

ストリームデータ処理基盤

uCosminexus Stream Data Platform - Application Framework アプリケーショ ン開発ガイド

手引・文法書

3020-3-V03-20

■ 対象製品

●適用 OS : Windows Server 2008 R2 Standard, Windows Server 2008 R2 Enterprise, Windows Server 2008 R2 Datacenter

P-2964-9B14 uCosminexus Stream Data Platform - Application Framework 01-05**

●適用 OS : Red Hat Enterprise Linux 5, Red Hat Enterprise Linux 6

P-9W64-9B11 uCosminexus Stream Data Platform - Application Framework 01-05

●適用 OS : Red Hat Enterprise Linux 6

P-9W64-9V11 uCosminexus Stream Data Platform - Application Framework 01-50

注※ この製品については、サポート時期をご確認ください。

これらのプログラムプロダクトのほかにもこのマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

BSAFE は、EMC Corporation の米国およびその他の国における登録商標または商標です。

Microsoft は、米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

Linux は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。

Oracle と Java は、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。

Red Hat は、米国およびその他の国で Red Hat, Inc. の登録商標もしくは商標です。

RSA は、EMC Corporation の米国およびその他の国における登録商標または商標です。

Windows は、米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

Windows Server は、米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

その他記載の会社名、製品名は、それぞれの会社の商標もしくは登録商標です。

Portions of this software were developed at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England. The original software is available from <ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/>

This product includes software developed by Andy Clark.

This product includes software developed by Ben Laurie for use in the Apache-SSL HTTP server project.

This product includes software developed by Daisuke Okajima and Kohsuke Kawaguchi (<http://relaxngcc.sf.net/>).

This product includes software developed by IAIK of Graz University of Technology.

This product includes software developed by Ralf S. Engelschall <rse@engelschall.com> for use in the mod_ssl project (<http://www.modssl.org/>).

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

This product includes software developed by the Java Apache Project for use in the Apache JServ servlet engine project (<http://java.apache.org/>).

This product includes software developed by the University of California, Berkeley and its contributors.

This software contains code derived from the RSA Data Security Inc. MD5 Message-Digest Algorithm, including various modifications by Spyglass Inc., Carnegie Mellon University, and Bell Communications Research, Inc (Bellcore).

uCosminexus Stream Data Platform - Application Framework は、米国 EMC コーポレーションの RSA BSAFE(R)ソフトウェアを搭載しています。

■ マイクロソフト製品の表記について

このマニュアルでは、マイクロソフト製品の名称を次のように表記しています。

表記		製品名
Windows Server 2008 または Windows	Windows Server 2008 R2 Standard	Microsoft(R) Windows Server(R) 2008 R2 Standard x64
	Windows Server 2008 R2 Enterprise	Microsoft(R) Windows Server(R) 2008 R2 Enterprise x64
	Windows Server 2008 R2 Datacenter	Microsoft(R) Windows Server(R) 2008 R2 Datacenter x64

■ 発行

2014 年 6 月 3020-3-V03-20

■ 著作権

All Rights Reserved. Copyright (C) 2010, 2014, Hitachi, Ltd.

変更内容

変更内容(3020-3-V03-20) uCosminexus Stream Data Platform - Application Framework 01-50

追加・変更内容	変更箇所
バージョンの変更に伴い、01-50 に対応した記述を追加した。	-

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルは、uCosminexus Stream Data Platform - Application Framework でデータを分析するために使用する、CQL の記述方法、およびカスタムアダプターの作成方法について説明したものです。

分析目的に合わせて CQL を記述できるようになること、および API でカスタムアダプターを作成できるようになることを目的としています。

■ 対象読者

CQL によるクエリの定義、およびアダプターとして使用するアプリケーションプログラムを作成するプログラマーの方を対象としています。

次の知識をお持ちの方を前提としています。

- OS の基礎的な知識
- Java によるプログラミングの知識

なお、このマニュアルは、マニュアル「uCosminexus Stream Data Platform - Application Framework 解説」を前提としていますので、あらかじめお読みいただくことをお勧めします。

■ 適用 OS の違いによる機能相違点の表記

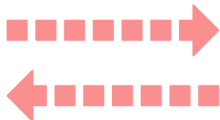
このマニュアルで説明する機能は、適用 OS の種類（Windows または Linux）によって、異なる場合があります。OS によって機能差がある場合、OS 名を明記しています。

なお、Windows のパスの区切り文字として使用している「¥」は、Linux の場合には、特に断りのないかぎり、「/」に読み替えてください。

■ 図中で使用する記号

このマニュアルの図中で使用する記号を次のように定義します。

- ストリームデータの
流れ



目次

第 1 編 お読みいただく前に

1	Stream Data Platform - AF でのアプリケーション開発の概要	1
1.1	導入から運用までの流れ	2
1.2	このマニュアルの構成	4
1.3	アプリケーション開発の概要	6
1.3.1	クエリの定義でできること	6
1.3.2	カスタムアダプターの作成でできること	7

第 2 編 CQL プログラミング

2	CQL によるクエリの定義	9
2.1	CQL の体系	10
2.2	ウィンドウ演算による入力リレーションの生成	13
2.2.1	ウィンドウ演算の種類	13
2.2.2	ウィンドウ演算の例	15
2.3	関係演算によるデータの抽出	19
2.3.1	関係演算の種類	19
2.3.2	結合処理の例	19
2.3.3	集合関数による演算処理の例	21
2.3.4	結合処理と ROWS ウィンドウを併用する場合の注意事項	23
2.4	ストリーム化演算による出力ストリームデータへの変換	24
2.4.1	ストリーム化演算の種類	24
2.4.2	ストリーム化演算の例	25
2.5	時刻解像度の指定によるメモリ使用量増加の抑止	29
2.6	定義例	31
2.6.1	基本的なクエリの定義例	31
2.6.2	時刻解像度を指定した定義例	32

3	CQL の基本項目とデータ型	35
3.1	CQL によるクエリ定義	36
3.1.1	CQL の記述形式	36
3.1.2	CQL で使用できる文字	37
3.1.3	CQL 文法説明で使用する記号	38
3.2	CQL の基本項目の指定方法	39

3.2.1	キーワードの指定	39
3.2.2	数値の指定	40
3.2.3	区切り文字の挿入	40
3.2.4	名前の指定	42
3.2.5	名前の修飾	42
3.2.6	定数の指定	44
3.3	CQL のデータ型	49
3.3.1	CQL のデータ型と Java のデータ型のマッピング	49
3.3.2	DECIMAL 型および NUMERIC 型についての注意事項	51
3.4	データの比較	53
3.4.1	比較できるデータ型の組み合わせ	53
3.4.2	データを比較する際の留意点	53
3.5	クエリ定義での注意事項および制限値	55
3.5.1	クエリ定義での注意事項	55
3.5.2	クエリ定義での制限値	55

4

CQL リファレンス	57	
4.1	CQL 文法説明で使用する記述形式	58
4.2	CQL の一覧	59
4.3	定義系 CQL	61
4.3.1	REGISTER STREAM 句 (ストリームの定義)	61
4.3.2	REGISTER QUERY 句 (クエリの定義)	62
4.3.3	REGISTER QUERY_ATTRIBUTE 句 (時刻解像度の指定)	63
4.4	操作系 CQL	67
4.4.1	問い合わせ	67
4.4.2	ストリーム句	68
4.4.3	リレーション式	69
4.4.4	SELECT 句	69
4.4.5	FROM 句	70
4.4.6	WHERE 句	71
4.4.7	GROUP BY 句	72
4.4.8	HAVING 句	72
4.4.9	UNION 句	73
4.4.10	選択リスト	74
4.4.11	選択式	75
4.4.12	列指定リスト	76
4.4.13	リレーション参照	77
4.4.14	ウィンドウ指定	79
4.4.15	時間指定	80
4.4.16	探索条件	81

4.4.17	比較述語	83
4.4.18	値式	84
4.4.19	定数	87
4.4.20	集合関数	88
4.4.21	組み込み集合関数	90
4.4.22	スカラー関数	91
4.4.23	組み込みスカラー関数	91
4.4.24	ストリーム間演算関数	92
4.4.25	外部定義ストリーム間演算関数	93

5

CQL で指定する組み込み関数 95

5.1	組み込み関数の一覧	96
5.2	統計関数の詳細	99
	統計関数	99
	CORREL 関数	99
	COVAR 関数	100
	COVAR_POP 関数	101
	STDDEV 関数	102
	STDDEV_POP 関数	103
	VAR 関数	104
	VAR_POP 関数	105
5.3	数学関数の詳細	106
	数学関数	106
	ABS 関数	108
	ACOS 関数	109
	ASIN 関数	109
	ATAN 関数	111
	ATAN2 関数	112
	CEIL 関数	116
	COS 関数	117
	COSH 関数	118
	DISTANCE 関数	119
	DISTANCE3 関数	120
	EXP 関数	121
	FLOOR 関数	122
	LN 関数	123
	LOG 関数	124
	MOD 関数	127
	NAN 関数	127
	NEGATIVE_INFINITY 関数	128

PI 関数	128
POSITIVE_INFINITY 関数	129
POWER 関数	129
ROUND 関数	138
SIN 関数	139
SINH 関数	140
SQRT 関数	141
TAN 関数	142
TANH 関数	143
TODEGREES 関数	144
TORADIANS 関数	145
5.4 文字列関数の詳細	147
文字列関数	147
CONCAT 関数	148
LENGTH 関数	149
LEQ 関数	150
LGE 関数	151
LGT 関数	152
LLE 関数	152
LLT 関数	153
REGEXP_FIRSTSEARCH 関数	154
REGEXP_REPLACE 関数	155
REGEXP_SEARCH 関数	156
SLENGTH 関数	157
5.5 時刻関数の詳細	158
時刻関数	158
TIMEDIFF 関数	158
TIMESTAMPDIFF 関数	159
5.6 変換関数の詳細	161
変換関数	161
BIGINT_TOSTRING 関数	162
DECIMAL_TOSTRING 関数	162
DOUBLE_TOSTRING 関数	163
INT_TOSTRING 関数	164
NUMBER_TODATE 関数	164
NUMBER_TOTIME 関数	165
REAL_TOSTRING 関数	166
TIME_TONUMBER 関数	166
TIMESTAMP_TONUMBER 関数	167

6	クエリ定義のサンプル	169
6.1	クエリ定義のサンプルの種類	170

第3編 カスタムアダプター作成

7	カスタムアダプターの作成	171
7.1	作成するカスタムアダプターの種類	172
7.1.1	データ送信 AP とデータ受信 AP	172
7.1.2	データの送受信方法による分類 (RMI 連携カスタムアダプターとインプロセス連携カスタムアダプター)	172
7.2	RMI 連携カスタムアダプターの作成	175
7.2.1	ストリームデータの送信 (RMI 連携カスタムアダプター)	175
7.2.2	クエリ結果データの受信 (RMI 連携カスタムアダプター)	176
7.3	インプロセス連携カスタムアダプターの作成	178
7.3.1	ストリームデータの送信 (インプロセス連携カスタムアダプター)	178
7.3.2	クエリ結果データの受信 (インプロセス連携カスタムアダプター)	179
7.3.3	注意事項	181
7.4	カスタムアダプターで実装する処理	182
7.4.1	データ受信 AP の終了契機の把握 (データ処理終了の通知)	182
7.4.2	データソースモードでの時刻情報の設定	185
7.4.3	キューあふれの事前防止	188
7.5	コンパイル手順	195
7.6	カスタムアダプター作成時の留意事項	196
8	データ送受信 API	197
8.1	データ送受信 API の記述形式	198
8.2	データ送受信 API の一覧	199
8.3	SDPConnector インタフェース (共通 API)	201
	close()メソッド	202
	isClosed()メソッド	202
	openStreamInput(String group_name,String stream_name)メソッド	203
	openStreamOutput(String group_name,String stream_name)メソッド	203
8.4	SDPConnectorFactory クラス (RMI 連携用)	205
	connect()メソッド	205
8.5	StreamEventListener インタフェース (インプロセス連携用)	206
	onEvent(StreamTuple tuple)メソッド	206
8.6	StreamInprocessUP インタフェース (インプロセス連携用)	207
	execute(SDPConnector con)メソッド	208

stop()メソッド	208
8.7 StreamInput インタフェース (共通 API)	210
close()メソッド	210
getFreeQueueSize()メソッド	211
getMaxQueueSize()メソッド	211
isStarted()メソッド	212
put(ArrayList<StreamTuple> tuple_list)メソッド	213
put(StreamTuple tuple)メソッド	214
putEnd()メソッド	215
8.8 StreamOutput インタフェース (共通 API)	216
close()メソッド	217
get()メソッド	217
get(int count)メソッド	218
get(int count, long timeout)メソッド	219
getAll()メソッド	221
getAll(long timeout)メソッド	222
getFreeQueueSize()メソッド	223
getMaxQueueSize()メソッド	224
registerForNotification(StreamEventListener n)メソッド	225
unregisterForNotification(StreamEventListener n)メソッド	225
8.9 StreamTime クラス (共通 API)	227
equals(StreamTime when)メソッド	227
getTimeMillis()メソッド	228
hashCode()メソッド	228
toString()メソッド	229
8.10 StreamTuple クラス (共通 API)	230
StreamTuple(Object[] dataArray)コンストラクター	231
equals(Object obj)メソッド	231
getDataArray()メソッド	232
getSystemTime()メソッド	232
hashCode()メソッド	233
toString()メソッド	233
8.11 例外クラス (共通 API)	234
SDPClientException クラス (共通 API)	234
SDPClientFreeInputQueueSizeThresholdOverException クラス (共通 API)	235
SDPClientFreeInputQueueSizeLackException クラス (共通 API)	236
9 データ送受信 API を使用したサンプルプログラム	237
9.1 サンプルプログラムの構成	238
9.2 RMI 連携カスタムアダプターのサンプルプログラム	240

9.2.1	RMI 連携カスタムアダプターのサンプルプログラムの実行手順	240
9.2.2	クエリグループの定義内容 (RMI 連携カスタムアダプター)	241
9.2.3	RMI 連携データ送信 AP の内容	241
9.2.4	RMI 連携データ受信 AP の内容	243
9.3	インプロセス連携カスタムアダプターのサンプルプログラム	246
9.3.1	インプロセス連携カスタムアダプターのサンプルプログラムの実行手順	246
9.3.2	クエリグループの定義内容 (インプロセス連携カスタムアダプター)	247
9.3.3	インプロセス連携送受信制御 AP の内容	247
9.3.4	インプロセス連携データ送信 AP の内容	249
9.3.5	インプロセス連携データ受信 AP の内容 (ポーリング方式)	251

第 4 編 外部定義関数作成

10	外部定義関数の作成	253
10.1	外部定義関数の概要	254
10.2	外部定義関数の実装	255
10.2.1	外部定義関数のクラスの実装	255
10.2.2	外部定義関数のメソッドの実装	257
10.2.3	外部定義関数の変更	259
10.3	外部定義関数の作成例	260

11	外部定義関数インタフェース	263
11.1	外部定義関数インタフェースの記述形式	264
11.2	外部定義関数インタフェースの一覧	265
11.3	SDPEExternalStreamFunction インタフェース	266
	executeStreamFunc(Collection<Object[]>[] ObjectArrayCollection, Timestamp timestamp)メソッド	266
	initialize()メソッド	267
	terminate()メソッド	267

付録		269
付録 A	各バージョンの変更内容	270
付録 B	このマニュアルの参考情報	271
付録 B.1	関連マニュアル	271
付録 B.2	このマニュアルでの表記	271
付録 B.3	英略語	272
付録 B.4	KB (キロバイト) などの単位表記について	272

1

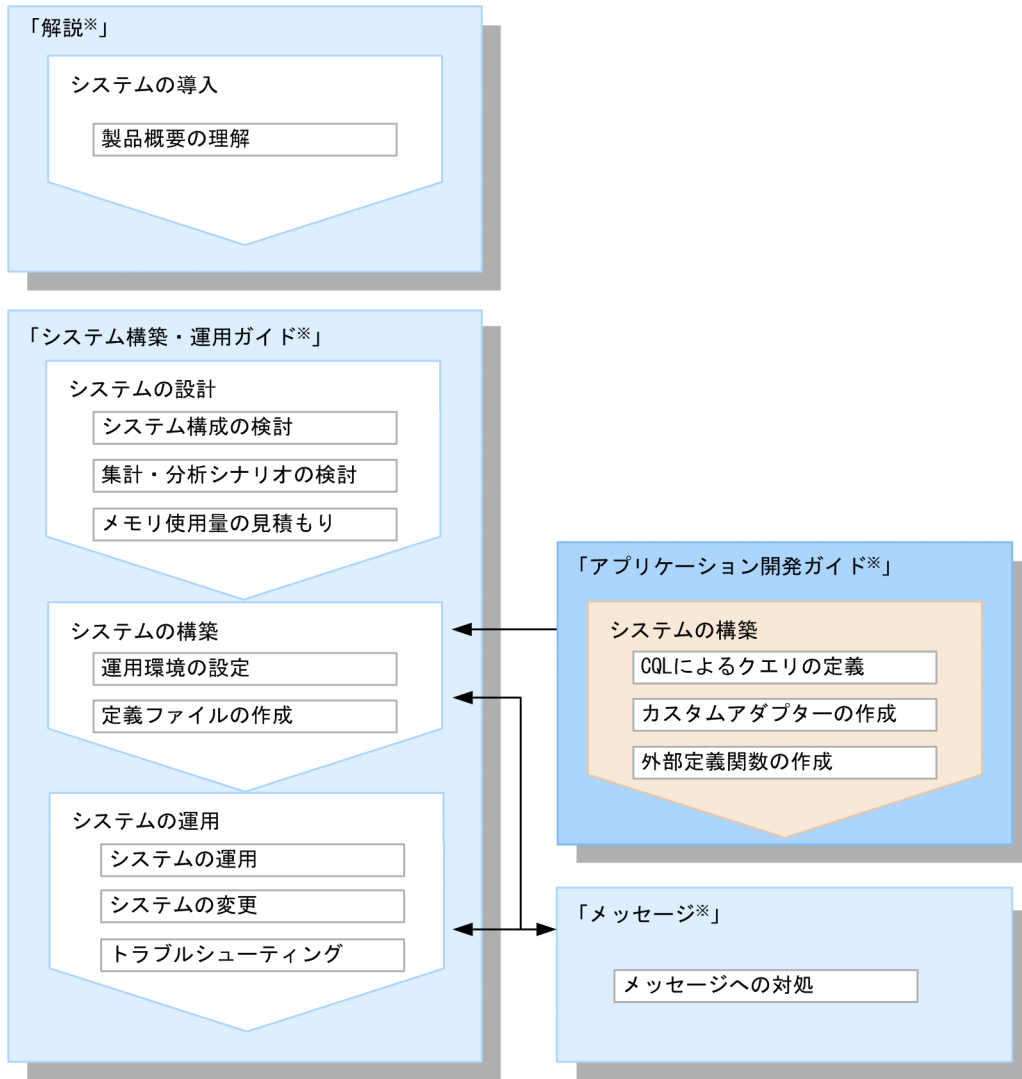
Stream Data Platform - AF での アプリケーション開発の概要

この章では、このマニュアルを読み進める前に知っておいていただきたい内容として、Stream Data Platform - AF の導入から運用までの流れ、このマニュアルの構成、および Stream Data Platform - AF でのアプリケーション開発の概要について説明します。

1.1 導入から運用までの流れ

Stream Data Platform - AF の導入から運用までの流れと関連マニュアルとの関係を、次の図に示します。

図 1-1 導入から運用までの流れと関連マニュアルとの関係



(凡例)

- : このマニュアル
- : ほかのシリーズマニュアル
- : 作業フェーズ
- : ユーザーの作業項目
- : 必要に応じて参照

注※ マニュアル名称のうち、「uCosminexus Stream Data Platform - Application Framework」は省略して表記しています。

このマニュアルでは、「システムの構築」作業のうち、「CQLによるクエリの定義」、「カスタムアダプターの作成」および「外部定義関数の作成」について説明します。

なお、このマニュアルを読む前に、マニュアル「uCosminexus Stream Data Platform - Application Framework 解説」を参照して、Stream Data Platform - AF の製品概要を理解しておいてください。

また、必要に応じて、各作業フェーズで関連マニュアルを参照してください。

1.2 このマニュアルの構成

このマニュアルの構成を紹介します。このマニュアルは、次のように四つの編と付録で構成されています。

- 第1編：お読みいただく前に
この編です。このマニュアルを読み進める前に知っておいていただきたい内容（Stream Data Platform - AF の導入から運用までの流れ、このマニュアルの構成、および Stream Data Platform - AF でのアプリケーション開発の概要）について説明しています。
- 第2編：CQL プログラミング
CQL によるクエリの定義方法、クエリ定義で使用する CQL の基本項目、CQL の文法などについて説明しています。
- 第3編：カスタムアダプター作成
カスタムアダプター作成時の実装内容、コンパイル方法、Stream Data Platform - AF が提供する API の文法などについて説明しています。
- 第4編：外部定義関数作成
外部定義関数の作成方法、外部定義関数インタフェースの文法について説明しています。
- 付録
このマニュアルの参考情報について説明しています。

それぞれの編に含まれる章と付録の記載内容について、次の表で説明します。

表 1-1 各章と付録の記載内容

編	章・付録	記載内容
第1編 お読みいただく前に	第1章 Stream Data Platform - AF でのアプリケーション開発の概要	この章です。このマニュアルを読み進める前に知っておいていただきたい内容について説明しています。
第2編 CQL プログラミング	第2章 CQL によるクエリの定義	CQL によるクエリの定義方法について説明しています。
	第3章 CQL の基本項目とデータ型	CQL の基本項目とデータ型について説明しています。
	第4章 CQL リファレンス	CQL の文法について説明しています。
	第5章 CQL で指定する組み込み関数	CQL で指定する組み込み関数の文法について説明しています。
	第6章 クエリ定義のサンプル	クエリ定義のサンプルについて説明しています。
第3編 カスタムアダプター作成	第7章 カスタムアダプターの作成	カスタムアダプターの作成方法について説明しています。
	第8章 データ送受信 API	カスタムアダプターの作成で使用するデータ送受信 API の文法について説明しています。
	第9章 データ送受信 API を使用したサンプルプログラム	データ送受信 API を使用したサンプルプログラムについて説明しています。
第4編 外部定義関数作成	第10章 外部定義関数の作成	外部定義関数の作成方法について説明しています。

編	章・付録	記載内容
第 4 編 外部定義関数作成	第 11 章 外部定義関数インターフェース	外部定義関数の作成で使用する外部定義関数インターフェースの文法について説明しています。
付録	付録 A このマニュアルの参考情報	このマニュアルを読むに当たっての参考情報について説明しています。

1.3 アプリケーション開発の概要

Stream Data Platform - AF でのアプリケーション開発の概要について説明します。

アプリケーション開発では、次の作業を実施します。

- 分析シナリオの定義（クエリの定義）
- ストリームデータ処理エンジンとのデータ送受信用のアプリケーションの作成（カスタムアダプターの作成）

分析シナリオは、分析するデータや分析内容に合わせて定義する必要があります。

カスタムアダプターは、標準提供アダプターを使用するシステムでは不要なアプリケーションです。必要に応じて作成してください。

1.3.1 クエリの定義でできること

Stream Data Platform - AF では、ストリームデータ処理エンジンに入力されたストリームデータに対して、あらかじめ登録されたシナリオに沿った分析を実行し、目的に応じた分析結果を出力します。

シナリオは、クエリ定義ファイルに、クエリとして定義します。クエリに定義する内容を次に示します。

- **分析対象の特定方法**

ストリームデータは、途切れることなく続く時系列順のデータです。分析するためには、分析対象とする範囲を特定する必要があります。

Stream Data Platform - AF では、ウィンドウという概念を使用して、ストリームデータを時間またはタプルの個数で区切られた有限のデータとして扱い、分析対象の範囲を特定します。

- **分析処理の内容**

ウィンドウで区切られたストリームデータに対して、目的に沿った分析処理を実施します。特定の列について値の推移を分析したり、複数のストリームデータの値を組み合わせる新たなストリームデータを生成して分析したりします。

分析処理の内容は、選択、結合などの関係演算や、集合関数を使用した集合演算などによって定義します。

- **分析結果の出力方法**

分析した結果を新たなストリームデータとして出力します。

分析結果のストリームデータは、分析結果が増減したときに出力したり、一定間隔ですべての分析結果を出力したりできます。

分析結果の出力には、ストリーム化演算という演算を使用します。

ポイント

ストリーム間演算を使用すると、リレーションを生成しないで、直接ストリームデータに対して演算を実行し、演算の結果を別のストリームデータに変換できます。

ストリーム間演算では、入出力がストリームデータであること以外は特に規定がなく、入力されたストリームデータに対して実行する処理は任意です。ストリーム間演算関数の処理ロジックを、ユーザーが Java で記述して作成するクラスファイルにメソッドとして実装することで、任意の処理を実行できます。

ストリーム間演算を使用する場合は、CQL でストリーム間演算関数を定義するほかに、外部定義関数の作成が必要です。外部定義関数の作成については、「10. 外部定義関数の作成」を参照してください。

クエリは、CQL という SQL に類似したクエリ言語を使用して定義します。

CQL を使用したクエリの定義方法については、「2. CQL によるクエリの定義」で説明します。また、CQL の記述方法の詳細は、「3. CQL の基本項目とデータ型」、「4. CQL リファレンス」、および「5. CQL で指定する組み込み関数」で説明します。なお、クエリ定義ファイルについては、マニュアル「uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド」を参照してください。

また、クエリ定義ファイルは、サンプルを基に定義できます。サンプルの内容は、「6. クエリ定義のサンプル」で説明します。

1.3.2 カスタムアダプターの作成でできること

Stream Data Platform - AF では、ストリームデータの入出力に使用する機能として、標準提供アダプターを提供しています。標準提供アダプターを使用する場合、API を使用したアプリケーションの開発は不要です。

ただし、標準提供アダプターには次のような制限があります。

標準提供アダプターの制限事項

- 扱えるストリームデータの形式は、ファイルまたは HTTP パケットだけです。
- 定義できるシナリオ（クエリグループ）の個数に制限があります。
- ストリームデータ処理エンジンに対するデータ送信とデータ受信に使用するアダプター（アプリケーション）のプロセス構成を任意に決定できません（データ送信 AP をインプロセスで開始する場合は、データ受信 AP もインプロセスで開始する必要があるなど）。

標準提供アダプターについては、マニュアル「uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド」を参照してください。

任意の形式のストリームデータを分析したい場合など、標準提供アダプターを使用できないときに、カスタムアダプターを作成してください。

カスタムアダプターは、Stream Data Platform - AF が提供するデータ送受信 API を使用した Java アプリケーションとして作成します。データ送受信 API の使用方法、アプリケーション開発時の留意事項、コンパイル方法などについては、「7. カスタムアダプターの作成」で説明します。データ送受信 API の詳細は、「8. データ送受信 API」で説明します。

また、Stream Data Platform - AF では、カスタムアダプター作成のためのサンプルプログラムを提供しています。サンプルプログラムの内容については、「9. データ送受信 API を使用したサンプルプログラム」で説明します。

2

CQLによるクエリの定義

この章では、CQLによるクエリの定義方法について説明します。

なお、この章で示すCQLの記述方法の詳細は、「4. CQLリファレンス」を参照してください。

2.1 CQLの体系

Stream Data Platform - AFでは、ストリームデータ処理エンジンに入力されたストリームデータに対して、あらかじめ登録されたシナリオに沿った分析を実行し、目的に応じた分析結果を出力します。

シナリオは、CQLというクエリ言語を使用して定義します。

Stream Data Platform - AFで使用するCQLの体系を次の表に示します。

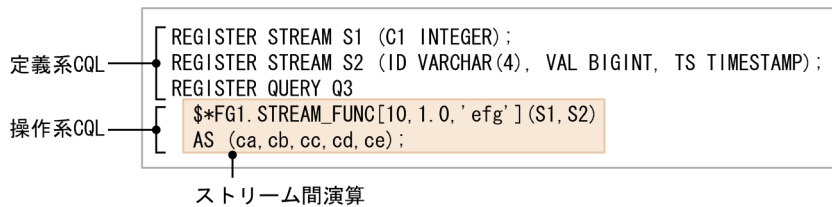
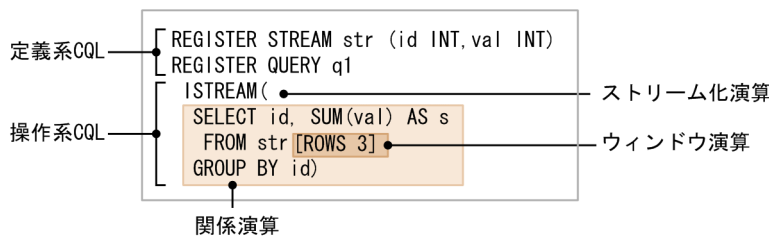
表 2-1 Stream Data Platform - AF で使用する CQL の体系

項番	CQLの分類	用途	該当するCQLの例
1	定義系CQL	ストリームやクエリを定義して、ストリームデータ処理エンジンに登録します。	REGISTER STREAM REGISTER QUERY など
2	操作系CQL	クエリで実行するストリームデータの処理内容について定義します。REGISTER QUERY 句に続けて記述します。 操作系CQLでは、次の4種類の演算実行をクエリに定義できます。 <ul style="list-style-type: none"> • ウィンドウ演算 • 関係演算 • ストリーム化演算 • ストリーム間演算 	SELECT FROM WHERE GROUP BY HAVING UNION など

CQLの詳細については、「4. CQLリファレンス」で説明します。

CQLによるクエリの指定例を次の図に示します。

図 2-1 CQLによるクエリの指定例



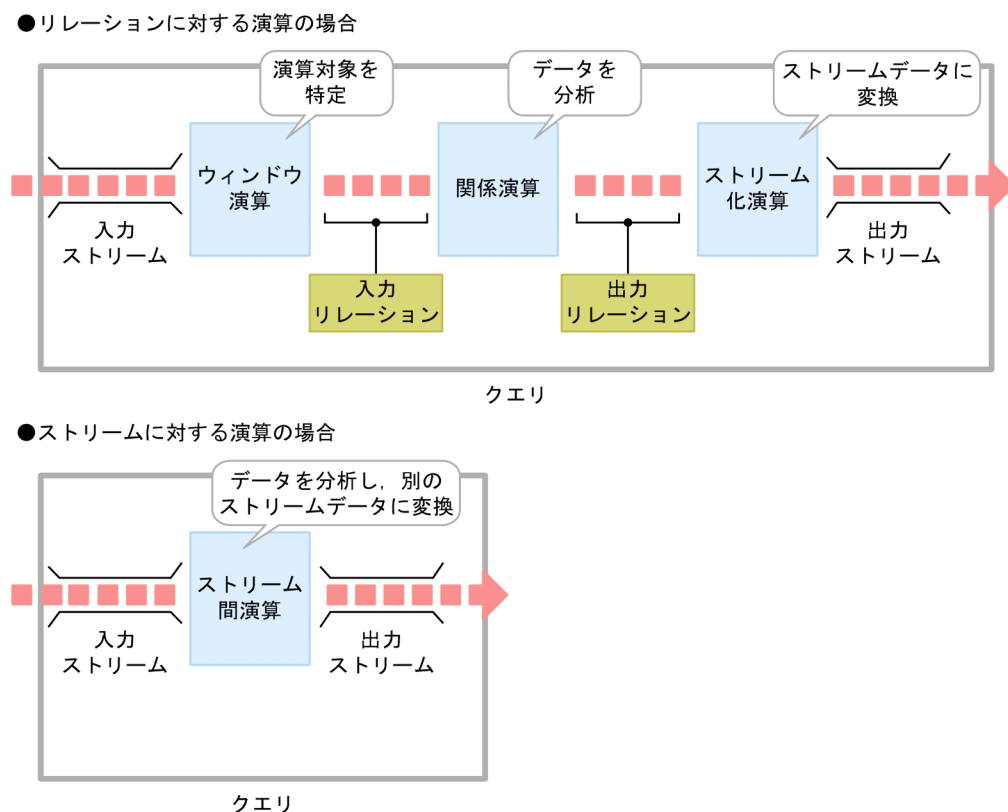
定義系CQLでは、ストリームデータ処理エンジンに、次の情報を登録します。

- 処理の対象になるストリームの情報
- 操作系CQLによって処理内容を記載したクエリ

操作系 CQL では、入力ストリームキューから入力されたデータに対してシナリオに沿った分析処理を実行するための、演算処理を定義します。

クエリで定義するウィンドウ演算、関係演算、ストリーム化演算、およびストリーム間演算の関係を次の図に示します。

図 2-2 演算の関係



各演算での実施内容について説明します。

ウィンドウ演算

ウィンドウ演算によって、演算対象を特定します。時系列集合である入力ストリームデータまたは出力ストリームデータは、そのままの状態では演算の対象にできません。ウィンドウ演算によって、生存期間を持つ、 n 項組のタプルの集合（リレーション）として切り取ることで、演算の対象にできます。演算の対象として特定したリレーションを入力リレーションといいます。

ウィンドウ演算では、タプルの個数や時間などによって演算対象を特定するために、ROWS ウィンドウ、RANGE ウィンドウ、PARTITION BY ウィンドウなどを使用します。

関係演算

関係演算によって、分析の目的に沿った処理を実行します。

関係演算では、選択、結合、集合関数といった演算を実行して、結果データをリレーションとして抽出します。このリレーションを出力リレーションといいます。

ストリーム化演算

ストリーム化演算によって、関係演算の結果データをストリームデータに変換します。出力リレーションの変化内容に応じて、追加分、削除分、集合などのデータをストリームデータとして出力できます。

ストリーム間演算

ストリーム間演算によって、ストリームデータに対して演算を実行し、別のストリームデータに変換します。

ストリーム間演算では、リレーションを生成しません。入力ストリームデータに対して演算を実行し、演算の結果を出力ストリームデータとして出力します。

また、入力されたストリームデータに対して実行する処理は任意です。ストリーム間演算関数の処理ロジックを、ユーザーがJavaで記述して作成するクラスファイルにメソッドとして実装することで、任意の処理を実行できます。

ストリーム間演算を使用する場合は、CQLでストリーム間演算関数を定義するほかに、外部定義関数の作成が必要です。外部定義関数の作成については、「10. 外部定義関数の作成」を参照してください。

2.2 ウィンドウ演算による入力リレーションの生成

この節では、ウィンドウ演算による入力リレーションの生成について説明します。

ウィンドウ演算は、ストリームデータの一部を切り取り、データ処理の対象を特定する演算です。タプルの数、時間など、異なる軸のウィンドウを組み合わせることでデータ処理の対象を特定することもできます。

2.2.1 ウィンドウ演算の種類

ウィンドウ演算では、4種類のウィンドウによって、処理対象を特定します。

それぞれのウィンドウによって生成される入力リレーションについて説明します。

(1) データの数による指定 (ROWS ウィンドウ)

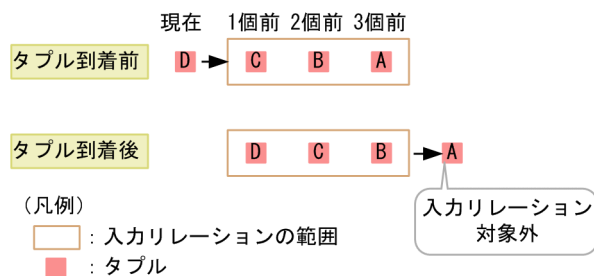
ROWS ウィンドウは、タプルの数によって入力リレーションの範囲を特定するウィンドウです。

例えば、ストリーム s1 の入力ストリームデータの直前 3 個分を入力リレーションとして保持する場合、CQL は次のように定義します。

```
SELECT ... FROM s1 [ROWS 3]...
```

この場合の入力リレーションの内容を次の図に示します。

図 2-3 ROWS ウィンドウの例



すでにリレーションに A, B, C の三つのタプルがある場合に新たなタプル D が到着した場合、到着タプルを含む三つのタプルが入力リレーションの内容になります。このとき、いちばん古いタプル A は、入力リレーションの対象から外れます。

(2) 時間による指定 (RANGE ウィンドウ)

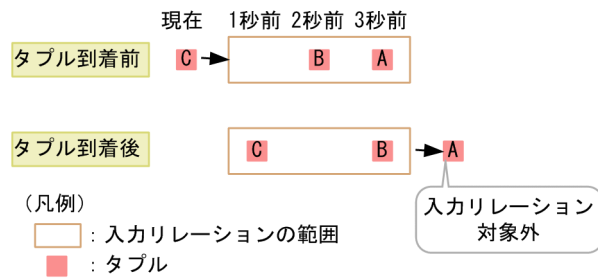
RANGE ウィンドウは、時間によって入力リレーションの範囲を特定するウィンドウです。

例えば、ストリーム s2 の入力ストリームデータの直前 3 秒分を入力リレーションとして保持する場合、CQL は次のように定義します。

```
SELECT ... FROM s2 [RANGE 3 SECOND]...
```

この場合の入力リレーションの内容を次の図に示します。

図 2-4 RANGE ウィンドウの例



タプル C の到着時、現在から直前 3 秒間分のタプルだけが入力リレーションに残ります。タプル C 到着後、タプル B が 3 秒前のタプルとなり、それよりも前に到着していたタプル A は入力リレーションの対象から外れます。

(3) 到着したタプルのタイムスタンプ時刻による指定 (NOW ウィンドウ)

NOW ウィンドウは、その時点で到着したタプルのタイムスタンプ時刻だけを演算の対象にするためのウィンドウです。

ほかのウィンドウで特定する入力リレーションが個数や時間の範囲を持つ「線」のリレーションであるのに対して、NOW ウィンドウは到着したタプルと同時刻という「点」のリレーションになります。

例えば、ストリーム s3 の入力ストリームデータについて、到着したタプルと同時刻のタプルを演算の対象とする場合、CQL は次のように定義します。

```
SELECT ... FROM s3 [NOW]
```

(4) データのグループによる指定 (PARTITION BY ウィンドウ)

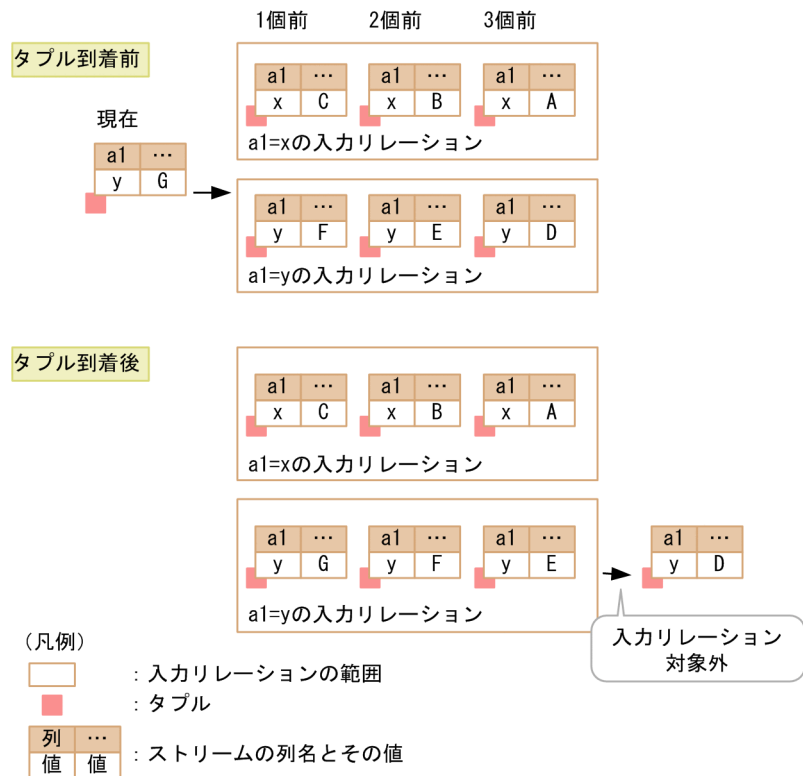
PARTITION BY ウィンドウは、タプルの種類ごとに、タプルの個数によって入力リレーションの範囲を特定するウィンドウです。

例えば、ストリーム s4 の入力ストリームデータについて、列指定リスト a1 の各値について直前 3 個分を入力リレーションとして保持する場合、CQL は次のように定義します。

```
SELECT ... FROM s4 [PARTITION BY a1 ROWS 3]
```

この場合の入力リレーションの内容を次の図に示します。

図 2-5 PARTITION BY ウィンドウの例



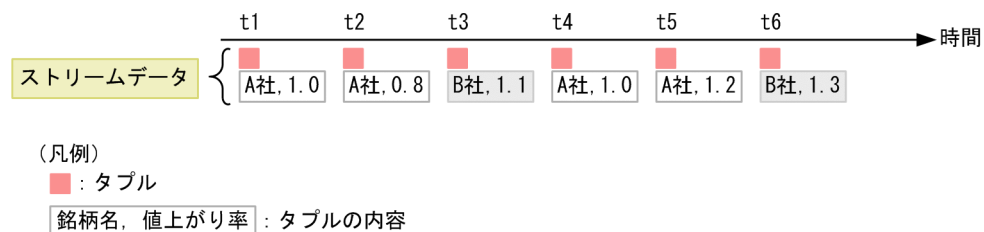
a1 列の値として、x と y があり、それぞれ直前 3 個分を入力レレーションとして保持しています。a1 が y のタプルが到着すると、a1 の値が y のタプル (タプル D) が一つ、入力レレーションの対象から外れます。

2.2.2 ウィンドウ演算の例

それぞれのウィンドウ演算の例を示します。

ここでは、次の図に示す構成のストリームデータが到着した場合を例に、それぞれのウィンドウの指定によって作成される入力レレーションについて説明します。

図 2-6 ウィンドウ演算の例で使用するタプルの構成



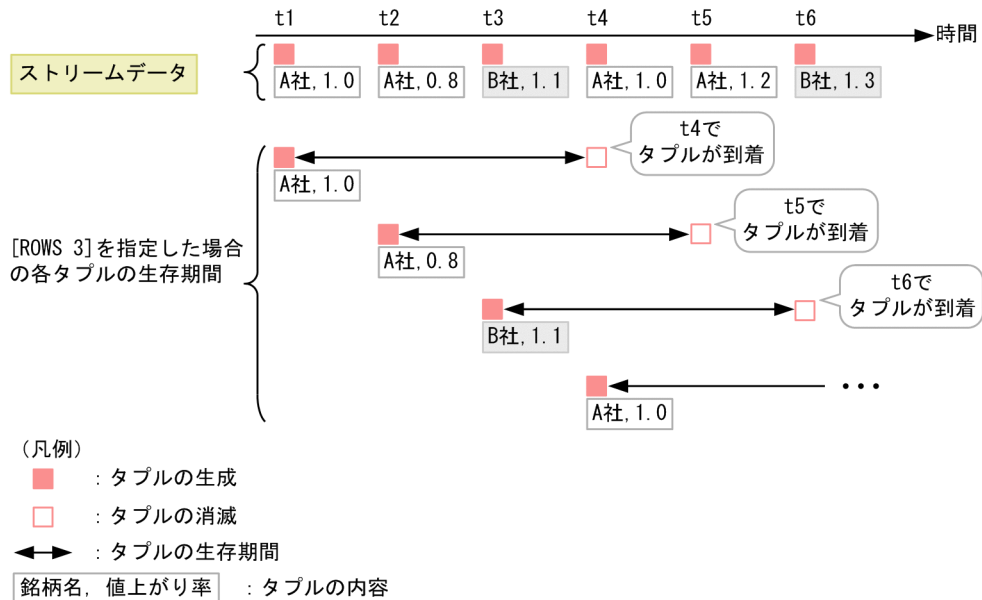
図の横軸は時間軸です。右に行くほど時間が経過しています。t1~t6 は、タプルが到着した時刻を示します。また、各タプルは、「銘柄名, 値上がり率」で構成されています。

(1) データの数による指定の例 ([ROWS 3]の例)

ここでは、[ROWS 3]を指定した場合のウィンドウ演算の例について説明します。この指定は、入力リレーション内に同時に生存するタプル数が3であることを示します。

[ROWS 3]を指定した場合の入力リレーション内の各タプルの生存期間について、次の図に示します。

図 2-7 [ROWS 3]を指定した場合の入力リレーション内の各タプルの生存期間



時刻が t1, t2, t3 と経過していく中で、「(A 社,1.0)」「(A 社,0.8)」「(B 社,1.1)」のタプルが順次到着し、入力リレーションに生成されていきます。

時刻 t4 に「(A 社,1.0)」のタプルが到着して、同時生存タプル数が3を超えた時点で、入力リレーション内でいちばん古いタプルの生存期間が終了します。この例では、時刻 t1 に生成された「(A 社,1.0)」のタプルが入力リレーションから削除されて、時刻 t4 に到着した「(A 社,1.0)」のタプルが新たに入力リレーションに生成されます。

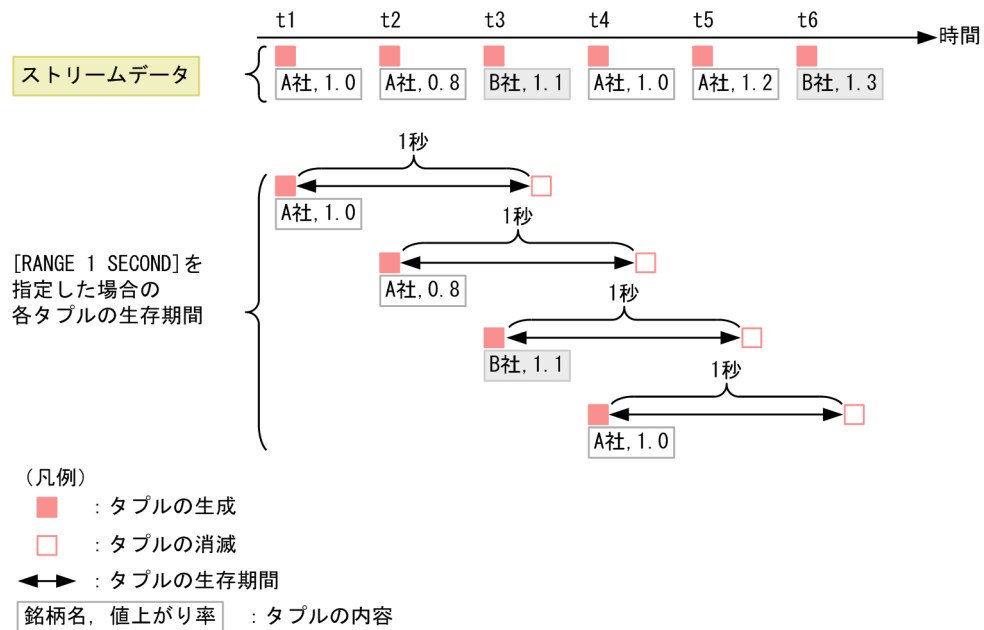
(2) 時間による指定の例 ([RANGE 1 SECOND]の例)

ここでは、[RANGE 1 SECOND]を指定した場合のウィンドウ演算の例について説明します。この指定は、入力リレーション内の各タプルの生存期間を1秒とすることを示します。

リレーション内に生成された各タプルは、その後到着するタプルの個数には関係なく、1秒たつと消滅します。

[RANGE 1 SECOND]を指定した場合の入力リレーション内の各タプルの生存期間について、次の図に示します。

図 2-8 [RANGE 1 SECOND]を指定した場合の入力リレーション内の各タプルの生存期間



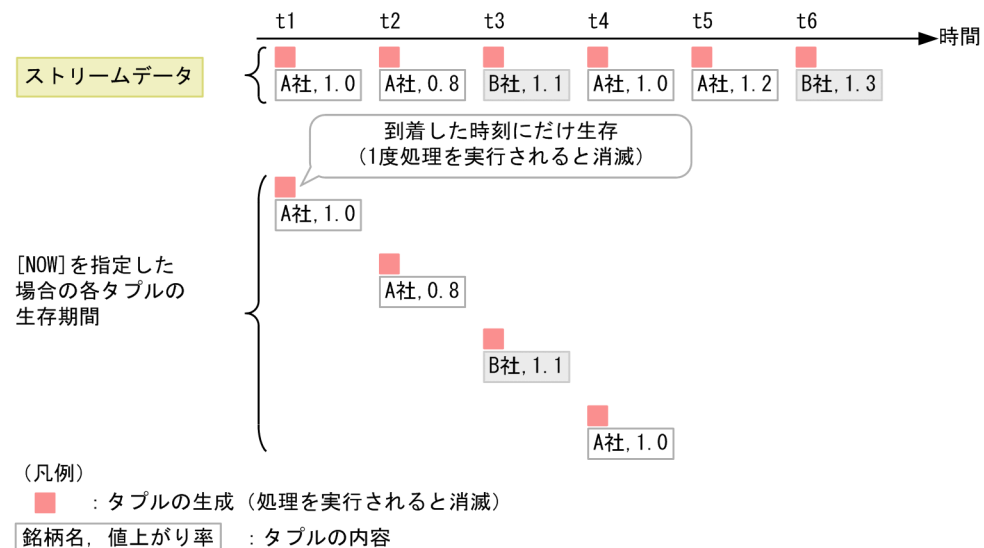
t1～t4の時刻に到着したタプルは、ほかのタプルの到着状況に関係なく、1秒たつと消滅します。

(3) 到着したタプルのタイムスタンプ時刻の指定による例 (NOW の例)

ここでは、[NOW]を指定した場合のウィンドウ演算の例について説明します。この指定は、入力リレーション内に現時刻のタプルだけを保持することを示します。

[NOW]を指定した場合の入力リレーション内の各タプルの生存期間について、次の図に示します。

図 2-9 [NOW]を指定した場合の入力リレーション内の各タプルの生存期間



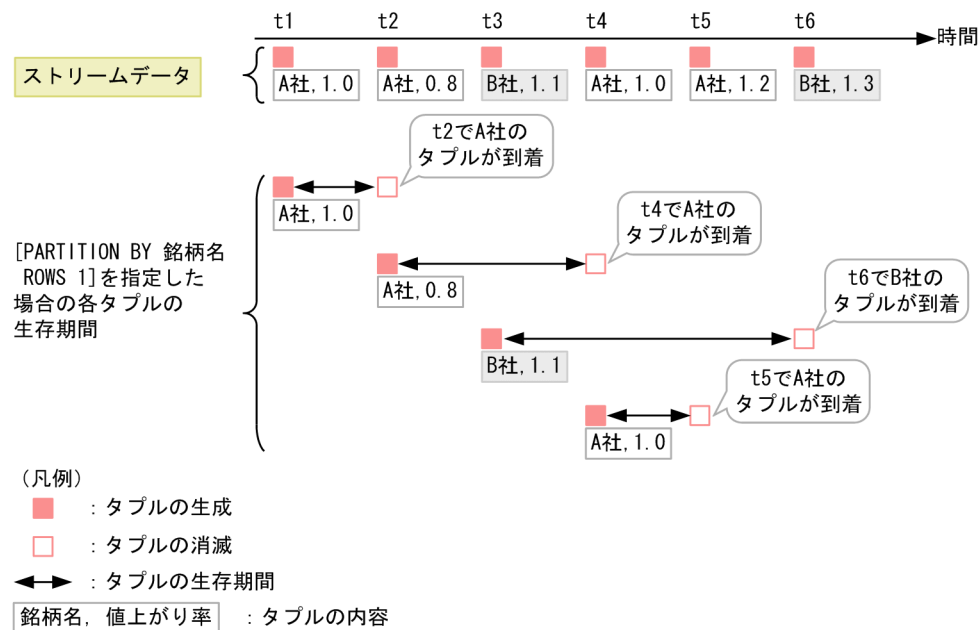
[NOW]を指定した場合、入力リレーション内のタプルは到着した時刻にだけ生存しています。一度演算処理が実行されたタプルは入力リレーションから消滅します。

(4) データのグループによる指定の例 ([PARTITION BY 銘柄名 ROWS 1]の例)

ここでは、[PARTITION BY 銘柄名 ROWS 1]を指定した場合のウィンドウ演算の例について説明します。この指定は、入力レーション内に銘柄ごとに、ROWSに指定した個数ずつのタプルを保持することを示します。

[PARTITION BY 銘柄名 ROWS 1]を指定した場合の入力レーション内の各タプルの生存期間について、次の図に示します。

図 2-10 [PARTITION BY 銘柄名 ROWS 1]を指定した場合の入力レーション内の各タプルの生存期間



この指定は、銘柄名 A 社および B 社ごとに、タプルを 1 個ずつ保存する指定です。

時刻 t1 に到着した「(A 社,1.0)」のタプルが入力レーションに生存している状態で時刻 t2 に「(A 社, 0.8)」のタプルが到着すると、t1 に到着した「(A 社,1.0)」のタプルが消滅して「(A 社,0.8)」のタプルが入力レーション内に生成されます。続いて時刻 t3 に「(B 社,1.1)」のタプルが到着した場合は、銘柄名が異なるため、「(A 社,0.8)」のタプルは消滅しないまま、新たに「(B 社,1.1)」のタプルが生成されます。時刻 t2 に到着した「(A 社,0.8)」のタプルは時刻 t4 に「(A 社,1.0)」のタプルが到着するまで、時刻 t3 に到着した「(B 社,1.1)」のタプルは時刻 t6 に「(B 社,1.3)」のタプルが到着するまで、それぞれ入力レーション内で保持されます。

2.3 関係演算によるデータの抽出

この節では、関係演算によるデータの抽出について説明します。

関係演算では、選択、結合、および集合関数によって入力リレーション内のストリームデータを操作して、結果のデータを出力リレーションとして抽出します。

2.3.1 関係演算の種類

関係演算では、次の3種類の操作によって結果を抽出します。

- 選択

n 個のタプルを含む入力リレーションから、特定の条件を満たすタプルを抽出します。

例を示します。

```
REGISTER QUERY q1 SELECT s1.a FROM s1 [ROWS 10] WHERE s1.a > 10;
```

この例では、s1 の入力リレーションから、WHERE 句以下で指定した条件に該当するタプルを抽出します。

- 結合

n 個のタプルを含む複数の入力リレーションから、同じデータの組み合わせを結合した結果を抽出します。

例を示します。

```
REGISTER QUERY q2 SELECT s1.a, s1.b, s2.b FROM s1 [ROWS 10], s2 [ROWS 10] WHERE s1.a = s2.a;
```

この例では、s1 および s2 の入力リレーションから、WHERE 句以下で指定した条件に該当するタプルを抽出します。

- 集合関数

n 個のタプルを含む入力リレーションに対して集合関数による演算を実行した結果を抽出します。

例を示します。

```
REGISTER QUERY q3 SELECT SUM(s1.a) AS c1 FROM s1 [ROWS 10];
```

この例では、s1 の入力リレーションに対して、集合関数 SUM による演算を実行した結果を抽出します。

関係演算での処理内容の詳細については、「4. CQL リファレンス」を参照してください。

以降では、関係演算で実行する処理のうち、結合処理の例、および集合関数による演算処理の例を示します。

2.3.2 結合処理の例

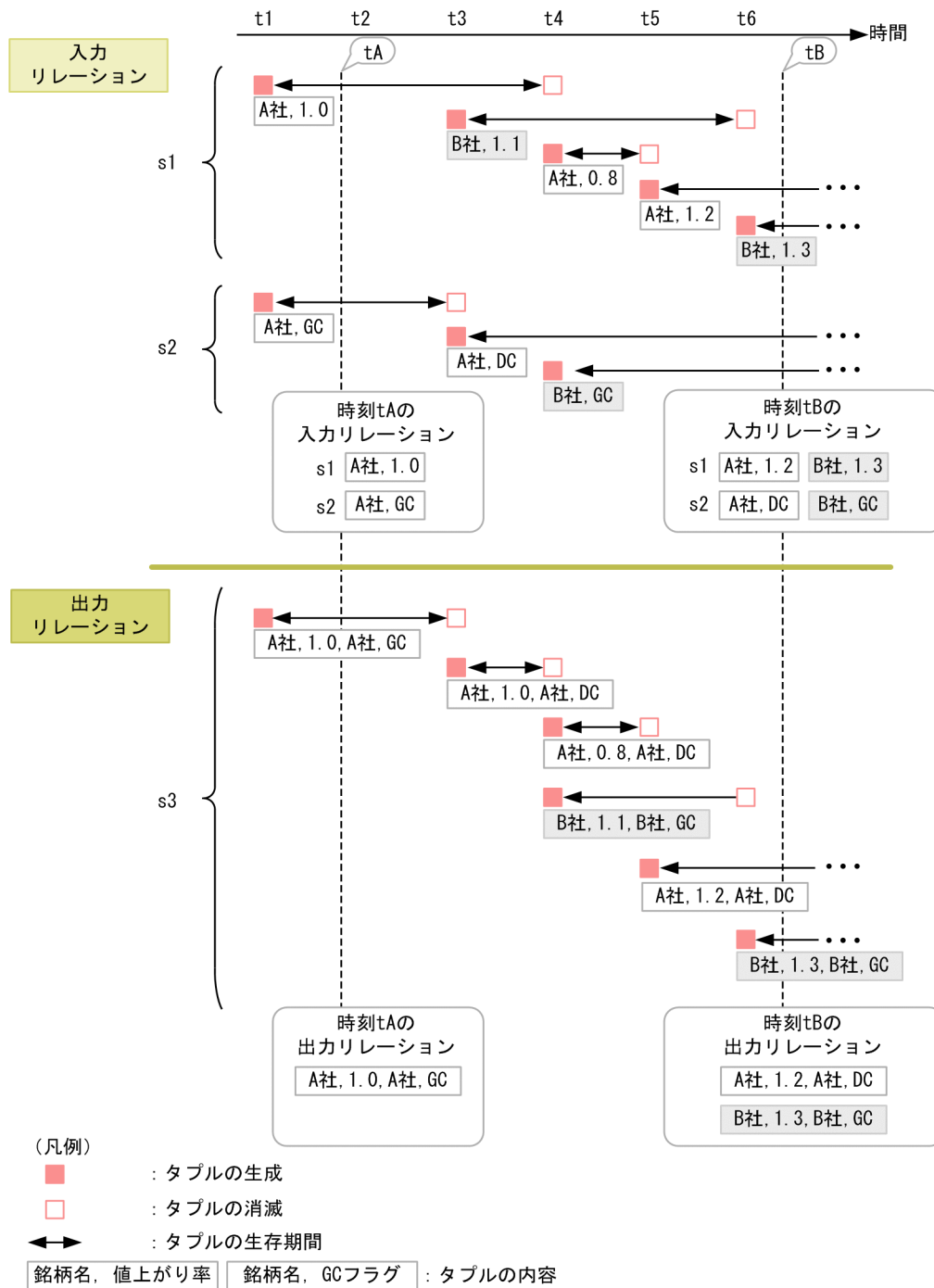
ここでは、s1, s2 の2種類の入力リレーションを銘柄ごとに結合した結果を出力リレーションとして抽出する例を示します。

CQL の定義例を次に示します。なお、実際の CQL の定義では、列名は半角英数字で指定してください。

```
REGISTER QUERY q
  SELECT s1.銘柄名, s1.値上がり率, s2.銘柄名, s2.GCフラグ
  FROM s1[PARTITION BY s1.銘柄名 ROWS 1],
       s2[PARTITION BY s2.銘柄名 ROWS 1]
  WHERE s1.銘柄名=s2.銘柄名;
```

この CQL を定義した場合の入力リレーションと出力リレーションの例を次の図に示します。

図 2-11 結合処理の例



図の横軸は時間軸です。右に行くほど時間が経過しています。t1～t6は、タプルが到着した時刻を示します。また、入力リレーションのs1のタプルは、「銘柄名, 値上がり率」で構成されており、s2のタプルは、「銘柄名, GCフラグ」で構成されています。なお、GCフラグの値は、「DC (Dead Cross)」, 「GC (Golden Cross)」のどちらかです。

この例では、銘柄名によって結合処理を実行して出力します。

時刻 tA では、入力リレーションの s1 にはタプル「(A 社,1.0)」, s2 にはタプル「(A 社,GC)」があります。これらを銘柄名で結合した結果として、出力リレーションには、タプル「(A 社,1.0,A 社,GC)」が出力されます。

同様に、時刻 tB では、入力リレーションの s1 のタプルとして「(A 社,1.2)」 「(B 社,1.3)」があり、s2 のタプルとして「(A 社,DC)」 「(B 社,GC)」があります。これらを銘柄名で結合した結果として、出力リレーションには、タプル「(A 社,1.2,A 社,DC)」 「(B 社,1.3,B 社,GC)」が出力されます。

このように、特定時刻での結合処理の結果は、入力リレーションのタプルの生存期間に応じたタプルとして、出力リレーションに抽出されます。

2.3.3 集合関数による演算処理の例

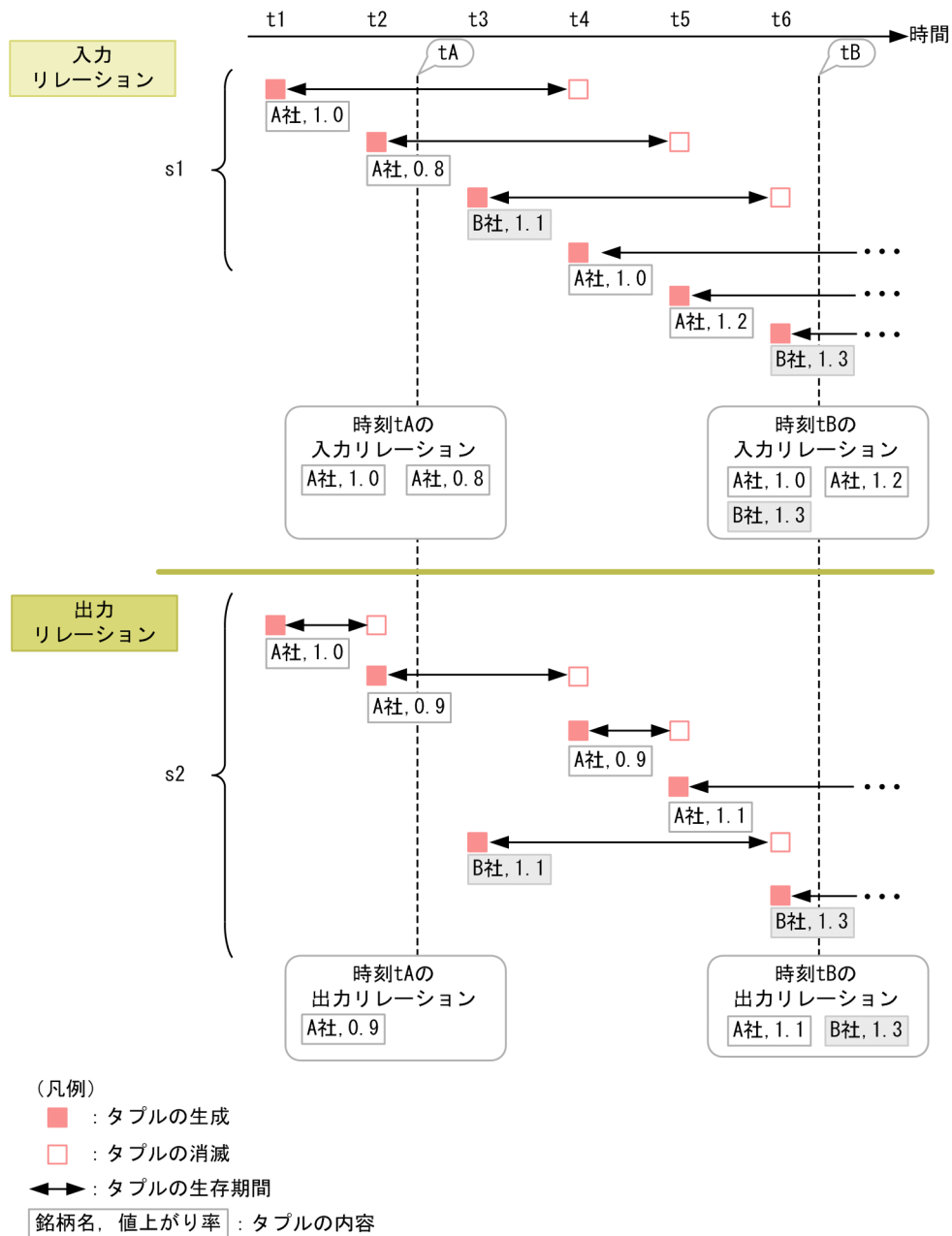
ここでは、「銘柄名」と「値上がり率」という内容を持つタプルに対して、GROUP BY 句と集合関数 AVG を使用して、銘柄名別に値上がり率の平均値を算出する関係演算の例を示します。

CQL の定義例を次に示します。なお、実際の CQL の定義では、列名は半角英数字で指定してください。また、AVG には相関名を指定してください。

```
REGISTER QUERY q
  SELECT AVG(値上がり率)
  FROM s1[ROWS 3] GROUP BY 銘柄名;
```

この CQL を定義した場合の入力リレーションと出力リレーションの例を次の図に示します。

図 2-12 集合関数による演算処理の例



図の横軸は時間軸です。右に行くほど時間が経過しています。t1～t6は、タブルが到着した時刻を示します。

この例は、GROUP BY句で指定した銘柄別に、値上がり率を対象に集合演算 AVG (平均値の算出) を指定しています。なお、入力リレーションは、[ROWS 3]の指定による、期間付きのリレーションです。

入力リレーションの時刻 tA での集合は「(A 社,1.0)(A 社,0.8)」となり、銘柄名別の平均値は「(A 社,0.9)」となります。

同様に、時刻 tB での集合は「(A 社,1.0)(A 社,1.2)(B 社,1.3)」となり、銘柄名別の平均値は、「(A 社,1.1)(B 社,1.3)」となります。

このように、特定時刻での銘柄名別の値上がり率の平均値は、入力リレーションのタプルの生存期間に応じたタプルとして、出力リレーションに抽出されます。

2.3.4 結合処理と ROWS ウィンドウを併用する場合の注意事項

ここでは、結合処理と ROWS ウィンドウを併用する場合の注意事項について説明します。

結合処理の結果として出力されるタプルの数が ROWS ウィンドウで指定したタプル数よりも多い場合、処理結果は ROWS ウィンドウで指定した数分だけ出力され、すべての処理結果が出力されません。

この問題は、次の二つの条件を満たすクエリを指定した場合に発生します。

1. あるクエリで、結合演算によって同一時刻を持つ複数のデータを生成する。
2. 別のクエリで、1.のクエリの結果を入力して処理を実行するが、このクエリの ROWS ウィンドウで指定した生存タプル数が 1.で生成されるデータ数よりも少ない。

この条件に該当するクエリの例を示します。

```
REGISTER STREAM s1 (c1 INT);
REGISTER STREAM s2 (c2 INT);
REGISTER QUERY q1 ISTREAM(                条件1.のクエリ
  SELECT * FROM s1[ROWS 3], s2[ROWS 3]);
REGISTER QUERY q2 ISTREAM(                条件2.のクエリ
  SELECT * FROM q1[ROWS 1]);
```

このクエリに対して、時刻 t に、ストリーム $s1$ または $s2$ へのタプルが到着した場合を想定します。クエリ $q1$ では、二つのストリームデータを入力して、結合演算を実施します。入力ストリームデータとして $[ROWS 3]$ を指定しているため、時刻 t での結合演算の結果としては、同一時刻を持つ三つのタプルが生成されます。

クエリ $q2$ で ROWS ウィンドウに指定した同時生存タプル数 (1) は、クエリ $q1$ で生成されるデータ数 (3) よりも少ないため、時刻 t での生存タプルは、ウィンドウに到着したタプルのうち最後の一つのタプルだけになります。クエリ $q1$ の処理結果のうち、残り二つに対する $q2$ の処理結果は出力されません。

この例の場合、クエリ $q2$ の ROWS ウィンドウで 3 以上の数を指定するなど、CQL を見直す必要があります。

2.4 ストリーム化演算による出力ストリームデータへの変換

この節では、ストリーム化演算による出力ストリームデータへの変換について説明します。

ストリーム化演算は、出力リレーション内のデータの変化に応じて、分析結果をストリームデータとして出力する方法を指定する演算です。ストリーム化演算によって、次の3種類のタプルをストリームデータとして出力できます。

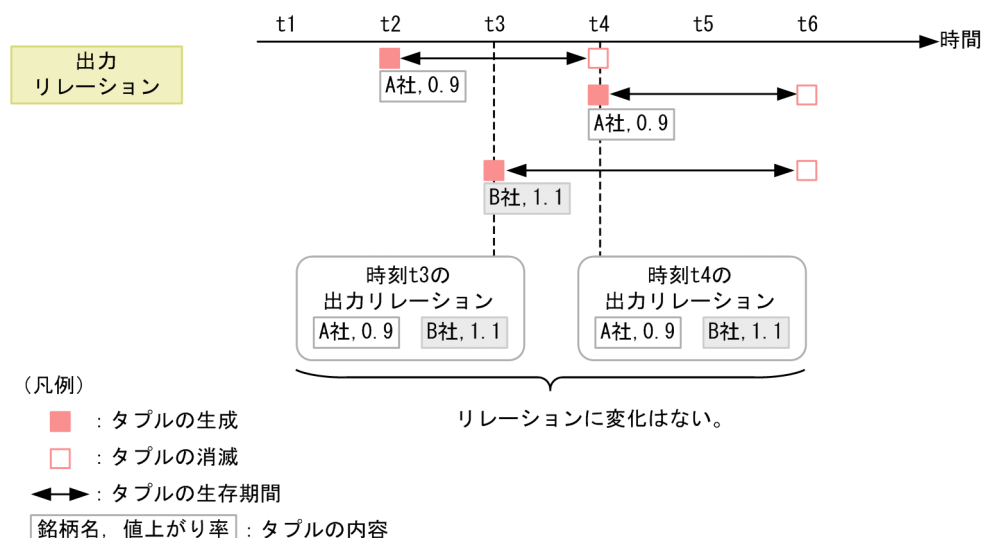
- 出力リレーションに追加されたタプル
- 出力リレーションから削除されたタプル
- 出力リレーション内のタプルの集合

ポイント

出力リレーション内のデータの変化とは、リレーションとしての変化を意味します。出力リレーション内のタプルが変化しても、結果の内容が同じであれば、変化したとは見なされません。

例を示します。

図 2-13 リレーションが変化しない例



図の場合、時刻 t3 では、時刻 t2 の出力リレーション「(A 社,0.9)」に対してタプル「(B 社,1.1)」が追加されたことに伴い、出力リレーションが「(A 社,0.9)(B 社,1.1)」に変化します。一方、時刻 t4 では、t2 に生成された「(A 社,0.9)」が消滅しますが、同時に別のタプルとして「(A 社,0.9)」が生成されるため、出力リレーションの内容は「(A 社,0.9)(B 社,1.1)」のままになります。この場合、時刻 t4 で出力リレーションは変化していないものと見なされます。

2.4.1 ストリーム化演算の種類

ストリーム化演算の種類と指定方法について説明します。

(1) 出力リレーションに追加されたタプルを出力する指定 (ISTREAM)

ISTREAM は、出力リレーションにタプルが追加されたときに、そのタプルを出力する演算です。指定例を次に示します。

```
ISTREAM( SELECT ... FROM s1 [ROWS 10] ... )
```

(2) 出力リレーションから削除されたタプルを出力する指定 (DSTREAM)

DSTREAM は、出力リレーションからタプルが削除されたときに、そのタプルを出力する演算です。指定例を次に示します。

```
DSTREAM( SELECT ... FROM s2 [ROWS 10] ... )
```

(3) 出力リレーション内のタプルの集合を一定間隔で出力する指定 (RSTREAM)

RSTREAM は、出力リレーション内のタプルの集合を一定間隔で出力する演算です。指定例を次に示します。

```
RSTREAM[1 SECOND]( SELECT ... FROM s3 [ROWS 10] ... )
```

! 注意事項

タイムスタンプモードをデータソースモードにしている場合、RSTREAM を指定しても意図した内容が出力されないおそれがあります。

データソースモードでは、タプルが到着したタイミングでリレーションの時刻を進めます。例えば、RSTREAM で出力間隔を 1 分とした場合でも、「09:01:00」にタプルが到着したあと、次のタプルが「10:00:00」に到着した場合、09:02 から 09:59 の間、ストリームデータは出力されません。「10:00:00」のタプルが到着したタイミングで、1 時間分のタプルがまとめて出力されてしまいます。

データソースモードで RSTREAM を使用する場合は、この動作を理解した上で、注意して使用してください。

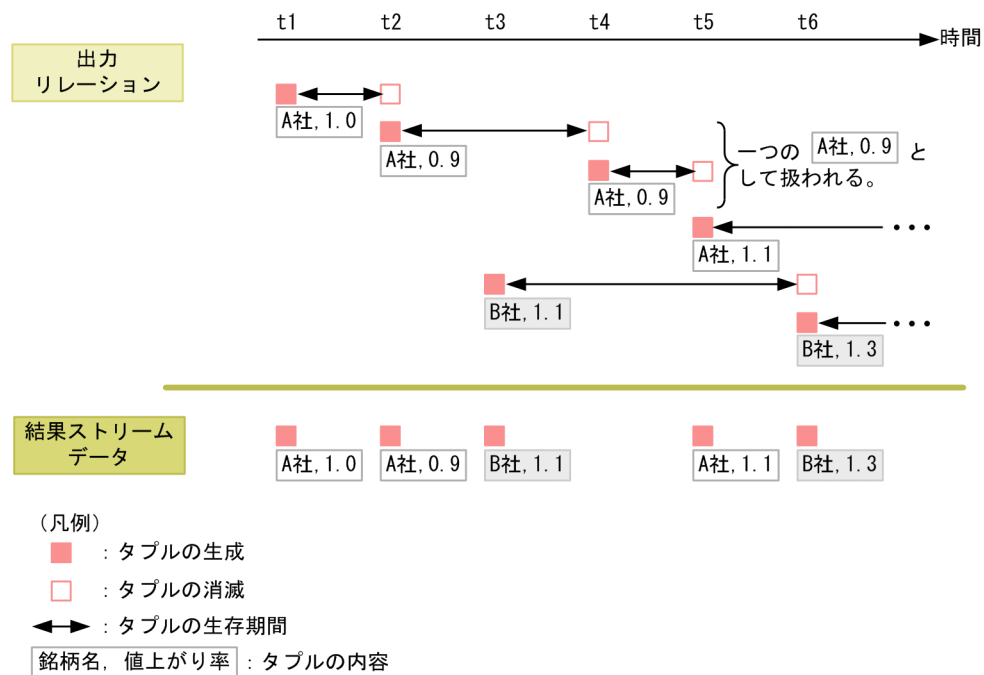
2.4.2 ストリーム化演算の例

ここでは、それぞれのストリーム化演算を指定した場合に、結果として出力されるストリームデータの例について説明します。なお、ここで示すのは、ウィンドウ演算に「PARTITION BY 銘柄名 ROWS 1」を指定した場合の例です。

(1) 出力リレーションに追加されたタプルを出力する指定の例 (ISTREAM)

ISTREAM を指定した場合の出力リレーションと結果ストリームデータの例を次の図に示します。

図 2-14 ISTREAM を指定した場合の出力リレーションと結果ストリームデータの例



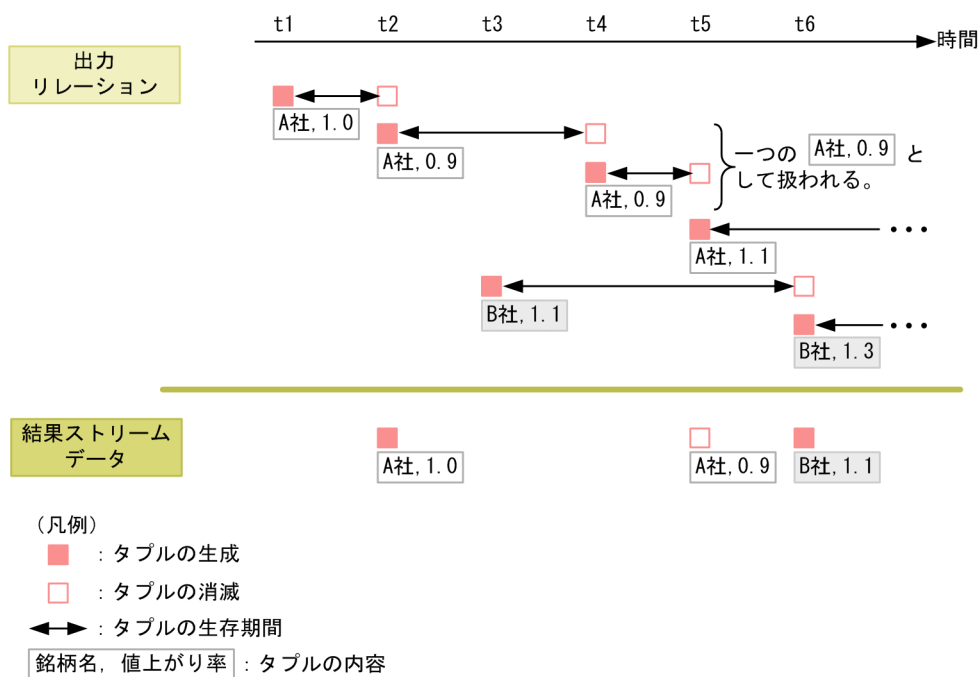
出力リレーションにタプルが追加されたときに、結果ストリームデータとして対応するタプルが出力されます。この例の場合は、出力リレーションの時刻 t1 にタプル「(A 社,1.0)」が追加された時点で、結果ストリームデータにも時刻 t1 にタプル「(A 社,1.0)」が出力されます。以降、時刻 t2、時刻 t3 にタプルが追加されるたびに、結果ストリームデータに対応するタプルが出力されます。

なお、時刻 t4 では、タプル「(A 社,0.9)」が追加されるのと同時に時刻 t2 で追加されたタプル「(A 社, 0.9)」が削除されるため、リレーションとしては変化しません。このため、結果ストリームデータには何も出力されません。

(2) 出力リレーションから削除されたタプルを出力する指定の例 (DSTREAM)

DSTREAM を指定した場合の出力リレーションと結果ストリームデータの例を次の図に示します。

図 2-15 DSTREAM を指定した場合の出力リレーションと結果ストリームデータの例



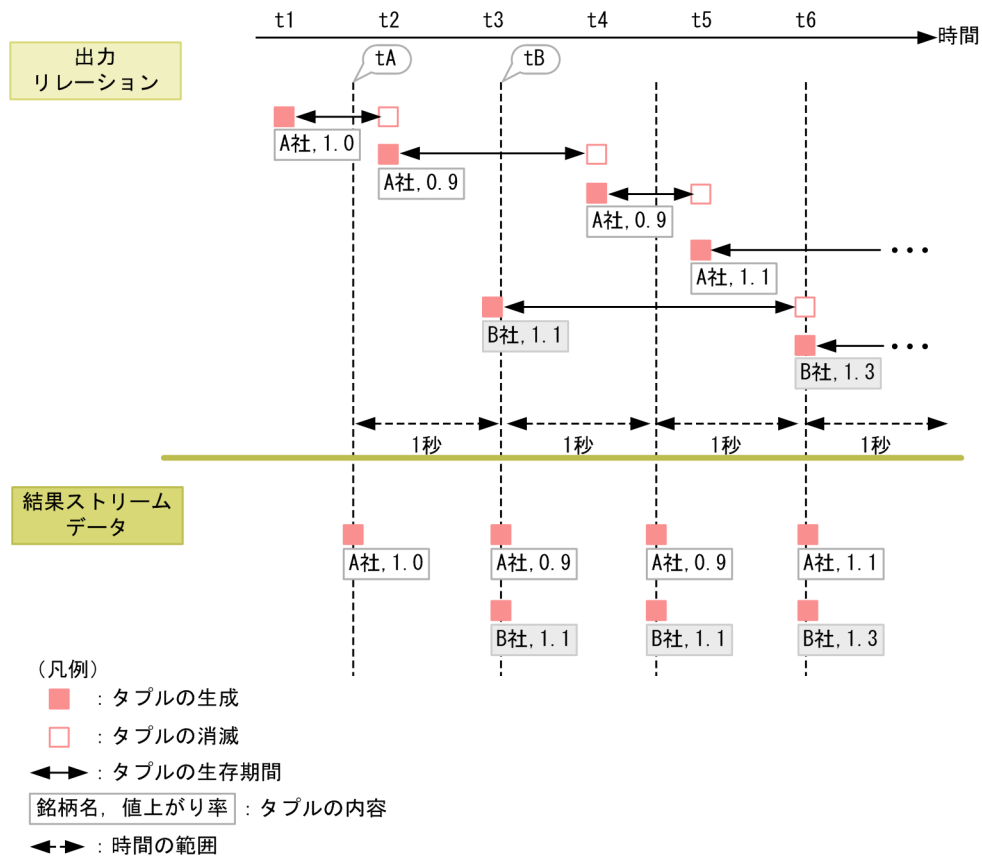
出力リレーションからタプルが削除されたときに、結果ストリームデータとしてタプルが出力されます。この例の場合は、出力リレーションのタプル「(A社,1.0)」が時刻 t2 で削除された時点で、結果ストリームデータにタプル「(A社,1.0)」が出力されます。

なお、時刻 t4 では、タプル「(A社,0.9)」が削除されるのと同時にタプル「(A社,0.9)」が追加されるため、リレーションとしては変化しません。このため、ISTREAM を指定した場合と同様に、結果ストリームデータには何も出力されません。

(3) 出力リレーション内のタプルの集合を一定間隔で出力する指定の例 (RSTREAM)

RSTREAM を指定した場合の出力リレーションと結果ストリームデータの例を次の図に示します。

図 2-16 RSTREAM を指定した場合の出力リレーションと結果ストリームデータの例



出力リレーション内の集合が、結果ストリームデータのタプルとして 1 秒間隔で出力されます。時刻 tA には、出力リレーション内には「(A 社,1.0)」だけしかいないため、結果ストリームデータには「(A 社,1.0)」だけが出力されます。1 秒経過後の時刻 tB では、出力ストリームの集合が「(A 社,0.9)(B 社,1.1)」になっているため、結果ストリームデータには「(A 社,0.9)(B 社,1.1)」が出力されます。

以降、1 秒ごとの出力リレーションの内容が、結果ストリームデータに出力されます。

2.5 時刻解像度の指定によるメモリ使用量増加の抑止

この節では、RANGE ウィンドウを指定して大量の受信データを処理する場合などに、時刻解像度を指定することによってメモリ使用量の増加を抑止する方法について説明します。

RANGE ウィンドウを指定したクエリの場合、一定時間内に受信したデータがすべて処理の対象になります。このため、指定する時間やデータの量によっては、メモリ使用量が増加して、システムの動作に影響が出るおそれがあります。

これに対して、時刻解像度を指定することで、RANGE ウィンドウの処理対象となるデータの量を一定に抑えられます。

時刻解像度の指定とは、RANGE ウィンドウで指定した範囲のリレーションを任意の単位時間に分割して、分割した時間ごとに演算処理を実行する機能のことです。なお、分割したリレーションのことをメッシュといいます。分割する間隔（ミリ秒単位）をメッシュ間隔といいます。

RANGE ウィンドウで指定した時間の範囲で扱う受信データが多い場合、あらかじめメッシュ単位で演算を実行し、その結果を擬似タプルとして保持します。RANGE ウィンドウでは、この擬似タプルを処理対象とすることで、RANGE ウィンドウ内のタプル数を一定に保ちます。これによって、メモリ使用量の増加を抑止できます。

例えば、RANGE ウィンドウ内のタプル数が 10 万個ある場合に、時刻解像度を指定して、10 万個のタプルをあらかじめ 8 個の擬似タプルにしておけば、メモリ内に保持しておくタプルの個数は 8 個に抑えられます。

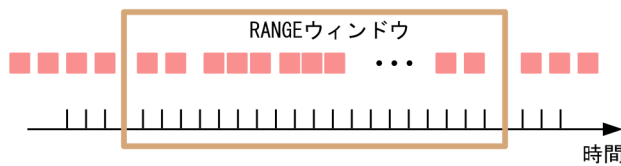
時刻解像度は、次の二つの条件を満たすクエリで指定できます。

- RANGE ウィンドウを使用している。
- 関係演算として、次のどれかの集合関数を使用している。
 - SUM（合計値の算出）
 - MAX（最大値の算出）
 - MIN（最小値の算出）

時刻解像度を指定しない場合と指定した場合の演算の違いを次に示します。

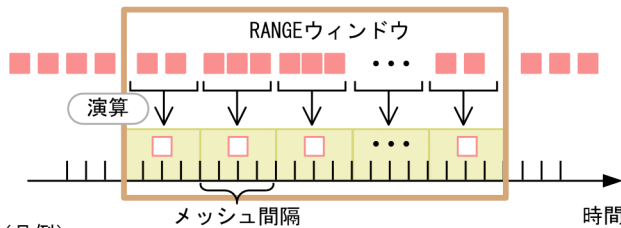
図 2-17 時刻解像度を指定しない場合と指定した場合の演算の違い

●時刻解像度を指定しない場合のRANGEウィンドウ内での演算



RANGEウィンドウ内のすべての到着
タプルが演算対象になる。

●時刻解像度を指定した場合のRANGEウィンドウ内での演算



まずメッシュ単位で演算を実行し、
実行結果である擬似タプルをRANGEウ
ィンドウで処理する。

(凡例)

- : 到着タプル
- : 擬似タプル
- : メッシュ

時刻解像度を指定しない場合、演算の対象は、RANGE ウィンドウ内のすべてのタプルになります。例えば、RANGE ウィンドウの範囲として1分間を指定している場合に、1分間に到着するタプルの総数が10万個ある場合、演算の対象は10万個になります。最大値を求める演算の場合、10万個のタプルをすべて比較して、最大値を求める必要があります。

一方、時刻解像度を指定した場合、演算の対象は、擬似タプルになります。擬似タプルは、あらかじめ指定したメッシュ間隔に応じて演算処理が実行された結果です。例えば、メッシュ間隔の指定によってRANGE ウィンドウを8個に分割していた場合、擬似タプルは8個作成されます。最大値を求める演算の場合、8個の擬似タプルの内容を比較すれば、最大値を求められます。このように、時刻解像度を指定することで、RANGE ウィンドウ内に保持するタプル数を削減し、メモリ使用量の増加を抑えられます。

ポイント

時刻解像度は、ストリームデータの送信レートが高く、RANGE ウィンドウ内に生存する到着タプルが多い場合に指定すると効果的です。

時刻解像度を指定することで、Java ヒープ領域内のメモリ使用量を一定に保つことができるので、メモリ使用量の単調増加を抑止できます。

時刻解像度は、単位時間当たりの最大値 (MAX)、最小値 (MIN)、合計値 (SUM) を求める処理ロジックを持つクエリに適用できます。

時刻解像度の指定例については、「2.6.2 時刻解像度を指定した定義例」を参照してください。

2.6 定義例

この節では、CQL によるクエリの定義例について説明します。

2.6.1 基本的なクエリの定義例

ここでは、ウィンドウ演算、関係演算およびストリーム化演算を使用した、基本的なクエリの定義例として、株価情報を対象としたクエリの定義例を示します。

この例で扱う株価情報ストリームデータの例を次の図に示します。

図 2-18 株価情報ストリームデータの例

株価情報ストリームデータ (stock)

時刻	銘柄コード (stockID)	銘柄名 (stockName)	株価 (currentPrice)	出来高 (tradingVolume)
10:00:00	6501	A社	780	12442000
10:00:05	1468	B社	1650	3318000
10:00:10	2282	C社	1518	1347000
...

時刻列は「タイムスタンプ」として、他の列は「株価情報」としてラベルされています。

この例で扱うストリームデータは、銘柄コード (stockID)、銘柄名 (stockName)、株価 (currentPrice) および出来高 (tradingVolume) という株価情報を持つデータに対して、Stream Data Platform - AF でタイムスタンプを設定したストリームデータです。

このストリームデータを使用して、株価の値上がり率を計算します。値上がり率は、現在のデータと 1 分前のデータを比較して算出します。

値上がり率を算出するためには、次の 2 種類のクエリを使用します。

1 分間データ算出クエリ

```
REGISTER QUERY stockDStream
DSTREAM ( SELECT * FROM stock[RANGE 1 MINUTE] );
```

- 範囲を 1 分間と指定した RANGE ウィンドウによって、入力レシーションのタプルを 1 分間保持します。
- DSTREAM を指定して、1 分間の生存期間が終了したタプルを出力します。これによって、現在から 1 分前のデータが、ストリームデータとして出力されます。

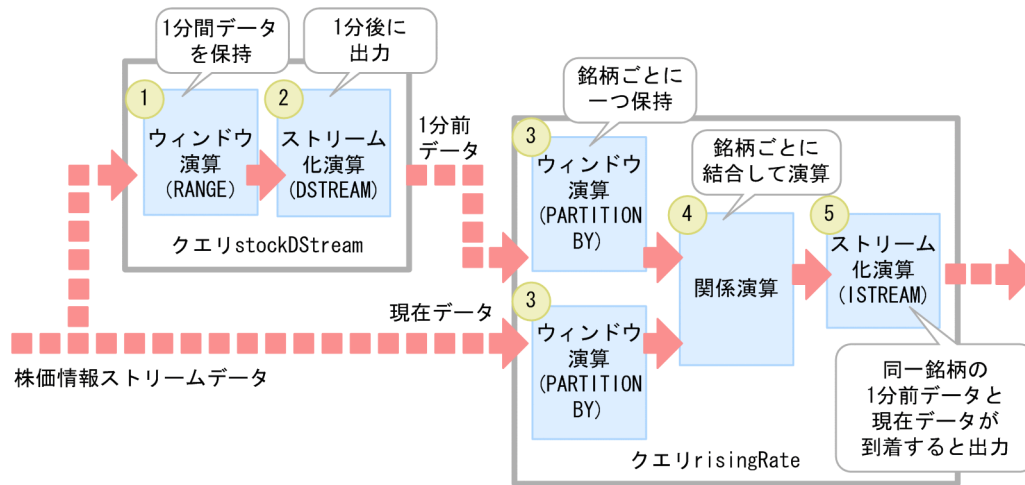
値上がり率計算クエリ

```
REGISTER QUERY risingRate
ISTREAM ( SELECT stock.stockID, stock.stockName,
stock.currentPrice / stockDStream.currentPrice AS stockRate
FROM stock[PARTITION BY stock.stockID ROWS 1],
stockDStream[PARTITION BY stockDStream.stockID ROWS 1]
WHERE stock.stockID = stockDStream.stockID );
```

- PARTITION BY ウィンドウを使用して、銘柄ごとに最新の 1 件のタプルを保持します。
- 現在の株価情報ストリームデータと 1 分前のストリームデータを統合して、値上がり率を計算します。

これらのクエリを使用した演算処理の流れを次の図に示します。

図 2-19 株価値上がり率を算出する演算処理の流れ



処理の流れについて説明します。説明の番号は図中の番号と対応しています。

1. [クエリ stockDStream のウィンドウ演算]

ウィンドウ演算を実行します。

「FROM stock」の指定によって入力したストリーム stock を、「[RANGE 1 MINUTE]」の指定によって 1 分間保持します。

2. [クエリ stockDStream のストリーム化演算]

1 分経過したデータは、ストリーム化演算「DSTREAM」によってストリームデータ（1 分前データ）として出力されます。

3. [クエリ risingRate のウィンドウ演算]

ストリーム stock と、2.で出力した結果ストリーム（1 分前データ）の 2 種類のデータを「FROM stock..., stockDStream...」の指定によって入力します。

「[PARTITION BY... ROWS 1]」の指定によって、入力データを銘柄コード (stockID) ごとにグルーピングした上で、銘柄コードごとの最新 1 件のデータを保持します。これによって、銘柄コードごとに、次のデータが保持されます。

- 入力ストリーム stock による現在の株価データのうち、最新の 1 件
- 入力ストリーム stockDStream（クエリ stockDStream の結果ストリーム）による 1 分前の株価データのうち、最新の 1 件

4. [クエリ stockDStream の関係演算]

銘柄コードごとに、銘柄コード、銘柄名、および値上がり率（現在の株価/1 分前の株価）を結合します。

5. [クエリ stockDStream ストリーム化演算]

同一銘柄の 1 分前データと現在データが到着したタイミングで、4.の結合データを結果ストリームとして出力します。

2.6.2 時刻解像度を指定した定義例

ここでは、時刻解像度を指定したクエリの定義例として、次の 2 種類の例について説明します。

- 一つの集合関数を定義する例
- 複数の集合関数を定義する例

この例では、1 分間の RANGE ウィンドウ内にある到着タプルに対して、集合関数による関係演算を実行します。この例で扱うストリームは次のとおりです。

時刻解像度の指定例で扱うストリーム

- ストリーム名は「stock」である。
- 「price」という名称の列がある。

(1) 一つの集合関数を定義する例

ストリーム stock に対して、RANGE ウィンドウ内にある到着タプルから列名 price の合計値を求める場合の定義例を次に示します。

```
REGISTER STREAM stock(price INTEGER, name VARCHAR(10));           ... (1)
REGISTER QUERY_ATTRIBUTE q1 STREAM_NAME=stock PERIOD=300MS TARGETS=SUM(price);           ... (2)
REGISTER QUERY q1                                               ... (3)
ISTREAM(
  SELECT name, SUM (price) AS s1
  FROM stock[RANGE 1 MINUTE]
  GROUP BY name
);
```

注

(1)~(3)は次に示す説明と対応している番号です。実際の定義には不要です。

時刻解像度を指定するクエリの名称は、(3)で指定している q1 です。このため、(2)の QUERY_ATTRIBUTE 文でも、クエリ名として q1 を指定します。

この例は、(1)で指定しているストリーム名 stock について、時刻解像度を指定して列名 price の合計値を求める例です。このため、(2)の STREAM_NAME= のデータ識別子には stock を指定して、TARGETS= には集合関数 SUM(price) を指定します。

合計値ではなく、最大値または最小値を求める場合には、集合関数 MAX または MIN を指定してください。

(2) 複数の集合関数を定義する例

ストリーム stock に対して、RANGE ウィンドウ内にある到着タプルから列名 price の合計値、最大値および最小値を求める場合の定義例を次に示します。

```
REGISTER STREAM stock(price INTEGER, name VARCHAR(10));
REGISTER QUERY q0
ISTREAM(
  SELECT name, price AS price0, price AS price1, price AS price2
  FROM stock[NOW]
);
REGISTER QUERY_ATTRIBUTE q1 STREAM_NAME=q0 PERIOD=300MS
TARGETS=SUM(price0),MAX(price1),MIN(price2);
REGISTER QUERY q1
ISTREAM(
  SELECT q0.name, SUM(q0.price0) AS sum_price,
         MAX(q0.price1) AS max_price, MIN(q0.price2) AS min_price
  FROM q0[RANGE 1 MINUTE]
  GROUP BY q0.name
);
```

時刻解像度を指定して複数の集合演算を実行する場合、同じデータ識別子を持つ同一名称の列名に対して、複数の集合演算は実行できません。例えば、データ識別子 stock の列名 price のクエリに対して、「TARGETS=SUM(price),MAX(price),MIN(price)」のような指定はできません。

2 CQLによるクエリの定義

複数の集合関数を指定する場合は、集合関数の対象とする列名をそれぞれ用意する必要があります。この例では、SUM、MAX、MINそれぞれの対象とする列名として、列名 price を price0、price1、price2 という列名に置き換えた上で、それぞれの集合関数による演算を実行しています。

3

CQL の基本項目とデータ型

この章では、CQL の基本項目とデータ型について説明します。

3.1 CQLによるクエリ定義

クエリは、CQLを使用して定義します。CQLは、次の項目（基本項目）を組み合わせて処理を記述します。

- キーワード
- 数値
- 区切り文字
- 名前
- 定数

キーワードは、「SELECT」や「WHERE」のように、データ処理のための機能を示す文字列です。キーワードに対して、処理対象や処理範囲などを定義するために指定する数値、区切り文字、名前、定数などは、オペランドといいます。

これらの基本項目を指定する場合は、使用できる文字、データ型などに留意する必要があります。

ここでは、CQLでクエリを定義する際の、基本項目の記述形式および使用できる文字について説明します。また、以降のCQLの文法説明で使用する記号についても説明します。

3.1.1 CQLの記述形式

CQLは、次の内容に従って記述してください。

- CQLは、フリーフォーマット形式です。オペランドは、各CQLの形式で記述している順序に従って指定します。CQLの形式については、「4. CQLリファレンス」を参照してください。
- CQLの文は、セミコロン (;) で区切って記述します。

一つの文は複数行にわたって記述できます。1行に複数の文を記述することはできません。

正しい記述例と誤った記述例を次に示します。

(正しい記述例)

```
REGISTER STREAM
  s1(id INT,name VARCHAR(10));
REGISTER QUERY q1
  SELECT s1.name FROM s1[ROWS 10];
```

(誤った記述例)

```
REGISTER STREAM s1(id INT); REGISTER QUERY q1 SELECT * FROM s1[ROWS 10];
```

二つのREGISTER文を1行に記載しているため、誤りとなります。

- 注釈は、ダブルスラッシュ (//) で指定します。ダブルスラッシュを指定した場合、ダブルスラッシュを含めた以降の文は注釈として扱われます。この場合、注釈の有効範囲は1行です。一つの定義文の途中で注釈を挿入することはできません。

正しい記述例と誤った記述例を次に示します。

(正しい記述例)

```
// 注釈
REGISTER STREAM
  s1(id INT,name VARCHAR(10)); // 注釈
```

(誤った記述例)

```
REGISTER QUERY q1 // 注釈
  SELECT s1.name FROM s1[ROWS 10];
```

REGISTER QUERY 文の途中に注釈を挿入しているため、誤りとなります。

なお、文字列中に注釈以外の用途で「//」を指定する場合、「//」が行の先頭にならないように記述してください。正しい記述例と誤った記述例を次に示します。

(正しい記述例)

```
REGISTER QUERY q1 SELECT * FROM s1[NOW] WHERE s1.c1=' ab//cd' ;
```

(誤った記述例)

```
REGISTER QUERY q1 SELECT * FROM s1[NOW] WHERE s1.c1=' ab
//cd' ;
```

行の先頭が「//」となっているため、誤りとなります（注釈行と見なされてしまいます）。

3.1.2 CQL で使用できる文字

ここでは、CQL で使用できる文字について説明します。

(1) 使用できる文字コード

CQL で使用できる文字コードを次に示します。

- Windows の場合：MS932
- Linux の場合：UTF-8

(2) 使用できる文字

CQL で使用できる文字を次の表に示します。

表 3-1 CQL で使用できる文字

項番	種別	使用できる文字
1	名前	半角英数と下線 (_) が使用できます。ただし、先頭に使用できる文字は、半角英文字だけです。名前の指定については、「3.2.4 名前の指定」を参照してください。
2	文字列定数	半角文字コードおよび全角文字コードが使用できます。
3	上記以外	次に示す半角文字コードの文字および特殊記号が使用できます。 <ul style="list-style-type: none"> • 半角文字コード 英大文字 (A~Z)、英小文字 (a~z)、数字 (0~9)、空白、下線文字 (_) • 特殊記号 コンマ (,) : 選択リスト、値式など ピリオド (.) : 浮動小数点定数、列指定、日付/時刻データなど コロン (:) : 時刻/時刻印データ ハイフン (-) : 日付/時刻印データ 引用符 (') : 文字列の囲み 左括弧 (()) : リレーション式、キャスト指定の囲みなど 小なり演算子 (<) , 大なり演算子 (>) , 等号演算子 (=) , 感嘆符 (!) : 比較演算子 正符号 (+) , 負記号 (-) , 斜線 (/) : 算術演算子 アスタリスク (*) : 算術演算子、すべての列出力等 セミコロン (;) : 文の区切り 角括弧 ([]) : 時間指定の囲み、タブコード、改行コード、復帰コード 斜線 (/) + 斜線 (/) : 注釈 ドル記号、アスタリスクの組み合わせ (\$*) : 外部定義関数

3.1.3 CQL 文法説明で使用する記号

CQL の文法説明で使用する記号の用法を次の表に示します。以降、3 章および 4 章では、これらの記号を使用して CQL の文法を説明します。

表 3-2 CQL の文法説明で使用する記号

記号	意味	例
{ }	この記号で囲まれた複数の項目から、一つを選択することを示します。	{文字列定数 数定数} 文字列定数、または数定数のどちらかを選択して記述します。
[]	この記号で囲まれた項目は省略できることを示します。複数の項目が並べて記述されている場合は、すべてを省略するか、記号 { } と同じくどれか一つを選択します。	[SECOND MILLISECOND] すべてを省略するか、SECOND または MILLISECOND のどちらかを選択して記述します。
…	この記号の直前に示された項目を繰り返し複数個指定できることを示します。	リレーション参照 [,リレーション参照] … リレーション参照を繰り返し複数個記述します。
'()'	この記号で囲まれた項目は、() を省略しないでそのまま記述することを示します。	'(TINYINT)' TINYINT の前後を() で囲んで記述します。
'[]'	この記号で囲まれた項目は、[] を省略しないでそのまま記述することを示します。	'[ウィンドウ指定]' ウィンドウ指定の前後を[] で囲んで記述します。
::=	この記号の左にあるものを右にあるもので定義することを示します。	選択リスト ::= 選択式 [,選択リスト]
▲ _n	n 個以上の区切り文字を示します。n が省略されている場合、n=1 を仮定します。	NOT {▲ ₀ '(探索条件)' ▲比較述語} NOT と(探索条件)の間に 0 個以上の区切り文字、または NOT と比較述語の間に 1 個以上の区切り文字を記述します。

3.2 CQLの基本項目の指定方法

この節では、CQLの基本項目の指定方法について説明します。

3.2.1 キーワードの指定

CQL文の名称 (SELECT, UNION など) のように、機能を使用するために指定する語をキーワードといいます。

キーワードの意味については、「4. CQL リファレンス」を参照してください。

キーワードの指定例を次の図に示します。

図 3-1 キーワードの指定例

REGISTER	STREAM	(ID	C1	CHAR	,	C2	INTEGER)	;
_____	_____		_____	_____	_____		_____	_____		
キーワード	キーワード		名前	名前	キーワード		名前	キーワード		
					(予約語)			(予約語)		

ほとんどのキーワードは、システムの予約語として登録されているため、決められた位置以外には指定できません。

ただし、予約語として登録されていないキーワードの文字列は、名前としても使用できます。その場合、大文字、小文字の区別はされません。

予約語としてシステムに登録されている文字列を、先頭文字のアルファベットごとに次の表に示します。

表 3-3 予約語としてシステムに登録されている文字列

先頭文字	予約語
A	ALL, AND, ANY, AS, ASC, AST, AVG
B	BETWEEN, BIGINT, BIGINT_TOSTRING, BINARY, BOOLEAN, BY
C	CASE, CHAR, CHARACTER, CON, CONCAT, CORREL, COUNT, COVAR, COVAR_POP
D	DABS, DACOS, DASIN, DATAN, DATAN2, DATE, DAY, DCEIL, DCOS, DCOSH, DDISTANCE, DDISTANCE3, DEC, DECIMAL, DECIMAL_TOSTRING, DEFAULT, DELETABLE, DELETE, DESC, DEXP, DFLOOR, DISTINCT, DLN, DLOG, DMOD, DNAN, DNEGATIVE_INFINITY, DOUBLE, DOUBLE_TOSTRING, DPI, DPOSITIVE_INFINITY, DPOWER, DROUND, DSIN, DSINH, DSQRT, DSTREAM, DTAN, DTANH, DTODEGREES, DTORADIANS
E	ELSE, END, EOF, EQ, EXISTS
F	FALSE, FLOAT, FROM
G	GE, GT, GROUP
H	HAVING, HOUR
I	IABS, IDSTREAM, IMOD, IN, INSERT, INSTANT, INT, INT_TOSTRING, INTEGER, IPOWER, IS, ISTREAM
J	予約語はありません。

先頭文字	予約語
K	予約語はありません。
L	LABS, LATEST, LE, LENGTH, LEQ, LGE, LGT, LIKE, LIMIT, LLE, LLP, LLT, LMOD, LP, LPOWER, LRP, LT
M	MAX, MILLISECOND, MIN, MINUTE, MIS
N	NE, NOT, NOW, NROUND, NULL, NUMBER_TODATE, NUMBER_TOTIME, NUMERIC
O	OR, ORDER, OTHER
P	P_INT, PARTITION, PLS, POM, PR, PREV
Q	予約語はありません。
R	RABS, RACOS, RANGE, RANKING, RASIN, RATAN, RATAN2, RCEIL, RCOS, RCOSH, RDISTANCE, RDISTANCE3, REAL, REAL_TOSTRING, RECENT, REGEXP_FIRSTSEARCH, REGEXP_REPLACE, REGEXP_SEARCH, RELATIONAL, REXP, RFLOOR, RLN, RLOG, RMOD, RNAN, RNEGATIVE_INFINITY, ROWS, RPI, RPOSITIVE_INFINITY, RPOWER, RROUND, RSIN, RSINH, RSQRT, RSTREAM, RTAN, RTANH, RTODEGREES, RTORADIANS
S	SECOND, SELECT, SMALLINT, SOME, SLENGTH, SQU, STDDEV, STDDEV_POP, SUB, SUM
T	THEN, TIME, TIME_TONUMBER, TIMEDIFF, TIMESTAMP, TIMESTAMP_TONUMBER, TIMESTAMPDIFF, TINYINT, TRUE
U	UNBOUNDED, UNION, UNKNOWN, UPDATE
V	VAR, VAR_POP, VARBINARY, VARCHAR
W	WHEN, WHERE, WITH, WQU
X	予約語はありません。
Y	予約語はありません。
Z	予約語はありません。

3.2.2 数値の指定

CQL中に記述する値式以外の数値（データ型の括弧中の指定、時間、行数など）は、符号なし整数の表記法に従って指定します。

3.2.3 区切り文字の挿入

(1) 区切り文字の種類

区切り文字として指定できるのは、次の文字です。

- 半角空白
- 復帰コード
- 改行コード
- タブコード

なお、区切り文字に空白を指定する場合は、半角空白を使用してください。全角空白は区切り文字とは見なされません。

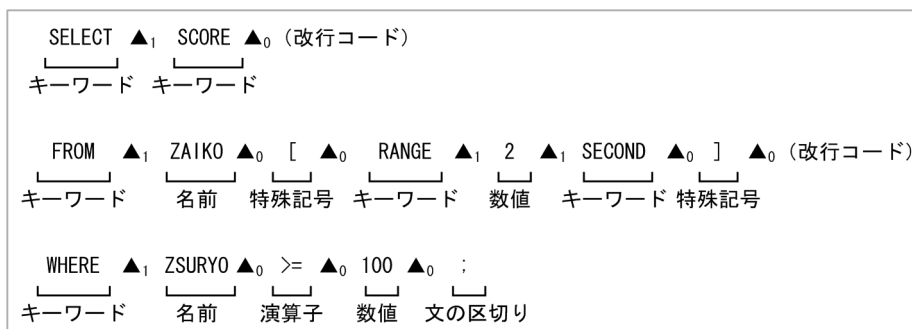
(2) 区切り文字を挿入する位置

区切り文字を挿入する位置を次に示します。

- キーワードとキーワードの間
- キーワードと名前の間
- キーワードと数値の間

区切り文字の挿入例を次の図に示します。

図 3-2 区切り文字の挿入例



(凡例)

- ▲₀ : 0個以上の半角空白。
- ▲₁ : 1個以上の半角空白。

(3) 区切り文字を挿入できる位置

区切り文字は、次に示す特殊文字の前後で、かつ「(4) 区切り文字を挿入できない位置」で挿入を禁止されていない位置に挿入できます。

「,」 「.」 「-」 「+」 「*」 「'」 「(」 「)」 「<」 「>」 「=」 「^」 「!」 「#」 「\$」 「/」 「[」 「]」 「復帰コード」 「改行コード」 「タブコード」 「;」

ただし、「'」を使用する文字列定数部分、および「>=」などの比較演算子の場合は、特殊記号と特殊記号の間に区切り文字を挿入できません。

区切り文字を挿入できる位置の例を次に示します。

(例 1) 特殊文字 (.) の前に空白を挿入

SCORE▲.NAME

(例 2) 特殊文字 (=) のあとに空白を挿入

SCORE=▲1

(4) 区切り文字を挿入できない位置

区切り文字を挿入できない位置を次に示します。

- キーワードの途中
(例) S▲ELECT
- 名前の途中

(例) ZA▲IKO

- 数定数の途中

(例) 678▲9

- 演算子の途中

(例) <▲=

- 外部定義関数の記号「\$*」の特殊記号と特殊記号の間

(例) \$▲*

3.2.4 名前の指定

名前の指定についての規則を次に示します。

- 名前に使用できる文字は、半角英数字と下線 (_) です。
- 名前の先頭に使用できる文字は、半角英文字だけです。数字や下線 (_) は使用できません。
- 名前の指定では、大文字と小文字は区別されません。すべて半角大文字として扱われます。
- 名前に空白を含めることはできません。
- CQLの予約語と重複した名前は指定できません (予約語でないキーワードとは重複できます)。
- 名前を引用符 (") で囲むことはできません。
- 名前として指定できる文字数は 1~100 文字です。

3.2.5 名前の修飾

名前の修飾は、「データ識別子.列名」のように、ピリオド (.) によってデータ識別子と列名を連結して、名前を一意にするときに使用します。なお、データ識別子と列名を連結した名前の指定方法を、**列指定**といいます。

(1) データ識別子指定

データ識別子指定とは、一つのCQL文中に二つ以上のデータ識別子を指定する場合に、指定する列名がどのストリーム、リレーションまたは相関名に対応するかを一意にするための修飾子を使用した指定方法です。

次の指定形式に示すように、修飾子としてストリーム名、リレーション名、または相関名を指定します。

データ識別子の指定形式

データ識別子::={ストリーム名 | リレーション名 | 相関名}

ストリーム名およびリレーション名については、「4.4.13 リレーション参照」を参照してください。また、相関名については、「4.4.11 選択式」および「4.4.13 リレーション参照」を参照してください。

(2) 列指定

列指定とは、データ識別子で修飾された列名のことです。

複数のデータ識別子を指定した検索 (二つ以上のデータ識別子を結合した検索) をする場合に、検索対象の複数のデータ識別子と同じ名称の列名があるとき、列指定によって指定した列がどのデータ識別子に対応する列なのかを明確にします。

指定形式を次に示します。

列指定の指定形式

列指定 ::= データ識別子.列名

データ識別子で列名を修飾する場合、指定するデータ識別子が有効な範囲内の列名を指定する必要があります。データ識別子の有効範囲は、FROM句に指定したリレーション参照のリレーション、ストリーム、または相関名で定義されている列名です。定義されていない列名を指定した場合、エラーとなります。

問い合わせの結果として取得されるストリームおよびリレーションの名前には、問い合わせを定義しているクエリ名と同じ名前が付けられます。

列名は、次のどちらかに含まれる必要があります。

- 列名を指定する位置が有効範囲であるデータ識別子によって示されるストリームまたはリレーション
- 問い合わせ結果として取得されるストリームまたはリレーション

ストリームおよびリレーションの列名は、次の規則に従って問い合わせ結果として取得されます。

- UNION句を持つリレーション式で取得されるリレーションの場合、1番目に指定したリレーション式（UNION句を除く）の結果として取得されるリレーションの列名が、問い合わせ後の結果として取得されるリレーションの列名になります。

次の例では、q2のリレーション参照に指定したq1から導出される列名はc1だけです。d1およびe1は取得されません。

```
REGISTER QUERY q1
  SELECT s1.c1 FROM s1[NOW] ...
  UNION
  SELECT s2.d1 FROM s2[NOW] ...
  UNION
  SELECT s3.e1 FROM s3[NOW] ...;
```

```
REGISTER QUERY q2
  SELECT q1.c1 FROM q1 ...;
```

- UNION句を持たないリレーション式で取得されるリレーションの場合、i番目の選択式によって取得される列名が、i番目に取得されるリレーションの列名になります。

次の例では、q2のリレーション参照に指定したq1から取得される列名は、q1の選択式に指定した順番に従い、列名c1、d1、e1の順に取得されます。

```
REGISTER QUERY q1
  SELECT s1.c1, s2.d1, s3.e1 FROM s1[NOW], s2[NOW], s3[NOW];
```

```
REGISTER QUERY q2
  SELECT * FROM q1;
```

- ストリーム句で取得されるストリームの場合、ストリーム句で囲ったリレーション式の結果として取得されるリレーションの列名が、問い合わせ後の結果として取得されるストリームの列名になります。
- ストリーム間演算関数で導出されるストリームの場合、ストリーム間演算の結果として導出されるストリームの列名が、問い合わせ後の結果として導出されるストリームの列名になります。

ストリーム句については、「4.4.2 ストリーム句」を参照してください。リレーション式については、「4.4.3 リレーション式」を参照してください。また、ストリーム間演算関数については、「4.4.24 ストリーム間演算関数」を参照してください。

(3) 範囲変数

列指定の修飾子となる識別子を範囲変数といいます。範囲変数は、有効範囲と名称を持ちます。また、範囲変数で指定されたデータ識別子は、表識別子ともいいます。

相関名は範囲変数です。相関名が指定された場合、相関名で指定された元のデータ識別子は、列名の有効範囲ではなくなります。

リレーション参照、選択式に相関名を指定した場合の列名の有効範囲について、例を次に示します。

- リレーション参照に相関名を指定した場合

(正しい記述例)

```
REGISTER QUERY q1 SELECT a1.a FROM s1[NOW] AS a1 WHERE a1.a > 1;
```

(誤った記述例)

```
REGISTER QUERY q2 SELECT s1.a FROM s1[NOW] AS a1 WHERE s1.a > 1;
```

s1は相関名a1として定義されているため、列名aの有効範囲ではなくなっています。このため、SELECT句やWHERE句の列指定にs1は使用できません。

- 選択式に相関名を指定した場合

(正しい記述例)

```
REGISTER QUERY q1 SELECT s1.a AS a1 FROM s1[NOW];
REGISTER QUERY q2 SELECT a1 FROM q1 WHERE a1 > 1;
```

(誤った記述例)

```
REGISTER QUERY q1 SELECT s1.a AS a1 FROM s1[NOW];
REGISTER QUERY q3 SELECT q1.a FROM q1 WHERE q1.a > 1;
REGISTER QUERY q4 SELECT a FROM q1 WHERE a > 1;
```

s1.aは相関名a1として定義されているため、列名aの有効範囲ではなくなっています。このため、SELECT句やWHERE句の列指定にq1.aやaは使用できません。

3.2.6 定数の指定

定数は、プログラム中で値を変更できないデータです。ここでは定数の指定について説明します。なお、定数で指定する値のデータ型については、「3.3 CQLのデータ型」を参照してください。

(1) 定数の種類と表記方法

定数には、数値を表す**数定数**と文字列を表す**文字列定数**があります。

数定数には、次の種類があります。

- 整数定数
- 浮動小数点定数
- 10進定数

文字列定数には、次の種類があります。

- 文字列
- 日付データ
- 時刻データ
- 時刻印データ

なお、日付データおよび時刻データを指定する場合、指定値は、Stream Data Platform - AFが動作しているJavaVMのタイムゾーンに従った表現で指定する必要があります。

定数の表記方法を次の表に示します。

表 3-4 定数の表記方法

項番	定数		表記方法		ストリームデータ処理エンジンが解釈するデータ型
1	数定数	整数定数	INTEGER の場合 [符号] 整数 BIGINT の場合 [符号] 整数 [L] l] (例) -123 45 6789L	整数は、数字の並びで表します。 符号は、+または-で表します。 文字列 L または l を末尾に付けた定数のデータ型は BIGINT 型になります。	INTEGER, BIGINT
		浮動小数点定数	小数点形式 [符号] 整数部. 小数部 (例) -12.3 456.0 0.789	整数部と小数部を整数で表して、小数点(.)で区切ります。整数部、小数部、および小数点(.)は必ず付けてください。 符号は、+または-で表します。	DOUBLE
		10 進定数	小数点形式 [符号] 整数部 [小数部] {D d} (例) -12.3D 456.0d 0.789d -123D 45d	整数部と小数部を整数で表して、小数点(.)で区切ります。 符号は+、または-で表します。 整数部、末尾の文字 D、または d は必ず付けてください。 小数部および小数点(.)は、省略できます。	DECIMAL NUMERIC
2	文字列定数	'文字列' (例) 'HITACHI' '98' 'DB0002'	文字列は半角文字、および全角文字の列で表し、アポストロフィ (') で囲みます*。 文字列にアポストロフィ (') を含める場合は、1 個のアポストロフィ (') を表すのにアポストロフィ (') を続けて 2 個記述してください。 例えば、文字列としてアポストロフィ'を指定する場合は、「'''」となります。 日付データ、時刻データおよび時刻印データの指定形式の詳細については、「(2) 日付データの規定の文字列表現」「(3) 時刻データの規定の文字列表現」および「(4) 時刻印データの規定の文字列表現」を参照してください。	VARCHAR	

注※

文字の囲みに引用符 (") は使用できません。また、文字列中で引用符 (") は使用できません (文字列中の引用符 (") はエスケープできません)。

(2) 日付データの規定の文字列表現

日付データは、次の形式で指定します。

'YYYY-MM-DD'

YYYY

0001~9999 (年)

MM

01~12 (月)

DD

01~MM で指定した月の最終日 (日)

日付を既定の文字列表現の定数にする場合、年 (YYYY)、月 (MM)、日 (DD) をハイフン (-) でつないで表現します。年 (YYYY)、月 (MM)、日 (DD) のけた数に満たない場合は、足りないけた数分だけ左側に 0 を補ってください。また、数字とハイフンの間には空白を挿入しないでください。

日付を規定の文字列表現の定数として指定した例を次に示します。

2010年3月25日の場合の指定例

'2010-03-25'

日付データを指定する際は、次の点に注意してください。

- 形式に従わないで指定した文字列は、日付データとして認識されません。単なる文字列定数として認識されます。
- 年、月、日に、範囲外の値を指定した場合、クエリグループ登録時にメッセージ KFSP32014-E がメッセージログファイルに出力されます。

(3) 時刻データの規定の文字列表現

時刻データは、次の形式で指定します。

'hh:mm:ss'

hh

00~23 (時)

mm

00~59 (分)

ss

00~59 (秒) ※

注※

うるう秒の場合は 00~61 (秒) です。

時刻を既定の文字列表現の定数にする場合、時 (hh)、分 (mm)、秒 (ss) をコロン (:) でつないで表現します。時 (hh)、分 (mm)、秒 (ss) のけた数に満たない場合は、足りないけた数分だけ左側に 0 を補ってください。また、数字とコロンの間には空白を挿入しないでください。

時刻を規定の文字列表現の定数として指定した例を次に示します。

午後 10 時 8 分 26 秒の場合の指定例`'22:08:26'`

時刻データを指定する際は、次の点に注意してください。

- 形式に従わないで指定した文字列は、時刻データとして認識されません。単なる文字列定数として認識されます。
- 時、分、秒に、範囲外の値を指定した場合、次の例に示すように、値が繰り上げられて認識されます。

範囲外の値を指定した場合の例表記：`'25:08:26'`

認識される時刻：午前 1 時 8 分 26 秒

(4) 時刻印データの規定の文字列表現

時刻印データは、次の形式で指定します。

`'YYYY-MM-DD▲hh:mm:ss [.SSSSSSSS]'`

YYYY

0001～9999 (年)

MM

01～12 (月)

DD

01～MM で指定した月の最終日 (日)

hh

00～23 (時)

mm

00～59 (分)

ss

00～59 (秒) ※

注※

うるう秒の場合は 00～61 (秒) です。

SSSSSSSS

0～9 けたの小数秒 (S:0～9)

時刻印データを既定の文字列表現の定数にする場合、年 (YYYY)、月 (MM)、日 (DD) をハイフンでつなぎ、半角空白を指定して、時 (hh)、分 (mm)、秒 (ss) をコロン (:) でつないで表現します。その際、年 (YYYY)、月 (MM)、日 (DD)、時 (hh)、分 (mm)、秒 (ss) のけた数に満たない場合、足りない数分だけ左側に 0 を補ってください。数字とハイフン、数字とコロンの間に空白を入れてはいけません。ただし、日 (DD) と時 (hh) の間には、半角空白を必ず入れてください。

時刻印データを文字列表現の定数として指定した例を次に示します。

2008 年 8 月 1 日 午後 10 時 8 分 26 秒の場合の指定例`'2008-08-01 22:08:26'`

時刻印データを指定する際は、次の点に注意してください。

- 形式に示した書式に従わないで指定した文字列は、時刻印データとして認識されません。単なる文字列定数として認識されます。
- 小数秒の精度を表現する場合、秒 (ss) と小数秒 (SSSSSSSS) の間を、ピリオドでつないで指定してください。小数秒精度を省略した場合、小数秒精度が0のデータとして扱われます。小数秒が3けたの場合は1ミリ秒単位、6けたの場合は1マイクロ秒単位、9けたの場合は1ナノ秒単位の指定を意味します。
- 形式で示した範囲外の値を指定した場合、次の例に示すように、値が繰り上げられて認識されます。

範囲外の値を指定した場合の例

表記: '2008-02-30 25:08:26'

認識される時刻: 2008年3月2日01時08分26秒

3.3 CQL のデータ型

ここでは、CQL のデータ型について説明します。

3.3.1 CQL のデータ型と Java のデータ型のマッピング

CQL のデータ型は、Java のデータ型にマッピングされます。

CQL のデータ型と Java のデータ型のマッピングについて、次の表に示します。

表 3-5 CQL のデータ型と Java のデータ型のマッピング

分類	CQL のデータ型	データ形式	Java のデータ型	データの範囲	備考
数データ	INT [EGER]	整数型 4 バイト	Integer クラス	-2,147,483,648～ 2,147,483,647	—
	SMALLINT	整数型 2 バイト	Short クラス	-32,768～32,767	—
	TINYINT	整数型 1 バイト	Byte クラス	-128～127	—
	BIGINT	整数型 8 バイト	Long クラス	-9,223,372,036,8 54,775,808～ 9,223,372,036,85 4,775,807	—
	DEC [IMAL] ['(m)'] ※1	10 進形式	java.math.Big Decimal クラス	-10 ³⁸⁺¹ ～10 ³⁸⁻¹	精度 (全体のけた数) が m けた ('+', '-' の符 号は含みません) の 10 進数です。 m は正整数で、 1 ≤ m ≤ 38 です。m を 省略すると、15 が仮定 されます。
	NUMERIC ['(m)'] ※1				
	REAL	実数型 4 バイト	Float クラス	<ul style="list-style-type: none"> -3.402823466 E +38～-1.1754 94351E-38 0 1.175494351E -38～ 3.402823466E +38 	入力データに指数表現 は使用できません。
	FLOAT	実数型 8 バイト	Double クラス	<ul style="list-style-type: none"> -1.797693134 8623157E +308～-2.225 07385850720 14E-308 0 2.2250738585 072014E-308 ～ 	入力データに指数表現 は使用できません。
DOUBLE					

3 CQLの基本項目とデータ型

分類	CQLのデータ型	データ形式	Javaのデータ型	データの範囲	備考
数データ	DOUBLE	実数型 8 バイト	Double クラス	1.7976931348 623157E+308	入力データに指数表現は使用できません。
文字データ	CHAR [ACTER] ['(n)']	固定長文字列 (文字数 n 個)	java.lang.String クラス	1~255	n は正整数です。 $1 \leq n \leq 255$ です。n を省略すると、1 が仮定されます。列には、列を埋めるために必要なだけ空白が追加されます。データの長さが指定文字数 n を超えた場合は、エラーになります。
	VARCHAR ['(n)']	可変長文字列 (最大文字数 n 個)		1~32,767	n は正整数です。 $1 \leq n \leq 32,767$ です。列には、列を埋めるための空白は追加されません。データの長さが指定文字数 n を超えた場合は、エラーになります。
日付データ	DATE*2	日付 (年月日)	java.sql.Date クラス	YYYYMMDD YYYY : 0001~9999 (年) MM : 01~12 (月) DD : 01~該当年月の最終日 (日)	—
時刻データ	TIME*2	時間 (時分秒)	java.sql.Time クラス	hhmmss hh : 00~23 (時) mm : 00~59 (分) ss : 00~59 (秒)	—
時刻印データ	TIMESTAMP ['(p)'] *2	日時 (年月日+時分秒+ナノ秒)	java.sql.Timestamp クラス	YYYYMMDDhhmmss [nn...n] YYYY : 0001~9999 (年) MM : 01~12 (月) DD : 01~該当年月の最終日 (日) hh :	p は正整数です。 $0 \leq p \leq 9$ です。p を省略すると、3 が仮定されます。 p のけたは、3 けたでミリ秒、6 けたでマイクロ秒、9 けたでナノ秒単位となります。p を超えた場合は、エラーになります。

分類	CQL のデータ型	データ形式	Java のデータ型	データの範囲	備考
時刻印データ	TIMESTAMP ['(p)'] ※2	日時 (年月日+時分秒+ナノ秒)	java.sql.Timestamp クラス	00~23 (時) mm : 00~59 (分) ss : 00~59 (秒) nn...n : p けたの小数秒 (n: 0~9)	p は正整数です。 0 ≤ n ≤ 9 です。p を省略すると、3 が仮定されます。 p のけたは、3 けたでミリ秒、6 けたでマイクロ秒、9 けたでナノ秒単位となります。p を超えた場合は、エラーになります。

(凡例)

— : 該当しません。

注※1

DECIMAL 型および NUMERIC 型についての注意事項は、「3.3.2 DECIMAL 型および NUMERIC 型についての注意事項」を参照してください。

注※2

DATE 型、TIME 型および TIMESTAMP 型のデータには範囲外の値を指定しないように注意してください。これらのデータ型に範囲外の値を指定した場合、例外などは発生しないで、意図しない処理が実行されるおそれがあります。

なお、「データ型(n)」のように、データ型の後ろに()で囲んで記載する数値は、型の長さを示します。

参考

CQL のデータ型と Java のデータ型とのマッピングは、SQL のデータ型と Java のデータ型とのマッピングを定義した JDBC API に従っています。

3.3.2 DECIMAL 型および NUMERIC 型についての注意事項

DECIMAL 型および NUMERIC 型のデータで、送信データが指定したけた数を超えた場合、エラーになります。

また、次の処理を実行した場合に、結果のけた数が query.decimalMaxPrecision パラメーターで指定したけた数を超えた場合、指定したけた数に丸められて表現されます (これらの処理を実行しない場合、値は丸められません)。

- 四則演算 (2 項演算) の演算項、または集合関数 SUM もしくは AVG の引数に DECIMAL 型 (NUMERIC 型) が含まれる演算を実行した場合
- DECIMAL 型 (NUMERIC 型) 以外のデータ型から DECIMAL 型 (NUMERIC 型) へのキャスト (暗黙的なキャストを含む) を実行した場合

値をどのように丸めて表現するかは、query.decimalRoundingMode パラメーターで指定します。パラメーターの詳細については、マニュアル「uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド」を参照してください。

値の丸め方の例を次の表に示します。なお、この例は query.decimalMaxPrecision パラメーターに「1」(けた数は 1 に丸める) と設定している場合です。

表 3-6 値の丸め方の例 (query.decimalMaxPrecision=1 の場合)

演算結果 (けた数 2)	query.decimalRoundingMode パラメーターの指定値	
	DOWN	HALF_UP
5.5	5	6
2.5	2	3
1.6	1	2
1.1	1	1
1.0	1	1
-1.0	-1	-1
-1.1	-1	-1
-1.6	-1	-2
-2.5	-2	-3
-5.5	-5	-6

なお、DECIMAL 型 (NUMERIC 型) の数定数 (10 進定数) の場合、けた数のチェックは実行されません。値がデータの範囲外の場合、query.decimalMaxPrecision パラメーターおよび query.decimalRoundingMode パラメーターの指定値に関係なく、CQL に記述した値として表現されます。

DECIMAL 型 (NUMERIC 型) の送信データ、演算結果、数定数について、けた数の指定方法と指定したけた数を超えた場合の動作を次の表に示します。

表 3-7 DECIMAL 型 (NUMERIC 型) のけた数の指定方法とけた数を超えた場合の動作

種類	けた数の指定方法	指定したけた数を超えた場合の動作
送信データ	スキーマ指定文字列の型名	データ送信時にけた数を超えていた場合、エラーになります。
演算結果	<ul style="list-style-type: none"> 四則演算 集合関数 SUM/AVG キャスト 	query.decimalMaxPrecision パラメーターで指定したけた数になるよう、query.decimalRoundingMode パラメーターで指定した方法で丸められます。
	上記以外	演算結果がそのまま表示されます (値は丸められません)。
数定数 (10 進定数)	指定なし	CQL に記述した値で表現されます。

3.4 データの比較

この節では、クエリ内でのデータの比較について説明します。

3.4.1 比較できるデータ型の組み合わせ

WHERE 句、または HAVING 句に指定した探索条件の中で比較できるデータ型の組み合わせを次の表に示します。WHERE 句については、「4.4.6 WHERE 句」を、HAVING 句については、「4.4.8 HAVING 句」を参照してください。

なお、ここで説明する比較の規則は、GROUP BY 句、および集合関数にも適用されます。

表 3-8 比較できるデータ型の組み合わせ

項番	データ型	数データ	文字データ	日付データ	時刻データ	時刻印データ
1	数データ	○	△	×	×	×
2	文字データ	△	○	△	△	△
3	日付データ	×	△	○	×	×
4	時刻データ	×	△	×	○	×
5	時刻印データ	×	△	×	×	○

(凡例)

○：比較できます。

△：条件によっては比較できます。比較できる場合については、「3.4.2(2) 文字データの比較」を参照してください。

×：比較できません。

3.4.2 データを比較する際の留意点

ここでは、データを比較する際の留意点について説明します。

(1) データ型一般の比較

GROUP BY 句を指定した検索では、列すべての結果の値が同一である行が、同じグループとして扱われます。

(2) 文字データの比較

文字データの比較は、Java の String 型データの比較に準じます。

文字データと日付、時刻、または時刻印データを比較する場合、日付、時刻、または時刻印データの形式に沿った文字列定数との比較だけが有効になります。

WHERE 句の数データと文字データの比較では、文字データが明示的に数データの型にキャストされ、かつ、文字データの書式が比較対象の数データの型に従っている場合にだけ比較できます。

ポイント

文字データのキャストについて

比較に限らず、値式に現れた文字データについても同様にキャストができます。

キャストによって数データと比較できる文字データの書式について次の表に示します。

表 3-9 キャストによって数データと比較できる文字データの書式

数データの型	キャスト指定	文字データの書式	
		規則	例
整数型	(TINYINT) (SMALLINT) (INT[EGER]) (BIGINT)	<ul style="list-style-type: none"> 0～9の数字を指定します。 ピリオドを含んではいけません。 値は数データの範囲内に収まる必要があります。 正の値の場合、プラス (+) の符号を指定できません。 	<p>正常に比較できる例</p> <ul style="list-style-type: none"> '12' '-1' (マイナスの符号を付ける) <p>比較がエラーになる例</p> <ul style="list-style-type: none"> '12.0' (ピリオドを含む) '1b3' (数字以外を含む) '+1' (プラスの符号を付ける)
実数型および10進数	(REAL) (FLOAT) (DOUBLE) (DEC[IMAL])	<ul style="list-style-type: none"> 0～9の数字を指定します。 ピリオドを含められます。 値は数データの範囲内に収まる必要があります。 値にプラス (+) またはマイナス (-) の符号を指定できます。 	<p>正常に比較できる例</p> <ul style="list-style-type: none"> '12' '12.0' (ピリオドを含む) '-1.0' (マイナスの符号を付ける) '+1.0' (プラスの符号を付ける) <p>比較がエラーになる例</p> <ul style="list-style-type: none"> '1b3.0' (数字以外を含む)

注 正負の符号の付加は、java.lang.Number クラスの実装方式に従います。

キャスト指定の形式については、「4.4.18 値式」を参照してください。

(3) 数データの比較

比較するデータのデータ型が異なる場合、範囲の広い方のデータ型に暗黙的にキャストして比較されます。範囲の広さを次に示します。

DECIMAL = NUMERIC > FLOAT = DOUBLE > REAL > BIGINT > INTEGER > SMALLINT > TINYINT

(4) 日付、時刻、および時刻印データの比較

日付、時刻、および時刻印データの比較は、Java の java.sql.Date クラス、java.sql.Time クラス、および java.sql.Timestamp クラスの比較に準じます。

3.5 クエリ定義での注意事項および制限値

この節では、クエリを定義する際の注意事項と、クエリ定義での制限値について説明します。

3.5.1 クエリ定義での注意事項

- クエリ名およびストリーム名は、SDP サーバで一意になるように定義してください。登録済みの名前を定義した場合は、クエリグループの登録時にエラーになります。
- あるクエリ定義ファイルで定義されたストリームまたはクエリについて、別のクエリ定義ファイルで定義されたクエリから参照することはできません。

3.5.2 クエリ定義での制限値

クエリ定義での制限値について次の表に示します。

表 3-10 クエリ定義での制限値

項番	項目	値の範囲	
1	1 入力ファイルに記述できる CQL コマンド数	1~1,024	
2	1CQL コマンドの長さ	1~300,000 文字	
3	1 ストリーム中の列数	1~3,000	
4	ストリームデータ 1 件の長さ (1 行のサイズ)	1~2,147,483,647 バイト	
5	名前の長さ (ストリーム名, リレーション名, クエリ名, 列名, 相関名)	1~100 文字	
6	検索項目数 (SELECT 句の選択式の個数)	1~3,000	
7	列指定リスト	1~255	
8	探索条件内の比較述語および括弧の個数	1~255	
9	1CQL クエリ中に指定できるストリームとリレーションの合計数	1~64	
10	ROWS ウィンドウの生存する行数	1~100,000	
11	時間指定	SECOND	1~2,678,400
12		MILLISECOND	1~86,400,000
13		MINUTE	1~44,640
14		HOUR	1~744
15		DAY	1~31

4

CQL リファレンス

この章では, CQL の文法について説明します。

4.1 CQL 文法説明で使用する記述形式

ここでは、CQL 文法説明で使用する記述形式について説明します。

各 CQL で説明する項目は次のとおりです。ただし、CQL によっては説明しない項目もあります。なお、CQL の文法で使用する記号については、「3.1.3 CQL 文法説明で使用する記号」を参照してください。

形式

記述形式を説明しています。

機能

機能を説明しています。

オペランド

オペランドごとに、指定する項目や、どのような場合に指定するかを説明しています。

引数

引数ごとに、指定する項目や、どのような場合に指定するかを説明しています。

構文規則

構文規則を説明しています。

戻り値

関数の戻り値を説明しています。

注意事項

注意事項を説明しています。

使用例

使用例を説明しています。

4.2 CQL の一覧

CQL には、ストリームおよびクエリを定義するために使用する定義系 CQL と、定義系 CQL の REGISTER QUERY 句や REGISTER QUERY_ATTRIBUTE 句中に指定する操作系 CQL があります。

定義系 CQL の一覧を次の表に示します。

表 4-1 定義系 CQL の一覧

項番	CQL	説明
1	REGISTER STREAM 句	ストリームを定義します。
2	REGISTER QUERY 句	クエリを定義します。
3	REGISTER QUERY_ATTRIBUTE 句	クエリに時刻解像度を指定します。時刻解像度指定の対象となるクエリは、この句のあとに定義する必要があります。

操作系 CQL の一覧を次の表に示します。

表 4-2 操作系 CQL の一覧

項番	種類	説明
1	問い合わせ	リレーションまたは REGISTER STREAM 句で定義したストリームに対して、データの検索を実行します。
2	ストリーム句	出力されるデータをストリームに変換します。
3	リレーション式	一つ以上のリレーションからのデータ検索や検索結果のフィルタリングなどを実行します。
4	SELECT 句	検索した結果を出力する項目（選択式）を指定します。検索結果はリレーションとして取得します。
5	FROM 句	一つ以上のリレーション（リレーション参照）を指定します。FROM 句で取得されるリレーションが、WHERE 句または HAVING 句の対象となります。
6	WHERE 句	FROM 句によって取得されるリレーションに対する探索条件を指定します。
7	GROUP BY 句	直前に指定する句によって取得されるリレーション中の列（グループ化列）を指定します。指定したグループ化列でグルーピングが実行されます。
8	HAVING 句	FROM 句、WHERE 句、または GROUP BY 句によって取得されるリレーションに対して、探索条件を指定します。HAVING 句の探索条件では、HAVING 句中に指定した探索条件によって論理演算を実行し、真の結果だけをリレーションとして取得します。
9	UNION 句	複数の SELECT 句を結合して、一つの CQL 文として実行します。
10	選択リスト	一つまたは複数の選択式を指定します。
11	選択式	検索結果として出力する項目を指定します。
12	列指定リスト	一つまたは複数の列を指定します。

4 CQL リファレンス

項番	種類	説明
13	リレーション参照	検索するリレーションを指定します。リレーション参照は、FROM 句で指定します。
14	ウィンドウ指定	ストリームの生存期間を指定します。ウィンドウ指定は、リレーション参照で指定します。
15	時間指定	時間の単位を指定します。
16	探索条件	指定した条件に従って論理演算を実行し、真の結果だけをリレーションとして取得します。 探索条件は、WHERE 句および HAVING 句で指定します。
17	比較述語	真または偽の論理値を与えるための条件を指定します。
18	値式	値を指定します。
19	定数	定数を指定します。
20	集合関数	複数行から算出される値を求めます。
21	組み込み集合関数	
22	スカラ関数	単一行から算出される値を求めます。
23	組み込みスカラ関数	
24	ストリーム間演算関数	ストリームデータ間の演算を行います。
25	外部定義ストリーム間演算関数	

4.3 定義系 CQL

ここでは定義系 CQL について説明します。

4.3.1 REGISTER STREAM 句 (ストリームの定義)

(1) 形式

```
REGISTER STREAM ::= REGISTER ▲ STREAM ▲ <ストリーム名>
                  ▲ スキーマ指定文字列
                  スキーマ指定文字列 ::= ' (<列名> ▲ <型名> [, <列名> ▲ <型名>] ... ) '
```

(2) 機能

ストリームを定義します。

ストリームは、ストリームの管理テーブルなどを格納する領域 (システムカタログ) に登録され、同時にストリームの受付が開始されます。

(3) オペランド

<ストリーム名>

ストリーム (入力ストリーム) の名称を、任意の名称で、ストリームデータ処理システム内で一意になるように指定します。名称の指定については、「3.2.4 名前の指定」を参照してください。

スキーマ指定文字列

ストリーム名で定義されるストリームに含まれる列名と、その型名を指定します。

<列名>

列名を任意の名称で、ストリームデータ処理システム内で一意になるように指定します。名称の指定については、「3.2.4 名前の指定」を参照してください。

列名に指定できる個数は、1~3,000 個です。

<型名>

型名については、「3.3 CQL のデータ型」を参照してください。

(4) 構文規則

一つの定義で、スキーマ指定文字列をすべて指定してください。例えば、s1 という名称でストリームを定義する場合は、次の例 1 で示すように、必要なスキーマ指定文字列をすべて指定します。

例 1 (正しい指定例)

```
REGISTER STREAM s1(id INT, name VARCHAR(10));
```

次の例 2 で示すように、s1 という名称でストリームを定義する場合に、スキーマ指定文字列を分けて指定すると、同一名称のストリームの二重指定となるためエラーになります。

例 2 (誤った指定例)

```
REGISTER STREAM s1 (id INT);
REGISTER STREAM s1 (name VARCHAR(10));
```

(5) 注意事項

ありません。

(6) 使用例

スキーマとして INT 型の列 id, および 10 個の文字データから構成される列 name を持つストリーム s1 をシステムカタログに登録します。

```
REGISTER STREAM s1(id INT, name VARCHAR(10));
```

4.3.2 REGISTER QUERY 句 (クエリの定義)**(1) 形式**

```
REGISTER QUERY ::= REGISTER ▲ QUERY ▲ <クエリ名> ▲ <問い合わせ>
```

(2) 機能

クエリを定義します。

クエリは、クエリの管理テーブルなどを格納する領域 (クエリリポジトリ) に登録され、同時にクエリ処理が開始されます。

(3) オペランド

<クエリ名>

クエリの名称を任意の名称で、ストリームデータ処理システム内で一意になるように指定します。名称の指定については、「3.2.4 名前の指定」を参照してください。

ここで指定した名称は、問い合わせによって取得されるストリーム (出力ストリーム)、またはリレーションの名称となります。

<問い合わせ>

問い合わせを定義します。定義内容については、「4.4.1 問い合わせ」を参照してください。

(4) 構文規則

一つの定義で、一つのクエリに対する問い合わせをすべて指定してください。例えば、q1 としてクエリを定義する場合は、次の例 1 で示すように、必要な問い合わせをすべて指定します。

例 1 (正しい指定例)

```
REGISTER QUERY q1 SELECT s1.a, s1.b FROM s1[ROWS 10];
```

次の例 2 で示すように、q1 として定義したクエリに対して、問い合わせを分けて指定すると、同一名称のクエリの二重指定となるためエラーになります。

例 2 (誤った指定例)

```
REGISTER QUERY q1 SELECT s1.a FROM s1[ROWS 10];
REGISTER QUERY q1 SELECT s1.b FROM s1[ROWS 10];
```

(5) 注意事項

ありません。

(6) 使用例

ストリーム s1 の列 a のデータを出力するクエリ q1 を定義します。

```
REGISTER QUERY q1 SELECT s1.a FROM s1[ROWS 10];
```

4.3.3 REGISTER QUERY_ATTRIBUTE 句 (時刻解像度の指定)

(1) 形式

```
REGISTER QUERY_ATTRIBUTE ::= REGISTER ▲ QUERY_ATTRIBUTE ▲ <クエリ名>
                             ▲ STREAM_NAME = <データ識別子>
                             ▲ PERIOD = <メッシュ間隔>
                             ▲ TARGETS = <集合関数1> ('<列名1>'),
                             ..., <集合関数N> ('<列名N>');
```

(2) 機能

クエリに時刻解像度を指定します。

(3) オペランド

<クエリ名>

この句のあとに定義する時刻解像度指定の対象となるクエリ名を、ストリームデータ処理システム内で一意になるように指定します。名称の指定については、「3.2.4 名前指定」を参照してください。

クエリ名は、ほかの REGISTER QUERY_ATTRIBUTE 句で指定する名称と重複してはいけません。また、時刻解像度指定の対象となるクエリを定義する REGISTER QUERY 句では、必ず RANGE ウィンドウを指定してください。

<データ識別子>

時刻解像度を指定する列名が定義されている、ストリーム名またはクエリ名を指定します。指定するストリーム名またはクエリ名は、REGISTER QUERY_ATTRIBUTE 句よりも前に定義されている必要があります。

<メッシュ間隔>

メッシュ間隔をミリ秒単位で任意に指定します。必ず、単位として ms を記述してください。

100 ミリ秒未満の値は設定できません。また、時刻解像度を指定するクエリ名の句に指定される RANGE ウィンドウの間隔がメッシュ間隔を超える場合は、メッシュ間隔は RANGE ウィンドウの間隔に設定されます。

例えば、メッシュ間隔が 1,000 ミリ秒で、RANGE ウィンドウの間隔が 500 ミリ秒の場合、メッシュ間隔は 500 ミリ秒に設定されます。また、メッシュ間隔が RANGE ウィンドウより小さく、かつ 86,400,000 ミリ秒より大きい場合は、86,400,000 ミリ秒に設定されます。

なお、100 ミリ秒未満の値を指定した場合は、100 ミリ秒に設定されます。

<集合関数>

集合関数を指定します。ここで指定できる集合関数を次の表に示します。

表 4-3 指定できる集合関数

項番	集合関数	意味
1	SUM	合計値の計算に使用します。
2	MAX	最大値の計算に使用します。
3	MIN	最小値の計算に使用します。

指定できる集合関数の個数は、1～256 個です。

<列名>

STREAM_NAME で指定したデータ識別子が持つ列名を指定します。

列名は、重複してはいけません。

指定できる列名の個数は、1～256 個です。

(4) 構文規則

ありません。

(5) 注意事項

- REGISTER QUERY_ATTRIBUTE 句では、クエリ名、データ識別子、列名がそれぞれ重複してはいけません。
- STREAM_NAME で指定するストリーム名またはクエリ名は、REGISTER QUERY_ATTRIBUTE 句より前に定義しておくか、またはあらかじめ登録しておいてください。
- TARGETS で指定する集合関数の引数に指定する列名は、データ識別子の列名に含まれている必要があります。
- クエリ名、データ識別子、メッシュ間隔、集合関数、および列名に、特殊文字（セミコロンやピリオドなど）を含めてはいけません。
- クエリ名、データ識別子、および列名として、「REGISTER」、「QUERY_ATTRIBUTE」、「STREAM_NAME」、「PERIOD」、「TARGETS」を指定してはいけません。
- 時刻解像度を指定するクエリの定義（REGISTER QUERY 句）は、次の条件を満たす必要があります。
 - FROM 句に複数のストリームを記述しない。
 - WHERE 句を指定しない。
 - GROUP BY 句を指定する。
 - REGISTER QUERY_ATTRIBUTE 句のあとに定義する。
- REGISTER QUERY_ATTRIBUTE 句で TARGETS 以下に指定する列名と STREAM_NAME に指定するデータ識別子の持つ列名は、1 対 1 に対応させておく必要があります。また、時刻解像度を指定するクエリの列名と STREAM_NAME に指定するデータ識別子の持つ列名も、1 対 1 に対応させる必要があります。

クエリ q1 に対して、時刻解像度を指定する場合を例に説明します。下線は、説明で示している個所です。

（正しい指定例）

```
REGISTER STREAM stock(price INTEGER, name VARCHAR(10));
REGISTER QUERY q0
```

```

ISTREAM(
SELECT name, price AS price0, price AS price1
FROM stock[NOW]
);
REGISTER QUERY ATTRIBUTE q1
STREAM NAME=q0 PERIOD=300ms TARGETS=SUM(price0),MAX(price1);
REGISTER QUERY q1
ISTREAM(
SELECT q0.name, SUM(q0.price0) AS sum_price, MAX(q0.price1) AS max_price
FROM q0[RANGE 1 MINUTE]
GROUP BY q0.name
);

```

「REGISTER QUERY q0」で指定した列名 (price0, price1), 「REGISTER QUERY_ATTRIBUTE q1」で指定した列名 (price0, price1), 「REGISTER QUERY q1」で指定した列名 (price0, price1) が, 1 対 1 に対応しています。時刻解像度を指定する場合は, この例で示すように列名を対応させる必要があります。

(誤った指定例)

```

REGISTER STREAM stock(price INTEGER, name VARCHAR(10));
REGISTER QUERY q0
ISTREAM(
SELECT name, price AS price0, price AS price1, price AS price2, price AS price3
FROM stock[NOW]
);
REGISTER QUERY ATTRIBUTE q1
STREAM NAME=q0 PERIOD=300ms TARGETS=SUM(price0),MAX(price1),MIN(price2);
REGISTER QUERY q1
ISTREAM(
SELECT q0.name, SUM(q0.price0) AS sum_price, MAX(q0.price1) AS max_price
FROM q0[RANGE 1 MINUTE]
GROUP BY q0.name
);

```

「REGISTER QUERY q0」で指定した列名 (price0, price1, price2, price3), 「REGISTER QUERY_ATTRIBUTE q1」で指定した列名 (price0, price1, price2), 「REGISTER QUERY q1」で指定した列名 (price0, price1) が, 1 対 1 に対応していないため, エラーとなります。

- 時刻解像度を指定したクエリでは, タプルをメッシュ間隔内で擬似タプルとして集約するため, 集約結果の出力頻度が異なります。例えば, SUM を指定して集計する場合, 集約した擬似タプルごとに結果が生成されるために, 入力タプルごとに時々刻々と生成される結果の出力頻度とは異なります。

(6) 使用例

下線は, この例のポイントとなる箇所です。

(例 1)

RANGE ウィンドウ内にあるタプルで, あらかじめ定義されているストリーム名 stock の, 列名 price の合計値を求めます。

```

REGISTER STREAM stock(price INTEGER, name VARCHAR(10));
REGISTER QUERY ATTRIBUTE q1 STREAM NAME=stock PERIOD=300ms TARGETS=SUM(price);
REGISTER QUERY q1
ISTREAM(
SELECT name, SUM (price) AS s1
FROM stock[RANGE 1 MINUTE]
GROUP BY name
);

```

REGISTER QUERY_ATTRIBUTE 句で指定するクエリには, 時刻解像度を指定するクエリ q1 を指定します。また, REGISTER STREAM 句で定義したストリーム名 stock の列名 price の合計を求めるため, データ識別子に stock を指定し, TARGETS に SUM(price) を指定します。なお, price の合計値ではなく, 最大値や最小値を求める場合には, MAX, MIN を指定します。

(例 2)

RANGE ウィンドウ内にあるタプルで、あらかじめ定義されているストリーム名 stock の、列名 price の合計値、最大値、最小値を求めます。

```
REGISTER STREAM stock(price INTEGER, name VARCHAR(10));
REGISTER QUERY q0
  ISTREAM(
  SELECT name, price AS price0, price AS price1, price AS price2
  FROM stock[NOW]
  );
REGISTER QUERY ATTRIBUTE q1 STREAM NAME=q0 PERIOD=300ms
  TARGETS=SUM(price0), MAX(price1), MIN(price2);
REGISTER QUERY q1
  ISTREAM(
  SELECT q0.name, SUM(q0.price0) AS sum_price,
  MAX(q0.price1) AS max_price, MIN(q0.price2) AS min_price
  FROM q0[RANGE 1 MINUTE]
  GROUP BY q0.name
  );
```

データ識別子 stock、列名 price のクエリについて、複数の集合演算を実行する場合に、TARGETS に同一の列名 price は指定できません。例えば、「TARGETS=SUM(price), MAX(price), MIN(price)」のように指定できません。

この場合、列名 price を合計値用、最大値用、最小値用にそれぞれ定義する必要があります。この例では、CQL 文によって列名 price を置き換えた列名 price0, price1, price2 を持つデータ識別子（ここでは q0）を定義しています。q0 で定義した列名を使用して q1 でそれぞれ演算を実行することで、複数の文の集合関数を実行できます。

4.4 操作系 CQL

ここでは操作系 CQL について説明します。

4.4.1 問い合わせ

(1) 形式

```
問い合わせ ::= { <リレーション式>
                | { <ストリーム句> '(<リレーション式>)'
                | <ストリーム間演算関数> }
```

(2) 機能

リレーションまたは REGISTER STREAM 句で定義したストリームに対して、データの検索を実行します。

(3) オペランド

<リレーション式>

リレーション式の指定については、「4.4.3 リレーション式」を参照してください。

ストリーム句を指定しないでリレーション式を指定すると、問い合わせ結果はリレーションで取得されます。

<ストリーム句>

ストリーム句を指定する場合、ストリーム句に続けてリレーション式を括弧 (()) で囲んで記述します。ストリーム句の指定については、「4.4.2 ストリーム句」を参照してください。

ストリーム句を指定すると、問い合わせ結果はストリームで取得されます。

<ストリーム間演算関数>

ストリーム間演算関数の指定については、「4.4.24 ストリーム間演算関数」を参照してください。

ストリーム間演算関数を指定すると、問い合わせ結果はストリームで取得されます。

(4) 構文規則

ありません。

(5) 注意事項

ありません。

(6) 使用例

リレーション式を指定し、ストリーム s1 の列 a のデータをリレーションとして生成します。生成されたリレーションはリレーション q1 として出力されます。下線部が問い合わせの部分です。

```
REGISTER QUERY q1 SELECT s1.a FROM s1[ROWS 10];
```

4.4.2 ストリーム句

(1) 形式

```
ストリーム句 ::= { ISTREAM | DSTREAM
                  | RSTREAM [ '[' <整数定数> [ ▲ <時間指定> ] ' ]' }
```

(2) 機能

出力されるデータをストリームに変換します。

(3) オペランド

ISTREAM

出力レーション内の増加分だけを出力します。

DSTREAM

出力レーション内の減少分だけを出力します。

RSTREAM

出力レーション内の集合すべてを一定間隔で出力します。

<整数定数>

時間の値そのものを指定します。整数定数の値の範囲については、「4.4.15 時間指定」を参照してください。

指定を省略した場合、指定値には 1 秒を仮定します。

<時間指定>

整数定数で指定した時間の単位を指定します。時間指定の指定については、「4.4.15 時間指定」を参照してください。

指定を省略した場合、整数定数で指定した値を秒単位として認識します。

(4) 構文規則

問い合わせに定義するストリーム句の詳細を記述します。ストリーム句のあとにレーション式を括弧 (()) で囲んで指定します。

(5) 注意事項

ありません。

(6) 使用例

レーション s1 の全データの増加時刻をタイムスタンプに持つストリームを生成します。生成されたストリームは、ストリーム q1 として出力されます。下線部がストリーム句の部分です。

```
REGISTER QUERY q1 ISTREAM (SELECT * FROM s1[ROWS 100]);
```

4.4.3 リレーション式

(1) 形式

リレーション式 ::= <SELECT句> ▲ <FROM句> [▲ <WHERE句>]
 [▲ <GROUP BY句> [▲ <HAVING句>]] [▲ <UNION句>]

(2) 機能

一つ以上のリレーションからのデータ検索や検索結果のフィルタリングなどを実行します。

(3) オペランド

< SELECT 句 >

SELECT 句の指定については、「4.4.4 SELECT 句」を参照してください。

< FROM 句 >

FROM 句の指定については、「4.4.5 FROM 句」を参照してください。

< WHERE 句 >

WHERE 句の指定については、「4.4.6 WHERE 句」を参照してください。

< GROUP BY 句 >

GROUP BY 句の指定については、「4.4.7 GROUP BY 句」を参照してください。

< HAVING 句 >

HAVING 句の指定については、「4.4.8 HAVING 句」を参照してください。

< UNION 句 >

UNION 句の指定については、「4.4.9 UNION 句」を参照してください。

(4) 構文規則

問い合わせのリレーション式の詳細を記述します。SELECT 句、FROM 句以降に検索方法やフィルタリング方法を指定します。

(5) 注意事項

ありません。

(6) 使用例

リレーション s1 の列 b の値が 20 より小さい列 a のデータを出力します。下線部がリレーション式の部分です。

```
REGISTER QUERY q1 SELECT s1.a FROM s1[ROWS 100] WHERE s1.b < 20;
```

4.4.4 SELECT 句

(1) 形式

SELECT 句 ::= SELECT ▲ { <選択リスト> | * }

(2) 機能

リレーションに対して検索した結果を出力する項目（選択式）を指定し、選択式をリレーションとして取得します。

(3) オペランド

<選択リスト>

選択リストの指定については、「4.4.10 選択リスト」を参照してください。

*

リレーションのすべての列を出力します。

リレーション式の FROM 句で指定したリレーションの順序に従い、すべての列を指定することを意味します。各リレーション中の列の順序は、ストリームを定義したときのスキーマ指定文字列、またはクエリを定義したときの選択リストに指定した順序です。

(4) 構文規則

ありません。

(5) 注意事項

ありません。

(6) 使用例

リレーション s1 の全データを出力します。下線部が SELECT 句の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[ROWS 10];
```

4.4.5 FROM 句

(1) 形式

FROM句 ::= FROM ▲ <リレーション参照> [, <リレーション参照>] …

(2) 機能

一つ以上のリレーション（リレーション参照）を指定します。FROM 句で取得されるリレーションが、WHERE 句または HAVING 句の対象となります。

WHERE 句および HAVING 句を指定しない場合は、FROM 句で取得されるリレーションが SELECT 句の対象となります。

複数のリレーションを指定した場合、指定したリレーションの順序で各リレーションから 1 行ずつ組み合わせて連結された行が、FROM 句で取得されるリレーションの行になります。行数は、各リレーションの行数の積になります。

(3) オペランド

<リレーション参照>

リレーション参照の指定については、「4.4.13 リレーション参照」を参照してください。

指定できるリレーション参照の個数は、1～64 個です。

(4) 構文規則

ありません。

(5) 注意事項

FROM 句で取得されるリレーションは、グループ化列を持たない一つのグループになります。

(6) 使用例

取得するリレーション s1 を指定します。下線部が FROM 句の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[ROWS 100];
```

4.4.6 WHERE 句

(1) 形式

WHERE 句 ::= WHERE ▲ <探索条件>

(2) 機能

FROM 句によって取得されるリレーションに対する探索条件を指定します。

WHERE 句を指定し、かつ HAVING 句を指定しない場合は、WHERE 句によって取得されるリレーションが SELECT 句の対象となります。

WHERE 句を省略すると、FROM 句で取得されるリレーションに含まれるすべての行が SELECT 句の対象となります。

(3) オペランド

<探索条件>

探索条件の指定については、「4.4.16 探索条件」を参照してください。

先行する FROM 句のリレーション参照が一つの場合、探索条件には列名だけを指定します。FROM 句のリレーション参照が複数の場合、探索条件に指定する各列名は、データ識別子と列名を指定して、一意にしなければいけません。

(4) 構文規則

先行する FROM 句で取得したリレーションを t とした場合、探索条件は t のそれぞれの行に適用されます。WHERE 句で取得されるリレーションは、探索条件の結果が真となる、t の行の集合で構成されるリレーションです。

(5) 注意事項

WHERE 句で取得されるリレーションは、グループ化列を持たない一つのグループになります。

(6) 使用例

リレーション s1 の列 b の値が 10 より大きい列 a, b のデータを出力します。下線部が WHERE 句の部分です。

```
REGISTER QUERY q1 SELECT s1.a, s1.b FROM s1[ROWS 100] WHERE s1.b > 10;
```

4.4.7 GROUP BY 句

(1) 形式

GROUP BY句 ::= GROUP ▲ BY ▲ <列指定リスト>

(2) 機能

直前に指定する句によって取得されるリレーション中の列 (グループ化列) を指定します。指定したグループ化列でグルーピングが実行されます。

グルーピングでは、GROUP BY 句で指定した列の値が同一の行をグルーピングして、グループ単位に 1 行にまとめて出力します。

(3) オペランド

<列指定リスト>

列指定リストの指定については、「4.4.12 列指定リスト」を参照してください。

列指定リストに指定する列指定は、同じ列指定を重複して指定できません。また、先行する FROM 句のリレーション参照が一つの場合、列指定リストには列名だけを指定します。FROM 句のリレーション参照が複数の場合、列指定リストに指定する各列名は、データ識別子と列名を指定して、一意にしなければいけません。

(4) 構文規則

ありません。

(5) 注意事項

リレーション式に GROUP BY 句を指定した場合、SELECT 句中の選択式には GROUP BY 句で指定された要素 (グループ化列の名称) 以外の列指定または列名を一次子とする値式は指定できません。

(6) 使用例

リレーション s1 の列 b, c をグループ化し、列 a, b, c のデータを出力します。下線部が GROUP BY 句の部分です。

```
REGISTER QUERY q1 SELECT s1.b, s1.c
FROM s1[ROWS 100] GROUP BY s1.b, s1.c;
```

4.4.8 HAVING 句

(1) 形式

HAVING句 ::= HAVING ▲ <探索条件>

(2) 機能

FROM 句, WHERE 句, または GROUP BY 句によって取得されるリレーションに対して、探索条件を指定します。HAVING 句の探索条件では、HAVING 句中に指定した探索条件によって論理演算を実行し、真の結果だけをリレーションとして取得します。

HAVING 句を指定する場合は、HAVING 句によって取得されるリレーションが SELECT 句の対象となります。

(3) オペランド

<探索条件>

探索条件の指定については、「4.4.16 探索条件」を参照してください。

探索条件中に列指定を指定する場合は、グループ化列を指定するか、または集合関数の引数として指定してください。グループ化列については、「4.4.7 GROUP BY 句」、集合関数については「4.4.20 集合関数」を参照してください。

(4) 構文規則

HAVING 句中に指定した探索条件の結果が真となるグループが選択されます。

(5) 注意事項

HAVING 句を省略すると、先行する GROUP BY 句、または FROM 句の結果の全グループが選択されます。

(6) 使用例

リレーション s1 の列 b, c をグループ化し、列 a の平均値が 10 より大きい列 a, b, c のデータを出力します。下線部が HAVING 句の部分です。

```
REGISTER QUERY q1 SELECT AVG(s1.a) AS a1,s1.b,s1.c FROM s1[ROWS 100]
GROUP BY s1.b,s1.c HAVING AVG(s1.a) > 10;
```

4.4.9 UNION 句

(1) 形式

UNION句 ::= UNION ▲ [ALL ▲] <リレーション式>

(2) 機能

複数の SELECT 句を結合して、一つの CQL 文として実行します。

(3) オペランド

ALL

指定した場合、行の重複を許可します。

ALL を指定しないで UNION を指定した場合、行を重複排除します。

重複排除とは、検索結果に重複した行（選択式で指定した項目で一つ以上の行を構成し、その行が同一のもの）が存在する場合、重複した行を排除し、一つの行と見なして出力することをいいます。

<リレーション式>

リレーション式の指定については、「4.4.3 リレーション式」を参照してください。

(4) 構文規則

次の二つの条件を満たすクエリは、構文エラーになります。

- UNION ALL オペランドを使用している
- 複数の SELECT 句で、FROM 句に同一ストリームまたはリレーションを一つだけ指定している

構文エラーとなる例を次に示します。下線部がエラーになる箇所です。

```
REGISTER QUERY q1 ISTREAM(
  SELECT * FROM s1[RANGE 3 SECOND]
  UNION ALL SELECT * FROM s1[RANGE 5 SECOND]);
```

(5) 注意事項

- UNION 句で結合する場合、すべての SELECT 句の選択リストは、数、型、および文字データの文字数をすべて同じにしてください。
- UNION 句を持つリレーション式で取得されるリレーションの場合、1 番目に指定したリレーション式 (UNION 句を除く) の結果として取得されるリレーションの列名が、問い合わせ後の結果として取得されるリレーションの列名になります。

詳細については、「3.2.5(2) 列指定」を参照してください。

- FROM 句で指定するストリームデータが同一の場合、入力したすべての列をそのまま出力する SELECT 句を UNION ALL で結合することはできません。

誤った例を次に示します。下線部が該当する SELECT 句です。

```
SELECT * FROM s1[RANGE 3 SECOND] UNION ALL SELECT * FROM s1[RANGE 5 SECOND];
```

どちらの SELECT 句も同一のストリーム s1 を一つだけ入力し、オペランドに*を指定しているので、入力したすべての列をそのまま出力し、エラーとなります。

(6) 使用例

リレーション s1 とリレーション s2 の全データを出力します。下線部が UNION 句の部分です。

```
REGISTER QUERY s1 SELECT * FROM s1[ROWS 100] UNION SELECT * FROM s2[ROWS 100];
```

4.4.10 選択リスト

(1) 形式

選択リスト ::= <選択式> [, <選択リスト>]

(2) 機能

一つまたは複数の選択式を指定します。

(3) オペランド

<選択式>

選択式の指定については、「4.4.11 選択式」を参照してください。

指定できる選択式の個数は、1~3,000 個です。

(4) 構文規則

SELECT 句のあとに指定します。

(5) 注意事項

選択リストに指定できるのは、FROM 句によって取得されたリレーシンのデータだけです。

(6) 使用例

リレーション s1 の列 a および b のデータを出力します。下線部が選択リストの部分です。

```
REGISTER QUERY q1 SELECT s1.a, s1.b FROM s1[ROWS 100];
```

4.4.11 選択式

(1) 形式

選択式 ::= { {<列指定> | <列名>} [▲AS▲<相関名>]
| {<値式> | <集合関数>} ▲AS▲<相関名> }

(2) 機能

検索結果として出力する項目を指定します。

(3) オペランド

<列指定>

列指定については、「3.2.5(2) 列指定」を参照してください。

<列名>

後続の FROM 句のリレーション参照が一つである場合、列名を指定できます。

後続の FROM 句のリレーション参照が複数ある場合は、列指定を指定してください。

<相関名>

次のような場合に区別できる名称を指定します。

- 列名が重複する場合
- 列指定の名称が長く複雑である場合
- 値式が列名と列指定のどちらでもない場合

名称は名前の規則に従います。名前の規則の詳細については、「3.2.4 名前の指定」を参照してください。

相関名には、一つの SELECT 句中に指定したほかの相関名と同じ名称は指定できません。また、ほかの範囲変数の表識別子と同じ名称も指定できません。範囲変数と表識別子については、「3.2.5 名前の修飾」を参照してください。

<値式>

値式については、「4.4.18 値式」を参照してください。

i 番目の選択式に対して列名を指定した場合、リレーション式によって取得されるリレーションの i 番目の列名は、指定した列名となります。i 番目の選択式に対して列名を指定しない場合は、次のようになります。

- 列指定から取得された値式のと看、その列指定で指定した列名が、リレーション式によって取得されるリレーションの i 番目の列名になります。
- 列指定以外から取得された値式のと看、相関名の指定が必要になります。

<集合関数>

集合関数の指定については、「4.4.20 集合関数」を参照してください。

集合関数の引数中には、列指定または列名を含める必要があります。

(4) 構文規則

- 選択式に指定した相関名は取得されるリレーションだけで利用できます。選択式の後続の句では利用できません。

例えば、次のように、q1 から取得された相関名 a1 および a2 は、q2 で利用できます。

(正しい指定例)

```
REGISTER QUERY q1 SELECT s1.c1 AS a1, s2.c1 AS a2 FROM s1[NOW], s2[NOW];
REGISTER QUERY q2 SELECT a1, a2 FROM q1 WHERE a1 > 1 GROUP BY a1, a2
HAVING a2 > 1;
```

次のように、選択式に指定した相関名 a1 および a2 は、後続の句で利用できません。

(誤った指定例)

```
REGISTER QUERY q1 SELECT s1.c1 AS a1, s2.c1 AS a2 FROM s1[NOW], s2[NOW]
WHERE a1 > 1 GROUP BY a1, a2 HAVING a2 > 1;
```

- 選択式に相関名を指定する場合は、ほかの選択式で使用した列名と異なる名称を指定します。

(正しい指定例)

```
REGISTER QUERY q1 SELECT s1.c1, s1.c2+1 AS a1 ~ ;
```

次の例では、ほかの選択式に使用している c1 を使用しているため、エラーになります。

(誤った指定例)

```
REGISTER QUERY q1 SELECT s1.c1, s1.c2+1 AS c1 ~ ;
```

(5) 注意事項

選択式に指定できるのは、定義済みのストリームのデータだけです。

(6) 使用例

リレーション s1 の列 a および b のデータを出力します。下線部が選択式の部分です。

```
REGISTER QUERY q1 SELECT s1.a, s1.b FROM s1[ROWS 100];
```

4.4.12 列指定リスト

(1) 形式

列指定リスト ::= <列指定> [, <列指定リスト>]

(2) 機能

一つ、または複数の列を指定します。

(3) オペランド

<列指定>

列指定については、「3.2.5(2) 列指定」を参照してください。

指定できる列指定の個数は、1～255 個です。

(4) 構文規則

列指定リストの構文規則については、「3.2.5(2) 列指定」を参照してください。

(5) 注意事項

列指定リストの注意事項については、「3.2.5(2) 列指定」を参照してください。

(6) 使用例

リレーション s1 の列 b および c をグループ化し、列 a, b, および c のデータを出力します。下線部が列指定リストの部分です。

```
REGISTER QUERY q1 SELECT SUM(s1.a) AS a1, s1.b, s1.c FROM s1[ROWS 100]
GROUP BY s1.b, s1.c;
```

4.4.13 リレーション参照

(1) 形式

リレーション参照 ::= {<リレーション名>
| <ストリーム名>['<ウィンドウ指定>']}
[▲AS▲<相関名>]

(2) 機能

検索するリレーションを指定します。リレーション参照は、FROM 句で指定します。

(3) オペランド

<リレーション名>

検索するリレーションの名称を指定します。

リレーション名に指定できるのは、問い合わせ結果をストリーム句でストリーム化しないで取得した場合のクエリ名です。

<ストリーム名>

検索するストリームの名称を指定します。

ストリーム名に指定できるのは、問い合わせ結果をストリーム句でストリーム化して取得した場合のクエリ名です。

<ウィンドウ指定>

ウィンドウ指定については、「4.4.14 ウィンドウ指定」を参照してください。

<相関名>

リレーション参照の名称が長く複雑である場合、別名として指定します。名称は名前の規則に従います。名前の規則の詳細については、「3.2.4 名前の指定」を参照してください。

相関名には、一つの SELECT 句中に指定したほかの相関名と同じ名称は指定できません。また、ほかの表識別子と同じ名称も指定できません。表識別子については、「3.2.5 名前の修飾」を参照してください。

(4) 構文規則

- リレーション参照は、FROM 句で指定します。
- リレーション参照に指定した相関名は一つのクエリ文の中でだけ有効です。複数のクエリ文にわたる利用はできません。

例えば、次のように、リレーション参照に指定した相関名 a1 および a2 は、一つのクエリ文の中で利用できます。

(正しい指定例)

```
REGISTER QUERY q1 SELECT a1.c1, a2.c1 FROM s1[NOW] AS a1, s2[NOW] AS a2
WHERE a1.c1 > 1 GROUP BY a1.c1, a2.c1
HAVING a2.c1 > 1;
```

次の例では、q2 でリレーション参照に指定した q1 から相関名 a1 および a2 は取得されないため、利用できません。

(誤った指定例)

```
REGISTER QUERY q1 SELECT a1.c1, a2.c1 FROM s1[NOW] AS a1, s2[NOW] AS a2;
REGISTER QUERY q2 SELECT a1.c1, a2.c1 FROM q1 WHERE a1.c1 > 1
GROUP BY a1.c1, a2.c1 HAVING a2.c1 > 1;
```

- リレーション参照に指定する相関名には、ストリーム名およびクエリ名と異なる名称を指定します。

次の例では、相関名に指定した s2 はストリーム名として定義されているので指定できません。

(誤った指定例)

```
REGISTER STREAM s1 (id INT, name VARCHAR(10));
REGISTER STREAM s2 (id INT, name VARCHAR(10));
REGISTER QUERY q1 SELECT * FROM s1[NOW] AS s2;
```

次の例では、相関名に指定した q1 はクエリ名として定義されているので指定できません。

(誤った指定例)

```
REGISTER QUERY q1 SELECT * FROM s1[NOW];
REGISTER QUERY q2 SELECT * FROM s2[NOW] AS q1;
```

- リレーション参照に同一名称のストリームを指定する場合は、次のように相関名を指定して一意に識別できるようにします。

(正しい指定例)

```
REGISTER QUERY q1 ISTREAM(
SELECT ~ FROM s1[RANGE 5] AS old, s1[NOW] AS new ~);
```

次の例では、ストリームが一意に識別できないため、エラーになります。

(誤った指定例)

```
REGISTER QUERY q1 ISTREAM(
SELECT ~ FROM s1[RANGE 5], s1[NOW] ~);
```

(5) 注意事項

ありません。

(6) 使用例

リレーション s1 の列 a および b のデータを出力します。下線部がリレーション参照の部分です。

```
REGISTER QUERY q1 SELECT s1.a, s1.b FROM s1[ROWS 100];
```

4.4.14 ウィンドウ指定

(1) 形式

```
ウィンドウ指定 ::= {ROWS ▲ <整数定数>
                    RANGE ▲ <整数定数> [▲ <時間指定>]
                    NOW
                    PARTITION ▲ BY ▲ <列指定リスト> ▲ ROWS ▲ <整数定数>}
```

(2) 機能

ストリームの生存期間を指定します。ウィンドウ指定は、リレーション参照で指定します。

(3) オペランド

ROWS

生存する行数として、1～100,000 の値を指定します。

RANGE

生存する時間を指定します。時刻解像度を指定するクエリでは、必ず RANGE を指定してください。

<整数定数>

時間の値を指定します。整数定数の値の範囲については、「4.4.15 時間指定」を参照してください。

<時間指定>

整数定数で指定した時間の単位を指定します。時間指定の指定については、「4.4.15 時間指定」を参照してください。

指定を省略した場合、整数定数で指定した値は秒単位の値として認識されます。

NOW

生存する時間を現時刻だけにします。

PARTITION BY

列指定リストで指定した列名の値ごとに ROWS の処理をします。

<列指定リスト>

列指定リストの指定については、「4.4.12 列指定リスト」を参照してください。

列指定リストには、列名の指定ができます。

(4) 構文規則

ありません。

(5) 注意事項

- ROWS に生存する行数の値の範囲を超えた値を指定した場合、その最大値が仮定され、処理が継続されます。
- ROWS を指定した場合に、生存する行数の値が 0 のときは、その最小値が仮定され、処理が継続されます。負または整数定数以外の場合、エラーになります。

(6) 使用例

使用例 1

リレーション s1 のストリームの生存期間を 2 秒間に指定します。下線部がウィンドウ指定の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[RANGE 2 SECOND];
```

使用例 2

リレーション s1 のストリームを列 a の値ごとに生存する行を 2 に指定します。下線部がウィンドウ指定の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[PARTITION BY s1.a ROWS 2];
```

4.4.15 時間指定

(1) 形式

時間指定 ::= {SECOND | MILLISECOND | MINUTE | HOUR | DAY}

(2) 機能

時間の単位を指定します。

(3) オペランド

SECOND

時間の単位として、秒を指定します。

MILLISECOND

時間の単位として、ミリ秒を指定します。

MINUTE

時間の単位として、分を指定します。

HOUR

時間の単位として、時間を指定します。

DAY

時間の単位として、日を指定します。

(4) 構文規則

時間の値そのものは、時間指定とは別に、整数定数で指定します。時間指定と整数定数の関係を次の表に示します。

表 4-4 時間指定と整数定数の関係

項番	時間指定 (時間の単位を指定)		整数定数 (時間の値を指定)	
	オペランド	意味	最小値	最大値
1	SECOND	秒	1	2678400
2	MILLISECOND	ミリ秒	1	86400000
3	MINUTE	分	1	44640
4	HOUR	時	1	744
5	DAY	日	1	31

(5) 注意事項

- 時間指定の値の範囲を超えた値を整数定数に指定した場合、その最大値を仮定し処理を継続します。
- 時間指定の値として、整数定数に 0 を指定した場合、最小値である 1 を仮定し処理を継続します。負または整数定数以外の値を指定した場合、エラーになります。

(6) 使用例

リレーション s1 のストリームの生存期間を 2 秒間に指定します。下線部が時間指定の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[RANGE 2 SECOND];
```

4.4.16 探索条件

(1) 形式

HAVING 句の場合

探索条件 ::= 比較述語 [▲AND▲<探索条件>]

WHERE 句の場合

探索条件 ::= { { ('<探索条件>') | <比較述語> }
 | NOT { ▲₀' (<探索条件>') | ▲<比較述語> }
 | <探索条件> ▲OR▲ { ('<探索条件>') | <比較述語> }
 | <探索条件> ▲AND▲ { ('<探索条件>') | <比較述語> } }

(2) 機能

指定した条件に従って論理演算を実行し、真の結果だけをリレーションとして取得します。

探索条件は、WHERE 句および HAVING 句で指定します。

(3) オペランド

<比較述語>

比較述語の指定については、「4.4.17 比較述語」を参照してください。

(4) 構文規則

探索条件は、WHERE 句または HAVING 句に指定します。句によって指定できる論理演算が異なります。指定できる論理演算を次の表に示します。

表 4-5 指定できる論理演算

項番	論理演算	WHERE 句	HAVING 句
1	AND	指定できます。	指定できます。
2	NOT	指定できます。	指定できません。
3	OR	指定できます。	指定できません。

(5) 注意事項

- WHERE 句の指定には、次の制限があります。
 - 論理演算子 OR は、二つの項の比較述語がすべて同一リレーションであるときだけ指定できます。
 - 論理演算子 NOT は、その右単項の比較述語がすべて同一リレーションであるときだけ利用できます。

なお、複数リレーションは指定できません。s1, s2 がスキーマ(c1 INT, c2 INT, c3 INT)であるリレーションのときの例を次に示します。

(正しい指定例：同一リレーションの OR を指定)

```
s1.c1 < 1 OR s1.c2 < 1
```

(誤った指定例：複数リレーションの OR を指定)

```
s1.c1 < 1 OR s2.c1 < 1
```

(正しい指定例：同一リレーションの NOT を指定)

```
NOT(s1.c1 < 1) AND NOT(s2.c1 < 1)
```

(誤った指定例：複数リレーションの NOT を指定)

```
NOT((s1.c1 < 1) AND (s2.c1 < 1))
```

(正しい指定例：同一リレーションの OR と NOT を指定)

```
(s1.c1 < 1 OR s1.c2 < 1) AND NOT(s2.c1 < 1)
```

(誤った指定例：複数リレーションの OR と NOT を指定)

```
s1.c1 < 1 OR s1.c2 < 1 AND NOT(s2.c1 < 1)...
```

- 論理演算の評価順序は、括弧内、NOT、AND、OR の順です。例えば、次の二つの演算は同じ意味になります。

```
s1.c1 < 1 OR (s1.c2 < 1 AND NOT (s1.c3 < 3))
s1.c1 < 1 OR (s1.c2 < 1 AND s1.c3 >= 3)
```

- 探索条件の括弧 (()) は、省略できます。

- 探索条件内に指定できる比較述語および括弧の個数は、1~255 個です。個数の数え方の例を次に示します。

```
NOT c1>1
```

: 比較述語が c1>1 の一つだけなので、個数は 1 になります。

```
NOT(c1>1)
```

: 比較述語が c1>1 の一つ、括弧が一組なので、個数は 2 になります。

```
c1>1 AND c2>1
```

: 比較述語が c1>1, c2>1 の二つなので個数は 2 になります。

```
(c1>1) AND (c2>1)
```

: 比較述語が c1>1, c2>1 の二つ、括弧が二組なので、個数は 4 になります。

(6) 使用例

リレーション s1 の列 a の値が 5 より小さく、1 より大きいものを出力します。下線部が探索条件の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[ROWS 100] WHERE s1.a < 5 AND s1.a > 1;
```

4.4.17 比較述語

(1) 形式

WHERE 句の場合

比較述語 ::= <値式> 比較演算子 <値式>
比較演算子 ::= {<|<=|>|=|!=}

HAVING 句の場合

比較述語 ::= 比較演算項 比較演算子 比較演算項
比較演算項 ::= {<列指定> | <列名> | <スカラ関数> | <集合関数> | <定数>}
比較演算子 ::= {<|<=|>|=|!=}

(2) 機能

真、または偽の論理値を与えるための条件を指定します。

(3) オペランド

<値式>

値式の指定については、「4.4.18 値式」を参照してください。

比較演算子

比較演算項を比較するための比較演算子として、「<」、「<=」、「>」、「>=」、「=」、または「!=」を指定します。

比較演算項

比較対象の項として、列指定、スカラ関数、集合関数、または定数を指定します。

なお、先行する FROM 句のリレーション参照が一つであれば、その比較演算項では列名の指定ができません。

<列指定>

列指定については、「3.2.5(2) 列指定」を参照してください。

<列名>

後続の FROM 句のリレーション参照が一つである場合、列名を指定できます。

後続の FROM 句のリレーション参照が複数ある場合は、列指定を指定してください。

<スカラ関数>

スカラ関数の指定については、「4.4.22 スカラ関数」を参照してください。

<集合関数>

集合関数の指定については、「4.4.20 集合関数」を参照してください。

<定数>

定数については、「4.4.19 定数」を参照してください。

(4) 構文規則

比較述語を「比較演算項 X 比較演算子 比較演算項 Y」として、比較演算子の意味を次の表に示します。

表 4-6 比較演算子の種類と機能

項番	比較演算子の指定	意味
1	比較演算項 X=比較演算項 Y	比較演算項 X と比較演算項 Y が等しいことを示します。
2	比較演算項 X!=比較演算項 Y	比較演算項 X と比較演算項 Y が等しくないことを示します。
3	比較演算項 X<比較演算項 Y	比較演算項 X が比較演算項 Y より小さいことを示します。
4	比較演算項 X<=比較演算項 Y	比較演算項 X が比較演算項 Y 以下であることを示します。
5	比較演算項 X>比較演算項 Y	比較演算項 X が比較演算項 Y より大きいことを示します。
6	比較演算項 X>=比較演算項 Y	比較演算項 X が比較演算項 Y 以上であることを示します。

- 左右の比較演算項のデータは、比較できるデータ型にしてください。比較できるデータ型については、「3.4 データの比較」を参照してください。
- 数データ同士の比較で、比較するデータの型が異なる場合、範囲の広い方の型で比較します。範囲の広さを次に示します。

DECIMAL = NUMERIC > FLOAT = DOUBLE > REAL > BIGINT > INTEGER > SMALLINT > TINYINT

(5) 注意事項

- 定数と定数の比較はできません。比較した場合、エラーになります。
- 比較演算項または値式のどちらかに、列指定または列名を含める必要があります。
- 比較述語に指定する値式では、異なるデータ識別子の四則演算はできません。例えば、次のような指定はできません。

```
WHERE s1.a + s2.a < 5
```

(6) 使用例

リレーション s1 の列 a の値が 5 より小さいものを出力します。下線部が比較述語の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[RANGE 10 SECOND] WHERE s1.a < 5;
```

4.4.18 値式

(1) 形式

値式 ::= [符号] 項 [演算子 [符号] 項] ...

項 ::= [キャスト指定] 値式一次子

演算子 ::= {+ | - | * | /}

符号 ::= {+ | -}

キャスト指定 ::= {'(' TINYINT ')', (' SMALLINT ')', (' INT [EGER] ')',
'(' BIGINT ')', (' REAL ')', (' FLOAT ')', (' DOUBLE ')',
'(' DEC [IMAL] ')'}
'(' <値式> ')'

値式一次子 ::= {<列指定> | <列名> | <定数> | <スカラ関数> | ('(<値式>'))}

(2) 機能

値を指定します。

(3) オペランド

符号

「+」または「-」を指定します。

項

キャスト指定, および値式一次子を指定します。

キャスト指定

キャスト指定に指定するデータ型については、「3.3 CQL のデータ型」を参照してください。

値式一次子

列指定, 列名, 定数, スカラ関数, または値式を指定します。

<列指定>

列指定については、「3.2.5(2) 列指定」を参照してください。

<列名>

後続の FROM 句のリレーション参照が一つである場合, 列名を指定できます。

後続の FROM 句のリレーション参照が複数ある場合は, 列指定を指定してください。

<定数>

定数については、「4.4.19 定数」を参照してください。

<スカラ関数>

スカラ関数の指定については、「4.4.22 スカラ関数」を参照してください。

演算子

「+」, 「-」, 「*」, または「/」を指定します。演算子の後ろに符号を指定した項を指定する場合, その項は括弧で囲む必要があります。符号と演算子の例を次に示します。

```
-a+(-b)
a*(-b+1)
a/(-b)+1
```

(4) 構文規則

- 値式中に四則演算子を指定することで, 四則演算ができます。
- 値式の結果のデータ型は, 四則演算または値式一次子の結果と同じデータ型になります。また, データ構造も四則演算または値式一次子の結果と同じになります。
- 四則演算 (2 項演算) は数データで行います。演算中に数データ以外のデータ型 (文字データや時刻データなど) を含めることはできません。含めた場合はエラーとなります。
数データでは, 異なるデータ型間で演算できます。データ型と演算結果のデータ型の関係を次の表に示します。

表 4-7 四則演算（2 項演算）の演算項のデータ型と演算結果のデータ型の関係

項番	第 1 演算項データ型	第 2 演算項データ型								
		TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
1	TINYINT	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
2	SMALLINT	SMALLINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
3	INTEGER	INTEGER	INTEGER	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
4	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
5	REAL	REAL	REAL	REAL	REAL	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
6	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	DECIMAL	NUMERIC
7	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DECIMAL	NUMERIC
8	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL
9	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC

- DECIMAL 型または NUMERIC 型を含む演算の場合、DECIMAL 型または NUMERIC 型以外のデータ型の値は、次のパラメーターの値に従って丸められ、DECIMAL 型（NUMERIC 型）の値に変換されて演算されます。
 - system_config.properties の query.decimalMaxPrecision
精度を指定するパラメーターです。
 - system_config.properties の query.decimalRoundingMode
丸め方を指定するパラメーターです。
- また、演算結果も同様に丸められます。

(5) 注意事項

- 演算途中でオーバーフローが発生した場合、処理は継続されます。このとき、整数であればオーバーフローしたままの値（入りきれない上位バイトをカットし、下位バイトをそのまま使用する）となり、浮動小数点値であれば無限大になります。ただし、DECIMAL 型または NUMERIC 型の値と、浮動小数点値との演算で、浮動小数点値でオーバーフローが発生した場合は、エラー（クエリグループが閉塞）になります。
- 除算で第 2 演算項の値に 0 を指定した場合は、エラーになります。
- 文字データおよび文字列定数に対する演算は、エラーになります。
- 日付/時刻データ（文字列定数を含む）に対するキャストは、エラーになります。

- 文字データの場合、文字データの書式が数データに合うものだけがキャストできます。文字データの書式については、「3.4 データの比較」を参照してください。
- DECIMAL 型にキャストした場合、次のパラメーターの値に従って丸められ、DECIMAL 型に変換されます。
 - system_config.properties の query.decimalMaxPrecision
精度を指定するパラメーターです。
 - system_config.properties の query.decimalRoundingMode
丸め方を指定するパラメーターです。

(6) 使用例

リレーション s1 の列 a を DECIMAL 型にキャストした値と、列 b を INT 型にキャストした値を出力します。下線部が値式の部分です。

```
REGISTER QUERY q1 SELECT (DECIMAL)s1.a AS xxx, (INT)s1.b AS yyy FROM s1[ROWS 100];
```

4.4.19 定数

(1) 形式

```
定数 ::= {文字列定数 | 数定数}
文字列定数 ::= {<日付データ> | <時刻データ> |
               <時刻印データ> | <文字列>}
数定数 ::= {<整数定数> | <浮動小数点定数> | <10進定数>}
```

(2) 機能

定数を指定します。一つ以上の値を指定できます。

(3) オペランド

文字列定数

日付データ、時刻データ、時刻印データ、または文字列を指定します。データ型は VARCHAR 型です。

<日付データ>

日付データの指定については、「3.2.6(2) 日付データの規定の文字列表現」を参照してください。

<時刻データ>

時刻データの指定については、「3.2.6(3) 時刻データの規定の文字列表現」を参照してください。

<時刻印データ>

時刻印データの指定については、「3.2.6(4) 時刻印データの規定の文字列表現」を参照してください。

<文字列>

文字列の指定については、「3.2.6(1) 定数の種類と表記方法」を参照してください。

数定数

整数定数、浮動小数点定数または 10 進定数を指定します。

<整数定数>

整数定数のデータ型は、数字の末尾に L または l を指定すると BIGINT 型になります。数字だけを指定すると、INTEGER 型になります。

整数定数の指定については、「3.2.6(1) 定数の種類と表記方法」を参照してください。

<浮動小数点定数>

浮動小数点定数のデータ型は DOUBLE 型です。

浮動小数点定数の指定については、「3.2.6(1) 定数の種類と表記方法」を参照してください。

< 10 進定数 >

10 進定数のデータ型は DECIMAL 型または NUMERIC 型です。

10 進定数の指定については、「3.2.6(1) 定数の種類と表記方法」を参照してください。

(4) 構文規則

ありません。

(5) 注意事項

- 文字列にエスケープ文字はありません。文字列にタブを入れた場合や途中で改行した場合は、その文字コードが文字列定数に埋め込まれます。
- 値がオーバーフローした浮動小数点定数はエラーになりません。値は無限大になります。
- 値が DECIMAL 型 (NUMERIC 型) の上下限値の範囲外となった 10 進定数はエラーになりません。値は CQL で記述した値で表現されます。

(6) 使用例

リレーション s1 の列 a の値に -5.3 を加算し、列 b に 4 を乗じた結果を出力します。下線部が定数の部分です。

```
REGISTER QUERY q1 SELECT -5.3+s1.a AS xxx, 4*s1.b AS xxy FROM s1[ROWS 100];
```

4.4.20 集合関数

(1) 形式

集合関数 ::= {COUNT(' <値式> | *') | 一般集合関数 | <組み込み集合関数>}
 一般集合関数 ::= {MAX | MIN | SUM | AVG} (' <値式>')

(2) 機能

複数行から算出される値を求めます。

(3) オペランド

COUNT

入力行数を求めます。

<値式>

集合関数の引数を指定します。値式については、「4.4.18 値式」を参照してください。

*

すべての行数を求めます。

一般集合関数

MAX, MIN, SUM, AVG のどれかを指定します。引数中には、列指定を含む値式を指定してください。

MAX

最大値を求める集合関数です。

MIN

最小値を求める集合関数です。

SUM

合計値を求める集合関数です。

AVG

平均値を求める集合関数です。

<組み込み集合関数>

組み込み集合関数の指定については、「4.4.21 組み込み集合関数」を参照してください。

(4) 構文規則

- 一般集合関数には、引数に ALL（全件取得）を制御する修飾子は存在しません。すべて ALL で動作します。
- 集合関数は、その集合関数が含まれるリレーション式の選択式中、または HAVING 句中に指定してください。
- FROM 句、WHERE 句、および GROUP BY 句のうち、最後に指定された句で取得されるリレーションを集合関数の入力とします。ただし、GROUP BY 句を指定しない場合、FROM 句または WHERE 句の結果がグループ化列を持たない一つのグループとなります。
- リレーション式の SELECT 句の選択式に集合関数および列指定、または列名を指定する場合は、GROUP BY 句を適用します。
- 集合関数の引数の型（列のデータ型）と結果の型の対応は、次の表に示すとおりです。

表 4-8 集合関数の引数の型と結果の型の対応

項番	引数の型	集合関数				
		COUNT(引数)	MAX(引数)	MIN(引数)	SUM(引数)	AVG(引数)
1	INTEGER	INTEGER	INTEGER	INTEGER	INTEGER	INTEGER
2	SMALLINT	INTEGER	SMALLINT	SMALLINT	SMALLINT	SMALLINT
3	TINYINT	INTEGER	TINYINT	TINYINT	TINYINT	TINYINT
4	BIGINT	INTEGER	BIGINT	BIGINT	BIGINT	BIGINT
5	DECIMAL	INTEGER	DECIMAL	DECIMAL	DECIMAL	DECIMAL
6	NUMERIC	INTEGER	NUMERIC	NUMERIC	NUMERIC	NUMERIC
7	REAL	INTEGER	REAL	REAL	REAL	REAL
8	FLOAT	INTEGER	FLOAT	FLOAT	FLOAT	FLOAT
9	DOUBLE	INTEGER	DOUBLE	DOUBLE	DOUBLE	DOUBLE

項番	引数の型	集合関数				
		COUNT(引数)	MAX(引数)	MIN(引数)	SUM(引数)	AVG(引数)
10	CHAR	INTEGER	CHAR	CHAR	—	—
11	VARCHAR	INTEGER	VARCHAR	VARCHAR	—	—
12	DATE	INTEGER	DATE	DATE	—	—
13	TIME	INTEGER	TIME	TIME	—	—
14	TIMESTAMP	INTEGER	TIMESTAMP	TIMESTAMP	—	—

(凡例)

—：使用できません。

この表で、集合関数の引数の型が TINYINT の場合、集合関数 AVG には、入力行数が 128 以上になる引数を指定しないでください。また、引数の型が SMALLINT の場合、入力行数が 32,768 以上になる引数を指定しないでください。

(5) 注意事項

演算途中でオーバーフローが発生した場合は処理を継続します。このとき、整数であればオーバーフローしたままの値（入りきれない上位バイトをカットし、下位バイトをそのまま使用する）となり、浮動小数点値であれば無限大になります。ただし、DECIMAL 型または NUMERIC 型の値と、浮動小数点値との演算で、浮動小数点値でオーバーフローが発生した場合は、エラー（クエリグループが閉塞）になります。

(6) 使用例

リレーション s1 の列 a の行数を出力します。下線部が集合関数の部分です。

```
REGISTER QUERY q1 SELECT COUNT(s1.a) AS a1 FROM s1[ROWS 100];
```

4.4.21 組み込み集合関数

(1) 形式

組み込み集合関数 ::= <統計関数>

(2) 機能

「4.4.20 集合関数」に指定する組み込み集合関数を指定します。

(3) オペランド

<統計関数>

統計関数の指定については、「5.2 統計関数の詳細」を参照してください。

(4) 構文規則

組み込み集合関数の構文規則は、集合関数の構文規則と同じです。「4.4.20 集合関数」の構文規則を参照してください。

(5) 注意事項

- クエリに記述した組み込み集合関数の形式に不正がある場合、クエリグループの登録に失敗します。

- 組み込み集合関数は、演算方法について特に説明がない場合、差分演算を行います。このため、組み込み集合関数の計算量は一定で、入力リレーションのタプル数に比例しません。
- このほかの組み込み集合関数の注意事項は、集合関数の注意事項と同じです。「4.4.20 集合関数」の注意事項を参照してください。

(6) 使用例

リレーション s1 の列 a, b の相関係数を出力します。下線部が組み込み集合関数の部分です。

```
REGISTER QUERY q1 SELECT CORREL(s1.a, s1.b) AS a1 FROM s1[ROWS 100];
```

4.4.22 スカラ関数

(1) 形式

スカラ関数 ::= <組み込みスカラ関数>

(2) 機能

単一行から算出される値を求めます。

(3) オペランド

<組み込みスカラ関数>

組み込みスカラ関数の指定については、「4.4.23 組み込みスカラ関数」を参照してください。

(4) 構文規則

比較述語の左辺または右辺では、スカラ関数の引数の列指定で、すべて同じデータ識別子の列を指定する必要があります。

(5) 注意事項

ありません。

(6) 使用例

リレーション s1 の列 a の絶対値を出力します。下線部がスカラ関数の部分です。

```
REGISTER QUERY q1 SELECT DABS(s1.a) AS a1 FROM s1[ROWS 100];
```

4.4.23 組み込みスカラ関数

(1) 形式

組み込みスカラ関数 ::= {<数学関数> | <文字列関数> | <時刻関数> | <変換関数>}

(2) 機能

「4.4.22 スカラ関数」に指定する組み込みスカラ関数を指定します。

(3) オペランド

<数学関数>

数学関数の指定については、「5.3 数学関数の詳細」を参照してください。

<文字列関数>

文字列関数の指定については、「5.4 文字列関数の詳細」を参照してください。

<時刻関数>

時刻関数の指定については、「5.5 時刻関数の詳細」を参照してください。

<変換関数>

変換関数の指定については、「5.6 変換関数の詳細」を参照してください。

(4) 構文規則

- FLOAT 型は DOUBLE 型のシノニムのため、引数の値式の結果が FLOAT 型の場合、引数は DOUBLE 型として扱います。
- CQL で次の個所に指定できます。
 - SELECT 句の選択式
 - WHERE 句
 - HAVING 句

(5) 注意事項

- 数学関数の演算途中でオーバーフローが発生した場合は、浮動小数点値は無限大になります。
- 整数値の 0 除算が発生した場合は、クエリグループが閉塞します。
- 正規表現を使用する文字列関数で、指定した正規表現の構文に誤りがある場合は、正規表現の解析ができなため、クエリグループが閉塞します。
- クエリに記述した組み込みスカラ関数の形式に不正がある場合は、クエリグループの登録に失敗します。

(6) 使用例

リレーション s1 の列 a の正弦値を出力します。下線部が組み込みスカラ関数の部分です。

```
REGISTER QUERY q1 SELECT DSIN(s1.a) AS a1 FROM s1[ROWS 100];
```

4.4.24 ストリーム間演算関数

(1) 形式

ストリーム間演算関数 ::= <外部定義ストリーム間演算関数>

(2) 機能

ストリームデータ間の演算を行います。

(3) オペランド

<外部定義ストリーム間演算関数>

外部定義ストリーム間演算関数の指定については、「4.4.25 外部定義ストリーム間演算関数」を参照してください。

(4) 構文規則

ありません。

(5) 注意事項

ありません。

(6) 使用例

「4.4.25 外部定義ストリーム間演算関数」の使用例を参照してください。

4.4.25 外部定義ストリーム間演算関数

(1) 形式

```
外部定義ストリーム間演算関数 ::= $* <関数グループ名> . <関数名>
    { '[' <定数> [ , <定数> ] ... ' ] ' }
    { ' (<ストリーム名> [ , <ストリーム名> ] ... ) ' }
    { ' (▲AS▲ ( <相関名> [ , <相関名> ] ... ) ) ' }
```

(2) 機能

「4.4.24 ストリーム間演算関数」に指定する外部定義ストリーム間演算関数を指定します。

(3) オペランド

\$*

外部定義ストリーム間演算関数であることを示す記号です。

<関数グループ名>

外部定義ストリーム間演算関数の実装クラスを定義するグループ (関数グループ) の名称 (外部定義関数定義ファイルの FunctionGroup タグの name 属性の指定値) を指定します。

<関数名>

外部定義ストリーム間演算関数の実装メソッドを識別する関数の名称 (外部定義関数定義ファイルの StreamFunction タグの name 属性の指定値) を指定します。

<定数>

外部定義ストリーム間演算関数に設定する初期化パラメーターを定数で指定します。定数については、「4.4.19 定数」を参照してください。

定数に指定できる個数は、0~16 個です。

<ストリーム名>

外部定義ストリーム間演算関数の入力ストリーム名を指定します。

ストリーム名に指定できる個数は、1~16 個です。

<相関名>

ストリーム間演算関数の使用個所ごとに、出力ストリームの列名を区別して指定したい場合に、列名の別名を指定します。なお、外部定義ストリーム間演算関数の定義数（外部定義関数定義ファイルの ReturnInformation タグの数）と合わせる必要があります。

相関名に指定できる個数は、0~3,000 個です。

(4) 構文規則

相関名を指定した場合、後続のクエリの列指定または列名では、相関名を指定してください。外部定義関数定義ファイルで指定した列名（ReturnInformation タグの name 属性）は使用できません。

(5) 注意事項

- ストリーム名として指定できるのは、登録されたストリーム名（REGISTER STREAM 句に指定）と問い合わせの結果を、ストリーム句または別のストリーム間演算関数でストリーム化して取得したクエリ名です。
- 定数で指定するパラメーターは、実装クラスのコンストラクターの引数で渡されます。なお、定数で指定するパラメーターの型（CQL のデータ型）と、実装クラスのコンストラクターの引数の型（Java のデータ型）のマッピングについては、「3.3.1 CQL のデータ型と Java のデータ型のマッピング」を参照してください。定数で指定するパラメーターは、「3.2.6(1) 定数の種類と表記方法」の定数の表記方法に従って、CQL のデータ型として解釈されます。
- 相関名で指定する名称は名前の規則に従います。名前の規則の詳細については、「3.2.4 名前の指定」を参照してください。相関名には、同一の外部定義ストリーム間演算関数内で一意の名称を指定してください。

(6) 使用例

ストリーム s1 のデータを外部定義ストリーム間演算関数に渡し、結果をストリーム q1 として出力します。下線部が外部定義ストリーム間演算関数の部分です。

```
REGISTER QUERY q1 *$FG1.STREAM FUNC[10,1.0,'efg'](s1) AS (ca,cb,cc,cd,ce);
```

5

CQL で指定する組み込み関数

この章では、CQL で指定する組み込み関数の文法について説明します。

なお、文法説明で使用する記述形式については、「4.1 CQL 文法説明で使用する記述形式」を参照してください。

5.1 組み込み関数の一覧

組み込み関数には、組み込み集合関数と組み込みスカラ関数があります。それぞれの一覧を次の表に示します。

表 5-1 組み込み集合関数の一覧

関数の分類	関数名
統計関数	CORREL 関数
	COVAR 関数
	COVAR_POP 関数
	STDDEV 関数
	STDDEV_POP 関数
	VAR 関数
	VAR_POP 関数

表 5-2 組み込みスカラ関数の一覧

関数の分類	関数名
数学関数	ABS 関数
	ACOS 関数
	ASIN 関数
	ATAN 関数
	ATAN2 関数
	CEIL 関数
	COS 関数
	COSH 関数
	DISTANCE 関数
	DISTANCE3 関数
	EXP 関数
	FLOOR 関数
	LN 関数
	LOG 関数
	MOD 関数
	NAN 関数
	NEGATIVE_INFINITY 関数
	PI 関数

関数の分類	関数名
数学関数	POSITIVE_INFINITY 関数
	POWER 関数
	ROUND 関数
	SIN 関数
	SINH 関数
	SQRT 関数
	TAN 関数
	TANH 関数
	TODEGREES 関数
	TORADIANS 関数
文字列関数	CONCAT 関数
	LENGTH 関数
	LEQ 関数
	LGE 関数
	LGT 関数
	LLE 関数
	LLT 関数
	REGEXP_FIRSTSEARCH 関数
	REGEXP_REPLACE 関数
	REGEXP_SEARCH 関数
	SLENGTH 関数
	時刻関数
TIMESTAMPDIFF 関数	
変換関数	BIGINT_TOSTRING 関数
	DECIMAL_TOSTRING 関数
	DOUBLE_TOSTRING 関数
	INT_TOSTRING 関数
	NUMBER_TODATE 関数
	NUMBER_TOTIME 関数
	REAL_TOSTRING 関数
	TIME_TONUMBER 関数

5 CQL で指定する組み込み関数

関数の分類	関数名
変換関数	TIMESTAMP_TONUMBER 関数

5.2 統計関数の詳細

ここでは、「4.4.21 組み込み集合関数」で指定する統計関数について説明します。

統計関数

形式

統計関数 ::= <統計関数名> ' (' [<値式> [, <値式>] ...]) '

機能

統計関数は、複数行から成る集合の数値を入力し、統計データを分析する数値を求めます。

オペランド

<統計関数名>

統計関数の名称です。統計関数の一覧を次の表に示します。

表 5-3 統計関数の一覧

関数名	説明
CORREL 関数	ピアソンの相関係数を算出します。
COVAR 関数	標本共分散を算出します。
COVAR_POP 関数	母集団共分散を算出します。
STDDEV 関数	標準偏差を算出します。
STDDEV_POP 関数	母集団標準偏差を算出します。
VAR 関数	分散を算出します。
VAR_POP 関数	母集団分散を算出します。

<値式>

値式の指定については、「4.4.18 値式」を参照してください。なお、値式の個数は関数によって異なります。統計関数の各関数の説明を参照してください。

使用例

DOUBLE 型の値 S1.HEIGHT と S1.WEIGHT から相関係数を算出します。

```
register query FILTER ISTREAM (
  SELECT CORREL(S1.HEIGHT,S1.WEIGHT)
  FROM S1[ROWS 10] GROUP BY S1.HEIGHT, S1.WEIGHT );
```

CORREL 関数

形式

CORREL ' (' <value1>, <value2>) '

機能

次に示す計算式に従ってピアソンの相関係数を算出します。

$$\text{Correl}(x_i, y_i) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}$$

(x_i, y_i) は2組の数値のデータ列。 $i=1, 2, \dots, n$ で、 n は列数。

\bar{x}, \bar{y} は、 $x = \{x_i\}, y = \{y_i\}$ の相加平均。

引数

< value1 >

独立した変数 (value2) に従属する変数を値式で指定します。

< value2 >

独立した変数を値式で指定します。

戻り値

相関係数を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
CORREL	value1	DOUBLE	DOUBLE
		FLOAT	
	value2	DOUBLE	
		FLOAT	

- value1, value2 のどちらかが非数 (NaN), 負の無限大 (-Infinity), または正の無限大 (Infinity) の場合は、非数 (NaN) を返します。
- 上記の計算式で、 x_i の合計値が負の無限大 (-Infinity), または正の無限大 (Infinity) の場合は、非数 (NaN) を返します。
- 上記の計算式で、 y_i の合計値が負の無限大 (-Infinity), または正の無限大 (Infinity) の場合は、非数 (NaN) を返します。

注意事項

入力レレーションのタプル数が 1 の場合は、非数 (NaN) を返します。

COVAR 関数

形式

COVAR (' <value1>, <value2>')

機能

次に示す計算式に従って標本共分散を算出します。

$$\text{Covar}(x_i, y_i) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

引数

< value1 >

独立した変数 (value2) に従属する変数を値式で指定します。

< value2 >

独立した変数を値式で指定します。

戻り値

標本共分散を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
COVAR	value1	DOUBLE	DOUBLE
		FLOAT	
	value2	DOUBLE	
		FLOAT	

- value1, value2 のどちらかが非数 (NaN), 負の無限大 (-Infinity), または正の無限大 (Infinity) の場合は, 非数 (NaN) を返します。
- 上記の計算式で, x_i の合計値, y_i の合計値のどちらかが正の無限大 (Infinity) で, かつ $x_i y_i$ の積算結果の合計値が正の無限大 (Infinity) の場合は, 非数 (NaN) を返します。
- 上記の計算式で, x_i の合計値, y_i の合計値のどちらかが負の無限大 (-Infinity) で, かつ $x_i y_i$ の積算結果の合計値が負の無限大 (-Infinity) の場合は, 非数 (NaN) を返します。

注意事項

入力レーションのタプル数が 1 の場合は, 非数 (NaN) を返します。

COVAR_POP 関数

形式

COVAR_POP(' <value1>, <value2>')

機能

次に示す計算式に従って母集団共分散を算出します。

$$\text{CovarPop}(x_i, y_i) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n}$$

引数

< value1 >

独立した変数 (value2) に従属する変数を値式で指定します。

< value2 >

独立した変数を値式で指定します。

戻り値

母集団共分散を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
COVAR_POP	value1	DOUBLE	DOUBLE
		FLOAT	
	value2	DOUBLE	
		FLOAT	

- value1, value2 のどちらかが非数 (NaN), 負の無限大 (-Infinity), または正の無限大 (Infinity) の場合は, 非数 (NaN) を返します。
- 上記の計算式で, x_i の合計値, y_i の合計値のどちらかが正の無限大 (Infinity) で, かつ $x_i y_i$ の積算結果の合計値が正の無限大 (Infinity) の場合は, 非数 (NaN) を返します。
- 上記の計算式で, x_i の合計値, y_i の合計値のどちらかが負の無限大 (-Infinity) で, かつ $x_i y_i$ の積算結果の合計値が負の無限大 (-Infinity) の場合は, 非数 (NaN) を返します。

注意事項

入力レレーションのタプル数が 1 の場合は, 正のゼロ (0.0) を返します。

STDDEV 関数

形式

STDDEV ('<value>')

機能

次に示す計算式に従って標準偏差を算出します。

$$\text{Stddev}(x_i) = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}$$

引数

< value >

演算対象の数値を値式で指定します。

戻り値

標準偏差を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
STDDEV	value	DOUBLE	DOUBLE
		FLOAT	

- value が非数 (NaN), 負の無限大 (-Infinity), または正の無限大 (Infinity) の場合は, 非数 (NaN) を返します。
- 上記の計算式で, x_i の合計値が負の無限大 (-Infinity), または正の無限大 (Infinity) の場合は, 非数 (NaN) を返します。

注意事項

入力レーションのタプル数が 1 の場合は, 非数 (NaN) を返します。

STDDEV_POP 関数

形式

STDDEV_POP(' <value> ')

機能

次に示す計算式に従って母集団標準偏差を算出します。

$$\text{StddevPop}(x_i) = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n}}$$

引数

< value >

演算対象の数値を値式で指定します。

戻り値

母集団標準偏差を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
STDDEV_POP	value	DOUBLE	DOUBLE
		FLOAT	

- value が非数 (NaN), 負の無限大 (-Infinity), または正の無限大 (Infinity) の場合は, 非数 (NaN) を返します。
- 上記の計算式で, x_i の合計値が負の無限大 (-Infinity), または正の無限大 (Infinity) の場合は, 非数 (NaN) を返します。

注意事項

入力レレーションのタプル数が 1 の場合は, 正のゼロ (0.0) を返します。

VAR 関数

形式

VAR' (<value>)'

機能

次に示す計算式に従って分散を算出します。

$$\text{Var}(x_i) = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}$$

引数

< value >

演算対象の数値を値式で指定します。

戻り値

分散を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
VAR	value	DOUBLE	DOUBLE
		FLOAT	

- value が非数 (NaN), 負の無限大 (-Infinity), または正の無限大 (Infinity) の場合は, 非数 (NaN) を返します。
- 上記の計算方式で, x_i の合計値が負の無限大 (-Infinity), または正の無限大 (Infinity) で, かつ x_i の 2 乗の合計値が正の無限大 (Infinity) の場合は, 非数 (NaN) を返します。

注意事項

入力レーションのタプル数が 1 の場合は、非数 (NaN) を返します。

VAR_POP 関数

形式

VAR_POP ('<value>')

機能

次に示す計算式に従って母集団分散を算出します。

$$\text{VarPop}(x_i) = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n}$$

引数

< value >

演算対象の数値を値式で指定します。

戻り値

母集団分散を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
VAR_POP	value	DOUBLE	DOUBLE
		FLOAT	

- value が非数 (NaN), 負の無限大 (-Infinity), または正の無限大 (Infinity) の場合は、非数 (NaN) を返します。
- 上記の計算式で、 x_i の合計値が負の無限大 (-Infinity), または正の無限大 (Infinity) で、かつ x_i の 2 乗の合計値が正の無限大 (Infinity) の場合は、非数 (NaN) を返します。

注意事項

入力レーションのタプル数が 1 の場合は、正のゼロ (0.0) を返します。

5.3 数学関数の詳細

ここでは、「4.4.23 組み込みスカラ関数」で指定する数学関数について説明します。

数学関数

形式

数学関数 ::= <プリフィックス文字><数学関数名>' (' [<値式> [, <値式>] ...])'

機能

数学関数は、数値を引数とし、演算した値を返します。

オペランド

<プリフィックス文字>

数学関数の戻り値のデータ型に対応したアルファベット 1 文字です。プリフィックス文字を次に示します。

表 5-4 プリフィックス文字と戻り値のデータ型

プリフィックス文字	戻り値のデータ型
D	倍精度実数型 (DOUBLE/FLOAT)
R	単精度実数型 (REAL)
L	倍精度整数型 (BIGINT)
I	単精度整数型 (INTEGER)

<数学関数名>

数学関数の名称です。

数学関数名の前に、数学関数の戻り値のデータ型に対応したプリフィックス文字を指定してください。なお、プリフィックス文字と数学関数名の間に空白を入れることはできません。

数学関数の一覧を次の表に示します。

表 5-5 数学関数の一覧

関数名	説明
ABS 関数	絶対値を返します。
ACOS 関数	逆余弦を返します。
ASIN 関数	逆正弦を返します。
ATAN 関数	逆正接を返します。
ATAN2 関数	直交座標 (x, y) から逆正接を返します。
CEIL 関数	引数の値以上で、最も近い整数の浮動小数点値を返します。

関数名	説明
COS 関数	余弦を返します。
COSH 関数	双曲線余弦を返します。
DISTANCE 関数	2次元の2点間の距離を返します。
DISTANCE3 関数	3次元の2点間の距離を返します。
EXP 関数	オイラー数の累乗値を返します。
FLOOR 関数	引数の値以下で、最も近い整数の浮動小数点値を返します。
LN 関数	自然対数値を返します。
LOG 関数	対数値を返します。
MOD 関数	剰余を返します。
NAN 関数	NaN を返します。
NEGATIVE_INFINITY 関数	負の無限大 (-Infinity) を返します。
PI 関数	π の近似値を返します。
POSITIVE_INFINITY 関数	正の無限大 (Infinity) を返します。
POWER 関数	累乗値を返します。
ROUND 関数	指定した小数点のけた数に四捨五入した値を返します。
SIN 関数	正弦を返します。
SINH 関数	双曲線正弦を返します。
SQRT 関数	平方根を返します。
TAN 関数	正接を返します。
TANH 関数	双曲線正接を返します。
TODEGREES 関数	度数に変換した値を返します。
TORADIANS 関数	ラジアンに変換した値を返します。

<値式>

値式については、「4.4.18 値式」を参照してください。なお、値式の個数は関数によって異なります。数学関数の各関数の説明を参照してください。

使用例

DOUBLE 型の値 S1.C1 の絶対値が 5.0 と等しいタプルだけを出力します。DOUBLE 型の絶対値を取得するため、数学関数名「ABS」の前にプレフィックス文字「D」を付けて、「DABS」と指定しています。

```
register query FILTER ISTREAM (
  SELECT *
  FROM S1[ROWS 10]
  WHERE DABS(S1.C1) = 5.0 );
```

ABS 関数

形式

<プリフィックス文字>ABS '(<value>)'

機能

引数 value の絶対値を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ABS	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	Infinity
負の最小値	正の最大値
value<0	value の絶対値 (符号を逆にした値)
負の最大値	正の最小値
-0.0 (負のゼロ)	0.0 (正のゼロ)
0.0 (正のゼロ)	0.0 (正のゼロ)
正の最小値	正の最小値
0<value	value
正の最大値	正の最大値
Infinity	Infinity

注意事項

ありません。

ACOS 関数

形式

<プリフィックス文字>ACOS '(<value>）'

機能

引数 value の逆余弦（アークコサイン）を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ACOS	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	NaN
負の最小値	NaN
value < -1.0	NaN
$-1.0 \leq \text{value} \leq 1.0$	0.0 ~ π の範囲の値
1.0 < value	NaN
正の最大値	NaN
Infinity	NaN

注意事項

ありません。

ASIN 関数

形式

<プリフィックス文字>ASIN '(<value>）'

機能

引数 value の逆正弦（アークサイン）を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ASIN	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	NaN
負の最小値	NaN
value < -1.0	NaN
$-1.0 \leq \text{value} < -0.0$	$-\pi/2 \sim \pi/2$ の範囲の値
負の最大値	$-\pi/2 \sim \pi/2$ の範囲の値
-0.0 (負のゼロ)	-0.0 (負のゼロ)
0.0 (正のゼロ)	0.0 (正のゼロ)
正の最小値	$-\pi/2 \sim \pi/2$ の範囲の値
$0.0 < \text{value} \leq 1.0$	$-\pi/2 \sim \pi/2$ の範囲の値
$1.0 < \text{value}$	NaN
正の最大値	NaN
Infinity	NaN

注意事項

ありません。

ATAN 関数

形式

<プリフィックス文字>ATAN '(<value>)'

機能

引数 value の逆正接（アークタンジェント）を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ATAN	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	$-\pi/2$
負の最小値	計算した逆正接の値 ($-\pi/2 \sim \pi/2$)
value < -0.0	計算した逆正接の値 ($-\pi/2 \sim \pi/2$)
負の最大値	計算した逆正接の値 ($-\pi/2 \sim \pi/2$)
-0.0 (負のゼロ)	-0.0 (負のゼロ)
0.0 (正のゼロ)	0.0 (正のゼロ)
正の最小値	計算した逆正接の値 ($-\pi/2 \sim \pi/2$)
0.0 < value	計算した逆正接の値 ($-\pi/2 \sim \pi/2$)
正の最大値	計算した逆正接の値 ($-\pi/2 \sim \pi/2$)
Infinity	$\pi/2$

注意事項

ありません。

ATAN2 関数

形式

<プリフィックス文字>ATAN2 '(<valueX>, <valueY>)'

機能

直交座標 (valueX, valueY) から逆正接 (アークタンジェント) を返します。

直交座標 (valueX, valueY) を極座標 (r, θ) に変換し, θ 成分を返します。

引数

< valueX >

直交座標 X を値式で指定します。

< valueY >

直交座標 Y を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ATAN2	valueX	DOUBLE	DOUBLE
			FLOAT	
		valueY	DOUBLE	
			FLOAT	

- valueX, valueY の条件値と戻り値を次の表に示します。

valueX の値	valueY の値	戻り値
NaN	任意の値	NaN
-Infinity	NaN	NaN
	-Infinity	$-3\pi/4$
	負の最小値	$-\pi$
	valueY < -0.0	$-\pi$
	負の最大値	$-\pi$
	-0.0 (負のゼロ)	$-\pi$
	0.0 (正のゼロ)	π
	正の最小値	π
	0.0 < valueY	π

valueX の値	valueY の値	戻り値
-Infinity	正の最大値	π
	Infinity	$3\pi/4$
負の最小値	NaN	NaN
	-Infinity	$-\pi/2$
	負の最小値	計算した θ 成分 ($-3\pi/4$ の近似値)
	valueY<-0.0	計算した θ 成分 ($-\pi$ の近似値)
	負の最大値	計算した θ 成分 ($-\pi$ の近似値)
	-0.0 (負のゼロ)	$-\pi$
	0.0 (正のゼロ)	π
	正の最小値	計算した θ 成分 (π の近似値)
	0.0<valueY	計算した θ 成分 (π の近似値)
	正の最大値	計算した θ 成分 ($3\pi/4$ の近似値)
	Infinity	$\pi/2$
	valueX<-0.0	NaN
-Infinity		$-\pi/2$
負の最小値		計算した θ 成分 ($-\pi/2$ の近似値)
valueY<-0.0		計算した θ 成分
負の最大値		計算した θ 成分 ($-\pi$ の近似値)
-0.0 (負のゼロ)		$-\pi$
0.0 (正のゼロ)		π
正の最小値		計算した θ 成分 (π の近似値)
0.0<valueY		計算した θ 成分
正の最大値		計算した θ 成分 ($\pi/2$ の近似値)
Infinity		$\pi/2$
負の最大値		NaN
	-Infinity	$-\pi/2$
	負の最小値	計算した θ 成分 ($-\pi/2$ の近似値)
	valueY<-0.0	計算した θ 成分 ($-\pi/2$ の近似値)
	負の最大値	計算した θ 成分
	-0.0 (負のゼロ)	$-\pi$
	0.0 (正のゼロ)	π

valueX の値	valueY の値	戻り値
負の最大値	正の最小値	計算した θ 成分
	$0.0 < \text{valueY}$	計算した θ 成分 ($\pi/2$ の近似値)
	正の最大値	計算した θ 成分 ($\pi/2$ の近似値)
	Infinity	$\pi/2$
-0.0 (負のゼロ)	NaN	NaN
	-Infinity	$-\pi/2$
	負の最小値	$-\pi/2$
	$\text{valueY} < -0.0$	$-\pi/2$
	負の最大値	$-\pi/2$
	-0.0 (負のゼロ)	$-\pi$
	0.0 (正のゼロ)	π
	正の最小値	$\pi/2$
	$0.0 < \text{valueY}$	$\pi/2$
	正の最大値	$\pi/2$
	Infinity	$\pi/2$
0.0 (正のゼロ)	NaN	NaN
	-Infinity	$-\pi/2$
	負の最小値	$-\pi/2$
	$\text{valueY} < -0.0$	$-\pi/2$
	負の最大値	$-\pi/2$
	-0.0 (負のゼロ)	-0.0 (負のゼロ)
	0.0 (正のゼロ)	0.0 (正のゼロ)
	正の最小値	$\pi/2$
	$0.0 < \text{valueY}$	$\pi/2$
	正の最大値	$\pi/2$
	Infinity	$\pi/2$
正の最小値	NaN	NaN
	-Infinity	$-\pi/2$
	負の最小値	計算した θ 成分 ($-\pi/2$ の近似値)
	$\text{valueY} < -0.0$	計算した θ 成分 ($-\pi/2$ の近似値)
	負の最大値	計算した θ 成分

valueX の値	valueY の値	戻り値
正の最小値	-0.0 (負のゼロ)	-0.0 (負のゼロ)
	0.0 (正のゼロ)	0.0 (正のゼロ)
	正の最小値	計算した θ 成分
	$0.0 < \text{valueY}$	計算した θ 成分 ($\pi/2$ の近似値)
	正の最大値	計算した θ 成分 ($\pi/2$ の近似値)
	Infinity	$\pi/2$
$0.0 < \text{valueX}$	NaN	NaN
	-Infinity	$-\pi/2$
	負の最小値	計算した θ 成分 ($-\pi/2$ の近似値)
	$\text{valueY} < -0.0$	計算した θ 成分
	負の最大値	計算した θ 成分
	-0.0 (負のゼロ)	-0.0 (負のゼロ)
	0.0 (正のゼロ)	0.0 (正のゼロ)
	正の最小値	計算した θ 成分
	$0.0 < \text{valueY}$	計算した θ 成分
	正の最大値	計算した θ 成分 ($\pi/2$ の近似値)
	Infinity	$\pi/2$
	正の最大値	NaN
-Infinity		$-\pi/2$
負の最小値		計算した θ 成分 ($-\pi/4$ の近似値)
$\text{valueY} < -0.0$		計算した θ 成分
負の最大値		計算した θ 成分 (-0.0 の近似値)
-0.0 (負のゼロ)		-0.0 (負のゼロ)
0.0 (正のゼロ)		0.0 (正のゼロ)
正の最小値		計算した θ 成分 (0.0 の近似値)
$0.0 < \text{valueY}$		計算した θ 成分
正の最大値		計算した θ 成分 ($\pi/4$ の近似値)
Infinity		$\pi/2$
Infinity	NaN	NaN
	-Infinity	$-\pi/4$
	負の最小値	-0.0 (負のゼロ)

valueX の値	valueY の値	戻り値
Infinity	valueY < -0.0	-0.0 (負のゼロ)
	負の最大値	-0.0 (負のゼロ)
	-0.0 (負のゼロ)	-0.0 (負のゼロ)
	0.0 (正のゼロ)	0.0 (正のゼロ)
	正の最小値	0.0 (正のゼロ)
	0.0 < valueY	0.0 (正のゼロ)
	正の最大値	0.0 (正のゼロ)
	Infinity	$\pi/4$

注意事項

ありません。

CEIL 関数

形式

<プリフィックス文字>CEIL '(<value>)'

機能

引数 value の値以上で、最も近い整数の浮動小数点値を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	CEIL	value	DOUBLE	DOUBLE
			FLOAT	

- value が整数の場合は、value と同じ値を返します。
- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	-Infinity

value の値	戻り値
負の最小値	負の最小値
$value \leq -1.0$	value の値以上で最も近い整数の浮動小数点値
$-1.0 < value < -0.0$	-0.0 (負のゼロ)
負の最大値	-0.0 (負のゼロ)
-0.0 (負のゼロ)	-0.0 (負のゼロ)
0.0 (正のゼロ)	0.0 (正のゼロ)
正の最小値	1.0
$0.0 < value$	value の値以上で最も近い整数の浮動小数点値
正の最大値	正の最大値
Infinity	Infinity

注意事項

ありません。

COS 関数

形式

<プリフィックス文字>COS' (<value>)'

機能

引数 value の余弦（コサイン）を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	COS	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN

value の値	戻り値
-Infinity	NaN
負の最小値	計算した余弦の値
value<-0.0	計算した余弦の値
負の最大値	1.0 (計算した余弦の値)
-0.0 (負のゼロ)	1.0
0.0 (正のゼロ)	1.0
正の最小値	1.0 (計算した余弦の値)
0.0<value	計算した余弦の値
正の最大値	計算した余弦の値
Infinity	NaN

注意事項

ありません。

COSH 関数

形式

<プリフィックス文字>COSH '(<value>)'

機能

引数 value の双曲線余弦を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	COSH	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN

value の値	戻り値
-Infinity	Infinity
負の最小値	Infinity
value<-0.0	計算した双曲線余弦の値
負の最大値	1.0 (計算した双曲線余弦の値)
-0.0 (負のゼロ)	1.0
0.0 (正のゼロ)	1.0
正の最小値	1.0 (計算した双曲線余弦の値)
0.0<value	計算した双曲線余弦の値
正の最大値	Infinity
Infinity	Infinity

注意事項

ありません。

DISTANCE 関数

形式

<プリフィックス文字>DISTANCE '(<x1>, <y1>, <x2>, <y2>)'

機能

2次元の2点 (x1, y1), (x2, y2) 間の距離を返します。

引数

< x1 >

点 (x1, y1) の x 軸上の位置を値式で指定します。

< y1 >

点 (x1, y1) の y 軸上の位置を値式で指定します。

< x2 >

点 (x2, y2) の x 軸上の位置を値式で指定します。

< y2 >

点 (x2, y2) の y 軸上の位置を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	DISTANCE	x1	DOUBLE	DOUBLE
			FLOAT	
		y1	DOUBLE	
			FLOAT	
		x2	DOUBLE	
			FLOAT	
y2	DOUBLE			
	FLOAT			

- x1, y1, x2, y2 のどれかが非数 (NaN) の場合は、非数 (NaN) を返します。
- x1, y1, x2, y2 のどれかが正の無限大 (Infinity), または負の無限大 (-Infinity) で、かつそれ以外の引数が非数 (NaN) でない場合は、正の無限大 (Infinity) を返します。

注意事項

ありません。

DISTANCE3 関数

形式

<プリフィックス文字>DISTANCE3 '(<x1>, <y1>, <z1>, <x2>, <y2>, <z2>)'

機能

3次元の2点 (x1, y1, z1), (x2, y2, z2) 間の距離を返します。

引数

< x1 >

点 (x1, y1, z1) の x 軸上の位置を値式で指定します。

< y1 >

点 (x1, y1, z1) の y 軸上の位置を値式で指定します。

< z1 >

点 (x1, y1, z1) の z 軸上の位置を値式で指定します。

< x2 >

点 (x2, y2, z2) の x 軸上の位置を値式で指定します。

< y2 >

点 (x2, y2, z2) の y 軸上の位置を値式で指定します。

< z2 >

点 (x2, y2, z2) の z 軸上の位置を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	DISTANCE3	x1	DOUBLE	DOUBLE
			FLOAT	
		y1	DOUBLE	
			FLOAT	
		z1	DOUBLE	
			FLOAT	
		x2	DOUBLE	
			FLOAT	
		y2	DOUBLE	
			FLOAT	
		z2	DOUBLE	
			FLOAT	

- x1, y1, z1, x2, y2, z2 のどれかが非数 (NaN) の場合は、非数 (NaN) を返します。
- x1, y1, z1, x2, y2, z2 のどれかが正の無限大 (Infinity), または負の無限大 (-Infinity) で、かつそれ以外の引数が非数 (NaN) でない場合は、正の無限大 (Infinity) を返します。

注意事項

ありません。

EXP 関数

形式

<プリフィックス文字>EXP' (<value>)'

機能

引数 value を指数としオイラー数 e を累乗した値 e^{value} を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	EXP	value	DOUBLE	DOUBLE
			FLOAT	

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	0.0
負の最小値	0.0 (オイラー数 e を累乗した値 e^{value})
$\text{value} < -0.0$	オイラー数 e を累乗した値 e^{value}
負の最大値	1.0 (オイラー数 e を累乗した値 e^{value})
-0.0 (負のゼロ)	1.0
0.0 (正のゼロ)	1.0
正の最小値	1.0 (オイラー数 e を累乗した値 e^{value})
$0.0 < \text{value}$	オイラー数 e を累乗した値 e^{value}
正の最大値	Infinity
Infinity	Infinity

注意事項

ありません。

FLOOR 関数

形式

<プリフィックス文字>FLOOR' (<value>)'

機能

引数 value の値以下で、最も近い整数の浮動小数点値を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	FLOOR	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	-Infinity
負の最小値	負の最小値
value < -0.0	value の値以下で最も近い整数の浮動小数点値
負の最大値	-1.0
-0.0 (負のゼロ)	-0.0
0.0 (正のゼロ)	0.0
正の最小値	0.0
0.0 < value	value の値以下で最も近い整数の浮動小数点値
正の最大値	正の最大値
Infinity	Infinity

注意事項

ありません。

LN 関数

形式

<プリフィックス文字>LN '(<value>)'

機能

引数 value の自然対数値を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	LN	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	NaN
負の最小値	NaN
value < -0.0	NaN
負の最大値	NaN
-0.0 (負のゼロ)	-Infinity
0.0 (正のゼロ)	-Infinity
正の最小値	value の自然対数値
0.0 < value	value の自然対数値
正の最大値	value の自然対数値
Infinity	Infinity

注意事項

ありません。

LOG 関数

形式

<プリフィックス文字>LOG '(<base>, <value>)'

機能

引数 base を底とした引数 value の対数値を返します。戻り値は、 $\log_{\text{base}} \text{value}$ の値となります。

引数

< base >

底を値式で指定します。

< value >

真数を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	LOG	base	DOUBLE	DOUBLE
			FLOAT	
		value	DOUBLE	
			FLOAT	

- base, value の条件値と戻り値を次の表に示します。

base の値	value の値	戻り値
NaN	任意の値	NaN
-Infinity	任意の値	NaN
負の最小値	任意の値	NaN
base<0.0	任意の値	NaN
負の最大値	任意の値	NaN
0.0	任意の値	NaN
正の最小値	NaN	NaN
	-Infinity	NaN
	負の最小値	NaN
	value<0.0	NaN
	負の最大値	NaN
	0.0	Infinity
	正の最小値	1.0
	0.0<value<1.0	log _{base} value の値
	value=1.0	-0.0 (負のゼロ)
	1.0<value	log _{base} value の値
正の最大値	log _{base} value の値	
Infinity	-Infinity	
0.0<base<1.0	NaN	NaN
	-Infinity	NaN

base の値	value の値	戻り値
0.0 < base < 1.0	負の最小値	NaN
	value < 0.0	NaN
	負の最大値	NaN
	0.0	Infinity
	正の最小値	$\log_{\text{base}} \text{value}$ の値
	0.0 < value < 1.0	$\log_{\text{base}} \text{value}$ の値
	value = 1.0	-0.0 (負のゼロ)
	1.0 < value	$\log_{\text{base}} \text{value}$ の値
	正の最大値	$\log_{\text{base}} \text{value}$ の値
	Infinity	-Infinity
base = 1.0	任意の値	NaN
1.0 < base	NaN	NaN
	-Infinity	NaN
	負の最小値	NaN
	value < 0.0	NaN
	負の最大値	NaN
	0.0	-Infinity
	正の最小値	$\log_{\text{base}} \text{value}$ の値
	0.0 < value < 1.0	$\log_{\text{base}} \text{value}$ の値
	value = 1.0	0.0
	1.0 < value	$\log_{\text{base}} \text{value}$ の値
	正の最大値	$\log_{\text{base}} \text{value}$ の値
	Infinity	Infinity
正の最大値	NaN	NaN
	-Infinity	NaN
	負の最小値	NaN
	value < 0.0	NaN
	負の最大値	NaN
	0.0	-Infinity
	正の最小値	$\log_{\text{base}} \text{value}$ の値
	0.0 < value < 1.0	$\log_{\text{base}} \text{value}$ の値

base の値	value の値	戻り値
正の最大値	value=1.0	0.0
	1.0<value	log _{base} value の値
	正の最大値	1.0
	Infinity	Infinity
Infinity	任意の値	NaN

注意事項

ありません。

MOD 関数

形式

<プリフィックス文字>MOD '(<value1>, <value2>)'

機能

引数 value1 を引数 value2 で割ったあとの剰余を返します。

引数

< value1 >

被除数を値式で指定します。

< value2 >

除数を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
I	MOD	value1	INTEGER	INTEGER
		value2	INTEGER	

注意事項

value2 が 0 と等しい場合、整数値の 0 除算が発生するため、クエリグループが閉塞します。

NAN 関数

形式

<プリフィックス文字>NAN ' ()'

機能

NaN を返します。

引数

ありません。

戻り値

戻り値のデータ型を次の表に示します。

プリフィックス文字	数学関数名	戻り値のデータ型
D	NAN	DOUBLE

注意事項

ありません。

NEGATIVE_INFINITY 関数

形式

<プリフィックス文字>NEGATIVE_INFINITY ' ()'

機能

負の無限大 (-Infinity) を返します。

引数

ありません。

戻り値

戻り値のデータ型を次の表に示します。

プリフィックス文字	数学関数名	戻り値のデータ型
D	NEGATIVE_INFINITY	DOUBLE

注意事項

ありません。

PI 関数

形式

<プリフィックス文字>PI ' ()'

機能

π の近似値を返します。

引数

ありません。

戻り値

戻り値のデータ型を次の表に示します。

プリフィックス文字	数学関数名	戻り値のデータ型
D	PI	DOUBLE

注意事項

ありません。

POSITIVE_INFINITY 関数

形式

<プリフィックス文字>POSITIVE_INFINITY ' ()'

機能

正の無限大 (Infinity) を返します。

引数

ありません。

戻り値

戻り値のデータ型を次の表に示します。

プリフィックス文字	数学関数名	戻り値のデータ型
D	POSITIVE_INFINITY	DOUBLE

注意事項

ありません。

POWER 関数

形式

<プリフィックス文字>POWER ' (<base>, <value>)'

機能

引数 base を引数 value で累乗した値を返します。戻り値は、 $base^{value}$ の値となります。

引数

< base >

基数を値式で指定します。

< value >

指数を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	DPOWER	base	DOUBLE	DOUBLE
			FLOAT	
		value	DOUBLE	
			FLOAT	

- 演算結果が DOUBLE 型, FLOAT 型のデータの範囲外となる場合は, 正の無限大 (Infinity), または負の無限大 (-Infinity) を返します。
- base, value の条件値と戻り値を次の表に示します。

base の値	value の値	戻り値
NaN	NaN	NaN
	-Infinity	NaN
	負の最小値	NaN
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	NaN
	value<-0.0 で奇数	NaN
	負の最大値	NaN
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	NaN
	1.0	NaN (base と同じ値)
	0.0<value で 1 以外の奇数	NaN
	0.0<value で偶数	NaN
	0.0<value で小数	NaN
正の最大値	NaN	

base の値	value の値	戻り値
NaN	Infinity	NaN
-Infinity	NaN	NaN
	-Infinity	0.0 (正のゼロ)
	負の最小値	0.0 (正のゼロ)
	value<-0.0 で小数	0.0 (正のゼロ)
	value<-0.0 で偶数	0.0 (正のゼロ)
	value<-0.0 で奇数	-0.0 (負のゼロ)
	負の最大値	0.0 (正のゼロ)
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	Infinity
	1.0	-Infinity (base と同じ値)
	0.0<value で 1 以外の奇数	-Infinity
	0.0<value で偶数	Infinity
	0.0<value で小数	Infinity
	正の最大値	Infinity
Infinity	Infinity	
負の最小値	NaN	NaN
	-Infinity	0.0 (正のゼロ)
	負の最小値	0.0 (正のゼロ)
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	0.0 (正のゼロ) (base の絶対値を value で累乗した値)
	value<-0.0 で 1 以外の奇数	-0.0 (負のゼロ) (base の絶対値を value で累乗した負の値)
	value=-1.0	base の絶対値を value で累乗した負の値
	負の最大値	NaN
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	NaN
	1.0	負の最小値 (base と同じ値)
	0.0<value で 1 以外の奇数	-Infinity (base の絶対値を value で累乗した負の値)

base の値	value の値	戻り値
負の最小値	0.0<value で偶数	Infinity (base の絶対値を value で累乗した値)
	0.0<value で小数	NaN
	正の最大値	Infinity (base の絶対値を value で累乗した値)
	Infinity	Infinity
-1.0<base<-0.0	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity (base の絶対値を value で累乗した値)
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	base の絶対値を value で累乗した値
	value<-0.0 で奇数	base の絶対値を value で累乗した負の値
	負の最大値	NaN
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	NaN
	1.0	base と同じ値
	0.0<value で 1 以外の奇数	base の絶対値を value で累乗した負の値
	0.0<value で偶数	base の絶対値を value で累乗した値
	0.0<value で小数	NaN
	正の最大値	0.0 (正のゼロ) (base の絶対値を value で累乗した値)
	Infinity	0.0 (正のゼロ)
base<-1.0	NaN	NaN
	-Infinity	0.0 (正のゼロ)
	負の最小値	0.0 (正のゼロ) (base の絶対値を value で累乗した値)
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	base の絶対値を value で累乗した値
	value<-0.0 で奇数	base の絶対値を value で累乗した負の値
	負の最大値	NaN
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	NaN

base の値	value の値	戻り値
base<-1.0	1.0	base と同じ値
	0.0<value で 1 以外の奇数	base の絶対値を value で累乗した負の値
	0.0<value で偶数	base の絶対値を value で累乗した値
	0.0<value で小数	NaN
	正の最大値	Infinity (base の絶対値を value で累乗した値)
	Infinity	Infinity
base=-1.0	NaN	NaN
	-Infinity	NaN
	負の最小値	1.0
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	1.0
	value<-0.0 で奇数	-1.0
	負の最大値	NaN
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	NaN
	1.0	-1.0
	0.0<value で 1 以外の奇数	-1.0
	0.0<value で偶数	1.0
	0.0<value で小数	NaN
	正の最大値	1.0
	Infinity	NaN
負の最大値	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity (base の絶対値を value で累乗した値)
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	Infinity (base の絶対値を value で累乗した値)
	value<-0.0 で奇数	-Infinity (base の絶対値を value で累乗した負の値)
	負の最大値	NaN
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0

5 CQL で指定する組み込み関数

base の値	value の値	戻り値
負の最大値	正の最小値	NaN
	1.0	base と同じ値
	0.0<value で 1 以外の奇数	-0.0 (負のゼロ) (base の絶対値を value で累乗した負の値)
	0.0<value で偶数	0.0 (正のゼロ) (base の絶対値を value で累乗した値)
	0.0<value で小数	NaN
	正の最大値	0.0 (正のゼロ) (base の絶対値を value で累乗した値)
	Infinity	0.0 (正のゼロ)
-0.0 (負のゼロ)	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity
	value<-0.0 で小数	Infinity
	value<-0.0 で偶数	Infinity
	value<-0.0 で奇数	-Infinity
	負の最大値	Infinity
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	0.0 (正のゼロ)
	1.0	-0.0 (base と同じ値)
	0.0<value で 1 以外の奇数	-0.0 (負のゼロ)
	0.0<value で偶数	0.0 (正のゼロ)
	0.0<value で小数	0.0 (正のゼロ)
	正の最大値	0.0 (正のゼロ)
Infinity	0.0 (正のゼロ)	
0.0 (正のゼロ)	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity
	value<-0.0 で小数	Infinity
	value<-0.0 で偶数	Infinity
	value<-0.0 で奇数	Infinity

base の値	value の値	戻り値
0.0 (正のゼロ)	負の最大値	Infinity
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	0.0 (正のゼロ)
	1.0	0.0 (base と同じ値)
	0.0<value で 1 以外の奇数	0.0 (正のゼロ)
	0.0<value で偶数	0.0 (正のゼロ)
	0.0<value で小数	0.0 (正のゼロ)
	正の最大値	0.0 (正のゼロ)
	Infinity	0.0 (正のゼロ)
正の最小値	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity (base の絶対値を value で累乗した値)
	value<-0.0 で小数	base の絶対値を value で累乗した値
	value<-0.0 で偶数	Infinity (base の絶対値を value で累乗した値)
	value<-0.0 で奇数	Infinity (base の絶対値を value で累乗した値)
	負の最大値	1.0 (base の絶対値を value で累乗した値)
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	1.0 (base の絶対値を value で累乗した値)
	1.0	正の最小値 (base と同じ値)
	0.0<value で 1 以外の奇数	0.0 (正のゼロ) (base の絶対値を value で累乗した値)
	0.0<value で偶数	0.0 (正のゼロ) (base の絶対値を value で累乗した値)
	0.0<value で小数	base の絶対値を value で累乗した値
	正の最大値	0.0 (正のゼロ) (base の絶対値を value で累乗した値)
	Infinity	0.0 (正のゼロ)
1.0	NaN	NaN
	-Infinity	NaN
	負の最小値	1.0

5 CQL で指定する組み込み関数

base の値	value の値	戻り値
1.0	value < -0.0 で小数	1.0
	value < -0.0 で偶数	1.0
	value < -0.0 で奇数	1.0
	負の最大値	1.0
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	1.0
	1.0	1.0
	0.0 < value で 1 以外の奇数	1.0
	0.0 < value で偶数	1.0
	0.0 < value で小数	1.0
	正の最大値	1.0
	Infinity	NaN
	1.0 < base	NaN
-Infinity		0.0 (正のゼロ)
負の最小値		0.0 (正のゼロ) (base の絶対値を value で累乗した値)
value < -0.0 で小数		base の絶対値を value で累乗した値
value < -0.0 で偶数		base の絶対値を value で累乗した値
value < -0.0 で奇数		base の絶対値を value で累乗した値
負の最大値		1.0 (base の絶対値を value で累乗した値)
-0.0 (負のゼロ)		1.0
0.0 (正のゼロ)		1.0
正の最小値		1.0 (base の絶対値を value で累乗した値)
1.0		base と同じ値
0.0 < value で 1 以外の奇数		base の絶対値を value で累乗した値
0.0 < value で偶数		base の絶対値を value で累乗した値
0.0 < value で小数		base の絶対値を value で累乗した値
正の最大値		Infinity (base の絶対値を value で累乗した値)
Infinity		Infinity

base の値	value の値	戻り値
0.0<base<1.0	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity (base の絶対値を value で累乗した値)
	value<-0.0 で小数	base の絶対値を value で累乗した値
	value<-0.0 で偶数	base の絶対値を value で累乗した値
	value<-0.0 で奇数	base の絶対値を value で累乗した値
	負の最大値	1.0 (base の絶対値を value で累乗した値)
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	1.0 (base の絶対値を value で累乗した値)
	1.0	base と同じ値
	0.0<value で 1 以外の奇数	base の絶対値を value で累乗した値
	0.0<value で偶数	base の絶対値を value で累乗した値
	0.0<value で小数	base の絶対値を value で累乗した値
	正の最大値	0.0 (正のゼロ) (base の絶対値を value で累乗した値)
Infinity	0.0 (正のゼロ)	
正の最大値	NaN	NaN
	-Infinity	0.0 (正のゼロ)
	負の最小値	0.0 (正のゼロ) (base の絶対値を value で累乗した値)
	value<-0.0 で小数	base の絶対値を value で累乗した値
	value<-0.0 で偶数	base の絶対値を value で累乗した値
	value<-0.0 で奇数	base の絶対値を value で累乗した値
	負の最大値	1.0 (base の絶対値を value で累乗した値)
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	1.0 (base の絶対値を value で累乗した値)
	1.0	base と同じ値
	0.0<value で 1 以外の奇数	Infinity (base の絶対値を value で累乗した値)
	0.0<value で偶数	Infinity (base の絶対値を value で累乗した値)

base の値	value の値	戻り値
正の最大値	0.0<value で小数	base の絶対値を value で累乗した値
	正の最大値	Infinity (base の絶対値を value で累乗した値)
	Infinity	Infinity
Infinity	NaN	NaN
	-Infinity	0.0 (正のゼロ)
	負の最小値	0.0 (正のゼロ)
	value<-0.0 で小数	0.0 (正のゼロ)
	value<-0.0 で偶数	0.0 (正のゼロ)
	value<-0.0 で奇数	0.0 (正のゼロ)
	負の最大値	0.0 (正のゼロ)
	-0.0 (負のゼロ)	1.0
	0.0 (正のゼロ)	1.0
	正の最小値	Infinity
	1.0	Infinity
	0.0<value で 1 以外の奇数	Infinity
	0.0<value で偶数	Infinity
	0.0<value で小数	Infinity
	正の最大値	Infinity
Infinity	Infinity	

注意事項

ありません。

ROUND 関数

形式

<プリフィックス文字>ROUND '(<value>, <scale>)'

機能

引数 value の値を、引数 scale で指定した小数点けた数に丸めた値 (scale + 1 の小数点位置で四捨五入した値) にして返します。

引数

< value >

丸め対象の数値を値式で指定します。

< scale >

小数点以下のけた数を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ROUND	value	DOUBLE	DOUBLE
			FLOAT	
		scale	INTEGER	

- DOUBLE 型、FLOAT 型の有効けた数は、16～17 けたです。このため、value、scale の指定が有効けた数に収まらない場合、有効けた数に丸めた数値を返します。
DROUND('4235678.28571437218732731273',10)の場合、戻り値として、4235678.285714372 を返します。
DROUND('1.28571445674562133729372137',20)の場合、戻り値として、1.2857144567456213 を返します。
- value が非数 (NaN)、正の無限大 (Infinity)、負の無限大 (-Infinity) の場合、処理が継続できないため、クエリグループが閉塞します。

注意事項

ありません。

SIN 関数

形式

<プリフィックス文字>SIN ('<value>')

機能

引数 value の正弦 (サイン) を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	SIN	value	DOUBLE	DOUBLE

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	SIN	value	FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	NaN
負の最小値	計算した正弦の値
value < -0.0	計算した正弦の値
負の最大値	計算した正弦の値
-0.0 (負のゼロ)	-0.0 (負のゼロ)
0.0 (正のゼロ)	0.0 (正のゼロ)
正の最小値	計算した正弦の値
0.0 < value	計算した正弦の値
正の最大値	計算した正弦の値
Infinity	NaN

注意事項

ありません。

SINH 関数

形式

<プリフィックス文字>SINH '(<value>)'

機能

引数 value の双曲線正弦を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	SINH	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	-Infinity
負の最小値	-Infinity (計算した双曲線正弦の値)
value < -0.0	計算した双曲線正弦の値
負の最大値	計算した双曲線正弦の値
-0.0 (負のゼロ)	-0.0 (負のゼロ)
0.0 (正のゼロ)	0.0 (正のゼロ)
正の最小値	計算した双曲線正弦の値
0.0 < value	計算した双曲線正弦の値
正の最大値	Infinity (計算した双曲線正弦の値)
Infinity	Infinity

注意事項

ありません。

SQRT 関数

形式

<プリフィックス文字>SQRT '(<value>)'

機能

引数 value の正の平方根を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	SQRT	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	NaN
負の最小値	NaN
value < -0.0	NaN
負の最大値	NaN
-0.0 (負のゼロ)	-0.0 (負のゼロ)
0.0 (正のゼロ)	0.0 (正のゼロ)
正の最小値	計算した正の平方根
0.0 < value	計算した正の平方根
正の最大値	計算した正の平方根
Infinity	Infinity

注意事項

ありません。

TAN 関数

形式

<プリフィックス文字>TAN '(<value>)'

機能

引数 value の正接（タンジェント）を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	TAN	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	NaN
負の最小値	計算した正接の値
value < -0.0	計算した正接の値
負の最大値	計算した正接の値
-0.0 (負のゼロ)	-0.0 (負のゼロ)
0.0 (正のゼロ)	0.0 (正のゼロ)
正の最小値	計算した正接の値
0.0 < value	計算した正接の値
正の最大値	計算した正接の値
Infinity	NaN

注意事項

ありません。

TANH 関数

形式

<プリフィックス文字>TANH '(<value>)'

機能

引数 value の双曲線正接を返します。

引数

< value >

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	TANH	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	-1.0
負の最小値	-1.0 (計算した双曲線正接の値)
value < -0.0	計算した双曲線正接の値
負の最大値	計算した双曲線正接の値
-0.0 (負のゼロ)	-0.0 (負のゼロ)
0.0 (正のゼロ)	0.0 (正のゼロ)
正の最小値	計算した双曲線正接の値
0.0 < value	計算した双曲線正接の値
正の最大値	1.0 (計算した双曲線正接の値)
Infinity	1.0

注意事項

ありません。

TODEGREES 関数

形式

<プリフィックス文字>TODEGREES ('<value>')

機能

ラジアンで計測した角度を度数に変換した値を返します。

なお、ラジアンから度数への変換は、正確ではありません。例えば、TORADIANS(90.0)は、浮動小数で変換するため、正確な $\pi/2$ になりません。そのため、COS(TORADIANS(90.0))は、正確に 0.0 になりません。

引数

< value >

ラジアンで計測した角度を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	TODEGREES	value	DOUBLE FLOAT	DOUBLE

- 度数への変換結果が DOUBLE 型、FLOAT 型のデータの範囲外の場合は、正の無限大 (Infinity)、または負の無限大 (-Infinity) を返します。
- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	-Infinity
負の最小値	-Infinity (度数に変換した値)
value < -0.0	度数に変換した値
負の最大値	度数に変換した値
-0.0 (負のゼロ)	-0.0 (負のゼロ)
0.0 (正のゼロ)	0.0 (正のゼロ)
正の最小値	度数に変換した値
0.0 < value	度数に変換した値
正の最大値	Infinity (度数に変換した値)
Infinity	Infinity

注意事項

ありません。

TORADIANS 関数

形式

<プリフィックス文字>TORADIANS (' <value>')

機能

度数で計測した角度をラジアンに変換した値を返します。

なお、度数からラジアンへの変換は正確ではありません。

引数

< value >

度数で計測した角度を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	TORADIANS	value	DOUBLE FLOAT	DOUBLE

- value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	-Infinity
負の最小値	ラジアンに変換した値
value < -0.0	ラジアンに変換した値
負の最大値	-0.0 (負のゼロ) (ラジアンに変換した値)
-0.0 (負のゼロ)	-0.0 (負のゼロ)
0.0 (正のゼロ)	0.0 (正のゼロ)
正の最小値	0.0 (正のゼロ) (ラジアンに変換した値)
0.0 < value	ラジアンに変換した値
正の最大値	ラジアンに変換した値
Infinity	Infinity

注意事項

ありません。

5.4 文字列関数の詳細

ここでは、「4.4.23 組み込みスカラ関数」で指定する文字列関数について説明します。

文字列関数

形式

文字列関数 ::= <文字列関数名> ('<値式> [, <値式>] ...)'

機能

文字列関数は、文字列を引数とし、検索または操作した結果を返します。

オペランド

<文字列関数名>

文字列関数の名称です。文字列関数の一覧を次の表に示します。

表 5-6 文字列関数の一覧

関数名	機能内容
CONCAT 関数	二つの引数の文字列を結合した文字列を返します。
LENGTH 関数	引数の文字列の文字数を返します。文字列にサロゲートペアが含まれている場合、サロゲートペア 1 文字を 2 文字として数えて結果を返します。
LEQ 関数	辞書的な順序で、引数の文字列が等しいかどうかを比較し、比較結果を返します。
LGE 関数	辞書的な順序で、引数の文字列が大きいかどうか、または等しいかどうかを比較し、比較結果を返します。
LGT 関数	辞書的な順序で、引数の文字列が大きいかどうかを比較し、比較結果を返します。
LLE 関数	辞書的な順序で、引数の文字列が小さいかどうか、または等しいかどうかを比較し、比較結果を返します。
LLT 関数	辞書的な順序で、引数の文字列が小さいかどうかを比較し、比較結果を返します。
REGEXP_FIRSTSEARCH 関数	引数の文字列を正規表現パターンで検索し、最初に一致した文字列を返します。
REGEXP_REPLACE 関数	引数の文字列を正規表現パターンで検索し、最初に一致した文字列を置換文字列に置き換えて返します。
REGEXP_SEARCH 関数	引数の文字列を正規表現パターンで検索し、パターンに一致する部分があるかどうかを返します。
SPLength 関数	引数の文字列の文字数を返します。文字列にサロゲートペアが含まれている場合、サロゲートペア 1 文字を 1 文字として数えて結果を返します。

<値式>

値式については、「4.4.18 値式」を参照してください。なお、値式の個数は関数によって異なります。文字列関数の各関数の説明を参照してください。

使用例

日付データ YYYY-MM-DD の固定形式である CHAR 型の S1.DATE1 から年の文字列を抽出します。正規表現'^\d\d\d\d'は、先頭の数値文字列 4 文字を表現しています。

```
register query FILTER ISTREAM (
  SELECT REGEXP_FIRSTSEARCH(S1.DATE1, '^d\d\d\d')
  FROM S1[ROWS 10] );
```

S1.DATE1 が 2013-01-23 の場合、2013 を返します。

注意事項

- 「3.3.1 CQL のデータ型と Java のデータ型のマッピング」に示すように、CHAR 型、VARCHAR 型は、Java の java.lang.String クラスに対応しています。
- 引数にサロゲートペアが含まれる場合の文字列関数の動作について、次の表に示します。

表 5-7 サロゲートペア入力時の動作

文字列関数名	文字列関数の動作
CONCAT	○：通常に動作します。
LENGTH	×：通常に動作しません。サロゲートペア 1 文字を 2 文字として数えて結果を返します。
LEQ	○：通常に動作します。
LGE	○：通常に動作します。
LGT	○：通常に動作します。
LLE	○：通常に動作します。
LLT	○：通常に動作します。
REGEXP_FIRSTSEARCH	○：通常に動作します。
REGEXP_REPLACE	○：通常に動作します。
REGEXP_SEARCH	○：通常に動作します。
SPLength	○：通常に動作します。サロゲートペア 1 文字を 1 文字として数えて結果を返します。

CONCAT 関数

形式

```
CONCAT ('<string1>,<string2>')
```

機能

第 1 引数 string1 と第 2 引数 string2 の文字列を結合した結果を返します。

引数

< string1 >

文字列を値式で指定します。

< string2 >

文字列を値式で指定します。

戻り値

string1 の文字列の最後に string2 の文字列を連結した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
CONCAT	string1	CHAR	VARCHAR
		VARCHAR	
	string2	CHAR	
		VARCHAR	

- 戻り値の例を次に示します。
string1 が'Abc', string2 が'Def'の場合, 'AbcDef'を返します。
- 結合した結果の文字列が VARCHAR 型の最大文字数 32,767 文字を超えた場合は, 超えた分は切り捨てられます。

注意事項

ありません。

LENGTH 関数

形式

LENGTH '(<string>)'

機能

引数 string の文字数を返します。string 内にサロゲートペアが含まれている場合, サロゲートペア 1 文字を 2 文字として数えて結果を返します。

引数

< string >

文字列を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LENGTH	string	CHAR	INTEGER
		VARCHAR	

- 「は+ゝ（濁点）」→「はゝ」、 「は+°（半濁点）」→「は°」のように1字を表す結合文字については、2文字として結果を返します。
 結合文字：「はゝすけっと」 = 6文字
 非結合文字：「ばすけっと」 = 5文字
- 文字列にサロゲートペアが含まれている場合、サロゲートペア1文字を、2文字として数えて結果を返します。
 「〇〇である」の「〇〇」がサロゲートペアの場合、戻り値は、7となります。

注意事項

ありません。

LEQ 関数

形式

LEQ '(<string1>, <string2>)'

機能

辞書的な順序で、引数 string1 と string2 が等しいかどうかを比較し、比較結果を返します。

引数

< string1 >

文字列を値式で指定します。

< string2 >

文字列を値式で指定します。

戻り値

- string1 = string2 の場合、1 を返します。
- string1 ≠ string2 の場合、0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LEQ	string1	CHAR	INTEGER
		VARCHAR	
	string2	CHAR	
		VARCHAR	

注意事項

辞書的な順序の定義について説明します。

次の場合に、二つの文字列が異なると判定します。

1. 両方の文字列に対して有効なインデックスに位置する文字が異なる場合
2. 文字列の長さが異なる場合
3. 1.と 2.の両方に該当する場合

なお、二つの文字列が異なる場合で、異なる文字を指す最も小さいインデックスを k とします。位置 k にある文字の CHAR 型の値を比較し「より小さい」値と判定される文字を持つ文字列が、もう一方の文字列の辞書的な前になります。

また、有効なインデックス位置での文字がすべて同じ場合、短い方の文字列が辞書的な前になります。

LGE 関数

形式

LGE '(<string1>, <string2>)'

機能

辞書的な順序で、引数 string1 が string2 より大きいかどうか、または引数 string1 と string2 が等しいかどうかを比較し、比較結果を返します。

引数

< string1 >

文字列を値式で指定します。

< string2 >

文字列を値式で指定します。

戻り値

- string1 \geq string2 の場合、1 を返します。
- string1 < string2 の場合、0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LGE	string1	CHAR	INTEGER
		VARCHAR	
	string2	CHAR	
		VARCHAR	

注意事項

辞書的な順序の定義については、「LEQ 関数」の注意事項を参照してください。

LGT 関数

形式

LGT '(<string1>, <string2>)'

機能

辞書的な順序で、引数 string1 が string2 より大きいかどうかを比較し、比較結果を返します。

引数

< string1 >

文字列を値式で指定します。

< string2 >

文字列を値式で指定します。

戻り値

- string1 > string2 の場合、1 を返します。
- string1 ≤ string2 の場合、0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LGT	string1	CHAR	INTEGER
		VARCHAR	
	string2	CHAR	
		VARCHAR	

注意事項

辞書的な順序の定義については、「LEQ 関数」の注意事項を参照してください。

LLE 関数

形式

LLE '(<string1>, <string2>)'

機能

辞書的な順序で、引数 string1 が string2 より小さいかどうか、または引数 string1 と string2 が等しいかどうかを比較し、比較結果を返します。

引数

< string1 >

文字列を値式で指定します。

< string2 >

文字列を値式で指定します。

戻り値

- string1 ≤ string2 の場合、1 を返します。
- string1 > string2 の場合、0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LLE	string1	CHAR	INTEGER
		VARCHAR	
	string2	CHAR	
		VARCHAR	

注意事項

辞書的な順序の定義については、「LEQ 関数」の注意事項を参照してください。

LLT 関数

形式

LLT '(<string1>, <string2>)'

機能

辞書的な順序で、引数 string1 が string2 より小さいかどうかを比較し、比較結果を返します。

引数

< string1 >

文字列を値式で指定します。

< string2 >

文字列を値式で指定します。

戻り値

- string1 < string2 の場合、1 を返します。
- string1 ≥ string2 の場合、0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LLT	string1	CHAR	INTEGER
		VARCHAR	
	string2	CHAR	
		VARCHAR	

注意事項

辞書的な順序の定義については、「LEQ 関数」の注意事項を参照してください。

REGEXP_FIRSTSEARCH 関数

形式

REGEXP_FIRSTSEARCH '(<string>, <regexp>)'

機能

引数 string に指定された文字列を、引数 regexp に指定した正規表現の文字列で検索し、最初に一致した文字列を返します。

引数

< string >

文字列を値式で指定します。

< regexp >

正規表現を定数で指定します。

戻り値

引数 string の文字列を、引数 regexp の正規表現の文字列で検索し、最初に一致した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
REGEXP_FIRSTSEARCH	string	CHAR	VARCHAR
		VARCHAR	
	regexp	CHAR	
		VARCHAR	

- 戻り値の例を次に示します。
REGEXP_FIRSTSEARCH('abcdefabc', '[d-z]*')の場合、'def'を返します。
- 正規表現で検索し、一致した文字列がない場合、空文字''を返します。

注意事項

- CQL 上では、正規表現を定数で指定してください。定数で指定しない場合、クエリの登録時にエラーとなります。

(正しい指定例)

文字列 input.Str1 を正規表現'.A*'の定数で検索する場合

```
SELECT
  REGEXP_FIRSTSEARCH( input.Str1, '.A*' ) AS S1
FROM input[ROWS 100];
```

(誤った指定例)

文字列 input.Str1 を文字列 input.Str2 の列名を指定して検索する場合

```
SELECT
  REGEXP_FIRSTSEARCH( input.Str1, input.Str2 ) AS S1
FROM input[ROWS 100];FROM input[ROWS 100];
```

- 正規表現の構文に誤りがある場合、正規表現の解析ができないため、クエリグループが閉塞します。
- 正規表現の解析には、java.util.regex.Pattern クラスを使用します。このため、正規表現は、java.util.regex.Pattern クラスがサポートする正規表現の範囲で記述してください。

REGEXP_REPLACE 関数

形式

```
REGEXP_REPLACE ( '<string>', <regexp>, <replacement>' )'
```

機能

引数 string に指定された文字列を、引数 regexp に指定された正規表現の文字列で検索します。最初に一致した文字列を、引数 replacement に指定された置換文字列に置き換えて返します。

引数

< string >

文字列を値式で指定します。

< regexp >

正規表現を定数で指定します。

< replacement >

置換文字列を値式で指定します。

戻り値

引数 string の文字列を、引数 regexp の正規表現の文字列で検索し、最初に一致した文字列を、引数 replacement の置換文字列に置き換えて返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
REGEXP_REPLACE	string	CHAR	VARCHAR

文字列関数名	引数	引数のデータ型	戻り値のデータ型
REGEXP_REPLACE	string	VARCHAR	VARCHAR
		CHAR	
	VARCHAR		
	replacement	CHAR	
		VARCHAR	

- 戻り値の例を次に示します。
REGEXP_REPLACE('abcdefabcdef', '[d-z]*', 'xyz')の場合、'abcxyzabcdef'を返します。
- 正規表現で検索し、一致した文字列がない場合、文字列 string をそのまま返します。

注意事項

「REGEXP_FIRSTSEARCH 関数」の注意事項を参照してください。

REGEXP_SEARCH 関数

形式

REGEXP_SEARCH '(<string>, <regexp>)'

機能

引数 string に指定された文字列を、引数 regexp に指定された正規表現の文字列で検索し、パターンに一致する部分があるかどうかを返します。

引数

< string >

文字列を値式で指定します。

< regexp >

正規表現を定数で指定します。

戻り値

- 一致する文字列がある場合、1 を返します。
- 一致する文字列がない場合、0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
REGEXP_SEARCH	string	CHAR	INTEGER
		VARCHAR	
	regexp	CHAR	

文字列関数名	引数	引数のデータ型	戻り値のデータ型
REGEXP_SEARCH	regexp	VARCHAR	INTEGER

- 戻り値の例を次に示します。
REGEXP_SEARCH('abcdefabc', '[d-z]*')の場合、1 を返します。

注意事項

「REGEXP_FIRSTSEARCH 関数」の注意事項を参照してください。

SLENGTH 関数

形式

SLENGTH '(<string>)'

機能

引数 string の文字数を返します。string 内にサロゲートペアが含まれている場合、サロゲートペア 1 文字を 1 文字として数えて結果を返します。

引数

< string >

文字列を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
SLENGTH		CHAR	INTEGER
		VARCHAR	

- 「は+^ゝ（濁点）」→「は^ゝ」、 「は+[゜]（半濁点）」→「は[゜]」のように 1 字を表す結合文字については、2 文字として結果を返します。
結合文字：「は^ゝすけっと」 = 6 文字
非結合文字：「ばすけっと」 = 5 文字
- 文字列にサロゲートペアが含まれている場合、サロゲートペア 1 文字を、1 文字として数えて結果を返します。
「○○である」の「○○」がサロゲートペアの場合、戻り値は、5 となります。

注意事項

ありません。

5.5 時刻関数の詳細

ここでは、「4.4.23 組み込みスカラ関数」で指定する時刻関数について説明します。

時刻関数

形式

時刻関数 ::= <時刻関数名> ('<値式> , <値式>')

機能

時刻関数は、日付時刻型の入力値に対して演算を実行し、演算結果を数値で返します。

オペランド

<時刻関数名>

時刻関数の名称です。時刻関数の一覧を次に示します。

表 5-8 時刻関数の一覧

関数名	説明
TIMEDIFF 関数	TIME 型の時刻の差分を計算し、結果をミリ秒の数値で返します。
TIMESTAMPDIFF 関数	TIMESTAMP 型の時刻の差分を計算し、結果をミリ秒の数値で返します。

<値式>

値式については、「4.4.18 値式」を参照してください。なお、値式の個数は関数によって異なります。時刻関数の各関数の説明を参照してください。

注意事項

- タイムゾーンは、Stream Data Platform - AF が動作している JavaVM に従います。
- 時刻関数は、夏時間に対応していません。夏時間を実施するタイムゾーンで使用した場合でも、日付データ、時刻データ、時刻印データは、標準時間に従って扱います。

使用例

TIME 型の時刻データ S1.DATETIME が 01:00:00 以前のタプルだけを出力します。

```
register query FILTER ISTREAM (
  SELECT * FROM S1[ROWS 10]
  WHERE TIMEDIFF('01:00:00', S1.DATETIME) < 0 );
```

TIMEDIFF 関数

形式

TIMEDIFF ('<time1> , <time2>')

機能

引数 time1 と time2 の差分を計算し、結果をミリ秒の数値で返します。

引数

< time1 >

時刻データを値式で指定します。

< time2 >

時刻データを値式で指定します。

戻り値

引数 time1, time2 を 1970 年 1 月 1 日 00:00:00 GMT からのミリ秒に変換し、「time2-time1 = 差分」で計算した値を返します。単位はミリ秒です。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

時刻関数名	引数	引数のデータ型	戻り値のデータ型
TIMEDIFF	time1	TIME	BIGINT
	time2	TIME	

- TIME 型は、java.sql.Time 型に対応するため、引数の時刻データは 1970 年 1 月 1 日 00:00:00 GMT が基準となります。引数に日付の繰り上がりがある時刻データを指定した場合、結果の差分は繰り上がった日付を含んだミリ秒の数値を返します。

例えば、time1 に「09:00:00」、time2 に「33:00:00」を指定した場合、time1 は「1970-01-01 09:00:00」、time2 は「1970-01-02 09:00:00」となり、time2 の日付が繰り上がります。また、戻り値は、86400000 となります。下線部が、日付の部分です。

注意事項

入力値によっては、想定しない値が返る場合があります。

例えば、TIME1='00:00:00' (時刻印データで表すと 1970-01-01 00:00:00.000)、および BIGINT1=253402236000000 を TIME 型に変換した値を入力した場合、TIMEDIFF(TIME1,NUMBER_TO_TIME(BIGINT1))は、戻り値 253402268400000 のミリ秒数値を返します。戻り値 253402268400000 を時刻印データで表すと、'10000-01-01 00:00:00.000'となります。

TIMESTAMPDIFF 関数

形式

TIMESTAMPDIFF(' <timestamp1>,<timestamp2>')

機能

引数 timestamp1 と timestamp2 の差分を計算し、結果をミリ秒の数値で返します。

引数

< timestamp1 >

時刻印データを値式で指定します。

< timestamp2 >

時刻印データを値式で指定します。

戻り値

引数 timestamp1, timestamp2 を 1970 年 1 月 1 日 00:00:00 GMT からのミリ秒に変換し、「timestamp2 - timestamp1 = 差分」で計算した値を返します。単位はミリ秒です。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

時刻関数名	引数	引数のデータ型	戻り値のデータ型
TIMESTAMPDIFF	timestamp1	TIMESTAMP	BIGINT
	timestamp2	TIMESTAMP	

注意事項

「TIMEDIFF 関数」の注意事項を参照してください。

5.6 変換関数の詳細

ここでは、「4.4.23 組み込みスカラー関数」で指定する変換関数について説明します。

変換関数

形式

変換関数 ::= <変換関数名> '(<値式>')

機能

変換関数は、引数のデータ型を別のデータ型に変換する機能を提供します。

オペランド

<変換関数名>

変換関数の名称です。変換関数の一覧を次に示します。

表 5-9 変換関数の一覧

関数名	説明
BIGINT_TOSTRING 関数	BIGINT 型の数値を文字列型に変換します。
DECIMAL_TOSTRING 関数	DECIMAL 型または NUMERIC 型の数値を文字列型に変換します。
DOUBLE_TOSTRING 関数	DOUBLE 型または FLOAT 型の数値を文字列型に変換します。
INT_TOSTRING 関数	INTEGER 型の数値を文字列型に変換します。
NUMBER_TODATE 関数	ミリ秒の数値を DATE 型の日付データに変換します。
NUMBER_TOTIME 関数	ミリ秒の数値を TIME 型の時刻データに変換します。
REAL_TOSTRING 関数	REAL 型の数値を文字列型に変換します。
TIME_TONUMBER 関数	TIME 型の時刻データをミリ秒の数値に変換します。
TIMESTAMP_TONUMBER 関数	TIMESTAMP 型の時刻印データをミリ秒の数値に変換します。

<値式>

値式については、「4.4.18 値式」を参照してください。なお、値式の個数は関数によって異なります。変換関数の各関数の説明を参照してください。

注意事項

- 変換関数の引数・戻り値での日付データ、時刻データ、時刻印データは、Stream Data Platform - AF が動作している JavaVM のタイムゾーンに従います。また、変換関数の引数・戻り値でのミリ秒の数値は、1970 年 1 月 1 日 00:00:00 GMT を基準とした値です。
- 時刻を扱う変換関数は、夏時間に対応していません。夏時間を実施するタイムゾーンで使用了した場合でも、日付データ、時刻データ、時刻印データは、標準時間に従って扱います。

使用例

INTEGER 型の S1.CODE を文字列型に変換し出力します。

```
register query FILTER ISTREAM (
SELECT INT_TOSTRING(S1.CODE) FROM S1[ROWS 10] );
```

S1.CODE が 34512 の場合、34512 を返します。

BIGINT_TOSTRING 関数

形式

BIGINT_TOSTRING '(<value>)'

機能

BIGINT 型の数値 value を文字列型に変換します。

引数

< value >

BIGINT 型の数値を値式で指定します。

戻り値

数値を文字列型に変換し、変換した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
BIGINT_TOSTRING	value	BIGINT	VARCHAR

注意事項

ありません。

DECIMAL_TOSTRING 関数

形式

DECIMAL_TOSTRING '(<value>)'

機能

DECIMAL 型または NUMERIC 型の数値 value を文字列型に変換します。

引数

< value >

DECIMAL 型または NUMERIC 型の数値を値式で指定します。

戻り値

数値を文字列型に変換し、変換した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
DECIMAL_TOSTRING	value	DECIMAL	VARCHAR
		NUMERIC	

- 数値が 10^{-6} 未満の場合、指数表現を使用した文字形式に変換します。

(例)

入力数値 「0.00000100121」 → 変換後文字列 「0.00000100121」

入力数値 「0.000000100121」 → 変換後文字列 「1.00121E-7」

注意事項

DECIMAL 型 (NUMERIC 型) 以外のデータ型から DECIMAL 型 (NUMERIC 型) へのキャスト (暗黙的なキャストを含む) では、結果のけた数が query.decimalMaxPrecision パラメーターで指定したけた数を超えた場合、指定したけた数に丸められます。引数は丸められた数値となるため、DECIMAL_TOSTRING 関数も丸められた数値を変換した文字列を返します。

(例)

query.decimalMaxPrecision=5 の場合は、次のようになります。

- DECIMAL_TOSTRING((DECIMAL)DBL1)
入力数値 (DOUBLE 型の DBL1) 「23.45666」 → 変換後文字列 「23.457」
- DECIMAL_TOSTRING(DEC1)
入力数値 (DECIMAL 型の DEC1) 「23.45666」 → 変換後文字列 「23.45666」

DOUBLE_TOSTRING 関数

形式

DOUBLE_TOSTRING ' (<value>)'

機能

DOUBLE 型または FLOAT 型の数値 value を文字列型に変換します。

引数

< value >

DOUBLE 型または FLOAT 型の数値を値式で指定します。

戻り値

数値を文字列型に変換し、変換した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
DOUBLE_TOSTRING	value	DOUBLE	VARCHAR
		FLOAT	

- 数値が 10^{-3} 未満の場合、または 10^7 以上の場合、指数表現を使用した文字形式に変換します。

(例)

入力数値「2768237.5693」 → 変換後文字列「2768237.5693」

入力数値「27682371.5693」 → 変換後文字列「2.76823715693E7」

入力数値「0.005693442」 → 変換後文字列「0.005693442」

入力数値「0.0005693442」 → 変換後文字列「5.693442E-4」

注意事項

浮動小数点型は、IEEE 754 に準拠しているため、正確に変換できない場合があります。

INT_TOSTRING 関数

形式

INT_TOSTRING '(<value>)'

機能

INTEGER 型の数値 value を文字列型に変換します。

引数

< value >

INTEGER 型の数値を値式で指定します。

戻り値

数値を文字列型に変換し、変換した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
INT_TOSTRING	value	INTEGER	VARCHAR

注意事項

ありません。

NUMBER_TODATE 関数

形式

NUMBER_TODATE '(<value>)'

機能

ミリ秒の数値 value を DATE 型の日付データに変換します。

引数

< value >

ミリ秒の数値 (BIGINT 型) を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
NUMBER_TODATE	value	BIGINT	DATE

注意事項

「NUMBER_TOTIME 関数」の注意事項を参照してください。

NUMBER_TOTIME 関数

形式

NUMBER_TOTIME ' (' <value> ')'

機能

ミリ秒の数値 value を TIME 型の時刻データに変換します。

引数

< value >

ミリ秒の数値 (BIGINT 型) を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
NUMBER_TOTIME	value	BIGINT	TIME

注意事項

入力値によっては、想定しない値が返る場合があります。

例えば、BIGINT1=9223372036854775807 を入力した場合、NUMBER_TOTIME(BIGINT1)は、戻り値 16:12:55 を返します。戻り値 16:12:55 は、時刻印データで表すと 292278994-08-17 16:12:55.807 となります。

REAL_TOSTRING 関数

形式

REAL_TOSTRING '(<value>)'

機能

REAL 型の数値 value を文字列型に変換します。

引数

< value >

REAL 型の数値を値式で指定します。

戻り値

数値を文字列型に変換し、変換した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
REAL_TOSTRING	value	REAL	VARCHAR

- 数値が 10^{-3} 未満の場合、または 10^7 以上の場合、指数表現を使用した文字形式に変換します。

(例)

入力数値「2768237.5693」 → 変換後文字列「2768237.5」

入力数値「27682371.5693」 → 変換後文字列「2.7682372E7」

入力数値「0.005693442」 → 変換後文字列「0.005693442」

入力数値「0.0005693442」 → 変換後文字列「5.693442E-4」

注意事項

浮動小数点型は、IEEE754 に準拠しているため、正確に変換できない場合があります。

TIME_TONUMBER 関数

形式

TIME_TONUMBER '(<time>)'

機能

TIME 型の時刻データ time をミリ秒の数値に変換します。

引数

< time >

TIME 型の時刻データを値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
TIME_TONUMBER	time	TIME	BIGINT

注意事項

「TIMEDIFF 関数」の注意事項を参照してください。

TIMESTAMP_TONUMBER 関数

形式

TIMESTAMP_TONUMBER '(<timestamp>)'

機能

TIMESTAMP 型の時刻印データ timestamp をミリ秒の数値に変換します。

引数

< timestamp >

TIMESTAMP 型の時刻印データを値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
TIMESTAMP_TONUMBER	timestamp	TIMESTAMP	BIGINT

注意事項

「TIMEDIFF 関数」の注意事項を参照してください。

6

クエリ定義のサンプル

この章では、Stream Data Platform - AF が提供しているクエリ定義のサンプルの種類について説明しています。

6.1 クエリ定義のサンプルの種類

Stream Data Platform - AF では、次の表に示すクエリ定義のサンプルを提供しています。これらのサンプルは、<インストールディレクトリ>%samples%下の「api」「external」「file」「httppacket」ディレクトリ下にそれぞれ格納されています。

表 6-1 クエリ定義のサンプルの種類

格納先	ファイル名	説明
api%query%	Inprocess_QueryTest	インプロセス連携カスタムアダプター用のクエリ定義のサンプルです。
	RMI_QueryTest	RMI 連携カスタムアダプター用のクエリ定義のサンプルです。
external%query%	Inprocess_QueryTest	外部定義関数を使用する場合のクエリ定義のサンプルです。
file%query%	Inprocess_QueryTest	標準提供アダプターでファイルを入力データとする場合のクエリ定義のサンプルです。
httppacket%query%	Inprocess_QueryTest	標準提供アダプターで HTTP パケットを入力データとする場合のクエリ定義のサンプルです。

参考

<インストールディレクトリ>%samples%en%下には、英語版のサンプルを提供しています。

7

カスタムアダプターの作成

この章では、カスタムアダプターの作成方法について説明します。

カスタムアダプターは、標準提供アダプターで扱えない形式のデータを扱う場合などに、必要に応じて作成してください。

なお、カスタムアダプターの作成で使用する API の詳細については、「8. データ送受信 API」を参照してください。

7.1 作成するカスタムアダプターの種類

この節では、作成するカスタムアダプターの種類について説明します。

カスタムアダプターとは、入出力データと SDP サーバの間でデータをやり取りする機能を持つアプリケーションです。Stream Data Platform - AF が提供する API を使用して作成します。

カスタムアダプターは、処理内容によって、データ送信 AP とデータ受信 AP の 2 種類に分類できます。また、データ送信 AP とデータ受信 AP は、SDP サーバとのデータの送受信方法によって、RMI 連携カスタムアダプターとインプロセス連携カスタムアダプターの 2 種類に分類できます。

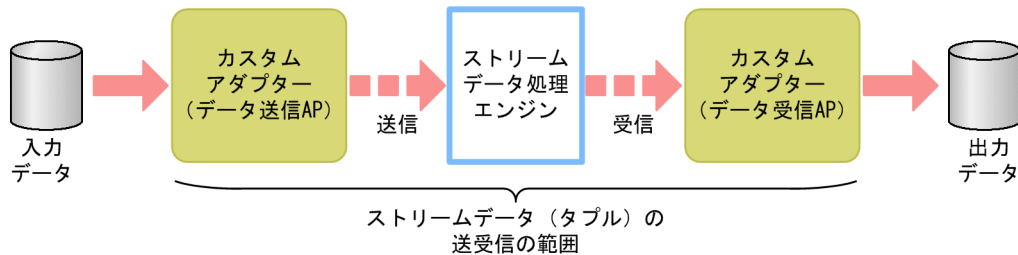
7.1.1 データ送信 AP とデータ受信 AP

カスタムアダプターは、処理内容ごとに、次の 2 種類に分類できます。

- データ送信 AP
入力データを受け付け、SDP サーバに対してストリームデータを送信します。
- データ受信 AP
SDP サーバで分析した結果（クエリ結果）のストリームデータを受信します。

カスタムアダプターの位置づけについて、次の図に示します。

図 7-1 カスタムアダプターの位置づけ



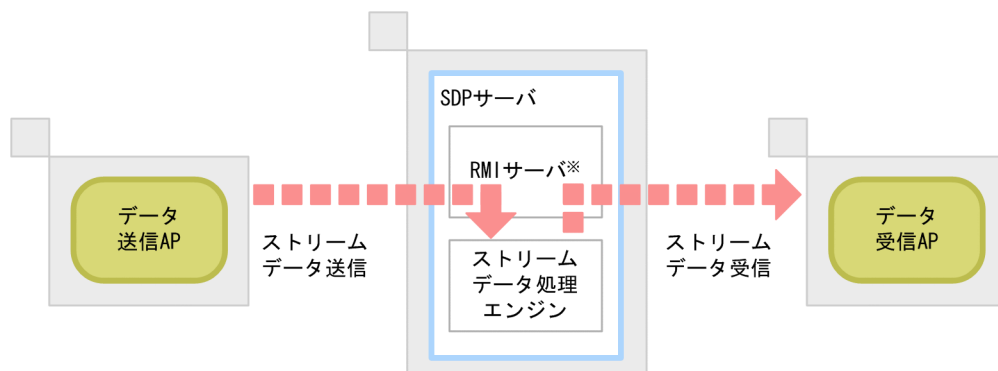
データ送信 AP では、任意の形式のデータを受け付け、ストリームデータとしてストリームデータ処理エンジンに送信します。データ受信 AP では、ストリームデータ処理エンジンで処理されたストリームデータを受信して、任意の形式のデータとして出力します。

7.1.2 データの送受信方法による分類（RMI 連携カスタムアダプターとインプロセス連携カスタムアダプター）


カスタムアダプターは、SDP サーバとのデータの送受信方法（アプリケーション連携方式）によって、次の 2 種類に分類できます。

- RMI 連携カスタムアダプター
SDP サーバとのデータの送受信に、Java の RMI の仕組みを使用します。カスタムアダプターを SDP サーバとは別のプロセスとして起動します。
RMI 連携カスタムアダプターを使用する場合のストリームデータの流れを次の図に示します。

図 7-2 RMI 連携カスタムアダプター



(凡例)

 : プロセス

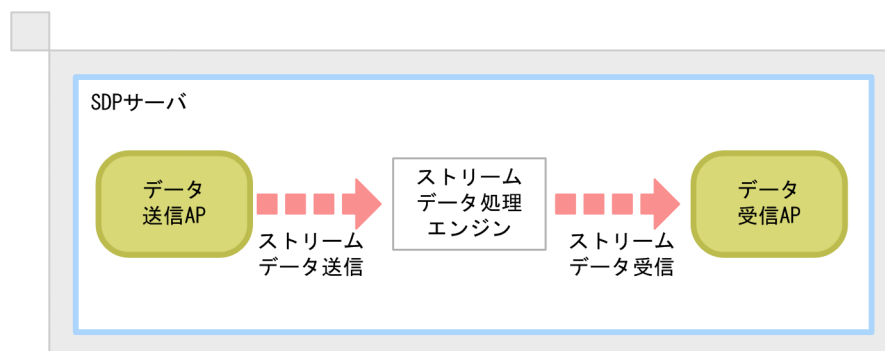
注※ RMIサーバは、RMI通信の処理を実行する機能です。SDPサーバ上で動作します。

- インプロセス連携カスタムアダプター


SDPサーバとのデータの送受信をSDPサーバのプロセス内で実行します。カスタムアダプターは、SDPサーバの一部として動作します。

インプロセス連携カスタムアダプターを使用する場合のストリームデータの流れを次の図に示します。

図 7-3 インプロセス連携カスタムアダプター



(凡例)

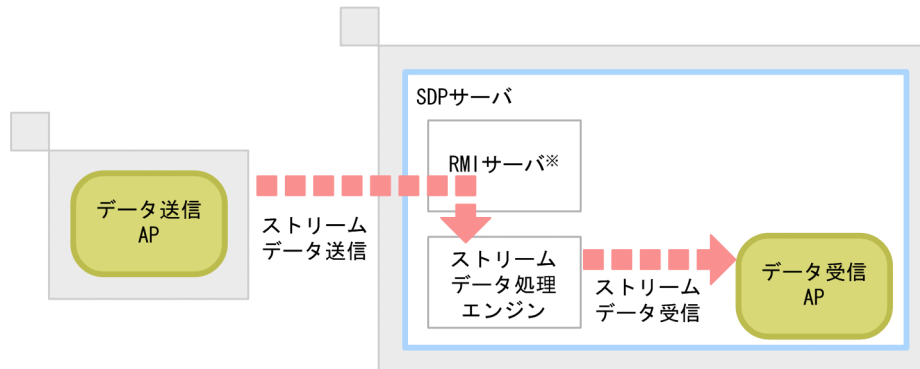
 : プロセス

なお、カスタムアダプターでは、データ送信 AP とデータ受信 AP で、異なるデータの送受信方法を使用することもできます。

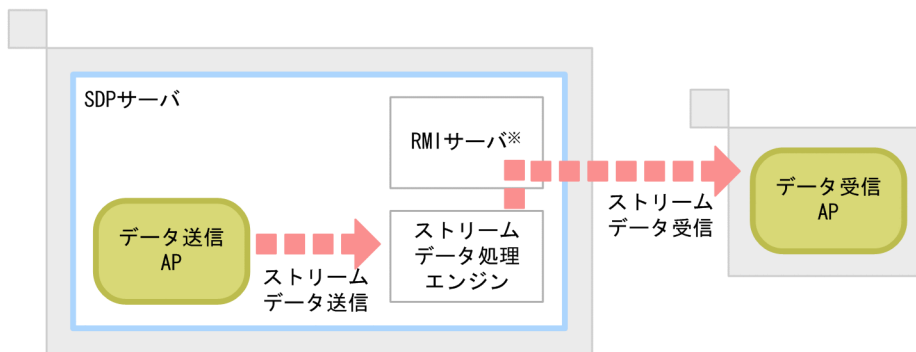
例えば、次の図に示すように、データ送信 AP を RMI 連携カスタムアダプターとして作成してデータ受信 AP をインプロセス連携カスタムアダプターとして作成したり、データ送信 AP をインプロセス連携カスタムアダプターとして作成してデータ受信 AP を RMI 連携カスタムアダプターとして作成したりすることもできます。

図 7-4 データ送信 AP とデータ受信 AP で異なるデータ送受信方法を使用する場合

- データ送信APをRMI連携カスタムアダプター、データ受信APをインプロセス連携カスタムアダプターとして作成する場合



- データ送信APをインプロセス連携カスタムアダプター、データ受信APをRMI連携カスタムアダプターとして作成する場合



(凡例)

□ : プロセス

注※ RMIサーバは、RMI通信の処理を実行する機能です。SDPサーバ上で動作します。

7.2 RMI 連携カスタムアダプターの作成

この節では、RMI 連携カスタムアダプターの作成方法について説明します。

RMI 連携カスタムアダプターは、カスタムアダプターとは別のプロセスである SDP サーバに RMI 通信で接続し、SDPConnector 型オブジェクトを取得します。SDP サーバとの接続には、SDPConnectorFactory クラスの connect メソッドを使用します。

ストリームデータの送受信には、StreamInput インタフェースおよび StreamOutput インタフェースのメソッドを使用します。StreamInput 型オブジェクトおよび StreamOutput 型オブジェクトは、SDPConnector インタフェースのメソッドによって生成します。なお、一つの SDPConnector 型オブジェクトから、StreamInput 型オブジェクトおよび StreamOutput 型オブジェクトの両方のオブジェクトを生成することもできます。

ストリームデータの送受信時に使用するインタフェースについて、概要を示します。

ストリームデータ送信時

ストリームデータの送信には、StreamInput インタフェースを使用します。

カスタムアダプターでは、SDPConnectorFactory クラスの connect メソッドによって取得した SDPConnector 型オブジェクトを使用して、StreamInput 型オブジェクトを生成します。生成した StreamInput 型オブジェクトの put メソッドを使用して、データを送信します。

ストリームデータ受信時

ストリームデータの受信には、StreamOutput インタフェースを使用します。

カスタムアダプターでは、SDPConnectorFactory クラスの connect メソッドによって取得した SDPConnector 型オブジェクトを使用して、StreamOutput 型オブジェクトを生成します。生成した StreamOutput 型オブジェクトの get メソッドまたは getAll メソッドを使用して、データを受信します。

get メソッドおよび getAll メソッドでは、SDP サーバから能動的に結果データを取得します。このデータ取得方式を、ポーリング方式といいます。

7.2.1 ストリームデータの送信 (RMI 連携カスタムアダプター)

ここでは、RMI 連携カスタムアダプターでストリームデータを送信する処理の基本的な流れについて、実装例を基に説明します。

実装例

```
public class RMI_SendSample {
    public static void main(String[] args) {
        try {
            // 1. SDPサーバに接続
            SDPConnector con = SDPConnectorFactory.connect();

            // 2. 送信対象の入カストリームに接続
            StreamInput in = con.openStreamInput("GROUP", "STREAM1");

            // 3. タプルを送信
            Object[] data=new Object[]{new Integer(1)};
            StreamTuple tuple=new StreamTuple(data);
            try {
                in.put(tuple);

                // 4. データ送信完了を通知
                in.putEnd();
            } catch (SDPClientQueryGroupStateException e) {
                System.out.println("クエリグループ停止");
            }
        }
    }
}
```

```

    }
    // 5. 入力ストリームとの接続を切断
    in.close();

    // 6. SDPサーバとの接続を切断
    con.close();
} catch (SDPClientException e) {
    System.err.println(e.getMessage());
}
}
}
}

```

実装内容の説明

それぞれの処理の意味について説明します。番号は実装例中のコメントの番号に対応しています。

- 1.SDP サーバに接続して、SDPConnector 型オブジェクト (con) を取得します。
SDPConnector con = SDPConnectorFactory.connect();
- 2.SDPConnector 型オブジェクト (con) を使用して、グループ名が"GROUP", ストリーム名が"STREAM1"である入力ストリームに接続し、StreamInput 型オブジェクト (in) を取得します。
StreamInput in = con.openStreamInput("GROUP","STREAM1");
- 3.StreamInput 型オブジェクト (in) を使用して、タプル (ストリームデータ) を送信します。
in.put(tuple);
- 4.StreamInput 型オブジェクト (in) を使用して、データ送信完了を通知します。
in.putEnd();
- 5.StreamInput 型オブジェクト (in) を使用して、入力ストリームとの接続を切断します。
in.close();
- 6.SDPConnector 型オブジェクト (con) を使用して、SDP サーバとの接続を切断します。
con.close();

7.2.2 クエリ結果データの受信 (RMI 連携カスタムアダプター)

ここでは、RMI 連携カスタムアダプターでのクエリ結果データの受信処理の基本的な流れについて、実装例を基に説明します。

実装例

```

public class RMI_ReceiveSample {
    public static void main(String[] args) {
        try {
            // 1. SDPサーバに接続
            SDPConnector con = SDPConnectorFactory.connect();

            // 2. 出力ストリームに接続
            StreamOutput o = con.openStreamOutput("GROUP","QUERY1");

            // 3. タプルを受信
            try {
                while(true) {
                    ArrayList tupleList = o.getAll();
                }
            } catch (SDPClientEndOfStreamException e) {
                System.out.println("データ受信完了");
            } catch (SDPClientQueryGroupStateException e) {
                System.out.println("クエリグループ停止");
            }

            // 4. 出力ストリームとの接続を切断
            o.close();
        }
    }
}

```

```
        // 5. SDPサーバとの接続を切断
        con.close();
    } catch (SDPClientException e) {
        System.err.println(e.getMessage());
    }
}
}
```

実装内容の説明

それぞれの処理の意味について説明します。番号は実装例中のコメントの番号に対応しています。

- 1.SDP サーバに接続して、SDPConnector 型オブジェクト (con) を取得します。
`SDPConnector con = SDPConnectorFactory.connect();`
- 2.SDPConnector 型オブジェクト (con) を使用して、クエリグループ名が"GROUP", クエリ名が "QUERY1"の出力ストリームに接続し、StreamOutput 型オブジェクト (o) を取得します。
`StreamOutput o = con.openStreamOutput("GROUP", "QUERY1");`
- 3.StreamOutput 型オブジェクト (o) を使用して、クエリ結果のタプル (tupleList) を取得します。
`ArrayList tupleList = o.getAll();`
- 4.受信が完了したら、StreamOutput 型オブジェクト (o) を使用して、出力ストリームとの接続を切断します。
`o.close();`
- 5.SDPConnector 型オブジェクト (con) を使用して、SDP サーバとの接続を切断します。
`con.close();`

7.3 インプロセス連携カスタムアダプターの作成

この節では、インプロセス連携カスタムアダプターの作成方法について説明します。

インプロセス連携カスタムアダプターの実装では、StreamInprocessUP インタフェースを実装したクラスを作成する必要があります。

インプロセス連携カスタムアダプターは、SDP サーバで生成した SDPConnector 型オブジェクトを、StreamInprocessUP インタフェースの実装クラスの実装メソッドの execute メソッドのパラメーターとして受け取ります。execute メソッドには、カスタムアダプターの main メソッドに相当する処理を記述します。

ストリームデータの送受信には、StreamInput インタフェースおよび StreamOutput インタフェースを使用します。StreamInput 型オブジェクトおよび StreamOutput 型オブジェクトは、execute メソッドのパラメーターとして渡された SDPConnector 型オブジェクトのメソッドによって生成します。

ストリームデータの送受信時に使用するインタフェースについて、概要を示します。

ストリームデータ送信時

ストリームデータの送信には、StreamInput インタフェースを使用します。

カスタムアダプターでは、execute メソッドのパラメーターとして渡された SDPConnector 型オブジェクトを使用して、StreamInput 型オブジェクトを生成します。生成した StreamInput 型オブジェクトの put メソッドを使用して、データを送信します。

ストリームデータ受信時

ストリームデータの受信には、StreamOutput インタフェースを使用します。

カスタムアダプターでは、execute メソッドのパラメーターとして渡された SDPConnector 型オブジェクトを使用して、StreamOutput 型オブジェクトを生成します。生成した StreamOutput 型オブジェクトを使用して、次のどちらかの方式で SDP サーバからクエリ結果のストリームデータ (タプル) を取得します。

- **ポーリング方式**

SDP サーバから能動的にタプルを取得する、レイテンシ重視の方式です。get メソッドまたは getAll メソッドを使用して、タプルを受信します。

- **コールバック方式**

SDP サーバ上でタプルが生成されたときに、タプルを取得するためのメソッドが SDP サーバによって呼び出される方式です。この方式を使用する場合、SDP サーバに呼び出されるメソッドは、事前に登録しておく必要があります。ポーリング方式と比較してレイテンシは増えますが、結果が生成された場合だけ処理するので CPU 効率を重視した方式です。

なお、インプロセス連携カスタムアダプターを使用する場合は、user_app.<インプロセス連携 AP 名>.properties に次の記述が必要になります。

```
user_app.classname=カスタムアダプターのメインクラス名
user_app.classpath_dir=カスタムアダプターのメインクラスのパス名
```

user_app.<インプロセス連携 AP 名>.properties の指定方法については、マニュアル「uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド」を参照してください。

7.3.1 ストリームデータの送信 (インプロセス連携カスタムアダプター)

ここでは、インプロセス連携カスタムアダプターでストリームデータを送信する処理の基本的な流れについて、実装例を基に説明します。

実装例

```
public class Inpro_SendSample implements StreamInprocessUP {
    // StreamInprocessUPの実装
    public void execute(SDPConnector con) {
        // 1. 送信対象の入カストリームに接続
        StreamInput in = con.openStreamOutput("GROUP", "STREAM1");

        // 2. タプルを送信
        Object[] data=new Object[]{new Integer(1)};
        StreamTuple tuple=new StreamTuple(data);
        in.put(tuple);

        // 3. データ送信完了を通知
        in.putEnd();

        // 4. 入カストリームとの接続を切断
        in.close();

        // 5. SDPサーバとの接続を切断
        con.close();
    }
}
```

実装内容の説明

それぞれの処理の意味について説明します。番号は実装例中のコメントの番号に対応しています。

1. execute メソッドのパラメーターとして渡された SDPConnector 型オブジェクト (con) を使用して、クエリグループ名が"GROUP", ストリーム名が"STREAM1"の入カストリームへ接続し, StreamInput 型オブジェクト (in) を取得します。

```
StreamInput in = con.openStreamInput("GROUP", "STREAM1");
```

2. StreamInput 型オブジェクト (in) を使用して, タプルを送信します。

```
in.put(tuple);
```

3. StreamInput 型オブジェクト (in) を使用して, データ送信完了を通知します。

```
in.putEnd();
```

4. StreamInput 型オブジェクト (in) を使用して, 入カストリームとの接続を切断します。

```
in.close();
```

5. SDPConnector 型オブジェクトを使用して, SDP サーバとの接続を切断します。

```
con.close();
```

7.3.2 クエリ結果データの受信 (インプロセス連携カスタムアダプター)

ここでは, インプロセス連携カスタムアダプターでのクエリ結果データの受信処理の基本的な流れについて, 実装例を基に説明します。

インプロセス連携アダプターでのクエリ結果データの受信方式には, ポーリング方式とコールバック方式の2種類があります。それぞれの方式について説明します。

(1) ポーリング方式

SDP サーバから能動的にストリームクエリ結果データを受信する方式です。

ポーリング方式によるクエリ結果データの受信処理について, 実装例を次に示します。

実装例

```
public class Inpro_ReceiveSample1 implements StreamInprocessUP {
    // StreamInprocessUPの実装
```

```

public void execute(SDPConnector con) {
    try {
        // 1. 出力ストリームに接続
        StreamOutput o = con.openStreamOutput("GROUP", "QUERY1");

        // 2. タプルを受信
        try {
            while(true) {
                ArrayList tupleList = o.getAll();
            }
        } catch (SDPClientEndOfStreamException e) {
            System.out.println("データ受信完了");
        }

        // 3. 出力ストリームとの接続を切断
        o.close();

        // 4. SDPサーバとの接続を切断
        con.close();
    } catch (SDPClientException e) {
        System.err.println(e.getMessage());
    }
}
}

```

実装内容の説明

それぞれの処理の意味について説明します。番号は実装例中のコメントの番号に対応しています。

1. execute メソッドのパラメーターで渡された SDPConnector 型オブジェクト (con) を使用して、クエリグループ名が"GROUP", クエリ名が"QUERY1"の出力ストリームに接続し、StreamOutput 型オブジェクト (o) を取得します。
StreamOutput o = con.openStreamOutput("GROUP", "QUERY1");
2. StreamOutput 型オブジェクト (o) を使用して、クエリ結果データ (タプル) を取得します。
ArrayList tupleList = o.getAll();
3. クエリ結果データの受信完了後、StreamOutput 型オブジェクト (o) を使用して、出力ストリームとの接続を切断します。
o.close();
4. SDPConnector 型オブジェクト (con) を使用して、SDP サーバとの接続を切断します。
con.close();

(2) コールバック方式

クエリ結果データ受信用のメソッドが SDP サーバによって呼び出される方式です。

コールバック方式によるクエリ結果データの受信処理について、実装例を次に示します。

コールバック方式によってクエリ結果データを受信する場合、次の二つのメソッドを実装する必要があります。

- StreamInprocessUP インタフェースを実装したクラスの execute メソッド
- StreamEventListener インタフェースを実装したコールバッククラスの onEvent メソッド

それぞれの実装例について説明します。

実装例 (execute メソッドの実装)

```

public class Inpro_ReceiveSample2 implements StreamInprocessUP {
    // StreamInprocessUPの実装
    public void execute(SDPConnector con) { // AP起動時に実行される処理
        // 1. 出力ストリームに接続

```

```

StreamOutput o = con.openStreamOutput("GROUP", "QUERY2");
// 2. コールバック用リスナーオブジェクトの生成
CallBack notifiable = new CallBack();
// 3. コールバック用リスナーオブジェクトの登録
o.registerForNotification(notifiable);
}
public void stop() { // AP停止時に実行される処理
// 4. コールバック用リスナーオブジェクトの解除
o.unregisterForNotification(notifiable);
// 5. SDPサーバとの接続を切断
con.close();
}
}
}

```

実装例 (onEvent メソッドの実装)

```

public class CallBack implements StreamEventListener {
// StreamEventListenerの実装
public void onEvent(StreamTuple tuple) {
// 6. タプル受信時の処理
System.out.println("タプル受信:" + tuple);
}
}
}

```

実装内容の説明

それぞれの処理の意味について説明します。番号は実装例中のコメントの番号に対応しています。

- execute メソッドのパラメーターに指定された SDPConnector 型オブジェクト (con) を使用して、クエリグループ名が"GROUP", クエリ名が"QUERY2"の出力ストリームに接続し、StreamOutput 型オブジェクト (o) を取得します。

```
StreamOutput o = con.openStreamOutput("GROUP", "QUERY2");
```
- StreamEventListener インタフェースの実装クラス (CallBack) のオブジェクト (notifiable) を生成します。

```
CallBack notifiable = new CallBack();
```
- StreamOutput 型オブジェクト (o) を使用して、手順 2.で作成したコールバック用リスナーオブジェクトを登録します。

```
o.registerForNotification(notifiable);
```
- カスタムアダプターを停止するときに、StreamOutput 型オブジェクト (o) を使用して、コールバック用リスナーオブジェクト (notifiable) を解除します。

```
o.unregisterForNotification(notifiable);
```
- カスタムアダプターを停止するときに、SDPConnector 型オブジェクト (con) を使用して、SDP サーバとの接続を切断します。

```
con.close();
```
- 出力ストリームにクエリ結果データのタプルが到着したときに、StreamEventListener インタフェースの onEvent メソッドに記述した内容が実行されます。

```
public void onEvent(StreamTuple tuple) {
// 6. タプル受信時の処理
}
}
```

7.3.3 注意事項

インプロセス連携カスタムアダプターの実装では、System.exit メソッドを使用しないでください。インプロセス連携カスタムアダプターは SDP サーバと同じプロセスで実行されるため、インプロセス連携カスタムアダプター内で System.exit メソッドが呼び出された場合、SDP サーバも終了してしまいます。

7.4 カスタムアダプターで実装する処理

ここでは、カスタムアダプターで実装する次の処理について説明します。「7.2 RMI 連携カスタムアダプターの作成」または「7.3 インプロセス連携カスタムアダプターの作成」で示した基本的な処理と組み合わせ、必要に応じて実装してください。

- データ受信 AP の終了契機の把握（データ処理終了の通知）
- データソースモードでの時刻情報の設定
- キューあふれの事前防止

なお、この節で説明する機能の一部は、定義ファイルでのパラメーターの指定が必要になります。各定義ファイルの詳細については、マニュアル「uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド」を参照してください。

また、処理で使用する API の詳細については、「8. データ送受信 API」を参照してください。

7.4.1 データ受信 AP の終了契機の把握（データ処理終了の通知）

データ受信 AP は、Stream Data Platform - AF のコマンドでは終了できないため、データ受信 AP の中で終了処理を実装しておく必要があります。

ここでは、データ受信 AP を終了するための契機をデータ受信 AP で把握するための実装について説明します。ここで説明する方法は、ストリームデータ処理エンジンであらかじめ送信ストリームデータの終了契機がわかる、次のような業務に使用できます。

- ファイルのデータをストリームデータとして処理する場合
- データ送信元からのデータの配信が定刻で終わることがわかっている場合

参考

データ送信 AP を終了させる場合、データ処理終了の通知は不要です。データ送信 AP については、入力データの送信がなくなり、ストリームデータ処理エンジンへのデータ送信が完了した時点で終了させることができます。

(1) データ受信 AP の終了契機の把握方法

データ受信 AP は、ストリームデータ処理エンジンでクエリが実行されている間、結果データの受信を繰り返し実行します。データ受信 AP を終了するためには、次の状態をデータ受信 AP で把握する必要があります。

- ストリームデータ処理エンジンでのクエリの実行がすべて終了していること
- 出力ストリームにクエリ結果データが残っていないこと

データ受信 AP 側では、これらの状態をデータ受信 API（get メソッドまたは getAll メソッド）呼び出し時に発生する例外 `SDPClientEndOfStreamException` または例外 `SDPClientQueryGroupStopException` をキャッチすることで把握できます。これらの例外をキャッチしたことを契機として、データ受信 AP を終了させてください。

なお、これらの例外を発生させるためには、データ送信 AP でのメソッドの呼び出し、またはコマンドの実行によって、出力ストリームに対してデータ処理終了を通知する必要があります。

それぞれの方法での通知方法について、「(2) データ送信 AP のメソッド呼び出し (putEnd メソッド) によるデータ送信終了の通知」および「(3) sdpcqlstop コマンドの実行によるクエリグループ停止の通知」で説明します。

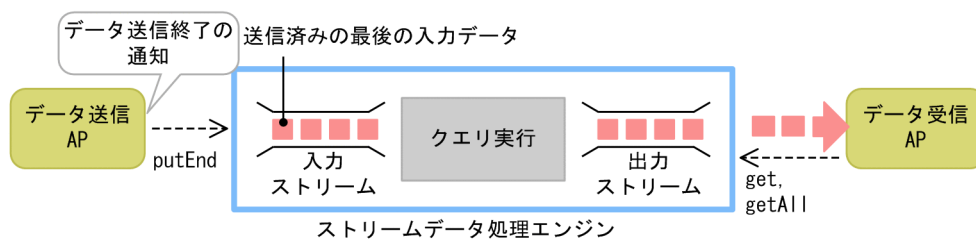
(2) データ送信 AP のメソッド呼び出し (putEnd メソッド) によるデータ送信終了の通知

データ送信 AP からのデータ送信が終了したときに、putEnd メソッドを呼び出します。出力ストリームにクエリ結果データがなくなると、データ受信 AP に対して例外 SDPClientEndOfStreamException がスローされます。

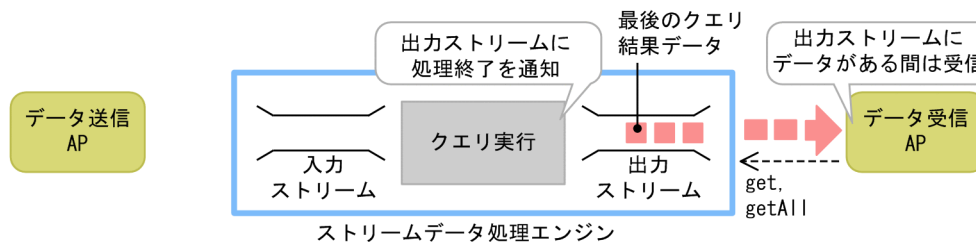
putEnd メソッドの呼び出しによってデータ送信終了を通知する流れを次の図に示します。

図 7-5 データ送信 AP によるデータ送信終了通知の流れ

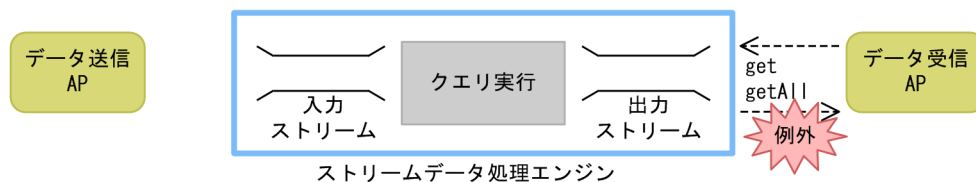
1. データ送信 AP から、putEnd メソッドによってデータ送信終了を通知します。



2. ストリームデータ処理エンジンは、出力ストリームに対して処理終了を通知します。データ受信 AP は、出力ストリームにデータがある間、データを受信します。



3. 出力ストリームにデータがなくなると、データ受信 AP での get メソッドまたは getAll メソッド呼び出し時に例外がスローされます。これによって、データ処理完了を認識できます。



(凡例)

----> : メソッドの呼び出し, およびメソッドの呼び出しによって発生する例外の流れ

図で示した流れの詳細を示します。なお、項番は、図中の番号と対応しています。

1. データ送信 AP では、ストリームデータ処理エンジンに対するストリームデータの送信がすべて終了した際、putEnd メソッドによって該当する入力ストリームに対して、データ送信終了を通知します。

2. ストリームデータ処理エンジンは、クエリグループ内のすべての入力ストリームに対して送信終了が通知されると、入力ストリーム内のすべてのタプルに対してクエリを実行して、クエリ結果データを出力ストリームに出力したあとで、すべての出力ストリームに対して処理終了を通知します。

なお、処理終了が通知されたあとでも、データ受信 AP は、出力ストリーム内にクエリ結果データがある間はデータを受信し続けます。データの受信には、`get` メソッドまたは `getAll` メソッドを使用します。

3. 出力ストリーム内のすべてのクエリ結果データの受信が完了して、出力ストリームにデータがなくなると、データ受信 AP での `get` メソッドまたは `getAll` メソッド呼び出しに対して例外 `SDPClientEndOfStreamException` がスローされます。

データ受信 AP では、この例外を契機として、データ受信 AP を停止します。

なお、クエリグループ内の入力ストリームの中に `putEnd` メソッドを呼び出していない入力ストリームがある場合、データ受信 AP からの `get` メソッドまたは `getAll` メソッド呼び出し時に例外 `SDPClientEndOfStreamException` が発生することはありません。

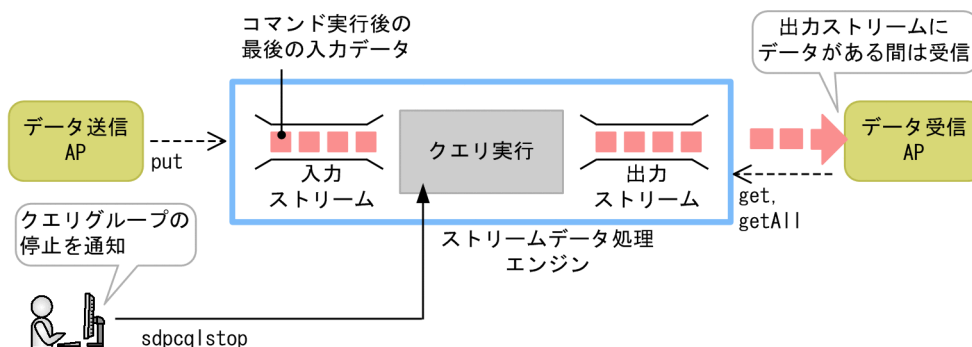
(3) `sdpcqlstop` コマンドの実行によるクエリグループ停止の通知

`sdpcqlstop` コマンドを実行して、クエリグループを停止します。出力ストリームにクエリ結果データがなくなると、データ受信 AP に対して例外 `SDPClientQueryGroupStopException` がスローされます。

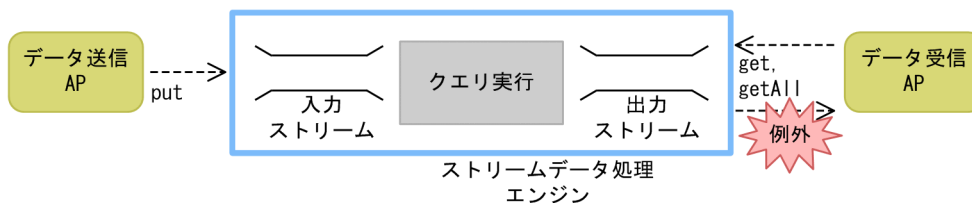
`sdpcqlstop` コマンドによるクエリグループ停止の流れを次の図に示します。

図 7-6 sdpcqlstop コマンドによるクエリグループ停止の流れ

1. sdpcqlstop コマンドによってクエリグループ停止を通知します。



2. 出力ストリームにデータがなくなると、データ受信 AP での get メソッドまたは getAll メソッド呼び出し時に例外がスローされます。これによって、データ処理完了を認識できます。



(凡例)

----> : メソッドの呼び出し, およびメソッドの呼び出しによって発生する例外の流れ

—> : コマンド実行の流れ

図で示した流れの詳細を示します。なお、項番は、図中の番号と対応しています。

1. sdpcqlstop コマンドによってクエリグループ停止を通知すると、ストリームデータ処理エンジンでは、入力ストリームデータの受け付けを停止します。その後、すでに入力ストリームに格納されているデータの処理を実行してから、処理を終了します。
なお、データ受信 AP では、コマンド実行後も出力ストリームにクエリ結果データがある間は、get メソッドまたは getAll メソッドでデータを受信します。
2. 出力ストリームにデータがなくなると、データ受信 AP での get メソッドまたは getAll メソッドの呼び出し時に例外 SDPClientQueryGroupStopException がスローされます。データ受信 AP では、この例外を契機として、データ受信 AP を停止します。

7.4.2 データソースモードでの時刻情報の設定

ストリームデータのタプルの時刻情報をデータソースモードで管理する場合、データ送信 AP で時刻情報を設定する必要があります。また、送信するタプルの時刻情報が昇順になるように、データ送信 AP で制御する必要があります。

ここでは、タプルに時刻情報を設定するためにデータ送信 AP で実装する内容について説明します。また、データ送信後のストリームデータ処理エンジンでの処理についても説明します。

なお、ここで説明する処理は、サーバモードの場合は不要な処理です。

(1) タプルへの時刻情報の設定

データソースモードでは、送信するタプルのカラムデータに時刻情報を設定する必要があります。

時刻情報を設定するためには、クエリの定義と、データ送信 AP での実装が必要です。それぞれの例を次に示します。

クエリの定義

クエリ定義ファイルに記述するストリーム定義のスキーマ指定文字列に、時刻情報を `TIMESTAMP` 型で定義します。定義例を次に示します。

```
REGISTER STREAM s1(t1 TIMESTAMP, id INT, name VARCHAR(10));
```

データ送信 AP での実装

データ送信 AP で、タプルのカラムデータに格納する時刻情報として、`java.sql.Timestamp` クラスの `Java` オブジェクトを作成します。実装例を次に示します。

```

        :
        :
//タプルオブジェクトを生成します
Object[] data = new Object[]{
    new Timestamp(System.currentTimeMillis()),
    new Integer(100),
    new String("data1")
};

//オブジェクトをタプルに設定します
StreamTuple tuple1 = new StreamTuple(data);
        :
        :

```

注

実際にデータ送信 AP を作成する場合、`new Timestamp` には、ログファイルなどに含まれるデータソースの時刻情報を抽出して指定してください。

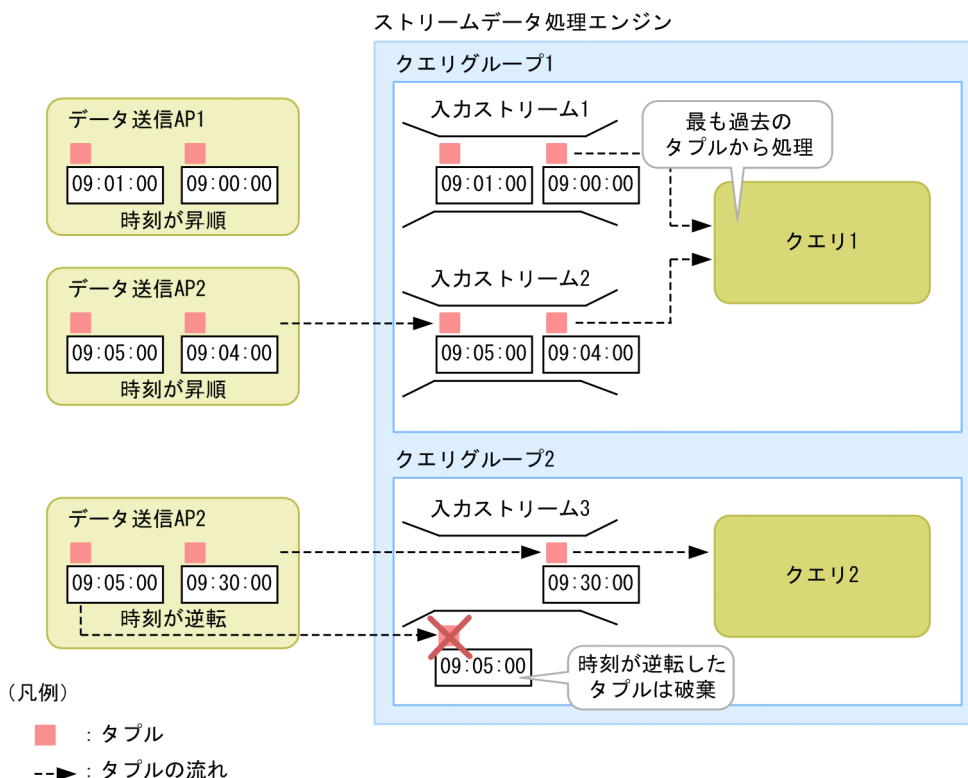
(2) タプル送信時の時刻情報の制御

データソースモードの場合、送信するタプルの時刻情報が昇順になるように、データ送信 AP で制御する必要があります。

ストリームデータ処理エンジンでは、入力ストリームごとに、時刻情報の昇順で送信されたタプルに対して処理を実行します。時刻が逆転したタプルが送信された場合、そのタプルは SDP サーバで破棄されますので注意してください。

次の図に、複数の入力ストリームに対して複数のデータ送信 AP からタプルを送信する場合の処理順序の例を示します。この図では、入力ストリーム 1 と入力ストリーム 2 は同じクエリグループに含まれています。また、図中のタプル下に示した数値（「09:00:00」など）は、そのタプルに設定された時刻情報です。

図 7-7 複数の入力ストリームに対して複数のデータ送信 AP からタプルを送信する場合の処理順序の例



一つのクエリグループ内に複数の入力ストリームがある場合、ストリームデータ処理エンジンでは、最も過去の時刻情報が設定されたタプルから処理します。図の例の場合は、入力ストリーム 1 のタプルを処理したあとで、入力ストリーム 2 のタプルを処理します。なお、クエリの実行には、putEnd メソッドで処理の終了を通知済みのストリーム以外、すべてのストリームにタプルがあることが必要です。一つでも空のストリームがある場合、最も過去のタプルがどのストリームから送信されるかが判断できないため、ストリームデータ処理エンジンは処理を実行しません。

また、時刻情報が逆転したタプルが到着した場合、ストリームデータ処理エンジンでは、そのタプルを破棄します。図の場合は、入力ストリーム 3 に到着したタプルの時刻情報の順序が逆転していたため、あとから到着したタプルは破棄します。

参考

データソースモードでデータを送信する場合、複数のデータ送信 AP から送信されたデータについて、誤差によってストリームデータ処理エンジンに到着する順序が逆転して、処理がタイムスタンプ順の昇順にならないことがあります。この場合に、順序が逆転したファイルを破棄しないで処理するためには、タイムスタンプ調整機能を使用して時刻調整を実行する必要があります。タイムスタンプ調整機能による時刻調整については、マニュアル「uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド」の、タプルのタイムスタンプの調整の検討についての説明を参照してください。

(3) タプル入力完了後の処理

データ送信 AP からのタプルの入力完了したら、ストリームデータ処理エンジンに対してデータ送信の完了を通知してください。

データソースモードの場合、ストリームデータ処理エンジンでは、タプルの時刻情報を基に、時系列に処理を実行します。データ送信 AP からのタプルの入力完了して、入力ストリームにタプルがなくなると、ストリームデータ処理エンジンの時刻は停止します。ウィンドウ演算などのクエリの処理は動作時刻が進ん

だときに実行されるため、入力ストリームにタプルがない場合、ストリームデータがストリーム処理エンジン内で滞留してしまふことがあります。

このような滞留を防ぐためには、データ送信 AP 側で、データ送信の完了を明示的に通知する必要があります。データ送信の完了が通知されると、ストリームデータ処理エンジンでは、滞留しているすべてのタプルの処理を実行して、結果を出力ストリームに出力します。

データ送信完了の通知方法については、「7.4.1 データ受信 AP の終了契機の把握（データ処理終了の通知）」を参照してください。

7.4.3 キューあふれの事前防止

ストリームデータ処理エンジンとカスタムアダプター間でタプルをやり取りする際、通路として入力ストリームのキュー（入力ストリームキュー）と出力ストリームのキュー（出力ストリームキュー）を使用します。これらのキューは、FIFO 方式でタプルを制御します。

入力ストリームキューおよび出力ストリームキューで管理できるタプルの最大値は、次の定義ファイルで指定できます。

- system_config.properties
- クエリグループ用プロパティファイル
- ストリーム用プロパティファイル

指定した値を超えた数のタプルが送信されると、キューあふれが発生します。

ここでは、入力ストリームキューまたは出力ストリームキューでキューあふれが発生した場合の動作、およびキューあふれを事前に防止するための実装方法について説明します。

(1) キューあふれが発生した場合の動作

キューあふれが発生した場合の動作は、入力ストリームキューがあふれた場合と出力ストリームキューがあふれた場合で異なります。

入力ストリームキューでキューあふれが発生した場合の動作について、次の表に示します。

表 7-1 入力ストリームキューでキューあふれが発生した場合の動作

項番	タイムスタンプモード	タイムスタンプ調整機能	動作
1	サーバモード	—	<ul style="list-style-type: none"> • データ送信 AP では、put メソッドの呼び出しに対して例外 SDPClientFreeInputQueueSizeLackException がスローされます。 • ストリームデータ処理エンジンでは、データ送信 AP から送信されたタプル数と入力ストリームキューの空きサイズを比較して入力ストリームキューがあふれる場合、送信されたタプルを一切入力ストリームキューに登録しません。ただし、クエリグループは閉塞しません。
2	データソースモード	有効	<ul style="list-style-type: none"> • データ送信 AP では、put メソッドの呼び出しに対して例外 SDPClientException がスローされます。 • ストリームデータ処理エンジンでは、クエリグループを閉塞します。また、クエリグループの閉塞に伴って、入力ストリームキューに登録されていたすべての入力タプルを

項番	タイムスタンプモード	タイムスタンプ調整機能	動作
2	データソースモード	有効	破棄します。ただし、その閉塞処理によって出力ストリームキューに登録されたタプルは破棄しません。
3		無効	<ul style="list-style-type: none"> データ送信 AP では、put メソッドの呼び出しに対して例外 SDPClientFreeInputQueueSizeLackException がスローされます。 ストリームデータ処理エンジンでは、データ送信 AP から送信されたタプル数と入力ストリームキューの空きサイズを比較して入力ストリームキューがあふれる場合、送信されたタプルを一切入力ストリームキューに登録しません。ただし、クエリグループは閉塞しません。

(凡例)

－：該当しません。

出力ストリームキューのキューあふれが発生した場合の動作について、次の表に示します。動作は、定義ファイル (system_config.properties またはクエリグループ用プロパティファイル) の querygroup.sleepOnOverStoreRetryCount パラメーターの指定によって異なります。

表 7-2 出力ストリームキューのキューあふれが発生した場合の動作

項番	定義ファイルの指定	動作
1	querygroup.sleepOnOverStoreRetryCount ≠ 0 を指定した場合	クエリグループに登録する出力タプル数と出力ストリームキューの空きサイズを比較して、出力ストリームキューがあふれる場合、クエリグループの実行を一時的に停止します (クエリを実行するスレッドをスリープさせます)。スリープ後、出力ストリームキューに出力タプルを登録した場合に出力ストリームキューがあふれたときは、クエリグループを閉塞します。また、クエリグループの閉塞に伴って、入力ストリームキューに登録されていたすべてのタプルを破棄します。ただし、その閉塞処理の際に出力ストリームキューに登録されたタプルについては、破棄しません。
2	querygroup.sleepOnOverStoreRetryCount = 0 を指定した場合	クエリグループを閉塞します。また、クエリグループの閉塞に伴って入力ストリームキューに登録されたすべてのタプルを破棄します。ただし、その閉塞処理の際に出力ストリームキューに登録されたタプルについては、破棄しません。

(2) 入力ストリームキューのキューあふれの事前防止

ここでは、データ送信 AP の実装によって、入力ストリームキューのキューあふれを事前に防止する方法について説明します。

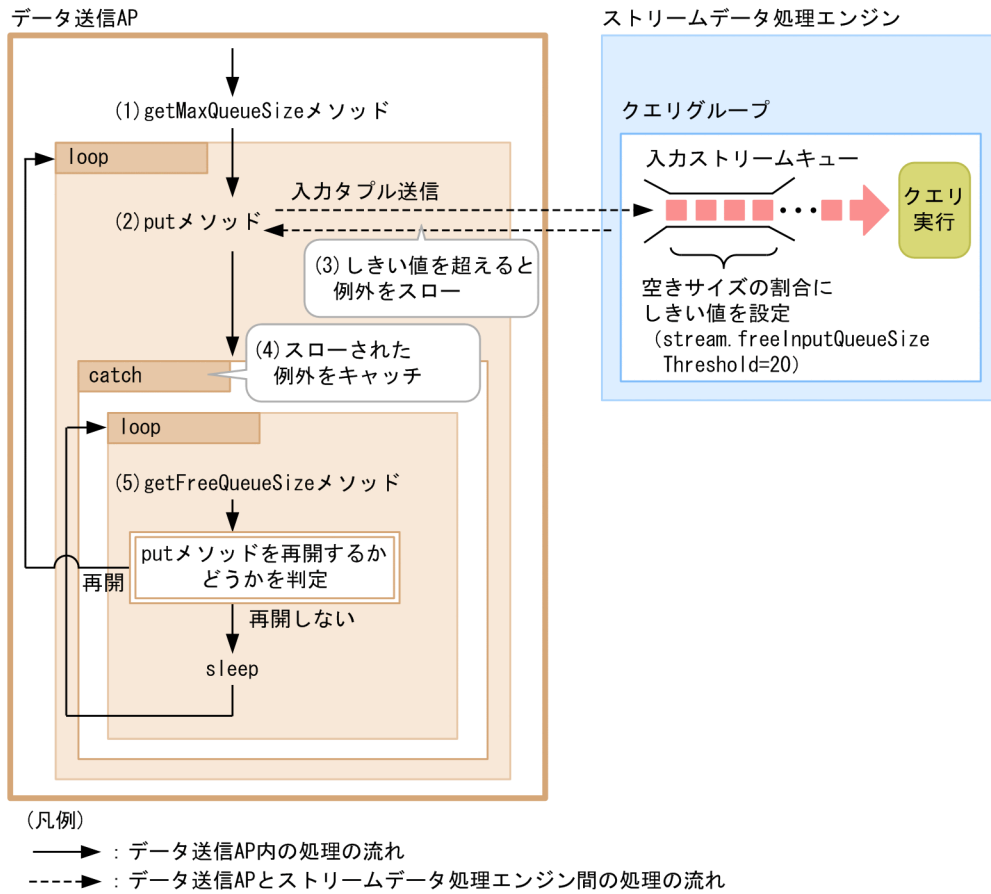
ポイント

ここで説明する処理を有効にするためには、SDP サーバの定義ファイル (system_config.properties, クエリグループ用プロパティファイル, またはストリーム用プロパティファイル) で stream.freeInputQueueSizeThreshold パラメーターを指定しておく必要があります。

なお、以降の例は、stream.freeInputQueueSizeThreshold=20 と指定した場合の例です。

キューあふれを事前に防止するデータ送信 AP の実装方法について、概要を次の図に示します。

図 7-8 入力ストリームキューのキューあふれを事前に防止するデータ送信 AP の実装方法



図で示した処理の内容について説明します。説明の番号は図中の番号と対応しています。

1. データ送信 AP で `getMaxQueueSize` メソッドを呼び出し、入力ストリームのキューの最大サイズを取得します。この値は、手順 5. で入力ストリームキューの空きサイズが全体の何割残っているかを算出するときに使用します。
2. データ送信 AP では、`put` メソッドによって、ストリームデータ処理エンジンに入力タプルを送信します。送信されたタプルは、ストリームデータ処理エンジンのクエリグループで管理されている入力ストリームキューに格納されます。
3. 入力ストリームキューの空きサイズが、定義ファイルに指定したしきい値 (`stream.freeInputQueueSizeThreshold` の指定値) の割合以下になると、ストリームデータ処理エンジンからデータ送信 AP に例外 `SDPClientFreeInputQueueSizeThresholdOverException` が通知されます。
4. データ送信 AP では、ストリームデータ処理エンジンからの通知によって `put` メソッドに対してスローされた例外 `SDPClientFreeInputQueueSizeThresholdOverException` をキャッチします。
 なお、このとき、定義ファイルに `stream.freeInputQueueSizeThresholdOutputMessage=true` を指定していた場合は、メッセージログに次のメッセージが出力されます。
`KFSP42032-W` ストリームキューの空きサイズが閾値以下になりました。ストリーム名=..., 要素数=..., 上限値=..., 閾値(%)=...
5. データ送信 AP で、`getFreeQueueSize` メソッドを呼び出して、入力ストリームキューの空きサイズを取得します。この値を基に、`put` メソッドの呼び出しを再開できるかどうかを判定します。
 再開できると判定した場合、`put` メソッドの呼び出しを再開します。

再開できないと判定した場合、一定時間スリープしたあとで、再度 `getFreeQueueSize` メソッドを呼び出して、`put` メソッドの呼び出しを再開できるか判定します。

この判定処理によって、入力ストリームキューの空きサイズが確保できるまで、`put` メソッドによるタプルの送信は抑止されます。

! 注意事項

`stream.freeInputQueueSizeThreshold` を指定しても入力ストリームキューがあふれてしまった場合、ストリームデータ処理エンジンは、データ送信 AP に対して、例外 `SDPClientFreeInputQueueSizeLackException` をスローします。この場合、例外 `SDPClientFreeInputQueueSizeThresholdOverException` はスローされません。

入力ストリームキューのキューあふれを事前に防止する実装例を次に示します。なお、実装例中の処理 1 と処理 2 は、判定方法に応じてどちらかの処理を実装してください。

実装例

```

:
try {
    streamInput = connector.openStreamInput(TARGET_QUERYGROUP, TARGET_STREAM);

    //キューの最大サイズを取得する。
    int maxQueueSize = streamInput.getMaxQueueSize();

    for (int i=0; i<1000; i++){
        Object[] data = new Object[] {
            new Integer(i),
            new String("data:"+i)
        };
        StreamTuple tuple = new StreamTuple(data);

        try {
            streamInput.put(tuple);
        } catch (SDPClientFreeInputQueueSizeThresholdOverException sce) {
            //キューの空きサイズが20%以下になった。
            // (stream.freeInputQueueSizeThreshold=20指定時)。
            while (true) {
                int freeQueueSize = streamInput.getFreeQueueSize();

                //処理1: キューの空きサイズを確認してputメソッドを再開する場合。
                //キューの空きサイズが500以上になったら、putメソッドを再開する。
                if (freeQueueSize >= 500) {
                    break;
                } else {
                    Thread.sleep(100);
                }
                //処理1はここまで。

                //処理2: キューの空きサイズと最大サイズの割合を確認して
                //putメソッドを再開する場合。
                double freePerMax = ((double)freeQueueSize / (double)maxQueueSize) * 100;
                //キューの空きサイズが50%以上になったら、putメソッドを再開する。
                if (freePerMax >= 50) {
                    break;
                } else {
                    Thread.sleep(100);
                }
                //処理2はここまで。
            }
        } catch (SDPClientFreeInputQueueSizeLackException sce2) {
            //キュー空きサイズがない。
            //(putメソッドで送信したタプルはストリームデータ処理エンジンに
            //受け付けられなかった)。
            //このため、タプルを再送する前にスリープする。
            Thread.sleep(100);
            try {
                //タプルを再送する。
            }
        }
    }
}

```

```

        streamInput.put(tuple);
    } catch (SDPClientFreeInputQueueSizeThresholdOverException sce3) {
        //タプルの再送が成功したので、ここでのしきい値例外は無視する。
    }
}
}
streamInput.putEnd();
} catch (SDPClientFreeInputQueueSizeLackException sce4) {
    //キューの空きサイズがない。
    //(putメソッドで送信したタプルはストリームデータ処理エンジンに
    //受け付けられなかった)。
    : (省略)
    //送信を停止する。
    streamInput.putEnd();
    : (省略)
} catch (...) {
    //その他の例外をcatchしたときの処理を実装する。
    : (省略)
}

```

(3) 出力ストリームキューのキューあふれの事前防止

出力ストリームキューのキューあふれは、定義ファイルの `querygroup.sleepOnOverStoreRetryCount` パラメーターおよび `querygroup.sleepOnOverStore` パラメーターの指定で防止します。これらのパラメーターの指定によって、出力ストリームキューがあふれそうな場合に、クエリグループの実行が一時停止（スリープ）されます。

また、「(2) 入力ストリームキューのキューあふれの事前防止」で示した実装をしておくことで、クエリグループの実行が一時停止した場合に、入力ストリームキューに対する入力タプルの送信も抑止できるため、キューをあふれさせることなく、システムの処理を正常に戻すことができます。

出力ストリームキューのキューあふれが事前に防止される処理の概要について、次の図に示します。データ送信 AP の実装は、「(2) 入力ストリームキューのキューあふれの事前防止」と同じです。

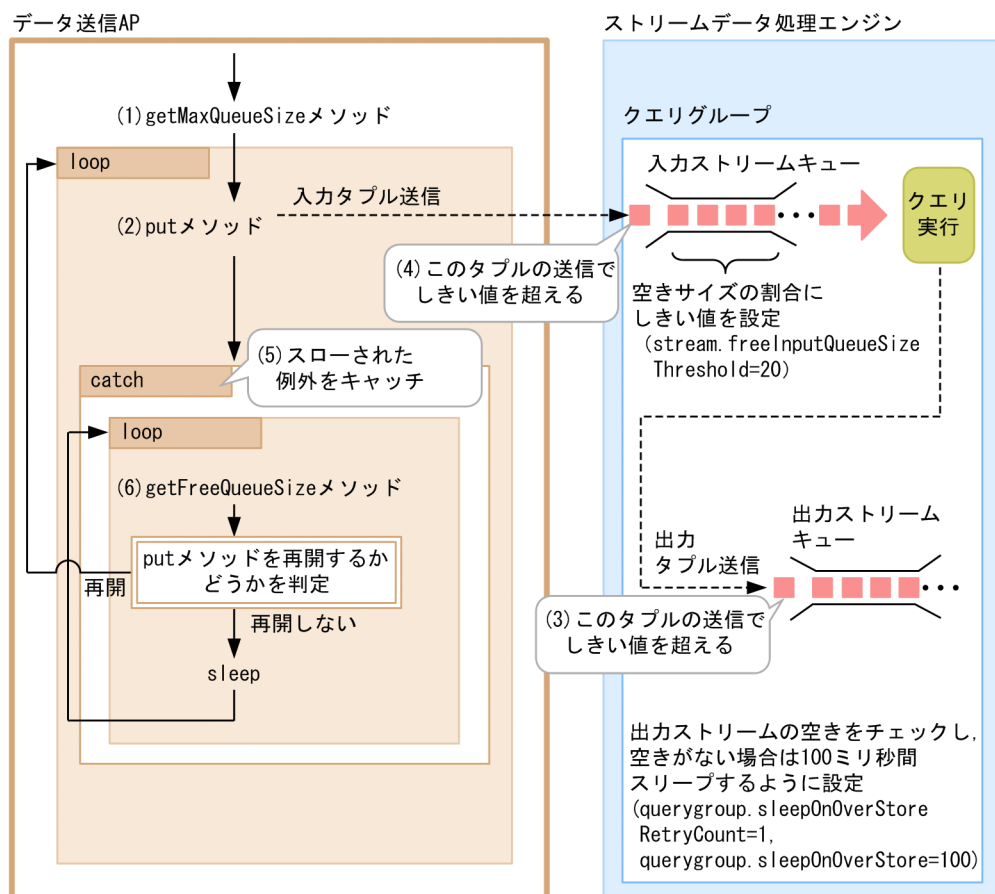
ポイント

ここで説明する処理を有効にするためには、SDP サーバの定義ファイルで次のパラメーターを指定しておく必要があります。

- `stream.freeInputQueueSizeThreshold`
- `querygroup.sleepOnOverStoreRetryCount`
- `querygroup.sleepOnOverStore`

なお、この例は、それぞれ、`stream.freeInputQueueSizeThreshold=20`、`querygroup.sleepOnOverStoreRetryCount=1`、`querygroup.sleepOnOverStore=100` と指定した場合の例です。

図 7-9 出力ストリームキューのキューあふれが事前に防止される処理の概要



(凡例)

—▶: データ送信AP内の処理の流れ

- - -▶: データ送信APとストリームデータ処理エンジン間の処理の流れ

図で示した処理の内容について説明します。説明の番号は図中の番号と対応しています。

- データ送信 AP で `getMaxQueueSize` メソッドを呼び出し、入力ストリームのキューの最大サイズを取得します。この値は、入力ストリームキューの空きサイズが全体の何割残っているかを算出するときに使用します。
- データ送信 AP では、`put` メソッドによって、ストリームデータ処理エンジンにタプルを送信します。送信されたタプルは、ストリームデータ処理エンジンのクエリグループで管理されている入力ストリームキューに格納されます。
- ストリームデータ処理エンジンでは、クエリ実行によって送信された出力タプルによって出力ストリームキューがあふれることを検知すると、クエリグループの実行を一時停止（スリープ）します。スリープするかどうかの設定、およびスリープさせる時間については、`querygroup.sleepOnOverStoreRetryCount` パラメーターおよび `querygroup.sleepOnOverStore` パラメーターの指定値に従います。
クエリグループがスリープすると、ストリームデータ処理エンジンは、メッセージログに次のメッセージを出力します。
KFSP42033-W ストリームキューへのタプルの登録をリトライします。クエリグループ名=…、ストリーム名=…、リトライまでのスリープ時間=…、リトライ回数=…
なお、このとき、スリープ中のクエリグループのステータスは「実行中」となっています。データ送信 AP による入力ストリームキューに対するタプルの送信、およびデータ受信 AP による出力ストリームキューからのタプルの取得は許可されています。
- クエリグループの実行がスリープしている間、データ送信 AP が `put` メソッドを呼び出し続けると、入力ストリームキューに処理されていないタプルが蓄積されていきます。入力ストリームキューの空きサイズが、

定義ファイルに指定したしきい値（`stream.freeInputQueueSizeThreshold` パラメーターの指定値）の割合以下になると、ストリームデータ処理エンジンからデータ送信 AP に例外 `SDPClientFreeInputQueueSizeThresholdOverException` が通知されます。

5. データ送信 AP では、ストリームデータ処理エンジンからの通知によって `put` メソッドに対してスローされた例外 `SDPClientFreeInputQueueSizeThresholdOverException` をキャッチします。

6. データ送信 AP で、`getFreeQueueSize` メソッドを呼び出して、入力ストリームキューの空きサイズを取得します。この値を基に、`put` メソッドの呼び出しを再開できるかどうかを判定します。

再開できると判定した場合、`put` メソッドの呼び出しを再開します。

再開できないと判定した場合、一定時間スリープしたあとで、再度 `getFreeQueueSize` メソッドを呼び出して、`put` メソッドの呼び出しを再開できるか判定します。

この処理によって、入力ストリームキューの空きサイズが確保できるまで、`put` メソッドによる入力タプルの送信は抑止されます。出力ストリームキューの空きサイズが確保され、クエリグループの実行が再開されて入力ストリームキューのタプルが処理されると、再度、入力タプルの送信ができるようになり、システムが正常な状態に戻ります。

7.5 コンパイル手順

カスタムアダプターのコンパイルは、次の手順で実行してください。

1. 準備として、環境変数を定義します。

Stream Data Platform - AF のセットアップ実行時に設定した運用ユーザーの環境変数 PATH に、次のパスを追加してください。この手順は、2 回目以降のコンパイルでは不要な手順です。

```
<インストールディレクトリ>%psb%jdk%bin
```

2. コンパイルを実行します。

次のコマンドを実行してください。

```
javac -cp <インストールディレクトリ>%lib%sdp.jar Javaソースプログラム
```

3. jar ファイルを作成します (インプロセス連携カスタムアダプターの場合)。

インプロセス連携カスタムアダプターの場合、コンパイルして生成されたクラスファイルをまとめた jar ファイルを作成してください。

次に、test ディレクトリ下のクラスファイル [A.class] と [B.class] を [test.jar] というファイルにまとめる場合のコマンドの実行例を示します。

```
jar cf test.jar test%A.class test%B.class
```

7.6 カスタムアダプター作成時の留意事項

ここでは、カスタムアダプター作成時の留意事項について説明します。

カスタムアダプターのデータ送信 AP およびデータ受信 AP は、次の方針で作成してください。

- 同一ストリームに対して複数回接続しないでください。
一つのストリームは、それぞれ 1 回だけ接続するようにしてください。
一つの SDPConnector 型オブジェクトでは、同一のストリームに対して複数回接続できませんが、複数の SDPConnector 型オブジェクトを使用すると、同一ストリームに複数回接続できます。しかし、複数回接続した場合、ストリームデータ処理エンジンのスループットが下がります。
- データ受信 AP の処理負荷に対して、クエリ結果データの出力ストリームへの出力頻度が高い場合は、次のどちらかの方法を使用して、データ受信 AP の通信コストを抑えてください。
 - 複数のクエリ結果のポーリング方式での受信を getAll メソッドで一括受信してください。これによって、データ転送の通信コストを抑えられます。
 - データ受信 AP を SDP サーバと同一プロセスで動作するように、インプロセス連携カスタムアダプターとして作成してください。これによって、プロセス間通信に掛かる通信コストを抑えられます。

通信コストを抑えることで、出力ストリームキューのタプルの滞留を防止できるため、キューあふれ防止にもなります。

- クエリ結果の出力頻度が小さい場合は、データ受信 AP を SDP サーバと同一プロセスで動作するインプロセス連携カスタムアダプターとして作成して、データ受信をコールバック受信方式にすることをお勧めします。これによって、システムの CPU 利用率を低減できます。
- RMI 連携カスタムアダプターを作成する場合、main メソッドは次のように宣言してください。
`public static void main(String[])`
RMI 連携カスタムアダプター起動時に実行するコマンド (sdpstartap) では、メインクラスの main メソッドが実行されます。
- RMI 連携カスタムアダプターを作成する場合、アプリケーションの終了コードとして 1 は使用しないでください。

8

データ送受信 API

この章では、カスタムアダプターの作成で使用するデータ送受信 API の文法について説明します。

8.1 データ送受信 API の記述形式

ここでは、データ送受信 API の記述形式について説明します。

各 API で説明する項目は次のとおりです。ただし、API によっては説明しない項目もあります。

形式

API の記述形式を示します。

説明

API の機能について説明します。

パラメーター

API のパラメーターについて説明します。

例外

API を利用する際に発生する例外について説明します。

戻り値

API の戻り値について説明します。

注意事項

API を利用する上での注意事項について説明します。

8.2 データ送受信 API の一覧

データ送受信 API は、次の 3 種類に分類できます。作成するカスタムアダプターに合わせて使用してください。

- 共通 API
RMI 連携カスタムアダプター作成時、およびインプロセス連携カスタムアダプター作成時の両方で使用する API です。
- RMI 連携用 API
RMI 連携カスタムアダプター作成時に使用する API です。
- インプロセス連携用 API
インプロセス連携カスタムアダプター作成時に使用する API です。

データ送受信 API のインタフェースおよびクラスの一覧を示します。

表 8-1 データ送受信 API 一覧 (インタフェース)

項番	パッケージ名	インタフェース名	機能	種類
1	jp.co.Hitachi.soft.sdp.api	SDPConnector	SDP サーバとカスタムアダプターを結ぶコネクタのインタフェースです。入力ストリームへ接続する場合、または出力ストリームへ接続する場合に使用します。	共通 API
2		StreamInput	SDP サーバにストリームデータを送信する場合、および入力ストリームとの接続を切断する場合に使用するインタフェースです。	共通 API
3		StreamOutput	SDP サーバからのクエリ結果データの受信、コールバック用リスナーオブジェクトの登録・解除、および出力ストリームとの接続を閉じるために使用するインタフェースです。	共通 API
4	jp.co.Hitachi.soft.sdp.api.inprocess	StreamEventListener	コールバックされるメソッドの処理を実装するためのインタフェースです。	インプロセス連携用 API
5		StreamInprocessUP	カスタムアダプターの main メソッドに相当するメイン処理を実装するためのインタフェースです。SDP サーバ起動時に StreamInprocessUP インタフェースの実装クラスがロードされ、実行されます。	インプロセス連携用 API

表 8-2 データ送受信 API 一覧 (クラス)

項番	パッケージ名	クラス名	機能	種類
1	jp.co.Hitachi.soft.sdp.api	SDPConnectorFactory	SDPConnector 型オブジェクトを生成するためのクラスです。	RMI 連携用 API
2	jp.co.Hitachi.soft.sdp.common.util	StreamTime	日付・時刻指定処理のためのクラスです。	共通 API
3	jp.co.Hitachi.soft.sdp.common.data	StreamTuple	ストリームタプルを表現するクラスです。	共通 API

以降では、インタフェースおよびクラスをアルファベット順に説明します。

8.3 SDPConnector インタフェース (共通 API)

説明

SDP サーバとの接続で使用するコネクタとしての機能を持つインタフェースです。コネクタは、SDPConnector 型オブジェクトとして生成されます。

SDP サーバに登録されているクエリグループの入力ストリームまたは出力ストリームには、コネクタを使用して接続します。

ストリームには、SDPConnector インタフェースの openStreamInput メソッドまたは openStreamOutput メソッドを使用して接続します。ただし、一つのコネクタで同一名称のストリームに複数回接続することはできません。

コネクタは、SDP サーバとカスタムアダプター間のデータ連携の方法ごとに、次の処理によって生成されます。

- SDP サーバと RMI で連携するコネクタの生成

SDPConnectorFactory クラスの connect メソッドを呼び出すことで、SDP サーバと RMI で連携したコネクタが生成されます。

- SDP サーバとインプロセスで連携するコネクタの生成

sdpstartinpro コマンド実行時に、SDP サーバとインプロセスで連携したコネクタが生成されます。インプロセス連携カスタムアダプターでは、生成された SDPConnector 型オブジェクトを StreamInprocessUP インタフェースの execute メソッドのパラメーターとして受け取ります。

フィールド

なし。

コンストラクター

なし。

メソッド

SDPConnector インタフェースのメソッド一覧を次の表に示します。

戻り値	メソッド名	機能
void	close()	コネクタを閉じます。
boolean	isClosed()	コネクタが閉じているかどうかを確認します。
StreamInput	openStreamInput(String group_name,String stream_name)	パラメーターに指定したグループ名およびストリーム名に対応する入力ストリームに接続します。 同一の SDPConnector 型オブジェクトでは、同一名称のストリームに複数回接続できません。
StreamOutput	openStreamOutput(String group_name,String stream_name)	パラメーターに指定したグループ名およびストリーム名に対応する出力ストリームへ接続します。同一の SDPConnector 型オブジェクトでは、同一名称のストリームに複数回接続できません。

注意事項

なし。

close()メソッド

形式`void close()`**説明**

コネクタを閉じます。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientException	コネクタがすでに閉じていた場合

戻り値

なし。

isClosed()メソッド

形式`boolean isClosed()`**説明**

コネクタが閉じているかどうかを確認します。

パラメーター

なし。

例外

なし。

戻り値`true`

コネクタは閉じています。

false

コネクタは開いています。

openStreamInput(String group_name,String stream_name)メソッド

形式

```
StreamInput openStreamInput(String group_name,String stream_name)
```

説明

パラメーターに指定したグループ名およびストリーム名に対応する入力ストリームに接続します。同一の SDPConnector 型オブジェクトでは、同一名称のストリームに複数回接続できません。

パラメーター

group_name

クエリグループ名を指定します。

stream_name

ストリーム名を指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターに null を指定した場合
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientException	<ul style="list-style-type: none"> 存在しないグループ名またはストリーム名を指定した場合 同じ SDPConnector 型オブジェクトを使用して、同一の入力ストリームとすでに接続している場合 入力ストリームではないストリームを指定した場合 コネクタがすでに閉じている場合

戻り値

接続した入力ストリーム (StreamInput 型オブジェクト)。

openStreamOutput(String group_name,String stream_name)メソッド

形式

```
StreamOutput openStreamOutput(String group_name,String stream_name)
```

説明

パラメーターに指定したグループ名およびストリーム名に対応する出力ストリームに接続します。同一の SDPConnector 型オブジェクトでは、同一名称のストリームに複数回接続できません。

パラメーター

group_name

クエリグループ名を指定します。

stream_name

ストリーム名を指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターに null を指定した場合
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientException	<ul style="list-style-type: none"> 存在しないグループ名またはストリーム名を指定した場合 同じ SDPConnector 型オブジェクトを使用して、同一の出力ストリームにすでに接続をしている場合 コネクタがすでに閉じている場合
SDPClientRelationStateException	指定したストリームがリレーション状態である場合

戻り値

接続した出力ストリーム (StreamOutput 型オブジェクト)。

8.4 SDPConnectorFactory クラス (RMI 連携用)

クラス階層

jp.co.Hitachi.soft.sdp.api.SDPConnectorFactory

説明

SDPConnector 型オブジェクトの生成処理を実装するファクトリクラスです。このクラスの connect メソッドを実行すると、SDP サーバと RMI 通信で接続した状態の SDPConnector 型オブジェクトが生成されます。

メソッド

SDPConnectorFactory クラスのメソッド一覧を次の表に示します。

戻り値	メソッド名	機能
SDPConnector	connect()	SDP サーバと RMI 通信で接続した状態の SDPConnector 型オブジェクトを生成します。

注意事項

connect メソッドは static メソッドです。

connect()メソッド

形式

SDPConnector connect()

説明

SDP サーバと RMI 通信で接続した状態の SDPConnector 型オブジェクトを生成します。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientException	system_config.properties の解析に失敗した場合

戻り値

SDP サーバとのコネクター (SDPConnector 型オブジェクト)。

8.5 StreamEventListener インタフェース (インプロセス連携用)

説明

SDP サーバ上で、タプルが生成されたときにコールバックされるメソッドの処理を実装するインタフェースです。

カスタムアダプターでは、StreamEventListener インタフェースを実装したクラスを作成して、onEvent メソッドにコールバック時の処理を記述します。

StreamOutput インタフェースの registerForNotification メソッドを使用して、StreamEventListener インタフェースを実装したクラスのオブジェクトを登録すると、以降、SDP サーバ上でタプルが生成されたときに、登録したオブジェクトの onEvent メソッドがコールバックされます。

メソッド

StreamEventListener インタフェースのメソッド一覧を次の表に示します。

戻り値	メソッド名	機能
void	onEvent(StreamTuple tuple)	SDP サーバ上でタプルが生成されたときにコールバックされるメソッドです。

注意事項

StreamEventListener インタフェースは java.rmi.Remote クラスを継承しています。

onEvent(StreamTuple tuple)メソッド

形式

```
void onEvent(StreamTuple tuple)
```

説明

SDP サーバ上でタプルが生成されたときにコールバックされるメソッドです。

パラメーター

tuple

生成されたタプルが指定されます。

例外

なし。

戻り値

なし。

8.6 StreamInprocessUP インタフェース (インプロセス連携用)

説明

カスタムアダプターが SDP サーバとインプロセスで連携するためには、StreamInprocessUP インタフェースを実装するクラスを作成する必要があります。

StreamInprocessUP インタフェースには、execute メソッドと stop メソッドがあります。

execute メソッドには、カスタムアダプターの main メソッドに相当するメイン処理を記述します。stop メソッドには、カスタムアダプターの終了処理を記述します。

sdpstartinpro コマンドを実行してインプロセス連携カスタムアダプターを起動すると、SDP サーバによって StreamInprocessUP インタフェースを実装したクラスがロードされ、execute メソッドを実行するためのスレッド (カスタムアダプター実行スレッド) が生成されます。execute メソッドはこのスレッドから呼び出されるメソッドです。なお、SDP サーバによってロードされる実装クラス名、およびその実装クラスが格納された jar ファイルのパス名は、プロパティファイル user_app.<インプロセス連携カスタムアダプター名>.properties に記述します。プロパティファイルについては、マニュアル「uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド」を参照してください。

stop メソッドは、sdpstopinpro コマンドを実行してインプロセス連携カスタムアダプターを停止するときに、SDP サーバ側のスレッドから呼び出されます。

メソッド

StreamInprocessUP インタフェースのメソッド一覧を次の表に示します。

戻り値	メソッド名	機能
void	execute(SDPConnector con)	カスタムアダプターのメイン処理を記述します。 このメソッドは、sdpstartinpro コマンド実行時に、SDP サーバが生成したカスタムアダプター実行スレッドによって呼び出されます。
void	stop()	カスタムアダプターの終了処理を記述します。 このメソッドは、sdpstopinpro コマンド実行時に、SDP サーバ側のスレッドによって呼び出されます。

注意事項

- StreamInprocessUP インタフェースの execute メソッドで起動したユーザースレッドは、stop メソッドの中で必ず停止してください。stop メソッドに停止処理を記述しなかった場合、または stop メソッドで停止できない無限ループなどの処理がユーザースレッドに含まれる場合、ユーザースレッドが停止されないで残ることがあります。
- StreamInprocessUP インタフェースの execute メソッドに無限ループなどの処理を記述した場合、execute メソッド自身の実行スレッドが残ってしまうことがあります。
- カスタムアダプターに任意の引数を渡す場合は、String 型の可変長引数を持つコンストラクターを実装してください。リフレクション経由で作成した String 型の可変長引数を持つコンストラクターが呼ばれます。その引数として、sdpstartinpro コマンドで定義した任意の引数が渡されます。なお、String 型の可変長引数を持つコンストラクターを実装していない場合は、デフォルトコンストラクターが呼ばれます (ただし、引数は渡されません)。

- カスタムアダプターに任意の引数を渡さない場合は、String 型の可変長引数を持つコンストラクターまたはデフォルトコンストラクターのどちらかを実装してください。どちらも実装している場合は、String 型の可変長引数を持つコンストラクターが呼ばれます。
- String 型の可変長引数を持つコンストラクターまたはデフォルトコンストラクターのどちらも実装していない場合は、エラーとなり、カスタムアダプターは起動しません。

execute(SDPConnector con)メソッド

形式

```
void execute(SDPConnector con)
```

説明

カスタムアダプターのメイン処理を記述します。

このメソッドは、sdpstartinpro コマンド実行時に、SDP サーバが生成したカスタムアダプター実行スレッドによって呼び出されます。

パラメーター

con

SDPConnector 型オブジェクトを指定します。

例外

なし。

戻り値

なし。

stop()メソッド

形式

```
void stop()
```

説明

カスタムアダプターの終了処理を記述します。

このメソッドは、sdpstopinpro コマンド実行時に、SDP サーバ側のスレッドによって呼び出されます。

パラメーター

なし。

例外

なし。

戻り値

なし。

8.7 StreamInput インタフェース (共通 API)

説明

カスタムアダプターが SDP サーバに対してタプルを送信するために使用するインタフェースです。

タプルを送信するには、put メソッドを使用します。put メソッドには、複数のタプルをまとめて送信するメソッドと、単一のタプルを送信するメソッドの 2 種類があります。

メソッド

StreamInput インタフェースのメソッド一覧を次の表に示します。

戻り値	メソッド名	機能
void	close()	入力ストリームとの接続を閉じます。
int	getFreeQueueSize()	入力ストリームキューの空きサイズを取得します。
int	getMaxQueueSize()	入力ストリームキューの最大サイズを取得します。
boolean	isStarted()	データソースモードの場合に、入力ストリームが新たなストリームデータの受付を開始しているかどうかを確認します。
void	put(ArrayList<StreamTuple> tuple_list)	複数のタプルを送信します。
void	put(StreamTuple tuple)	単一のタプルを送信します。
void	putEnd()	入力ストリームに対して、ストリームデータ入力終了したことを通知します。 このメソッドによって入力完了を通知したあと、クエリグループの再開前に put メソッドや putEnd メソッドを実行した場合は、例外が返されます。

注意事項

なし。

close()メソッド

形式

```
void close()
```

説明

入力ストリームとの接続を閉じます。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	入力ストリームがすでに閉じていた場合

戻り値

なし。

getFreeQueueSize()メソッド

形式

```
int getFreeQueueSize()
```

説明

入力ストリームキューの空きサイズを取得します。

入力ストリームキューの空きサイズは、system_config.properties の engine.maxQueueSize パラメーターに指定した値から、使用中のサイズを引いた値です。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientException	入力ストリームがすでに閉じている場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合 (SDPClientQueryGroupStateException の詳細例外)
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合
SDPClientQueryGroupStopException	クエリグループが停止中の場合 (SDPClientQueryGroupStateException の詳細例外)

戻り値

入力ストリームキューの空きサイズ (int)。

getMaxQueueSize()メソッド

形式

```
int getMaxQueueSize()
```

説明

入力ストリームキューの最大サイズを取得します。

入力ストリームキューの最大サイズは、`system_config.properties` の `engine.maxQueueSize` パラメーターに指定した値です。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
<code>SDPClientException</code>	入力ストリームがすでに閉じている場合
<code>SDPClientCommunicationException</code>	RMI 連携時に通信例外が発生した場合

戻り値

入力ストリームキューの最大サイズ (int)。

isStarted() メソッド

形式

```
boolean isStarted()
```

説明

データソースモードの場合に、入力ストリームが新たなストリームデータの受付を開始しているかどうかを確認します。

パラメーター

なし。

例外

例外の一覧を次の表に示します。

例外	発生条件
<code>SDPClientCommunicationException</code>	RMI 連携時に通信例外が発生した場合
<code>SDPClientException</code>	<ul style="list-style-type: none"> 入力ストリームがすでに閉じている場合 サーバモードで動作中のストリームの場合
<code>SDPClientQueryGroupHoldException</code>	クエリグループが閉塞中の場合
<code>SDPClientQueryGroupNotExistException</code>	クエリグループが削除された場合
<code>SDPClientQueryGroupStateException</code>	クエリグループが実行中ではない場合

例外	発生条件
SDPClientQueryGroupStopException	クエリグループが停止中の場合

戻り値

true

ストリームデータの受付を開始しています。

false

ストリームデータの受付を開始していません。

put(ArrayList<StreamTuple> tuple_list)メソッド

形式

```
void put(ArrayList<StreamTuple> tuple_list)
```

説明

複数のタプルを送信します。

パラメーター

tuple_list

送信するタプル (StreamTuple 型オブジェクト) のリストを指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
ClassCastException	パラメーターのリスト内に StreamTuple 以外の要素が含まれていた場合
NullPointerException	<ul style="list-style-type: none"> パラメーターに null または null を含むリストを指定した場合 リストの要素であるデータオブジェクト配列内に null を含む StreamTuple 型オブジェクトを指定した場合
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientException	<ul style="list-style-type: none"> タプルのリストに含まれる要素のデータ型がストリーム定義と異なる場合 入力ストリームがすでに閉じていた場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中ではない場合
SDPClientQueryGroupStopException	クエリグループが停止中の場合

戻り値

なし。

put(StreamTuple tuple)メソッド

形式

```
void put(StreamTuple tuple)
```

説明

単一のタプルを送信します。

パラメーター

tuple

送信するタプル (StreamTuple 型オブジェクト) を指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	<ul style="list-style-type: none"> パラメーターに null を指定した場合 データオブジェクト配列内に null を含む StreamTuple 型オブジェクトを指定した場合
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientException	<ul style="list-style-type: none"> タプルの要素のデータ型がストリーム定義と異なる場合 入力ストリームがすでに閉じていた場合
SDPClientFreeInputQueueSizeLackException	入力ストリームキューの空きサイズが不足している場合 (この場合, 入力ストリームキューにタプルは投入されません)
SDPClientFreeInputQueueSizeThresholdOverException	入力ストリームキューの空きサイズが stream.freeInputQueueSizeThreshold パラメーターで指定したしきい値以下になった場合 (この場合, 入力ストリームキューにタプルは投入されます)
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中ではない場合
SDPClientQueryGroupStopException	クエリグループが停止中の場合

戻り値

なし。

putEnd()メソッド

形式

```
void putEnd()
```

説明

入力ストリームに対して、ストリームデータの入力が終了したことを通知します。

このメソッドによって入力完了を通知したあと、クエリグループの再開前に put メソッドや putEnd メソッドを実行した場合は、例外が返されます。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientException	<ul style="list-style-type: none"> 入力ストリームがすでに閉じている場合 入力ストリームがすでにストリームデータの受付を停止している場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合
SDPClientQueryGroupStopException	クエリグループが停止中の場合

戻り値

なし。

8.8 StreamOutput インタフェース (共通 API)

説明

カスタムアダプターが SDP サーバからタプルを受信するために使用するインタフェースです。

タプルを受信するための API には、次の 2 種類があります。

- ポーリング用メソッド
- コールバック用メソッド

ポーリング用メソッドは、カスタムアダプターが SDP サーバから能動的にタプルを取得するためのメソッドです。get メソッドと getAll メソッドがあります。get メソッドは、単一のタプルを受信するメソッドです。getAll メソッドは、複数のタプルをまとめて受信するメソッドです。

コールバック用メソッドは、SDP サーバ上でタプルが生成されたときに呼び出され、受動的にタプルを取得するためのメソッドです。コールバック方式でデータを受信するには、SDP サーバ上でタプルが生成されたときに SDP サーバから呼び出されるメソッドを持つオブジェクト (リスナーオブジェクト) を SDP サーバに登録しておく必要があります。

リスナーオブジェクトは、StreamEventListener インタフェースを実装したオブジェクトです。StreamOutput インタフェースには、このリスナーオブジェクトを SDP サーバに登録するためのメソッドとして、registerForNotification メソッドがあります。また、リスナーオブジェクトの SDP サーバへの登録を解除するためのメソッドとして、unregisterForNotification メソッドがあります。

メソッド

StreamOutput インタフェースのメソッド一覧を次の表に示します。

戻り値	メソッド名	機能
void	close()	出力ストリームとの接続を閉じます。
StreamTuple	get()	SDP サーバに登録されているタプルを一つだけ取得します。
ArrayList<StreamTuple>	get(int count)	SDP サーバに登録されているデータを count パラメーターに指定した数だけ取得します。
ArrayList<StreamTuple>	get(int count, long timeout)	SDP サーバに登録されているデータを count パラメーターに指定した数だけ取得します。SDP サーバに結果データがない場合、結果データが到着するか、または timeout パラメーターに指定した時間が経過するまで待機します。
ArrayList<StreamTuple>	getAll()	SDP サーバに登録されているすべてのタプルを取得します。
ArrayList<StreamTuple>	getAll(long timeout)	SDP サーバに登録されているすべてのタプルを取得します。SDP サーバに結果データがない場合、結果データが到着するか、または timeout パラメーターに指定した時間が経過するまで待機します。
int	getFreeQueueSize()	出力ストリームキューの空きサイズを取得します。
int	getMaxQueueSize()	出力ストリームキューの最大サイズを取得します。
void	registerForNotification(StreamEventListener n)	コールバック用リスナーオブジェクトを登録します。

戻り値	メソッド名	機能
void	unregisterForNotification(StreamEventListener n)	登録したリスナーオブジェクトを解除して、以降のコールバック処理の実行を解除します。

注意事項

なし。

close()メソッド

形式

```
void close()
```

説明

出力ストリームとの接続を閉じます。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientException	出力ストリームがすでに閉じている場合

戻り値

なし。

get()メソッド

形式

```
StreamTuple get()
```

説明

SDP サーバに登録されているタプルを一つだけ取得します。

出力ストリームキューにタプルがない場合は、null が返却されます。

このメソッドを呼び出した時点でクエリグループに対して停止が通知されていた場合（メソッドによってデータ送信終了が通知された場合、またはコマンドによってクエリグループ停止が通知された場合）、次のように処理されます。

- 停止通知後、最初のメソッド呼び出しの場合
null が返却されます。

- 停止通知後、2 回目以降のメソッド呼び出しの場合
例外がスローされます。

このメソッドはポーリング用メソッドです。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientEndOfStreamException	送信データの処理が終了している場合
SDPClientException	<ul style="list-style-type: none"> • 出力ストリームがすでに閉じている場合 • コールバック用リスナーオブジェクトがすでに登録されている場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中ではない場合
SDPClientQueryGroupStopException	結果データがなく、クエリグループが停止中の場合

戻り値

取得したタプル (StreamTuple 型オブジェクト)。

get(int count)メソッド

形式

```
ArrayList<StreamTuple> get(int count)
```

説明

SDP サーバに登録されているデータを count パラメーターに指定した数だけ取得します。

SDP サーバに結果データがない場合、空の ArrayList 型オブジェクトが返却されます。

このメソッドを呼び出した時点でクエリグループに対して停止が通知されていた場合 (メソッドによってデータ送信終了が通知された場合、またはコマンドによってクエリグループ停止が通知された場合)、次のように処理されます。

- 停止通知後、最初のメソッド呼び出しの場合
空の ArrayList 型オブジェクトが返却されます。
- 停止通知後、2 回目以降のメソッド呼び出しの場合
例外がスローされます。

このメソッドは、ポーリング用メソッドです。

パラメーター

count

SDP サーバから取得するデータの数を指定します。指定可能範囲は 1~1,048,576 です。

なお、実際に取得できるデータの数は、このパラメーターと出力ストリームキューの状態によって次に示すようになります。

条件	取得できる数
count の指定値 ≤ 出力ストリームキューのデータ数	count パラメーターに指定した数
count の指定値 > 出力ストリームキューのデータ数	出力ストリームキューのデータ数
count の指定値 > 出力ストリームキューの最大サイズ	

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientEndOfStreamException	送信データの処理が終了している場合
SDPClientException	<ul style="list-style-type: none"> 出力ストリームがすでに閉じている場合 コールバック用リスナーオブジェクトがすでに登録されている場合 パラメーターが不正の場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合 (SDPClientQueryGroupStateException の詳細例外)
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合
SDPClientQueryGroupStopException	結果データがなく、かつクエリグループが停止中の場合 (SDPClientQueryGroupStateException の詳細例外)

戻り値

タプルのリスト (ArrayList<StreamTuple>型オブジェクト)。

get(int count, long timeout)メソッド

形式

```
ArrayList<StreamTuple> get(int count, long timeout)
```

説明

SDP サーバに登録されているデータを count パラメーターに指定した数だけ取得します。

SDP サーバに結果データがない場合、結果データが到着するか、または timeout パラメーターに指定した時間が経過するまで待機します。待機中に結果データが到着した場合、到着したデータを格納した ArrayList 型オブジェクトが返却されます。timeout パラメーターに指定した時間が経過した場合、または待機中のスレッドに割り込みが発生した場合は、次のどちらかのオブジェクトが返却されます。

- データが登録されている場合
登録されたデータが格納された ArrayList 型オブジェクトが返却されます。
- データが登録されていない場合
空の ArrayList 型オブジェクトが返却されます。

このメソッドを呼び出した時点でクエリグループに対して停止が通知されていた場合（メソッドによってデータ送信終了が通知された場合、またはコマンドによってクエリグループ停止が通知された場合）、次のように処理されます。

- 停止通知後、最初のメソッド呼び出しの場合
空の ArrayList 型オブジェクトが返却されます。
- 停止通知後、2 回目以降のメソッド呼び出しの場合
例外がスローされます。

このメソッドは、ポーリング用メソッドです。

パラメーター

count

SDP サーバから取得するデータの数を指定します。指定可能範囲は 1~1,048,576 です。

なお、実際に取得できるデータの数は、このパラメーターと出力ストリームキューの状態によって次に示すようになります。

条件	取得できる数
count の指定値 ≤ 出力ストリームキューのデータ数	count パラメーターに指定した数
count の指定値 > 出力ストリームキューのデータ数	出力ストリームキューのデータ数
count の指定値 > 出力ストリームキューの最大サイズ	

timeout

データがない場合に、待機する最大時間をミリ秒で指定します。

指定した値によって、次の処理が実行されます。

指定した値	実行される処理
負の数	待機しません。
0	結果データが到着するか、またはストリームが終了するまで待機します。
そのほかの値	結果データが到着するか、または指定時間が経過するまで待機します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientEndOfStreamException	送信データの処理が終了している場合
SDPClientException	<ul style="list-style-type: none"> 出力ストリームがすでに閉じている場合 コールバック用リスナーオブジェクトがすでに登録されている場合 パラメーターが不正の場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合 (SDPClientQueryGroupStateException の詳細例外)
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合
SDPClientQueryGroupStopException	結果データがなく、かつクエリグループが停止中の場合 (SDPClientQueryGroupStateException の詳細例外)

戻り値

タプルのリスト (ArrayList<StreamTuple>型オブジェクト)。

getAll()メソッド

形式

```
ArrayList<StreamTuple> getAll()
```

説明

SDP サーバに登録されているすべてのタプルを取得します。

SDP サーバに結果データがない場合、空の ArrayList 型オブジェクトが返却されます。

このメソッドを呼び出した時点でクエリグループに対して停止が通知されていた場合 (メソッドによってデータ送信終了が通知された場合、またはコマンドによってクエリグループ停止が通知された場合)、次のように処理されます。

- 停止通知後、最初のメソッド呼び出しの場合
空の ArrayList 型オブジェクトが返却されます。
- 停止通知後、2 回目以降のメソッド呼び出しの場合
例外がスローされます。

このメソッドはポーリング用メソッドです。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientEndOfStreamException	送信データの処理が終了している場合
SDPClientException	<ul style="list-style-type: none"> 出力ストリームがすでに閉じている場合 コールバック用リスナーオブジェクトがすでに登録されている場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合
SDPClientQueryGroupStopException	結果データがなく、クエリグループが停止中の場合

戻り値

タプルのリスト (ArrayList<StreamTuple>型オブジェクト)。

getAll(long timeout)メソッド

形式

```
ArrayList<StreamTuple> getAll(long timeout)
```

説明

SDP サーバに登録されているすべてのタプルを取得します。

SDP サーバに結果データがない場合、結果データが到着するか、または timeout パラメーターに指定した時間が経過するまで待機します。待機中に結果データが到着した場合、到着したデータを格納した ArrayList 型オブジェクトが返却されます。timeout パラメーターに指定した時間が経過した場合、または待機中のスレッドに割り込みが発生した場合は、次のどちらかのオブジェクトが返却されます。

- データが登録されている場合
登録されたデータが格納された ArrayList 型オブジェクトが返却されます。
- データが登録されていない場合
空の ArrayList 型オブジェクトが返却されます。

このメソッドを呼び出した時点でクエリグループに対して停止が通知されていた場合 (メソッドによってデータ送信終了が通知された場合、またはコマンドによってクエリグループ停止が通知された場合)、次のように処理されます。

- 停止通知後、最初のメソッド呼び出しの場合
空の ArrayList 型オブジェクトが返却されます。
- 停止通知後、2 回目以降のメソッド呼び出しの場合
例外がスローされます。

このメソッドは、ポーリング用メソッドです。

パラメーター

timeout

データがない場合に、待機する最大時間をミリ秒で指定します。

指定した値によって、次の処理が実行されます。

指定した値	実行される処理
負の数	待機しません。
0	結果データが到着するか、またはストリームが終了するまで待機します。
そのほかの値	結果データが到着するか、または指定時間が経過するまで待機します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientEndOfStreamException	送信データの処理が終了している場合
SDPClientException	<ul style="list-style-type: none"> 出力ストリームがすでに閉じている場合 コールバック用リスナーオブジェクトがすでに登録されている場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合 (SDPClientQueryGroupStateException の詳細例外)
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合
SDPClientQueryGroupStopException	結果データがなく、かつクエリグループが停止中の場合 (SDPClientQueryGroupStateException の詳細例外)

戻り値

タプルのリスト (ArrayList<StreamTuple>型オブジェクト)。

getFreeQueueSize()メソッド

形式

```
int getFreeQueueSize()
```

説明

出力ストリームキューの空きサイズを取得します。

出力ストリームキューの空きサイズは、system_config.properties の engine.maxQueueSize パラメーターに指定した値から、使用中のサイズを引いた値です。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientException	出力ストリームがすでに閉じている場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合 (SDPClientQueryGroupStateException の詳細例外)
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合

戻り値

出力ストリームキューの空きサイズ (int)。

getMaxQueueSize() メソッド

形式

```
int getMaxQueueSize()
```

説明

出力ストリームキューの最大サイズを取得します。

入力ストリームキューの最大サイズは、system_config.properties の engine.maxQueueSize パラメーターに指定した値です。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientCommunicationException	RMI 連携時に通信例外が発生した場合
SDPClientException	出力ストリームがすでに閉じている場合

戻り値

出力ストリームキューの最大サイズ (int)。

registerForNotification(StreamEventListener n)メソッド

形式

```
void registerForNotification(StreamEventListener n)
```

説明

コールバック用リスナーオブジェクトを登録します。

このメソッド実行以降は、ポーリング用メソッド (get メソッドまたは getAll メソッド) を実行できません。

一つのストリームに対して一つのリスナーオブジェクトしか登録できません。また、登録済みのリスナーオブジェクトは、複数のストリームに登録できません。

パラメーター

n

リスナーオブジェクトを指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターに null を指定した場合
SDPClientException	<ul style="list-style-type: none"> 出力ストリームがすでに閉じている場合 ストリームにリスナーオブジェクトが登録済みの場合 指定したリスナーオブジェクトがほかのストリームで登録済みの場合 RMI 連携時にこのメソッドを呼び出した場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合

戻り値

なし。

unregisterForNotification(StreamEventListener n)メソッド

形式

```
void unregisterForNotification(StreamEventListener n)
```

説明

登録したリスナーオブジェクトを解除して、以降のコールバック処理を解除します。

パラメーター

n

リスナーオブジェクトを指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターに null を指定した場合
SDPClientException	<ul style="list-style-type: none">リスナーオブジェクトが未定義の場合出力ストリームがすでに閉じている場合RMI 連携時にこのメソッドを呼び出した場合

戻り値

なし。

8.9 StreamTime クラス (共通 API)

クラス階層

```

java.lang.Object
├─ java.util.Date
└─ jp.co.Hitachi.soft.sdp.common.util.StreamTime
  
```

説明

StreamTime クラスは、Date クラスを拡張したクラスです。

タプルの特定の時刻を表すためのクラスです。

メソッド

StreamTime クラスのメソッド一覧を次の表に示します。

戻り値	メソッド名	機能
boolean	equals(StreamTime when)	自身の時刻と、パラメーターに指定した StreamTime 型オブジェクトが示す時刻が等しいかどうか判定します。
long	getTimeMillis()	自身が保持するミリ秒単位の時刻を取得します。
int	hashCode()	自身を示すハッシュコードを取得します。
java.lang.String	toString()	自身の保持する時刻の情報を文字列表現にして取得します。

注意事項

なし。

equals(StreamTime when)メソッド

形式

```
boolean equals(StreamTime when)
```

説明

自身の時刻と、パラメーターに指定した StreamTime 型オブジェクトの時刻が等しいかどうか判定します。

パラメーター

when

比較対象の StreamTime 型オブジェクトを指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターに null を指定した場合

戻り値

true

パラメーターで指定したオブジェクトが StreamTime 型オブジェクトであり、かつ、自身と同じ時刻です。

false

パラメーターで指定したオブジェクトが StreamTime 型オブジェクトではありません。または、自身と同じ時刻ではありません。

getTimeMillis()メソッド

形式

```
long getTimeMillis()
```

説明

自身が保持するミリ秒単位の時刻を取得します。

パラメーター

なし。

例外

なし。

戻り値

自身が保持するミリ秒単位の時刻 (long)。

hashCode()メソッド

形式

```
int hashCode()
```

説明

自身を示すハッシュコードを取得します。

パラメーター

なし。

例外

なし。

戻り値

自身を示すハッシュコード (int)。

toString()メソッド

形式

```
java.lang.String toString()
```

説明

自身の保持する時刻の情報を文字列表現にして取得します。

パラメーター

なし。

例外

なし。

戻り値

自身の保持する時刻の情報を次に示す形式に変換した文字列表現 (java.lang.String 型。△は半角スペースを示します)。

```
aaa△bbb△cc△dd:ee:ff△ggg△hhh[timeMillis:iii]
```

出力内容を次の表に示します。

出力項目	内容	出力形式 (範囲)
aaa	曜日	Sun, Mon, Tue, Wed, Thu, Fri, Sat のどれかが出力されます。
bbb	月	Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec のどれかが出力されます。
cc	日	2けたの10進数 (01~31) で出力されます。
dd	時	2けたの10進数 (00~23) で出力されます。
ee	分	2けたの10進数 (00~59) で出力されます。
ff	秒	2けたの10進数 (00~59) で出力されます。
ggg	タイムゾーン	タイムゾーンが出力されます。 タイムゾーンの設定がなければ空白が出力されます。
hhh	年	4けたの10進数で出力されます。
iii	自身の保持するミリ秒単位の時刻	自身の保持するミリ秒単位の時刻が出力されます。

出力例を次に示します。
"Thu Jan 01 09:00:01 JST 1970[timeMillis:1234]"

8.10 StreamTuple クラス (共通 API)

クラス階層

```
java.lang.Object
└─jp.co.Hitachi.soft.sdp.common.data.StreamTuple
```

説明

StreamTuple クラスは、タプルを表現するためのクラスです。

コンストラクター

StreamTuple クラスのコンストラクター一覧を次の表に示します。

コンストラクター名	機能
StreamTuple(Object[] dataArray)	パラメーターにデータオブジェクト配列を指定して、StreamTuple クラスのインスタンス (StreamTuple 型オブジェクト) を新しく生成します。

メソッド

StreamTuple クラスのメソッド一覧を次の表に示します。

戻り値	メソッド名	機能
boolean	equals(Object obj)	パラメーターで指定した StreamTuple 型オブジェクトのデータオブジェクト配列および時刻が、自身 (StreamTuple 型オブジェクト) とすべて同じ内容かどうかを判定します。
Object[]	getDataArray()	自身 (StreamTuple 型オブジェクト) のデータオブジェクト配列を取得します。
StreamTime	getSystemTime()	システム時刻を取得します。
int	hashCode()	ハッシュコードを取得します。
java.lang.String	toString()	自身 (StreamTuple 型オブジェクト) のタプル情報を表す文字列を取得します。

注意事項

タプルの時刻 (StreamTime 型オブジェクト) には、タプルが SDP サーバに到着したときのシステム時刻が自動的に設定されます。カスタムアダプターでタプルを生成したときには、初期値としてタイムオブジェクトには null が設定されています。

使用例

Java のデータ型のデータを使用して data を作成して、StreamTuple 型オブジェクトを生成する例を示します。

この例では、Integer 型データおよび String 型データを要素とするタプルを生成します。

```
Object[] data = new Object[] {
    new Integer (1),
    new String ("AAA")
};
tuple = new StreamTuple(data);
```

CQL のデータ型に対応する Java のデータ型については、「3.3 CQL のデータ型」を参照してください。

StreamTuple(Object[] dataArray)コンストラクター

形式

```
public StreamTuple(Object[] dataArray)
```

説明

パラメーターにデータオブジェクト配列を指定して、StreamTuple クラスのインスタンス (StreamTuple 型オブジェクト) を新しく生成します。

パラメーター

dataArray

タプルを構成するデータオブジェクトの配列を指定します。データは Java のデータ型で指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターに null を指定した場合

equals(Object obj)メソッド

形式

```
boolean equals(Object obj)
```

説明

パラメーターで指定した StreamTuple 型オブジェクトのデータオブジェクト配列および時刻が、自身 (StreamTuple 型オブジェクト) とすべて同じ内容かどうかを判定します。

パラメーター

obj

比較する StreamTuple 型オブジェクトを指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターに null を指定した場合

戻り値

true

タプルの内容（データオブジェクト配列および時刻）がすべて同じです。

false

タプルの内容が異なります（内容が一つでも異なる場合は false になります）。

getDataArray()メソッド

形式

```
Object[] getDataArray()
```

説明

自身（StreamTuple 型オブジェクト）のデータオブジェクト配列を取得します。

パラメーター

なし。

例外

なし。

戻り値

Java のデータ型のデータを含んだデータオブジェクト配列（Object[]型）。

getSystemTime()メソッド

形式

```
StreamTime getSystemTime()
```

説明

システム時刻を取得します。

パラメーター

なし。

例外

なし。

戻り値

システム時刻を表すオブジェクト（StreamTime 型オブジェクト）。

hashCode()メソッド

形式

```
int hashCode()
```

説明

ハッシュコードを取得します。

このメソッドは、`java.util.Hashtable` クラスによって提供されるようなハッシュテーブルで使用するために用意されているメソッドです。

パラメーター

なし。

例外

なし。

戻り値

`StreamTuple` 型オブジェクトのハッシュコード (`int`)。

toString()メソッド

形式

```
java.lang.String toString()
```

説明

自身 (`StreamTuple` 型オブジェクト) のタプル情報を表す文字列を取得します。次のデータを文字列に変換したデータが取得できます。

- データオブジェクト配列 (`Object[]`型オブジェクト)
- システム時刻 (`StreamTime` 型オブジェクト)

パラメーター

なし。

例外

なし。

戻り値

自身のタプル情報を表す文字列 (`java.lang.String` 型)。

8.11 例外クラス (共通 API)

クラス階層

```

java.lang.Object
├─java.lang.Throwable
│   └─java.lang.Exception
│       ├──jp.co.Hitachi.soft.sdp.common.exception.SDPClientException
│       ├──jp.co.Hitachi.soft.sdp.common.exception.SDPClientFreeInputQueueSizeThresholdOverException
│       └─jp.co.Hitachi.soft.sdp.common.exception.SDPClientFreeInputQueueSizeLackException

```

SDPClientException クラス (共通 API)

クラス階層

```

java.lang.Object
├─java.lang.Throwable
│   └─java.lang.Exception
│       └─jp.co.Hitachi.soft.sdp.common.exception.SDPClientException

```

説明

ストリームデータ処理システムの SDP サーバで検知された例外の内容をカスタムアダプターに返す例外クラスです。

フィールド

なし。

コンストラクター

なし。

メソッド

java.lang.Exception クラスを継承しているため、getMessage メソッドなどの java.lang.Exception クラスのメソッドを使用できます。

サブクラス

SDPClientException クラスのサブクラスを次の表に示します。

表 8-3 SDPClientException クラスのサブクラス

項番	サブクラス	説明
1	SDPClientQueryGroupException (共通 API)	カスタムアダプター実行時に、SDP サーバで検知したクエリグループの例外をカスタムアダプターに返すための例外クラスです。
2	SDPClientQueryGroupNotExistException (共通 API)	指定したクエリグループがない場合に生成される例外クラスです。

項番	サブクラス	説明
3	SDPClientQueryGroupStateException (共通 API)	指定したクエリグループが実行中ではない場合に生成される例外クラスです。
4	SDPClientQueryGroupHoldException (共通 API)	閉塞状態のクエリグループに対してメソッドを呼び出した場合に生成される例外クラスです。
5	SDPClientQueryGroupStopException (共通 API)	停止状態のクエリグループに対してメソッドを呼び出した場合に生成される例外クラスです。
6	SDPClientEndOfStreamException (共通 API)	入力ストリームデータが終了している場合に生成される例外クラスです。
7	SDPClientRelationStateException (共通 API)	指定したストリームがリレーション状態の場合に生成される例外クラスです。
8	SDPClientCommunicationException(RMI 連携用)	ストリームデータ処理システムで発生した RMI 通信例外をカスタムアダプターに返すための通信例外クラスです。

サブクラスのクラス階層を次に示します。

```

jp.co.Hitachi.soft.sdp.common.exception.SDPClientException
├─jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupException
│   ├──jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupNotExistException
│   ├──jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupStateException
│   ├──jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupHoldException
│   └─jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupStopException
├─jp.co.Hitachi.soft.sdp.common.exception.SDPClientEndOfStreamException
├─jp.co.Hitachi.soft.sdp.common.exception.SDPClientRelationStateException
└─jp.co.Hitachi.soft.sdp.common.exception.SDPClientCommunicationException

```

SDPClientFreeInputQueueSizeThresholdOverException クラス (共通 API)

クラス階層

```

java.lang.Object
├─java.lang.Throwable
│   └─java.lang.Exception
│       └─jp.co.Hitachi.soft.sdp.common.exception.
│           SDPClientFreeInputQueueSizeThresholdOverException

```

説明

入力ストリームキューの空きサイズが system_config.properties の stream.freeInputQueueSizeThreshold キーで指定したしきい値以下になったことを示す例外クラスです。

フィールド

なし。

コンストラクター

なし。

サブクラス

なし。

SDPClientFreeInputQueueSizeLackException クラス (共通 API)

クラス階層

```
java.lang.Object
├─ java.lang.Throwable
│   └─ java.lang.Exception
│       └─ jp.co.Hitachi.soft.sdp.common.exception.
│           └─ SDPClientFreeInputQueueSizeLackException
```

説明

入力ストリームキューの空きサイズが不足していることを示す例外クラスです。

フィールド

なし。

コンストラクター

なし。

サブクラス

なし。

9

データ送受信 API を使用したサンプルプログラム

この章では、データ送受信 API を使用したサンプルプログラムについて説明します。

なお、ここで説明するサンプルプログラムは、インストール後、運用ユーザーの登録と運用ディレクトリの作成を実施したあとで実行してください。運用ディレクトリ作成までの手順については、マニュアル「uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド」を参照してください。

9.1 サンプルプログラムの構成

Stream Data Platform - AF が提供するサンプルプログラムの構成について説明します。

サンプルプログラムを実行する場合は、次の表に示すファイルを<インストールディレクトリ>%samples%api%から<運用ディレクトリ>%samples%api%にディレクトリごとコピーして使用します。

なお、カスタムアダプターのプロセス構成によって、使用するサンプルプログラムのファイルが異なります。

表 9-1 サンプルプログラムの構成

ディレクトリ	ファイル	カスタムアダプターのプロセス構成		説明
		RMI 連携	インプロセス連携	
query%	Inprocess_QueryTest	—	○	インプロセス連携クエリ定義ファイル
	RMI_QueryTest	○	—	RMI 連携クエリ定義ファイル
conf%	Inprocess_QueryGroupTest* ¹	—	○	インプロセス連携クエリグループ用プロパティファイル
	jvm_client_options.cfg* ²	○	—	RMI 連携用 JavaVM オプションファイル
	jvm_options.cfg* ³	○	○	SDP サーバ用 JavaVM オプションファイル
	logger.properties	○	○	ログファイル出力用プロパティファイル
	RMI_QueryGroupTest* ⁴	○	—	RMI 連携クエリグループ用プロパティファイル
	system_config.properties	○	○	システムコンフィグプロパティファイル
	user_app.InprocessAPTest.properties* ⁵	—	○	インプロセス連携用プロパティファイル
src%samples%	Inprocess_Main.java	—	○	インプロセス連携送受信制御 AP
	Inprocess_Receiver.java	—	○	インプロセス連携データ受信 AP
	Inprocess_Sender.java	—	○	インプロセス連携データ送信 AP
	RMI_ReceiveTupleTest.java	○	—	RMI 連携データ受信 AP
	RMI_SendTupleTest.java	○	—	RMI 連携データ送信 AP

(凡例)

○：使用します。 -：使用しません。

注※1

Inprocess_QueryGroupTest には、インプロセス連携用クエリ定義ファイルのパスとして、次のパスが設定されています。

```
querygroup.cqlFilePath=.¥¥samples¥¥api¥¥query¥¥Inprocess_QueryTest
```

注※2

jvm_client_options.cfg には、ログファイルの出力先 (<運用ディレクトリ>¥logs¥) とクラスパスとして、次のパスが設定されています。

```
SDP_JVM_LOG=-XX:HitachiJavaLog:.¥¥logs¥¥SDPClientVM
```

```
SDP_CLASS_PATH=.¥¥samples¥¥api¥¥src
```

注※3

jvm_options.cfg には、ログファイルの出力先 (<運用ディレクトリ>¥logs¥) として、次のパスが設定されています。

```
SDP_JVM_LOG=-XX:HitachiJavaLog:.¥¥logs¥¥SDPServerVM
```

注※4

RMI_QueryGroupTest には、RMI 連携用クエリ定義ファイルのパスとして、次のパスが設定されています。

```
querygroup.cqlFilePath=.¥¥samples¥¥api¥¥query¥¥RMI_QueryTest
```

注※5

user_app.InprocessAPTTest.properties には、インプロセス連携用のメインクラス名とクラスパスとして、次のパスが設定されています。

```
user_app.classname=samples.Inprocess_Main
```

```
user_app.classpath_dir=.¥¥samples¥¥api¥¥src
```

9.2 RMI 連携カスタムアダプターのサンプルプログラム

この節では、RMI 連携カスタムアダプターのサンプルプログラムの実行手順、およびサンプルプログラムの内容について説明します。

まず、実行手順に従ってサンプルプログラムを動作させたあと、サンプルプログラムを構成する定義ファイルやソースファイルがどのように定義・実装されているかを確認してください。

9.2.1 RMI 連携カスタムアダプターのサンプルプログラムの実行手順

RMI 連携カスタムアダプターのサンプルプログラムの実行手順を示します。なお、この手順では、四つのコンソールを使用します。実行するコンソールを【 】で囲んで示しますので、必要に応じて新しいコンソールを開いて実行してください。

1. 運用ディレクトリに移動します。【コンソール 1】

```
cd <運用ディレクトリ>
```

2. <インストールディレクトリ>%samples%api% 下のサンプルプログラムを <運用ディレクトリ>直下にディレクトリごとコピーします。【コンソール 1】

なお、<運用ディレクトリ>直下にサンプル環境を構築済みの場合は、この手順および 3. の手順は不要です。4. に進んでください。

```
xcopy /EY <インストールディレクトリ>%samples%api% .%samples%api%
```

3. サンプルプログラムの conf ディレクトリ以下を運用ディレクトリ直下の conf ディレクトリ下にコピーします。【コンソール 1】

<運用ディレクトリ>%conf% 下にすでに各プロパティファイルがある場合は、上書きします。上書きしたくない場合は、あらかじめ <運用ディレクトリ>%conf% 下のプロパティファイルをほかのディレクトリなどに退避しておいてください。

```
xcopy /Y samples%api%conf .%conf%
```

4. ソースファイルをコンパイルします。【コンソール 1】

ソースファイルのコンパイルは、javac コマンドを実行できる環境で実行してください。

```
javac※ -classpath <インストールディレクトリ>%lib%sdp.jar samples%api%src%samples%RMI*.java
```

注※

「7.5 コンパイル手順」を参照し、事前に環境変数を定義してください。または、環境に応じて、javac コマンドのパスを指定してください。

5. SDP サーバを起動します。【コンソール 1】

SDP サーバのポート番号には、デフォルトでは 20400 が設定されています。

ポート番号を変更したい場合は、起動前に、<運用ディレクトリ>%conf%system_config.properties の rmi.serverPort プロパティで値を変更してください。

```
.%bin%sdpstart
```

6. クエリグループ (ストリームとクエリ) を登録します。【コンソール 2】

新しいコンソールを表示して、運用ディレクトリに移動します。

```
cd <運用ディレクトリ>
```

その後、次のコマンドを実行します。

```
.%bin%sdpcql RMI_QueryGroupTest
```

7. クエリグループを開始します。【コンソール 2】

```
./bin/sdpcqlstart RMI_QueryGroupTest
```

8. データ受信 AP を起動します。【コンソール 3】

新しいコンソールを表示して、運用ディレクトリに移動します。

```
cd <運用ディレクトリ>
```

その後、次のコマンドを実行します。-clientcfg オプションには、サンプルプログラムとして提供されている RMI 連携用 JavaVM オプションファイルを指定します。

```
./bin/sdpstartap -clientcfg ./conf/jvm_client_options.cfg samples.RMI_ReceiveTupleTest
```

9. データ送信 AP を起動します。【コンソール 4】

新しいコンソールを表示して、運用ディレクトリに移動します。

```
cd <運用ディレクトリ>
```

その後、次のコマンドを実行します。-clientcfg オプションには、サンプルプログラムとして提供されている RMI 連携用 JavaVM オプションファイルを指定します。

```
./bin/sdpstartap -clientcfg ./conf/jvm_client_options.cfg samples.RMI_SendTupleTest
```

10. 実行結果を確認します。【コンソール 3】

データ受信 AP を起動したコンソール【コンソール 3】に、ID1, VAL1, および VAL2 の値として 0~9 が表示されることを確認します。

```
Get Data:ID1=0, VAL1=data1:0, VAL2=data2:0, TIME=...
Get Data:ID1=1, VAL1=data1:1, VAL2=data2:1, TIME=...
Get Data:ID1=2, VAL1=data1:2, VAL2=data2:2, TIME=...
:
Get Data:ID1=9, VAL1=data1:9, VAL2=data2:9, TIME=...
```

11. クエリグループを停止します。【コンソール 2】

```
./bin/sdpcqlstop RMI_QueryGroupTest
```

12. SDP サーバを停止します。【コンソール 2】

```
./bin/sdpstop
```

9.2.2 クエリグループの定義内容 (RMI 連携カスタムアダプター)

ここでは、「9.2.1 RMI 連携カスタムアダプターのサンプルプログラムの実行手順」の手順 6. で登録したクエリグループ (RMI_QueryGroupTest) の内容について説明します。

この RMI 連携クエリグループ用プロパティファイルでは、s1, s2 という名称の 2 種類のストリームと、join という名称のクエリを定義しています。生成されたストリームタプルは、クエリ名 join をストリーム名とする出力ストリームとして出力されます。

定義内容を次に示します。なお、大文字小文字の表記はサンプルと異なります。

```
REGISTER STREAM s1(c1 INT, c2 VARCHAR(20));
REGISTER STREAM s2(c1 INT, c2 VARCHAR(20));

REGISTER QUERY join
ISTREAM(SELECT s1.c1 AS ID1, s1.c2 AS VAL1, s2.c2 AS VAL2
FROM s1[ROWS 3], s2[ROWS 2]
WHERE s1.c1 = s2.c1);
```

ストリーム s1 と s2 のデータを連結して、ストリームタプルを生成する操作が定義されています。

9.2.3 RMI 連携データ送信 AP の内容

ここでは、「9.2.1 RMI 連携カスタムアダプターのサンプルプログラムの実行手順」の手順 4. でコンパイルした RMI 連携データ送信 AP (RMI_SendTupleTest.java) の内容について説明します。

このアプリケーションは、ストリーム s1, s2 を SDP サーバに送信するアプリケーションです。データ送信では、StreamInput インタフェースを使用します。

ソースコードを次に示します。なお、「// 【1】」「//~ 【1】」などのコメントは、以降の説明の番号と対応しています（これらのコメントは、実際のサンプルプログラムには記載されていません。また、一部コメントの説明は実際のサンプルプログラムと異なります）。

サンプルプログラムの内容

```
package samples;

import jp.co.Hitachi.soft.sdp.common.data.StreamTuple;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupHoldException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupStopException;
import jp.co.Hitachi.soft.sdp.api.SDPConnector;
import jp.co.Hitachi.soft.sdp.api.SDPConnectorFactory;
import jp.co.Hitachi.soft.sdp.api.StreamInput;

public class RMI_SendTupleTest{
    //送信対象のストリーム名称を設定します
    private static final String TARGET_STREAM1 = "s1";
    private static final String TARGET_STREAM2 = "s2";
    private static final String TARGET_QUERYGROUP = "RMI_QueryGroupTest";

    private static SDPConnector connector;
    private static StreamInput streamInput1;
    private static StreamInput streamInput2;
    private static long interval=1000;

    // 【1】
    public static void main(String[] args) {
        try {
            //SDPサーバに接続します
            connector = SDPConnectorFactory.connect();
            execute();
        } catch (SDPClientException sce) {
            System.err.println(sce.getMessage());
        }
    }
    //~ 【1】

    // データ送信処理を実行します
    public static void execute() {
        try {
            // 【2】
            //送信対象のストリームに接続します
            streamInput1 = connector.openStreamInput(TARGET_QUERYGROUP, TARGET_STREAM1);
            streamInput2 = connector.openStreamInput(TARGET_QUERYGROUP, TARGET_STREAM2);
            //~ 【2】

            //データを送信します
            for(int i=0; i<10; i++){
                try{
                    Thread.sleep(interval);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }

            // 【3】
            //タプルに設定するオブジェクトを生成します
            Object[] data1 = new Object[]{
                new Integer(i),
                new String("data1:"+i)
            };
            Object[] data2 = new Object[]{
                new Integer(i),
                new String("data2:"+i)
            };

            //オブジェクトをタプルに設定します
```



```

StreamTuple tuple1 = new StreamTuple(data1);
StreamTuple tuple2 = new StreamTuple(data2);
//~ 【3】

// 【4】
// タブルを送信します
streamInput1.put(tuple1);
streamInput2.put(tuple2);
}
// データ送信完了を通知します
streamInput1.putEnd();
streamInput2.putEnd();
//~ 【4】
} catch (SDPClientQueryGroupStopException sce) {
// クエリグループが停止しています (sdpcqlstop コマンド)
System.err.println(sce.getMessage());
} catch (SDPClientQueryGroupHoldException sce) {
// クエリグループが閉塞しています
System.err.println(sce.getMessage());
} catch (SDPClientException sce) {
System.err.println(sce.getMessage());
}

// 【5】
} finally {
if (!connector.isClosed()) {
try {
// 入力ストリームと接続を閉じます
streamInput1.close();
streamInput2.close();
// SDPサーバと接続を閉じます
connector.close();
} catch (SDPClientException sce2) {
System.err.println(sce2.getMessage());
}
}
}
}
//~ 【5】
}

```

ソースコードの内容について説明します。

1. SDPConnectorFactory.connect メソッドで SDP サーバに接続して、SDPConnector 型オブジェクトを取得します。
2. SDPConnector 型オブジェクトを使用して、openStreamInput メソッドを呼び出します。このメソッドによって、送信先ストリームの入力ストリームに接続して、StreamInput 型オブジェクトを取得します。
3. 送信データをオブジェクトとして生成して、タブルに設定します。
4. StreamInput 型オブジェクトを使用して put メソッドを呼び出し、2. で接続した入力ストリームに対してデータ (タブル) を送信します。すべてのデータの送信が完了したあと、putEnd メソッドで送信完了を通知します。
5. StreamInput 型オブジェクトを使用して close メソッドを呼び出して、入力ストリームとの接続を切断します。その後、SDPConnector 型オブジェクトを使用して close メソッドを呼び出して、SDPConnector を閉じます。

9.2.4 RMI 連携データ受信 AP の内容

ここでは、「9.2.1 RMI 連携カスタムアダプターのサンプルプログラムの実行手順」の手順 4. でコンパイルした RMI 連携データ受信 AP (RMI_ReceiveTupleTest.java) の内容について説明します。

このアプリケーションは、ストリーム s1, s2 を JOIN した結果を受け取るアプリケーションです。データ受信では、StreamOutput クラスを使用します。

ソースコードを次に示します。なお、「// 【1】」「//~ 【1】」などのコメントは、以降の説明の番号と対応しています（これらのコメントは、実際のサンプルプログラムには記載されていません。また、一部コメントの説明は実際のサンプルプログラムと異なります）。

サンプルプログラムの内容

```
package samples;

import java.util.Iterator;
import java.util.List;

import jp.co.Hitachi.soft.sdp.api.SDPConnectorFactory;
import jp.co.Hitachi.soft.sdp.api.SDPConnector;
import jp.co.Hitachi.soft.sdp.api.StreamOutput;
import jp.co.Hitachi.soft.sdp.common.data.StreamTuple;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientCommunicationException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupHoldException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupStopException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientEndOfStreamException;
import jp.co.Hitachi.soft.sdp.common.util.StreamTime;
/**
 * ストリームデータを取得し、表示します
 *
 */
public class RMI_ReceiveTupleTest {
    //取得するストリーム名称を設定します
    private static final String STREAM_NAME = "JOIN";
    private static final String TARGET_QUERYGROUP = "RMI_QueryGroupTest";

    private static SDPConnector connector;
    private StreamOutput streamOutput;
    private boolean run=true;
    private long interval=10;
    public static void main(String[] args) {

        // 【1】
        try {
            // SDPサーバに接続します
            connector = SDPConnectorFactory.connect();

            //受信処理を開始します
            RMI_ReceiveTupleTest receiver = new RMI_ReceiveTupleTest();
            receiver.execute();
        } catch (SDPClientException sce) {
            System.err.println(sce.getMessage());
        }
        //~ 【1】
    }

    //受信処理を実行します
    public void execute() {
        try {
            // 【2】
            //結果ストリームに接続します
            streamOutput = connector.openStreamOutput(TARGET_QUERYGROUP, STREAM_NAME);
            //~ 【2】

            while (run) {
                try {
                    Thread.sleep(interval);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }

            // 【3】
            //タプルを受信します
            List<StreamTuple> tupleList = streamOutput.getAll();
            if (tupleList.isEmpty()) continue;
            //~ 【3】
        }
    }
}
```


9.3 インプロセス連携カスタムアダプターのサンプルプログラム

この節では、インプロセス連携カスタムアダプターのサンプルプログラムの実行手順、およびサンプルプログラムの内容について説明します。

まず、実行手順に従ってサンプルプログラムを動作させたあと、サンプルプログラムを構成する定義ファイルやソースファイルがどのように定義・実装されているかを確認してください。

9.3.1 インプロセス連携カスタムアダプターのサンプルプログラムの実行手順

インプロセス連携カスタムアダプターのサンプルプログラムの実行手順を示します。なお、この手順では、二つのコンソールを使用します。実行するコンソールを【 】で囲んで示しますので、必要に応じて新しいコンソールを開いて実行してください。

1. 運用ディレクトリに移動します。【コンソール 1】

```
cd <運用ディレクトリ>
```

2. <インストールディレクトリ>%samples%api%下のサンプルプログラムを<運用ディレクトリ>直下にディレクトリごとコピーします。【コンソール 1】

なお、<運用ディレクトリ>直下にサンプル環境を構築済みの場合は、この手順および 3. の手順は不要です。4. に進んでください。

```
xcopy /EY <インストールディレクトリ>%samples%api . %samples%api%
```

3. サンプルプログラムの conf ディレクトリ以下を運用ディレクトリ直下の conf ディレクトリ下にコピーします。【コンソール 1】

<運用ディレクトリ>%conf 下にすでに各プロパティファイルがある場合は、上書きします。上書きしたくない場合は、あらかじめ<運用ディレクトリ>%conf 下のプロパティファイルをほかのディレクトリなどに退避しておいてください。

```
xcopy /Y samples%api%conf . %conf%
```

4. ソースファイルをコンパイルします。【コンソール 1】

ソースファイルのコンパイルは、javac コマンドを実行できる環境で実行してください。

```
javac※ -classpath <インストールディレクトリ>%lib%sdp.jar samples%api%src%samples%Inprocess*.java
```

注※

「7.5 コンパイル手順」を参照し、事前に環境変数を定義してください。または、環境に応じて、javac コマンドのパスを指定してください。

5. SDP サーバを起動します。【コンソール 1】

```
./%bin%sdpstart
```

6. クエリグループ (ストリームとクエリ) を登録します。【コンソール 2】

新しいコンソールを表示して、運用ディレクトリに移動します。

```
cd <運用ディレクトリ>
```

その後、次のコマンドを実行します。

```
./%bin%sdpcql Inprocess_QueryGroupTest
```

7. クエリグループを開始します。【コンソール 2】

```
./%bin%sdpcqlstart Inprocess_QueryGroupTest
```

8. インプロセス連携用アプリケーションを開始します。【コンソール 2】

```
./bin%sdpstartinpro InprocessAPTest
```

9. 実行結果を確認します。【コンソール 1】

SDP サーバを起動したコンソール【コンソール 1】に、結果が表示されることを確認します。サンプルプログラムのデータ受信 AP（ポーリング方式）では、["ad0", 0]~["ad24", 24]の結果を受信した結果が、コンソールに出力されます。

```
Receiver : Tuple Get on FILTER1 [ VAL=ad0, ID=0, TIME=... ]
:
Receiver : Tuple Get on FILTER1 [ VAL=ad23, ID=23, TIME=... ]
Receiver : Tuple Get on FILTER1 [ VAL=ad24, ID=24, TIME=... ]
```

10. インプロセス連携用アプリケーションを停止します。【コンソール 2】

```
./bin%sdpstopinpro InprocessAPTest
```

11. クエリグループを停止します。【コンソール 2】

```
./bin%sdpcqlstop Inprocess_QueryGroupTest
```

12. SDP サーバを停止します。【コンソール 2】

```
./bin%sdpstop
```

9.3.2 クエリグループの定義内容（インプロセス連携カスタムアダプター）

ここでは、「9.3.1 インプロセス連携カスタムアダプターのサンプルプログラムの実行手順」の手順 6.で登録したインプロセス連携クエリグループ用プロパティファイル（Inprocess_QueryGroupTest）の内容について説明します。

このクエリグループでは、送信先ストリーム名として data0 を定義して、受信元ストリーム名として、ポーリング受信に filter1 を定義しています。

定義内容を次に示します。なお、大文字小文字の表記はサンプルと異なります。

```
REGISTER STREAM data0(name VARCHAR(10), num BIGINT);
REGISTER QUERY filter1 ISTREAM(SELECT * FROM data0[ROWS 1] WHERE data0.num <= 24);
```

9.3.3 インプロセス連携送受信制御 AP の内容

ここでは、「9.3.1 インプロセス連携カスタムアダプターのサンプルプログラムの実行手順」の手順 4.でコンパイルしたインプロセス連携送受信制御 AP（Inprocess_Main.java）の内容について説明します。このプログラムは、タプルを送受信するアプリケーションを制御する、メインプログラムです。

ソースコードを次に示します。なお、「// 【1】」「//~ 【1】」などのコメントは、以降の説明の番号と対応しています（これらのコメントは、実際のサンプルプログラムには記載されていません。また、一部コメントの説明は実際のサンプルプログラムと異なります）。

サンプルプログラムの内容

```
package samples;

import jp.co.Hitachi.soft.sdp.api.SDPConnector;
import jp.co.Hitachi.soft.sdp.api.inprocess.StreamInprocessUP;
import jp.co.Hitachi.soft.sdp.common.exception.SDPLientException;

public class Inprocess_Main implements StreamInprocessUP {

    // 受信スレッドオブジェクト
    Inprocess_Receiver receiver = null;
```

```

// 送信用スレッドオブジェクト
Inprocess_Sender sender = null;

// SDPサーバとのコネクタ
SDPConnector connector = null;

// 【1】
public void execute(SDPConnector sc) {
//~ 【1】

    this.connector = sc;

// 【2】
// ポーリング用受信スレッドを開始します
this.receiver = new Inprocess_Receiver(sc);
receiver.start();

// 送信スレッドを開始します
this.sender = new Inprocess_Sender(sc);
sender.start();
//~ 【2】
}

// 【1】
public void stop() {
//~ 【1】

    try {
// 【3】
// 送信スレッドを停止します
if(sender != null) {
    sender.terminate();
    sender.join();
}
// 受信スレッドを停止します
if(receiver != null){
    receiver.terminate();
    receiver.join();
}
} catch (InterruptedException e) {
    System.err.println("Main : " + e.getMessage());
}

    try {
// コネクタを閉じます
connector.close();
} catch (SDPClientException sce) {
    System.err.println("Main : " + sce.getMessage());
}
//~ 【3】
}
}

```

ソースコードの内容について説明します。

1. このアプリケーションの Inprocess_Main クラスは、StreamInprocessUP インタフェースを実装したクラスです。このため、StreamInprocessUP インタフェースで定義されている execute メソッドと stop メソッドを実装します。
2. アプリケーション開始時に実行される execute メソッドで、データ受信スレッドとデータ送信スレッドを開始します。
3. アプリケーション停止時に実行される stop メソッドでは、2.で開始したデータ受信スレッドとデータ送信スレッドを停止します。その後、SDPConnector 型オブジェクトの close メソッドを呼び出して、SDPConnector を閉じます。

9.3.4 インプロセス連携データ送信 AP の内容

ここでは、「9.3.1 インプロセス連携カスタムアダプターのサンプルプログラムの実行手順」の手順 4.でコンパイルしたインプロセス連携データ送信 AP (Inprocess_Sender.java) の内容について説明します。

参考

このサンプルプログラムでは、送信専用のスレッドクラスを別に定義して、クライアント AP のメインクラス (StreamInprocessUP インタフェースを実装した Inprocess_Main クラス) 側からそのスレッドを生成して送信を委譲する例を示しています。これ以外の方法として、Inprocess_Main クラス自身が SDP サーバにタプルを送信する処理もできます。

ソースコードを次に示します。なお、「// 【1】」「//~ 【1】」などのコメントは、以降の説明の番号と対応しています (これらのコメントは、実際のサンプルプログラムには記載されていません)。

サンプルプログラムの内容

```
package samples;

import java.util.ArrayList;

import jp.co.Hitachi.soft.sdp.common.data.StreamTuple;
import jp.co.Hitachi.soft.sdp.common.exception.SDPCClientException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPCClientQueryGroupHoldException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPCClientQueryGroupStopException;
import jp.co.Hitachi.soft.sdp.api.SDPConnector;
import jp.co.Hitachi.soft.sdp.api.StreamInput;

public class Inprocess_Sender extends Thread{

    // 【2】
    // スレッドの起動状態
    private volatile boolean running = true;
    //~ 【2】

    // SDPサーバとのコネクタ
    private SDPConnector connector;

    // 送信スレッドの生成処理
    public Inprocess_Sender(SDPConnector c) {
        this.connector = c;
    }

    ArrayList<StreamTuple> list = new ArrayList<StreamTuple>();

    public void run() {

        final String group_name = "Inprocess_QueryGroupTest";
        final String stream_name = "DATA0";

        // 送信ストリームオブジェクト
        StreamInput si = null;

        // 送信ストリームに接続します
        try {
            si = connector.openStreamInput(group_name, stream_name);
        } catch (SDPCClientException sce) {
            System.err.println("Sender : " + sce.getMessage());
            running = false;
        }

        for(int i = 0; i < 50; i++){
            if(!running) {
                break;
            }

            // 送信データを生成します
            Object[] data = new Object[]{
                new String("ad"+i),
```

```

        new Long(i)
    };

    // タプルオブジェクトを生成します
    StreamTuple tuple = new StreamTuple(data);

    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        System.err.println("Sender : Thread is interrupted");
        e.printStackTrace();
    }

    // 単一のタプルを送信します
    try {
        si.put(tuple);
    } catch (SDPClientQueryGroupStopException sce) {
        //クエリグループが停止しています(sdpqclstopコマンド)
        //何もせずに処理を続行します
    } catch (SDPClientQueryGroupHoldException sce) {
        //クエリグループが閉塞しています
        //何もせずに処理を続行します
    } catch (SDPClientException sce) {
        System.err.println("Sender : " + sce.getMessage());
        running = false;
    }
}

// 【1】
if(running) {
    try {
        si.putEnd();
    } catch (SDPClientQueryGroupStopException sce) {
        //クエリグループが停止しています(sdpqclstopコマンド)
        //何もせずに処理を続行します
    } catch (SDPClientQueryGroupHoldException sce) {
        //クエリグループが閉塞しています
        //何もせずに処理を続行します
    } catch (SDPClientException sce) {
        System.err.println("Sender : " + sce.getMessage());
    }
}

try {
    if (si != null) {
        // 送信ストリームを閉じます
        si.close();
    }
} catch (SDPClientException sce) {
    System.err.println("Sender : " + sce.getMessage());
}

}

// 【2】
public void terminate() {
    System.out.println("Sender : terminate called");
    this.running = false;
}

//~ 【2】
}
}

```

ソースコードの内容について説明します。

1. データの送信方法は、RMI 連携の場合と同様です。ただし、データ送信完了通知は、terminate メソッドによって終了条件が設定されていない場合に実行します。
2. terminate メソッドにはタプルの送信を停止させる処理、つまり Inprocess_Sender スレッドの終了条件を記述しています。

terminate メソッドは、SDP サーバのスレッドから呼び出されるため、Inprocess_Receiver クラスの running フィールドは二つのスレッドから同時に読み取られる場合があります。このため、running フィールドは、volatile 属性にする必要があります。

9.3.5 インプロセス連携データ受信 AP の内容（ポーリング方式）

ここでは、「9.3.1 インプロセス連携カスタムアダプターのサンプルプログラムの実行手順」の手順 4. でコンパイルしたインプロセス連携データ受信 AP (Inprocess_Receiver.java) の内容について説明します。このプログラムでは、Inprocess_Receiver スレッドが 100 ミリ秒間隔で SDP サーバに対してポーリング方式でタプル取得処理を実行します。このスレッドは、送受信制御プログラムである Inprocess_Main クラスによって起動されます。

ソースコードを次に示します。なお、「// 【1】」「//~ 【1】」などのコメントは、以降の説明の番号と対応しています（これらのコメントは、実際のサンプルプログラムには記載されていません）。

サンプルプログラムの内容

```
package samples;

import jp.co.Hitachi.soft.sdp.common.data.StreamTuple;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupHoldException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupStopException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientEndOfStreamException;
import jp.co.Hitachi.soft.sdp.common.util.StreamTime;
import jp.co.Hitachi.soft.sdp.api.SDPConnector;
import jp.co.Hitachi.soft.sdp.api.StreamOutput;

public class Inprocess_Receiver extends Thread {

    // SDPサーバとのコネクタ
    private SDPConnector connector;

    // 【2】
    // スレッドの起動状態
    private volatile boolean running = true;
    //~ 【2】

    // 受信スレッドの生成処理
    public Inprocess_Receiver(SDPConnector c) {
        this.connector = c;
    }

    public void run() {

        final String groupName = "Inprocess_QueryGroupTest";
        final String streamName = "FILTER1";

        // 結果ストリームオブジェクト
        StreamOutput so = null;

        // 結果ストリームに接続します
        try {
            so = connector.openStreamOutput(groupName, streamName);
        } catch (SDPClientException sce) {
            System.err.println("Receiver : " + sce.getMessage());
            running = false;
        }

        // 【1】
        // タプルの受信処理を開始します
        while(running){
            //~ 【1】
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                System.err.println("Receiver : Thread is interrupted");
            }
        }
    }
}
```

```

        e.printStackTrace();
    }

    // タプルオブジェクト
    StreamTuple tuple = null;

    // タプルをポーリングで取得します
    try {
        tuple = so.get();
    } catch (SDPClientEndOfStreamException see) {
        //送信ストリームデータが終了しました
        System.out.println("Receiver : " + see.getMessage());
        break;
    } catch (SDPClientQueryGroupStopException sce) {
        //クエリグループが停止しています(sdpcqlstopコマンド)
        System.err.println("Receiver : " + sce.getMessage());
        break;
    } catch (SDPClientQueryGroupHoldException sce) {
        //クエリグループが閉塞しています
        System.err.println("Receiver : " + sce.getMessage());
        break;
    } catch (SDPClientException sce) {
        System.err.println("Receiver : " + sce.getMessage());
        break;
    }
}

//受信したタプルを表示します
if (tuple != null) {
    Object[] data = tuple.getDataArray();
    String val = (String) data[0];
    Long id = (Long) data[1];
    StreamTime time = tuple.getSystemTime();
    System.out.println("Receiver : Tuple Get on " + streamName
        + " [ VAL="+val+", ID="+id+", TIME="+time.toString()+"]");
}

try {
    if (so != null) {
        // 結果ストリームを閉じます
        so.close();
    }
} catch (SDPClientException sce) {
    System.err.println("Receiver : " + sce.getMessage());
}

}

// [2]
public void terminate() {
    System.out.println("Receiver : terminate called");
    this.running = false;
}
//~ [2]
}

```

ソースコードの内容について説明します。

1. データ受信処理は、2.の terminate メソッドによって終了条件が設定されるまで、実行されます。
2. terminate メソッドには、タプルの受信を停止させる処理、つまり Inprocess_Receiver スレッドの終了条件を記述しています。

terminate メソッドは SDP サーバのスレッドから呼び出されるため、Inprocess_Receiver クラスの running フィールドは二つのスレッドから同時に読み取られる場合があります。このため、running フィールドは、volatile 属性にする必要があります。

10 外部定義関数の作成

ストリーム間演算を使用する場合は、CQL でストリーム間演算関数を定義するほかに、外部定義関数の作成が必要です。

この章では、外部定義関数の作成方法について説明します。

10.1 外部定義関数の概要

外部定義関数は、ユーザーが Java の API などを使用して作成する関数です。外部定義関数の処理ロジックを、ユーザーが Java で記述して作成するクラスファイルにメソッドとして実装することで、任意の処理を実行できます。

ストリーム間演算を使用する場合は、CQL でストリーム間演算関数を定義するほかに、外部定義関数を作成します。外部定義関数の作成手順を次に示します。

1. 関数の実装

外部定義関数の処理ロジックを、Java で記述して作成するクラスファイルにメソッドとして実装します。関数の実装方法については、「10.2 外部定義関数の実装」を参照してください。

2. 定義ファイルでの関数の記述

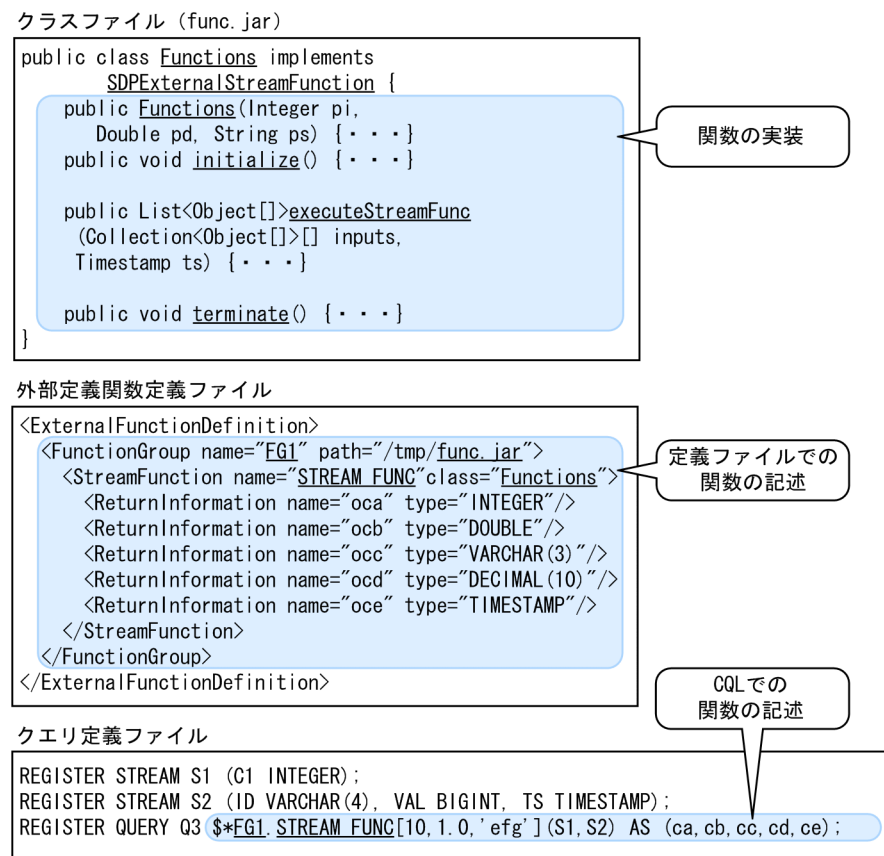
外部定義関数を実装したクラスとの関連づけを外部定義関数定義ファイルに記述して、SDP サーバに外部定義関数を認識させます。外部定義関数定義ファイルについては、マニュアル「uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド」を参照してください。

3. CQL での関数の記述

外部定義関数定義ファイルに記述した外部定義関数をクエリ定義ファイルの CQL に記述します。外部定義関数の CQL への記述方法については、「4.4.25 外部定義ストリーム間演算関数」を参照してください。

外部定義関数で使用するファイルの関連を次の図に示します。

図 10-1 外部定義関数で使用するファイルの関連



10.2 外部定義関数の実装

外部定義関数の実装では、外部定義関数の処理ロジックを、Java で記述して作成するクラスファイルにメソッドとして実装します。

ここでは、外部定義関数のクラスとメソッドの実装方法について説明します。また、外部定義関数の変更について説明します。

なお、外部定義関数インタフェースの詳細については、「11. 外部定義関数インタフェース」を参照してください。

10.2.1 外部定義関数のクラスの実装

外部定義関数のクラスを実装する場合の作成規則、および注意事項を説明します。

(1) 作成規則

外部定義関数を実装するクラスファイルは、次の規則に従って作成してください。

- 使用できるフィールド

外部定義関数を実装するクラスで、static フィールドを使用できます。

static フィールドの値は、同一のクエリグループ内で同じクラスファイルを使用する、すべての外部定義関数の使用個所で共有されます。ただし、クエリグループが異なる場合は、同じクラスファイルを使用しても static フィールドの値は個別に管理されます。

- パッケージ名、およびクラス名

パッケージ名、およびクラス名には、任意の名前を使用できます。

ただし、「jp.co.Hitachi.soft.sdp」から始まるパッケージ名は指定できません。

- コンストラクターの指定

外部定義関数を実装するクラスは、クエリ内で外部定義関数の使用個所ごとに、CQL の定義内容に一致するコンストラクターを使用して、クエリグループの登録・削除時にインスタンスを作成・破棄します。このため、次の条件を満たすように作成してください。

- クラスは、修飾子に必ず public を指定して作成してください。インスタンス化できない抽象クラス (abstract) や内部クラスは使用できません。

- コンストラクターは、修飾子に必ず public を指定し、CQL で指定する初期化パラメーターの数およびデータ型が一致するように作成してください。

ただし、プリミティブ型を引数とするコンストラクターは使用できません。

なお、CQL で初期化パラメーターを省略する場合には、コンストラクターを作成しないでデフォルトコンストラクターを使用するか、または引数を持たないコンストラクターを作成してください。

これらの条件を満たしていない場合、外部定義関数を実装するクラスのインスタンスを生成できないため、クエリグループの登録でエラーになります。

- 実装するインタフェース

外部定義関数を実装するクラスは、SDPEExternalStreamFunction インタフェースを実装してください。SDPEExternalStreamFunction インタフェースは、次のメソッドを実装する必要があります。

- executeStreamFunc メソッド

このメソッドには、外部定義関数の処理を実装してください。

- initialize メソッド

このメソッドには、外部定義関数を実装するクラスが持つフィールドの初期設定など、関数実行前の初期化処理を実装してください。

- terminate メソッド

このメソッドには、外部定義関数を実装するクラスが持つフィールドのクリアなど、関数実行後の終了処理を実装してください。

SDPEExternalStreamFunction インタフェースの詳細については、「11.3 SDPEExternalStreamFunction インタフェース」を参照してください。

(2) 注意事項

外部定義関数を実装するクラスファイルの注意事項を次に示します。

- 外部定義関数を実装するクラスファイルのパッケージ名を含むクラス名は、jvm_options.cfg の SDP_CLASS_PATH パラメーターに指定したクラスパスのクラスファイルと重複しないようにしてください。

重複した場合は、SDP_CLASS_PATH パラメーターに指定したクラスパスのクラスが優先して読み込まれるため、次に示すようなエラーが発生して外部定義関数が正しく動作しなくなるおそれがあります。

(例)

- SDPEExternalStreamFunction インタフェースの未実装、またはメソッドが存在しないなどの要因で、クエリグループの登録でエラーになる。
- SDP_CLASS_PATH パラメーターに指定したクラスパスのクラスのメソッドが外部定義関数として実行されてしまう。
- 外部定義関数を実装するクラスの中で、自身のクラスパス以外のクラスパスにあるクラスを参照している場合は、jvm_options.cfg の SDP_CLASS_PATH パラメーターにそのクラスパスを指定してください。

指定しない場合は、参照個所がフィールドやコンストラクターのときは、インスタンスの生成ができないため、クエリグループの登録でエラーになります。参照個所がメソッドのときは、そのメソッドの実行時に例外 java.lang.NoClassDefFoundError が発生してクエリグループが閉塞します。

特に、次の場合には、クエリグループが閉塞し、クエリグループを削除しないと再開できなくなるため注意してください。

- クエリグループの停止・閉塞または putEnd メソッドの処理の途中で実行する、terminate メソッドで例外が発生した。
- クエリグループの開始または putEnd メソッドの処理の途中で実行する、initialize メソッドで例外が発生した。

なお、jvm_options.cfg は、SDP サーバの起動後には変更できません。SDP サーバの起動前に jvm_options.cfg を編集してください。

- 外部定義関数を実装するクラスファイルのクラスパスは、jvm_options.cfg の SDP_CLASS_PATH パラメーターには指定しないでください。指定した場合は、次のようになるため注意してください。
 - SDP サーバの起動中に、クラスやメソッドの変更を反映できなくなります。
 - 外部定義関数を実装するクラスの static フィールドの値は、すべてのクエリグループで共有されます。
- 外部定義関数を実装するクラスの中で、標準出力および標準エラー出力への出力を行った場合、SDP サーバを実行している JavaVM プロセスに出力します。また、相対パスでファイル参照などを行った場合は、運用ディレクトリからの相対パスとして扱います。

- 外部定義関数はクエリ実行中に呼び出されるため、外部定義関数の実行時間が長いほど、ストリームデータ処理システムの処理性能が低下します。また、単位時間あたりに処理できるタプル数が減り、入力ストリームキューでキューあふれが発生する要因にもなります。このため、最長でも10ミリ秒程度で外部定義関数の処理が完了するように実装することをお勧めします。
- 外部定義関数でJNIを使用する場合、外部定義関数の実装クラスとは別にネイティブライブラリのロード専用のJavaクラスを作成し、そのクラスパスをjvm_options.cfgのSDP_CLASS_PATHパラメーターに指定してください。外部定義関数実装クラス内でネイティブライブラリをロードすると、クエリの登録に失敗するおそれがあります。

10.2.2 外部定義関数のメソッドの実装

外部定義関数のメソッドを実装する場合の作成規則、および注意事項を説明します。

(1) 作成規則

外部定義関数を実装するメソッドは、次の規則に従って作成してください。

- **修飾子とメソッド名**
メソッドは、修飾子に必ずpublicを指定し、名前は「executeStreamFunc」で作成してください。
- **メソッドの引数**
外部定義ストリーム間演算関数に使用するメソッドの引数は、次の二つです。
 - 第1引数：Collection<Object[]>型
メソッドの実行時、この引数には、クエリ定義ファイルで指定した入力ストリームごとに、同時に発生したタプルが格納されたデータオブジェクト配列を渡します。
 - 第2引数：Timestamp型
メソッドの実行時、この引数には、第1引数で渡されるタプルの時刻情報を渡します。
同時刻のタイムスタンプを持つタプルを複数回に分けてメソッドに渡す場合があるため、第2引数の時刻情報には、前回のメソッド実行時と同時刻のタイムスタンプが渡されることがあります。
- **メソッドの戻り値**
外部定義関数に使用するメソッドの戻り値は、List<Object[]>型とします。
 - メソッドの実行時、出力ストリームのタプルが格納されたデータオブジェクト配列のリストを返してください。出力対象のタプルが存在しない場合は、空のリストを返してください。
 - メソッドの戻り値 (List<Object[]>型) のObject[]に格納する各要素 (タプル) のデータ型は、CQLのデータ型に対応するJavaのデータ型をすべて使用できます。しかし、外部定義関数定義ファイルで指定する出力ストリームのカラムのデータ型 (CQLのデータ型) に対応する型で実装してください。
また、これらのデータ型でプリミティブ型と対応するラッパークラスを持つデータ型 (int など) は、プリミティブ型とラッパークラスのどちらも使用できます。
それ以外のデータ型を指定した場合は、クエリ実行時にエラーになります。
- **メソッドの戻り値の検証**
外部定義関数を使用するクエリの実行木は、外部定義関数定義ファイルに指定した戻り値の情報を基に生成されます。
クエリ実行時の外部定義関数の戻り値が、外部定義関数定義ファイルの戻り値の情報と一致していない場合、結果の不正や例外の発生などの意図しない動作が発生するおそれがあります。これらの現象を回避するため、双方の戻り値の情報が一致しているかどうかを検証できます。

戻り値の情報的一致を検証するかどうかは、`system_config.properties` の `engine.externalStreamFuncVerifyMode` パラメーターで指定します。

このパラメーターに `true` を指定し、検証結果が不正だった場合は、エラーメッセージが出力されて、クエリグループが閉塞されます。この場合は、実装メソッドの処理および外部定義関数定義ファイルを見直してください。

外部定義関数のクラスファイルまたは外部定義関数定義ファイルを修正する場合は、クエリグループの削除と再登録が必要です。外部定義関数の変更、またはクエリグループの変更については、マニュアル「`uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド`」を参照してください。

(2) 注意事項

外部定義関数を実装するメソッドの注意事項を次に示します。

- メソッドの実行時に、一つの入力ストリームに対して複数のタプルが入力された場合、そのタプル間の順序は不定となります。メソッドでは、引数で渡されるタプルの順序に依存しない処理を実装する必要があります。
- メソッドでは、次の処理を実行しないでください。
 - SDP サーバのプロセスを終了する処理 (`System#exit()`, `Runtime#halt()` など)
 - `Thread#currentThread()` など取得した SDP サーバのスレッドに対するメソッドの実行
 - 無限ループなど、メソッドが終了しない処理
 - SDP サーバに対して、直接 API (`put()` など) を発行する処理や、運用コマンドを実行する処理
 これらの処理を外部定義関数で実行した場合、SDP サーバの動作は保障されません。
- メソッドでは、新たにスレッドを起動する場合には、スレッドを停止させる処理も実装してください。
- メソッドでは、次に示すような事象の発生でクエリを続行できないと判断する場合には、戻り値を返さないで例外をスローしてください。
 - エラーの発生によって、次の関数の実行でもエラーが発生する確率が高く、関数が正しい戻り値を返せない場合。
 - 関数内での演算によって NaN が発生したなど、以降の関数の戻り値が NaN になって正しい戻り値を返せない場合。

例外をスローすると、メッセージ `KFSP42401-E` を出力して、クエリグループを閉塞します。

なお、今回は一時的な要因で正しい戻り値を返せなくても、次の関数の実行では正常に動作可能と判断できる場合には、例外をスローしないで空のリストを返すことで、クエリ処理を継続できます。

- メソッドの戻り値、および戻り値の各要素として `null` を返さないでください。`null` を返した場合、メッセージ `KFSP42400-E` または `KFSP42403-E` を出力して、クエリグループを閉塞します。
- メソッドでは、長さを指定できるデータ型を返すときに、定義値を超えるデータを返さないでください。返した場合には、メッセージを出力して、クエリグループを閉塞します。

ただし、`system_config.properties` の `engine.externalStreamFuncVerifyMode` パラメーターに `false` を指定した場合には、外部定義関数が返したデータをクエリ実行結果としてそのまま利用するため、メソッドを実装するときに注意してください。

- クエリ定義ファイルで外部定義関数の引数に指定した入力ストリームの数が、実装したメソッドの処理内容に対応しているかは SDP サーバではチェックしません。引数のチェック処理が必要な場合は、メソッド内でチェック処理を実装してください。

10.2.3 外部定義関数の変更

実装したメソッドの処理内容が原因で障害が発生した場合などに、クラスやメソッドを変更するときは、クエリグループの削除と再登録が必要です。

外部定義関数定義ファイルに指定した外部定義関数の実装クラスとメソッドは、クエリグループの登録時に読み込まれます。このため、外部定義関数の実装クラスとメソッドの変更は、クエリグループを削除して再登録することで反映されます。再登録をしていないクエリグループには、メソッドの変更は反映されません。

外部定義関数の変更、またはクエリグループの変更については、マニュアル「uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド」を参照してください。

10.3 外部定義関数の作成例

外部定義ストリーム間演算関数を使用する場合の、外部定義関数の作成例を次に示します。

外部定義関数のクラスファイルの作成例

```
package samples;

import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import jp.co.Hitachi.soft.sdp.plugin.function.stream.SDPEExternalStreamFunction;

public class ExternalStreamFunction implements SDPEExternalStreamFunction {
    // 合計値格納マップ
    Map<String, Long> tempMap = new HashMap<String, Long>();
    // 乗数
    int mul;
    // 実行回数
    int loopCount;

    // 外部定義関数の実装クラスのコンストラクタ。
    // mul:乗数。クエリに記述した外部定義関数の初期化パラメーターに指定した値。
    public ExternalStreamFunction(Integer mul) {
        this.mul = mul;
        loopCount = 0;
    }

    // 外部定義関数の処理を実装するメソッド。
    // ID毎にVALの合計値を保持し、乗数で乗算した結果を出力します。
    // inputs:タプルの集合。入力ストリーム毎にタプルが格納されています。
    // ts :inputsに格納されているタプルの時刻(タイムスタンプ)。
    public List<Object[]> executeStreamFunc(Collection<Object[]>[] inputs, Timestamp ts) {
        // 戻り値格納リスト
        List<Object[]> res = new ArrayList<Object[]>();
        // 入力ストリーム数分ループ
        for (int i = 0; i < inputs.length; i++) {
            // 入力ストリームを取得します
            Collection<Object[]> values = inputs[i];
            // タプル数分ループ
            for (Object[] value : values) {
                // IDを取得します
                String id = (String) value[0];
                // VALを取得します
                Long val = (Long) value[1];
                // 合計値をID毎に合計値格納マップに保存します
                if (tempMap.containsKey(id)) {
                    // 既出IDの場合、加算します
                    tempMap.put(id, tempMap.get(id) + val);
                } else {
                    // 新規IDの場合、新規に登録します
                    tempMap.put(id, val);
                }
            }
        }
        // IDの数分ループ
        for (String key : tempMap.keySet()) {
            // 戻り値格納リストに出力タプルを格納します
            res.add(new Object[] { loopCount, key, tempMap.get(key) * mul, ts });
        }
        loopCount++;
        return res;
    }
    // 初期化メソッド
    public void initialize() {
        tempMap.clear();
        loopCount = 0;
    }
}
```

```

// 終了メソッド
public void terminate() {
    // 処理なし
}
}

```

外部定義関数定義ファイルの作成例

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- All Rights Reserved. Copyright (C) 2013, Hitachi, Ltd. -->
<root:ExternalFunctionDefinition
xmlns:root="http://www.hitachi.co.jp/soft/xml/sdp/function"
xmlns:group="http://www.hitachi.co.jp/soft/xml/sdp/function/functiongroup">
  <!-- 外部定義関数グループ定義 -->
  <group:FunctionGroup name="FG1" path="samples/external/src">
    <!-- 関数定義 -->
    <group:StreamFunction name="FUNC1" class="samples.ExternalStreamFunction">
      <!-- 戻り値定義 -->
      <group:ReturnInformation name="R1" type="INT" />
      <group:ReturnInformation name="R2" type="VARCHAR(10)" />
      <group:ReturnInformation name="R3" type="BIGINT" />
      <group:ReturnInformation name="R4" type="TIMESTAMP(9)" />
    </group:StreamFunction>
  </group:FunctionGroup>
</root:ExternalFunctionDefinition>

```

クエリ定義ファイルの作成例

```

REGISTER STREAM DATA0(ID VARCHAR(10), VAL BIGINT);

// 入力ストリームDATA0から、カラムID毎に最新のタプルを保持するクエリ
REGISTER QUERY Q1 SELECT ID, VAL FROM DATA0[PARTITION BY ID ROWS 1];
// 最新のタプルを出力するクエリ
REGISTER QUERY Q2 ISTREAM (SELECT * FROM Q1);
// 直前のタプルを出力するクエリ
REGISTER QUERY Q3 DSTREAM (SELECT * FROM Q1);
// 入力タプルのカラムID毎にカラムVALの値を合計し、乗数3で乗算した結果を出力するクエリ
REGISTER QUERY SUM1 $*FG1.FUNC1[3](Q2, Q3);

```

上記の作成例での入力ストリーム DATA0 と出力ストリーム SUM1 の関係を、一例として次の図に示します。

図 10-2 入力ストリーム DATA0 と出力ストリーム SUM1 の関係



図中のクエリでは、次の四つの列を持つ出力ストリーム SUM1 を出力します。

- R1 (1 列目)：外部定義関数の呼び出し回数-1
- R2 (2 列目)：カラム ID
- R3 (3 列目)：カラム ID ごとに VAL の値を合計して 3 で乗算した値
- R4 (4 列目)：時刻 (タプルのタイムスタンプ)

図に示した一例での executeStreamFunc メソッドの引数を次に示します。

10 外部定義関数の作成

外部定義関数の呼び出し回数	第 1 引数				第 2 引数
	Collection<Object[]>[0] (出カストリーム Q2)		Collection<Object[]>[1] (出カストリーム Q3)		Timestamp (タプルのタイムスタンプ)
	Object[0] (ID)	Object[1] (VAL)	Object[0] (ID)	Object[1] (VAL)	
1 回目	A	100	なし	なし	10:00:00
2 回目	B	200	A	100	10:00:01
	A	300	なし	なし	

11 外部定義関数インタフェース

この章では, 外部定義関数の作成で使用する外部定義関数インタフェースの文法について説明します。

11.1 外部定義関数インタフェースの記述形式

ここでは、外部定義関数インタフェースの記述形式について説明します。

各インタフェースで説明する項目は次のとおりです。ただし、インタフェースによっては説明しない項目もあります。

形式

インタフェースの記述形式を示します。

説明

インタフェースの機能について説明します。

パラメーター

インタフェースのパラメーターについて説明します。

例外

インタフェースを利用する際に発生する例外について説明します。

戻り値

インタフェースの戻り値について説明します。

注意事項

インタフェースを利用する上での注意事項について説明します。

11.2 外部定義関数インタフェースの一覧

外部定義関数インタフェースの一覧を次に示します。

表 11-1 外部定義関数インタフェースの一覧

項番	パッケージ名	インタフェース名	機能
1	jp.co.Hitachi.soft.sdp.plugin.function.stream	SDPExternalStreamFunction	外部定義ストリーム間演算関数を実装するクラスが継承するインタフェースです。

11.3 SDPExternalStreamFunction インタフェース

説明

外部定義ストリーム間演算関数を実装するクラスが継承するインタフェースです。ストリーム間演算関数の処理の抽象メソッドを実装します。

メソッド

SDPExternalStreamFunction インタフェースのメソッド一覧を次に示します。

戻り値	メソッド名	機能
List<Object[]>	executeStreamFunc(Collection<Object[]>[] ObjectArrayCollection, Timestamp timestamp)	外部定義ストリーム間演算関数の処理を実装します。
void	initialize()	外部定義関数の実装クラスが持つフィールドなどの初期化処理を実装します。
void	terminate()	外部定義関数の実装クラスが持つフィールドのクリアなど、終了処理を実装します。

executeStreamFunc(Collection<Object[]>[] ObjectArrayCollection, Timestamp timestamp)メソッド

形式

```
abstract List<Object[]> executeStreamFunc(Collection<Object[]>[] ObjectArrayCollection, Timestamp timestamp)
```

説明

外部定義ストリーム間演算関数の処理を実装する抽象メソッドです。

SDP サーバは、クエリ定義で指定した外部定義関数の入力ストリームにタプルが到着したときに、このメソッドを実行します。

パラメーター

ObjectArrayCollection

Collection<Object[]>[]型で、入力ストリーム別に、同時刻に発生したタプルが格納されたデータオブジェクト配列を指定します。

timestamp

Timestamp 型で、ストリームデータのタプルの時刻情報を指定します。

例外

なし。

戻り値

出カストリームが格納されたデータオブジェクト配列のリスト (List <Object[]>型オブジェクト)。

注意事項

一つの入カストリームに対して複数のタプルが入力された場合、そのタプル間の順序は不定となります。メソッドでは、引数で渡されるタプルの順序に依存しない処理を実装する必要があります。

initialize()メソッド

形式

```
abstract void initialize()
```

説明

外部定義関数の実装クラスが持つフィールドなどの初期化処理を実装する抽象メソッドです。

SDP サーバは、外部定義関数の使用個所ごとに実装クラスのインスタンスを作成し、次の契機でこのメソッドを実行します。

- クエリグループを開始したとき
- StreamInput インタフェースの putEnd メソッドを実行したときの terminate メソッド実行後

パラメーター

なし。

例外

なし。

戻り値

なし。

注意事項

なし。

terminate()メソッド

形式

```
abstract void terminate()
```

説明

外部定義関数の実装クラスが持つフィールドのクリアなど、終了処理を実装する抽象メソッドです。

SDP サーバは、外部定義関数の使用個所ごとに実装クラスのインスタンスを作成し、次の契機でこのメソッドを実行します。

- クエリグループを停止または閉塞したとき
- クエリグループのすべての入力ストリームに StreamInput インタフェースの putEnd メソッドを実行したとき

パラメーター

なし。

例外

なし。

戻り値

なし。

注意事項

このメソッドでクリアしないフィールドは、クエリグループを削除するまで値の変更が保持されます。

付録

付録 A 各バージョンの変更内容

変更内容 (3020-3-V03-10) uCosminexus Stream Data Platform - Application Framework 01-05

追加・変更内容

リレーションを生成しないで直接ストリームデータに対して演算（ストリーム間演算）を実行できるようにした。

また、CQL で、ユーザーが Java でプログラミングした外部定義関数を使用できるようにした。

ストリームデータに対する演算機能の追加に伴い、操作系 CQL に次の関数を追加した。

- ストリーム間演算関数
- 外部定義ストリーム間演算関数

次のファイルを追加した。

- 外部定義関数定義ファイル

また、予約語としてシステムに登録されている文字列を追加した。

操作系 CQL に次の関数を追加した。

- 組み込み集合関数
- スカラ関数
- 組み込みスカラ関数

これに伴い、予約語としてシステムに登録されている文字列を追加した。

DECIMAL 型および NUMERIC 型についての注意事項で、`query.decimalRoundingMode` パラメーターの指定値を変更した。

StreamInprocessUP インタフェースの説明で、String 型の可変長引数を持つコンストラクターの実装に関する注意事項を追加した。

uCosminexus Stream Data Platform - Application Framework 01-01

追加・変更内容

適用 OS に Linux を追加した。

日付データの指定で、年、月、日に、範囲外の値を指定した場合の注意事項を変更した。

英語版のサンプルファイルを追加した。

付録 B このマニュアルの参考情報

このマニュアルを読むに当たっての参考情報を示します。

付録 B.1 関連マニュアル

関連マニュアルを次に示します。必要に応じてお読みください。

- ストリームデータ処理基盤 uCosminexus Stream Data Platform - Application Framework 解説 (3020-3-V01)
 Stream Data Platform - AF の概要や前提知識について説明しています。
 Stream Data Platform - AF の特長やシステム構成などの概要、およびシステムを構築・運用するために必要な前提知識を習得したい場合に参照してください。
- ストリームデータ処理基盤 uCosminexus Stream Data Platform - Application Framework システム構築・運用ガイド (3020-3-V02)
 Stream Data Platform - AF の設計・構築・運用方法、および構築時に設定できる機能の詳細について説明しています。
 Stream Data Platform - AF のシステムを設計・構築・運用して、ストリームデータの分析を実施する場合に参照してください。
- ストリームデータ処理基盤 uCosminexus Stream Data Platform - Application Framework メッセージ (3020-3-V04)
 Stream Data Platform - AF が出力するメッセージについて説明しています。
 メッセージが出力された際に、必要に応じて参照してください。

なお、このマニュアルでは、関連マニュアルについて、「ストリームデータ処理基盤」を省略して表記しています。

付録 B.2 このマニュアルでの表記

このマニュアルでは、製品名および Java 関連用語を次のように表記しています。

表記		製品名または Java 関連用語
jar		Java™ Archive
Java		Java™
JavaVM		Java™ Virtual Machine
JDBC		Java™ Database Connectivity
JDK		Java™ Development Kit
JNI		Java™ Native Interface
Linux		Linux(R)
Linux	Red Hat Enterprise Linux 5	Red Hat Enterprise Linux(R) 5 Advanced Platform (AMD/Intel 64) Red Hat Enterprise Linux(R) 5 (AMD/Intel 64)

表記		製品名または Java 関連用語
Linux	Red Hat Enterprise Linux 6	Red Hat Enterprise Linux(R) Server 6 (64-bit x86_64)
Stream Data Platform - AF		uCosminexus Stream Data Platform - Application Framework

付録 B.3 英略語

このマニュアルで使用する英略語を次に示します。

英略語	英語での表記
AP	Application Program
API	Application Programming Interface
CPU	Central Processing Unit
CQL	Continuous Query Language
FIFO	First-In First-Out
HTTP	Hyper Text Transfer Protocol
IEEE	Institute of Electrical and Electronic Engineers
RMI	Remote Method Invocation
UCS	Universal multi-octet coded Character Set
UTF	UCS Transformation Format

付録 B.4 KB (キロバイト) などの単位表記について

1KB (キロバイト), 1MB (メガバイト), 1GB (ギガバイト), 1TB (テラバイト) はそれぞれ $1,024$ バイト, $1,024^2$ バイト, $1,024^3$ バイト, $1,024^4$ バイトです。

索引

記号

\$* 93
* 70, 88

数字

10 進定数 45

A

ABS 関数 108
ACOS 関数 109
ALL 73
API の一覧 199
ASIN 関数 109
ATAN2 関数 112
ATAN 関数 111
AVG 89

B

BIGINT_TOSTRING 関数 162

C

CEIL 関数 116
close()メソッド [SDPConnector インタフェース] 202
close()メソッド [StreamInput インタフェース] 210
close()メソッド [StreamOutput インタフェース] 217
CONCAT 関数 148
connect()メソッド [SDPConnectorFactory クラス] 205
CORREL 関数 99
COSH 関数 118
COS 関数 117
COUNT 88
COVAR_POP 関数 101
COVAR 関数 100
CQL 10
CQL で使用できる文字 37
CQL の一覧 59
CQL の記述形式 36
CQL の基本項目の指定方法 39
CQL の体系 10
CQL のデータ型 49

CQL のデータ型と Java のデータ型のマッピング 49
CQL 文法説明で使用する記号 38
CQL リファレンス 57

D

DAY 80
DECIMAL_TOSTRING 関数 162
DECIMAL 型および NUMERIC 型についての注意事項 51
DISTANCE3 関数 120
DISTANCE 関数 119
DOUBLE_TOSTRING 関数 163
DSTREAM 25, 68

E

equals(Object obj)メソッド [StreamTuple クラス] 231
equals(StreamTime when)メソッド [StreamTime クラス] 227
execute(SDPConnector con)メソッド [StreamInprocessUP インタフェース] 208
executeStreamFunc(Collection<Object[]>[] ObjectArrayCollection, Timestamp timestamp)メソッド [SDPExternalStreamFunction インタフェース] 266
EXP 関数 121

F

FLOOR 関数 122
FROM 句 70

G

get()メソッド [StreamOutput インタフェース] 217
get(int count, long timeout)メソッド [StreamOutput インタフェース] 219
get(int count)メソッド [StreamOutput インタフェース] 218
getAll()メソッド [StreamOutput インタフェース] 221
getAll(long timeout)メソッド [StreamOutput インタフェース] 222
getDataArray()メソッド [StreamTuple クラス] 232
getFreeQueueSize()メソッド [StreamInput インタフェース] 211

getFreeQueueSize()メソッド [StreamOutput インタフェース] 223
 getMaxQueueSize()メソッド [StreamInput インタフェース] 211
 getMaxQueueSize()メソッド [StreamOutput インタフェース] 224
 getSystemTime()メソッド [StreamTuple クラス] 232
 getTimeMillis()メソッド [StreamTime クラス] 228
 GROUP BY 句 72

H

hashCode()メソッド [StreamTime クラス] 228
 hashCode()メソッド [StreamTuple クラス] 233
 HAVING 句 72
 HOUR 80

I

initialize()メソッド [SDPExternalStreamFunction インタフェース] 267
 INT_TOSTRING 関数 164
 isClosed()メソッド [SDPConnector インタフェース] 202
 isStarted()メソッド [StreamInput インタフェース] 212
 ISTREAM 24, 68

J

jar 195
 javac 195
 Java のデータ型 49
 jp.co.Hitachi.soft.sdp.api 199, 200
 jp.co.Hitachi.soft.sdp.api.inprocess 199
 jp.co.Hitachi.soft.sdp.common.data 200
 jp.co.Hitachi.soft.sdp.common.util 200
 jp.co.Hitachi.soft.sdp.plugin.function.stream 265

L

LENGTH 関数 149
 LEQ 関数 150
 LGE 関数 151
 LGT 関数 152
 LLE 関数 152
 LLT 関数 153
 LN 関数 123
 LOG 関数 124

M

MAX 89
 MILLISECOND 80
 MIN 89
 MINUTE 80
 MOD 関数 127

N

NAN 関数 127
 NEGATIVE_INFINITY 関数 128
 NOW 79
 NOW ウィンドウ 14
 NUMBER_TODATE 関数 164
 NUMBER_TOTIME 関数 165

O

onEvent(StreamTuple tuple)メソッド [StreamEventListener インタフェース] 206
 openStreamInput(String group_name,String stream_name)メソッド [SDPConnector インタフェース] 203
 openStreamOutput(String group_name,String stream_name)メソッド [SDPConnector インタフェース] 203

P

PARTITION BY 79
 PARTITION BY ウィンドウ 14
 PI 関数 128
 POSITIVE_INFINITY 関数 129
 POWER 関数 129
 put(ArrayList<StreamTuple> tuple_list)メソッド [StreamInput インタフェース] 213
 put(StreamTuple tuple)メソッド [StreamInput インタフェース] 214
 putEnd()メソッド [StreamInput インタフェース] 215

R

RANGE 79
 RANGE ウィンドウ 13, 29
 REAL_TOSTRING 関数 166
 REGEXP_FIRSTSEARCH 関数 154
 REGEXP_REPLACE 関数 155
 REGEXP_SEARCH 関数 156
 registerForNotification(StreamEventListener n)メソッド [StreamOutput インタフェース] 225

REGISTER QUERY_ATTRIBUTE 句 63
 REGISTER QUERY 句 62
 REGISTER STREAM 句 61
 RMI 連携カスタムアダプター 172
 RMI 連携カスタムアダプターの作成 175
 RMI 連携カスタムアダプターのサンプルプログラム
 240
 RMI 連携カスタムアダプターのサンプルプログラムの
 実行手順 240
 RMI 連携データ受信 AP の内容 243
 RMI 連携データ送信 AP の内容 241
 RMI 連携用 API 199
 ROUND 関数 138
 ROWS 79
 ROWS ウィンドウ 13
 RSTREAM 25, 68

S

SDPClientException クラス 234
 SDPClientException クラスのサブクラス 234
 SDPClientFreeInputQueueSizeLackException ク
 ラス 236
 SDPClientFreeInputQueueSizeThresholdOverEx
 ception クラス 235
 SDPConnectorFactory クラス 205
 SDPConnectorFactory クラスのメソッド一覧 205
 SDPConnector インタフェース 201
 SDPConnector インタフェースのメソッド一覧 201
 sdpcqlstop コマンドによるクエリグループ停止の流
 れ 185
 sdpcqlstop コマンドの実行によるクエリグループ停
 止の通知 184
 SDPExternalStreamFunction インタフェース 266
 SECOND 80
 SELECT 句 69
 SINH 関数 140
 SIN 関数 139
 SPLNGTH 関数 157
 SQRT 関数 141
 STDDEV_POP 関数 103
 STDDEV 関数 102
 stop()メソッド [StreamInprocessUP インタフェー
 ス] 208
 StreamEventListener インタフェース 206
 StreamEventListener インタフェースのメソッド一
 覧 206
 StreamInprocessUP インタフェース 207
 StreamInprocessUP インタフェースのメソッド一覧
 207

StreamInput インタフェース 175, 178, 210
 StreamInput インタフェースのメソッド一覧 210
 StreamOutput インタフェース 175, 178, 216
 StreamOutput インタフェースのメソッド一覧 216
 StreamTime クラス 227
 StreamTime クラスのメソッド一覧 227
 StreamTuple(Object[] dataArray)コンストラク
 ター 231
 StreamTuple クラス 230
 StreamTuple クラスのメソッド一覧 230
 SUM 89

T

TANH 関数 143
 TAN 関数 142
 terminate()メソッド [SDPExternalStreamFunction
 インタフェース] 267
 TIME_TONUMBER 関数 166
 TIMEDIFF 関数 158
 TIMESTAMP_TONUMBER 関数 167
 TIMESTAMPDIFF 関数 159
 TODEGREES 関数 144
 TORADIANS 関数 145
 toString()メソッド [StreamTime クラス] 229
 toString()メソッド [StreamTuple クラス] 233

U

UNION 句 73
 unregisterForNotification(StreamEventListener n)
 メソッド [StreamOutput インタフェース] 225

V

VAR_POP 関数 105
 VAR 関数 104

W

WHERE 句 71

あ

値式 84
 値式一次子 85
 値の丸め方 52
 アプリケーション連携方式 172

い

一般集合関数 89
 インプロセス連携カスタムアダプター 173

インプロセス連携カスタムアダプターの作成 178
 インプロセス連携カスタムアダプターのサンプルプログラム 246
 インプロセス連携カスタムアダプターのサンプルプログラムの実行手順 246
 インプロセス連携送受信制御 AP の内容 247
 インプロセス連携データ受信 AP の内容 (ポーリング方式) 251
 インプロセス連携データ送信 AP の内容 249
 インプロセス連携用 API 199

う

ウィンドウ演算 13
 ウィンドウ演算の種類 13
 ウィンドウ指定 79

え

演算子 85
 演算の関係 11

お

オペランド 36

か

改行コード 40
 外部定義関数インタフェース 263
 外部定義関数定義ファイル 254
 外部定義関数の概要 254
 外部定義関数の作成例 260
 外部定義関数の実装 255
 外部定義関数の変更 259
 外部定義ストリーム間演算関数 93
 カスタムアダプター 172
 カスタムアダプター作成時の留意事項 196
 カスタムアダプターで実装する処理 182
 カスタムアダプターの位置づけ 172
 カスタムアダプターの作成 171
 カスタムアダプターの作成でできること 7
 カスタムアダプターの種類 172
 数定数 44, 45
 数データ 49
 数データの比較 54
 環境変数 195
 環境変数 PATH 195
 関係演算 19
 関係演算の種類 19
 関数グループ名 93
 関数名 93

関連マニュアルとの関係 2

き

キーワード 36, 39
 キーワードの指定 39
 キーワードの指定例 39
 基本項目 36
 基本項目の指定方法 39
 キャスト指定 85
 キャストによって数データと比較できる文字データの書式 54
 キューあふれが発生した場合の動作 188
 キューあふれの事前防止 188
 共通 API 199

く

クエリ結果データの受信 (RMI 連携カスタムアダプター) 176
 クエリ結果データの受信 (インプロセス連携カスタムアダプター) 179
 クエリ定義での制限値 55
 クエリ定義での注意事項 55
 クエリ定義での注意事項および制限値 55
 クエリ定義のサンプル 169
 クエリの定義 62
 クエリの定義でできること 6
 クエリの定義例 31
 区切り文字の種類 40
 区切り文字の挿入 40
 区切り文字の挿入例 41
 区切り文字を挿入する位置 41
 区切り文字を挿入できない位置 41
 区切り文字を挿入できる位置 41
 組み込み関数の一覧 96
 組み込み集合関数 90
 組み込み集合関数の一覧 96
 組み込みスカラ関数 91
 組み込みスカラ関数の一覧 96

け

結合 19
 結合処理と ROWS ウィンドウを併用する場合の注意事項 23
 結合処理の例 19

こ

項 85
 コールバック方式 178, 180

このマニュアルの構成 4
コンパイル手順 195

さ

サンプルプログラムの構成 238

し

時間指定 80
時間による指定の例 16
時刻印データ 50
時刻印データの規定の文字列表現 47
時刻解像度の指定 29, 63
時刻解像度を指定した定義例 32
時刻関数 158
時刻データ 50
時刻データの規定の文字列表現 46
集合関数 19, 88
集合関数による演算処理の例 21
修飾 42
出力ストリーム 62
出力ストリームキュー 188
出力ストリームキューのキューあふれが事前に防止される処理 193
出力ストリームキューのキューあふれが発生した場合の動作 189
出力ストリームキューのキューあふれの事前防止 192
出力ストリームデータへの変換 24
出力リレーション 11
出力リレーションから削除されたタプルを出力する指定の例 26
出力リレーション内のタプルの集合を一定間隔で出力する指定の例 27
出力リレーションに追加されたタプルを出力する指定の例 25
使用できる文字 37
使用できる文字コード 37

す

数学関数 106
数値の指定 40
スカラ関数 91
ストリーム化演算 24
ストリーム化演算の種類 24
ストリーム化演算の例 25
ストリーム間演算関数 92
ストリーム句 68
ストリームデータ受信時 175, 178
ストリームデータ送信時 175, 178

ストリームデータの送信 (RMI 連携カスタムアダプター) 175
ストリームデータの送信 (インプロセス連携カスタムアダプター) 178
ストリームの定義 61
ストリーム名 77

せ

整数定数 45
選択 19
選択式 75
選択式に相関名を指定した場合 44
選択リスト 74

そ

相関名 44, 75
操作系 CQL 10, 67
操作系 CQL の一覧 59

た

タブコード 40
タプル送信時の時刻情報の制御 186
タプル入力完了後の処理 187
タプルへの時刻情報の設定 186
探索条件 81

て

定義系 CQL 10, 61
定義系 CQL の一覧 59
定数 44, 87
定数の種類 44
定数の表記方法 45
データ型一般の比較 53
データ識別子 42
データ識別子指定 42
データ識別子の有効範囲 43
データ受信 AP 172
データ受信 AP の終了契機の把握 (データ処理終了の通知) 182
データ受信 AP の終了契機の把握方法 182
データ送受信信用 API 197
データ送受信信用 API を使用したサンプルプログラム 237
データ送信 AP 172
データ送信 AP とデータ受信 AP で異なるデータ送受信方法を使用する場合 174
データ送信 AP によるデータ送信終了通知の流れ 183

データ送信 AP のメソッド呼び出し (putEnd メソッド) によるデータ送信終了の通知 183
 データソースモードでの時刻情報の設定 185
 データの数による指定の例 16
 データのグループによる指定の例 18
 データの送受信方法による分類 172
 データの抽出 19
 データの比較 53
 データを比較する際の留意点 53

と

問い合わせ 67
 統計関数 99
 到着したタプルのタイムスタンプ時刻の指定による例 17
 導入から運用までの流れ 2

な

名前の指定 42
 名前の修飾 42

に

入力ストリーム 61
 入力ストリームキュー 188
 入力ストリームキューでキューあふれが発生した場合の動作 188
 入力ストリームキューのキューあふれの事前防止 189
 入力ストリームキューのキューあふれを事前に防止するデータ送信 AP の実装方法 190
 入力リレーション 11
 入力リレーションの生成 13

は

範囲変数 43
 半角空白 40

ひ

比較演算項 83
 比較演算子 83
 比較述語 83
 比較できるデータ型の組み合わせ 53
 日付, 時刻, および時刻印データの比較 54
 日付データ 50
 日付データの規定の文字列表現 46
 表記方法 44
 表識別子 43

ふ

複数の入力ストリームに対して複数のデータ送信 AP からタプルを送信する場合の処理順序 187
 符号 85
 復帰コード 40
 浮動小数点定数 45
 フリーフォーマット形式 36

へ

変換関数 161

ほ

ポーリング方式 175, 178, 179

ま

マッピング 49

め

メッシュ 29
 メッシュ間隔 29
 メモリ使用量増加の抑止 29

も

文字データ 50
 文字データの比較 53
 文字列関数 147
 文字列定数 44, 45, 87

よ

予約語としてシステムに登録されている文字列 39

り

リスナーオブジェクト 216
 リレーション 11
 リレーション参照 77
 リレーション参照に相関名を指定した場合 44
 リレーション式 69
 リレーション名 77

れ

例外クラス 234
 列指定 42
 列指定リスト 76
 列名 75