Snowflake SQLエラー解決ガイド

Sigma・Power BI連携の原因切り分けと対処法

Start Challenge Consulting co., ltd.

2026年8月発行

Start Challenge
Consulting co., ltd.
挑む力が、世界を照らす

Confidential



はじめに：SQLエラー対応が難しい理由

Snowflake環境では、同じSQLでも実行する場所・ユーザー・BIツールの処理によって結果が変わる。
原因は必ずしもSQL構文にはなく、「どのレイヤーで失敗しているか」の特定が解決の起点となる。

LAYER 01 **SQL・データ**	■ SQL構文の誤り ■ データ型・NULLの問題 ■ データベース・スキーマの指定漏れ
LAYER 02 **Snowflake基盤・接続**	■ ロール・権限の不足 ■ ウェアハウスの停止・設定不備 ■ 接続ユーザー・実行環境の違い
LAYER 03 **BIツール（Sigma／Power BI）**	■ Sigma・Power BI側のSQL変換 ■ キャッシュ・更新タイミングの違い

本ページで伝えること

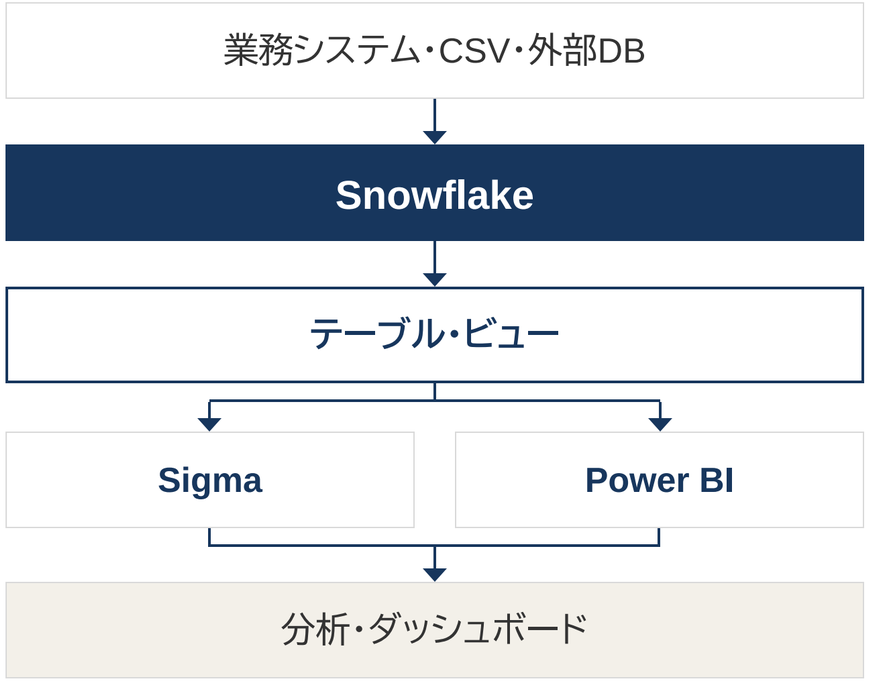
「SQLが悪い」と決めつけず、どのレイヤーで問題が起きているかを切り分ける。

事実：Snowflake・Sigma・Power BIは実行主体と処理経路が異なる ／ 解釈：代表的な原因は3レイヤー・8分類に整理できる ／ 提案：SQL修正の前にレイヤー特定を行う ／ 出典：提供された運用整理内容に基づき構成

Snowflake・Sigma・Power BIの全体構成

データはSnowflakeを起点にテーブル・ビューを経てSigma／Power BIへ流れるため、エラーは必ずどこかのレイヤーに帰属する。レイヤーごとの典型エラーを把握することが、原因特定の時間を短縮する。

データの流れ



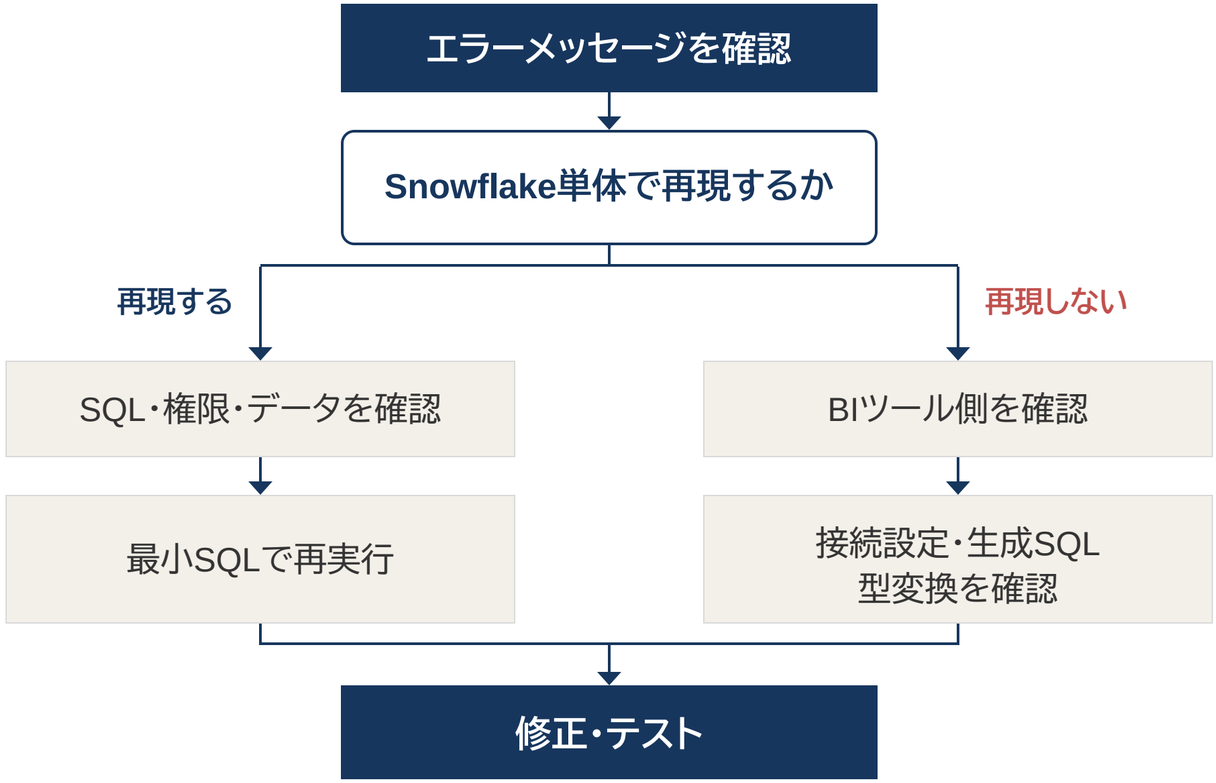
エラーの発生箇所

レイヤー	主なエラー
Snowflake	構文、権限、型変換、オブジェクト参照
接続・認証	アカウント、ロール、ウェアハウス
Sigma	カスタムSQL、データセット、関数変換
Power BI	Power Query、DirectQuery、型変換、更新
レポート	集計粒度、フィルター、リレーション

エラーは必ずいずれかのレイヤーに帰属する。
まず発生レイヤーを特定し、そのレイヤーの典型原因から順に確認する。

SQLエラー対応の基本フロー

「Snowflake単体で再現するか」が最初の分岐点となる。再現すればSQL・権限・データ、
再現しなければBIツール側の接続設定・生成SQL・型変換を確認する。



対応の基本原則

1. エラーメッセージを省略せず保存する
2. クエリIDを確認する
3. 実行ユーザー・ロール・ウェアハウスを確認する
4. 複雑なSQLを最小単位に分解する
5. Snowflake上で直接実行して再現性を確認する

解釈：再現性の有無で調査対象レイヤーを二分できる ／ 提案：エラー対応は「Snowflake単体での再現確認」を最初の分岐として標準化する ／ 出典：提供された運用整理内容に基づき構成

構文エラーの原因と対処法

構文エラーはエラー行そのものではなく直前の行に原因があることが多く、末尾カンマ・予約語・他DB方言の混入が典型パターンとなる。識別子の大文字小文字も再現性を左右する。

代表的なエラー

```
SQL compilation error
Syntax error
Unexpected …
Invalid identifier
```

よくある原因

- カンマや括弧の不足
- 予約語をカラム名に使用
- エイリアスの指定ミス
- 大文字・小文字の不一致
- 存在しないカラムの参照
- SQL方言の違い

修正例：末尾カンマ

エラーになる例

```
SELECT
    customer_id,
    customer_name,
FROM customer_master;
```

↓

修正後

```
SELECT
    customer_id,
    customer_name
FROM customer_master;
```

チェックポイント

1. エラー行だけでなく**直前の行**も確認する
2. 引用符で作成した識別子は**大文字・小文字を区別**する
3. 他DB向けのSQL関数をそのまま使っていないか確認する

Invalid identifierは「構文」ではなく「参照」の問題であることが多い（→P7）

事実：SELECT句末尾のカンマはFROM句の解析失敗を招く ／ 解釈：構文エラーの多くは記述ミスと方言差に集約される ／ 出典：提供された運用整理内容に基づき構成

データ型・NULL・日付エラーの対処法

型変換エラーはTRY系関数でクエリを止めずに処理できるが、それは対処であって解決ではない。
不正データの件数と発生元を特定し、データ品質側で是正することが再発防止につながる。

代表的な問題

- 文字列を数値に変換できない
- 日付形式が統一されていない
- NULLを含む計算が想定外になる
- 数値と文字列を比較している
- タイムゾーンによって日付がずれる

POINT

**エラー行を除外するだけでなく、
不正データの件数と発生元を特定する。**

修正例：数値変換

エラーになりやすい例

```
SELECT TO_NUMBER(sales_amount)
FROM sales;
```

↓

安全な書き方

```
SELECT TRY_TO_NUMBER(sales_amount)
FROM sales;
```

TRY系関数は変換不能値を**NULLとして返す**ため、
クエリは成功するが件数の欠落に注意する。

実務で使う関数

- `TRY_TO_NUMBER`
- `TRY_TO_DATE`
- `TRY_CAST`
- `COALESCE`
- `NULLIF`
- `TO_TIMESTAMP`
- `CONVERT_TIMEZONE`

事実：TRY系関数は変換失敗時にエラーを返さずNULLを返す ／ 解釈：クエリ成功と結果の正しさは別問題 ／ 提案：不正データの件数と発生元をあわせて確認する ／ 出典：提供された運用整理内容に基づき構成

テーブル・ビュー・スキーマ参照エラー

「オブジェクトが存在しない」エラーの多くは、オブジェクト自体の不在ではなく参照先の指定漏れである。
三層完全修飾名で記述し、実行時のコンテキストを明示的に確認することで切り分けが早くなる。

代表的なエラー

```
Object does not exist
Table does not exist
Schema does not exist
View definition is invalid
```

主な原因

- データベースやスキーマの指定漏れ
- 接続先環境の間違い
- テーブル名の変更
- ビューが参照するテーブルの削除
- 開発・検証・本番で名称が異なる

推奨する記述：三層完全修飾名

```
SELECT *
FROM PRODUCTION_DB.SALES_SCHEMA.SALES_DATA;
```

■ データベース　■ スキーマ　■ テーブル／ビュー

確認用SQL：実行コンテキストの明示

```
SELECT
    CURRENT_DATABASE(),
    CURRENT_SCHEMA(),
    CURRENT_ROLE(),
    CURRENT_WAREHOUSE();
```

なぜ確認するか

同じSQLでも、接続しているDB・スキーマ・ロールが違えば結果は変わる。「存在しない」の前に「どこを見ているか」を確定させる。

事実：Snowflakeのオブジェクトはデータベース・スキーマ・オブジェクトの三層で識別される ／ 提案：BI連携用SQLは完全修飾名で記述する ／ 出典：提供された運用整理内容に基づき構成

Snowflakeの権限エラーとロール設計

権限はWarehouse・Database・Schema・Table/Viewの4階層すべてが揃って初めて参照可能になる。
管理者ロールでの成功は、BI接続ロールでの成功を保証しない。

代表的なエラー

```
Insufficient privileges
Not authorized
Cannot operate on warehouse
```

特定のユーザーだけテーブルを参照できない

必要になる主な権限（4階層すべて必須）

- Warehouse へ **USAGE**
- Database へ **USAGE**
- Schema へ **USAGE**
- Table／View へ **SELECT**

付与例：BI専用ロールへの権限付与

```
GRANT USAGE ON WAREHOUSE BI_WH
  TO ROLE BI_ROLE;

GRANT USAGE ON DATABASE ANALYTICS_DB
  TO ROLE BI_ROLE;

GRANT USAGE ON SCHEMA ANALYTICS_DB.PUBLIC
  TO ROLE BI_ROLE;

GRANT SELECT ON ALL TABLES
  IN SCHEMA ANALYTICS_DB.PUBLIC
  TO ROLE BI_ROLE;
```

重要ポイント

1. Snowflake画面の利用者とBI接続ユーザーは**別の場合がある**
2. 管理者ロールで成功しても、BI用ロールで成功するとは限らない
3. 権限付与は**最小権限**を基本とする

切り分け時は、BI接続ロールに切り替えて同じSQLを実行する。

事実：SELECT権限だけでは参照できず上位オブジェクトのUSAGEが必要 ／ 提案：BI専用ロールを設け最小権限で付与する ／ 出典：提供された運用整理内容に基づき構成

パフォーマンス・タイムアウトエラーの対処法

SQLが正しくても終わらないケースは、全件取得と不要列取得が起点になっていることが多い。
取得列と期間を絞り込むことが、タイムアウト解消とコスト抑制の双方に効く。

主な症状

- SQLは正しいが処理が終わらない
- SigmaやPower BIでタイムアウトする
- 更新処理が途中で失敗する
- Snowflakeの利用コストが高くなる

主な原因

- 全件取得
- 不要な列の取得
- 巨大テーブル同士のJOIN
- JOIN条件の不足
- 重複データによる行数の爆発
- BIツールから同じSQLが繰り返し実行される

改善例：取得列と期間の絞り込み

改善前

```
SELECT *
FROM large_sales_table;
```

↓

改善後

```
SELECT
    sales_date,
    customer_id,
    sales_amount
FROM large_sales_table
WHERE sales_date >=
    DATEADD(month, -3, CURRENT_DATE());
```

確認項目

- Query Profile
- 実行時間
- スキャン量
- JOIN前後の件数
- ウェアハウスサイズ
- キャッシュ利用状況

事実：Snowflakeは列指向のため取得列を絞るとスキャン量が減る ／ 解釈：タイムアウトはSQLエラーではなく設計の問題 ／ 提案：期間・列の絞り込みを標準化する ／ 出典：提供された運用整理内容に基づき構成

Sigma連携後に起きるSQLエラー

「Snowflakeでは動くのにSigmaでは動かない」の主因は、Sigmaの接続ロールとSigmaが生成するSQLである。
Sigmaが実際に発行したSQLを取得し、同じロールでSnowflake上から再実行することが最短の切り分けとなる。

Sigmaで発生しやすい問題

- Snowflakeでは動くSQLがSigmaでは動かない
- カスタムSQLの結果が表示されない
- カラム変更後にWorkbookがエラーになる
- ユーザーによって表示結果が違う
- 日付・時刻の値がずれる

**「ユーザーによって違う」場合は
ロールまたはRow Access Policyを疑う。**

確認するポイント

- Sigmaの接続ユーザー
- 適用されているSnowflakeロール
- 使用ウェアハウス
- データセット・Workbookの参照先
- カラム名・型の変更
- Row Access Policy
- Sigmaが生成したSQL
- キャッシュ・更新状況

対処の基本ステップ

1. **Sigmaから実行されたSQLを確認**
2. 同じロールでSnowflake上から実行
3. カスタムSQLを最小化
4. 元テーブルとSigma側の型を比較
5. データセットの再同期・更新を確認

解釈：SigmaはSnowflakeへSQLを生成・発行するため、同一SQLとは限らない ／ 提案：生成SQLの取得と同一ロールでの再実行を切り分けの起点とする ／ 出典：提供された運用整理内容に基づき構成

Power BI連携後に起きるSQLエラー

Power BIはPower Queryによる自動型変換とQuery Foldingを経てSQLを発行するため、
表示されたエラーの発生源がSnowflakeであるとは限らない。DesktopとService の差異も切り分け対象となる。

Power BI特有の原因

- Power Queryによる自動型変換
- ImportとDirectQueryの動作差
- Query Foldingの失敗
- Snowflake側に存在しない関数への変換
- Power BI Serviceでの更新失敗
- DesktopとServiceの認証情報の違い
- リレーションによる重複集計

Power BIの処理経路とエラー発生点



赤枠の工程はPower BI内部で完結するため、Snowflakeのクエリ履歴には現れない。

特に確認する箇所

- Power Queryの「適用したステップ」
- データソース設定
- ネイティブクエリ
- プライバシーレベル
- DirectQuery制限
- Power BI Serviceの資格情報
- オンプレミスデータゲートウェイの要否
- タイムゾーンとロケール

Import と DirectQuery の違い

Import
取り込み時点のデータを保持。更新タイミングとキャッシュに依存する。

DirectQuery
操作ごとにSQLを発行。関数制限とタイムアウトの影響を受ける。

Power BIで表示されるエラーが、必ずしもSnowflakeのSQLエラーとは限らない。

事実：Power BIはPower Queryを経由してSQLを生成・発行する ／ 解釈：Power BI内部で完結する処理はSnowflake側に記録されない ／ 出典：提供された運用整理内容に基づき構成

Snowflakeでは成功、BIでは失敗する場合の切り分け

実行ユーザー・ロール・ウェアハウス・SQL・データ型・キャッシュ・更新の7項目は、3ツールで実体が異なる。
この7項目を並べて比較すれば、差異が生じている箇所を機械的に特定できる。

確認項目	Snowflake	Sigma	Power BI
実行ユーザー	手動実行者	接続ユーザー	接続資格情報
ロール	Current Role	接続ロール	接続設定
ウェアハウス	Current Warehouse	Sigma設定	Snowflake接続
SQL	手動SQL	生成・カスタムSQL	Power Query
データ型	Snowflake型	Sigma型	Power BI型
キャッシュ	Result Cache	Workbook Cache	Import Cache
更新	即時	接続設定による	スケジュール更新

同じSQLでも「誰が・どのロールで・どのウェアハウスで・どの型で」実行するかが違えば、結果は変わる。

推奨する切り分け順序

1. **Snowflake上でSQLが成功するか**
2. BI接続ユーザーと同じロールで成功するか
3. BIツールが実際に発行したSQLを確認
4. BI側の型変換処理を外して実行
5. 少量データに限定して再現確認

解釈：3ツールで7項目の実体が異なるため差異比較で原因箇所を特定できる ／ 提案：切り分けは1→5の順に固定して実施する ／ 出典：提供された運用整理内容に基づき構成

よくある失敗事例と修正方法

4つの典型事例はいずれも「SQL構文の誤り」ではなく、型変換・権限・JOIN粒度・認証情報の差に起因する。
したがって修正はSQLの書き換えだけでなく、Snowflake側での型統一や権限・接続設定の是正が必要となる。

事例1

Power BIだけ 日付変換エラー

原因

不正な日付文字列をPower BIが日付型へ自動変換

対処

Snowflake側で`TRY_TO_DATE`を使用し、型を統一

事例2

Sigmaでテーブルが 表示されない

原因

BI接続ロールにスキーマの`USAGE`がない

対処

Database・Schema・Tableの権限を順番に確認

事例3

JOIN後に 売上金額が増える

原因

1対多のJOINで売上行が重複

対処

JOIN前にキーの重複を確認し、必要に応じて事前集計

事例4

Desktopでは成功するが 自動更新で失敗

原因

Power BI Service側の資格情報・ロールが異なる

対処

Service側の接続情報と権限を再確認

解釈：4事例はいずれもSQL構文以外のレイヤーに原因がある ／ 提案：型統一・権限確認・JOIN粒度確認・接続情報確認を定型作業とする ／ 出典：提供された運用整理内容に基づき構成

エラーを予防する設計・運用ルール

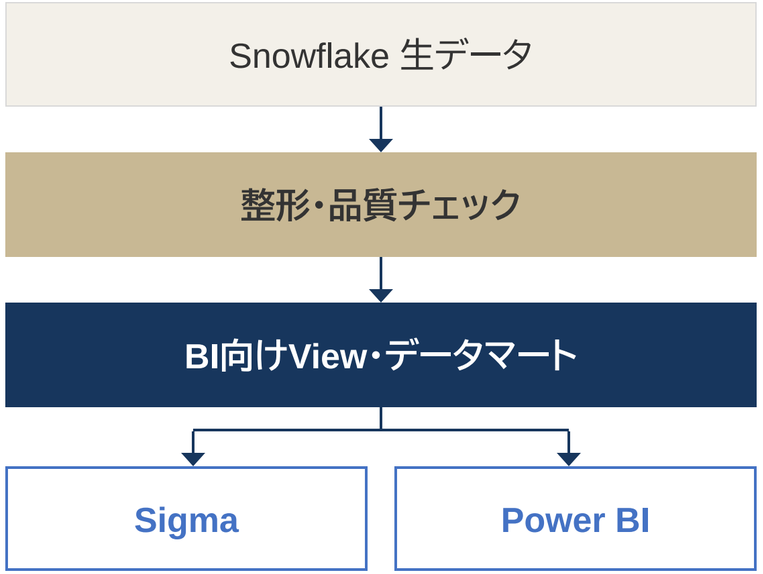
BIから本番テーブルを直接参照させず、BI向けView・データマートを中間層として設けることで、
カラム変更や型変更の影響をBI側から遮断でき、エラーの発生源そのものを減らせる。

推奨ルール

- BIから本番テーブルを直接参照しすぎない
- BI向けViewまたはデータマートを作成する
- カラム名・データ型を可能な限り固定する
- SQLの変更履歴をGitなどで管理する
- 開発・検証・本番環境を分ける
- BI専用ユーザー・ロールを用意する
- エラー時にクエリIDを記録する
- `SELECT *` を避ける
- データ品質チェックを自動化する
- 変更時の影響範囲を事前確認する

設計で防ぐ	権限で防ぐ	運用で防ぐ
中間層のView・データマート、名称と型の固定、環境分離	BI専用ユーザー・ロールの分離、最小権限での付与	変更履歴管理、クエリID記録、品質チェック自動化、影響範囲の事前確認

推奨構成：BI向け中間層を挟む



中間層を設けることで、生データ側の変更がBIへ直接波及しない。

解釈：予防策は設計・権限・運用の3分類に整理できる ／ 提案：BI向け中間層の設置を最優先で実施する ／ 出典：提供された運用整理内容に基づき構成

SQLエラー対応チェックリスト・まとめ

13項目のチェックリストを「記録」「Snowflake確認」「BI確認」の3段階で運用すれば、
属人的な調査から脱し、原因特定と再発防止を再現可能なプロセスにできる。

STEP 1 ｜ 記録する

- ☐ エラーメッセージ全文を保存したか
- ☐ クエリIDを記録したか
- ☐ 実行日時を確認したか
- ☐ 実行ユーザーとロールを確認したか

STEP 2 ｜ Snowflakeを確認する

- ☐ データベース・スキーマを確認したか
- ☐ ウェアハウスが起動しているか
- ☐ カラム名とデータ型を確認したか
- ☐ NULLや不正値を確認したか
- ☐ JOIN前後の件数を確認したか
- ☐ Snowflake上で直接再現したか

STEP 3 ｜ BIツールを確認する

- ☐ BIツールが生成したSQLを確認したか
- ☐ BI側の型変換・更新設定を確認したか
- ☐ 修正後に同じ条件で再テストしたか

まとめ

**SQLエラーを早く解決するために必要なのは、SQLの知識だけではない。
Snowflake、接続ユーザー、権限、データ型、BIツールの処理を一つずつ切り分けることで、
原因特定と再発防止が可能になる。**

ご支援できること

- Snowflake・BI環境のエラー診断
- Sigma／Power BI連携支援
- SQL・データモデルの改善相談
- データ基盤運用の伴走支援

提案：13項目のチェックリストを3段階の標準プロセスとして運用する ／ 出典：提供された運用整理内容に基づき構成

お問い合わせ先

Snowflake・Sigma・Power BI環境のエラー診断から、データ基盤運用の伴走支援まで対応いたします。
本資料の内容に関するご質問・ご相談は、下記までお問い合わせください。

Start Challenge
Consulting co., ltd.
挑む力が、世界を照らす

Start Challenge Consulting co., ltd.

挑戦力が、世界を変える。

お問い合わせ先

【Start Challenge Consulting】

E-mail　**info@startchg.com**

担当者　**平出**

ご支援できること

- Snowflake・BI環境のエラー診断
- Sigma／Power BI連携支援
- SQL・データモデルの改善相談
- データ基盤運用の伴走支援

本資料は情報提供を目的としたものであり、記載内容の利用により生じた損害について当社は責任を負いません。／ Confidential