



PostgreSQL クエリパフォーマンスの最適化

AWS 規範ガイド



AWS 規範ガイド: PostgreSQL クエリパフォーマンスの最適化

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

序章	1
クエリパフォーマンスチューニングのユースケース	1
EXPLAIN プラン	2
EXPLAIN ステートメント	2
EXPLAIN ANALYZE の使用	2
EXPLAIN クエリプランの読み方	2
.....	4
照合	13
データ型の不一致	16
SELECT での関数呼び出し	18
IN または EXISTS	19
サブクエリまたは CTEs	22
よくある質問	26
EXPLAIN とは	26
EXPLAIN ANALYZE とは	26
PostgreSQL での照合とは	27
CTE とは	27
PostgreSQL の関数のカテゴリは何ですか？	27
.....	29
寄稿者	30
ドキュメント履歴	31
用語集	32
#	32
A	33
B	35
C	37
D	40
E	44
F	47
G	48
H	49
I	51
L	53
M	54

O	58
P	61
Q	64
R	64
S	67
T	71
U	72
V	73
W	73
Z	74
.....	lxxv

PostgreSQL クエリパフォーマンスの最適化

Amazon Web Services ([寄稿者](#))

2024 年 4 月 ([ドキュメント履歴](#))

PostgreSQL は、強力、柔軟、信頼性の高いオープンソースのオブジェクトリレーショナルデータベースシステムです。PostgreSQL クエリのパフォーマンスを最適化するには、さまざまな方法があります。クエリを最適化するプロセスは、ユースケースによって異なります。現在のクエリプランを把握しておく、問題を特定して理解し、必要な変更を加えるのに役立ちます。データベース統計を最新の状態に保つために、テーブルの分析が必要になる場合があります。PostgreSQL オプティマイザは、これらの統計を使用してクエリをより速く実行します。このガイドでは、PostgreSQL クエリのパフォーマンスを向上させるためのベストプラクティスに焦点を当てています。

このガイドでは、PostgreSQL または Amazon Aurora PostgreSQL 互換データベースインスタンス用の既存の Amazon Relational Database Service (Amazon RDS) PostgreSQL 前提としています。

クエリパフォーマンスチューニングのユースケース

このガイドでは、説明と例を含む 5 つのユースケースについて説明します。

- 照合
- データ型の不一致
- SELECT ステートメントの関数呼び出し
- IN または EXISTS
- サブクエリまたは共通テーブル式 (CTEs)

各ユースケースには、初期実行計画の詳細、計画を分析して問題を特定する方法、および解決策が記載されています。これらのユースケースを実装すると、通常、クエリの応答時間が短縮され、サーバーの負荷が軽減され、全体的なシステム効率が向上します。これらの改善により、ユーザーエクスペリエンスが向上し、システムの信頼性が向上します。

EXPLAIN クエリプラン

PostgreSQL には、クエリの実行方法の詳細を含むクエリプランを返すための EXPLAIN および EXPLAIN ANALYZE オプションが用意されています。

EXPLAIN ステートメント

EXPLAIN ステートメントは、PostgreSQL プランナーが特定のステートメントに対して生成するクエリプランを返します。クエリプランには、以下が表示されます。

- ステートメントに関係するテーブルのスキャン方法 (インデックススキャンやシーケンシャルスキャンなど)
- 複数のテーブルを結合する方法 (ハッシュ結合、マージ結合、ネストされたループ結合など)

クエリのパフォーマンスを向上させるには、計画を理解することが不可欠です。計画を理解したら、クエリに時間がかかりすぎている箇所に集中し、時間を短縮するためのアクションを実行できます。

EXPLAIN ANALYZE の使用

PostgreSQL では、EXPLAIN は指定されたステートメントの計画のみを生成します。ANALYZE キーワードを追加すると、EXPLAIN はプランを返し、クエリを実行し、各ステップの実際のランタイムと行数を表示します。これはクエリのパフォーマンスを分析するために不可欠です。

Important

を使用する場合は EXPLAIN ANALYZE、INSERT、UPDATE および DELETE に注意してください。

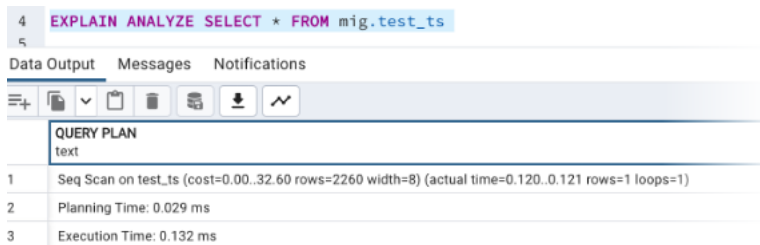
EXPLAIN クエリプランの読み方

PostgreSQL クエリプランは、複数のノードで構成されるツリー構造です。EXPLAIN クエリプランには、データベースエンジンがクエリの実行に使用するステップが表示されます。クエリプランには、次の情報が表示されます。

- シーケンシャルスキャン、インデックススキャン、ネストされたループ結合など、実行されるオペレーションのタイプ。

- 実行中のオペレーションを記述する Seq Scan Index Scan、Nested Loop、などのラベル。
- クエリによって処理されるテーブルまたはインデックスの名前。
- 任意の計算単位の推定コストと処理された行数に関する情報を含むコストと行列。
- 条件など、オペレーションに適用されるフィルターのフィルターwhere条件。
- ステップを視覚的に表現し、各オペレーションをノードとして表示し、オペレーションを接続する矢印を示します。オペレーションの順序は左から右に示され、以前のオペレーションは後のオペレーションにフィードされます。

次のスクリーンショットは、シーケンシャルスキンのクエリプランを示しています。



```
4 EXPLAIN ANALYZE SELECT * FROM mig.test_ts
5
Data Output Messages Notifications
QUERY PLAN
text
1 Seq Scan on test_ts (cost=0.00..32.60 rows=2260 width=8) (actual time=0.120..0.121 rows=1 loops=1)
2 Planning Time: 0.029 ms
3 Execution Time: 0.132 ms
```

コスト見積もりは、PostgreSQL がクエリが結果を返すために 32.60 単位の計算を必要とすることを想定している (cost=0.00..32.60 rows=2260 width=8) ことを意味します。

0.00 値は、このノードが動作を開始できるコストです (この場合は、クエリの起動時間)。rows 値は、シーケンシャルスキンの返す推定行数です。width 値は、返された行のバイト単位の推定サイズです。

この例では ANALYZE オプション EXPLAIN を指定しているため、クエリが実行され、タイミング情報がキャプチャされました。結果は次の (actual time=0.120..0.121 rows=1 loops=1) ことを意味します。

- シーケンシャルスキンの 1 回 (loops 値) 実行されました。
- スキンは 1 行を返しました。
- 実際の時間は 0.12 ミリ秒でした。

クエリを調整するためのユースケース

このガイドでは、クエリパフォーマンスを調整するための以下のユースケースについて説明します。

- 照合
- データ型の不一致
- SELECT ステートメントの関数呼び出し
- IN、または EXISTS
- サブクエリまたは共通テーブル式 (CTEs)

これらのクエリパフォーマンスのユースケースのパフォーマンスチューニングをテストするには、既存のデータベースと、このガイドで提供されているサンプルデータを使用します。この例では、架空の XX 航空会社のデータを使用します。サンプルデータを準備するには、次のサンプルコードを実行します。

```
--Creating required tables along with data.

--Creating user and schema
create user perf_user;
create schema perf_user AUTHORIZATION perf_user;
set search_path to perf_user;

--Table1:

CREATE TABLE IF NOT EXISTS perf_user.rnr_expiry_date
(
    airline_iata_code character(2) COLLATE pg_catalog."default",
    pnr_number character varying(15) COLLATE pg_catalog."default" NOT NULL,
    calculated_pnr_expiry_date timestamp(0) without time zone,
    row_num bigint,
    arc_expiry_date timestamp(0) without time zone,
    status character varying(10) COLLATE pg_catalog."default"
);

insert into perf_user.rnr_expiry_date
select 'XX' , upper(substring(concat(md5(random()::text), md5(random()::text)), 0,
7)), '2023-01-01 00:00:00', generate_series(1,100000), '2023-02-02 00:00:00' ,null;
```

```
CREATE INDEX rnr_expiry_date_idx1 ON perf_user.rnr_expiry_date (row_num ASC NULLS
LAST);

CREATE INDEX rnr_expiry_date_idx2 ON perf_user.rnr_expiry_date (airline_iata_code
COLLATE pg_catalog."default" ASC NULLS LAST, pnr_number COLLATE pg_catalog."default"
ASC NULLS LAST);

CREATE INDEX rnr_expiry_date_idx3 ON perf_user.rnr_expiry_date (pnr_number ASC NULLS
LAST);

vacuum analyze perf_user.rnr_expiry_date;

-----
--Table2:

CREATE TABLE IF NOT EXISTS perf_user.rnr_segment_pax
(
    airline_iata_code character varying(6) COLLATE pg_catalog."default" NOT NULL,
    pnr_number character varying(15) COLLATE pg_catalog."default" NOT NULL,
    segment_pax_id numeric(25,0) NOT NULL,
    oandd_id numeric(25,0) NOT NULL,
    segment_id numeric(25,0) NOT NULL,
    cabin_class character varying(15) COLLATE pg_catalog."default",
    pax_id numeric(25,0) NOT NULL,
    ticket_number character varying(25) COLLATE pg_catalog."default",
    ticket_type character varying(10) COLLATE pg_catalog."default",
    archive_status smallint NOT NULL DEFAULT (0)::smallint,
    certificate_number character varying(100) COLLATE pg_catalog."default",
    loyalty_number character varying(25) COLLATE pg_catalog."default",
    arc_expiry_date timestamp(0) without time zone,
    CONSTRAINT rnr_segment_pax_pk PRIMARY KEY (airline_iata_code, pnr_number,
segment_id, pax_id),
    CONSTRAINT rnr_segment_pax_ck1 CHECK (ticket_type::text = ANY (ARRAY['E'::character
varying::text, 'A'::character varying::text, 'C'::character varying::text,
'M'::character varying::text, 'I'::character varying::text]))
);

insert into perf_user.rnr_segment_pax (airline_iata_code, pnr_number, segment_pax_id,
oandd_id, segment_id, pax_id, ticket_type, arc_expiry_date )
select 'XX',upper(substring(concat(md5(random()::text), md5(random()::text)), 0, 7)),
generate_series(1,10000000),generate_series(1,10000000),
generate_series(1,10000000),generate_series(1,10000000),'A','2023-01-01 00:00:00';
```

```
insert into perf_user.rnr_segment_pax (airline_iata_code, pnr_number, segment_pax_id,
  oandd_id, segment_id, pax_id, ticket_type, arc_expiry_date )
select 'XX',upper(substring(concat(md5(random())::text), md5(random())::text)), 0, 7)),
generate_series(10000001,20000000),generate_series(10000001,20000000),
generate_series(10000001,20000000),generate_series(10000001,20000000),'I','2023-01-01
00:00:00';

insert into perf_user.rnr_segment_pax (airline_iata_code, pnr_number, segment_pax_id,
  oandd_id, segment_id, pax_id, ticket_type, arc_expiry_date)
select 'XX',upper(substring(concat(md5(random())::text), md5(random())::text)), 0, 7)),
generate_series(20000001,30000000),generate_series(20000001,30000000),
generate_series(20000001,30000000),generate_series(20000001,30000000),'E','2023-01-01
00:00:00';

insert into perf_user.rnr_segment_pax (airline_iata_code, pnr_number, segment_pax_id,
  oandd_id, segment_id, pax_id, ticket_type, arc_expiry_date)
select 'XX',upper(substring(concat(md5(random())::text), md5(random())::text)), 0, 7)),
generate_series(30000001,40000000),generate_series(30000001,40000000),
generate_series(30000001,40000000),generate_series(30000001,40000000),'M','2023-01-01
00:00:00';

CREATE INDEX rnr_segment_pax_idx1
  ON perf_user.rnr_segment_pax USING btree
  (loyalty_number COLLATE pg_catalog."default" ASC NULLS LAST, airline_iata_code
  COLLATE pg_catalog."default" ASC NULLS LAST, arc_expiry_date ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS rnr_segment_pax_pn_idx1
  ON perf_user.rnr_segment_pax USING btree
  (pnr_number COLLATE pg_catalog."default" ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS rnr_segment_pax_seq_idx1
  ON perf_user.rnr_segment_pax USING btree
  (segment_id ASC NULLS LAST);

vacuum analyze perf_user.rnr_segment_pax;

-----

--Table3:

CREATE TABLE IF NOT EXISTS perf_user.rnr_segment
(
  airline_iata_code character varying(6) COLLATE pg_catalog."default" NOT NULL,
```

```
    pnr_number character varying(15) COLLATE pg_catalog."C" NOT NULL,
    segment_id numeric(25,0) NOT NULL,
    oandd_id numeric(25,0),
    price_id numeric(25,0),
    flight_carrier character varying(6) COLLATE pg_catalog."default" ,
    flight_number integer ,
    flight_suffix character varying(1) COLLATE pg_catalog."default" ,
    flight_date_ltc timestamp(0) without time zone ,
    airline_company_code character varying(6) COLLATE pg_catalog."default",
    bd_airport_code character varying(5) COLLATE pg_catalog."default" ,
    off_airport_code character varying(5) COLLATE pg_catalog."default" ,
    segment_status character varying(50) COLLATE pg_catalog."default" ,
    flight_status character varying(30) COLLATE pg_catalog."default",
    flight_type character varying(15) COLLATE pg_catalog."default",
    cabin_class character varying(15) COLLATE pg_catalog."default",
    arc_expiry_date timestamp(0) without time zone,
    oandd_dep_date_ltc timestamp(0) without time zone,
    added_time timestamp(6) without time zone,
    dep_date_ltc timestamp(0) without time zone ,
    arr_date_utc timestamp(0) without time zone,
    dep_date_utc timestamp(0) without time zone,
    origin character varying(5) COLLATE pg_catalog."default",
    destination character varying(5) COLLATE pg_catalog."default",
    CONSTRAINT rnr_segment_pk PRIMARY KEY (pnr_number, segment_id, airline_iata_code)
);
```

```
insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER, FLIGHT_NUMBER, FLIGHT_SUFFIX, FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(1,1000000), 'XX', 110, '*', '2023-01-01 00:00:00';
```

```
insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER, FLIGHT_NUMBER, FLIGHT_SUFFIX, FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(10000001,20000000), 'XX', 120, '*', '2023-01-01 00:00:00';
```

```
insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER, FLIGHT_NUMBER, FLIGHT_SUFFIX, FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(20000001,30000000), 'XX', 130, '*', '2023-01-01 00:00:00';
```

```
insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER, FLIGHT_NUMBER, FLIGHT_SUFFIX, FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(30000001,40000000), 'XX', 140, '*', '2023-01-01 00:00:00';

insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER, FLIGHT_NUMBER, FLIGHT_SUFFIX, FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(40000001,50000000), 'XX', 150, '*', '2023-01-01 00:00:00';

insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER, FLIGHT_NUMBER, FLIGHT_SUFFIX, FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(50000001,60000000), 'XX', 160, '*', '2023-01-01 00:00:00';

CREATE INDEX rnr_segment_idx1 ON perf_user.rnr_segment USING btree
    (flight_date_ltc ASC NULLS LAST, bd_airport_code COLLATE pg_catalog."default"
    ASC NULLS LAST, off_airport_code COLLATE pg_catalog."default" ASC NULLS LAST,
    flight_number ASC NULLS LAST, flight_carrier COLLATE pg_catalog."default" ASC NULLS
    LAST, flight_suffix COLLATE pg_catalog."default" ASC NULLS LAST, airline_iata_code
    COLLATE pg_catalog."default" ASC NULLS LAST, arc_expiry_date ASC NULLS LAST);

CREATE INDEX rnr_segment_idx2
    ON perf_user.rnr_segment USING btree
    (dep_date_ltc ASC NULLS LAST, flight_number ASC NULLS LAST, bd_airport_code COLLATE
    pg_catalog."default" ASC NULLS LAST, off_airport_code COLLATE pg_catalog."default" ASC
    NULLS LAST, flight_carrier COLLATE pg_catalog."default" ASC NULLS LAST, flight_suffix
    COLLATE pg_catalog."default" ASC NULLS LAST, arc_expiry_date ASC NULLS LAST);

CREATE INDEX rnr_segment_idx3
    ON perf_user.rnr_segment USING btree
    (pnr_number COLLATE pg_catalog."default" ASC NULLS LAST, arr_date_utc ASC NULLS
    LAST, airline_iata_code COLLATE pg_catalog."default" ASC NULLS LAST, arc_expiry_date
    ASC NULLS LAST);

CREATE INDEX rnr_segment_idx4
    ON perf_user.rnr_segment USING btree
    (dep_date_utc ASC NULLS LAST, added_time ASC NULLS LAST, airline_iata_code COLLATE
    pg_catalog."default" ASC NULLS LAST, arc_expiry_date ASC NULLS LAST);
```

```
CREATE INDEX rnr_segment_idx5
  ON perf_user.rnr_segment USING btree
  (origin COLLATE pg_catalog."default" ASC NULLS LAST, destination COLLATE
  pg_catalog."default" ASC NULLS LAST, oandd_dep_date_ltc ASC NULLS LAST,
  airline_iata_code COLLATE pg_catalog."default" ASC NULLS LAST, arc_expiry_date ASC
  NULLS LAST);

CREATE INDEX rnr_segment_idx6
  ON perf_user.rnr_segment USING btree
  (pnr_number COLLATE pg_catalog."default" ASC NULLS LAST, oandd_id ASC NULLS LAST,
  segment_id ASC NULLS LAST, airline_iata_code COLLATE pg_catalog."default" ASC NULLS
  LAST, arc_expiry_date ASC NULLS LAST);

vacuum analyze perf_user.rnr_segment;

-----

--Table4:

CREATE TABLE IF NOT EXISTS perf_user.rnr_seat_numbers
(
  airline_iata_code character varying(6) COLLATE pg_catalog."default" NOT NULL,
  pnr_number character varying(15) COLLATE pg_catalog."default" NOT NULL,
  segment_id numeric(25,0) NOT NULL,
  pax_id numeric(25,0) NOT NULL,
  seat_id numeric(25,0) NOT NULL,
  bd_airport_code character varying(5) COLLATE pg_catalog."default",
  off_airport_code character varying(5) COLLATE pg_catalog."default",
  seat_number character varying(5) COLLATE pg_catalog."default",
  seat_status character varying(20) COLLATE pg_catalog."default",
  ssr_id character varying(100) COLLATE pg_catalog."default",
  archive_status smallint DEFAULT (0)::smallint,
  seat_alloc_id numeric(25,0),
  archive_date timestamp(0) without time zone,
  seat_attribute_code character varying(201) COLLATE pg_catalog."default",
  channel_code character varying(20) COLLATE pg_catalog."default",
  arc_expiry_date timestamp(0) without time zone,
  CONSTRAINT rnr_seat_numbers_pk PRIMARY KEY (pnr_number, segment_id, pax_id,
  seat_id, airline_iata_code)
);
```

```
insert into perf_user.rnr_seat_numbers (pnr_number, segment_id, pax_id, seat_id,
  airline_iata_code)
select upper(substring(concat(md5(random())::text), md5(random())::text), 0, 7)),
generate_series(1,1000000),generate_series(1,1000000),generate_series(1,1000000),'XX';

insert into perf_user.rnr_seat_numbers (pnr_number, segment_id, pax_id, seat_id,
  airline_iata_code)
select upper(substring(concat(md5(random())::text), md5(random())::text), 0, 7)),
generate_series(10000001,20000000),generate_series(10000001,20000000),generate_series(10000001,
20000000),generate_series(10000001,20000000),'XX';

insert into perf_user.rnr_seat_numbers (pnr_number, segment_id, pax_id, seat_id,
  airline_iata_code)
select upper(substring(concat(md5(random())::text), md5(random())::text), 0, 7)),
generate_series(20000001,30000000),generate_series(20000001,30000000),generate_series(20000001,
30000000),generate_series(20000001,30000000),'XX';

insert into perf_user.rnr_seat_numbers (pnr_number, segment_id, pax_id, seat_id,
  airline_iata_code)
select upper(substring(concat(md5(random())::text), md5(random())::text), 0, 7)),
generate_series(30000001,40000000),generate_series(30000001,40000000),generate_series(30000001,
40000000),generate_series(30000001,40000000),'XX';

vacuum Analyze perf_user.rnr_seat_numbers;

--Table5:
CREATE TABLE IF NOT EXISTS perf_user.test_veh
(
  test_veh_id bigint NOT NULL,
  oiltype_id bigint,
  vehicle_id character varying(50) COLLATE pg_catalog."default",
  serviceprogram_id character varying(100) COLLATE pg_catalog."default",
  startdate timestamp without time zone,
  enddate timestamp without time zone,
  last_update_dt timestamp without time zone,
  CONSTRAINT test_veh_pkey PRIMARY KEY (test_veh_id),
  CONSTRAINT test_veh_oiltype_id_fkey FOREIGN KEY (oiltype_id)
    REFERENCES perf_user.oiltype (oiltype_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
  CONSTRAINT test_veh_oiltype_id_fkey1 FOREIGN KEY (oiltype_id)
    REFERENCES perf_user.oiltype (oiltype_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
);

CREATE INDEX IF NOT EXISTS test_veh_enddate_ind
```

```
ON perf_user.test_veh USING btree
(enddate ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS test_veh_oiltype_id_ind
ON perf_user.test_veh USING btree
(oiltype_id ASC NULLS LAST);

--Table6:
CREATE TABLE IF NOT EXISTS perf_user.oiltype
(
    oiltype_id bigint NOT NULL,
    descr character varying(50) COLLATE pg_catalog."default",
    CONSTRAINT oiltype_pkey PRIMARY KEY (oiltype_id)
);

CREATE INDEX IF NOT EXISTS oiltype_oiltyp_in
ON perf_user.oiltype USING btree
(oiltype_id ASC NULLS LAST);

--Table7:
CREATE TABLE IF NOT EXISTS perf_user.serviceprogram
(
    serial bigint NOT NULL,
    serviceprogram_id character varying(50) COLLATE pg_catalog."default",
    proname character varying(150) COLLATE pg_catalog."default",
    CONSTRAINT serviceprogram_pkey PRIMARY KEY (serial)
);

CREATE INDEX IF NOT EXISTS proname_id_ind
ON perf_user.serviceprogram USING btree
(proname COLLATE pg_catalog."default" ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS serviceprogram_id_ind
ON perf_user.serviceprogram USING btree
(serviceprogram_id COLLATE pg_catalog."default" ASC NULLS LAST);

--Table8:
CREATE TABLE IF NOT EXISTS perf_user.vehicleservicehistory
(
    v_id bigint NOT NULL,
    test_veh_id bigint,
```

```
desc_1 character varying(50) COLLATE pg_catalog."default",
start_date timestamp without time zone,
end_date timestamp without time zone,
CONSTRAINT vehiclesservicehistory_pkey PRIMARY KEY (v_id)
);

CREATE INDEX IF NOT EXISTS veh_end_date_id_ind
ON perf_user.vehiclesservicehistory USING btree
(end_date ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS veh_ser_ind
ON perf_user.vehiclesservicehistory USING btree
(test_veh_id ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS vehiclesservicehistory_v_id_ind
ON perf_user.vehiclesservicehistory USING btree
(test_veh_id ASC NULLS LAST);

--Function creation
CREATE OR REPLACE FUNCTION perf_user.return_data()
RETURNS character varying
LANGUAGE 'plpgsql'
COST 100
VOLATILE PARALLEL UNSAFE
AS $BODY$
BEGIN
return 'EE9F41' ;
END;
$BODY$;

-----
CREATE TABLE IF NOT EXISTS ITEM_DETAILS
(
ITEMID INTEGER,
ORDID INTEGER,
ITEMNAME CHARACTER VARYING(200)
);

CREATE TABLE IF NOT EXISTS ORDER_DETAILS
(
ORDID INTEGER,
ORDNAME CHARACTER VARYING(200),
ORDEREDPLACE CHARACTER VARYING(55)
);
```

```
CREATE TABLE IF NOT EXISTS PAYMENT_DETAILS
(
    PAYID INTEGER,
    ORDID INTEGER,
    PAYPLACE CHARACTER VARYING(55)
);
```

ユースケース 1 – 照合順序

データベースでは、照合はデータのソートおよび比較方法を決定するための一連のルールです。照合は通常、テキスト値を比較するためのインデックス作成のために、テキストデータを異なる言語でソートする方法に適用されます。言語によって文字セットと順序が異なります。照合では、正しい文字シーケンスを定義するルールを使用して、特定の言語の文字データをソートできます。以下を指定することもできます。

- 大文字と小文字の区別
- アクセントマーク
- Kana 文字タイプ
- 記号または句読点の使用
- 文字幅
- 単語のソート

結合列が異なる照合順序を使用する場合、パフォーマンスに影響する可能性があります。次のクエリ例では、結合列の照合順序が異なる 3 つのテーブルを使用します。

テーブル名	[列名]
nrn_segment	nrn_number character varying(15) COLLATE pg_catalog."C" NOT NULL
nrn_segment_pax	nrn_number character varying(15) COLLATE pg_catalog."default" NOT NULL

`rnr_seat_numbers``pnr_number character varying(15)
COLLATE pg_catalog."default" NOT
NULL`

```
EXPLAIN ANALYZE SELECT  
A.PNR_NUMBER,  
A.PAX_ID,  
A.SEGMENT_ID,  
B.OANDD_ID,  
C.SEAT_ID,  
C.BD_AIRPORT_CODE,  
C.OFF_AIRPORT_CODE,  
C.SEAT_NUMBER ,  
B.CABIN_CLASS ,  
A.SEGMENT_PAX_ID,  
C.SEAT_ALLOC_ID,  
C.SSR_ID,  
C.SEAT_ATTRIBUTE_CODE  
from  
RNR_SEGMENT_PAX A,  
RNR_SEGMENT B,  
RNR_SEAT_NUMBERS C  
where  
B.AIRLINE_IATA_CODE = 'XX'  
and B.FLIGHT_CARRIER = 'XX'  
and B.FLIGHT_NUMBER = 140  
and B.FLIGHT_SUFFIX = '*'  
and B.FLIGHT_DATE_LTC = TO_DATE('01-JAN-2023', 'DD-MON-YYYY')  
and A.AIRLINE_IATA_CODE = B.AIRLINE_IATA_CODE  
and A.PNR_NUMBER = B.PNR_NUMBER  
and A.SEGMENT_ID = B.SEGMENT_ID  
and C.AIRLINE_IATA_CODE = B.AIRLINE_IATA_CODE  
and C.PNR_NUMBER = B.PNR_NUMBER  
and C.SEGMENT_ID = B.SEGMENT_ID  
and A.PAX_ID = C.PAX_ID  
and B.PNR_NUMBER in ('9F1588', 'E37DE0', '04E82B', '813D11', 'BFF10F');
```

前のクエリのクエリプランでは、`rnr_seat_numbers`結合された列に適切なインデックスがある場合でも、テーブルのシーケンススキャンが使用されます。これらの結合された列は異なる照合順序を使用しているため、プランナーはインデックススキャンを使用しません。

```

Nested Loop (cost=1112.14..927363.51 rows=1 width=833) (actual time=5395.367..5397.253
rows=0 loops=1)
  Join Filter: (((b.pnr_number)::text = (a.pnr_number)::text) AND (b.segment_id =
a.segment_id))
  -> Gather (cost=1111.58..670766.48 rows=1 width=843) (actual
time=5395.367..5397.251 rows=0 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    -> Hash Join (cost=111.58..669766.38 rows=1 width=843) (actual
time=5388.992..5388.993 rows=0 loops=3)
      Hash Cond: (((c.pnr_number)::text = (b.pnr_number)::text) AND
(c.segment_id = b.segment_id))
      -> Parallel Seq Scan on rnr_seat_numbers c (cost=0.00..582154.96
rows=16666637 width=760) (actual time=0.008..2963.019 rows=13333333 loops=3)
        Filter: ((airline_iata_code)::text = 'XX'::text)
      -> Hash (cost=111.52..111.52 rows=4 width=86) (actual time=0.121..0.121
rows=2 loops=3)
        Buckets: 1024 Batches: 1 Memory Usage: 9kB
        -> Index Scan using rnr_segment_pk on rnr_segment b
(cost=0.56..111.52 rows=4 width=86) (actual time=0.082..0.116 rows=2 loops=3)
          Index Cond: (((pnr_number)::text = ANY
('{9F1588,E37DE0,04E82B,813D11,BFF10F}'::text[])) AND ((airline_iata_code)::text =
'XX'::text))
          Filter: (((flight_carrier)::text = 'XX'::text) AND
(flight_number = 140) AND ((flight_suffix)::text = '*')::text) AND (flight_date_ltc =
to_date('01-JAN-2023'::text, 'DD-MON-YYYY'::text)))
          Rows Removed by Filter: 20
        -> Index Scan using rnr_segment_pax_pk on rnr_segment_pax a (cost=0.56..256597.02
rows=1 width=28) (never executed)
          Index Cond: (((airline_iata_code)::text = 'XX'::text) AND (segment_id =
c.segment_id) AND (pax_id = c.pax_id))
          Filter: ((c.pnr_number)::text = (pnr_number)::text)
Planning Time: 0.982 ms
Execution Time: 5397.314 ms

```

テーブル列の照合を"C"言語から PostgreSQL が提供するデフォルトの照合に変更するには、次の alter ステートメントを実行し、テーブルを分析します。

```

alter table rnr_segment alter column pnr_number type character varying(15) COLLATE
pg_catalog."default";

Analyze rnr_segment;

```

クエリプランでインデックススキャンが使用され、ランタイムが短縮されるようになりました。

```
Nested Loop (cost=1.69..146.63 rows=1 width=833) (actual time=0.155..0.155 rows=0
 loops=1)
  -> Nested Loop (cost=1.13..145.89 rows=1 width=111) (actual time=0.154..0.155
 rows=0 loops=1)
    -> Index Scan using rnr_segment_pk on rnr_segment b (cost=0.56..111.51 rows=4
 width=86) (actual time=0.048..0.097 rows=2 loops=1)
      Index Cond: (((pnr_number)::text = ANY
 ('{9F1588,E37DE0,04E82B,813D11,BFF10F}'::text[])) AND ((airline_iata_code)::text =
 'XX'::text))
      Filter: (((flight_carrier)::text = 'XX'::text) AND (flight_number =
 140) AND ((flight_suffix)::text = '*'::text) AND (flight_date_ltc = to_date('01-
JAN-2023'::text, 'DD-MON-YYYY'::text)))
      Rows Removed by Filter: 20
    -> Index Scan using rnr_segment_pax_pk on rnr_segment_pax a (cost=0.56..8.58
 rows=1 width=28) (actual time=0.027..0.027 rows=0 loops=2)
      Index Cond: (((airline_iata_code)::text = 'XX'::text) AND
 ((pnr_number)::text = (b.pnr_number)::text) AND (segment_id = b.segment_id))
  -> Index Scan using rnr_seat_numbers_pk on rnr_seat_numbers c (cost=0.56..0.72
 rows=1 width=760) (never executed)
      Index Cond: (((pnr_number)::text = (a.pnr_number)::text) AND (segment_id =
 a.segment_id) AND (pax_id = a.pax_id) AND ((airline_iata_code)::text = 'XX'::text))
Planning Time: 1.432 ms
Execution Time: 0.207 ms
```

ユースケース 2 – データ型の不一致

データに基づいて適切なデータ型を選択すると、ストレージサイズとパフォーマンスのバランスを最適化できます。

次のクエリ例では、`pnr_number`列を使用して 2 つのテーブルを結合します。`pnr_number`列には、テーブルごとに異なるデータ型があります。

テーブル名	列名とデータ型
<code>perf_user.rnr_segment_pax</code>	<code>pnr_number character varying(6)</code>
<code>perf_user.rnr_expiry_date</code>	<code>pnr_number character(2)</code>

```
EXPLAIN ANALYZE UPDATE perf_user.RNR_SEGMENT_PAX x SET ARC_EXPIRY_DATE =
y.ARC_EXPIRY_DATE
FROM (SELECT AIRLINE_IATA_CODE, PNR_NUMBER, ARC_EXPIRY_DATE, 0+row_num ROW_NUM
FROM perf_user.RNR_EXPIRY_DATE
WHERE airline_iata_code = 'XX'
AND row_num BETWEEN (1*5000)+0 AND (1+1)*5000) y
WHERE x.airline_iata_code = y.airline_iata_code
AND x.PNR_NUMBER =y.PNR_NUMBER;
```

```
-----
Update on rnr_segment_pax x (cost=290.97..1104986.32 rows=15515 width=460) (actual
time=14574.118..14574.120 rows=0 loops=1)
-> Hash Join (cost=290.97..1104986.32 rows=15515 width=460) (actual
time=16.967..14101.983 rows=11953 loops=1)
Hash Cond: ((x.pnr_number)::text = (rnr_expiry_date.pnr_number)::text)
-> Seq Scan on rnr_segment_pax x (cost=0.00..954539.00 rows=40000320
width=446) (actual time=0.011..9702.989 rows=40000000 loops=1)
Filter: ((airline_iata_code)::bpchar = 'XX'::bpchar)
-> Hash (cost=225.37..225.37 rows=5248 width=24) (actual time=16.540..16.541
rows=5001 loops=1)
Buckets: 8192 Batches: 1 Memory Usage: 338kB
-> Index Scan using rnr_expiry_date_idx1 on rnr_expiry_date
(cost=0.29..225.37 rows=5248 width=24) (actual time=3.102..15.331 rows=5001 loops=1)
Index Cond: ((row_num >= 5000) AND (row_num <= 10000))
Filter: (airline_iata_code = 'XX'::bpchar)
Planning Time: 4.445 ms
Execution Time: 14574.322 ms
```

を実行するとEXPLAIN ANALYZE、結合で使用される列にインデックスがある場合でも、プランナーはインデックススキャンrnr_segment_paxの代わりにシーケンススキャンを使用します。結合で使用される列の長さが異なるため、プランナーはインデックススキャンを使用しません。

テーブル列を変更して、結合条件に関係する両方のテーブルのデータ型を同じに保ち、テーブルを分析します。

```
alter table perf_user.rnr_expiry_date alter column airline_iata_code type character
varying(6) ;

analyze perf_user.rnr_expiry_date;
```

これで、結合条件で使用される両方の列でテーブルの長さになります。

EXPLAIN ANALYZE をもう一度実行します。プランナーはインデックススキャンを実行するため、クエリのパフォーマンスが大幅に向上します。

```
Update on rnr_segment_pax x (cost=0.86..59733.09 rows=14637 width=460) (actual
time=416.653..416.654 rows=0 loops=1)
-> Nested Loop (cost=0.86..59733.09 rows=14637 width=460) (actual
time=0.103..91.106 rows=11953 loops=1)
    -> Index Scan using rnr_expiry_date_idx1 on rnr_expiry_date
(cost=0.29..212.69 rows=4951 width=24) (actual time=0.025..3.023 rows=5001 loops=1)
        Index Cond: ((row_num >= 5000) AND (row_num <= 10000))
        Filter: ((airline_iata_code)::text = 'XX'::text)
    -> Index Scan using rnr_segment_pax_pk on rnr_segment_pax x (cost=0.56..11.99
rows=3 width=446) (actual time=0.014..0.016 rows=2 loops=5001)
        Index Cond: (((airline_iata_code)::text = 'XX'::text) AND
((pnr_number)::text = (rnr_expiry_date.pnr_number)::text))
Planning Time: 0.310 ms
Execution Time: 416.696 ms
```

ユースケース 3 – SELECT ステートメントの関数呼び出し

where 句で関数を呼び出すVOLATILEと、関数がであり、関数の呼び出し中にselectキーワードを使用しない場合にクエリパフォーマンスが低下する可能性があります。

```
Select * from tab_name where fieldName = FunctionName(parameters);
```

インデックススキャンは、関数の呼び出し中に selectステートメントが使用された場合に実行されます。

```
Select * from tab_name where fieldName = ( select FunctionName(parameters) );
```

pnr_number フィールドには、rnr_expiry_dateテーブルにインデックスがあります。インデックスは、where句の値を比較するときに使用されます。

```
explain analyze select * from perf_user.rnr_expiry_date where pnr_number= 'EE9F41';

"Index Scan using rnr_expiry_date_idx3 on rnr_expiry_date (cost=0.29..8.31 rows=1
width=72) (actual time=0.020..0.021 rows=1 loops=1)"
" Index Cond: ((pnr_number)::text = 'EE9F41'::text)"
"Planning Time: 0.063 ms"
"Execution Time: 0.038 ms"
```

シーケンシャルスキャンは、フィールドでインデックスが使用可能であっても、selectキーワードなしで関数が呼び出されたときに実行されます。

```
explain analyze select * from perf_user.rnr_expiry_date where pnr_number=
perf_user.return_data();

"Seq Scan on rnr_expiry_date (cost=0.00..27084.00 rows=1 width=72) (actual
time=0.112..135.917 rows=1 loops=1)"
"  Filter: ((pnr_number)::text = (perf_user.return_data())::text)"
"  Rows Removed by Filter: 99999"
"Planning Time: 0.053 ms"
"Execution Time: 136.803 ms"
```

インデックススキャンは、関数が selectキーワードで呼び出されたときに実行されます。

```
explain analyze select * from perf_user.rnr_expiry_date where pnr_number= (select
perf_user.return_data() );

"Index Scan using rnr_expiry_date_idx3 on rnr_expiry_date (cost=0.55..8.57 rows=1
width=72) (actual time=0.058..0.061 rows=1 loops=1)"
"  Index Cond: ((pnr_number)::text = ($0)::text)"
"  InitPlan 1 (returns $0)"
"    -> Result (cost=0.00..0.26 rows=1 width=32) (actual time=0.021..0.022 rows=1
loops=1)"
"Planning Time: 0.147 ms"
"Execution Time: 0.111 ms"
```

ユースケース 4 – IN または EXISTS

クエリに IN または NOT IN 演算子がある場合は、クエリプランをチェックして、適切なインデックスが使用されていることを確認することをお勧めします。適切なインデックスが使用されておらず、クエリのパフォーマンスに予想よりも時間がかかる場合は、EXISTS または NOT EXISTS 条件を使用してクエリを書き換えてみてください。

を使用する次の例を考えてみましょう NOT IN。

```
EXPLAIN ANALYZE SELECT
  TEST_VEH.TEST_VEH_ID,
  TEST_VEH.VEHICLE_ID,
  TEST_VEH.SERVICEPROGRAM_ID,
  TEST_VEH.STARTDATE,
```

```

TEST_VEH.ENDDATE,
TEST_VEH.OILTYPE_ID
FROM PERF_USER.TEST_VEH TEST_VEH
JOIN PERF_USER.OILTYPE OT ON OT.OILTYPE_ID =TEST_VEH.OILTYPE_ID
JOIN PERF_USER.SERVICEPROGRAM SP ON SP.SERVICEPROGRAM_ID = TEST_VEH.SERVICEPROGRAM_ID
WHERE SP.PROGNAME = '18FCE8FDAF365BB'
AND OT.OILTYPE_ID =3
AND TEST_VEH.ENDDATE IS NOT NULL
AND TEST_VEH.TEST_VEH_ID NOT IN
      (SELECT TEST_VEH_ID
       FROM PERF_USER.VEHICLESERVICEHISTORY
       WHERE TEST_VEH_ID > 1
      );

```

```

-----
"Nested Loop (cost=1009.16..1188860356305.01 rows=1 width=76) (actual
time=37299.891..37347.853 rows=0 loops=1)"
"  -> Gather (cost=1009.16..1188860356303.88 rows=1 width=76) (actual
time=37299.890..37347.849 rows=0 loops=1)"
"      Workers Planned: 2"
"      Workers Launched: 2"
"      -> Hash Join (cost=9.16..1188860355303.78 rows=1 width=76) (actual
time=37286.742..37286.751 rows=0 loops=3)"
"          Hash Cond: ((test_veh.serviceprogram_id)::text =
(sp.serviceprogram_id)::text)"
"              -> Parallel Index Scan using test_veh_oiltype_id_ind on test_veh
(cost=0.56..1188860351273.04 rows=1072570 width=76) (actual time=37276.290..37276.292
rows=1 loops=3)"
"                  Index Cond: (oiltype_id = 3)"
"                  Filter: ((enddate IS NOT NULL) AND (NOT (SubPlan 1)))"
"                  Rows Removed by Filter: 0"
"                  SubPlan 1"
"                      -> Materialize (cost=0.00..1025071.31 rows=3333332 width=8)
(actual time=0.418..23201.432 rows=25001498 loops=4)"
"                          -> Seq Scan on vehicleservicehistory
(cost=0.00..728195.65 rows=33333332 width=8) (actual time=0.416..13249.975
rows=25001498 loops=4)"
"                              Filter: (test_veh_id > 1)"
"                                  -> Hash (cost=8.58..8.58 rows=1 width=11) (actual time=9.045..9.046
rows=0 loops=3)"
"                                      Buckets: 1024 Batches: 1 Memory Usage: 8kB"
"                                          -> Index Scan using progname_id_ind on serviceprogram sp
(cost=0.56..8.58 rows=1 width=11) (actual time=9.043..9.044 rows=0 loops=3)"
"                                              Index Cond: ((progname)::text = '18FCE8FDAF365BB'::text)"
"                                                  -> Seq Scan on oiltype ot (cost=0.00..1.12 rows=1 width=8) (never executed)"

```

```
"          Filter: (oiltype_id = 3)"
"Planning Time: 37.696 ms"
"Execution Time: 37366.335 ms"
```

クエリは、400 万件のレコードを取得するまでに 37 秒 366 ミリ秒以上かかります。

クエリプランは、サブクエリで使用されるテーブルでシーケンススキャンが実行されることを示します。vehicleservicehistory。シーケンススキャンは多数のレコードを生成しています。サブクエリ内の各レコードについて、クエリは完全なテーブルスキャンを実行しているため、パフォーマンスの問題が発生しています。

サブクエリのシーケンススキャンを回避するには、との相関サブクエリを使用するようにサブクエリを書き換えます。NOT EXISTS。相関サブクエリはインデックススキャンを使用し、テーブルスキャンの数を減らします。

```
EXPLAIN ANALYZE SELECT
  TEST_VEH.TEST_VEH_ID,
  TEST_VEH.VEHICLE_ID,
  TEST_VEH.SERVICEPROGRAM_ID,
  TEST_VEH.STARTDATE,
  TEST_VEH.ENDDATE,
  TEST_VEH.OILTYPE_ID
FROM PERF_USER.TEST_VEH TEST_VEH
JOIN PERF_USER.OILTYPE OT ON OT.OILTYPE_ID =TEST_VEH.OILTYPE_ID
JOIN PERF_USER.SERVICEPROGRAM SP ON SP.SERVICEPROGRAM_ID = TEST_VEH.SERVICEPROGRAM_ID
WHERE SP.PROGNAME = '18FCE8FDAF365BB'
      AND OT.OILTYPE_ID =3
      AND TEST_VEH.ENDDATE IS NOT NULL
      AND NOT EXISTS
          (SELECT TEST_VEH_ID
           FROM PERF_USER.VEHICLESERVICEHISTORY
           WHERE
TEST_VEH.TEST_VEH_ID=VEHICLESERVICEHISTORY.TEST_VEH_ID
           AND TEST_VEH_ID > 1
          );
```

```
-----
"Nested Loop Anti Join (cost=1009.03..936146.10 rows=1 width=76) (actual
time=12.693..12.810 rows=0 loops=1)"
"  -> Nested Loop (cost=1008.59..936141.78 rows=1 width=76) (actual
time=12.692..12.809 rows=0 loops=1)"
"      -> Gather (cost=1008.59..936140.64 rows=1 width=76) (actual
time=12.691..12.807 rows=0 loops=1)"
```

```
"          Workers Planned: 2"
"          Workers Launched: 2"
"          -> Hash Join (cost=8.59..935140.54 rows=1 width=76) (actual
time=0.773..0.774 rows=0 loops=3)"
"              Hash Cond: ((test_veh.serviceprogram_id)::text =
(sp.serviceprogram_id)::text)"
"                  -> Parallel Seq Scan on test_veh (cost=0.00..927087.67
rows=2145139 width=76) (actual time=0.672..0.672 rows=1 loops=3)"
"                      Filter: ((enddate IS NOT NULL) AND (oiltype_id = 3))"
"                          Rows Removed by Filter: 7"
"                  -> Hash (cost=8.58..8.58 rows=1 width=11) (actual
time=0.040..0.040 rows=0 loops=3)"
"                      Buckets: 1024  Batches: 1  Memory Usage: 8kB"
"                  -> Index Scan using progname_id_ind on serviceprogram sp
(cost=0.56..8.58 rows=1 width=11) (actual time=0.039..0.040 rows=0 loops=3)"
"                      Index Cond: ((progname)::text =
'18FCE8FDAF365BB'::text)"
"                  -> Seq Scan on oiltype ot (cost=0.00..1.12 rows=1 width=8) (never executed)"
"                      Filter: (oiltype_id = 3)"
"          -> Index Only Scan using veh_ser_ind on vehicleservicehistory (cost=0.44..4.32
rows=1 width=8) (never executed)"
"              Index Cond: ((test_veh_id = test_veh.test_veh_id) AND (test_veh_id > 1))"
"              Heap Fetches: 0"
"Planning Time: 11.115 ms"
"Execution Time: 12.871 ms"
```

変更後、クエリが 400 万レコードを処理するまでに 13 ミリ秒未満かかる

変更されたクエリのクエリプランに従って、テーブルはインデックススキャンを持つ `vehicleservicehistory` ことができます。インデックススキャンを使用すると、コストと影響を受ける行の数が減少します。これにより、クエリの実行時間を短縮し、パフォーマンスを向上させることができます。

ユースケース 5 – サブクエリまたは CTEs

Common Table Expressions (CTEs) 大きなクエリを小さなクエリに分割するのに役立ちます。これにより、クエリ全体の保守が容易になります。

サブクエリ結合は CTE 結合に置き換えられます。この結合は、クエリの名前が付けられ、CTE セクション内で区切られるため、読みやすくなります。これは、クエリのサイズが大きくなり、クエリの維持が難しくなった場合に特に役立ちます。さらに、PostgreSQL の CTE 結果がマテリアライズさ

れます。CTE を複数の場所で呼び出すと、実際のクエリ定義は 1 回だけ実行されます。結果はメモリに保存されます。これは、同じクエリ内の複数の場所で使用する必要がある複雑なロジックに使用できます。このロジックを CTE 内に配置し、CTE を何度でも呼び出します。

たとえば、ある顧客がインラインアプリケーションクエリを使用していて、クエリ内に多数のサブクエリがありました。サブクエリは、アプリケーションから送信された入力パラメータ値でフィルタリングされました。

```
EXPLAIN ANALYZE
SELECT * FROM
ORDER_DETAILS A
WHERE A.ORDID IN (SELECT ORDID FROM PAYMENT_DETAILS)
AND A.ORDID IN (SELECT ORDID FROM ITEM_DETAILS )
AND A.ORDID = 1000000;
```

```
"Nested Loop Semi Join (cost=3000.00..194258.21 rows=5 width=74) (actual
time=201.605..747.945 rows=5 loops=1)"
"  -> Nested Loop Semi Join (cost=2000.00..135040.47 rows=5 width=74) (actual
time=146.016..666.779 rows=5 loops=1)"
"    -> Gather (cost=1000.00..78580.31 rows=5 width=74) (actual
time=58.893..463.570 rows=5 loops=1)"
"      Workers Planned: 2"
"      Workers Launched: 2"
"    -> Parallel Seq Scan on order_details a (cost=0.00..77579.81 rows=2
width=74) (actual time=165.627..549.702 rows=2 loops=3)"
"      Filter: (ordid = 1000000)"
"      Rows Removed by Filter: 1666665"
"    -> Materialize (cost=1000.00..56460.07 rows=3 width=4) (actual
time=17.424..40.638 rows=1 loops=5)"
"      -> Gather (cost=1000.00..56460.06 rows=3 width=4) (actual
time=87.113..203.178 rows=1 loops=1)"
"        Workers Planned: 2"
"        Workers Launched: 2"
"      -> Parallel Seq Scan on payment_details (cost=0.00..55459.76
rows=1 width=4) (actual time=174.431..423.792 rows=1 loops=3)"
"        Filter: (ordid = 1000000)"
"        Rows Removed by Filter: 1333002"
"      -> Materialize (cost=1000.00..59217.64 rows=4 width=4) (actual time=11.117..16.231
rows=1 loops=5)"
"        -> Gather (cost=1000.00..59217.62 rows=4 width=4) (actual
time=55.581..81.148 rows=1 loops=1)"
"          Workers Planned: 2"
```

```

"           Workers Launched: 2"
"           -> Parallel Seq Scan on item_details (cost=0.00..58217.22 rows=2
width=4) (actual time=287.030..411.004 rows=1 loops=3)"
"           Filter: (ordid = 1000000)"
"           Rows Removed by Filter: 1333080"
"Planning Time: 0.266 ms"
"Execution Time: 747.986 ms"

```

CTE を使用してサブクエリを変更し、必要な行セットのみを取得するようにフィルターを追加すると、クエリのパフォーマンスが向上します。

```

EXPLAIN ANALYZE
WITH PAYMENT AS
(
  SELECT * FROM PAYMENT_DETAILS WHERE  ORCID = 1000000
),
ITEM AS
(SELECT * FROM ITEM_DETAILS  WHERE  ORCID = 1000000)
SELECT * FROM
ORDER_DETAILS A JOIN PAYMENT B
ON A.ORCID=B.ORCID
JOIN ITEM C ON B.ORCID=C.ORCID

```

```

"Nested Loop (cost=3000.00..194258.91 rows=60 width=166) (actual time=586.410..732.918
rows=80 loops=1)"
"  -> Nested Loop (cost=2000.00..115677.83 rows=12 width=92) (actual
time=456.760..457.083 rows=16 loops=1)"
"    -> Gather (cost=1000.00..59217.62 rows=4 width=48) (actual
time=153.802..154.060 rows=4 loops=1)"
"      Workers Planned: 2"
"      Workers Launched: 2"
"        -> Parallel Seq Scan on item_details (cost=0.00..58217.22 rows=2
width=48) (actual time=85.417..249.045 rows=1 loops=3)"
"          Filter: (ordid = 1000000)"
"          Rows Removed by Filter: 1333332"
"        -> Materialize (cost=1000.00..56460.07 rows=3 width=44) (actual
time=75.738..75.753 rows=4 loops=4)"
"          -> Gather (cost=1000.00..56460.06 rows=3 width=44) (actual
time=302.947..303.005 rows=4 loops=1)"
"            Workers Planned: 2"
"            Workers Launched: 2"

```

```
"          -> Parallel Seq Scan on payment_details (cost=0.00..55459.76
rows=1 width=44) (actual time=184.609..294.784 rows=1 loops=3)"
"          Filter: (ordid = 1000000)"
"          Rows Removed by Filter: 1333332"
" -> Materialize (cost=1000.00..78580.34 rows=5 width=74) (actual time=8.103..17.238
rows=5 loops=16)"
"   -> Gather (cost=1000.00..78580.31 rows=5 width=74) (actual
time=129.641..275.795 rows=5 loops=1)"
"       Workers Planned: 2"
"       Workers Launched: 2"
"   -> Parallel Seq Scan on order_details a (cost=0.00..77579.81 rows=2
width=74) (actual time=78.556..268.994 rows=2 loops=3)"
"       Filter: (ordid = 1000000)"
"       Rows Removed by Filter: 1666665"
"Planning Time: 0.108 ms"
"Execution Time: 732.953 ms"
```

これらは、サンプルデータからの観測値です。巨大なデータセットでクエリを実行すると、パフォーマンスの差は非常に大きくなります。

よくある質問

クエリパフォーマンスのチューニングに関して頻繁に寄せられる質問に対する回答を見つけます。

EXPLAIN とは

EXPLAIN は、クエリプランを生成するために PostgreSQL クエリ (SELECT、UPDATE、INSERT、DELETE) の前に付加するキーワードです。PostgreSQL クエリプランは、データベースがクエリを実行する方法の詳細を表示します。このプランには、テーブルスキャンの順序、インデックスの使用状況、および結合に関する情報が含まれます。

クエリプランを使用して、潜在的なボトルネックを特定し、クエリを最適化し、全体的なパフォーマンスを向上させます。クエリプランを確認するときは、次の要素を考慮してください。

- テーブルアクセスアプローチ
- 結合アプローチ
- フィルター条件
- ソートオペレーション
- インデックスの使用状況
- 並列処理
- 統計
- コスト見積もり
- 各ステップから取得した行
- データのディストリビューション

[EXPLAIN](#) の詳細については、PostgreSQL ドキュメントを参照してください。

EXPLAIN ANALYZE とは

クエリの前 に を付加EXPLAIN ANALYZEしてクエリを実行すると、PostgreSQL はクエリを実行し、クエリプランとランタイム統計の両方を返します。実際のランタイム、各ステップで処理された行、およびその他の関連情報が、クエリプランとともに表示されます。クエリの実行は分析中のデータベースのパフォーマンスに影響を与える可能性があるため、本番データベースEXPLAIN ANALYZEでの の使用は慎重に行う必要があります。

[EXPLAIN ANALYZE](#) の詳細については、PostgreSQL ドキュメントを参照してください。

PostgreSQL での照合とは

PostgreSQL では、照合は文字列の比較とソート方法を決定するための一連のルールです。照合では、言語固有のルールと変換を考慮して、比較で文字が考慮される順序を定義します。

[照合](#)の詳細については、PostgreSQL ドキュメントを参照してください。

CTE とは

PostgreSQL データベースでは、共通テーブル式 (CTE) は参照できる名前付き一時結果セットです。CTEsは、複雑なロジックを小さな名前付き単位に分割することで、より読みやすくモジュール化された SQL クエリを作成する方法を提供します。

[CTEs](#)、PostgreSQL ドキュメントを参照してください。

PostgreSQL の関数のカテゴリは何ですか？

すべての PostgreSQL 関数にはボラティリティ分類があり、その可能性は VOLATILE、STABLE、または IMMUTABLE。

- VOLATILE – VOLATILE関数は、データベースの変更など、何でも実行できます。同じ引数を持つ連続した呼び出しに対して異なる結果を返すことができます。オプティマイザは、そのような関数の動作について仮定しません。揮発性関数を使用するクエリは、値が必要な行ごとに関数を再評価します。
- STABLE – STABLE関数はデータベースを変更できません。1つのステートメント内のすべての行に対して同じ引数を指定して、同じ結果を返すことが保証されます。この分類を使用すると、オプティマイザは1回の呼び出しに対して関数の複数の呼び出しを最適化できます。特に、インデックススキャン条件にこのような関数を含む式を使用すると安全です。(インデックススキャンでは、各行で1回ではなく1回だけ比較値が評価されるため、インデックススキャン条件でVOLATILE関数を使用することは有効ではありません)。
- IMMUTABLE – IMMUTABLE関数はデータベースを変更できず、同じ引数を永遠に持つと、同じ結果を返すことが保証されます。この分類を使用すると、クエリが定数引数で関数を呼び出すときに、オプティマイザは関数を事前に評価できます。例えば、`SELECT ... WHERE x = 4`、`SELECT ... WHERE x = 2 + 2` が見れば簡略化SELECT ... WHERE x = 2 + 2できますIMMUTABLE。

VOLATILE は、CREATE FUNCTION コマンドがカテゴリを指定しない場合のデフォルトです。[関数タイプ](#)の詳細については、PostgreSQL ドキュメントを参照してください。

リソース

リファレンス

- [説明](#)
- [EXPLAIN の使用](#)
- [照合のサポート](#)
- [WITH クエリ \(共通テーブル式\)](#)

ガイド

- [パフォーマンスの問題を回避するための Amazon RDS および Amazon Aurora の PostgreSQL データベースのメンテナンスアクティビティ](#)
- [Amazon RDS および Amazon Aurora での PostgreSQL パラメータのチューニング](#)

寄稿者

本ドキュメントの寄稿者は次のとおりです。

- Tirumala Dasari、リードコンサルタント – データベース、AWS
- Veeranjanyulu Grandhi、リードコンサルタント – データベース、AWS
- Vamsikrishna Jammula、コンサルタント – データベース、AWS
- Srinivas Potlacheruvu、シニアリードコンサルタント – データベース、AWS
- " Srinivas Reddy Ravulapati、コンサルタント – データベース、AWS

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、[RSS フィード](#) をサブスクライブできます。

変更	説明	日付
初版発行	—	2024 年 4 月 23 日

AWS 規範ガイドの用語集

以下は、AWS 規範ガイドによって提供される戦略、ガイド、パターンで一般的に使用される用語です。エントリを提案するには、用語集の最後のフィードバックの提供リンクを使用します。

数字

7 Rs

アプリケーションをクラウドに移行するための 7 つの一般的な移行戦略。これらの戦略は、ガートナーが 2011 年に特定した 5 Rs に基づいて構築され、以下で構成されています。

- リファクタリング/アーキテクチャの再設計 — クラウドネイティブ特徴を最大限に活用して、俊敏性、パフォーマンス、スケーラビリティを向上させ、アプリケーションを移動させ、アーキテクチャを変更します。これには、通常、オペレーティングシステムとデータベースの移植が含まれます。例: オンプレミスの Oracle データベースを Amazon Aurora PostgreSQL 互換エディションに移行する。
- リプラットフォーム (リフトアンドリシェイプ) — アプリケーションをクラウドに移行し、クラウド機能を活用するための最適化レベルを導入します。例: お客様のオンプレミスの Oracle データベースを AWS クラウドの Oracle 用の Amazon Relational Database Service (Amazon RDS) に移行する。
- 再購入 (ドロップアンドショップ) — 通常、従来のライセンスから SaaS モデルに移行して、別の製品に切り替えます。例: 顧客関係管理 (CRM) システムを Salesforce.com に移行する。
- リホスト (リフトアンドシフト) — クラウド機能を活用するための変更を加えずに、アプリケーションをクラウドに移行します。例: お客様のオンプレミスの Oracle データベースを AWS クラウドの EC2 インスタンス上の Oracle に移行する。
- 再配置 (ハイパーバイザーレベルのリフトアンドシフト) — 新しいハードウェアを購入したり、アプリケーションを書き換えたり、既存の運用を変更したりすることなく、インフラストラクチャをクラウドに移行できます。オンプレミスプラットフォームから同じプラットフォームのクラウドサービスにサーバーを移行します。例: Microsoft Hyper-V アプリケーションをに移行します AWS。
- 保持 (再アクセス) — アプリケーションをお客様のソース環境で保持します。これには、主要なリファクタリングを必要とするアプリケーションや、お客様がその作業を後日まで延期したいアプリケーション、およびそれらを行き移るためのビジネス上の正当性がないため、お客様が保持するレガシーアプリケーションなどがあります。
- 廃止 — お客様のソース環境で不要になったアプリケーションを停止または削除します。

A

ABAC

[「属性ベースのアクセス制御」](#)をご覧ください。

抽象化されたサービス

[「マネージドユーザー」](#)をご覧ください。

ACID

[「原子性、一貫性、分離性、耐久性 \(ACID\)」](#)をご覧ください。

アクティブ/アクティブ移行

(双方向レプリケーションツールまたは二重書き込み操作を使用して) ソースデータベースとターゲットデータベースを同期させ、移行中に両方のデータベースが接続アプリケーションからのトランザクションを処理するデータベース移行方法。この方法では、1 回限りのカットオーバーの必要がなく、管理された小規模なバッチで移行できます。[アクティブ/パッシブ移行](#)よりも柔軟な方法ですが、さらに多くの作業が必要となります。

アクティブ/パッシブ移行

ソースデータベースとターゲットデータベースを同期させながら、データがターゲットデータベースにレプリケートされている間、接続しているアプリケーションからのトランザクションをソースデータベースのみで処理するデータベース移行方法。移行中、ターゲットデータベースはトランザクションを受け付けません。

集計関数

複数行に処理を行い、グループ全体を対象に単一の戻り値を計算する SQL 関数。集計関数の例としては、SUM や MAX などがあります。

AI

[「人工知能」](#)をご覧ください。

AIOps

[「AI オペレーション」](#)をご覧ください。

匿名化

データセット内の個人情報を完全に削除するプロセス。匿名化は個人のプライバシー保護に役立ちます。匿名化されたデータは、もはや個人データとは見なされません。

アンチパターン

繰り返し起こる問題に対して頻繁に用いられる解決策で、その解決策が逆効果であったり、効果がなかったり、代替案よりも効果が低かったりするもの。

アプリケーション制御

マルウェアからシステムを保護するために、承認されたアプリケーションのみを使用できるようにするセキュリティアプローチ。

アプリケーションポートフォリオ

アプリケーションの構築と維持にかかるコスト、およびそのビジネス価値を含む、組織が使用する各アプリケーションに関する詳細情報の集まり。この情報は、[ポートフォリオの検出と分析プロセス](#)の重要な要素であり、移行、モダナイズ、最適化するアプリケーションを特定し、優先順位を付けるのに役立ちます。

人工知能 (AI)

コンピューティングテクノロジーを使用し、学習、問題の解決、パターンの認識など、通常は人間に関連づけられる認知機能の実行に特化したコンピュータサイエンスの分野。詳細については、「[人工知能 \(AI\) とは何ですか?](#)」をご覧ください。

AI オペレーション (AIOps)

機械学習技術を使用して運用上の問題を解決し、運用上のインシデントと人の介入を減らし、サービス品質を向上させるプロセス。AWS 移行戦略での AIOps の使用方法については、[オペレーション統合ガイド](#)を参照してください。

非対称暗号化

暗号化用のパブリックキーと復号用のプライベートキーから成る 1 組のキーを使用した、暗号化のアルゴリズム。パブリックキーは復号には使用されないため共有しても問題ありませんが、プライベートキーの利用は厳しく制限する必要があります。

原子性、一貫性、分離性、耐久性 (ACID)

エラー、停電、その他の問題が発生した場合でも、データベースのデータ有効性と運用上の信頼性を保証する一連のソフトウェアプロパティ。

属性ベースのアクセス制御 (ABAC)

部署、役職、チーム名など、ユーザーの属性に基づいてアクセス許可をきめ細かく設定する方法。詳細については、AWS Identity and Access Management (IAM) ドキュメントの「[の ABAC AWS](#)」を参照してください。

信頼できるデータソース

最も信頼性のある情報源とされるデータのプライマリバージョンを保存する場所。匿名化、編集、仮名化など、データを処理または変更する目的で、信頼できるデータソースから他の場所にデータをコピーすることができます。

アベイラビリティゾーン (AZ)

他のアベイラビリティゾーンの障害から AWS リージョン 隔離され、同じリージョン内の他のアベイラビリティゾーンへの低コストで低レイテンシーのネットワーク接続を提供する 内の別の場所。

AWS クラウド導入フレームワーク (AWS CAF)

組織がクラウドへの移行を成功させるための効率的で効果的な計画を立てるための、のガイドラインとベストプラクティスのフレームワークです。AWS CAF は、ビジネス、人材、ガバナンス、プラットフォーム、セキュリティ、運用という 6 つの重点分野にガイドランスを整理しています。ビジネス、人材、ガバナンスの観点では、ビジネススキルとプロセスに重点を置き、プラットフォーム、セキュリティ、オペレーションの視点は技術的なスキルとプロセスに焦点を当てています。例えば、人材の観点では、人事 (HR)、人材派遣機能、および人材管理を扱うステークホルダーを対象としています。この観点から、AWS CAF は、クラウド導入を成功させるための組織の準備に役立つ人材開発、トレーニング、コミュニケーションに関するガイドランスを提供します。詳細については、[AWS CAF ウェブサイト](#)と [AWS CAF のホワイトペーパー](#) を参照してください。

AWS ワークロード認定フレームワーク (AWS WQF)

データベース移行ワークロードを評価し、移行戦略を推奨し、作業見積もりを提供するツール。AWS WQF は AWS Schema Conversion Tool (AWS SCT) に含まれています。データベーススキーマとコードオブジェクト、アプリケーションコード、依存関係、およびパフォーマンス特性を分析し、評価レポートを提供します。

B

不正なボット

個人や組織に混乱や損害を与えることを目的とした [ボット](#)。

BCP

「[ビジネス継続性計画 \(BCP\)](#)」をご覧ください。

動作グラフ

リソースの動作とインタラクションを経時的に示した、一元的なインタラクティブビュー。Amazon Detective の動作グラフを使用すると、失敗したログオンの試行、不審な API 呼び出し、その他同様のアクションを調べることができます。詳細については、Detective ドキュメントの「[動作グラフのデータ](#)」を参照してください。

ビッグエンディアンシステム

最上位バイトを最初に格納するシステム。「[エンディアン性](#)」もご覧ください。

二項分類

バイナリ結果 (2 つの可能なクラスのうちの一つ) を予測するプロセス。例えば、お客様の機械学習モデルで「この E メールはスパムですか、それともスパムではありませんか」などの問題を予測する必要があるかもしれません。または「この製品は書籍ですか、車ですか」などの問題を予測する必要があるかもしれません。

ブルームフィルター

要素がセットのメンバーであるかどうかをテストするために使用される、確率的でメモリ効率の高いデータ構造。

ブルー/グリーンデプロイ

それぞれが独立しているが、同一の環境を 2 つ作成するデプロイ戦略。現在のアプリケーションバージョンを 1 つの環境 (ブルー) で実行し、新しいアプリケーションバージョンを別の環境 (グリーン) で実行します。この戦略は、最小限の影響で迅速にロールバックするのに役立ちます。

ボット

インターネット経由で自動タスクを実行し、人間のアクティビティややり取りをシミュレートするソフトウェアアプリケーション。インターネット上の情報のインデックスを作成するウェブクロウラーなど、一部のボットは有用または有益です。悪質なボットと呼ばれる他のボットの中には、個人や組織を混乱させたり、損害を与えたりすることを意図したものもあります。

ボットネット

[マルウェア](#)に感染しており、ボットハーダーまたはボットオペレーターと呼ばれる単一の当事者によって制御されている[ボット](#)のネットワーク。ボットネットは、ボットとその影響力を拡大する仕組みとして、非常によく知られています。

ブランチ

コードリポジトリに含まれる領域。リポジトリに最初に作成するブランチは、メインブランチといます。既存のブランチから新しいブランチを作成し、その新しいブランチで機能を開発した

り、バグを修正したりできます。機能を構築するために作成するブランチは、通常、機能ブランチと呼ばれます。機能をリリースする準備ができたなら、機能ブランチをメインブランチに統合します。詳細については、「[ブランチの概要](#)」(GitHub ドキュメント)を参照してください。

ブレイクグラスアクセス

例外的な状況では、承認されたプロセスを通じて、ユーザーが AWS アカウント 通常アクセス許可を持たない にすばやくアクセスできるようにします。詳細については、AWS Well-Architected ガイドの「[ブレイクグラス手順の実装](#)」インジケータを参照してください。

ブラウフィールド戦略

環境の既存インフラストラクチャ。システムアーキテクチャにブラウフィールド戦略を導入する場合、現在のシステムとインフラストラクチャの制約に基づいてアーキテクチャを設計します。既存のインフラストラクチャを拡張している場合は、ブラウフィールド戦略と[グリーンフィールド](#)戦略を融合させることもできます。

バッファキャッシュ

アクセス頻度が最も高いデータが保存されるメモリ領域。

ビジネス能力

価値を生み出すためにビジネスが行うこと (営業、カスタマーサービス、マーケティングなど)。マイクロサービスのアーキテクチャと開発の決定は、ビジネス能力によって推進できます。詳細については、[AWSでのコンテナ化されたマイクロサービスの実行](#)ホワイトペーパーの「[ビジネス機能を中心に組織化](#)」セクションを参照してください。

ビジネス継続性計画 (BCP)

大規模移行など、中断を伴うイベントが運用に与える潜在的な影響に対処し、ビジネスを迅速に再開できるようにする計画。

C

CAF

「[AWS クラウド導入フレームワーク](#)」を参照してください

カナリアデプロイ

エンドユーザーへのバージョンリリースを、時間をかけて段階的に行うこと。確信が持てたら新規バージョンをデプロイして、現在のバージョン全体を置き換えます。

CCoE

「[Cloud Center of Excellence](#)」を参照してください。

CDC

「[変更データキャプチャ](#)」を参照してください。

変更データキャプチャ (CDC)

データソース (データベーステーブルなど) の変更を追跡し、その変更に関するメタデータを記録するプロセス。CDC は、ターゲットシステムでの変更を監査またはレプリケートして同期を維持するなど、さまざまな目的に使用できます。

カオスエンジニアリング

障害や破壊的なイベントを意図的に導入して、システムの耐障害性をテストすること。[AWS Fault Injection Service \(AWS FIS\)](#) を使用して、AWS ワークロードにストレスを与え、その応答を評価する実験を実行できます。

CI/CD

「[継続的インテグレーションと継続的デリバリー](#)」を参照してください。

分類

予測を生成するのに役立つ分類プロセス。分類問題の機械学習モデルは、離散値を予測します。離散値は、常に互いに区別されます。例えば、モデルがイメージ内に車があるかどうかを評価する必要がある場合があります。

クライアント側の暗号化

ターゲットがデータ AWS のサービスを受信する前のローカルでのデータの暗号化。

Cloud Center of Excellence (CCoE)

クラウドのベストプラクティスの作成、リソースの移動、移行のタイムラインの確立、大規模変革を通じて組織をリードするなど、組織全体のクラウド導入の取り組みを推進する学際的なチーム。詳細については、AWS クラウド エンタープライズ戦略ブログの [CCoE 投稿](#) を参照してください。

クラウドコンピューティング

リモートデータストレージと IoT デバイス管理に通常使用されるクラウドテクノロジー。クラウドコンピューティングは、一般的に、[エッジコンピューティング](#)に接続されています。

クラウド運用モデル

IT 組織において、1 つ以上のクラウド環境を構築、成熟、最適化するために使用される運用モデル。詳細については、「[クラウド運用モデルの構築](#)」を参照してください。

導入のクラウドステージ

組織が、AWS クラウドへの移行時に通常実行する 4 つの段階。

- プロジェクト — 概念実証と学習を目的として、クラウド関連のプロジェクトをいくつか実行する
- 基礎固め — お客様のクラウドの導入を拡大するための基礎的な投資 (ランディングゾーン の作成、CCoE の定義、運用モデルの確立など)
- 移行 — 個々のアプリケーションの移行
- 再発明 — 製品とサービスの最適化、クラウドでのイノベーション

これらのステージは、AWS クラウド エンタープライズ戦略ブログのブログ記事「[クラウドファーストへのジャーニー](#)」と「[導入のステージ](#)」で Stephen Orban によって定義されました。移行戦略との関連性については、AWS「[移行準備ガイド](#)」を参照してください。

CMDB

「[構成管理データベース \(CMDB\)](#)」を参照してください。

コードリポジトリ

ソースコードやその他の資産 (ドキュメント、サンプル、スクリプトなど) が保存され、バージョン管理プロセスを通じて更新される場所。一般的なクラウドリポジトリには、GitHub や Bitbucket Cloud があります。コードの各バージョンはブランチと呼ばれます。マイクロサービスの構造では、各リポジトリは 1 つの機能専用です。1 つの CI/CD パイプラインで複数のリポジトリを使用できます。

コールドキャッシュ

空である、または、かなり空きがある、もしくは、古いデータや無関係なデータが含まれているバッファキャッシュ。データベースインスタンスはメインメモリまたはディスクから読み取る必要があり、バッファキャッシュから読み取るよりも時間がかかるため、パフォーマンスに影響します。

コールドデータ

めったにアクセスされず、通常は過去のデータです。この種類のデータをクエリする場合、通常は低速なクエリでも問題ありません。このデータを低パフォーマンスで安価なストレージ階層またはクラスに移動すると、コストを削減することができます。

コンピュータビジョン (CV)

機械学習を使用してデジタルイメージやビデオといった、ビジュアル形式の情報を分析および抽出する [AI](#) の分野。例えば、Amazon SageMaker AI では、CV 用の画像処理アルゴリズムを利用できます。

設定ドリフト

ワークロードにおいて、設定が想定した状態から変化すること。これによって、ワークロードが非準拠になる可能性があります。この状態は、徐々に生じ、意図的なものではありません。

構成管理データベース (CMDB)

データベースとその IT 環境 (ハードウェアとソフトウェアの両方のコンポーネントとその設定を含む) に関する情報を保存、管理するリポジトリ。通常、CMDB のデータは、移行のポートフォリオの検出と分析の段階で使用します。

コンフォーマンスパック

コンプライアンスチェックとセキュリティチェックをカスタマイズするためにアセンブルできる AWS Config ルールと修復アクションのコレクション。YAML テンプレートを使用して、コンフォーマンスパックを AWS アカウント および リージョンの単一のエンティティとしてデプロイすることも、組織全体にデプロイすることもできます。詳細については、AWS Config ドキュメントの「[コンフォーマンスパック](#)」を参照してください。

継続的インテグレーションと継続的デリバリー (CI/CD)

ソフトウェアリリースプロセスのソース、ビルド、テスト、ステージング、本番の各ステージを自動化するプロセス。CI/CD は一般的にパイプラインと呼ばれます。プロセスの自動化、生産性の向上、コード品質の向上、配信の加速化を可能にします。詳細については、「[継続的デリバリーの利点](#)」を参照してください。CD は継続的デプロイ (Continuous Deployment) の略語でもあります。詳細については「[継続的デリバリーと継続的なデプロイ](#)」を参照してください。

CV

[「コンピュータビジョン」](#) を参照してください。

D

保管中のデータ

ストレージ内にあるデータなど、常に自社のネットワーク内にあるデータ。

データ分類

ネットワーク内のデータを重要度と機密性に基づいて識別、分類するプロセス。データに適した保護および保持のコントロールを判断する際に役立つため、あらゆるサイバーセキュリティのリスク管理戦略において重要な要素です。データ分類は、AWS Well-Architected フレームワークのセキュリティの柱のコンポーネントです。詳細については、「[データ分類](#)」を参照してください。

データドリフト

実稼働データと ML モデルのトレーニングに使用されたデータとの間に有意な差異が生じたり、入力データが時間の経過と共に有意に変化したりすることです。データドリフトは、ML モデル予測の全体的な品質、精度、公平性を低下させる可能性があります。

転送中のデータ

ネットワーク内 (ネットワークリソース間など) を活発に移動するデータ。

データメッシュ

非一元的で分散型のデータ所有権を持つとともに、一元的な管理およびガバナンスを行えるアーキテクチャフレームワーク。

データ最小化

厳密に必要なデータのみを収集し、処理するという原則。でデータ最小化を実践 AWS クラウドすることで、プライバシーリスク、コスト、分析のカーボンフットプリントを削減できます。

データ境界

AWS 環境内の一連の予防ガードレール。信頼された ID のみが、期待されるネットワークから信頼されたリソースにアクセスできるようにします。詳細については、「[AWS でのデータ境界の構築](#)」を参照してください。

データの前処理

raw データをお客様の機械学習モデルで簡単に解析できる形式に変換すること。データの前処理とは、特定の列または行を削除して、欠落している、矛盾している、または重複する値に対処することを意味します。

データ出所

データの生成、送信、保存の方法など、データのライフサイクル全体を通じてデータの出所と履歴を追跡するプロセス。

データ件名

データを収集、処理している個人。

データウェアハウス

分析などのビジネスインテリジェンスをサポートするデータ管理システム。データウェアハウスには、一般的に、大量の履歴データが含まれており、多くの場合、それらはクエリや分析に使用されます。

データベース定義言語 (DDL)

データベース内のテーブルやオブジェクトの構造を作成または変更するためのステートメントまたはコマンド。

データベース操作言語 (DML)

データベース内の情報を変更 (挿入、更新、削除) するためのステートメントまたはコマンド。

DDL

「[データベース定義言語](#)」を参照してください。

ディープアンサンブル

予測のために複数の深層学習モデルを組み合わせます。ディープアンサンブルを使用して、より正確な予測を取得したり、予測の不確実性を推定したりできます。

深層学習

人工ニューラルネットワークの複数層を使用して、入力データと対象のターゲット変数の間のマッピングを識別する機械学習サブフィールド。

多層防御

一連のセキュリティメカニズムとコントロールをコンピュータネットワーク全体に層状に重ねて、ネットワークとその内部にあるデータの機密性、整合性、可用性を保護する情報セキュリティの手法。この戦略をに採用するときは AWS、リソースの保護に役立つように、AWS Organizations 構造の異なるレイヤーに複数のコントロールを追加します。たとえば、多層防御アプローチでは、多要素認証、ネットワークセグメンテーション、暗号化を組み合わせることができます。

委任管理者

では AWS Organizations、互換性のあるサービスが AWS メンバーアカウントを登録して組織のアカウントを管理し、そのサービスのアクセス許可を管理できます。このアカウントを、そのサービスの委任管理者と呼びます。詳細、および互換性のあるサービスの一覧は、AWS

Organizations ドキュメントの「[AWS Organizationsで利用できるサービス](#)」を参照してください。

トラブルシューティング

アプリケーション、新機能、コードの修正をターゲットの環境で利用できるようにするプロセス。デプロイでは、コードベースに変更を施した後、アプリケーションの環境でそのコードベースを構築して実行します。

開発環境

「[環境](#)」を参照してください。

検出管理

イベントが発生したときに、検出、ログ記録、警告を行うように設計されたセキュリティコントロール。これらのコントロールは副次的な防衛手段であり、実行中の予防的コントロールをすり抜けたセキュリティイベントをユーザーに警告します。詳細については、「AWSでのセキュリティコントロールの実装」の「[検出的コントロール](#)」を参照してください。

開発バリューストリームマッピング (DVSM)

ソフトウェア開発ライフサイクルのスピードと品質に悪影響を及ぼす制約を特定し、優先順位を付けるために使用されるプロセス。DVSM は、もともとリーンマニファクチャリング・プラクティスのために設計されたバリューストリームマッピング・プロセスを拡張したものです。ソフトウェア開発プロセスを通じて価値を創造し、動かすために必要なステップとチームに焦点を当てています。

デジタルツイン

建物、工場、産業機器、生産ラインなど、現実世界のシステムを仮想的に表現したものです。デジタルツインは、予知保全、リモートモニタリング、生産最適化をサポートします。

ディメンションテーブル

[スタースキーマ](#)において、ファクトテーブルの定量データに関するデータ属性が含まれる小さいテーブル。ディメンションテーブルの属性は、通常、テキストフィールド、またはテキストのように扱える個別の数値で示されます。これらの属性は、一般的に、クエリの制約、フィルタリング、結果セットのラベル付けに使用されます。

デザスタ

ワークロードまたはシステムが、導入されている主要な場所でのビジネス目標の達成を妨げるイベント。これらのイベントは、自然災害、技術的障害、または意図しない設定ミスやマルウェア攻撃などの人間の行動の結果である場合があります。

ディザスタリカバリ (DR)

[ディザスタ](#)によるダウンタイムとデータ損失を最小限に抑えるための戦略とプロセス。詳細については、AWS Well-Architected フレームワークの「[でのワークロードのディザスタリカバリ](#)」[AWS: クラウドでのリカバリ](#)」を参照してください。

DML

「[データベース操作言語](#)」を参照してください。

ドメイン駆動型設計

各コンポーネントが提供している変化を続けるドメイン、またはコアビジネス目標にコンポーネントを接続して、複雑なソフトウェアシステムを開発するアプローチ。この概念は、エリック・エヴァンスの著書、Domain-Driven Design: Tackling Complexity in the Heart of Software (ドメイン駆動設計:ソフトウェアの中心における複雑さへの取り組み) で紹介されています (ポストン: Addison-Wesley Professional、2003)。strangler fig パターンでドメイン駆動型設計を使用する方法の詳細については、「[コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET \(ASMX\) ウェブサービスを段階的にモダナイズ](#)」を参照してください。

DR

「[ディザスタリカバリ](#)」を参照してください。

ドリフト検出

ベースライン設定からの偏差を追跡します。たとえば、AWS CloudFormation を使用して[システムリソースのドリフトを検出](#)したり、を使用して AWS Control Tower、ガバナンス要件への準拠に影響する[ランディングゾーンの変更を検出](#)したりできます。

DVSM

「[開発バリューストリームマッピング](#)」を参照してください。

E

EDA

「[探索的データ分析](#)」を参照してください。

EDI

「[電子データ交換](#)」を参照してください。

エッジコンピューティング

IoT ネットワークのエッジにあるスマートデバイスの計算能力を高めるテクノロジー。[クラウドコンピューティング](#)と比較すると、エッジコンピューティングは通信レイテンシーを短縮し、応答時間を改善できます。

電子データ交換 (EDI)

組織間で行う、ビジネスドキュメントの自動交換。詳細については、[「電子データ交換とは」](#)を参照してください。

暗号化

人間が読み取り可能なプレーンテキストデータを暗号文に変換するコンピューティング処理。

暗号化キー

暗号化アルゴリズムが生成した、ランダム化されたビットからなる暗号文字列。キーの長さは決まっておらず、各キーは予測できないように、一意になるように設計されています。

エンディアン

コンピュータメモリにバイトが格納される順序。ビッグエンディアンシステムでは、最上位バイトが最初に格納されます。リトルエンディアンシステムでは、最下位バイトが最初に格納されます。

エンドポイント

[「サービスエンドポイント」](#)を参照してください。

エンドポイントサービス

仮想プライベートクラウド (VPC) 内でホストして、他のユーザーと共有できるサービス。を使用してエンドポイントサービスを作成し AWS PrivateLink、他の AWS アカウント または AWS Identity and Access Management (IAM) プリンシパルにアクセス許可を付与できます。これらのアカウントまたはプリンシパルは、インターフェイス VPC エンドポイントを作成することで、エンドポイントサービスにプライベートに接続できます。詳細については、Amazon Virtual Private Cloud (Amazon VPC) ドキュメントの [「エンドポイントサービスを作成する」](#)を参照してください。

エンタープライズリソースプランニング (ERP)

エンタープライズの主要なビジネスプロセス (会計、[MES](#)、プロジェクト管理など) を自動化および管理するシステム。

エンベロープ暗号化

暗号化キーを、別の暗号化キーを使用して暗号化するプロセス。詳細については、AWS Key Management Service (AWS KMS) ドキュメントの「[エンベロープ暗号化](#)」を参照してください。

環境

実行中のアプリケーションのインスタンス。クラウドコンピューティングにおける一般的な環境の種類は以下のとおりです。

- **開発環境** — アプリケーションのメンテナンスを担当するコアチームのみが利用できる、実行中のアプリケーションのインスタンス。開発環境は、上位の環境に昇格させる変更をテストするときに使用します。このタイプの環境は、テスト環境と呼ばれることもあります。
- **下位環境** — 初期ビルドやテストに使用される環境など、アプリケーションのすべての開発環境。
- **本番環境** — エンドユーザーがアクセスできる、実行中のアプリケーションのインスタンス。CI/CD パイプラインでは、本番環境が最後のデプロイ環境になります。
- **上位環境** — コア開発チーム以外のユーザーがアクセスできるすべての環境。これには、本番環境、本番前環境、ユーザー承認テスト環境などが含まれます。

エピック

アジャイル方法論で、お客様の作業の整理と優先順位付けに役立つ機能カテゴリ。エピックでは、要件と実装タスクの概要についてハイレベルな説明を提供します。例えば、AWS CAF セキュリティエピックには、ID とアクセスの管理、検出コントロール、インフラストラクチャセキュリティ、データ保護、インシデント対応が含まれます。AWS 移行戦略のエピックの詳細については、[プログラム実装ガイド](#)を参照してください。

ERP

「[エンタープライズリソース計画](#)」を参照してください。

探索的データ分析 (EDA)

データセットを分析してその主な特性を理解するプロセス。お客様は、データを収集または集計してから、パターンの検出、異常の検出、および前提条件のチェックのための初期調査を実行します。EDA は、統計の概要を計算し、データの可視化を作成することによって実行されます。

F

ファクトテーブル

[スタースキーマ](#)の中央にあるテーブル。ビジネスオペレーションに関する定量的データが保存されます。一般的に、ファクトテーブルは、2種類の列で構成されます。1つは測定値が含まれる列、もう1つはディメンションテーブルへの外部キーが含まれる列です。

フェイルファスト

開発ライフサイクルを短縮するために、頻繁かつ段階的にテストを行う哲学であり、アジャイルアプローチでは、この考え方がきわめて重要です。

障害分離境界

では AWS クラウド、障害の影響を制限し、ワークロードの耐障害性を高めるのに役立つアベイラビリティゾーン AWS リージョン、コントロールプレーン、データプレーンなどの境界。詳細については、「[AWS 障害分離境界](#)」を参照してください。

機能ブランチ

「[ブランチ](#)」を参照してください。

特徴量

お客様が予測に使用する入力データ。例えば、製造コンテキストでは、特徴量は製造ラインから定期的にキャプチャされるイメージの可能性もあります。

特徴量重要度

モデルの予測に対する特徴量の重要性。これは通常、Shapley Additive Deskonations (SHAP) や積分勾配など、さまざまな手法で計算できる数値スコアで表されます。詳細については、「[を使用した機械学習モデルの解釈可能性 AWS](#)」を参照してください。

機能変換

追加のソースによるデータのエンリッチ化、値のスケーリング、単一のデータフィールドからの複数の情報セットの抽出など、機械学習プロセスのデータを最適化すること。これにより、機械学習モデルはデータの恩恵を受けることができます。例えば、「2021-05-27 00:15:37」の日付を「2021年」、「5月」、「木」、「15」に分解すると、学習アルゴリズムがさまざまなデータコンポーネントに関連する微妙に異なるパターンを学習するのに役立ちます。

数ショットプロンプト

[LLM](#) に、タスクと望ましい出力を示す例を少数提示した後に、類似のタスクを実行させること。この手法は、プロンプトに記述された例(ショット)からモデルが学習する「インコンテキスト学

習」の一種です。数ショットプロンプトは、特定のフォーマット、推論、専門知識が必要なタスクに効果的です。「[ゼロショットプロンプト](#)」も参照してください。

FGAC

「[きめ細かなアクセス制御](#)」を参照してください。

きめ細かなアクセス制御 (FGAC)

複数の条件を使用してアクセス要求を許可または拒否すること。

フラッシュカット移行

[変更データのキャプチャ](#)による継続的なデータ複製を利用して、段階的なアプローチではなく、可能な限り短時間でデータを移行するデータベース移行方法。目的はダウンタイムを最小限に抑えることです。

FM

「[基盤モデル](#)」を参照してください。

基盤モデル (FM)

大規模な深層学習ニューラルネットワークであり、一般化およびラベル付けされていないデータからなる大規模データセットでトレーニングされています。FMにより、言語理解、テキストおよび画像生成、自然言語での会話といった、一般的な各種タスクを実行できます。詳細については、「[基盤モデルとは何ですか?](#)」を参照してください。

G

生成 AI

[AI](#) モデルのサブセット。大量のデータでトレーニングされており、シンプルなテキストプロンプトを使用して、画像、動画、テキスト、オーディオなどの新しいコンテンツやアーティファクトを作成できます。詳細については、「[生成 AI とは何ですか?](#)」を参照してください。

ジオブロッキング

「[地理的制限](#)」を参照してください。

地理的制限 (ジオブロッキング)

特定の国のユーザーがコンテンツ配信にアクセスできないようにするための、Amazon CloudFront のオプション。アクセスを許可する国と禁止する国は、許可リストまたは禁止リスト

を使って指定します。詳細については、CloudFront ドキュメントの「[コンテンツの地理的ディストリビューションの制限](#)」を参照してください。

Gitflow ワークフロー

下位環境と上位環境が、ソースコードリポジトリでそれぞれ異なるブランチを使用する方法。Gitflow ワークフローは古いと見なされている方法であり、[トランクベースのワークフロー](#)は推奨されている新しい方法です。

ゴールデンイメージ

システムまたはソフトウェアのスナップショットであり、システムまたはソフトウェアの新規インスタンスをデプロイするテンプレートとして使用されます。製造の例で言えば、ゴールデンイメージを使用すると、複数のデバイスにソフトウェアをプロビジョニングして、デバイス製造オペレーションの速度、スケーラビリティ、生産性を向上させることができます。

グリーンフィールド戦略

新しい環境に既存のインフラストラクチャが存在しないこと。システムアーキテクチャにグリーンフィールド戦略を導入する場合、既存のインフラストラクチャ (別名 [ブラウンフィールド](#)) との互換性の制約を受けることなく、あらゆる新しいテクノロジーを選択できます。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

ガードレール

組織単位 (OU) 全般のリソース、ポリシー、コンプライアンスを管理するのに役立つ概略的なルール。予防ガードレールは、コンプライアンス基準に一致するようにポリシーを実施します。これらは、サービスコントロールポリシーと IAM アクセス許可の境界を使用して実装されます。検出ガードレールは、ポリシー違反やコンプライアンス上の問題を検出し、修復のためのアラートを発信します。これらは AWS Config、AWS Security Hub CSPM、Amazon GuardDuty、AWS Trusted Advisor Amazon Inspector、およびカスタム AWS Lambda チェックを使用して実装されます。

H

HA

「[高可用性](#)」を参照してください。

異種混在データベースの移行

別のデータベースエンジンを使用するターゲットデータベースへお客様の出典データベースの移行 (例えば、Oracle から Amazon Aurora)。異種間移行は通常、アーキテクチャの再設計作業の一部であり、スキーマの変換は複雑なタスクになる可能性があります。[AWS は、スキーマの変換に役立つ AWS SCTを提供します。](#)

高可用性 (HA)

課題や災害が発生した場合に、介入なしにワークロードを継続的に運用できること。HA システムは、自動的にフェイルオーバーし、一貫して高品質のパフォーマンスを提供し、パフォーマンスへの影響を最小限に抑えながらさまざまな負荷や障害を処理するように設計されています。

ヒストリアンのモダナイゼーション

製造業のニーズによりよく応えるために、オペレーションテクノロジー (OT) システムをモダナイズし、アップグレードするためのアプローチ。ヒストリアンは、工場内のさまざまなソースからデータを収集して保存するために使用されるデータベースの一種です。

ホールドアウトデータ

[機械学習](#)モデルのトレーニング用データセットから保留される、ラベル付き履歴データの一部。ホールドアウトデータを使用すると、モデル予測をホールドアウトデータと比較して、モデルのパフォーマンスを評価できます。

同種データベースの移行

お客様の出典データベースを、同じデータベースエンジンを共有するターゲットデータベース (Microsoft SQL Server から Amazon RDS for SQL Server など) に移行する。同種間移行は、通常、リホストまたはリプラットフォーム化の作業の一部です。ネイティブデータベースユーティリティを使用して、スキーマを移行できます。

ホットデータ

リアルタイムデータや最近の翻訳データなど、頻繁にアクセスされるデータ。通常、このデータには高速なクエリ応答を提供する高性能なストレージ階層またはクラスが必要です。

ホットフィックス

本番環境の重大な問題を修正するために緊急で配布されるプログラム。緊急性が高いため、通常の DevOps のリリースワークフローからは外れた形で実施されます。

ハイパーケア期間

カットオーバー直後、移行したアプリケーションを移行チームがクラウドで管理、監視して問題に対処する期間。通常、この期間は 1~4 日です。ハイパーケア期間が終了すると、アプリケーションに対する責任は一般的に移行チームからクラウドオペレーションチームに移ります。

I

laC

「[Infrastructure as Code](#)」を参照してください。

ID ベースのポリシー

AWS クラウド 環境内のアクセス許可を定義する 1 つ以上の IAM プリンシパルにアタッチされたポリシー。

アイドル状態のアプリケーション

90 日間の平均的な CPU およびメモリ使用率が 5~20% のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するか、オンプレミスに保持するのが一般的です。

IIoT

「[インダストリアル IoT](#)」を参照してください。

イミュータブルインフラストラクチャ

既存インフラストラクチャの更新、パッチ適用、変更などを行わずに、本番環境ワークロードに使用する新規インフラストラクチャをデプロイするモデル。本質的に、イミュータブルインフラストラクチャは、[ミュータブルインフラストラクチャ](#)よりも一貫性、信頼性、予測性に優れています。詳細については、AWS Well-Architected フレームワークにある「[イミュータブルインフラストラクチャを使用してデプロイする](#)」のベストプラクティスを参照してください。

インバウンド (受信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーションの外部からネットワーク接続を受け入れ、検査し、ルーティングする VPC。[AWS Security Reference Architecture](#) では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

I

増分移行

アプリケーションを 1 回ですべてカットオーバーするのではなく、小さい要素に分けて移行するカットオーバー戦略。例えば、最初は少数のマイクロサービスまたはユーザーのみを新しいシステムに移行する場合があります。すべてが正常に機能することを確認できたら、残りのマイクロサービスやユーザーを段階的に移行し、レガシーシステムを廃止できるようにします。この戦略により、大規模な移行に伴うリスクが軽減されます。

インダストリー 4.0

2016 年に [Klaus Schwab](#) 氏が提唱した用語で、接続、リアルタイムデータ、オートメーション、分析、AI/ML の進歩による、ビジネスプロセスのモダナイズを意味します。

インフラストラクチャ

アプリケーションの環境に含まれるすべてのリソースとアセット。

Infrastructure as Code (IaC)

アプリケーションのインフラストラクチャを一連の設定ファイルを使用してプロビジョニングし、管理するプロセス。IaC は、新しい環境を再現可能で信頼性が高く、一貫性のあるものにするため、インフラストラクチャを一元的に管理し、リソースを標準化し、スケールを迅速に行えるように設計されています。

インダストリアル IoT (IIoT)

製造、エネルギー、自動車、ヘルスケア、ライフサイエンス、農業などの産業部門におけるインターネットに接続されたセンサーやデバイスの使用。詳細については、「[インダストリアル IoT \(IIoT\) デジタルトランスフォーメーション戦略の構築](#)」を参照してください。

インスペクション VPC

AWS マルチアカウントアーキテクチャでは、VPC (同一または異なる 内 AWS リージョン)、インターネット、オンプレミスネットワーク間のネットワークトラフィックの検査を管理する一元化された VPCs。 [AWS Security Reference Architecture](#) では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

IoT

インターネットまたはローカル通信ネットワークを介して他のデバイスやシステムと通信する、センサーまたはプロセッサが組み込まれた接続済み物理オブジェクトのネットワーク。詳細については、「[IoT とは](#)」を参照してください。

解釈可能性

機械学習モデルの特性で、モデルの予測がその入力にどのように依存するかを人間が理解できる度合いを表します。詳細については、[「を使用した機械学習モデルの解釈可能性 AWS」](#)を参照してください。

IoT

[「IoT」](#)を参照してください。

IT 情報ライブラリ (ITIL)

IT サービスを提供し、これらのサービスをビジネス要件に合わせるための一連のベストプラクティス。ITIL は ITSM の基盤を提供します。

IT サービス管理 (ITSM)

組織の IT サービスの設計、実装、管理、およびサポートに関連する活動。クラウドオペレーションと ITSM ツールの統合については、[オペレーション統合ガイド](#)を参照してください。

ITIL

[「IT 情報ライブラリ」](#)を参照してください。

ITSM

[「IT サービス管理」](#)を参照してください。

L

ラベルベースアクセス制御 (LBAC)

強制アクセス制御 (MAC) の実装で、ユーザーとデータ自体にそれぞれセキュリティラベル値が明示的に割り当てられます。ユーザーセキュリティラベルとデータセキュリティラベルが交差する部分によって、ユーザーに表示される行と列が決まります。

ランディングゾーン

ランディングゾーンは、スケーラブルで安全な、適切に設計されたマルチアカウント AWS 環境です。これは、組織がセキュリティおよびインフラストラクチャ環境に自信を持ってワークロードとアプリケーションを迅速に起動してデプロイできる出発点です。ランディングゾーンの詳細については、[「安全でスケーラブルなマルチアカウント AWS 環境のセットアップ」](#)を参照してください。

大規模言語モデル (LLM)

大量のデータで事前トレーニングされた深層学習 AI モデル。LLM では、質問への回答、ドキュメントの要約、他言語へのテキスト翻訳、文を完成させるなど、さまざまなタスクを実行できます。詳細については、「[大規模言語モデル \(LLM\) とは何ですか?](#)」を参照してください。

大規模な移行

300 台以上のサーバの移行。

LBAC

「[ラベルベースアクセス制御](#)」を参照してください。

最小特権

タスクの実行には必要最低限の権限を付与するという、セキュリティのベストプラクティス。詳細については、IAM ドキュメントの「[最小特権アクセス許可を適用する](#)」を参照してください。

リフトアンドシフト

「[7 Rs](#)」を参照してください。

リトルエンディアンシステム

最下位バイトを最初に格納するシステム。「[エンディアン性](#)」もご覧ください。

LLM

「[大規模言語モデル](#)」を参照してください。

下位環境

「[環境](#)」を参照してください。

M

機械学習 (ML)

パターン認識と学習にアルゴリズムと手法を使用する人工知能の一種。ML は、モノのインターネット (IoT) データなどの記録されたデータを分析して学習し、パターンに基づく統計モデルを生成します。詳細については、「[機械学習](#)」を参照してください。

メインブランチ

「[ブランチ](#)」を参照してください。

マルウェア

コンピュータのセキュリティやプライバシーを侵害するように設計されたソフトウェア。マルウェアは、コンピュータシステムの中断、機密情報の漏洩、不正アクセスを招く可能性があります。マルウェアの例には、ウイルス、ワーム、ランサムウェア、トロイの木馬、スパイウェア、キーロガーなどがあります。

マネージドサービス

AWS のサービスはインフラストラクチャレイヤー、オペレーティングシステム、プラットフォーム AWS を運用し、エンドポイントにアクセスしてデータを保存および取得します。マネージドサービスの例として、Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB が挙げられます。このサービスは、抽象化されたサービスとも呼ばれます。

製造実行システム (MES)

生産プロセスを追跡、モニタリング、文書化、制御するソフトウェアシステムであり、工場では、これによって、原材料から製品を完成させます。

MAP

[「Migration Acceleration Program」](#) を参照してください。

メカニズム

ツールを作成してその導入を推進し、導入結果を調べて調整を行うための包括的なプロセス。メカニズムとは、運用中にそれ自体を強化し改善するサイクルを意味します。詳細については、AWS 「Well-Architected フレームワーク」の [「メカニズムの構築」](#) を参照してください。

メンバーアカウント

組織の一部である管理アカウント AWS アカウント 以外のすべて AWS Organizations。アカウントが組織のメンバーになることができるのは、一度に 1 つのみです。

MES

[「製造実行システム」](#) を参照してください。

Message Queuing Telemetry Transport (MQTT)

[発行/サブスクリプション](#) のパターンに基づく、軽量のマシンツーマシン (M2M) 通信プロトコルであり、リソースに限りのある [IoT](#) デバイスに使用されます。

マイクロサービス

明確に定義された API を介して通信し、通常は小規模な自己完結型のチームが所有する、小規模で独立したサービスです。例えば、保険システムには、販売やマーケティングなどのビジネス

機能、または購買、請求、分析などのサブドメインにマッピングするマイクロサービスが含まれる場合があります。マイクロサービスの利点には、俊敏性、柔軟なスケーリング、容易なデプロイ、再利用可能なコード、回復力などがあります。詳細については、[AWS「サーバーレスサービスを使用したマイクロサービスの統合」](#)を参照してください。

マイクロサービスアーキテクチャ

各アプリケーションプロセスをマイクロサービスとして実行する独立したコンポーネントを使用してアプリケーションを構築するアプローチ。これらのマイクロサービスは、軽量 API を使用して、明確に定義されたインターフェイスを介して通信します。このアーキテクチャの各マイクロサービスは、アプリケーションの特定の機能に対する需要を満たすように更新、デプロイ、およびスケーリングできます。詳細については、「[でのマイクロサービスの実装 AWS](#)」を参照してください。

Migration Acceleration Program (MAP)

組織がクラウドに移行するための強力な運用基盤を構築し、移行の初期コストを相殺するのに役立つコンサルティングサポート、トレーニング、サービスを提供する AWS プログラム。MAP には、組織的な方法でレガシー移行を実行するための移行方法論と、一般的な移行シナリオを自動化および高速化する一連のツールが含まれています。

大規模な移行

アプリケーションポートフォリオの大部分を次々にクラウドに移行し、各ウェーブでより多くのアプリケーションを高速に移動させるプロセス。この段階では、以前の段階から学んだベストプラクティスと教訓を使用して、移行ファクトリー チーム、ツール、プロセスのうち、オートメーションとアジャイルデリバリーによってワークロードの移行を合理化します。これは、[AWS 移行戦略](#) の第 3 段階です。

移行ファクトリー

自動化された俊敏性のあるアプローチにより、ワークロードの移行を合理化する部門横断的なチーム。移行ファクトリーチームには、通常、運用、ビジネスアナリストおよび所有者、移行エンジニア、デベロッパー、およびスプリントで作業する DevOps プロフェッショナルが含まれます。エンタープライズアプリケーションポートフォリオの 20~50% は、ファクトリーのアプローチによって最適化できる反復パターンで構成されています。詳細については、このコンテンツセットの[移行ファクトリーに関する解説](#)と [Cloud Migration Factory ガイド](#)を参照してください。

移行メタデータ

移行を完了するために必要なアプリケーションおよびサーバーに関する情報。移行パターンごとに、異なる一連の移行メタデータが必要です。移行メタデータの例としては、ターゲットサブネット、セキュリティグループ、AWS アカウントなどがあります。

移行パターン

移行戦略、移行先、および使用する移行アプリケーションまたはサービスを詳述する、反復可能な移行タスク。例: AWS Application Migration Service を使用して Amazon EC2 への移行をリホストします。

Migration Portfolio Assessment (MPA)

オンラインツール。これによって、AWS クラウドに移行するビジネスケースの検証に必要な情報を得られます。MPA は、詳細なポートフォリオ評価 (サーバーの適切なサイジング、価格設定、TCO 比較、移行コスト分析) および移行プラン (アプリケーションデータの分析とデータ収集、アプリケーションのグループ化、移行の優先順位付け、およびウェーブプランニング) を提供します。[MPA ツール](#) (ログインが必要) は、すべての AWS コンサルタントと APN パートナー コンサルタントが無料で利用できます。

移行準備状況評価 (MRA)

AWS CAF を使用して、組織のクラウド準備状況に関するインサイトを取得し、長所と短所を特定し、特定されたギャップを埋めるためのアクションプランを構築するプロセス。詳細については、[移行準備状況ガイド](#)を参照してください。MRA は、[AWS 移行戦略](#)の第一段階です。

移行戦略

ワークロードを AWS クラウドに移行するために使用するアプローチ。詳細については、この用語集の [7 Rs](#) エントリと、「[組織を動員して大規模な移行を加速する](#)」を参照してください。

ML

「[機械学習](#)」を参照してください。

モダナイゼーション

古い (レガシーまたはモノリシック) アプリケーションとそのインフラストラクチャをクラウド内の俊敏で弾力性のある高可用性システムに変換して、コストを削減し、効率を高め、イノベーションを活用します。詳細については、「[AWS クラウドでのアプリケーションのモダナイズ戦略](#)」を参照してください。

モダナイゼーション準備状況評価

組織のアプリケーションのモダナイゼーションの準備状況を判断し、利点、リスク、依存関係を特定し、組織がこれらのアプリケーションの将来の状態をどの程度適切にサポートできるかを決定するのに役立つ評価。評価の結果として、ターゲットアーキテクチャのブループリント、モダナイゼーションプロセスの開発段階とマイルストーンを詳述したロードマップ、特定されたギャップに対処するためのアクションプランが得られます。詳細については、「[AWS クラウドでのアプリケーションのモダナイゼーションの準備状況を評価する](#)」を参照してください。

モノリシックアプリケーション (モノリス)

緊密に結合されたプロセスを持つ単一のサービスとして実行されるアプリケーション。モノリシックアプリケーションにはいくつかの欠点があります。1つのアプリケーション機能エクスペリエンスの需要が急増する場合は、アーキテクチャ全体をスケーリングする必要があります。モノリシックアプリケーションの特徴を追加または改善することは、コードベースが大きくなると複雑になります。これらの問題に対処するには、マイクロサービスアーキテクチャを使用できます。詳細については、「[モノリスをマイクロサービスに分解する](#)」を参照してください。

MPA

「[Migration Portfolio Assessment](#)」を参照してください。

MQTT

「[Message Queuing Telemetry Transport](#)」を参照してください。

多クラス分類

複数のクラスの予測を生成するプロセス (2 つ以上の結果の 1 つを予測します)。例えば、機械学習モデルが、「この製品は書籍、自動車、電話のいずれですか?」または、「このお客様にとって最も関心のある商品のカテゴリはどれですか?」と聞くかもしれません。

ミュータブルなインフラストラクチャ

本番ワークロードに使用する既存のインフラストラクチャを更新および変更するためのモデル。Well-Architected AWS フレームワークでは、一貫性、信頼性、予測可能性を向上させるために、[イミュータブルインフラストラクチャ](#)の使用をベストプラクティスとして推奨しています。

O

OAC

「[オリジンアクセス制御](#)」を参照してください。

OAI

「[オリジンアクセスアイデンティティ](#)」を参照してください。

OCM

「[組織変更管理](#)」を参照してください。

オフライン移行

移行プロセス中にソースワークロードを停止させる移行方法。この方法はダウンタイムが長くなるため、通常は重要ではない小規模なワークロードに使用されます。

OI

「[オペレーション統合](#)」を参照してください。

Ola

「[オペレーショナルレベルアグリーメント](#)」を参照してください。

オンライン移行

ソースワークロードをオフラインにせずにターゲットシステムにコピーする移行方法。ワークロードに接続されているアプリケーションは、移行中も動作し続けることができます。この方法はダウンタイムがゼロから最小限で済むため、通常は重要な本番稼働環境のワークロードに使用されます。

OPC-UA

「[Open Process Communications - Unified Architecture](#)」を参照してください。

Open Process Communications - Unified Architecture (OPC-UA)

産業オートメーション用のマシンツーマシン (M2M) 通信プロトコル。OPC-UA により、相互運用の際に、データ暗号化、認証、認可の各スキームを標準化できます。

オペレーショナルレベルアグリーメント (OLA)

サービスレベルアグリーメント (SLA) をサポートするために、どの機能的 IT グループが互いに提供することを約束するかを明確にする契約。

運用準備状況レビュー (ORR)

質問と関連するベストプラクティスのチェックリスト。インシデントや起こり得る障害を理解、評価、防止したり、その範囲を縮小したりする際に役立ちます。詳細については、AWS Well-Architected フレームワークの「[Operational Readiness Reviews \(ORR\)](#)」を参照してください。

運用テクノロジー (OT)

産業オペレーション、機器、インフラストラクチャを制御するために物理環境と連携させるハードウェアおよびソフトウェアシステム。製造分野では、[Industry 4.0](#) への変革を進める上で、OT と情報技術 (IT) システムの統合に焦点が当てられています。

オペレーション統合 (OI)

クラウドでオペレーションをモダナイズするプロセスには、準備計画、オートメーション、統合が含まれます。詳細については、[オペレーション統合ガイド](#)を参照してください。

組織の証跡

組織 AWS アカウント 内のすべてのイベント AWS CloudTrail をログに記録することによって作成された証跡 AWS Organizations。証跡は、組織に含まれている各 AWS アカウントに作成され、各アカウントのアクティビティを追跡します。詳細については、CloudTrail ドキュメントの「[組織の証跡の作成](#)」を参照してください。

組織変更管理 (OCM)

人材、文化、リーダーシップの観点から、主要な破壊的なビジネス変革を管理するためのフレームワーク。OCM は、変化の導入を加速し、移行問題に対処し、文化や組織の変化を推進することで、組織が新しいシステムと戦略の準備と移行するのを支援します。AWS 移行戦略では、クラウド導入プロジェクトに必要な変化のスピードにより、このフレームワークは人材アクセラレーションと呼ばれます。詳細については、[OCM ガイド](#)を参照してください。

オリジンアクセス制御 (OAC)

Amazon Simple Storage Service (Amazon S3) コンテンツを保護するための、CloudFront のアクセス制限の強化オプション。OAC は AWS リージョン、すべての S3 バケット、AWS KMS (SSE-KMS) によるサーバー側の暗号化、S3 バケットへの動的 PUT および DELETE リクエストをサポートします。

オリジンアクセスアイデンティティ (OAI)

CloudFront の、Amazon S3 コンテンツを保護するためのアクセス制限オプション。OAI を使用すると、CloudFront が、Amazon S3 に認証可能なプリンシパルを作成します。認証されたプリンシパルは、S3 バケット内のコンテンツに、特定の CloudFront ディストリビューションを介してのみアクセスできます。[OAC](#) も併せて参照してください。OAC では、より詳細な、強化されたアクセス制御が可能です。

ORR

「[運用準備状況レビュー](#)」を参照してください。

OT

「[運用テクノロジー](#)」を参照してください。

アウトバウンド (送信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーション内から開始されたネットワーク接続を処理する VPC。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

P

アクセス許可の境界

ユーザーまたはロールが使用できるアクセス許可の上限を設定する、IAM プリンシパルにアタッチされる IAM 管理ポリシー。詳細については、IAM ドキュメントの[アクセス許可の境界](#)を参照してください。

個人を特定できる情報 (PII)

直接閲覧した場合、または他の関連データと組み合わせた場合に、個人の身元を合理的に推測するために使用できる情報。PII の例には、氏名、住所、連絡先情報などがあります。

PII

「[個人を特定できる情報](#)」を参照してください。

プレイブック

クラウドでのコアオペレーション機能の提供など、移行に関連する作業を取り込む、事前定義された一連のステップ。プレイブックは、スクリプト、自動ランブック、またはお客様のモダナイズされた環境を運用するために必要なプロセスや手順の要約などの形式をとることができます。

PLC

「[プログラマブルロジックコントローラー](#)」を参照してください。

PLM

「[製品ライフサイクル管理](#)」を参照してください。

ポリシー

次の操作を可能にするオブジェクト: アクセス許可を定義する ([ID ベースのポリシー](#)を参照)。アクセス条件を指定する ([リソースベースのポリシー](#)を参照)。AWS Organizations の組織における全アカウントにアクセス許可の上限を定義する ([サービスコントロールポリシー](#)を参照)。

多言語の永続性

データアクセスパターンやその他の要件に基づいて、マイクロサービスのデータストレージテクノロジーを個別に選択します。マイクロサービスが同じデータストレージテクノロジーを使用している場合、実装上の問題が発生したり、パフォーマンスが低下する可能性があります。マイクロサービスは、要件に最も適合したデータストアを使用すると、より簡単に実装でき、パフォーマンスとスケーラビリティが向上します。

ポートフォリオ評価

移行を計画するために、アプリケーションポートフォリオの検出、分析、優先順位付けを行うプロセス。詳細については、「[移行の準備状況の評価](#)」を参照してください。

述語

true または false を返すためのクエリ条件。一般的に、WHERE 句に記述されます。

述語プッシュダウン

データベースクエリを最適化する手法。これによって、転送前にクエリ内のデータをフィルタリングします。この手法を取ると、リレーショナルデータベースから取得し処理する必要のあるデータの量が減少するため、クエリのパフォーマンスが向上します。

予防的コントロール

イベントの発生を防ぐように設計されたセキュリティコントロール。このコントロールは、ネットワークへの不正アクセスや好ましくない変更を防ぐ最前線の防御です。詳細については、「AWSでのセキュリティコントロールの実装」の「[予防的コントロール](#)」を参照してください。

プリンシパル

アクションを実行し AWS、リソースにアクセスできるのエンティティ。このエンティティは通常、IAM AWS アカウントロール、またはユーザーのルートユーザーです。詳細については、IAM ドキュメントの「[ロールに関する用語と概念](#)」にあるプリンシパルを参照してください。

プライバシーバイデザイン

開発プロセス全体を通してプライバシーが考慮されているシステムエンジニアリングのアプローチ。

プライベートホストゾーン

1 つ以上の VPC 内のドメインとそのサブドメインへの DNS クエリに対し、Amazon Route 53 がどのように応答するかに関する情報を保持するコンテナ。詳細については、Route 53 ドキュメントの「[プライベートホストゾーンの使用](#)」を参照してください。

プロアクティブコントロール

非準拠リソースのデプロイ防止を目的とした[セキュリティコントロール](#)。このコントロールにより、プロビジョニング前にリソースをスキャンします。コントロールに準拠していないリソースは、プロビジョニングされません。詳細については、AWS Control Tower ドキュメントの「[コントロールリファレンスガイド](#)」および「[セキュリティコントロールの実装](#)」の「[プロアクティブコントロール](#)」を参照してください。 AWS

製品ライフサイクル管理 (PLM)

製品の設計、開発、発売から、成長、成熟、衰退、廃棄に至る、製品のライフサイクル全体を通してデータとプロセスを管理すること。

本番環境

「[環境](#)」を参照してください。

プログラマブルロジックコントローラー (PLC)

製造分野で使用される、信頼性と適応性に優れたコンピュータであり、これによって、マシンをモニタリングするとともに、製造プロセスを自動化します。

プロンプトチェイニング

1 つの [LLM](#) プロンプトによる出力を次のプロンプトの入力に使用して、より良いレスポンスを生成します。この手法を使用すると、複雑なタスクをサブタスクに分割したり、事前レスポンスを繰り返し改良または拡張したりできます。これによって、モデルのレスポンスの精度と関連性が向上し、粒度の高いパーソナライズされた結果を得られます。

仮名化

データセット内の個人識別子をプレースホルダー値に置き換えるプロセス。仮名化は個人のプライバシー保護に役立ちます。仮名化されたデータは、依然として個人データとみなされます。

発行/サブスクライブ (pub/sub)

マイクロサービス間の非同期通信を可能にするパターン。これにより、スケーラビリティと応答性を向上させます。例えば、マイクロサービスベースの [MES](#) の場合、マイクロサービスは、他のマイクロサービスがサブスクライブ可能なチャンネルにイベントメッセージを発行できます。このシステムでは、発行サービスの変更なしに、新規マイクロサービスを追加できます。

Q

クエリプラン

手順などの一連のステップであり、SQL リレーショナルデータベースシステムのデータにアクセスするために使用されます。

クエリプランのリグレッション

データベースサービスのオプティマイザーが、データベース環境に特定の変更が加えられる前に選択されたプランよりも最適性の低いプランを選択すること。これは、統計、制限事項、環境設定、クエリパラメータのバインディングの変更、およびデータベースエンジンの更新などが原因である可能性があります。

R

RACI マトリックス

「[実行責任者、説明責任者、協業先、報告先 \(RACI\)](#)」を参照してください。

RAG

「[検索拡張生成](#)」を参照してください。

ランサムウェア

決済が完了するまでコンピュータシステムまたはデータへのアクセスをブロックするように設計された、悪意のあるソフトウェア。

RASCI マトリックス

「[実行責任者、説明責任者、協業先、報告先 \(RACI\)](#)」を参照してください。

RCAC

「[行と列のアクセス制御](#)」を参照してください。

リードレプリカ

読み取り専用で使用されるデータベースのコピー。クエリをリードレプリカにルーティングして、プライマリデータベースへの負荷を軽減できます。

リアーキテクト

「[7 Rs](#)」を参照してください。

目標復旧時点 (RPO)

最後のデータリカバリポイントからの最大許容時間です。これにより、最後の回復時点からサービスが中断されるまでの間に許容できるデータ損失の程度が決まります。

目標復旧時間 (RTO)

サービスが中断から復旧までの最大許容遅延時間。

リファクタリング

「[7 Rs](#)」を参照してください。

リージョン

地理的エリア内の AWS リソースのコレクション。各 AWS リージョンは、耐障害性、安定性、耐障害性を提供するために、他のから分離され、独立しています。詳細については、「[アカウントが使用できる AWS リージョンを指定する](#)」を参照してください。

リグレッション

数値を予測する機械学習手法。例えば、「この家はどれくらいの値段で売れるでしょうか?」という問題を解決するために、機械学習モデルは、線形回帰モデルを使用して、この家に関する既知の事実 (平方フィートなど) に基づいて家の販売価格を予測できます。

リホスト

「[7 Rs](#)」を参照してください。

リリース

デプロイプロセスで、変更を本番環境に昇格させること。

再配置

「[7 Rs](#)」を参照してください。

リプラットフォーム

「[7 Rs](#)」を参照してください。

再購入

「[7 Rs](#)」を参照してください。

回復性

中断に抵抗または中断から回復するアプリケーションの機能。AWS クラウドでの回復力を計画する際には、一般的に、[高可用性](#)と[ディザスタリカバリ](#)が考慮されます。詳細については、「[AWS クラウドの耐障害性](#)」を参照してください。

リソースベースのポリシー

Amazon S3 バケット、エンドポイント、暗号化キーなどのリソースにアタッチされたポリシー。このタイプのポリシーは、アクセスが許可されているプリンシパル、サポートされているアクション、その他の満たすべき条件を指定します。

実行責任者、説明責任者、協業先、報告先 (RACI) に基づくマトリックス

移行活動とクラウド運用に関わるすべての関係者の役割と責任を定義したマトリックス。マトリックスの名前は、マトリックスで定義されている責任の種類、すなわち責任 (R)、説明責任 (A)、協議 (C)、情報提供 (I) に由来します。サポート (S) タイプはオプションです。サポートが含まれる場合は RASCI マトリックスと呼ばれ、含まれない場合は RACI マトリックスと呼ばれます。

レスポンスコントロール

有害事象やセキュリティベースラインからの逸脱について、修復を促すように設計されたセキュリティコントロール。詳細については、「AWSでのセキュリティコントロールの実装」の「[レスポンスコントロール](#)」を参照してください。

保持

「[7 Rs](#)」を参照してください。

廃止

「[7 Rs](#)」を参照してください。

検索拡張生成 (RAG)

[生成 AI](#) の技術。これにより、[LLM](#) では、レスポンスの生成前に、トレーニングデータソースの外部にある信頼できるデータソースが参照されます。例えば、RAG モデルによって、組織のナレッジベースまたはカスタムデータのセマンティック検索を実行できる場合があります。細については、「[RAG \(検索拡張生成\) とは何ですか?](#)」を参照してください。

ローテーション

定期的に[シークレット情報](#)を更新して、攻撃者が認証情報にアクセスするのをより困難にするプロセス。

行と列のアクセス制御 (RCAC)

アクセスルールが定義された、基本的で柔軟な SQL 表現の使用。RCAC は行権限と列マスクで構成されています。

RPO

「[目標復旧時点](#)」を参照してください。

RTO

「[目標復旧時間](#)」を参照してください。

ランブック

特定のタスクを実行するために必要な手動または自動化された一連の手順。これらは通常、エラー率の高い反復操作や手順を合理化するために構築されています。

S

SAML 2.0

多くの ID プロバイダー (IdP) が使用しているオープンスタンダード。この機能を使用すると、フェデレーテッドシングルサインオン (SSO) が有効になるため、ユーザーは組織内のすべてのユーザーを IAM で作成しなくても、AWS マネジメントコンソールにログインしたり AWS、API オペレーションを呼び出すことができます。SAML 2.0 ベースのフェデレーションの詳細については、IAM ドキュメントの「[SAML 2.0 ベースのフェデレーションについて](#)」を参照してください。

SCADA

「[監視制御とデータ取得](#)」を参照してください。

SCP

「[サービスコントロールポリシー](#)」を参照してください。

シークレット

暗号化された形式で保存する AWS Secrets Manager パスワードやユーザー認証情報などの機密情報または制限付き情報。シークレット値とそのメタデータで構成されます。シークレット値には、バイナリ、1 つの文字列、複数の文字列を指定できます。詳細については、Secrets Manager ドキュメントの「[Secrets Manager シークレットの概要](#)」を参照してください。

セキュリティバイデザイン

開発プロセス全体を通してセキュリティが考慮されているシステムエンジニアリングのアプローチ。

セキュリティコントロール

脅威アクターによるセキュリティ脆弱性の悪用を防止、検出、軽減するための、技術上または管理上のガードレール。セキュリティコントロールには、主に 4 つの種類があります。4 つとは、[予防](#)、[検出](#)、[レスポンス](#)、[プロアクティブ](#)です。

セキュリティ強化

アタックサーフェスを狭めて攻撃への耐性を高めるプロセス。このプロセスには、不要になったリソースの削除、最小特権を付与するセキュリティのベストプラクティスの実装、設定ファイル内の不要な機能の無効化、といったアクションが含まれています。

Security Information and Event Management (SIEM) システム

セキュリティ情報管理 (SIM) とセキュリティイベント管理 (SEM) のシステムを組み合わせたツールとサービス。SIEM システムは、サーバー、ネットワーク、デバイス、その他ソースからデータを収集、モニタリング、分析して、脅威やセキュリティ違反を検出し、アラートを発信します。

セキュリティレスポンスの自動化

セキュリティイベントへの自動レスポンスまたは自動修復を目的として、事前定義およびプログラムされたアクション。これらの自動化は、セキュリティのベストプラクティスを実装するのに役立つ[検出的](#)または[応答的](#)な AWS セキュリティコントロールとして機能します。自動レスポンスアクションの例には、VPC セキュリティグループの変更、Amazon EC2 インスタンスへのパッチ適用、認証情報の更新などがあります。

サーバー側の暗号化

送信先で、それ AWS のサービスを受け取る によるデータの暗号化。

サービスコントロールポリシー (SCP)

AWS Organizationsの組織内の、すべてのアカウントのアクセス許可を一元的に管理するポリシー。SCP は、管理者がユーザーまたはロールに委任するアクションに、ガードレールを定義したり、アクションの制限を設定したりします。SCP は、許可リストまたは拒否リストとして、許可または禁止するサービスやアクションを指定する際に使用できます。詳細については、AWS Organizations ドキュメントの「[サービスコントロールポリシー](#)」を参照してください。

サービスエンドポイント

のエンドポイントの URL AWS のサービス。ターゲットサービスにプログラムで接続するには、エンドポイントを使用します。詳細については、「AWS 全般のリファレンス」の「[AWS のサービス エンドポイント](#)」を参照してください。

サービスレベルアグリーメント (SLA)

サービスのアップタイムやパフォーマンスなど、IT チームがお客様に提供すると約束したものを明示した合意書。

サービスレベルインジケータ (SLI)

エラー率、可用性、スループットといった、サービスパフォーマンス面の指標。

サービスレベル目標 (SLO)

[サービスレベルインジケータ](#)によって測定され、サービスの状態を表すターゲットメトリクス。

責任共有モデル

クラウドのセキュリティとコンプライアンス AWS について と共有する責任を説明するモデル。AWS はクラウドのセキュリティを担当しますが、 はクラウドのセキュリティを担当します。詳細については、「[責任共有モデル](#)」を参照してください。

SIEM

「[Security Information and Event Management システム](#)」を参照してください。

単一障害点 (SPOF)

特定のアプリケーションを構成する単一の重要なコンポーネントで発生し、システム稼働に支障をきたす可能性のある障害。

SLA

「[サービスレベルアグリーメント](#)」を参照してください。

SLI

「[サービスレベルインジケータ](#)」を参照してください。

SLO

「[サービスレベルの目標](#)」を参照してください。

スプリットアンドシードモデル

モダナイゼーションプロジェクトのスケーリングと加速のためのパターン。新機能と製品リリースが定義されると、コアチームは解放されて新しい製品チームを作成します。これにより、お客様の組織の能力とサービスの拡張、デベロッパーの生産性の向上、迅速なイノベーションのサポートに役立ちます。詳細については、「[AWS クラウドでのアプリケーションをモダナイズするための段階的アプローチ](#)」を参照してください。

SPOF

「[単一障害点](#)」を参照してください。

スタースキーマ

データベースの編成構造を意味し、1つの大きいファクトテーブルにトランザクションデータまたは測定データが保存され、1つ以上の小さいディメンションテーブルにデータ属性が保存されます。この構造は、[データウェアハウス](#)やビジネスインテリジェンスを用途とするように設計されています。

strangler fig パターン

レガシーシステムが廃止されるまで、システム機能を段階的に書き換えて置き換えることにより、モノリシックシステムをモダナイズするアプローチ。このパターンは、宿主の樹木から根を成長させ、最終的にその宿主を包み込み、宿主に取って代わるイチジクのつるを例えています。そのパターンは、モノリシックシステムを書き換えるときのリスクを管理する方法として [Martin Fowler](#) により提唱されました。このパターンの適用方法の例については、「[コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET \(ASMX\) ウェブサービスを段階的にモダナイズ](#)」を参照してください。

サブネット

VPC 内の IP アドレスの範囲。サブネットは、1つのアベイラビリティゾーンに存在する必要があります。

監視制御とデータ取得 (SCADA)

製造分野において、ハードウェアとソフトウェアを使用して物理アセットと本番運用をモニタリングするシステム。

対称暗号化

データの暗号化と復号に同じキーを使用する暗号化のアルゴリズム。

合成テスト

ユーザーとのやり取りをシミュレートして、起こり得る問題を検出したり、パフォーマンスをモニタリングしたりすることで、システムをテストします。[Amazon CloudWatch Synthetics](#) を使用すると、こうしたテストを作成できます。

システムプロンプト

コンテキスト、指示、ガイドラインなどを提示して、[LLM](#) に動作を指示する手法。システムプロンプトは、コンテキストを設定して、ユーザーとやり取りするルールを確立するのに有用です。

T

タグ

AWS リソースを整理するためのメタデータとして機能するキーと値のペア。タグは、リソースの管理、識別、整理、検索、フィルタリングに役立ちます。詳細については、「[AWS リソースのタグ付け](#)」を参照してください。

ターゲット変数

監督された機械学習でお客様が予測しようとしている値。これは、結果変数のことも指します。例えば、製造設定では、ターゲット変数が製品の欠陥である可能性があります。

タスクリスト

ランブックの進行状況を追跡するために使用されるツール。タスクリストには、ランブックの概要と完了する必要がある一般的なタスクのリストが含まれています。各一般的なタスクには、推定所要時間、所有者、進捗状況が含まれています。

テスト環境

「[環境](#)」を参照してください。

トレーニング

お客様の機械学習モデルに学習するデータを提供すること。トレーニングデータには正しい答えが含まれている必要があります。学習アルゴリズムは入力データ属性をターゲット (お客様が予測したい答え) にマッピングするトレーニングデータのパターンを検出します。これらのパターンをキャプチャする機械学習モデルを出力します。そして、お客様が機械学習モデルを使用して、ターゲットがわからない新しいデータでターゲットを予測できます。

トランジットゲートウェイ

VPC とオンプレミスネットワークを相互接続するために使用できる、ネットワークの中継ハブ。詳細については、AWS Transit Gateway ドキュメントの「[トランジットゲートウェイとは](#)」を参照してください。

トランクベースのワークフロー

デベロッパーが機能ブランチで機能をローカルにビルドしてテストし、その変更をメインブランチにマージするアプローチ。メインブランチはその後、開発環境、本番前環境、本番環境に合わせて順次構築されます。

信頼されたアクセス

ユーザーに代わって AWS Organizations およびそのアカウントで組織内でタスクを実行するために指定したサービスにアクセス許可を付与します。信頼されたサービスは、サービスにリンクされたロールを必要とときに各アカウントに作成し、ユーザーに代わって管理タスクを実行します。詳細については、ドキュメントの「[Using AWS Organizations with other AWS services](#) AWS Organizations」を参照してください。

チューニング

機械学習モデルの精度を向上させるために、お客様のトレーニングプロセスの側面を変更する。例えば、お客様が機械学習モデルをトレーニングするには、ラベル付けセットを生成し、ラベルを追加します。これらのステップを、異なる設定で複数回繰り返して、モデルを最適化します。

ツーピザチーム

2枚のピザを分け合えることができるくらい小さな DevOps チーム。ツーピザチームの規模では、ソフトウェア開発におけるコラボレーションに最適な機会が確保されます。

U

不確実性

予測機械学習モデルの信頼性を損なう可能性がある、不正確、不完全、または未知の情報を指す概念。不確実性には、次の2つのタイプがあります。認識論的不確実性は、限られた、不完全なデータによって引き起こされ、弁論的不確実性は、データに固有のノイズとランダム性によって引き起こされます。

未分化なタスク

ヘビーリフティングとも呼ばれ、アプリケーションの作成と運用には必要だが、エンドユーザーに直接的な価値をもたらさなかったり、競争上の優位性をもたらしたりしない作業です。未分化なタスクの例としては、調達、メンテナンス、キャパシティプランニングなどがあります。

上位環境

「[環境](#)」を参照してください。

V

バキューミング

ストレージを再利用してパフォーマンスを向上させるために、増分更新後にクリーンアップを行うデータベースのメンテナンス操作。

バージョンコントロール

リポジトリ内のソースコードへの変更など、変更を追跡するプロセスとツール。

VPC ピアリング

プライベート IP アドレスを使用してトラフィックをルーティングできる、2 つの VPC 間の接続。詳細については、Amazon VPC ドキュメントの「[VPC ピア機能とは](#)」を参照してください。

脆弱性

システムのセキュリティを脅かすソフトウェアまたはハードウェアの欠陥。

W

ウォームキャッシュ

頻繁にアクセスされる最新の関連データを含むバッファキャッシュ。データベースインスタンスはバッファキャッシュから、メインメモリまたはディスクからよりも短い時間で読み取りを行うことができます。

ウォームデータ

アクセス頻度の低いデータ。この種類のデータをクエリする場合、通常は適度に遅いクエリでも問題ありません。

ウィンドウ関数

現在のレコードに何らかの形で関連している行のグループに計算を実行する SQL 関数。ウィンドウ関数は、移動平均を計算したり、現在の行の相対位置に基づいて他の行の値にアクセスするといったタスクの処理に役立ちます。

ワークロード

ビジネス価値をもたらすリソースとコード (顧客向けアプリケーションやバックエンドプロセスなど) の総称。

ワークストリーム

特定のタスクセットを担当する移行プロジェクト内の機能グループ。各ワークストリームは独立していますが、プロジェクト内の他のワークストリームをサポートしています。たとえば、ポートフォリオワークストリームは、アプリケーションの優先順位付け、ウェーブ計画、および移行メタデータの収集を担当します。ポートフォリオワークストリームは、これらの設備を移行ワークストリームで実現し、サーバーとアプリケーションを移行します。

WORM

「[Write-Once-Read-Many](#)」を参照してください。

WQF

「[AWS ワークロード資格フレームワーク](#)」を参照してください

Write-Once-Read-Many (WORM)

データを 1 回のみ書き込むことで、データの削除や変更を防ぐストレージモデル。承認済みユーザーは、必要な回数だけデータを読み取ることができますが、変更することはできません。このデータストレージインフラストラクチャは、[イミュータブル](#)と見なされます。

Z

ゼロデイ 익스プロイト

[ゼロデイ脆弱性](#)を悪用した攻撃 (一般的にマルウェアによる)。

ゼロデイ脆弱性

実稼働システムにおける未解決の欠陥または脆弱性。脅威アクターは、このような脆弱性を利用してシステムを攻撃する可能性があります。開発者は、よく攻撃の結果で脆弱性に気付きます。

ゼロショットプロンプト

[LLM](#) にタスク実行の手順は提示するが、実行のガイドとして役立つ例 (ショット) は提示しない方法。LLM は、事前トレーニング済みの知識を使用してタスクを処理する必要があります。ゼロショットプロンプトの有効性は、タスクの複雑さとプロンプトの品質によって異なります。「[数ショットプロンプト](#)」も参照してください。

ゾンビアプリケーション

平均 CPU およびメモリ使用率が 5% 未満のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するのが一般的です。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。