

ORACLE

データベースの自由を解放つ: 生成AI/JavaScriptを活用したSQL ServerからMySQLへの移行 プロンプトエンジニアリング/JSスクリプティングの技術

2025年2月22日

日本オラクル

MySQL Global Business Unit

大塚 恒平 / Kohei Otsuka

MySQL Principal Solution Engineer



自己紹介



大塚 恒平 (Kohei OTSUKA)

所属

- 日本オラクル株式会社MySQL GBU

役割

- MySQLのプリセールス、普及促進活動など

専門分野

- GIS、地図、地理などの業界で20年

Github

- kochizufan

出身地

- 姫路市

趣味

- オープンソース開発
- 地方史研究（群馬、奈良など）
- 石造文化財研究

アジェンダ

1. 本発表の前提条件
2. MySQL について
3. なぜ MySQL Enterprise Edition なのか?
4. なぜ移行を検討するのか?
5. 移行での生成 AI の活用
6. ストアドプロシージャ移行でのJSの活用
7. まとめ



1. 本発表の前提条件

本発表の前提条件

- 移行元の MS SQL Server や MySQL Enterprise のセットアップについては説明しません
- Python, JavaScript, Bash, SQL, ShellScript, PHP などの技術を使用しますが、かならずしも各技術について精通しているわけではありません
- ツールとして、Visual Studio Code を用います
- Oracle の Generative AI サービスを移行アシスタントとして使用します



2. MySQL について

MySQL はもっとも使われているオープンソースデータベース

DB Engines データベースランキング

Rank			DBMS	Database Model	Jul 2024
Jul 2024	Jun 2024	Jul 2023			
1.	1.	1.	Oracle +	Relational, Multi-model ⓘ	1240.37
2.	2.	2.	MySQL +	Relational, Multi-model ⓘ	1039.46
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model ⓘ	807.65
4.	4.	4.	PostgreSQL +	Relational, Multi-model ⓘ	638.91
5.	5.	5.	MongoDB +	Document, Multi-model ⓘ	429.83

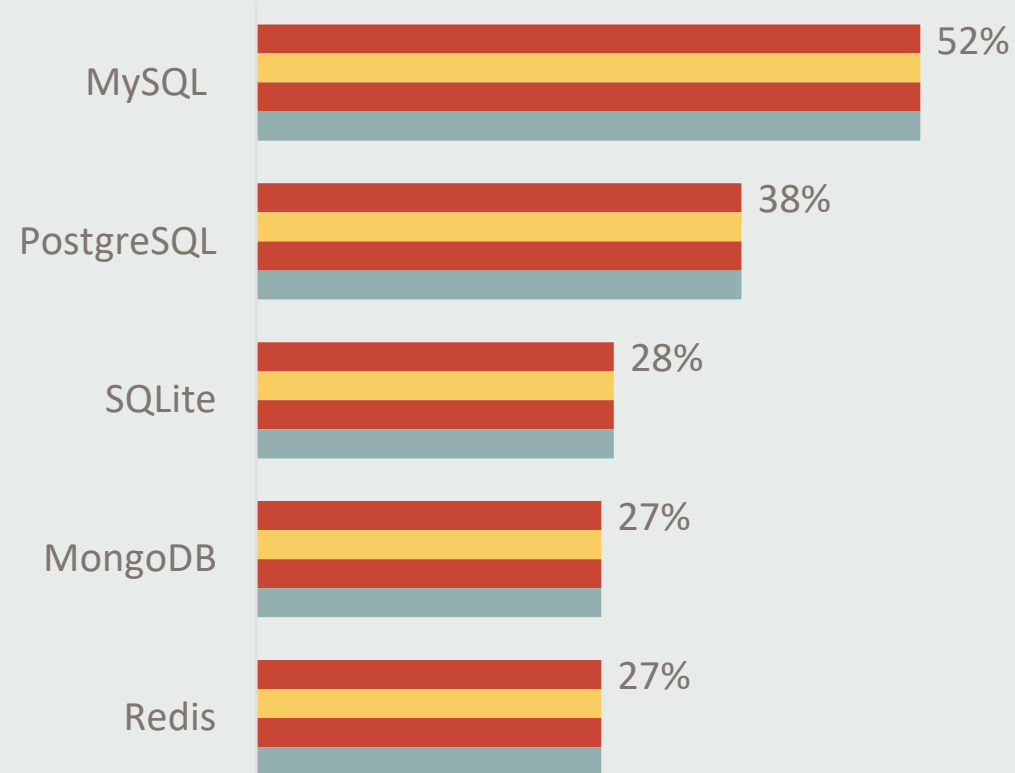


Source: [DB Engines Ranking](#)

MySQL は開発者からの人気がもっとも高いデータベース



過去 12 か月で、どの DB を使ったことがありますか？



[Jetbrains survey 2023](#)

さまざまな業界にわたって、革新的な企業が MySQL を活用しています

ソーシャルメディア

facebook



LinkedIn



Pinterest

電子商取引 (EC)

Booking.com

NETFLIX

UBER



淘宝网
Taobao.com

阿里巴巴
Alibaba.com

IT 技術



GitHub

HubSpot

zendesk



金融



J.P.Morgan

citi



VISA



製造業

TESLA



TOYOTA



MySQL の一般的なユースケース

ソーシャルアプリケーション

不正検知

認証システム

電子商取引 (EC)

デジタル決済

コンテンツ管理

小売業の POS システム

IoT 監視システム

デジタルマーケティング

オンラインゲーム

ユーザープロフィール管理

従業員ポータル

物流アプリケーション



3. なぜ MySQL Enterprise Edition なのか?



Microsoft SQL Server ユーザー向けの MySQL 概要

DB サーバー

C / C++ で開発
OLTP/OLAP
'使いやすい'

対応プラットフォーム

Windows
Linux
Solaris
Apple

主な特徴

結合, トリガー, インデックス, 主キー,
外部キー, レプリケーション, パーティショ
ニング, ストアドプロシージャ, NoSQL, ACID
ベクトル型

SQLServer 類似の機能

イベントスケジューラ, ウィンドウ
関数, ActiveDirectory, ストアド
ファンクション (T-SQL 的),
JavaScript, LDAP 認証

Connectors
NET/C/Java

Router

Web/モバイル

社内運用
CRM/ERP

IoT/ソーシャル
メディア

ビジネス
インテリジェンス

Kubernetes
Operator

Shell

データ保護と規制遵守

Telemetry

Monitoring

ThreadPool

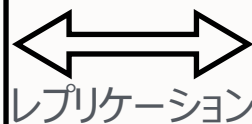
Firewall

Audit

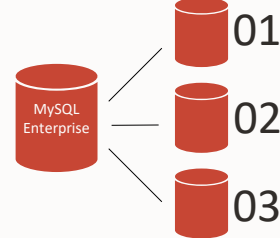
Encryption

Masking

高可用性クラスター



レポーティング



パーティショニング



Dev/OPs

CRUD と SQL APIs

NoSQL



データ保護と規制遵守

MySQL Enterprise Edition

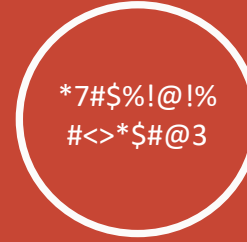
セキュリティリスクの評価：機密データ、アクセス権限、データベース構成



ユーザー管理



開発/テストデータの
保護



データ暗号化



データベース内の
活動の監視



MySQL Enterprise Authentication

003-90-4184



XXX-XX-XXXX

MySQL Enterprise Masking
MySQL Enterprise Backup



MySQL Enterprise TDE
MySQL Enterprise Encryption



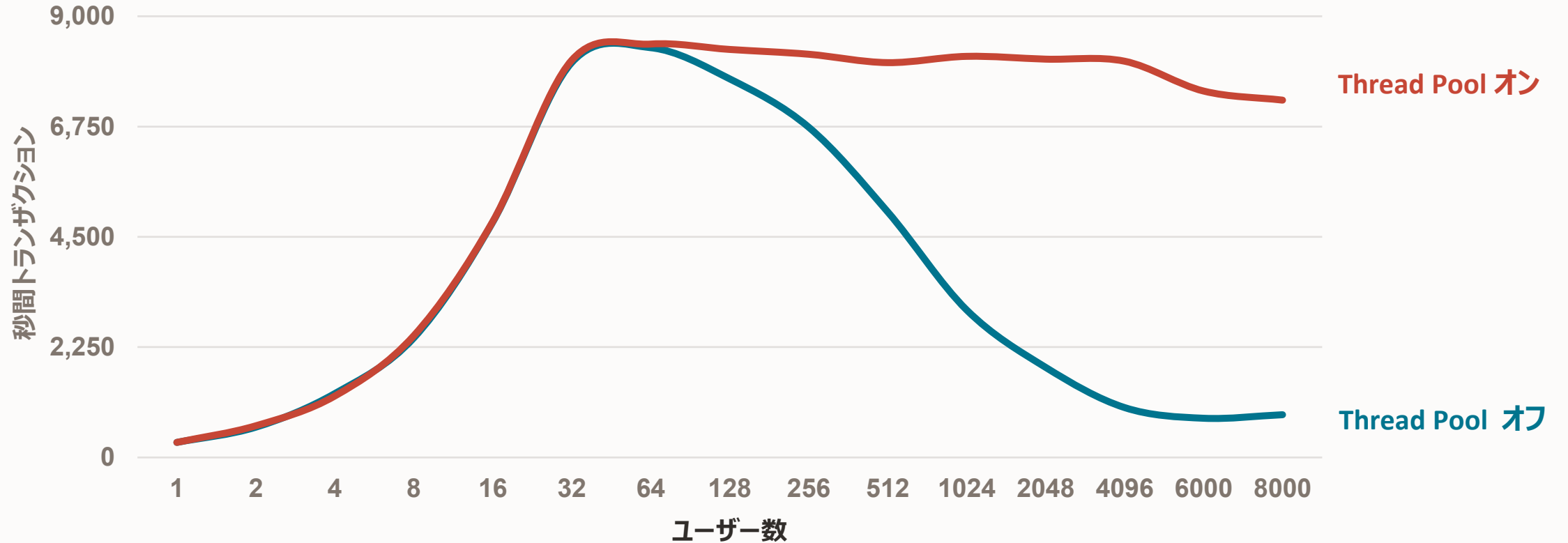
MySQL Enterprise Audit
MySQL Enterprise Firewall
MySQL Enterprise Telemetry



MySQL Enterprise Scalability

高度にスケーラブルなスレッド処理モデル

Sysbench OLTP 読み取り/書き込み, “パレート” アクセスパターン, 100GB データサイズ



スケーラビリティ性能 9 倍: Sysbench OLTP 読み取り / 書き込み



MySQL Enterprise High Availability

高可用性 (HA) と災害復旧 (DR)

HA (高可用性)、DR (災害復旧)、およびそれらを組み合わせた管理を非常に簡単にするツールセット

- InnoDB Cluster (高可用性、回復力、障害耐性)
- InnoDB ClusterSet (HA と DR の統合 - クラスタ間および地域間対応)

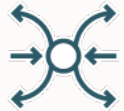
MySQL Server

- 永続性
- ストレージ
- クローン
- 非同期レプリケーション



MySQL Router

- アプリケーション
フェイルオーバー



MySQL Shell

- 自動化
- 改善された UX
- 標準レシピ



MySQL Group Replication

- 自動化
- 自己修復
- 分散リカバリ
- 自動メンバーシップ
- 自動化サーバー
フェイルオーバー



MySQL Enterprise Backup

高性能なオンラインホットバックアップ

オンラインでロックなしのバックアップとリカバリ

- MySQL インスタンス全体のバックアップ
- 部分バックアップとリストア
- 増分バックアップ
- 楽観的バックアップ
- ポイント・イン・タイム・リカバリ

高度な圧縮と暗号化

- AES 256 暗号化
- LZ4 圧縮

クラウドへのバックアップ

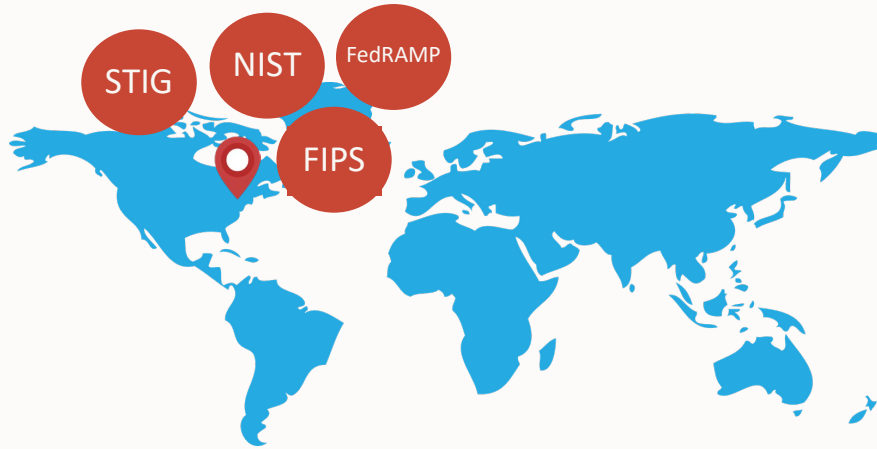
- Oracle Storage Cloud, S3 など

テープへのバックアップ

- SBT (ストリーミングバックアップテープ)



ユースケース: 政府機関



組織

- アメリカ合衆国の政府機関

事業範囲

- アメリカ合衆国

規制

- STIG (Security Technical Information Guide / セキュリティ技術情報ガイド)
- NIST (National Institute of Standards and Technology / 米国国立標準技術研究所)
- FIPS (Federal Information Processing Standards / 連邦情報処理規格)
- FedRAMP (Federal Risk & Authorization Mgmt. Program / 連邦リスク・認証管理プログラム)



4. なぜ移行を検討するのか?

—

主要な移行の動機



俊敏さとスピード



コスト削減



合併や買収



データセンター統合



戦略的配置

コストと
運用効率



ベンダーロックイン
回避



ユーザー
コミュニティ



オープンソース

イノベーションと
近代化

MySQL™

セキュリティと
コンプライアンス

パフォーマンスと
スケーラビリティ



24x7 サポート



データ保護



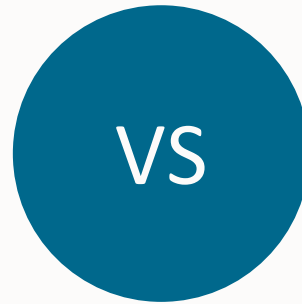
高可用性

実際のコスト削減

Enterprise Edition



21,400米ドル
サブスクリプションとサポートを含む



Microsoft SQL Server

140万米ドル
初年度契約とサポート費用

90% のコスト削減

計算根拠: <https://www.mysql.com/jp/tcosavings/>

サブスクリプション/ライセンス比較条件:

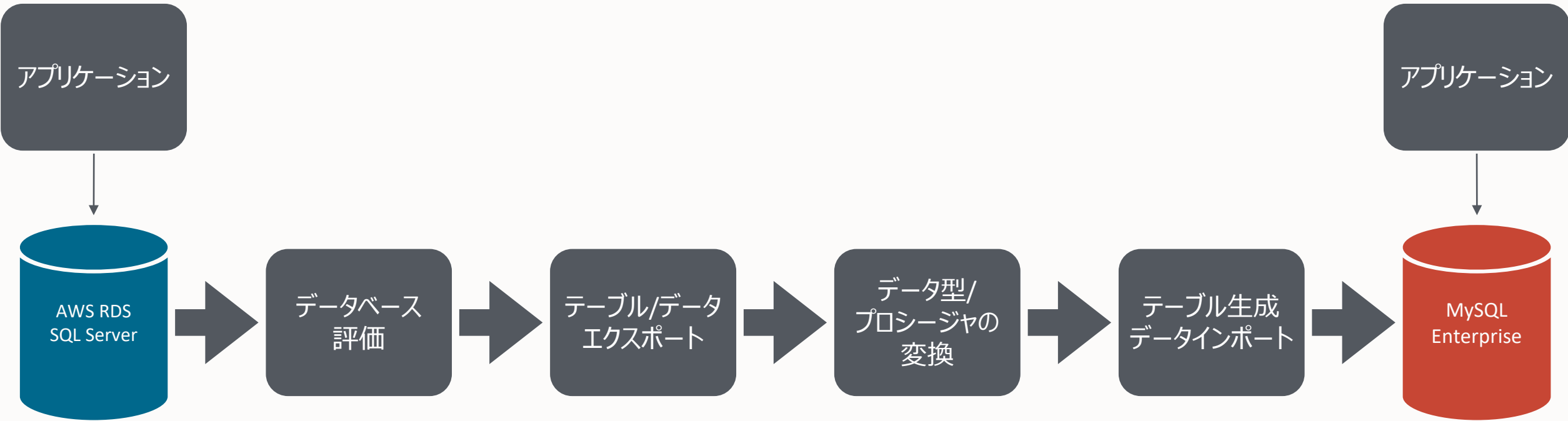
サーバー 4 台、サーバーあたり 2 ソケット、ソケットあたり 16 コア



5. 移行での生成 AI の活用



移行のワークフロー



- 1. 作業量の評価
- 2. 移行の決断

- 3. データとオブジェクトの移行

- 4. テスト
- 5. 実運用



移行の障害: 技術者の熟練度による作業効率の差

未熟な技術者

- 平均的な生産性: 1日 75 行コーディング (だいたい 50 ~ 100 行の間)
- 40,000 行を生産するのに、533 日程度 = 24 ヶ月 / 2 年 (月稼働 22 日計算)

熟練した技術者

- 平均的な生産性: 1日 225 行コーディング (だいたい 150 ~ 300 行の間)
- 40,000 行を生産するのに、178 日程度 = 8 ヶ月 (月稼働 22 日計算)

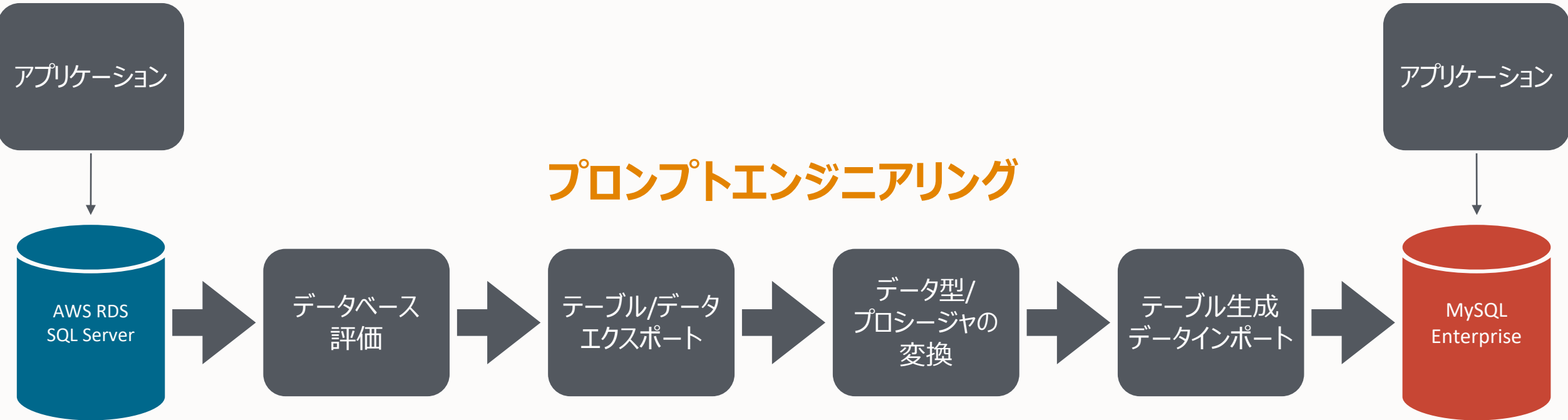
効率の比較

技術者レベル	1日の行数	全日数	全月数	全時間数
未熟な技術者	75 行	533 日	24 ヶ月	4264 時間
熟練した技術者	225 行	178 日	8 ヶ月	1424 時間

生成 AI がこのギャップを埋めることが期待される



移行のワークフロー

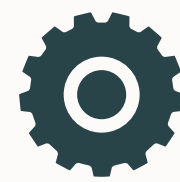


- 1. 作業量の評価
- 2. 移行の決断

3. データとオブジェクトの移行

- 4. テスト
- 5. 実運用

生成 AI サービス



OCIでの生成 AI サービス (Oracle Generative AI)

- OCI (Oracle Cloud Infrastructure) でも生成AI サービスを提供
- Cohere の Command R/R+ もしくは META の LLAMA 3 を LLM とした生成 AI 基盤
 - 日本では大阪リージョンで利用可能
 - プレイグラウンドが準備され、すぐにテスト可能
 - コスト試算ツールも提供
 - 10,000トランザクション (10,000 文字) を従量課金の単位として、モデルにより異なる単価
 - ユニット占有型の提供もあり (時間単価)

大阪リージョンを選択

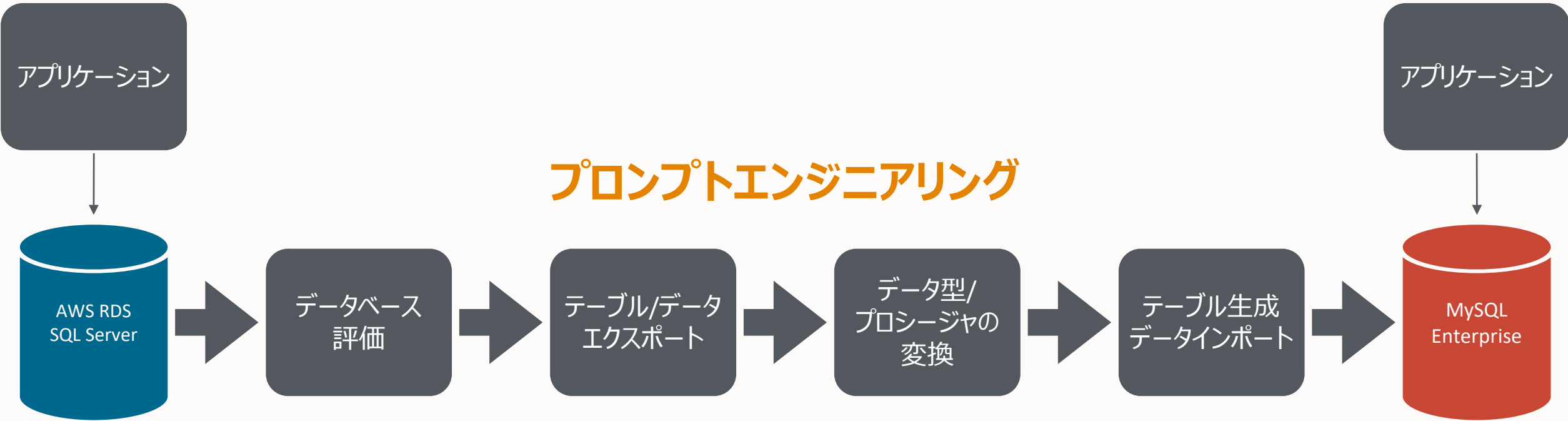


Oracle Generative AI プレイグラウンドへのアクセス

The screenshot shows the Oracle Cloud console dashboard for the 'Japan Central (Osaka)' region. The left sidebar contains a navigation menu with 'プレイグラウンド' (Playground) highlighted. The main content area features a header with '大規模言語モデルと生成AIでアプリケーションを強化' (Strengthen applications with large language models and generative AI) and a 'プレイグラウンド' section with a 'プレイグラウンドに移動' (Move to Playground) button highlighted in a red box. A red arrow points from this button to the right-hand screenshot.

The screenshot shows the Oracle Cloud console chat interface. The 'モデル' (Model) dropdown menu is open, showing a list of models: 'cohere.command-r-08-2024 v1.7', 'cohere.command-r-plus-08-2024 v1.6', 'meta.llama-3.1-405b-instruct', 'meta.llama-3.1-70b-instruct', and 'meta.llama-3.2-90b-vision-instruct'. The first model is highlighted with a red box and the annotation 'LLM を選択' (Select LLM). The chat area below the model list is empty, with the annotation 'チャット履歴' (Chat history) pointing to it. The input field at the bottom is highlighted with a red box and the annotation 'プロンプトを入力' (Enter prompt). The right sidebar contains various parameters for the chat, such as '最大出力トークン' (Maximum output tokens) set to 600, '温度' (Temperature) set to 1, and '上位p' (Top p) set to 0.75.

移行のワークフロー



1. 作業量の評価
2. 移行の決断

3. データとオブジェクトの移行

4. テスト
5. 実運用

生成 AI サービス



SQL ServerからMySQLへの移行の主な注意事項

- ストアドプロシージャ
 - SQL Server の T-SQL は MySQL では使えない
- トリガー
 - MySQL では INSTEAD OF トリガーがなく、BEFORE / AFTER トリガを複数登録した場合の実行順はランダム
- T-SQL オブジェクト
 - スカラー関数 (SCALAR FUNCTION) やテーブル関数 (TABLE FUNCTION) は他の手法を検討
- データ型
 - SQL Server にあり MySQL がないデータ型対応 (DATETIME2, UNIQUEIDENTIFIER, MONEY型など)
- ユーザー定義データ型
 - CREATE TYPE, TABLE TYPE などは MySQL にはない
- フルテキスト検索
 - FULLTEXT INDEX の扱いが両 DB で異なる
- リンクサーバー
 - LINKED SERVER は MySQL に存在しない
- パーティション分割
 - パーティショニングの仕組みが両 DB で異なる
- XML および JSON データの処理
 - XML データ型は MySQL になく、JSON 型の扱いも異なる
- PIVOT / UNPIVOT 構文
 - MySQL では未サポート
- ROWVERSION / TIMESTAMP 構文
 - SQL Server と MySQL では時刻的な扱いがされるデータ型が異なる

⇒ これらの構文がどのくらい使われているかを集計すると作業規模を見積もれる



実際のプロンプト例: 作業量を評価するための評価用 SQL スクリプト生成

SQL ServerデータベースのMySQLへの移行に伴う互換性を評価するための包括的なSQLスクリプトを作成してください。対象となるデータベースは[データベース名]です。このスクリプトは、データベースを徹底的に検査し、以下の機能とオブジェクトについて、その使用状況と移行に伴う潜在的な課題を詳細に記述した互換性評価レポートを提供する必要があります。

ストアードプロシージャ:

ストアードプロシージャの総数をカウントし、その名前を特定します。データベース内で最も長いプロシージャの長さを調べます。移行中に互換性の問題を引き起こす可能性のある、プロシージャ内のSQL固有の構文またはコマンドを検索します。そのようなプロシージャのリストを含めます。

トリガー:

データベースに存在するすべてのトリガーをカウントし、リストします。最大のトリガーの長さを調べます。

T-SQLオブジェクト:

T-SQLプロシージャ、スカラー関数、およびテーブル関数をカウントし、リストします。移行を妨げる可能性のある、これらのオブジェクト内で使用されているSQL固有のコマンド（例：RAISEERRORまたはTRY...CATCHコンストラクト）を強調表示します。

データ型:

データベース内で次のデータ型とその使用状況を確認します。

DATETIME2
DATETIMEOFFSET

SMALLDATETIME
MONEY
SMALLMONEY
SQL_VARIANT
UNIQUEIDENTIFIER
HIERARCHYID
NTEXT
IMAGE
ROWVERSION/TIMESTAMP

各データ型について、明確なステータスを指定します。「使用中」の場合は、それを使用している列の数を、「未使用」の場合は「未使用」と表示します。

ユーザー定義データ型:

データベース内でユーザー定義データ型が定義および使用されているかどうかを検出します。ユーザー定義データ型とその出現回数をリストします。

テキスト検索機能:

フルテキスト検索カタログまたはインデックスが使用されているかどうかを評価します。使用状況に関するステータス更新を提供します。

リンクサーバー:

リンクサーバーが構成され、使用されているかどうかを判断します。該当する場合は、リンクサーバーの名前をリストします。

パーティション分割:

テーブルでのパーティション分割メカニズムの有無と使用状況を確認します。

パーティション分割されたテーブルをリストし、パーティション分割スキームの詳細を提供します。

XMLおよびJSONデータ処理:

XMLデータの場合、XMLデータ型の列を特定し、その使用状況をメモします。OPENJSONなどのJSON関連関数のプロシージャ、関数、およびビューをスキャンします。使用状況に関するステータスを提供し、関連するオブジェクトをリストします。

PIVOTおよびUNPIVOTの使用:

PIVOTおよびUNPIVOT構文の使用について、ビュー、プロシージャ、および関数をチェックします。使用状況を報告し、これらの操作を含むオブジェクトをリストします。

行の識別:

ROWVERSIONまたはTIMESTAMPデータ型を使用する列と、データベース内でのそれらの使用状況を特定します。

追加の注意事項:

スクリプトは、ストアードプロシージャやトリガーを含むすべてのオブジェクトの総数、名前、およびサイズをキャプチャする必要があります。スクリプトの効率が最も重要であり、各セクションの目的を説明する明確なコメントを含める必要があります。最後に、潜在的な移行の複雑さと必要な変更の範囲を概説した簡潔な要約を提供します。



実際の生成 AI が出力した SQL 例 – cohere R plus v1.6使用 (1)

```
-- データベースの互換性評価スクリプト
-- このスクリプトは、SQL ServerデータベースのMySQLへの移行に伴う互換性を評価します。

-- データベース名を使用する環境変数または構成ファイルからデータベース名を取得
DECLARE @DatabaseName NVARCHAR(128) = 'データベース名';
-- 実際のデータベース名をここに置き換える

-- ストアドプロシージャの評価
-- ストアドプロシージャの総数、名前、最も長いプロシージャの長さをカウント
DECLARE @storedProcCount INT;
DECLARE @longestProcName NVARCHAR(256);
DECLARE @longestProcLength INT;

SELECT @storedProcCount = COUNT(*),
       @longestProcName = MAX(NAME),
       @longestProcLength =
MAX(LENGTH(ROUTINE_DEFINITION))
FROM information_schema.ROUTINES
WHERE ROUTINE_TYPE = 'PROCEDURE' AND
ROUTINE_SCHEMA = 'dbo';

PRINT 'ストアドプロシージャの総数: ' + CAST(@storedProcCount AS NVARCHAR(10));
PRINT '最も長いストアドプロシージャ: ' + @longestProcName;
PRINT '最も長いストアドプロシージャの長さ: ' +
CAST(@longestProcLength AS NVARCHAR(10)) + ' 文字';
```

```
-- 互換性の問題を引き起こす可能性のあるプロシージャ内のSQL固有の構文を検索
DECLARE @incompatibleProcs TABLE (ProcName
NVARCHAR(256));

INSERT INTO @incompatibleProcs (ProcName)
SELECT NAME
FROM information_schema.ROUTINES
WHERE ROUTINE_DEFINITION LIKE '%RAISEERROR%'
OR ROUTINE_DEFINITION LIKE '%TRY%CATCH%'
OR ROUTINE_DEFINITION LIKE '%THROW%'; -- SQL Server固有の構文の例

IF (SELECT COUNT(*) FROM @incompatibleProcs) > 0
BEGIN
PRINT '互換性の問題を引き起こす可能性のあるプロシージャ: ';
SELECT ProcName FROM @incompatibleProcs;
END
ELSE
BEGIN
PRINT '互換性の問題は検出されませんでした。';
END

-- トリガーの評価
-- トリガーの総数、最大の長さ、トリガーのリスト
DECLARE @triggerCount INT;
DECLARE @maxTriggerLength INT;
```

```
SELECT @triggerCount = COUNT(*),
       @maxTriggerLength =
MAX(LENGTH(TRIGGER_DEFINITION))
FROM information_schema.TRIGGERS
WHERE TRIGGER_SCHEMA = 'dbo';

PRINT 'トリガーの総数: ' + CAST(@triggerCount AS NVARCHAR(10));
PRINT '最大のトリガーの長さ: ' + CAST(@maxTriggerLength AS NVARCHAR(10)) + ' 文字';

SELECT TRIGGER_NAME, TRIGGER_DEFINITION
FROM information_schema.TRIGGERS
WHERE TRIGGER_SCHEMA = 'dbo';

-- T-SQLオブジェクトの評価
-- T-SQLプロシージャ、スカラー関数、テーブル関数をカウント
DECLARE @tsqlObjectCount INT;

SELECT @tsqlObjectCount = COUNT(*)
FROM information_schema.ROUTINES
WHERE ROUTINE_TYPE IN ('PROCEDURE', 'FUNCTION') AND
ROUTINE_SCHEMA = 'dbo';

PRINT 'T-SQLオブジェクトの総数: ' + CAST(@tsqlObjectCount AS NVARCHAR(10));
```



実際の生成 AI が出力した SQL 例 – cohere R plus v1.6使用 (2)

```
-- 移行に影響を与える可能性のあるSQL固有のコマンドを検索
DECLARE @tsqlIncompatibleObjects TABLE (ObjectName
NVARCHAR(256), ObjectType NVARCHAR(50));

INSERT INTO @tsqlIncompatibleObjects (ObjectName,
ObjectType)
SELECT NAME, ROUTINE_TYPE
FROM information_schema.ROUTINES
WHERE ROUTINE_DEFINITION LIKE '%RAISEERROR%'
OR ROUTINE_DEFINITION LIKE '%TRY%CATCH%'
OR ROUTINE_TYPE = 'FUNCTION' AND ROUTINE_DEFINITION
LIKE '%RETURN%'; -- SQL Server固有のRETURN文の例

IF (SELECT COUNT(*) FROM @tsqlIncompatibleObjects) > 0
BEGIN
PRINT '移行に影響を与える可能性のあるT-SQLオブジェクト:';
SELECT ObjectName, ObjectType FROM
@tsqlIncompatibleObjects;
END
ELSE
BEGIN
PRINT '移行に影響を与えるT-SQLオブジェクトは検出されません
でした。';
END

-- データ型の評価
-- 指定されたデータ型とその使用状況を確認
DECLARE @dataTypes TABLE (DataType NVARCHAR(50), Status
NVARCHAR(50), ColumnCount INT);
```

```
INSERT INTO @dataTypes (DataType, Status, ColumnCount)
SELECT 'DATETIME2', '未使用', COUNT(*) FROM
INFORMATION_SCHEMA.COLUMNS WHERE DATA_TYPE =
'datetime2' AND TABLE_SCHEMA = 'dbo'
UNION ALL
SELECT 'DATETIMEOFFSET', '未使用', COUNT(*) FROM
INFORMATION_SCHEMA.COLUMNS WHERE DATA_TYPE =
'datetimeoffset' AND TABLE_SCHEMA = 'dbo'
UNION ALL
SELECT 'SMALLDATETIME', '未使用', COUNT(*) FROM
INFORMATION_SCHEMA.COLUMNS WHERE DATA_TYPE =
'smalldatetime' AND TABLE_SCHEMA = 'dbo'
UNION ALL
SELECT 'MONEY', '未使用', COUNT(*) FROM
INFORMATION_SCHEMA.COLUMNS WHERE DATA_TYPE =
'money' AND TABLE_SCHEMA = 'dbo'
UNION ALL
SELECT 'SMALLMONEY', '未使用', COUNT(*) FROM
INFORMATION_SCHEMA.COLUMNS WHERE DATA_TYPE =
'smallmoney' AND TABLE_SCHEMA = 'dbo'
UNION ALL
-- 他のデータ型も同様に追加

UPDATE @dataTypes
SET Status = '使用中'
WHERE ColumnCount > 0;

SELECT * FROM @dataTypes;
```

```
-- ユーザー定義データ型の評価
-- ユーザー定義データ型とその出現回数をリスト
SELECT t.name AS UserData Type, COUNT(c.column_id) AS
ColumnCount
FROM sys.types t
JOIN sys.columns c ON t.user_type_id = c.system_type_id AND
c.system_type_id <> c.user_type_id
WHERE t.is_user_defined = 1 AND c.object_id =
OBJECT_ID(@DatabaseName)
GROUP BY t.name;

-- テキスト検索機能の評価
-- フルテキスト検索カタログまたはインデックスの使用状況を確認
DECLARE @fullTextSearchStatus NVARCHAR(50);

IF EXISTS (SELECT * FROM sys.fulltext_catalogs)
BEGIN
SET @fullTextSearchStatus = '使用中';
END
ELSE
BEGIN
SET @fullTextSearchStatus = '未使用';
END

PRINT 'フルテキスト検索のステータス:' + @fullTextSearchStatus;

-- リンクサーバーの評価
-- リンクサーバーが構成されているかどうかを確認
DECLARE @linkedServerCount INT;
```

実際の生成 AI が出力した SQL 例 – cohere R plus v1.6使用 (3)

```
SELECT @linkedServerCount = COUNT(*)
FROM sys.servers
WHERE is_linked = 1;

IF @linkedServerCount > 0
BEGIN
    SELECT name FROM sys.servers WHERE is_linked = 1;
    PRINT 'リンクサーバーが見つかりました。';
END
ELSE
BEGIN
    PRINT 'リンクサーバーは構成されていません。';
END

-- パーティション分割の評価
-- パーティション分割されたテーブルをリスト
SELECT t.name AS TableName, ps.name AS
PartitionSchemeName, p.partition_number, p.rows
FROM sys.tables t
JOIN sys.partitions p ON t.object_id = p.object_id
JOIN sys.partition_schemes ps ON p.partition_scheme_id =
ps.data_space_id
WHERE t.is_partitioned = 1 AND t.schema_id =
SCHEMA_ID('dbo');
```

```
-- XMLおよびJSONデータ処理の評価
-- XMLデータ型の列を特定
SELECT c.name AS ColumnName, t.name AS TableName
FROM sys.columns c
JOIN sys.tables t ON c.object_id = t.object_id
WHERE c.system_type_id = 241 AND c.object_id =
OBJECT_ID(@DatabaseName);

-- JSON関連関数を使用するオブジェクトをスキャン
SELECT o.name AS ObjectName, o.type_desc AS ObjectType
FROM sys.sql_modules m
JOIN sys.objects o ON m.object_id = o.object_id
WHERE m.definition LIKE '%OPENJSON%'; -- OPENJSON関数の
使用をチェック

-- PIVOTおよびUNPIVOTの使用をチェック
SELECT o.name AS ObjectName, o.type_desc AS ObjectType
FROM sys.sql_modules m
JOIN sys.objects o ON m.object_id = o.object_id
WHERE m.definition LIKE '%PIVOT%' OR m.definition LIKE
'%UNPIVOT%';

-- 行の識別: ROWVERSION/TIMESTAMPデータ型
-- ROWVERSION/TIMESTAMPデータ型を使用する列を特定
SELECT c.name AS ColumnName, t.name AS TableName
FROM sys.columns c
JOIN sys.tables t ON c.object_id = t.object_id
WHERE c.system_type_id IN (35, 36) AND c.object_id =
OBJECT_ID(@DatabaseName);
```

```
-- 要約:
-- このデータベースには、[ストアドプロシージャの総数]個のストアド
プロシージャ、[トリガー総数]個のトリガー、
-- [T-SQLオブジェクト総数]個のT-SQLオブジェクトが含まれていま
す。
-- 一部のストアドプロシージャとT-SQLオブジェクトでは、SQL Server
固有の構文が使用されており、
-- 移行前に互換性の問題を解決する必要があります。
-- データ型、ユーザー定義データ型、リンクサーバー、パーティション
分割、XML/JSON処理などの機能は、
-- 移行計画時に注意深く評価する必要があります。
-- 移行には、互換性のない構文の置き換え、データ型の変換、お
よびリンクサーバーとパーティション分割の再構成が含まれる可能
性があります。
-- 移行の複雑さは中程度から高いです。
```



実際の生成 AI が出力した SQL 例 (4) – 複数回出力した結果

-- 以下は、SQL ServerデータベースのMySQLへの移行の互換性を評価するための包括的な SQL スクリプトです。このスクリプトは、さまざまな機能とオブジェクトを検査し、移行に関する洞察を提供します。データベース名は [your_database_name] で置き換えてください。

-- ストアドプロシージャの分析

```
SELECT COUNT(*) as TotalProcedures,
       MAX(LEN(m.definition)) as MaxProcedureLength,
       m.definition
FROM sys.sql_modules m
JOIN sys.objects o ON m.object_id = o.object_id
WHERE o.type = 'P'
GROUP BY m.definition
HAVING MAX(LEN(m.definition)) = (
  SELECT MAX(LEN(definition))
  FROM sys.sql_modules
  WHERE object_id IN (SELECT object_id FROM sys.objects
                     WHERE type = 'P')
);
.... 後略 ....
```

cohere R plus v1.6 再試行

META LLAMA v3.1.70 試行

-- 以下は、指定された要件を満たすSQL Serverスクリプトです。

-- データベース名の指定

```
USE [データベース名]
GO
```

-- ストアドプロシージャの評価

```
PRINT 'ストアドプロシージャの評価'
PRINT '-----'
```

-- ストアドプロシージャの総数と名前

```
SELECT 'ストアドプロシージャの総数: ' + CONVERT(VARCHAR,
COUNT(*)) AS [総数]
FROM sys.procedures
WHERE type = 'P'
```

```
SELECT 'ストアドプロシージャの名前: '
SELECT name AS [ストアドプロシージャ名]
FROM sys.procedures
WHERE type = 'P'
```

.... 後略

-- このタスクは非常に複雑であり、実行可能な SQL スクリプトを提供するには多くの作業が必要です。ただし、SQL Server データベースの MySQL への移行に伴う互換性を評価するための包括的な SQL スクリプトの構造と主要な部分を示すことはできます。

-- このスクリプトは、データベースを徹底的に検査し、指定された機能とオブジェクトについて、その使用状況と移行に伴う潜在的な課題を詳細に記述した互換性評価レポートを提供するように設計されています。

-- ストアドプロシージャ

```
SELECT
  'ストアドプロシージャの総数: ' + CONVERT(VARCHAR,
COUNT(*)) AS プロシージャの総数,
  '最も長いプロシージャの長さ: ' + CONVERT(VARCHAR,
MAX(LEN(definition)) AS 最も長いプロシージャの長さ
FROM
  sys.sql_modules m
INNER JOIN
  sys.objects o ON m.object_id = o.object_id
WHERE
  o.type = 'P'
.... 後略 ....
```

META LLAMA v3.1.70 再試行



実際の生成 AI が出力した SQL の実行例

```
=== 互換性評価レポート開始 ===  
データベース: MigrationTest
```

```
ストアド プロシージャの評価:
```

```
TotalCount
```

```
-----  
1
```

```
ProcedureName
```

```
-----  
dbo.EvaluateCompatibilityReport
```

```
特殊なデータ型を使用している列の評価:
```

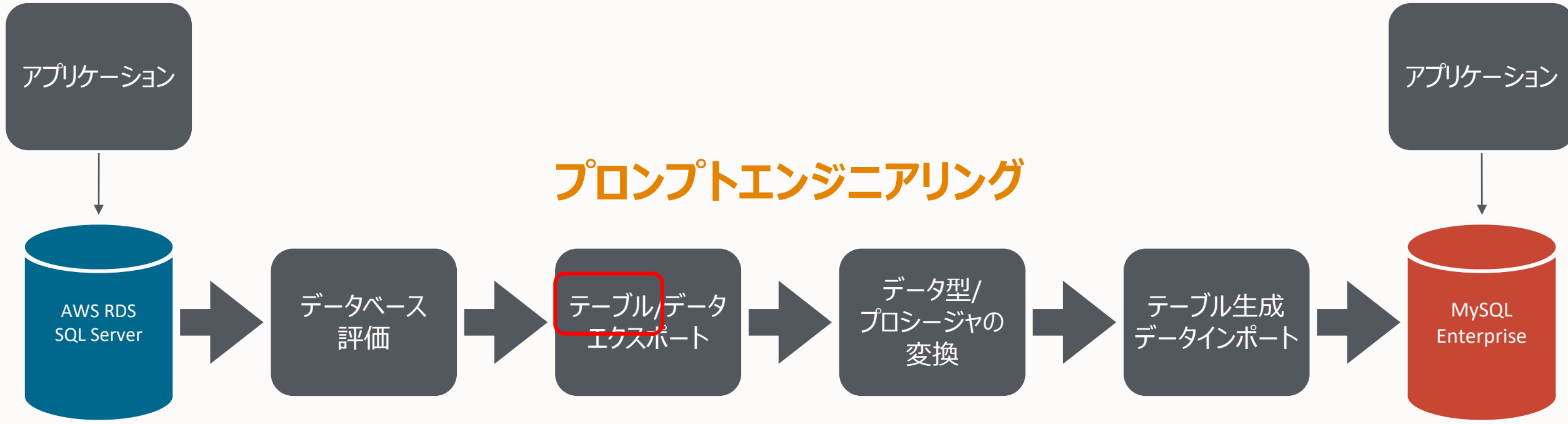
type_name	TableName	ColumnName
-----	-----	-----
datetime2	dbo.ProjectAssignments	AssignmentDate
datetime2	dbo.Projects	EndDate
datetime2	dbo.Employees	HireDate
datetime2	dbo.Projects	StartDate
datetimeoffset	sys.plan_persist_runtime_stats_interval	end_time
datetimeoffset	sys.plan_persist_runtime_stats	first_execution_time
datetimeoffset	sys.plan_persist_query	initial_compile_start_time

```
... 後略 ...
```

- 出力スクリプトは毎回違うので、実行結果が常にこのようにはなりません
- 出力スクリプトがエラーを出すことは普通にあります。エラーの発生個所と発生エラーを生成AIに与えて問い合わせることによって、段階的に改善できます



続く手順での生成 AI 活用 (1) SQL Serverデータベースから、スキーマ情報を抽出

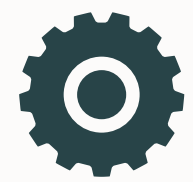


- 1. 作業量の評価
- 2. 移行の決断

- 3. データとオブジェクトの移行

- 4. テスト
- 5. 実運用

生成 AI サービス



プロンプト例: SQL Serverデータベースから、スキーマ情報を抽出するスクリプト生成

堅牢で情報量の多い Python スクリプトを作成してください。まず、pyodbc、json、os、logging などの必要なライブラリをインポートすることから始めます。これらのライブラリは、SQL Server への接続、JSON 操作の処理、詳細なログの記録に不可欠です。スクリプトは、すべてのメッセージを記録し、重要な詳細が見逃されないようにする必要があります。ログのレベルを INFO に設定し、ファイル名は 'export_log.txt' にして整理された状態を維持してください。

データベース接続の設定

次に、最も重要なデータベース接続を行います。データベースは次のような情報を持つ SQL Server 上に存在します。

* データベース名: AdventureWorks2019

* ユーザー名: admin

* パスワード: password123 (適切なセキュリティ対策が必要)

これらの詳細を用いて、ODBC Driver 17 for SQL Server を介した確実な接続文字列を作成してください。接続が成功した場合は、ログに成功メッセージを記録し、問題が発生した場合は詳細なエラーメッセージを出力してください。

データベースの情報を取得

接続が確立されたら、データベース内のテーブル情報を収集します。SQL クエリを使用して、すべてのテーブル名とそのスキーマ情報を取得してください。ただし、関心のあるのは 'BASE TABLES' のみです。

出力先のディレクトリ 'json_output' を作成し、各テーブルのスキーマ情報をこのディレクトリに JSON ファイルとして保存してください。ファイル名は <スキーマ>.<テーブル>.json の形式とします。問題が発生した場合は、それもログに記録してください。

テーブル構造を取得する関数

各テーブルの詳細な構造を取得するために、get_table_structure という関数を作成してください。この関数は、スキーマ名とテーブル名を受け取り、以下のカラム情報を取得する SQL クエリを実行します。

* COLUMN_NAME (カラム名)

* DATA_TYPE (データ型)

クエリが成功した場合、カラムの属性を含む辞書のリストを返してください。クエリが失敗した場合は、エラーメッセージをログに記録し、空のリストを返してください。

JSON ファイルの作成

データベース内の各テーブルについて、get_table_structure 関数を使用して JSON ファイルを生成してください。これらの JSON ファイルは、移行のための基礎情報となります。JSON ファイルは 'json_output' ディレクトリ内に保存し、各テーブルのエクスポートが成功した際にはログに記録してください。テーブルの構造が取得できなかった場合は、警告としてログに記録し、スクリプトの実行を中断することなく継続してください。

JSON ファイルの統合

すべての JSON ファイルを統合して、一つの JSON ファイル 'all_tables_structure.json' にまとめてください。以下の処理を実装してください。

1. 空の辞書を作成して、すべての JSON データを統合する。
2. データベース内の各テーブルについて、JSON ファイルを読み込み、辞書に追加する。
3. JSON ファイルが見つからない場合は、ログに記録する。
4. JSON のデコードエラーが発生した場合もログに記録する。

統合が完了したら 'all_tables_structure.json' に書き出し、成功をログに記録してください。エラーが発生した場合は、その詳細もログに記録してください。

データベース接続の終了

すべての処理が完了したら、データベース接続を適切に閉じてください。スクリプトの終了時にログを記録し、エクスポートと統合処理が完了したことを明確に伝えてください。

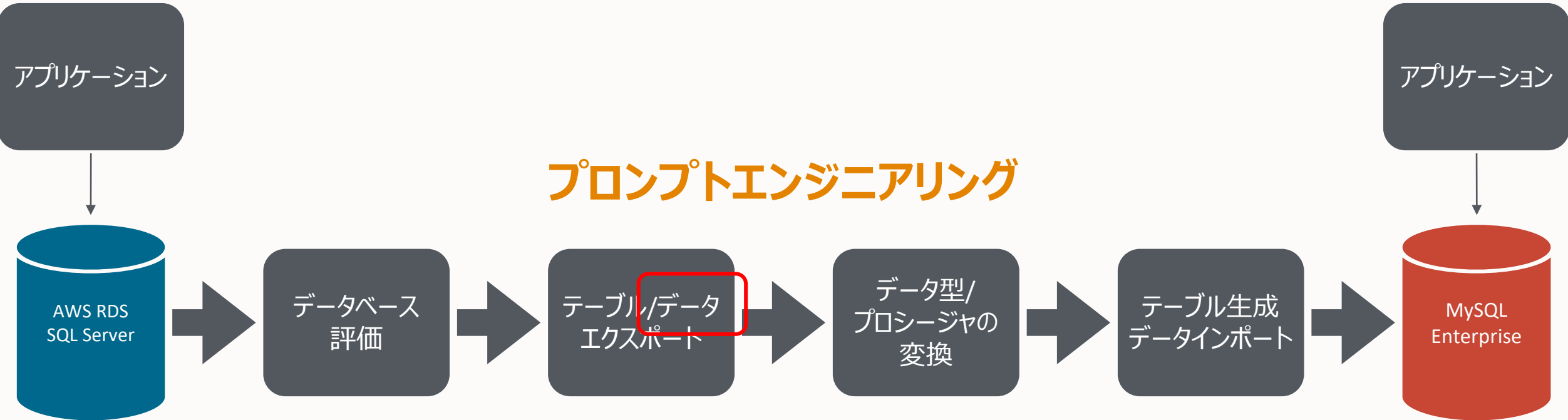
ベストプラクティスとエラーハンドリング

* スクリプトは PEP 8 に準拠し、可読性の高いフォーマットで記述すること。

* すべてのエラーは詳細なメッセージとともにログファイルに記録すること。

* エラーメッセージは明確にし、どの段階で問題が発生したのかを分かりやすくすること。

続く手順での生成 AI 活用 (2) SQL Serverデータベースから、データダンプを抽出



- 1. 作業量の評価
- 2. 移行の決断

- 3. データとオブジェクトの移行

- 4. テスト
- 5. 実運用

生成 AI サービス



プロンプト例: SQL Serverデータベースから、データダンプを抽出するbashスクリプト生成

export_all_tables.sh という名前の Bash スクリプトを作成してください。このスクリプトには以下の機能を含めてください。

スクリプトの基本設定

1. スクリプトの冒頭で必要な変数を定義する。

`SERVER`: 'sqlserver-xxxxxx-1.rds.amazonaws.com' (サーバーアドレス)

`USER`: 'admin' (データベースユーザー名)

`PASSWORD`: 'password123' (データベースパスワード)

`DATABASE`: 'AdventureWorks2019' (接続対象のデータベース名)

`OUTPUT_DIR`: '/xxxxx/Data' (エクスポートされたファイルの保存先ディレクトリ)

データベースからテーブル情報を取得

1. `sqlcmd`ユーティリティを使用してデータベース内の全テーブル名とスキーマを取得する。

* クエリの条件:

* INFORMATION_SCHEMA.TABLES から TABLE_SCHEMA と TABLE_NAME を選択する。

* BASE TABLE のみを対象とする。

* DATABASE 変数の値と一致するカタログのテーブルを取得する。

* 結果を TABLES 変数に格納する。

* sqlcmd の出力からヘッダーを削除 (-h -1) し、空白をトリム (-W) する。

2. 取得したテーブル名とスキーマを処理する。

* TABLES 変数から 1 行ずつ読み込む。

* 各行について:

* awk を使用して TABLE_SCHEMA と TABLE_NAME を抽出する。

* 変数 SCHEMA にスキーマ名を、TABLE にテーブル名を格納する。

テーブルデータのエクスポート準備

1. エクスポート用のテーブル参照を作成する。

* TABLE_NAME を "[スキーマ].[テーブル]" の形式で構築する。

* OUTPUT_FILE 変数を定義し、OUTPUT_DIR を基にテーブル名の .tsv ファイルパスを設定する。

2. 各テーブルを `bcp` コマンドでエクスポートする。

* エクスポート開始時に、対象テーブル名を表示する。

* bcp コマンドのフォーマット:

```
bcp "${DATABASE}.${SCHEMA}.${TABLE}" out
```

```
"${OUTPUT_FILE}" -c -t "¥t" -r "¥n" -S "$SERVER" -U "$USER" -P "$PASSWORD"
```

* コマンドのオプション:

* out オプションでエクスポートファイルを指定。

* フィールド区切りをタブ (-t "¥t")、行区切りを改行 (-r "¥n") に設定。

* サーバー (-s)、ユーザー名 (-u)、パスワード (-p) を指定。

* デバッグ用に構築された bcp コマンドを出力する。

3. `eval` コマンドを実行し、エクスポート結果を確認する。

* コマンドを eval で実行する。

* エラーチェック:

* \$? の値をチェックし、0 以外の場合はエラーをログに記録する。

* 失敗した場合は、エラーメッセージを表示し、生成された OUTPUT_FILE を削除する。

* 次のテーブルの処理に進む。

エクスポート後のデータ整理

1. エクスポートされたファイルのクリーンアップを行う。

* sed を使用して、ファイル内の不要な二重引用符を削除し、適切なタブ区切りを保持する。

* 一時ファイルとして保存し、元のファイルと置き換える。

2. エクスポートの成功メッセージを表示する。

* テーブルのエクスポートが完了したことを出力する。

スクリプトの最終処理

1. すべてのテーブルが処理された後、完了メッセージを表示する。

* すべてのテーブルのエクスポートが成功したことを示すメッセージを出力する。

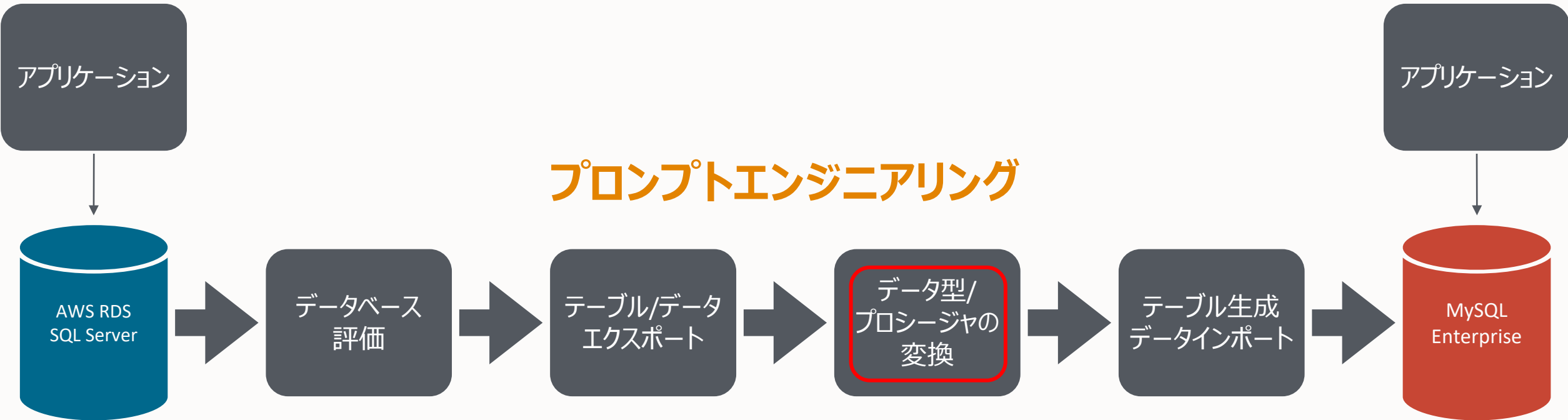
2. エラーハンドリングとユーザーフレンドリーな出力を強化する。

* すべてのコマンドの終了ステータスを確認し、成功または失敗を明確に表示する。

* スクリプト実行中に詳細なメッセージを表示し、進捗状況が分かるようにする。



続く手順での生成 AI 活用 (3) MySQLとの互換性のためにスキーマを変換するスクリプト



1. 作業量の評価
2. 移行の決断

3. データとオブジェクトの移行

4. テスト
5. 実運用

生成 AI サービス



プロンプト例: MySQLとの互換性のためにスキーマを変換するスクリプト生成 (1)

sqltomysql.py という名前の Python スクリプトを作成してください。
このスクリプトには以下の機能を含めてください。

必要なライブラリのインポート

* JSON データを扱うために json ライブラリをインポートする。

SQL Server のデータ型を MySQL のデータ型に変換する関数を定義

1. convert_sql_server_to_mysql 関数を定義する。

* この関数は 2 つのパラメータを受け取る。

* sql_server_type (SQL Server のデータ型)

* char_max_length (省略可能)

* SQL Server のデータ型を MySQL のデータ型にマッピングする
conversion_map 辞書を作成する。

* conversion_map には以下のデータ型の対応関係を定義する。

"INT" → "INT"

"TINYINT" → "TINYINT"

"SMALLINT" → "SMALLINT"

"BIGINT" → "BIGINT"

"BIT" → "TINYINT(1)"

"FLOAT" → "FLOAT"

"REAL" → "FLOAT"

"NUMERIC" → "DECIMAL(19,6)"

"DECIMAL" → "DECIMAL(19,6)"

"MONEY" → "DECIMAL(19,6)"

"SMALLMONEY" → "DECIMAL(10,6)"

"CHAR" → "CHAR"

"NCHAR" → "CHAR"

"VARCHAR" → "VARCHAR"

"NVARCHAR" → "VARCHAR"

"DATE" → "DATE"

"DATETIME" → "DATETIME(3)"

"DATETIME2" → "DATETIME(3)"

"SMALLDATETIME" → "DATETIME(3)"

"DATETIMEOFFSET" → "DATETIME(3)"

"TIME" → "TIME"

"TIMESTAMP" → "TIMESTAMP"

"ROWVERSION" → "TIMESTAMP"

"BINARY" → "BINARY"

"VARBINARY" → "BLOB"

"TEXT" → "LONGTEXT"

"NTEXT" → "LONGTEXT"

"IMAGE" → "LONGBLOB"

"UNIQUEIDENTIFIER" → "VARCHAR(64)"

"SYSNAME" → "VARCHAR(160)"

"XML" → "TEXT"

"GEOGRAPHY" → "BLOB"

"HIERARCHYID" → "BLOB"

2. 指定された SQL Server のデータ型に対応する MySQL のデータ型を返す。

* 該当するデータ型が conversion_map に存在しない場合は、元の sql_server_type を返す。

SQL Server のテーブル構造を MySQL 形式に変換する関数を定義

1. convert_table_structure 関数を定義する。

* 引数: sql_server_structure (SQL Server のテーブル構造を含む辞書)

* 処理内容:

1. 空の mysql_structure 辞書を初期化する。
 2. SQL Server の各テーブルを処理する。
 - * スキーマを除いたテーブル名を取得する。
 - * 空のリスト mysql_columns を初期化する。
 - * 各テーブルのカラム情報を処理する。
 - * カラム名、SQL Server のデータ型、最大長、NULL 許可の有無を取得する。
 - * convert_sql_server_to_mysql 関数を使用して、SQL Server のデータ型を MySQL のデータ型に変換する。
 - * CHAR, NCHAR, VARCHAR, NVARCHAR, BINARY, VARBINARY の場合は、最大長を付与する。
 - * 最大長が指定されていない場合、デフォルト値 255 を使用する。
 - * 変換されたカラム定義をリストに追加する。
 3. 変換したカラム情報を mysql_structure に格納する。
- * 戻り値: mysql_structure (MySQL のテーブル定義を含む辞書)

MySQL の CREATE TABLE 文を生成する関数を定義

1. generate_mysql_create_statements 関数を定義する。
 - * 引数: mysql_structure (MySQL のテーブル構造を含む辞書)
 - * 処理内容:
 1. create_statements リストを初期化し、最初に USE AdventureWorks2019; を追加する。
 2. 各テーブルについて以下の SQL を作成する。
 - * DROP TABLE IF EXISTS 文
 - * CREATE TABLE 文 (カラム定義を含む)
 3. 生成した SQL 文を create_statements に追加する。

プロンプト例: MySQLとの互換性のためにスキーマを変換するスクリプト生成 (2)

* 戻り値 : create_statements (MySQL の CREATE TABLE 文のリスト)

メイン処理を定義

1. main 関数を定義する。

* 処理内容 :

1. /xxxx/all_tables_structure.json の JSON ファイルを開く。
2. sql_server_structure という変数に JSON をロードする。
3. convert_table_structure 関数を実行し、SQL Server のテーブル構造を MySQL 形式に変換する。
4. generate_mysql_create_statements 関数を実行し、MySQL の CREATE TABLE 文を生成する。
5. 生成された SQL を /xxxx/Convert/mysql_create_statements.sql に書き出す。

スクリプトのエントリーポイント

* if __name__ == "__main__": ブロックを使用して main 関数を実行する。

* エラーハンドリングを実装する。ファイル操作や JSON 処理において適切な例外処理を行う。

ベストプラクティスとコーディングスタイル

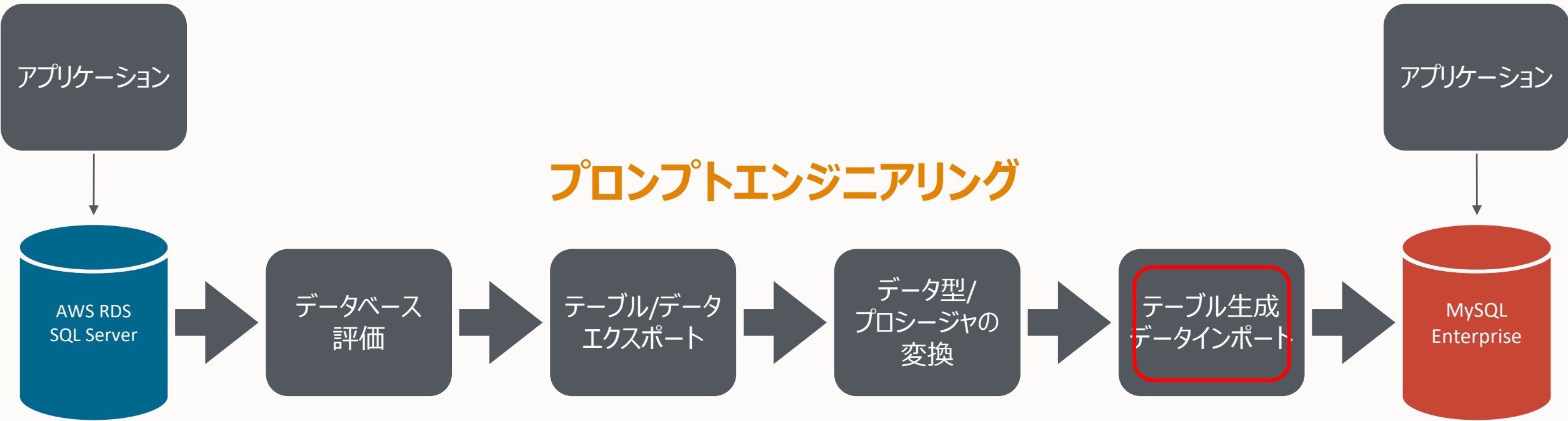
* PEP 8 に準拠したコードフォーマットを採用する。

* 変数名や関数名は分かりやすく命名する。

* 適切なインデントとコメントを使用して可読性を向上させる。

* エラー発生時には詳細なメッセージをログに記録する。

続く手順での生成 AI 活用 (4) MySQLデータベースにデータを投入するスクリプトを実行



1. 作業量の評価
2. 移行の決断

3. データとオブジェクトの移行

4. テスト
5. 実運用

生成 AI サービス



プロンプト例: MySQLデータベースにデータを投入するスクリプトを生成

import_all_tables.js という名前の JavaScript スクリプトを作成してください。このスクリプトには以下の機能を含めてください。

変数の定義

1. データファイルの保存場所を指定する。
 - * baseDir 変数を '/xxxxxxx/Data' に設定し、インポート対象のデータファイルのベースディレクトリとする。
2. インポート対象のテーブル一覧を定義する。
 - * tables 配列に以下のテーブル名を含める。
'PersonPhone', ...(中略)... 'SalesOrderHeaderSalesReason', 'JobCandidate'.
3. 使用するデータベーススキーマを指定する。
 - * schema 変数を 'AdventureWorks2019' に設定し、すべての操作で使用するスキーマとする。

データベースセッションの確立

1. MySQL との接続を確立する。
 - * mysql.getSession() メソッドを使用して MySQL のセッションを作成し、session という変数に格納する。

テーブルごとのデータインポート処理

1. 各テーブルについて処理を実行する。
 - * tables 配列の各テーブルに対して forEach メソッドを使用して処理を行う。
2. テーブルごとのデータファイルパスを作成する。
 - * ループ内で filePath 変数を作成し、baseDir と現在のテーブル名を結合して .tsv 拡張子のファイルパスを生成する。

テーブルの既存データを削除 (TRUNCATE)

1. TRUNCATE TABLE の実行
 - * 各テーブルのデータをクリアするため、TRUNCATE TABLE コマンドを実行する。
 - * session.sql() メソッドを使用し、テーブルを空にする SQL を実行する。
 - * 成功時: ログに成功メッセージを出力する。
 - * エラー発生時: try...catch ブロックを使用し、エラーメッセージをログに記録する。

データのインポート

1. TSV データを MySQL にインポートする。
 - * インポート開始時に、現在のテーブルのインポートを開始することをログに記録する。
 - * util.importTable メソッドを使用してデータをインポートする。
 - * オプション設定:
 - * schema: schema 変数を設定
 - * table: 現在のテーブル名を設定
 - * dialect: 'tsv' (タブ区切りフォーマット)
 - * linesTerminatedBy: ¥n (改行ごとにデータを区切る)
 - * showProgress: true (進捗状況を表示)
 - * fieldsTerminatedBy: ¥t (フィールドをタブで区切る)
 - * 成功時: テーブルが正常にインポートされたことをログに記録する。
 - * エラー発生時: try...catch を使用してエラーメッセージをログに記録する。

処理の完了

1. すべてのテーブルの処理が終了した後、完了メッセージを表示する。
 - * 全テーブルのデータインポートが終了したことをログに記録する。



6. ストアドプロシージャ移行でのJSの活用

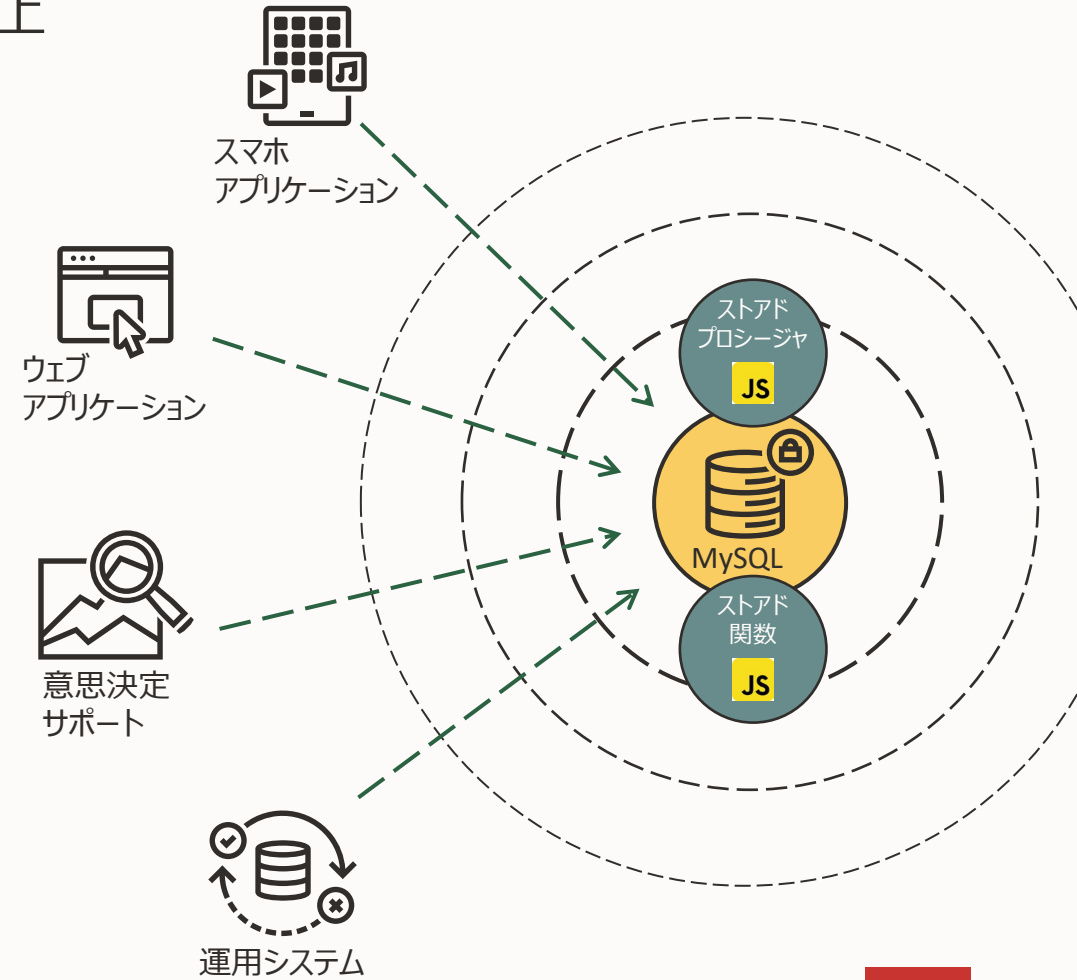
SQL ServerからMySQLへのストアドプロシージャの移行

- ストアドプロシージャの有用性
 - アプリケーションとデータベース間のデータ転送を最小限に抑え、開発プロセスを効率化
- SQL Server の T-SQL は MySQL では使えない
 - ストアドプロシージャ移行には柔軟性のある移行手段が望ましい
- MySQL のストアドプロシージャ (SQL/PSM) を使用する場合、いくつかの課題がある
 1. 言語としての制約
 - 柔軟性の欠如（複雑なロジックが苦手、ループや条件分岐の制約）
 - オブジェクト指向の欠如
 2. データ操作の制限
 - 動的 SQL の扱いの難しさ（PREPAREは複雑）
 - エラーハンドリング（DECLARE HANDLER などは直感的でない）
 3. JSON データの取り扱い

MySQL での JavaScript ストアドプログラム

Enterprise Editionで提供

- ストアドプログラムでデータ指向アプリケーションの機能性を向上
 - コスト削減
 - データ移動の低減
 - セキュリティ改善
 - 簡潔なアーキテクチャ



JavaScriptストアードプロシージャのメリット

項目	SQL/PSMストアードプロシージャ	JavaScript ストアードプロシージャ
言語の柔軟性	制約が多く、複雑なロジックに不向き	一般的なプログラミング言語として柔軟
JSONデータの扱い	SQL関数を駆使する必要がある	JavaScriptのオブジェクトとして直感的に操作可能
エラーハンドリング	DECLARE HANDLERを使うが複雑	try...catchで簡潔に記述可能
動的SQL	PREPAREを使うが扱いづらい	session.prepare()で柔軟に記述可能
開発者の学習コスト	SQLの拡張に特化した学習が必要	JavaScriptの既存の知識がそのまま使える



JavaScriptストアプロセスの有効化



- MySQL Enterprise Edition での JavaScript サポートについて、[The Oracle MySQL Japan Blog](#) の中で記事として採り上げています
 - [MySQLのJavaScriptサポートについて](#)
- MySQL EE で JavaScript サポートを有効にするには、以下の対応が必要になります
 - コンポーネントインストールのため、SELinux では一時的に Enforcing モードをオフにする必要があります
 - JavaScript コンパイラ導入のために、**INSTALL COMPONENT "file://component_mle";** を実行する必要があります
 - グローバル関数 `log_bin_trust_function_creators` をオンにする必要があります
(あるいはこれを行わない場合、ストアード関数の宣言時に都度 **DETERMINISTIC** キーワードを加えます)



SQL ServerからMySQLへのストアドプロシージャの移行 (1)

```
CREATE PROCEDURE GetEmployeeDetailsByDepartment
    @DepartmentID INT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        IF NOT EXISTS (
            SELECT 1 FROM HumanResources.Department WHERE DepartmentID = @DepartmentID
        )
        BEGIN
            RAISERROR('Invalid DepartmentID provided.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END
        WITH EmployeeDetails AS (
            SELECT e.BusinessEntityID, e.JobTitle, p.FirstName, p.LastName,
                d.Name AS DepartmentName, d.GroupName AS DepartmentGroup
            FROM HumanResources.Employee e
            JOIN Person.Person p ON e.BusinessEntityID = p.BusinessEntityID
            JOIN HumanResources.EmployeeDepartmentHistory edh ON e.BusinessEntityID = edh.BusinessEntityID
            JOIN HumanResources.Department d ON edh.DepartmentID = d.DepartmentID
            WHERE d.DepartmentID = @DepartmentID
        )
        SELECT * FROM EmployeeDetails ORDER BY LastName, FirstName;
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        RAISERROR (ERROR_MESSAGE(), ERROR_SEVERITY(), ERROR_STATE());
    END CATCH
END;
```

MS SQL Server
T-SQL ストアドプロシージャの例

- DepartmentID に基づいて従業員情報を取得
 - * try~catch
 - * トランザクション
 - * WITH句によるCTE (Common Table Expressions)

SQL ServerからMySQLへのストアドプロシージャの移行 (2)

```
CREATE PROCEDURE uspGetEmpJS (IN inputDepartmentID INT, OUT result JSON)
LANGUAGE JAVASCRIPT
AS $$
{
  try {
    let employees = [];
    let selectStatement = session.prepare(`
      SELECT e.JobTitle, COUNT(e.BusinessEntityID) AS EmployeeCount
      FROM Employee e
      JOIN EmployeeDepartmentHistory edh ON e.BusinessEntityID = edh.BusinessEntityID
      JOIN Department d ON edh.DepartmentID = d.DepartmentID
      WHERE d.DepartmentID = ?
      GROUP BY e.JobTitle
      ORDER BY EmployeeCount DESC
    `);
    let resultSet = selectStatement.bind(inputDepartmentID).execute();
    let row = null;
    while (row = resultSet.fetchOne()) {
      employees.push({
        JobTitle: row.JobTitle,
        EmployeeCount: row.EmployeeCount
      });
    }
    result = employees.length > 0 ? employees : { message: "No employees found for DepartmentID: " + inputDepartmentID };
  } catch (error) {
    result = { error: "An error occurred: " + error.message };
  }
}
$$;
```

MySQL

JavaScript ストアドプロシージャの例

- DepartmentID に基づいて従業員情報を取得
 - * try~catch対応
 - * トランザクション未対応
 - * CTE未対応

SQL ServerからMySQLへのストアドプロシージャの移行 (3)

```
CREATE PROCEDURE uspGetEmpJS (IN inputDepartmentID INT, OUT result JSON)
LANGUAGE JAVASCRIPT
AS $$
{
  try {
    session.sql("START TRANSACTION").execute();
    let employees = [];
    let selectStatement = session.prepare(`
      WITH EmployeeCTE AS (
        SELECT e.BusinessEntityID, e.JobTitle, d.DepartmentID
        FROM Employee e
        JOIN EmployeeDepartmentHistory edh ON e.BusinessEntityID = edh.BusinessEntityID
        JOIN Department d ON edh.DepartmentID = d.DepartmentID
        WHERE d.DepartmentID = ?
      )
      SELECT JobTitle, COUNT(BusinessEntityID) AS EmployeeCount
      FROM EmployeeCTE
      GROUP BY JobTitle
      ORDER BY EmployeeCount DESC
    `);
    let resultSet = selectStatement.bind(inputDepartmentID).execute();
    let row = null;
    while (row = resultSet.fetchOne()) {
      employees.push({
        JobTitle: row.JobTitle,
        EmployeeCount: row.EmployeeCount
      });
    }
    session.runSql(employees.length > 0 ? "COMMIT" : "ROLLBACK");
    result = employees.length > 0 ? employees : { message: "No employees found." };
  } catch (error) {
    session.runSql("ROLLBACK");
    result = { error: "An error occurred: " + error.message };
  }
}
$$;
```

MySQL

JavaScript ストアドプロシージャの例

- DepartmentID に基づいて従業員情報を取得
 - * try～catch対応
 - * トランザクション対応
 - * WITH句によるCTE対応
(Common Table Expressions)

7. まとめ



まとめ

データベースの自由を解放つ: 生成AI/JavaScriptを活用したSQL ServerからMySQLへの移行

- MySQL Enterprise Edition を採用する利点
 - データ保護と規制遵守
 - 高いスケーラビリティ
 - 高可用性と災害復旧
 - 大きな運用コスト削減
- 移行の障壁
 - マイグレーションの作業コスト
 - スキルのある技術者の確保が困難
- 生成 AI を用いたプロンプトスクリプティングの活用
 - スキルギャップを埋める
 - 作業コストの削減
- JavaScriptを用いたストアードプロシージャの移行
 - 柔軟な表現、直感的なデータ操作



MySQL for Developers License

学習、開発用途であれば、MySQL Enterprise EditionをOTNからダウンロードして無償で使用可能に！！

※OTN : Oracle Technology Network

Full access to MySQL Enterprise Edition

- Enterprise Server
- Backup
- Router
- Shell
- Connectors
- JavaScript

Learn, Develop, Prototype

 MySQL Enterprise Masking De-identify, Anonymize Sensitive Data	 MySQL Enterprise TDE AES 256 encryption, Key Management	 MySQL Enterprise Authentication External Authentication Modules	 MySQL Enterprise Encryption Public/Private Key Cryptography, Asymmetric Encryption
 MySQL Enterprise Firewall Block SQL Injection Attacks, Intrusion Detection	 MySQL Enterprise Audit User Activity Auditing, Regulatory Compliance	 MySQL Enterprise Thread Pool Performance & Scalability for enterprise workloads	 MySQL Enterprise Backup Secure Backups, AES 256 encryption

Download Now

<https://www.oracle.com/mysql/technologies/mysql-enterprise-edition-downloads.html>

ORACLE