NoSQL/SQLデュアルインターフェースを備えたIoT向けデータベースGridDB
～強化された時系列データ管理・検索機能について～

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

2023.9.29, 30

Contents

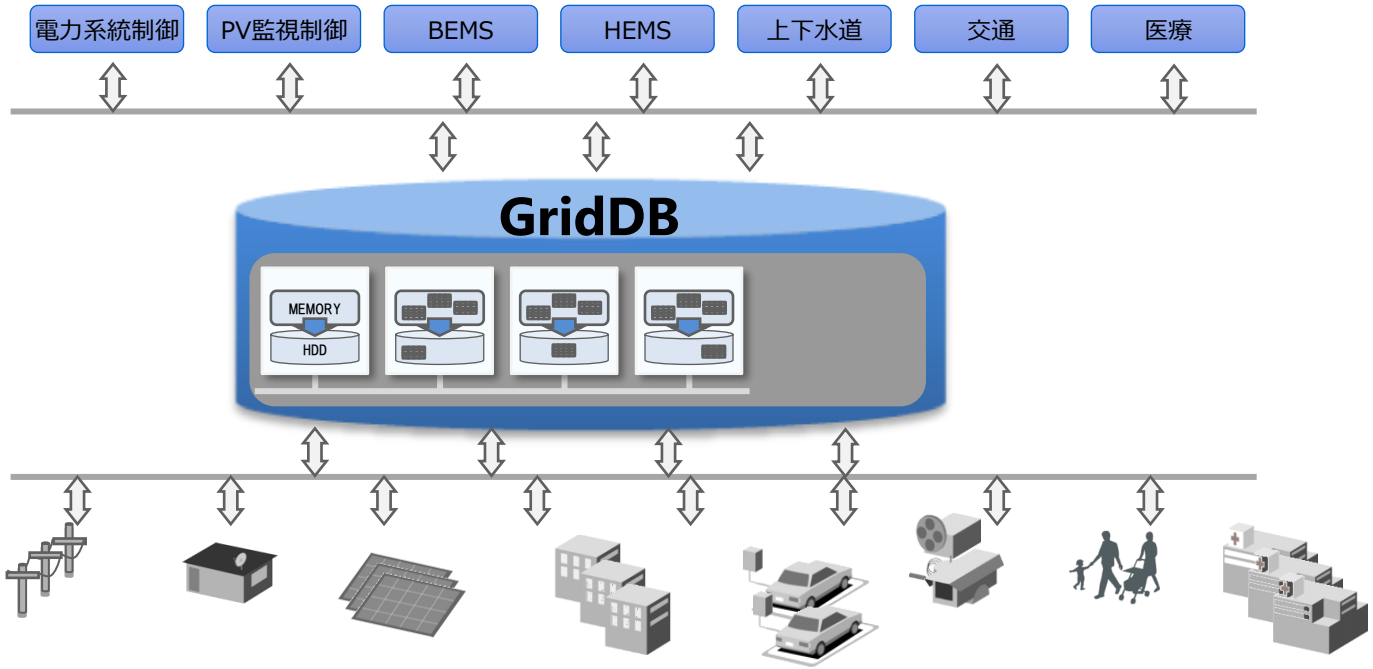
01

GridDBの概要

GridDB

- 日本発のビッグデータ／IoT向けデータベース

※IoT：モノのインターネット（Internet Of Things）。大量のモノ（センサなど）から得られるデータがインターネットにつながること。



GridDBはオープンソース？

GridDB Community 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

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る

主な適用事例

- 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

・フランス リヨン 太陽光発電 監視・診断システム
　発電量の遠隔監視、発電パネルの性能劣化を診断
・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
・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
・半導体製造ライン 履歴管理システム
　製造履歴や品質履歴、材料データなどを横串で分析し、製品の品質管理やトレーサビリティに利用
・半導体製造ライン 異常検知システム
　製造ラインのセンサーデータをリアルタイムにAIで分析し、製造ラインの異常を検知
・デンソー ファクトリー IoT
　工場のDigitalTwinを実現し、生産性向上
….

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、使ってみてもらいたい。
  - 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
  https://github.com/griddb/griddb_nosql
- V4.5 (2020年)
  SQL機能もソース公開
  https://github.com/griddb/griddb
- 最新版 V5.3 (2023年6月12日)

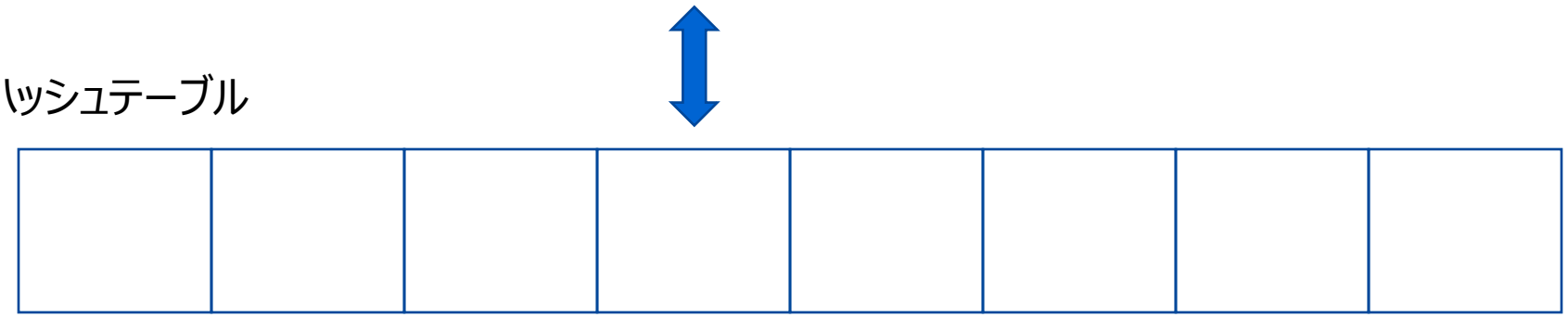
GridDB CEの特徴

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への対応

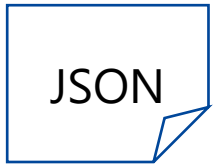
NoSQL DB (Key Value Store(KVS))とキー・コンテナモデル

<キー、バリュー>

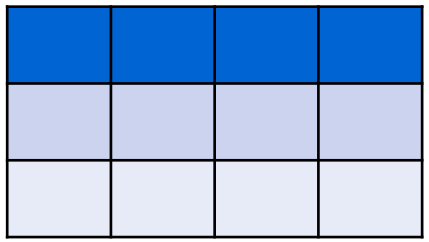


123

単純値：（例）Redis

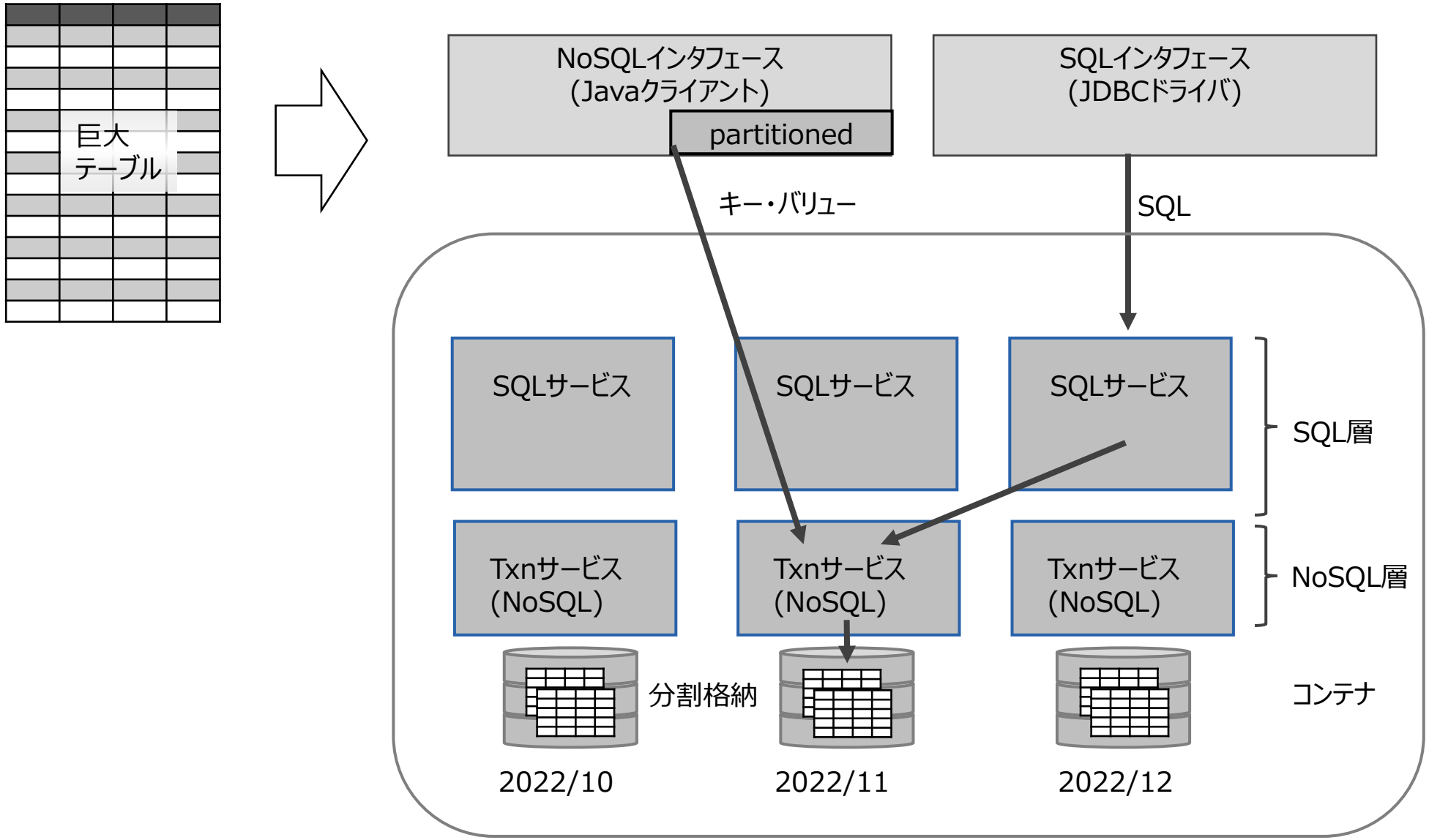


ドキュメント：（例）MongoDB



コンテナ（テーブル）： GridDB
※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

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



テーブルパーティショニング

- データ登録数が多い巨大なテーブルのデータを分散配置することで、プロセッサの並列実行を可能とし、巨大テーブルのアクセスを高速化するための機能
  - ハッシュパーティショニング
    - ✓ 選択基準：散らすべきキーにランダム性が高く、キーの間に処理上の関連性が無い場合
  - インターバルパーティショニング
    - ✓ 選択基準：散らすべきキーの数値的な範囲で散らしたい場合
  - インターバルハッシュパーティショニング
    - ✓ 選択基準：インターバルパーティショニングでは力不足の場合

```
-- ハッシュ
CREATE TABLE a3 (code INT, ts TIMESTAMP, dest STRING NOT NULL)
    PARTITION BY HASH(dest) PARTITIONS 10
-- インターバル
CREATE TABLE a1 (code INT, ts TIMESTAMP NOT NULL, dest STRING)
    PARTITION BY RANGE(ts) EVERY(1,DAY)
-- インターバルハッシュ
CREATE TABLE a4 (code INT NOT NULL, ts TIMESTAMP, dest STRING)
    PARTITION BY RANGE(ts) EVERY(1,DAY)
    SUBPARTITION BY HASH(dest) SUBPARTITIONS 2
```

（パーティション）期限開放

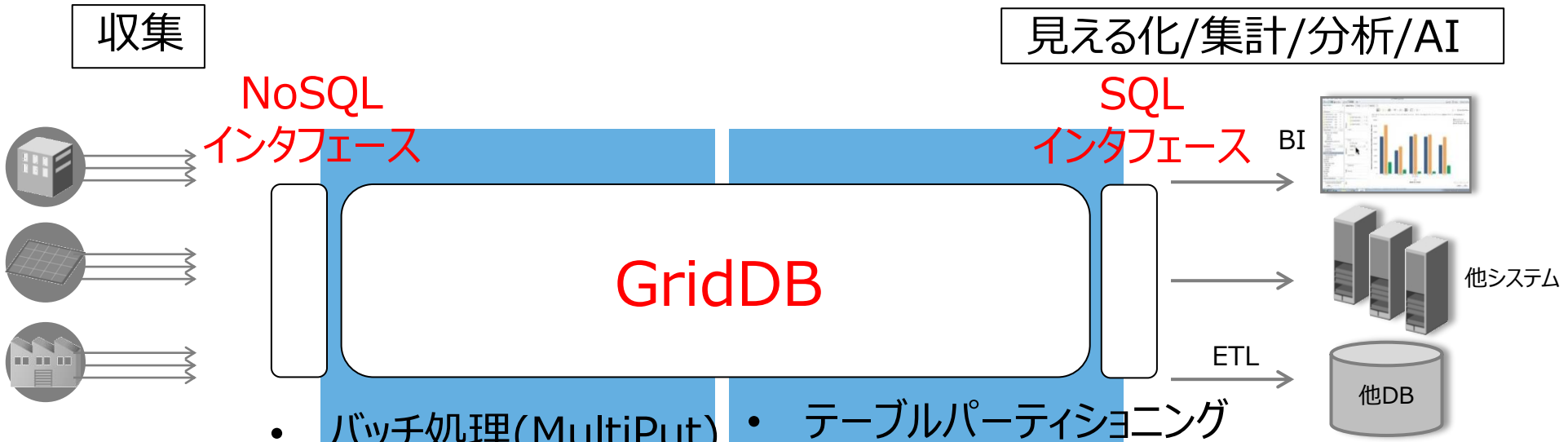
- 期限解放：設定した保持期間を超えたロウデータを、検索や削除などの操作対象から外して参照不可とした後、DBから物理的に削除する機能。テーブル作成時に保持期限を指定。
- ライフサイクル

  ①登録状態

  ②期限切れ状態：参照不可。検索でヒットしなくなる

  ③削除可能状態：この状態で自動削除
- 削除されたエリアが新たなデータ登録で再利用されることで、DBサイズの肥大化を抑えることができる。

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

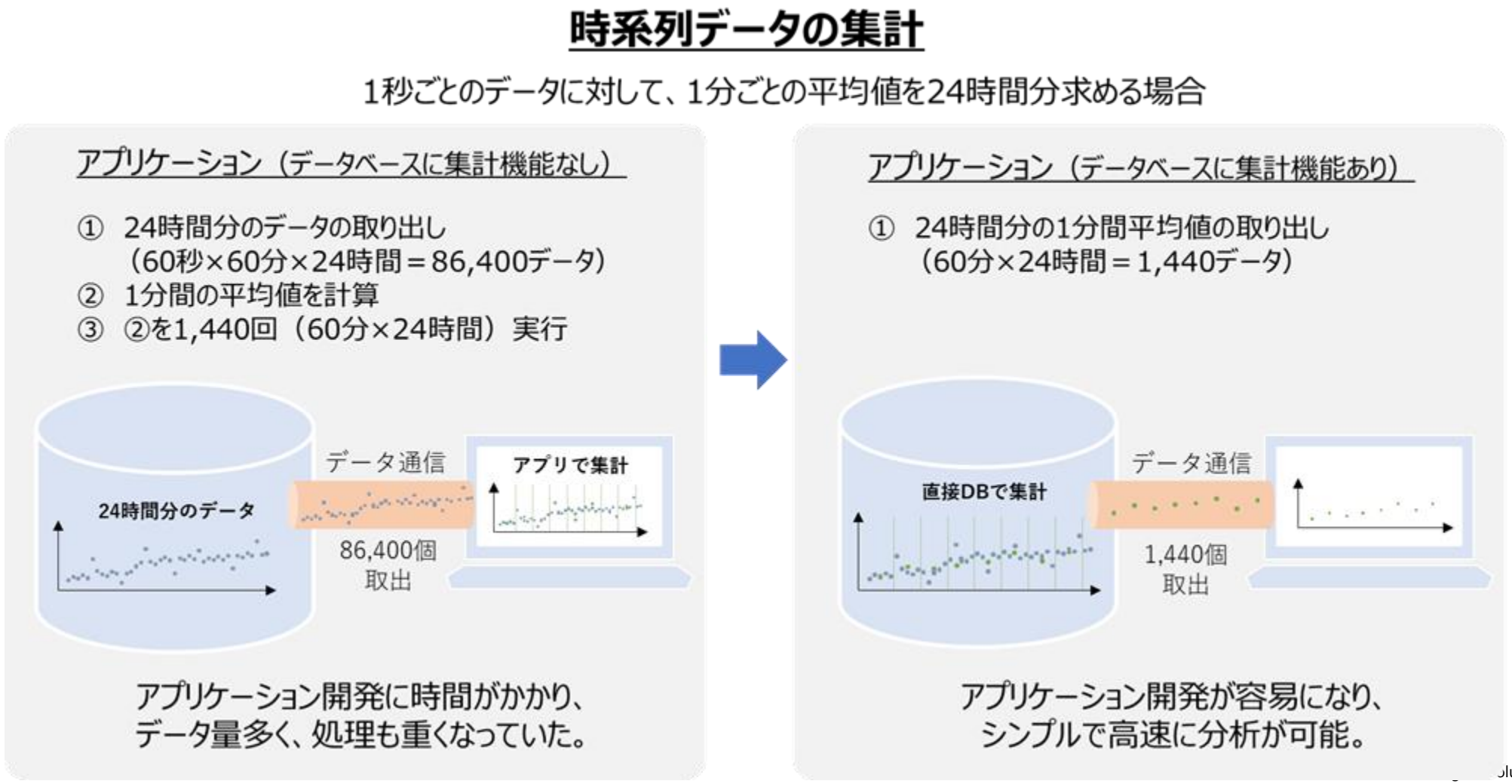
時系列機能の強化

- 高精度の時刻表現を行うための、精度指定値付TIMESTAMP型の追加
  - マイクロ秒精度、ナノ秒精度の日時表現が可能になった。
- 時系列データなど時刻値を持つデータに対する集計演算および補間演算の追加(SQL)
  - 時刻値を持つデータに対して一定時間間隔毎のデータ集合の集計演算や補間演算を行えるようになった。
  - SQLに専用の構文(GROUP BY RANGE)を追加した。

https://github.com/griddb/griddb/blob/master/docs/GridDB-5.3-CE-RELEASE_NOTES_ja.md

一定時間間隔毎のデータ集合の集計演算 (1/2)

ある一定時間間隔内のデータの最大値や最小値、平均値、合計値などを抽出する時系列データ集計機能を追加しました。これによりアプリケーション側でデータを取り出し、集計する必要がないため、アプリケーション開発が容易になり、かつ高速に分析することが可能になります。



一定時間間隔毎のデータ集合の集計演算 (2/2)

また近年はデータ収集の頻度が高まっており、保存するデータが巨大化し、コストがかかるという問題が起きています。本機能を使うことにより、長期保存用のデータを間引き（ダウンサンプリング）、保存するデータ量を大幅に削減することが可能になります。

時系列データの集計（ダウンサンプリング）

1ミリ秒ごとのデータを10ミリ秒ごとの平均値にダウンサンプリングする場合

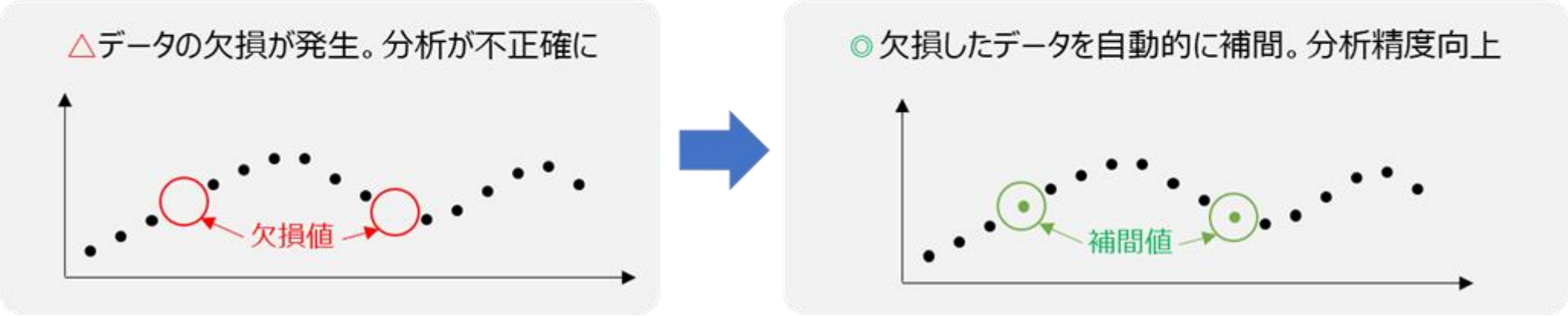
1ミリ秒ごとにデータを収集し詳細に分析

長期保存時は10ミリ秒ごとの平均値に置き換え、
データ量を1/10にダウンサンプリングして保存。
データ容量を大幅に削減。

一定時間間隔毎のデータ集合の補間演算

IoTデータの収集では、デバイスのセンサーや通信のエラーによりデータの欠損が発生することがあります。データの欠損があると、AIなどによる分析や予測が不正確になることがあります。そこで欠損したデータを線形補間や前値補間により自動的に補間する機能を用意しました。これによりデータの連続性が保たれ、精度の高い分析や予測が可能になります。

欠損値の補間



GROUP BY RANGE

SQLに専用の構文(GROUP BY RANGE)を追加しました。

SELECT <expr_list> FROM <table_list> WHERE <range_cond>
GROUP BY RANGE(<key>) EVERY(<interval>, <unit>[, <offset>])
[FILL(<fill_opt>)]

key: 集計・補間の基準として用いるカラムの名前。TIMESTAMP型(精度は任意)のカラムのみ記述可能である。

interval: 集計・補間間隔を示す整数値。

unit: 集計・補間間隔を示す値の単位。DAY、HOUR、MINUTE、SECOND、MILLISECOND

offset: 集計・補間間隔の開始位置オフセット。

fill_opt: 集計・補間対象のグループのうち、グループ内に演算対象の入力ロウが1件も存在しない場合の、各カラム値の補間方法の指定。LINEAR(前後カラムで線形補間)、NONE(ロウ出力なし)、NULL(NULL埋め)、PREVIOUS(前方のカラムと同値)

※ expr_listに指定できる集計関数: AVG, COUNT, MAX, MIN, SUM, TOTAL, GROUP_CONCAT, STDDEV, MEDIAN, PERCENTILE_CONTなど。

02

GridDBの利用方法

GridDBのインストール＆起動の手順 (Ubuntuの例)

【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/v5.3.0/griddb_5.3.0_amd64.deb
$ sudo dpkg -i griddb_5.3.0_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/v5.3.0/griddb-ce-cli_5.3.0_amd64.deb
$ sudo dpkg -i griddb-ce-cli_5.3.0_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
>  sqlcount false
```

※GridDBサービスの停止

```
$ systemctl stop gridstore
```

GridDBのインストール＆起動の手順 (Ubuntuの例)

【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/v5.3.0/griddb_5.3.0_amd64.deb
$ sudo dpkg -i griddb_5.3.0_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/v5.3.0/griddb-ce-cli_5.3.0_amd64.deb
$ sudo dpkg -i griddb-ce-cli_5.3.0_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
>  sqlcount false
```

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る。

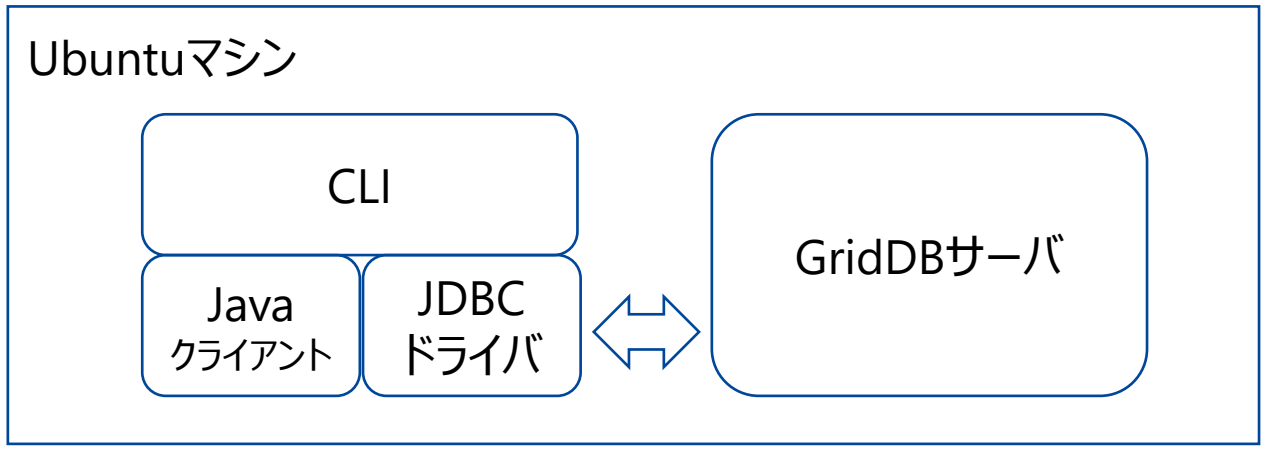
※GridDBサービスの停止

```
$ systemctl stop gridstore
```

＜動作環境の前提条件＞

- OSはUbuntu 22.04。Java 11(デフォルト)インストール済

- 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、Javaクライアント：https://github.com/griddb/griddb
※GridDB JDBCドライバ： https://github.com/griddb/jdbc
※GridDB CLI：https://github.com/griddb/cli

実行例１（SQL基本）

```
# 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
```

※SQL文の先頭が下記文字列のいずれかである場合、コマンド名sqlを省略することができます。
select update insert replace delete create drop alter grant revoke pragma explain

実行例２（テーブルパーティショニング）：テーブル作成

装置

id	type	floor	room_no

```
CREATE TABLE equipTable (
  id INTEGER PRIMARY KEY, -- 装置ID
  type STRING,   -- 装置タイプ
  floor INTEGER, -- 設置階
  room_no INTEGER -- 設置ルームNo
);
```

センサデータ

date	id	value
インターバルハッシュパーティショニング： 分割幅30日、サブパーティション数6 パーティション解放：60日		

```
CREATE TABLE sensorTable (
  date TIMESTAMP, -- 日時
  id INTEGER, -- 装置ID
  value DOUBLE, -- センサ値
  PRIMARY KEY(date, id)
) WITH (
  expiration_type='PARTITION',
  expiration_time=60,
  expiration_time_unit='DAY'
) PARTITION BY RANGE (date) EVERY (30, DAY);
SUBPARTITION BY HASH(id) SUBPARTITIONS 6;
```

実行例２（テーブルパーティショニング）：データの登録

装置

id	type	floor	room_no
1	CAMERA	1	1
2	THERMO	1	1
｡｡｡			

センサデータ

date	id	value
2021-11-01T10:30:00Z	2	18.5
2021-11-01T10:30:00Z	3	20.0
｡｡｡		

```
INSERT INTO equipTable VALUES(1, 'CAMERA', 1, 1);
INSERT INTO equipTable VALUES(2, 'THERMO', 1, 1);
INSERT INTO equipTable VALUES(3, 'THERMO', 4, 3);
INSERT INTO equipTable VALUES(4, 'THERMO', 6, 2);
INSERT INTO equipTable VALUES(5, 'WATT', 1, 1);
INSERT INTO equipTable VALUES(6, 'WATT', 6, 1);
```

```
INSERT INTO sensorTable
  VALUES(TIMESTAMP('2021-11-01T10:30:00Z'), 2, 18.5);
INSERT INTO sensorTable
  VALUES(TIMESTAMP('2021-11-01T10:30:00Z'), 3, 20.0);
｡｡｡
```

実行例２（テーブルパーティショニング）：メタテーブルに関する検索

・メタテーブル：GridDBの管理用のメタデータを参照することができるテーブル群

```
# テーブル情報の取得
SELECT * from "#tables";

# パーティショニング情報の取得
SELECT * from "#table_partitions";

# 索引情報の取得
SELECT * from "#index_info";
```

その他に、ビュー情報(#views)、実行中のSQL情報(#sqls)、実行中イベント情報(#events)、コネクション情報(sockets)、データベース一覧(databases)、データベース統計情報(database_stats)がある。

実行例３（マイクロ秒･ナノ秒）：テーブル作成、データの登録、検索

```
# テーブル作成
CREATE TABLE tbl1 (
  time TIMESTAMP(9) PRIMARY KEY,
  productName STRING,
  value INTEGER,
  time2 TIMESTAMP(6)
);

# データの登録
INSERT INTO tbl1
VALUES(TIMESTAMP_NS('2023-05-18T10:41:26.123456789Z'),
'display', 1, TIMESTAMP_NS('2023-05-18T10:41:26.123456789Z'));

# 検索
SELECT CAST(time AS STRING),
productName,value,CAST(time2 AS STRING) from tbl1;
```

tbl1

time	product Name	value	time2

※ TIMESTAMP(p)の形式で精度を指定してください。
TIMESTAMP or TIMESTAMP(3): ミリ秒精度
TIMESTAMP(6): マイクロ秒精度
TIMESTAMP(9): ナノ秒精度

```
# 検索結果
2023-05-18T10:41:26.123456789Z,‘display’,1,2023-05-18T10:41:26.123456Z
```

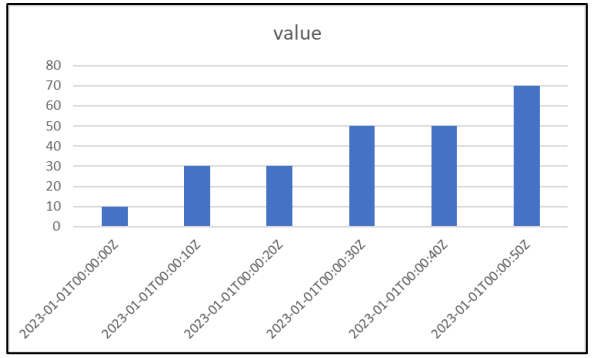
実行例４（GroupByRangeによる集約）：テーブル作成、データの登録、検索

```
# テーブル作成
CREATE TABLE trend_data1 (
  ts TIMESTAMP PRIMARY KEY,
  value INTEGER
);
```

trend_data1

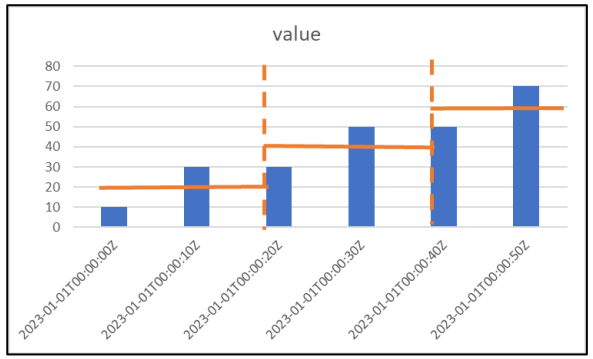
ts	value

```
# データの登録
INSERT INTO trend_data1 VALUES(TIMESTAMP('2023-01-01T00:00:00Z'), 10);
INSERT INTO trend_data1 VALUES(TIMESTAMP('2023-01-01T00:00:10Z'), 30);
INSERT INTO trend_data1 VALUES(TIMESTAMP('2023-01-01T00:00:20Z'), 30);
INSERT INTO trend_data1 VALUES(TIMESTAMP('2023-01-01T00:00:30Z'), 50);
INSERT INTO trend_data1 VALUES(TIMESTAMP('2023-01-01T00:00:40Z'), 50);
INSERT INTO trend_data1 VALUES(TIMESTAMP('2023-01-01T00:00:50Z'), 70);
```




```
# 検索
SELECT ts,avg(value) FROM trend_data1
WHERE
  ts BETWEEN TIMESTAMP('2023-01-01T00:00:00Z') AND
  TIMESTAMP('2023-01-01T00:00:50Z')
GROUP BY RANGE (ts) EVERY (20,SECOND);
```

```
# 検索結果
ts                      |
------------------------+-----
2023-01-01 00:00:00.000Z |   20
2023-01-01 00:00:20.000Z |   40
2023-01-01 00:00:40.000Z |   60
```



実行例５（GroupByRangeによる補間）：テーブル作成、データの登録、検索

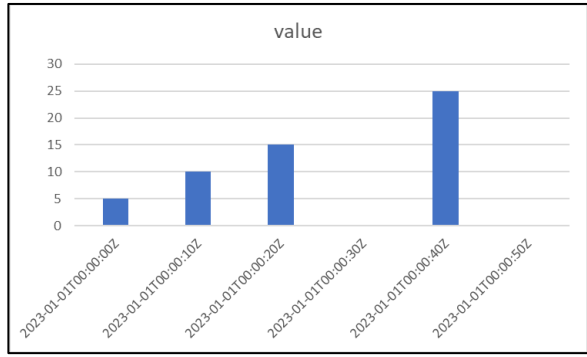
```
# テーブル作成
CREATE TABLE trend_data2 (
  ts TIMESTAMP PRIMARY KEY,
  value INTEGER
);
```

trend_data2

ts	value

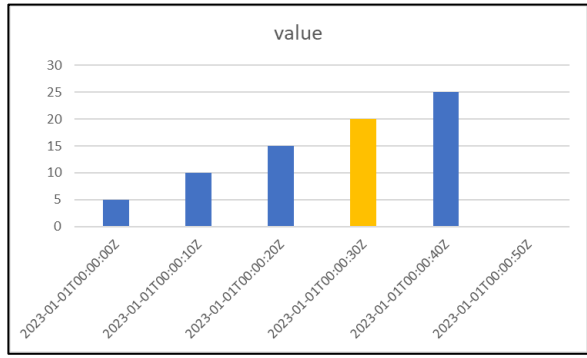
```
# データの登録
INSERT INTO trend_data2 VALUES(TIMESTAMP('2023-01-01T00:00:00Z'), 5);
INSERT INTO trend_data2 VALUES(TIMESTAMP('2023-01-01T00:00:10Z'), 10);
INSERT INTO trend_data2 VALUES(TIMESTAMP('2023-01-01T00:00:20Z'), 15);

INSERT INTO trend_data2 VALUES(TIMESTAMP('2023-01-01T00:00:40Z'), 25);
```




```
# 検索
SELECT * FROM trend_data2
WHERE
  ts BETWEEN TIMESTAMP('2023-01-01T00:00:00Z') AND
  TIMESTAMP('2023-01-01T00:00:40Z')
GROUP BY RANGE (ts) EVERY (10,SECOND) FILL (LINEAR);
```

```
# 検索結果
ts                   | value
---------------------+------
2023-01-01 09:00:00 |     5
2023-01-01 09:00:10 |    10
2023-01-01 09:00:20 |    15
2023-01-01 09:00:30 |    20
2023-01-01 09:00:40 |    25
```



03

OSS活動

主なOSS活動

① GridDB本体の機能強化

② 主要OSSとの連携強化

③ APIの拡充

④ GitHub以外のサイトからの情報発信

- パッケージ
- デベロッパーズサイト（WP、ブログなど）
- SNS

⑤ 主要OSSリポジトリへのコントリビュート

⑥ プラットフォームの拡充

⑦ その他

- OSCなどカンファレンス参加

OSS活動の全体イメージ

性能測定 | 収集 | 分散処理 | 可視化 | Webアプリ | 分析 | AI/機械学習 …

④GitHub以外のサイトからの情報発信
PyPI/npm/Maven/Packagist/…

②主要OSSとの連携強化

Spark コネクタ
Fluentd/Grafana/Redash プラグイン
YCSB コネクタ
Kafka コネクタ
Hadoop MapReduce コネクタ

⑥プラットフォームの拡充
CentOS＋
Ubuntu、openSUSE
Windows、MacOS
Docker

WebAPI
Python/Node.JS/Go/PHP/Ruby/Perl/Rustクライアント

③APIの拡充

Javaクライアント
JDBCドライバ
Cクライアント

①GridDB本体の機能強化
GridDB V5.3 CE(Community Edition)

https://github.com/griddb/
GitHub

⑤主要OSSリポジトリへのコントリビュート

デベロッパーズサイト

https://griddb.net/

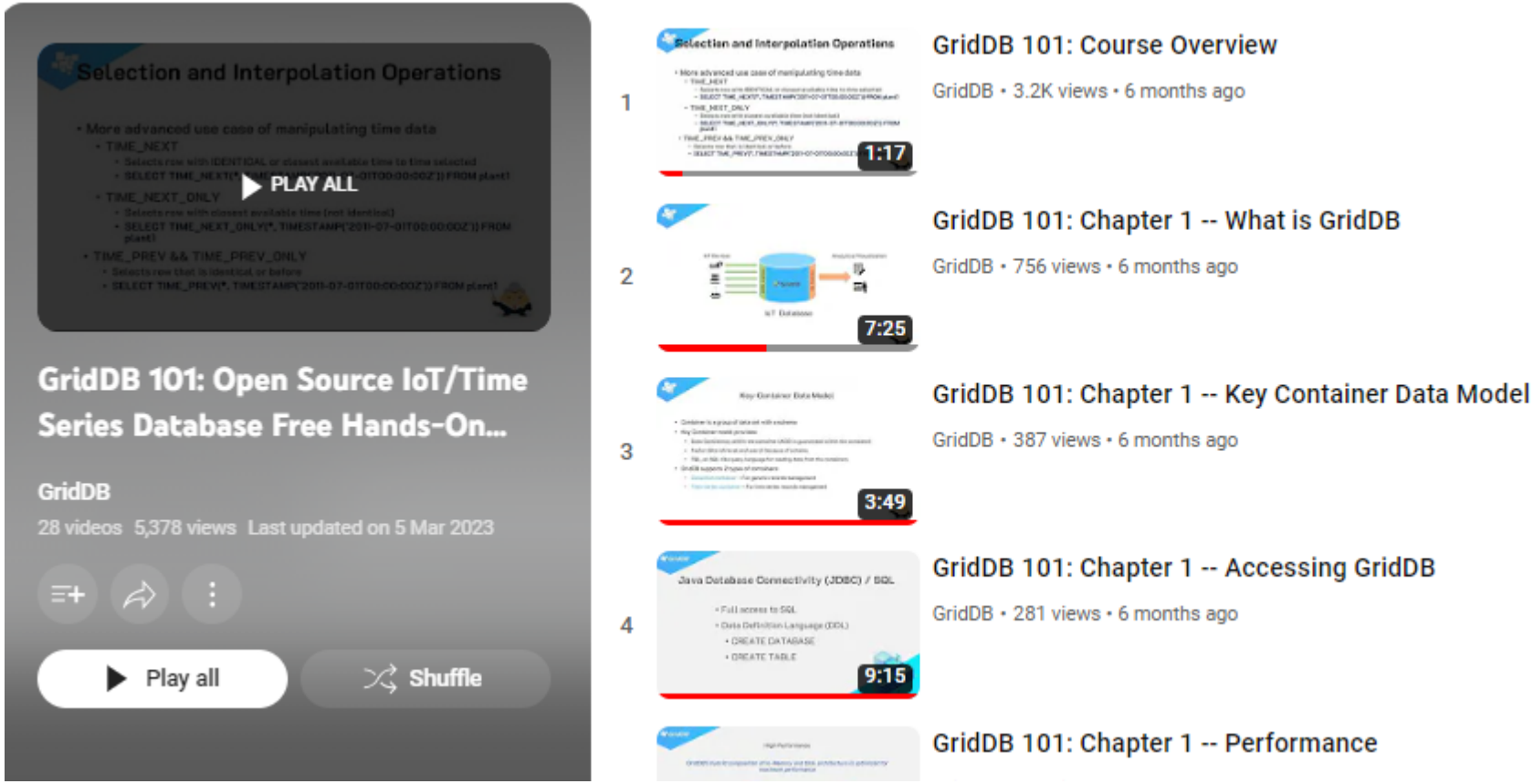
griddb net	検索

- **アプリケーション開発者向けのサイト**
- **様々なコンテンツを公開**
  - ホワイトペーパ
  - ブログ

  など

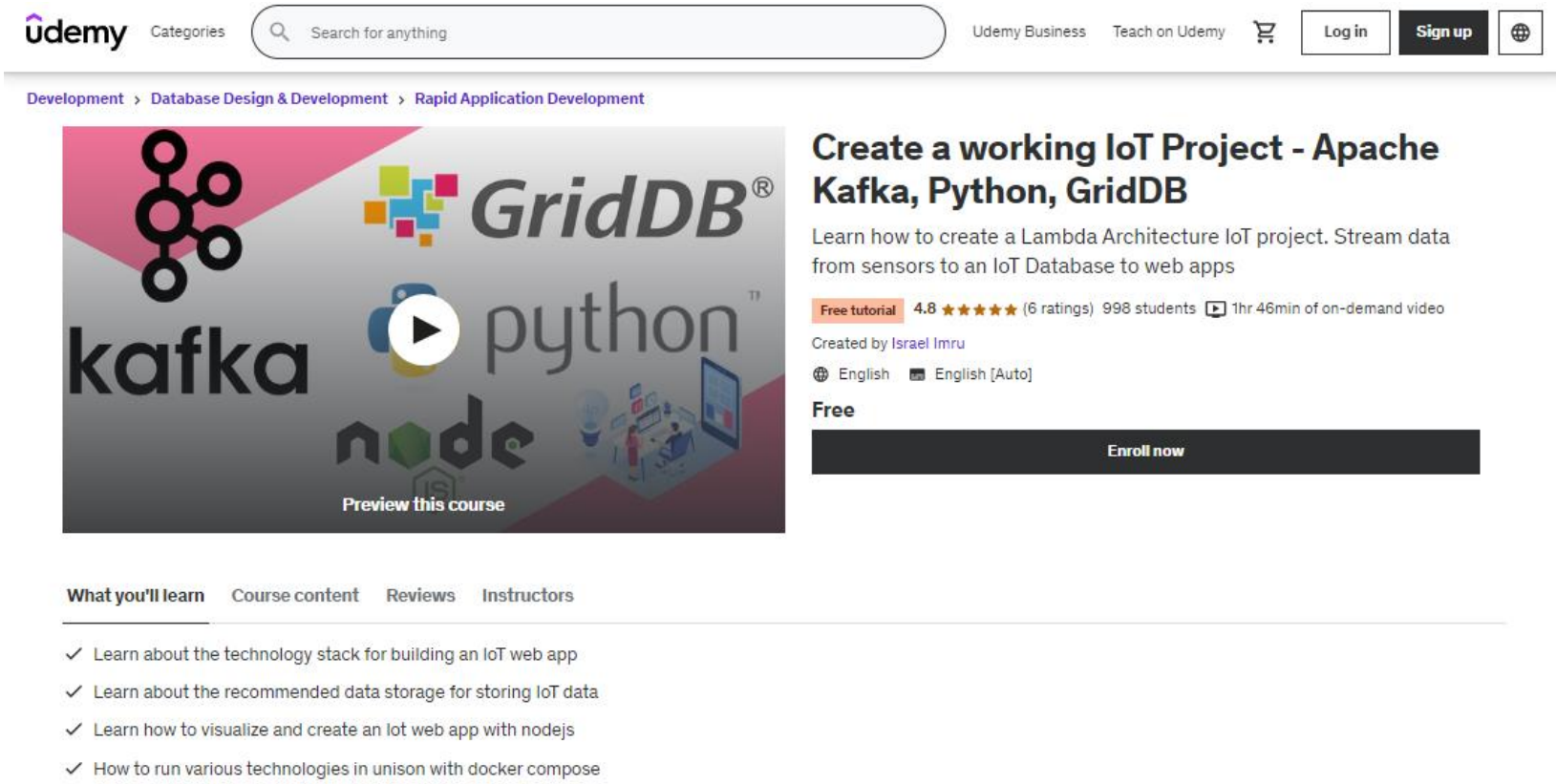
GridDBオンデマンド教育ビデオ（Youtube）

- GridDB 101: Open Source IoT/Time Series Database Free Hands-On Basic Beginner Course（https://www.youtube.com/playlist?list=PLZiizI6Euect9q64akYBkiqLMS78UTwjO）
- GridDBに関する28個のオンデマンドビデオをYoutubeに公開



GridDBオンデマンド教育ビデオ（Udemy）

- GridDBとKafkaを利用し、GridDBの中級コースをUdemyで公開（https://www.udemy.com/course/create-a-working-iot-project-with-iot-database-griddb/）
- 80か国で900名以上の受講者



X（旧ツイッター）

https://twitter.com/griddb_jp

twitter griddb	検索

- GridDBに関するリリース、イベント、などをお知らせします。
  (日本国内向け)

05

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、強化された時系列データ管理・検索機能とその使い方について、オープンソース活動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/

TOSHIBA

付録

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		✓	✓
	CLIツール	✓	✓	✓
セキュリティ	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 デベロッパーズサイト**
  - https://griddb.net/
- **Twitter: GridDB (日本)**
  - https://twitter.com/griddb_jp
- **Twitter: GridDB Community**
  - https://twitter.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

griddb github	検索

griddb net	検索

twitter griddb	検索

（ご参考）マイクロ秒・ナノ秒：テーブル作成、データの登録、検索　(JavaAPI)

テーブル作成

```
// (2)カラムの名前やデータ型をカラム情報オブジェクトにセット
List<ColumnInfo> columnList = new ArrayList<ColumnInfo>();
columnList.add(new ColumnInfo.Builder(
        new ColumnInfo("time", GSType.TIMESTAMP))
        .setTimePrecision(TimeUnit.NANOSECOND).toInfo());
columnList.add(new ColumnInfo("productName", GSType.STRING));
columnList.add(new ColumnInfo("value", GSType.INTEGER));
columnList.add(new ColumnInfo.Builder(
        new ColumnInfo("time2", GSType.TIMESTAMP))
        .setTimePrecision(TimeUnit.MICROSECOND).toInfo());
```

データの登録

```
//(3) ロウオブジェクトにカラム値をセット
row.setPreciseTimestamp(0, ts);
row.setString(1, "display");
row.setInteger(2, 1);
row.setPreciseTimestamp(3, ts);
```

検索

```
// (3)ロウからカラムの値を取り出す
Timestamp ts1 = row.getPreciseTimestamp(0);
String name = row.getString(1);
int value = row.getInteger(2);
Timestamp ts2 = row.getPreciseTimestamp(3);
```

https://github.com/griddb/griddb/blob/master/sample/guide/ja/NanoTimestamp.java

（ご参考）マイクロ秒・ナノ秒：テーブル作成、データの登録、検索 （C API)

```
# テーブル作成
        // (3)カラム情報を定義する
        columnInfo.name = "time";
        columnInfo.type = GS_TYPE_TIMESTAMP;
        columnInfo.options = GS_TYPE_OPTION_TIME_NANO;
        columnInfoList[0] = columnInfo;

        columnInfo.name = "productName";
        columnInfo.type = GS_TYPE_STRING;
        columnInfo.options = 0;
        columnInfoList[1] = columnInfo;

        columnInfo.name = "value";
        columnInfo.type = GS_TYPE_INTEGER;
        columnInfo.options = 0;
        columnInfoList[2] = columnInfo;

        columnInfo.name = "time2";
        columnInfo.type = GS_TYPE_TIMESTAMP;
        columnInfo.options = GS_TYPE_OPTION_TIME_MICRO;
        columnInfoList[3] = columnInfo;
```

```
# データの登録
        gsParsePreciseTime("2023-05-18T10:41:26.101123456Z", &pts, &opt);

        // (3)カラム値をセット
        gsSetRowFieldByPreciseTimestamp(row, 0, &pts);
        gsSetRowFieldByString(row, 1, "display");
        gsSetRowFieldByInteger(row, 2, 1);
        gsSetRowFieldByPreciseTimestamp(row, 3, &pts);
# 検索
        // (4)ロウから値を取得
        gsGetRowFieldAsPreciseTimestamp(row, 0, &pts1);
        gsGetRowFieldAsString(row, 1, &productName);
        gsGetRowFieldAsInteger(row, 2, &value);
        gsGetRowFieldAsPreciseTimestamp(row, 3, &pts2);

        gsFormatPreciseTime(&pts1, buf, GS_TIME_STRING_SIZE_MAX, &opt);
        printf("time=%s, ", buf);
        printf("productName=%s, value=%d, ", productName, value);
        gsFormatPreciseTime(&pts2, buf, GS_TIME_STRING_SIZE_MAX, &opt);
        printf("time2=%s)¥n", buf);
```

https://github.com/griddb/c_client/blob/master/sample/guide/ja/NanoTimestamp.c

（ご参考）エクスポート・インポートツール

https://github.com/griddb/expimp

(ダウンロード・ビルド)
#apt-get install -y unzip vim
wget https://github.com/griddb/expimp/archive/refs/tags/v5.3.0.zip
unzip v5.3.0.zip
cd expimp-5.3.0/expimp-ce
./gradlew shadowJar
cd ../bin
(設定)
gs_expimp.properties編集
    clusterName=myCluster
    notificationMember=127.0.0.1:10001
    jdbcNotificationMember=127.0.01.:20001

(エクスポート実行)
./gs_export -u admin/admin -d out -c t1
(日付範囲指定のエクスポート実行)
時系列コンテナもしくはパーティショニングキーが日時型のインターバル（ハッシュ）パーティショニングテーブルの場合
./gs_export -u admin/admin -d out5 -c testTable --intervals 20230918:20230920

(インポート実行)
./gs_import –u admin/admin –d ../impSample/collection -all

ご参考：

- SQL（テーブルパーティショニング）の例
  - ✓ https://github.com/knonomura/griddb-docker/blob/master/SQLSamples.md
  - ✓ https://github.com/knonomura/griddb-docker/blob/master/SQLSamples2.md

- NoSQLインタフェースでバッチ処理等を使いたい場合
  - ✓ https://github.com/griddb/griddb/tree/master/sample/guide/ja

  のSampleMultiPut.javaなどを参照願います。

- DockerでGridDBを使いたい場合
  - ✓ https://github.com/griddb/griddb-dockerのDockerfile
  - ✓ https://hub.docker.com/u/griddbのDockerイメージ

  を参照願います。