ViewLegacy.

# AIエージェントへの直接DBアクセスとケイパビリティベースアクセスの安全性比較

技術白書

発行: 合同会社ビューレガシー
著者: 増田朝陽
発行日: 2026年5月19日

本書は、AIエージェントを業務データベースへ接続する4つの方式を比較する。直接DB認証情報、自然言語からSQLを生成する方式、汎用CRUD API、エージェント定義の業務ケイパビリティである。

# 1. エグゼクティブサマリー

企業は、顧客、注文、在庫、会計、従業員などのシステムに対して、AIエージェントが業務上の質問に答えることを求め始めている。ここでの設計上の問いは、エージェントがデータベースを呼べるかどうかではない。プロンプト、ツール説明、検索されたコンテンツが操作されうる状況で、エージェント実行環境にどれだけの権限を渡すべきかである。

本書は、直接DB認証情報、SQL生成エージェント、汎用テーブルレベルCRUD API、エージェント定義ケイパビリティベースアクセスの4方式を比較する。結論はバランスを取ったものである。ケイパビリティベースアクセスは、AIエージェントの誤動作やプロンプトインジェクションに起因する悪用の影響範囲を大きく下げうるが、すべてのリスクを消すものではない。

ケイパビリティベースアクセスは、分析、大規模BI、柔軟なアドホッククエリを必要とするワークロードに常に適するわけではない。一方で、操作を明示的な業務ケイパビリティとして表現できる、読み取り中心のレガシーシステム連携には特に適している。

## 2. 背景

AIエージェントは、業務記録システムからデータを取得する必要性が高まっている。多くの組織では、そのシステムは自律的なエージェント実行環境ではなく、社内アプリケーション向けに設計されたレガシーなリレーショナルデータベースである。同じスキーマ内に、顧客、注文、在庫、会計、従業員などの機密データが含まれることも多い。

直接DBアクセスは、AIアプリケーションに広い技術的インターフェースを与える。認証情報、スキーマ可視性、DBロールが許可するSQL実行能力である。自然言語からSQLを生成する方式では、さらに変換の問題が加わる。モデルがユーザー意図を推定し、正しいSQLを生成し、認可を守り、無関係なデータを開示しないことが求められる。

MCPとツール呼び出しは、エージェントが業務システムを呼ぶことを容易にする。しかし、その利便性は最小権限の必要性を取り除かない。ツール公開は単なるドキュメント層ではなく、認可境界として設計されるべきである。

# 3. 比較対象のアクセスパターン（1/4）

## パターン1: 直接DB認証情報

この方式では、AIアプリケーションまたはエージェント実行環境がデータベース認証情報を受け取る。エージェントは、そのDBロールから見えるスキーマを確認でき、そのロールが許可するSQLを実行できる。認可は主に、データベース権限、アプリケーションプロンプト、場合によってはクエリ生成ガードレールによって行われる。

エージェントが侵害されたり誤誘導されたりした場合、攻撃者はテーブル列挙、無関係なデータの結合、本来の業務フローの迂回、書き込み権限があればデータ変更を指示できる。DBユーザーが読み取り専用であっても、広い読み取りロールは業務タスクに必要な範囲を超えるデータを公開しうるため、機密性リスクは高い。

この方式は生のスキーマを露出し、通常はDBエラーの挙動も露出する。また、認証情報がAI実行環境の設定、ログ、トレース、シークレットストアに置かれる可能性がある。

# 3. 比較対象のアクセスパターン（2/4）

## パターン2: SQL生成エージェント

自然言語からSQLを生成するエージェントは、モデルにデータベースインターフェースを与え、ユーザー意図からSQLを生成させる。エージェントは多くの場合、スキーマメタデータや例を参照できる。生成されたSQLとDBロールが許す範囲で、実行可能なことを実行できる。

認可の判断は難しくなる。プロンプト指示、スキーマ選択、クエリ生成ロジック、DB権限、後処理にまたがって分散するためである。プロンプトインジェクションは、より広いクエリ、スキーマ開示、行数制限の迂回、タスクに不要な列の取得へモデルを誘導しうる。

ユーザー値をパラメータとしてバインドする場合、プリペアドステートメントは有効である。しかし、モデルがクエリ構造そのものを合成できる場合、中心的な問題は解決されない。正確性、認可、監査可能性は、リクエストごとに変動しうる生成物に依存する。

# 3. 比較対象のアクセスパターン（3/4）

## パターン3: 汎用CRUDまたはREST API

汎用CRUD APIは、テーブルまたはリソースをHTTP操作として公開する。エージェントはDB認証情報を持たず、通常は生SQLを送信できない。代わりに、APIサーフェス、リソース名、フィールド名、自身のAPIキーで利用可能な操作を見ることができる。

認可はAPI層で行われる。これはDB認証情報を直接渡すより改善であるが、汎用的なテーブル形状のAPIは広いデータ面を公開しがちである。エンドポイントがテーブルに強く対応している場合、モデルはレコード列挙、不要フィールドの要求、キーが許せば更新や削除操作を行える。

この方式は直接SQLより監査しやすいが、監査ログは業務意図ではなくテーブルやリソース中心になりやすい。

# 3. 比較対象のアクセスパターン（4/4）

## パターン4: エージェント定義ケイパビリティベースアクセス

ケイパビリティベースアーキテクチャでは、エージェントは `get_customer_order_status` や `list_open_invoices_for_account` のような名前付き業務操作を呼び出す。エージェントはDB認証情報、生SQLアクセス、スキーマ全体のCRUDアクセスを受け取らない。

選択された操作はREST APIとMCPを通じて公開されるが、公開される面はデータベース契約ではなくケイパビリティ契約である。ゲートウェイはルーティング、ID、ケイパビリティ認可、OpenAPIフィルタリング、MCPツールフィルタリング、可観測性を担う。プロバイダーのクラウドに業務データ、生SQL、DB認証情報を保存すべきではない。オンプレミスエージェントは、SQL、DB認証情報、パラメータ検証、実行ポリシー、プリペアドステートメント実行、出力フィルタリングを担う。`capability.yaml` が存在可能な操作を定義する。

AIエージェントが誤誘導された場合でも、呼び出せるのは可視化されたケイパビリティと受理されるパラメータに限られる。これにより実行可能な面が狭まり、監査記録を業務操作に対応づけやすくなる。ただし、この設計は正しいケイパビリティ定義、スコープされたAPIキー、慎重なログ設計、オンプレミスエージェントホストの安全な運用に依存する。

# 3. アーキテクチャ図

## 直接DBアクセス

```
[AI Agent]
    |
    v
[App / Agent Runtime] -- credentials, SQL, schema access --> [(Legacy Database)]
    ^
    |
 prompt injection can influence query intent
```

## ケイパビリティベースアクセス

```
[AI Agent / MCP Client]
    |
    v
[Gateway: auth, routing, observability]
    |
    | capability name + validated-looking params
    v
[On-Prem Agent: validation, policy, prepared SQL, output allow-list]
    |
    v
[(Legacy Database)]
```

# 4. 評価基準

| 評価軸 | 直接DBアクセス | SQL生成エージェント | 汎用CRUD API | ケイパビリティベースアクセス |
|---|---|---|---|---|
| 最小権限 | DBロールが非常に狭くない限り弱い | 弱から中。モデルがクエリ権限を持つ | 中。エンドポイントとキーのスコープ化は可能 | ケイパビリティが狭ければより強い |
| プロンプトインジェクション時の影響範囲 | 高い | 高い | 中から高 | 低め。許可ケイパビリティに限定 |
| SQLインジェクション / クエリ操作耐性 | クエリ構築に依存 | 生成クエリ構造が大きな懸念 | 生SQLエンドポイントがなければ低め | プリペアドSQLと生SQL非公開で低め |
| スキーマ露出 | 高い | 高い | 中 | 低い。ケイパビリティメタデータのみ |
| データ最小化 | 弱い | 弱から中 | 中 | result allow-list により強化可能 |
| 出力フィルタリング | 通常は外部 | 多くは後処理 | API固有 | エージェントが allow-list を適用 |
| 監査可能性 | クエリ単位で技術的になりがち | 生成クエリ単位 | エンドポイント単位 | ケイパビリティ名単位 |
| 失効可能性 | DB認証情報 / ロールのローテーション | DB認証情報 / ロールのローテーション | エンドポイント/APIキー失効 | APIキーのケイパビリティ許可を失効 |
| 運用複雑性 | 初期は低いがリスク高 | 高い | 中 | 中から高 |
| レガシー環境適合性 | 危険な公開になりやすい | 危険な公開になりやすい | APIモデリングが必要 | 読み取り中心の業務操作に強い |
| MCP / ツール呼び出し適合性 | 低い | リスク高 | 利用可能 | 名前付きツールに強い |

# 4. パターン比較

| パターン | AIに露出する権限 | スキーマ露出 | 任意SQL実行 | 出力最小化 | 監査粒度 | 主なリスク |
|---|---|---|---|---|---|---|
| 直接DB認証情報 | DBロール権限 | 高 | ロール範囲内で可 | 手動制約がなければ弱い | SQL / セッションログ | プロンプトインジェクションがDB悪用になる |
| SQL生成エージェント | クエリ生成権限とDBロール | 高 | 生成SQL経由で可 | 弱から中 | プロンプトと生成SQL | 誤った、または広すぎるSQLが認可済み操作になる |
| 汎用CRUD API | エンドポイントとリソース権限 | 中 | 公開しなければ不可 | 中 | エンドポイント / リソースログ | テーブル形状APIが広いレコードとフィールドを露出 |
| ケイパビリティベース | 名前付き業務操作権限 | 低 | 生SQLインターフェースなし | result allow-list により強化 | ケイパビリティ名、キー、ステータス、レイテンシ | 設計不良のケイパビリティが広いアクセスを再現 |

# 5. 評価方法

この評価は例示的な脅威モデルシミュレーションであり、形式的な証明ではない。顧客と注文を含む小さなレガシーデータベース領域を想定し、10個の攻撃または失敗シナリオに各アクセス方式がどう反応するかを評価する。

評価モデルのソースは、このPDFと同時に公開する `src.zip` に含める。

- `evaluation-model.json` は、サンプル領域、前提、パターン、シナリオ、定性的結果、シナリオメモを定義する。
- `evaluate-scenarios.mjs` は、そのモデルから `scenario-outcomes.md` を生成する。

この方法は性能ベンチマークではなく、権限と失敗モードを評価する。目的は、セキュリティ上のトレードオフを議論可能な程度に具体化することであり、特定のアーキテクチャを認証することではない。

# 5. 評価シナリオ

| ID | シナリオ |
|---|---|
| S1 | プロンプトインジェクションが全顧客一覧を要求する。 |
| S2 | プロンプトインジェクションがデータベーススキーマの開示を要求する。 |
| S3 | プロンプトインジェクションが任意SQLの実行を要求する。 |
| S4 | プロンプトインジェクションが業務タスクに不要な列の取得を要求する。 |
| S5 | プロンプトインジェクションが行数制限の迂回を要求する。 |
| S6 | プロンプトインジェクションがデータ更新または削除を要求する。 |
| S7 | APIキーまたはエージェント実行環境が侵害される。 |
| S8 | ゲートウェイまたはAPI層が侵害される。 |
| S9 | ツールカタログまたはOpenAPI定義が露出する。 |
| S10 | 正当なユーザーが有効なパラメータで許可済み業務操作を要求する。 |

定性的ラベルとして、`High risk`、`Medium risk`、`Lower risk`、`Blocked by design`、`Depends on implementation`、`Allowed` を用いる。

# 6. 評価結果: シナリオ結果

| シナリオ | 直接DBアクセス | SQL生成エージェント | 汎用CRUD API | ケイパビリティベースアクセス |
|---|---|---|---|---|
| S1: 全顧客一覧 | High risk | High risk | Medium risk | Lower risk |
| S2: スキーマ開示 | High risk | High risk | Medium risk | Blocked by design |
| S3: 任意SQL | High risk | High risk | Blocked by design | Blocked by design |
| S4: 不要列 | High risk | High risk | Medium risk | Lower risk |
| S5: 行数制限迂回 | High risk | High risk | Depends on implementation | Lower risk |
| S6: 更新 / 削除 | High risk | High risk | High risk | Blocked by design |
| S7: APIキー / 実行環境侵害 | High risk | High risk | Medium risk | Medium risk |
| S8: ゲートウェイ / API侵害 | High risk | High risk | High risk | Medium risk |
| S9: ツールカタログ露出 | High risk | High risk | Medium risk | Lower risk |
| S10: 許可済み操作 | Allowed | Allowed with correctness risk | Allowed | Allowed |

ケイパビリティベース方式は、多くのプロンプトインジェクションシナリオでリスクを下げる。モデルが実行可能SQLを発明したり、実行時にデータベースインターフェースを拡張したりできないためである。残るリスクは、ケイパビリティ定義、APIキースコープ、出力フィルタリング、エージェントホストの安全性に集中する。

# 6. 評価結果: リスク比較

| リスク領域 | 直接DBアクセス | SQL生成エージェント | 汎用CRUD API | ケイパビリティベースアクセス |
|---|---|---|---|---|
| 機密性侵害 | 高 | 高 | 中 | 出力が狭ければ低め |
| 完全性侵害 | 書き込みロールがあれば高 | 書き込みロールがあれば高 | 書き込みエンドポイントがあれば高 | 読み取り専用範囲では低め |
| プロンプトインジェクション悪用 | 高 | 高 | 中 | 低め |
| スキーマ偵察 | 高 | 高 | 中 | 低め |
| 認証情報露出 | AIアプリ経路にDB認証情報 | AIアプリ経路にDB認証情報 | API認証情報のみ | ゲートウェイにAPI認証情報、オンプレエージェントにDB認証情報 |
| 監査の曖昧さ | 高 | 高 | 中 | 低め |
| 運用負荷 | DBロール管理 | モデル/クエリガードレールとDBロール | API認可設計 | ケイパビリティレビューとエージェント運用 |

ケイパビリティベースアクセスは、露出する権限を減らし、実行可能な面を狭め、影響範囲を限定し、監査可能性を改善し、ポリシー適用をより明示的にしうる。これは条件付きの主張である。狭いケイパビリティ設計、レビュー済みの `capability.yaml` 変更、保守的なAPIキー付与、機密業務データを避けるログ設計に依存する。

# 6. 評価結果: リスクの移動

| リスク | 直接アクセスで現れる場所 | ケイパビリティ方式で現れる場所 | 必要な残余統制 |
|---|---|---|---|
| DB認証情報の悪用 | AIアプリ、実行環境、シークレットストア、ログ | オンプレエージェントホストとローカル設定 | ホスト堅牢化、シークレット管理、最小権限DBユーザー |
| クエリ操作 | 実行時生成SQLまたは直接ユーザー駆動SQL | デプロイ前にレビューされたケイパビリティSQL | コードレビュー、SQL lint、読み取り専用ポリシー、プリペアドステートメント |
| 過剰なデータ開示 | 広いテーブル、結合、列 | 広すぎるケイパビリティ結果セット | 出力 allow-list レビュー、最大行数 / 最大バイト数 |
| 未認可操作 | DBロールが広すぎる | APIキーが多すぎるケイパビリティを許可 | ケイパビリティ単位のAPIキースコープと失効 |
| メタデータ露出 | スキーマ名とテーブル名 | ケイパビリティカタログと公開説明 | OpenAPI/MCP説明に秘密や内部情報を入れない |
| ゲートウェイ侵害 | DBパスや認証情報を露出しうる | ルーティング/認可に影響するがSQLとDB認証情報は持たない | エージェント側 enforcement、監視、キー rotation |

重要な変化は、生のデータベース権限がAI実行環境とゲートウェイから離れることである。セキュリティレビューの焦点は、ケイパビリティ定義、APIキー付与、オンプレエージェント実行境界へ移る。

# 7. 考察（1/2）

ケイパビリティベースアクセスは、アドホックSQLより柔軟性が低い。運用用途ではそれ自体が目的だが、実際のトレードオフでもある。柔軟な探索、多数テーブルにまたがる広い分析結合、一回限りの調査が必要な分析担当者には、管理されたデータウェアハウス、セマンティックレイヤー、BI環境の方が適する場合がある。主用途が大量の履歴データに対するバッチ分析や機械学習である場合も、別方式がより適することがある。

設計の悪いケイパビリティは、依然として過剰なデータを露出しうる。広い列と高い行数制限を持つ `export_all_customers` のようなケイパビリティは、技術的にはケイパビリティであっても危険である。ケイパビリティ集合が広がりすぎると、実質的にデータベースアクセスを再現してしまう。出力 allow-list の定義が不適切な場合も失敗しうるし、レビューされていないケイパビリティ変更は新しいリスクを導入しうる。

読み取り専用範囲は破壊的リスクを実質的に下げるが、機密性リスクを取り除くわけではない。読み取り専用エージェントでも、不要列を返すケイパビリティ、過剰権限のAPIキー、業務データを含むログ、侵害されたオンプレエージェントホストにより、機密データを漏えいしうる。

# 7. 考察（2/2）

ゲートウェイ侵害とエージェント侵害の意味は異なる。評価したアーキテクチャでは、ゲートウェイは生SQLやDB認証情報を知らない前提である。これは、ゲートウェイ侵害が直接開示できるものを制限する。しかし、侵害されたゲートウェイは、エージェント側の enforcement と監視が強くなければ、ルーティング、認可、可観測性、キャッシュ済みメタデータを悪用しうる。

オンプレエージェントと `capability.yaml` は重要なセキュリティ境界である。エージェントはDB認証情報、SQL実行、検証、ポリシー、出力フィルタリングを担う。したがって、ケイパビリティ変更はアプリケーション認可コードと同じレビュー規律を必要とする。

ケイパビリティベースアクセスは、操作が明示的であるため、運用AIエージェントでは通常ガバナンスしやすい。ただし、安全なDBロール、ネットワーク制御、シークレット管理、監視、インシデント対応の代替ではない。

# 8. 実務上の示唆（1/2）

レガシーシステムへAIエージェントを接続する企業は、読み取り専用操作から始め、データベーステーブルではなく業務意図を中心にケイパビリティを定義すべきである。良いケイパビリティ名は、クエリがたまたま読むテーブルではなく、組織が公開を許容する業務操作を表す。

入力検証、実行ポリシー、プリペアドSQL、出力 allow-list、行数制限、バイト数制限、タイムアウトを用いる。APIキーは許可されたケイパビリティ単位でスコープする。AI実行環境やゲートウェイに対して、生SQL、生スキーマアクセス、DB認証情報を公開しない。

OpenAPIとMCPツール定義は、認可スコープに基づいてフィルタされるべきである。ツール説明は許可された操作の公開契約テキストであり、秘密、内部インシデントメモ、DB認証情報、機密運用情報を含めるべきではない。

# 8. 実務上の示唆（2/2）

運用テレメトリは、誰がどのケイパビリティを呼び、成功したか、どのエラーコードが返ったか、どれだけ時間がかかったかを答えられるべきである。ログには、ケイパビリティ名、APIキー識別子、ステータス、エラーコード、リクエストID、レイテンシを含める。

ログには、業務データ、生SQL、送信されたパラメータ値、DB接続情報、認証情報、Authorizationヘッダー、トークン、APIキー、スタックトレース、DBドライバー詳細を保存すべきではない。必要な機密診断情報は、厳格に管理されたローカルエージェントログに留める。

ケイパビリティ定義はセキュリティ上重要な設定として扱う。変更をレビューし、ケイパビリティ集合を小さく保ち、読み取りケイパビリティには読み取り専用DBユーザーを優先し、アクセス要件が変わったらAPIキーをローテーションまたは失効する。

# 9. 制限事項と今後の課題

この評価は例示であり、完全なセキュリティ監査、形式的な脅威モデル、レッドチーム評価の代替ではない。すべてのデータベースエンジン、ドライバー、ORM、ネットワークトポロジ、デプロイモデル、MCPクライアント、エージェントフレームワークを網羅していない。

この比較は、ケイパビリティ定義がレビューされ正しく実装されていることを前提としている。主に読み取り中心のアクセスに焦点を当てている。書き込み操作には、追加の統制、承認フロー、冪等性、トランザクション設計、競合処理、より強い監査証跡、明示的な復旧手順が必要である。

性能とレイテンシは深くベンチマークしていない。変更承認、インシデント対応、アクセスレビュー、キー rotation 規律などの人的運用プロセスも十分には評価していない。

今後の課題として、実際のMCPクライアント、複数のデータベースエンジン、形式的脅威モデリング、敵対的プロンプトテスト、レッドチーム演習、DLP連携、書き込み操作の安全パターンを評価すべきである。

# 10. 参考文献


1. Model Context Protocol, "Tools", https://modelcontextprotocol.io/specification/2025-06-18/server/tools
2. Model Context Protocol, "Authorization", https://modelcontextprotocol.io/specification/2025-06-18/basic/authorization
3. OWASP, "Top 10 for Large Language Model Applications", https://owasp.org/www-project-top-10-for-large-language-model-applications/
4. OWASP Cheat Sheet Series, "SQL Injection Prevention Cheat Sheet", https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
5. OWASP Cheat Sheet Series, "Injection Prevention Cheat Sheet", https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html
6. NIST, "SP 800-207: Zero Trust Architecture", https://csrc.nist.gov/pubs/sp/800/207/final
7. OpenAPI Initiative, "The OpenAPI Specification Explained", https://learn.openapis.org/specification/
8. OpenAPI Initiative, "API Endpoints", https://learn.openapis.org/specification/paths.html
9. PostgreSQL Documentation, "Privileges", https://www.postgresql.org/docs/17/ddl-priv.html
10. PostgreSQL Documentation, "Row Security Policies", https://www.postgresql.org/docs/17/ddl-rowsecurity.html

# 最後に

この比較は、ケイパビリティベースアクセスが絶対的に安全であることを示すものではない。レガシーデータベースに対する運用AIアクセスについて、より狭く監査しやすい権限モデルを示している。

この設計は、ケイパビリティが少数で、明示的で、読み取り中心で、パラメータ化され、出力フィルタリングされ、セキュリティ上重要な設定としてレビューされる場合に最も強くなる。

発行: 合同会社ビューレガシー
著者: 増田朝陽
発行日: 2026年5月19日