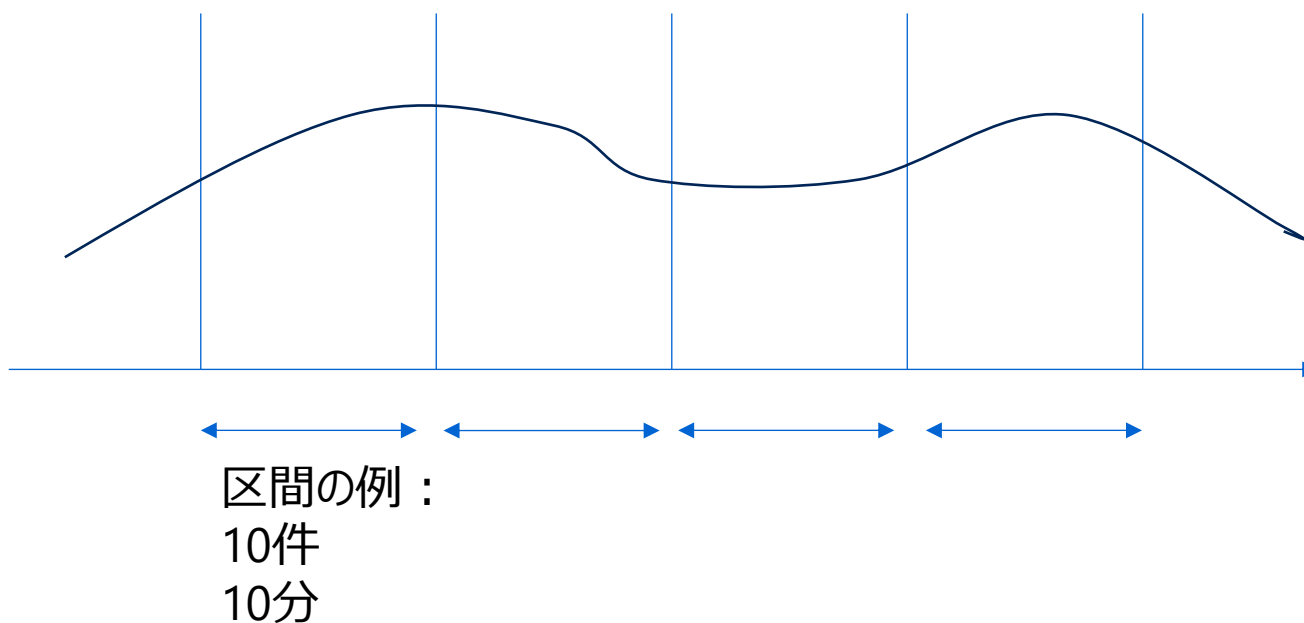
駆動表（外部表）：ジョイン処理で最初にアクセスするテーブル
内部表：次にアクセスするテーブル

① SQL最適化の強化（コストベースへの対応）：設定方法

- デフォルトではコストベースによる方法が有効になっています。
- クラスタ定義ファイル(gs_cluster.json)による方法（GridDBクラスタでの決定方法を設定）
 - /sql/costBasedJoin：trueの場合は、コストベースによる方法でジョイン順序の決定します。falseの場合は、ルールベースによる方法でジョイン順序を決定します。
 - /sql/costBasedJoinDriving：trueの場合は、コストベースによる方法でジョイン方向（駆動表・内部表）を決定します。falseの場合は、ルールベースによる方法でジョイン方向を決定します。
- ヒント文による方法（個々のSQLで切り替え可能）
 - CostBasedJoin()：コストベースによる方法でジョイン順序を決定します。
 - NoCostBasedJoin()：ルールベースによる方法でジョイン順序を決定します。
 - CostBasedJoinDriving()：コストベースによる方法で駆動表を決定します。
 - NoCostBasedJoinDriving()：ルールベースによる方法で駆動表を決定します。

② SQL・時系列機能の強化

- 移動平均演算への対応
 - 移動平均：時系列データを対象に、一定区間の窓をずらしながら計算して得られる平均
 - SQL実行機能における分析関数のオプション構文として、SQL標準で定義されているフレーム句(分析対象とするデータ範囲の件数・幅指定)に対応しました。これにより、SQLを用いた時系列データのトレンド分析が容易に実現できるようになりました。



② SQL・時系列機能の強化：移動平均の例

例：各ロウの10件前までを集計対象とする移動平均を求める。

```
SELECT  
    AVG(value1)  
      OVER (ORDER BY time  
            ROWS BETWEEN 10 PRECEDING AND CURRENT ROW  
FROM tbl1;
```

例：各ロウの10分前までを集計対象とする移動平均を求める。

```
SELECT  
    AVG(value1)  
      OVER (ORDER BY time  
            RANGE BETWEEN (10, MINUTE) PRECEDING AND CURRENT ROW  
FROM tbl1;
```

③ PythonAPIの強化

- SQLインターフェースの提供
 - 既存のGridDB JDBCドライバとJPytype(OSS)に含まれるDBAPI2、Apache Arrow(OSS)を活用
- NoSQLインタフェースの機能強化
 - GridDB CAPI、SWIG(OSS)ベースからGridDB JavaAPI、JPytype(OSS)、Apache Arrow(OSS)ベースに変更
 - パーティショニングテーブルに対する操作を可能に
 - マイクロ秒・ナノ秒精度のタイムスタンプ型、空間型に対応
 - 複合ロウキー、複合索引に対応
 - Apache Arrow形式でのデータ登録、検索結果取得に対応

※ **SWIG** (Simplified Wrapper and Interface Generator) : 様々な開発言語からCベースのAPIをコールするラッパのコードをインタフェース定義から生成するソフト(OSS)。

<https://www.swig.org/>

<https://github.com/swig/swig>

※ **JPytype** : Python から完全な Java アクセスを提供する Python モジュール(OSS)。DBAPI2にも対応。

<https://jpytype.readthedocs.io/en/latest/>

<https://github.com/jpytype-project/jpytype>

※ **DBAPI 2.0** : Python言語におけるRDBアクセスの標準API

<https://peps.python.org/pep-0249/>

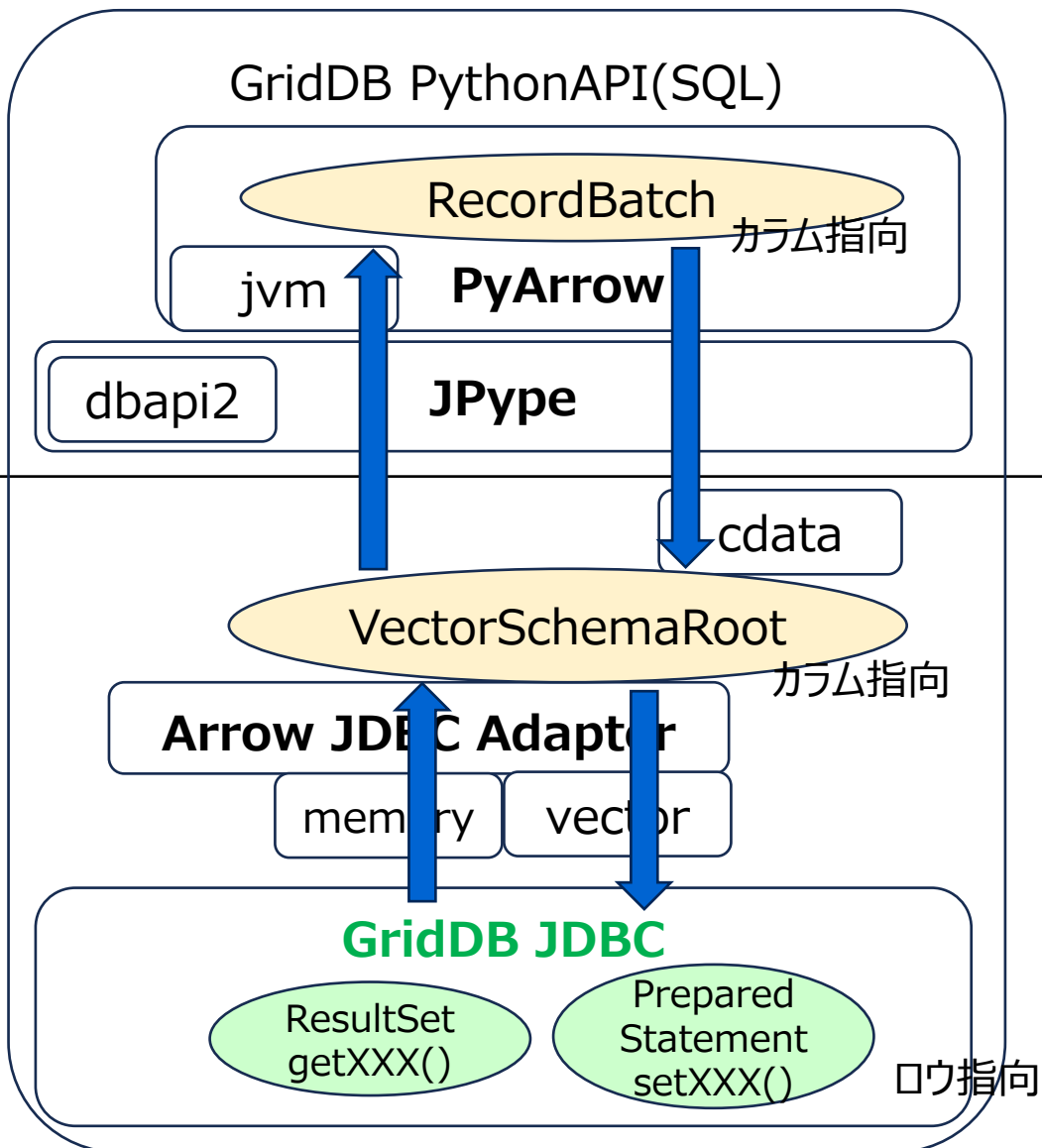
※ **Apache Arrow** : 効率的なデータ交換のために設計された、カラム指向のデータフォーマット、ソフト(OSS)

<https://arrow.apache.org/>

<https://github.com/apache/arrow>

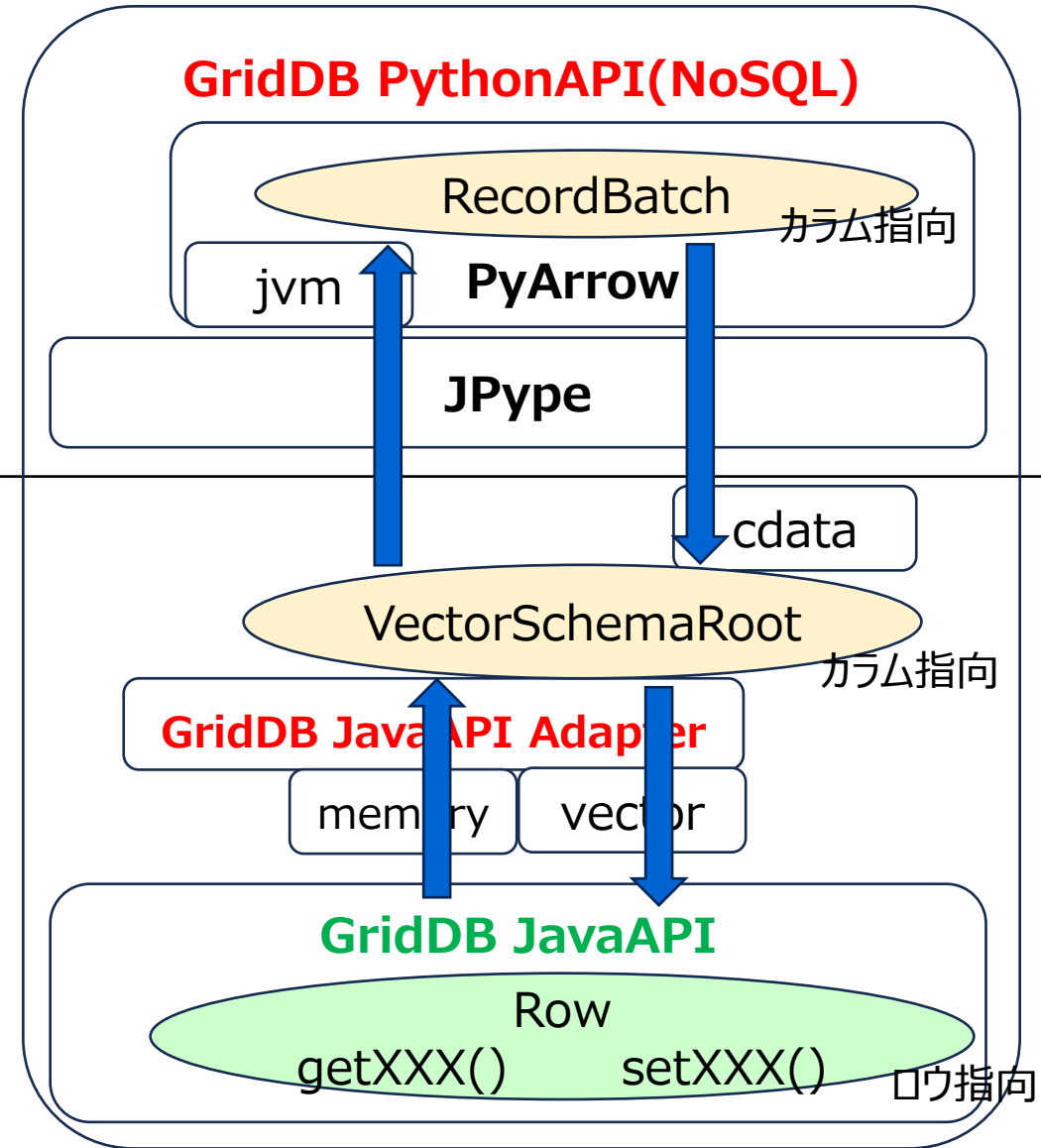
③ PythonAPIの強化：モジュール構成

赤：新規ソフト
○：クラス



Python層

Java層



03

GridDBの利用方法

- GridDBのインストール・実行
- GridDB PythonAPIのインストール・実行
 - PythonAPI(SQL)
 - PythonAPI(NoSQL)

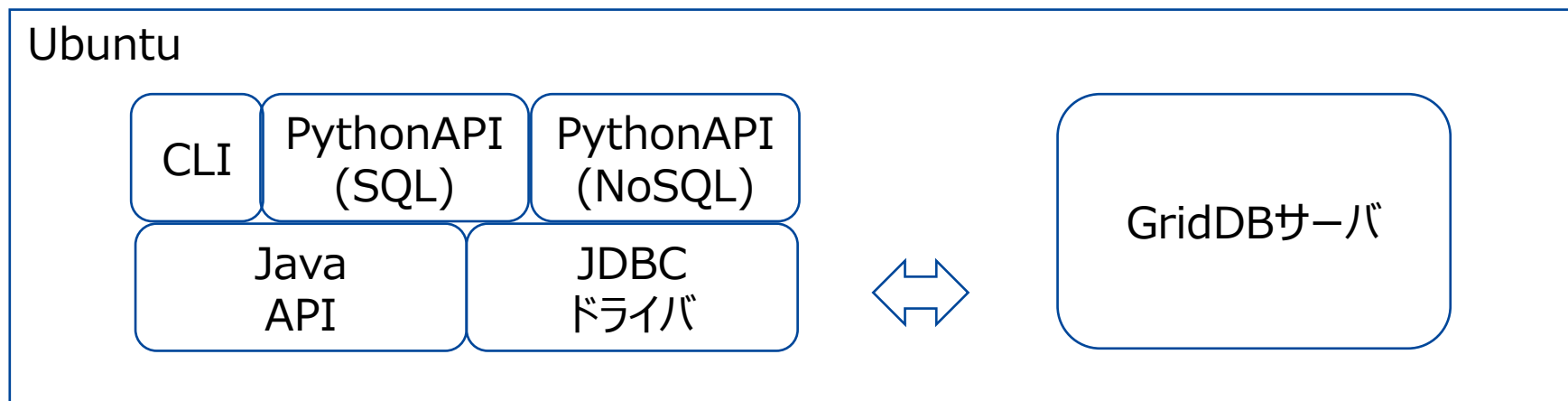
動作環境の例

<動作環境>

- Windows 11上にWSL2 Ubuntu 22.04。Java 11(デフォルト)インストール済
- メモリは16GB（ブラウザ、Zoom等のアプリを動かさないのであれば8GBも可）

<前提条件>

- 同一マシンに全ソフトウェアをインストール。ローカル実行
- GridDBのクラスタ名はmyCluster(デフォルト)
- GridDB管理者の名前はadmin、パスワードはadmin



※GridDBサーバ、JavaAPI : <https://github.com/griddb/griddb>

※GridDB JDBCドライバ : <https://github.com/griddb/jdbc>

※GridDB CLI : <https://github.com/griddb/cli>

GridDBのインストール&起動の手順

【インストール】

1. GridDBサーバのインストール

```
$ wget https://github.com/griddb/griddb/releases/download/v5.8.0/griddb_5.8.0_amd64.deb
```

```
$ sudo dpkg -i griddb_5.8.0_amd64.deb
```

2. GridDB CLI（コマンドライン・インタフェース）のインストール

```
$ wget https://github.com/griddb/cli/releases/download/v5.8.0/griddb-ce-cli_5.8.0_amd64.deb
```

```
$ sudo dpkg -i griddb-ce-cli_5.8.0_amd64.deb
```

【起動】

3. GridDBのサービス起動

```
$ sudo systemctl start gridstore
```

4. CLI起動

```
$ sudo su - gsadm
```

```
$ gs_sh
```

[SQLの例]

テーブル作成

```
> create table t1 (c0 long, c1 long);
```

データ登録

```
> insert into t1 values(1, 2);
```

検索

```
> select * from t1;
```

```
> get
```

※GridDBサービスの停止

```
$ systemctl stop gridstore
```

※ サービス機能(systemctl)が利用できない環境の場合は、gs_startnode等の運用コマンドをご利用ください。

GridDBのインストール&起動の手順

【インストール】

1. GridDBサーバのインストール

```
$ wget https://github.com/griddb/griddb/releases/download/v5.8.0/griddb_5.8.0_amd64.deb
```

```
$ sudo dpkg -i griddb_5.8.0_amd64.deb
```

2. GridDB CLI（コマンドライン・インタフェース）のインストール

```
$ wget https://github.com/griddb/cli/releases/download/v5.8.0/griddb-ce-cli_5.8.0_amd64.deb
```

```
$ sudo dpkg -i griddb-ce-cli_5.8.0_amd64.deb
```

【起動】

3. GridDBのサービス起動

```
$ sudo systemctl start gridstore
```

4. CLI起動

```
$ sudo su - gsadm
```

```
$ gs_sh
```

[SQLの例]

テーブル作成

```
> create table t1 (c0 long, c1 long);
```

データ登録

```
> insert into t1 values(1, 2);
```

検索

```
> select * from t1;
```

```
> get
```

※GridDBサービスの停止

```
$ systemctl stop gridstore
```

※ サービス機能(systemctl)が利用できない環境の場合は、gs_startnode等の運用コマンドをご利用ください。

わずかなステップだけで
CLIによるSQLなどの操作が開始できます。

PythonAPI(SQL) : インストール

【インストール】

1. JPytypeのインストール

```
$ pip install JPytype==1.5.0
```

2. GridDB JDBCドライバのインストール

```
$ mvn dependency:get -Dartifact=com.github.griddb:gridstore-jdbc:5.8.0  
-Ddest=./gridstore-jdbc.jar
```

⇒ カレントフォルダにgridstore-jdbc.jarファイルが配置されます。

3. (大量データ取得が必要な場合)Apache Arrowのインストール・設定

3-1. Apache ArrowのPython層の処理用(**pyarrow**)のインストール

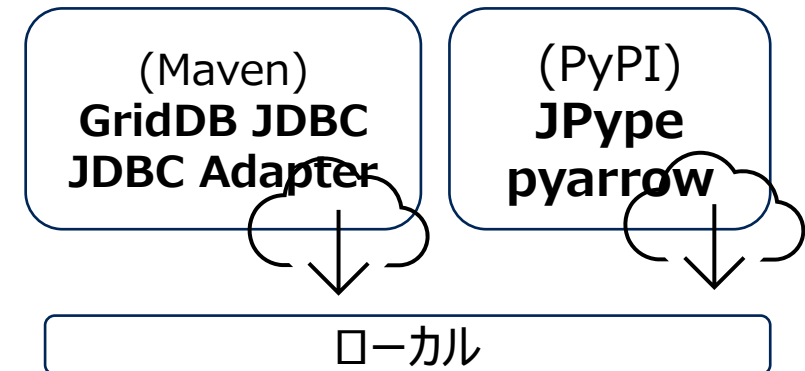
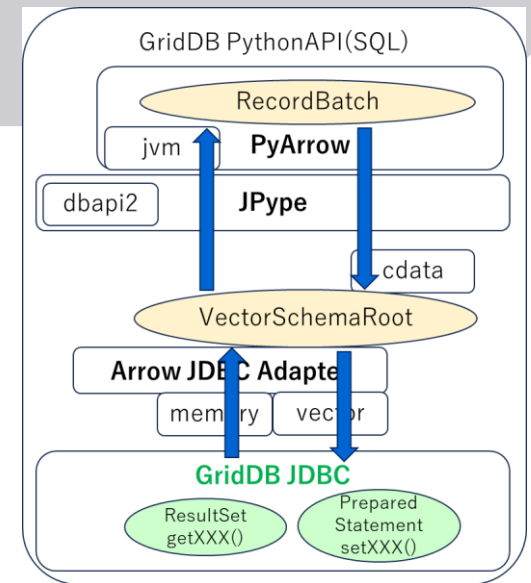
```
$ pip install pyarrow==16.0.0
```

3-2. Apache ArrowのJava層の処理用(**Arrow JDBC Adapter**)のインストール

```
$ wget https://github.com/griddb/jdbc/tree/master/sample/ja/python/dbapi2/jpytype/pom.xml  
$ mvn assembly:single
```

⇒ targetフォルダにgridstore-arrow-jdbc-0.1-jar-with-dependencies.jarファイルが生成されます。

```
$ export _JAVA_OPTIONS="--add-opens=java.base/java.nio=ALL-UNNAMED"
```



PythonAPI(SQL) : 事前処理

- 以下のインポートとJVMの起動が事前に必要です。

```
import jpye
```

```
import jpye.dbapi2
```

```
jpye.startJVM(classpath=['./gridstore-jdbc.jar'])
```

PythonAPI(SQL) : SQLの実行例

接続

```
url = "jdbc:gs://127.0.0.1:20001/myCluster"
conn = jpytype.dbapi2.connect(url, driver="com.toshiba.mwcloud.gs.sql.Driver",
    driver_args={"user":"admin", "password":"admin"})
curs = conn.cursor()
```

SQLの実行

```
curs.execute("CREATE TABLE Sample ( id integer PRIMARY KEY, value string );") # テーブル生成
curs.execute("INSERT INTO Sample values (1, 'test1');") # データ挿入
curs.execute("SELECT * from Sample where ID > 0;") # 検索
print(curs.fetchall())
```

※ PythonAPI(SQL)の詳細はOSC2025 Tokyo/Springの発表資料をご参考ください。

PythonAPI(NoSQL) : インストール・設定

【インストール】

1. GridDB JavaAPIのインストール

```
$ curl -L -o gridstore.jar
```

<https://repo1.maven.org/maven2/com/github/griddb/gridstore/5.8.0/gridstore-5.8.0.jar>

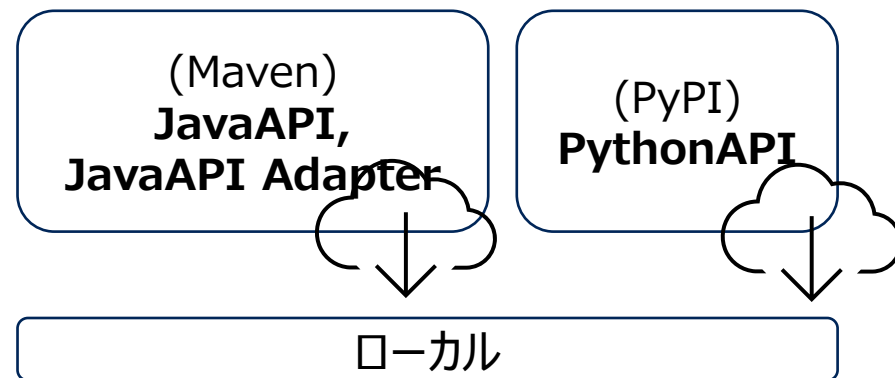
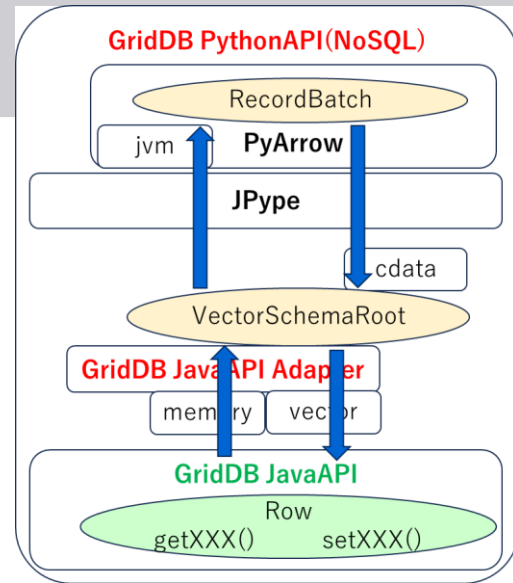
2. GridDB JavaAPI Adapter for Apache Arrowのインストール

```
$ curl -L -o gridstore-arrow.jar https://repo1.maven.org/maven2/com/github/griddb/gridstore-arrow/5.8.0/gridstore-arrow-5.8.0.jar
```

3. GridDB PythonAPIのインストール

```
$ python3 -m pip install griddb_python
```

※ Jpypeとpyarrowがインストールされる



PythonAPI(NoSQL) : 事前処理

(A) 推奨

以下のインポートとJVMの起動が事前に必要です。

```
import jpye  
jpye.startJVM(classpath=['./gridstore.jar', './gridstore-arrow.jar'])  
import griddb_python as griddb
```

(B) 旧版の方法

「jpye」のインポートやJVMの起動の記述なしで、従来のPython APIと同じ方法で使うこともできます。

```
import griddb_python as griddb
```

但し、環境変数CLASSPATHに以下のJarファイルの場所を事前に設定してください。

例 :

```
$ export CLASSPATH=$CLASSPATH:./gridstore.jar:./gridstore-arrow.jar
```

PythonAPI(NoSQL) : 簡単な例

```
#Storeオブジェクトの取得
factory = griddb.StoreFactory.get_instance()
gridstore = factory.get_store(host="127.0.0.1",
                               port=10001,
                               cluster_name="myCluster",
                               username="admin", password="admin")

#コンテナ生成
conInfo = griddb.ContainerInfo("col01",
                                [ ["id", griddb.Type.LONG],
                                  ["c1", griddb.Type.STRING],
                                  ["c2", griddb.Type.BOOL]],
                                griddb.ContainerType.COLLECTION, True)
col = gridstore.put_container(conInfo)
```

```
#データ登録
ret = col.put([1, "abc", True])
```

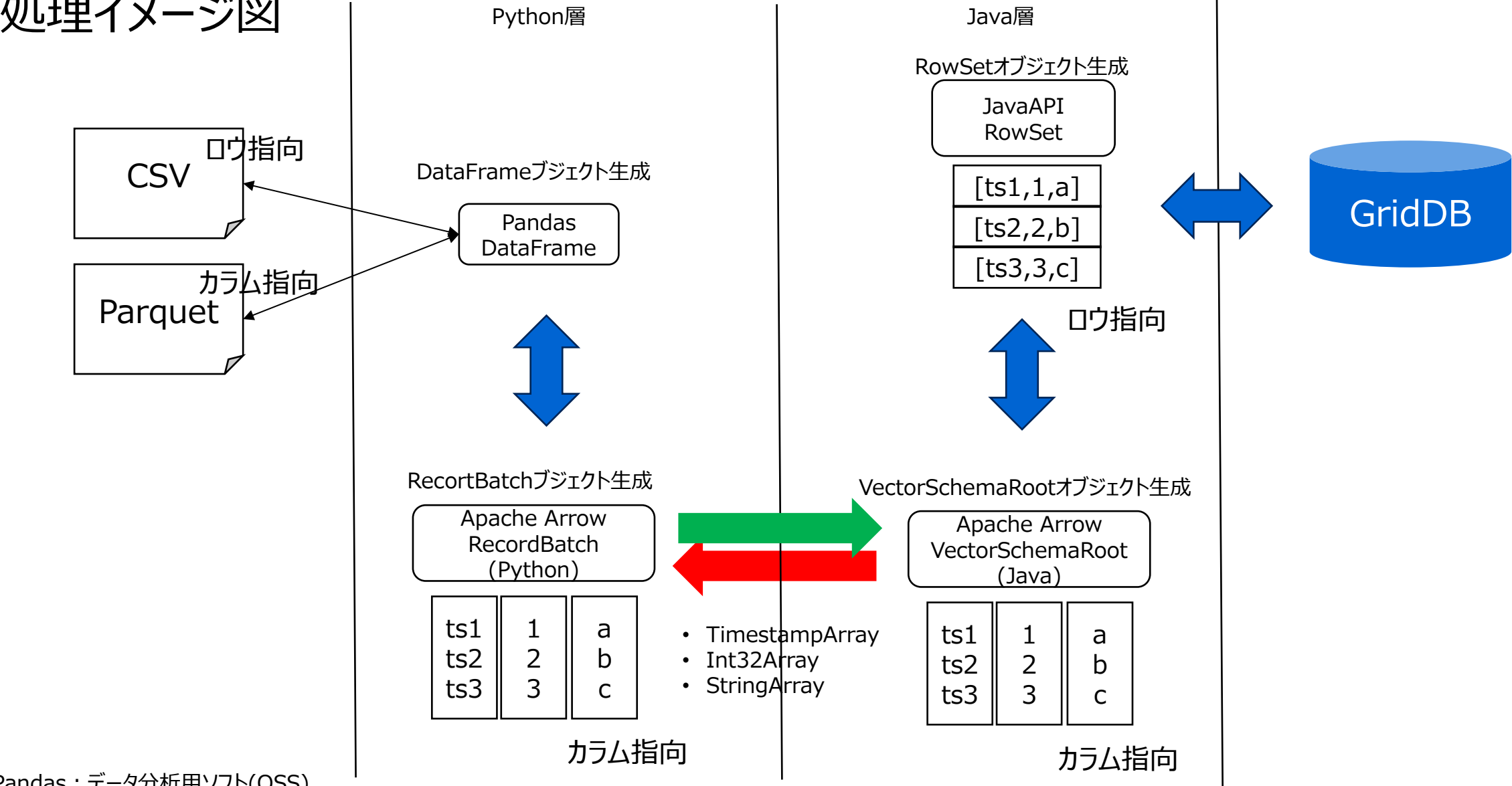
```
#検索
query=col.query("select * where c1 = 'abc'")
rs = query.fetch()
while rs.has_next():
    data = rs.next()
    print("Row: id={0} c1={1} c2={2}".format(data[0],
                                             data[1], data[2]))
```

※ 以下の流れで操作を行います。

1. コンテナの生成・取得
2. データ登録(PUT)
3. データ取得(GET)、検索(TQL)

PythonAPI(NoSQL) : Apache Arrowを利用した例 1/2

処理イメージ図



PythonAPI(NoSQL) : Apache Arrowを利用した例 2/2

```
import sys
import pyarrow as pa
import pandas as pd

ra = griddb.RootAllocator(sys.maxsize)
# Arrow RootAllocatorを生成
df1 = pd.read_csv("data.csv")
# CSVからPandas DataFrameオブジェクトを生成

# Storeオブジェクトの取得
factory = griddb.StoreFactory.get_instance()
gridstore = factory.get_store(host="127.0.0.1",
                              port=10001,
                              cluster_name="myCluster",
                              username="admin", password="admin")

# コンテナ生成
conInfo = griddb.ContainerInfo("col01",
                               [["id", griddb.Type.LONG],
                                ["c1", griddb.Type.STRING],
                                ["c2", griddb.Type.BOOL]],
                               griddb.ContainerType.COLLECTION, True)
col = gridstore.put_container(conInfo)
```

```
id,c1,c2
1,"abc",True
2,"def",False
```

```
# データ登録
rb = pa.record_batch(df1)
# Arrow RecordBatchオブジェクトを生成
col.multi_put(rb, ra)
# Arrow RecordBatchオブジェクトを使って
# N件ロウデータを一括登録

# 検索
query = col.query("select *")
query.set_fetch_options(root_allocator=ra)
# Arrow形式での取得を設定
rs = query.fetch()
rb = rs.next_record_batch()
# Arrow RecordBatchオブジェクトで結果取得
df2 = rb.to_pandas()
# Pandas DataFrameオブジェクトを生成
```

参考資料一覧 (1/2)

マニュアル :

- GridDB SQLリファレンス、GridDB SQLチューニングガイド、GridDB JDBCドライバ説明書、など

<https://github.com/griddb/docs-ja>

- GridDB PythonAPI(SQL)ガイド

<https://github.com/griddb/jdbc/tree/master/sample/ja/python/dbapi2/jpype/singlehtml>

- GridDB PythonAPI(NoSQL)リファレンス

https://griddb.org/python_client/index.html

サンプルコード :

- PythonAPI(SQL)

<https://github.com/griddb/jdbc/tree/master/sample/ja/python/dbapi2/jpype>

- PythonAPI(NoSQL)

https://github.com/griddb/python_client/tree/master/sample

参考資料一覧 (2/2)

過去のオープンソースカンファレンスのプレゼン資料：

- GridDB CLI (コマンドラインインタフェース) の使い方 (OSC2022 Online/Spring)

<https://speakerdeck.com/griddb/sqldeyuaruintahuesuwobei-etaiotxiang-kedetabesugriddb-komandorainintahuesu-cli-woshi-tutemimasiyou>

- GridDB PythonAPI(SQL)の使い方 (OSC2025 Tokyo/Spring)

<https://event.ospn.jp/slides/osc2025-spring/GridDB-OSC2025-TokyoSpring.pdf>

04

まとめ

まとめ

- GridDBはビッグデータ・IoT向けのデータベースです。
- GridDBの概要、最新の強化ポイント、GridDBの利用方法についてご紹介しました。
 - 今後も様々な拡張、拡充を進めて参ります。

GridDBのオープンソース版(GridDB CE)を是非とも使ってみてください。

<https://github.com/griddb/>

こんな使い方をしたい、こんな機能が欲しい等のご意見もお待ちしています。
(日本語でOKです)



GridDB Community Edition

高頻度・大量に発生するデータの蓄積とリアルタイムな活用をスムーズに実現する次世代のオープンソースデータベース



GridDB Enterprise Edition

高頻度・大量に発生する時系列データの蓄積とリアルタイムな活用をスムーズに実現し、ビジネスを大きく成長させるために最適化された次世代のデータベース



GridDB Cloud

高頻度・大量に発生する時系列データの蓄積とリアルタイムな活用をスムーズに実現するクラウドデータベースサービス

GridDB Cloudを無料で使ってみませんか？

griddb cloud

検索

GridDB Cloud無料プラン

申し込みフォームがより簡単になりました。必要なのは「お名前」と「メールアドレス」のみですので、ぜひお試しください。



<https://www.global.toshiba/jp/products-solutions/ai-iot/griddb/product/griddb-cloud.html>

A screenshot of the GridDB Cloud Free Plan Sign-up Form. The form is titled 'GridDB Cloud 無料プラン申し込みフォーム'. It includes instructions for users to sign up for the free plan. The form fields are: 'お名前 必須' (Name, Required) with sub-fields for '姓' (Surname) and '名' (Given Name), 'Email アドレス 必須' (Email Address, Required), and 'Email アドレス (確認) 必須' (Email Address (Confirmation), Required). A green arrow from the previous image points to this form.

https://form.ict-toshiba.jp/download_form_griddb_cloud_freeplan

TOSHIBA

各エディションの違い

- インタフェースはほぼ同じ
- クラスタ構成の有無の違い

項目	機能	Community Edition	Enterprise Edition	Cloud
サポート			✓	✓
プロフェッショナルサービス			✓	✓
データ管理	時系列コンテナ	✓	✓	✓
	コレクションコンテナ	✓	✓	✓
	索引	✓	✓	✓
	アフィニティ	✓	✓	✓
	テーブルパーティショニング	✓	✓	✓
クエリ言語	TQL	✓	✓	✓
	SQL	✓	✓	✓
NoSQLインタフェース	Java	✓	✓	✓
	C言語	✓	✓	✓
NewSQL(SQL) インタフェース	JDBC	✓	✓	✓
	ODBC		✓	✓
WebAPI		✓	✓	✓
時系列データ	時系列分析関数	✓	✓	✓
	期限付き解放機能	✓	✓	✓
クラスタリング	機能クラスタ構成		✓	✓
	分散データ管理		✓	✓
	レプリケーション		✓	✓
運用管理	ローリングアップグレード		✓	
	オンラインバックアップ		✓	✓
	エクスポート / インポート	✓	✓	✓
	運用管理GUI		✓	✓
セキュリティ	CLIツール	✓	✓	✓
	信暗号化 (TLS/SSL)		✓	✓
	認証機能 (LDAP)		✓	
オンプレミス環境	オンプレミス環境	✓	✓	
クラウドサービス	クラウドサービス			✓

ご参考 : GridDBに関する情報

- **GridDB GitHubサイト**

- <https://github.com/griddb/griddb/>

griddb github	検索
---------------	----

- **GridDB デベロッパーズサイト**

- <https://griddb.net/>

griddb net	検索
------------	----

- **X (旧Twitter) : GridDB (日本)**

- https://x.com/griddb_jp

twitter griddb	検索
----------------	----

- **X (旧Twitter) : GridDB Community**

- <https://x.com/GridDBCommunity>

- **Facebook: GridDB Community**

- <https://www.facebook.com/griddbcommunity/>

- **Wiki**

- <https://ja.wikipedia.org/wiki/GridDB>

- **GridDB お問い合わせ**

- OSS版のプログラミング関連 : Stackoverflow(<https://ja.stackoverflow.com/search?q=griddb>)もしくはGitHubサイトの各リポジトリのIssueをご利用ください

- プログラミング関連以外 : contact@griddb.netもしくはcontact@griddb.orgをご利用ください

