社における情報管理の問題点と改善策

ヒアリング調査

効果的な情報管理と業務効率向上のために各部門でヒアリング調査を行い問題点を調査分析を行った

はじめに

我社における情報管理の問題点は、多岐にわたります。これらの問題点は、会社の業務効率やサービスの品質に直接的な影響を与えるため、

早急に改善策を講じる必要があります。本稿では、主な問題点を詳しく分析し、それぞれに対する具体的な改善策を提案します。

各部署間で情報が一元管理できていない問題点

1. **上下番の連絡方法の多様性**:

現在、我社において、隊員が現場に到着した際の上番報告や、業務を終了した際の下番報告の連絡方法として、スマホ、LINE、電話など多様な手段が使用されています。

これらの多様な連絡手段は、情報の散逸や見落としを引き起こす原因となっています。

連絡方法が統一されていないため、重要な情報が適切に伝達されないことがあります。

連絡手段が多岐にわたることで、情報の管理が煩雑になることがあります。

2. **急な欠勤による調整不足**:

急な欠勤が発生した際に、代替要員の手配やシフト調整が適切に行われないことがあります。

代替要員の情報が適切に管理されていないため、迅速な手配が困難です。

欠勤情報が総務部門に適切に伝達されないことがあります。

3. **情報のクオリティー不足**:

上下番情報が不完全や不正確であるため、総務部門での処理が遅れることがあります。

情報の二重入力やミスが発生しやすいです。

情報の更新がリアルタイムで行われないため、最新の情報が共有されないことがあります。

4. 給与計算や請求処理のクオリティ確保の問題:

上下番情報が不完全や不正確であるため、給与計算や請求処理に支障が出ています。

情報の二重入力やミスが発生しやすく、これが給与計算や請求処理のクオリティに影響を与えます。

情報の更新がリアルタイムで行われないため、最新の情報が共有されず、給与計算や請求処理の正確性が損なわれることがあります。

5. エクセル管理による情報のクオリティ確保の問題:

エクセルでの管理は、情報の一元管理が難しく、データの整合性や正確性が確保されにくいです。

エクセルファイルのバージョン管理が難しく、最新の情報が共有されないことがあります。

エクセルの手動入力によるミスが発生しやすく、情報のクオリティが低下することがあります。

6. 24時間管理体制の課題:

担当社員ごとに管理方法が異なるため、情報の一貫性が保てないことがあります。

交代勤務の際に、情報の引き継ぎが不十分であることがあります。

管理方法の違いによって、サービスの質が変動することがあります。

7. 人員不足:

管理業務や現場打合せに十分な人員が確保できていないため、業務が滞ることがあります。

人員不足により、管理が行き届かず、サービスの質が低下することがあります。

一人の担当者に多くの業務が集中し、過重労働のリスクがあることがあります。

改善策

1. **連絡方法の統一**:

   主要な連絡方法を一つに絞ることで、情報の散逸を防ぐ。例えば、専用の業務連絡アプリを導入する。

   電話やメッセージの内容を自動的に記録・保存するシステムを導入することで、後からの確認が容易になる。

2. **シフト管理システムの導入**:

   急な欠勤が発生した際に、自動的に代替要員を手配できるシフト管理システムを導入する。

   あらかじめ代替要員のリストを整備し、迅速に手配できるようにする。

   総務部門や関係部署に対して、欠勤情報を迅速に伝達する仕組みを構築する。

3. **情報入力の標準化**:

   上下番情報の入力方法を標準化し、誤入力や漏れを防ぐ。

   情報がリアルタイムで更新され、共有されるシステムを導入する。

   総務部門と定期的に情報を確認し合い、不備や誤りを早期に発見・修正する。

4. **給与計算や請求処理のクオリティ向上**:

   上下番情報の入力方法を標準化し、誤入力や漏れを防ぐ。

   情報がリアルタイムで更新され、共有されるシステムを導入する。

   総務部門と定期的に情報を確認し合い、不備や誤りを早期に発見・修正する。

5. **エクセル管理の改善**:

   エクセル管理からMicrosoft AccessとMicrosoft Excelを連携させた統合管理システムに移行し、データの一元管理を実現する。

   バージョン管理が容易なシステムを導入し、最新の情報が常に共有されるようにする。

   手動入力のミスを減らすために、自動化されたデータ入力システムを導入する。

6. 標準業務手順書の作成:

24時間管理体制における標準業務手順書を作成し、全社員に徹底する。

交代勤務の際に、情報の引き継ぎを徹底し、重要な情報が漏れないようにする。

管理方法が適切に行われているかどうか、定期的に業務監査を実施する。

7. 業務分担の見直し:

業務分担を見直し、一人の担当者に業務が集中しないようにする。

外部の専門家や業務委託を利用し、人員不足を補う。

このように、各部署間で情報が一元管理できていない問題点を明確にし、それぞれに対する具体的な改善策を講じることで、業務効率の向上とサービスの質の向上に繋げる

1. システム仕様　　概要

システムはシンプルな構成で開発することを基本とするが、事前に想定できるリスクや課題にも対応可能な柔軟な計画を立てるものとする。

1.1 システムの目的

本システムは、我社（泉州警備保障株式会社）の業務統合管理を目的とし、Microsoft AccessとExcelを活用して各部門のデータを一元化することで、

部門間のスムーズな情報共有と業務効率の向上を図るものです。本システムを導入することで、以下のようなメリットが得られます。

データの一元管理

各部門が独立して管理していた情報を統合し、Microsoft Accessを用いたリアルタイムでのデータ共有を可能にします。

これにより、二重管理やデータ不整合を防ぎ、業務の正確性を向上させます。

業務プロセスの効率化

各種申請や報告の電子化を推進し、Microsoft AccessとExcelを組み合わせることで、業務の標準化を実現します。

従来、手作業で行っていた入力・確認作業をシステム化することで、業務負担を軽減し、ミスの発生を抑えます。

セキュリティの強化

重要情報の管理を徹底し、Microsoft Accessのアクセス権限管理機能を活用して、必要な人のみが適切なデータにアクセスできるようにします。

また、データの変更履歴を記録し、不正アクセスや誤操作によるデータ損失を防ぎます。

1.2 データ運用と拡張性

本システムは、最低3年間の運用データの蓄積を前提としており、Microsoft AccessとExcelでの運用を最適化しつつ、

将来的にSQL Serverへの移行も視野に入れた設計を行います。

Microsoft AccessとExcelの活用

Microsoft Access をメインのデータベースとして活用し、データの管理・検索・更新を行う。

Excel をレポート作成やデータ分析の補助ツールとして活用し、運用部門が柔軟にデータを活用できる仕組みを構築する。

AccessとExcelの連携機能を活用し、入力データのチェックや集計処理を効率化する。

SQL Server への移行を考慮

3年以上のデータが蓄積されることを想定し、Microsoft Accessでの運用を最適化。

大規模データ処理の効率化のため、SQL Serverへのスムーズな移行を可能とするデータ設計を行う。

履歴データの管理

履歴DBの構築 により、過去のデータを保管し、必要に応じて参照・検索できる仕組みを整備。

データ削除を極力回避し、過去の取引情報や業務記録を適切に保持。

長期間のデータ蓄積によるパフォーマンス劣化を防ぐため、定期的なデータアーカイブ処理を検討。

データ増加への対応

部門ごとのデータ量を考慮し、Microsoft Accessの最適な運用を行いつつ、将来的な拡張を想定したテーブル設計を実施。

必要に応じて 分割データベース や データ圧縮技術 の導入を検討。

負荷テスト基準:「同時10名が検索・登録を実行した際に、主要なクエリ（データベースからデータを取得・更新・削除するための命令）の平均応答時間を1秒以内とする」。

移行前後のテーブルのレコード総数、プライマリキー・外部キーの整合性、サンプル10%のランダムチェックを実施。

本システムは、Microsoft AccessとExcelを活用し、効率的かつ柔軟な業務管理を実現するとともに、将来的なSQL Serverへの移行を見据えた拡張性を確保するものです。

2. システム構成

2.1 運用環境

・Microsoft Access（分散管理）を利用。

・部門ごとに **accdb（Accessのデータベースファイル）** を分割し、管理を最適化。

・SQL Server への移行を視野に入れ、リンクテーブルの管理を強化。

2.2 SQL Server 移行のタイムライン

年数	実施内容
1年目	Access の最適化・運用開始、負荷テスト実施（最大10名）
2年目	SQL Server のテスト環境を構築し、勤務記録データを移行、履歴検索の動作確認
3年目	人事・契約情報の SQL Server 移行
4年目	Access と SQL Server のハイブリッド運用を6ヶ月実施し、移行のリスクを低減

2.3負荷テストのシナリオ詳細

単純検索（10名が同時検索）

データ更新（5名が検索、5名が更新）

テーブル競合（複数のユーザーが異なるテーブルを操作）

3. データ管理

3.1 データ構造

・SQL Server 移行後、履歴専用テーブルを導入し、データの整合性を維持。

・削除時のログ管理を強化し、削除前後のデータを記録。

3.2 データ競合防止策

「更新専用フォーム」を設計し、直接編集を制限。

「確定」ボタン を押してデータを登録し、変更前・変更後のログを記録。

「同一レコードの同時編集禁止」機能を導入し、警告を表示。

3.3 リレーションシップとリアル値の使い分け

本システムでは、データ管理の効率化と柔軟性を両立するために、リレーションシップとリアル値の使い分けを明確に定めています。

1 リレーションシップを利用するデータ

変更の確率が低いマスタデータ（例：部門マスタ、職種マスタ、役職マスタなど）については、リレーションシップを用いて管理します。

理由：

データの一貫性を確保できる

更新頻度が低く、IDによる管理が有効

マスタIDを参照することで冗長なデータを削減

2 リアル値をコピーするデータ

変更の可能性が高いデータ（例：顧客名、取引先名、施設名など）については、登録時にリアル値をコピーする方式を採用します。

理由：

過去データの整合性を保持するため

マスタIDのみで管理すると、マスタ変更時に過去の取引履歴が意図せず更新されるリスクがある

変更前後の名称を履歴として保持することで、業務上の追跡が容易になる

3 使い分けの方針

基本ルール:

変更が少ない情報 → リレーションシップを活用

変更が多い情報 → リアル値をコピー

運用のポイント:

リアル値をコピーした場合でも、マスタを照会する機能を提供し、最新情報との比較が可能な設計にする

リレーションシップを利用するマスタデータの更新が必要な場合は、影響範囲を考慮した上で慎重に対応する

この方式により、データの一貫性と運用の柔軟性を確保し、変更管理を容易にします。

4 システム構成

1. システム構成図(データフローの視覚化)

現在のフロー図(Access ↔ Excel ↔ SQL Server)を追加

データの流れ(登録 → 管理 → 検索 → 出力)

2. 使用技術の詳細

Access → フォーム・クエリ・テーブルの役割

Excel → 帳票出力、データ分析

SQL Server(将来的移行) → クエリ最適化、負荷テスト基準

ファイルサーバー → どのデータを保存するか

3. データの流れ

顧客データの登録 → 契約情報の紐づけ → スケジュール作成

給与計算との連携

警備スケジュール → 変更履歴の管理

追記位置:「4.1 システムの全体像」セクション

5. マルチユーザー環境

5.1 同時アクセス想定

Access 環境：最大10名（負荷テスト後に調整）。

SQL Server 移行後：最大50名（将来的には100名以上も想定し、負荷テスト実施）。

5.2 同時編集時のデータ競合対策

楽観ロック方式（データベースの同時更新による競合を防ぐ方法の一つ）を採用し、他ユーザーの変更が検出された場合は再入力を促す

「ロック解除を待つ」または「別のレコードに切り替える」対応を選択可能にする。

6. データ改ざん・削除リスクの対策

6.1 直接編集によるリスク回避

「確定」ボタン を押さない限りデータが本登録されない仕様。

ログ管理を強化 し、管理者がデータ操作履歴を追跡可能にする。

削除時のログを記録し、「削除申請→管理者確認→ログ記録→削除実行」 のプロセスを適用。

6.2 削除ルールの明確化

削除前にバックアップを取得し、90日間保管する」とすることで、誤削除時のリカバリー対応を強化。

削除後も30日間は復元可能な設計とする。

6.3データ削除の「復元30日間」の方法

削除データは「削除ログテーブル」に記録し、管理者専用ツール（VBAスクリプト）を利用して復元可能にしました。

削除後30日以内なら、管理者が削除ログから直接復元可能。30日を超えた場合はバックアップデータから復元。

7. セキュリティ仕様

7.1 アクセス制御

VBA（Access でプログラムを動作させる機能）を活用し、起動時にユーザー認証を実行。

管理者用の緊急アクセスを追加し、認証システム異常時の対応策を明記。

8. SQL Server への移行計画

各移行段階ごとに **チェックリスト** を策定し、データ移行の品質を確保。 各移行フェーズで負荷テストを実施（最大50名の同時アクセスを想定）。

SQL Server 移行後のパフォーマンス監視

移行後6か月間は週1回のレスポンスチェック、その後は月1回のパフォーマンス分析を実施。

レスポンス時間が2秒以上のクエリを抽出し、最適化。

管理者向けのダッシュボードを設置し、リアルタイム監視を可能に。

9. Access × Excel 連携で業務を効率化

9.1 データ抽出の最適化

検索フォーム で条件を指定し、「データ抽出ボタン」を押すと Excel に自動出力。

氏名・住所・電話番号・契約情報を含むデータのExcel出力を禁止し、例外的に役員承認があれば可能

9.2 Excel 連携の制限

Excel出力ログ を監査可能にし、不正出力を防止。

Excel へのデータ転送時、改変を防止するための保護機能を実装。

10. 今後のスケジュール

実施内容	期間
Access 分散運用開始、データ最適化	1年目
SQL Server のテスト運用開始	2年目
人事・契約情報の SQL Server 移行	3年目
SQL Server で本運用開始	4年目

1. システム仕様　　詳細

1.1 システムの目的

本システムは、警備保障会社の業務統合管理を目的とし、Microsoft Access（データ管理を行うソフトウェア）とExcel（表計算ソフト）を活用して各部門のデータを一元化することで、部門間のスムーズな情報共有と業務効率の向上を図るものです。本システムを導入することで、以下のようなメリットが得られます。

データの一元管理（複数の部門のデータを統合し、整合性を向上させること）

各部門が独立して管理していた情報を統合し、Microsoft Accessを用いたリアルタイム（即時反映）でのデータ共有を可能にします。

これにより、二重管理やデータ不整合（データのズレやミス）を防ぎ、業務の正確性を向上させます。

業務プロセスの効率化

各種申請や報告の電子化（ペーパーレス化）を推進し、Microsoft AccessとExcelを組み合わせることで、業務の標準化を実現します。

従来、手作業で行っていた入力・確認作業をシステム化することで、業務負担を軽減し、ミスの発生を抑えます。

セキュリティの強化

重要情報の管理を徹底し、Microsoft Accessのアクセス権限管理機能（データを誰が操作できるか制限する機能）を活用して、

必要な人のみが適切なデータにアクセスできるようにします。また、データの変更履歴（過去の修正記録）を記録し、不正アクセスや誤操作によるデータ損失を防ぎます。

1.2 データ運用と拡張性

本システムは、最低3年間の運用データの蓄積を前提としており、Microsoft AccessとExcelでの運用を最適化しつつ、

将来的にSQL Server（Microsoftの大規模データ管理システム）への移行も視野に入れた設計を行います。

Microsoft AccessとExcelの活用

Microsoft Access をメインのデータベースとして活用し、データの管理・検索・更新を行う。

Excel をレポート作成やデータ分析の補助ツールとして活用し、運用部門が柔軟にデータを活用できる仕組みを構築する。

AccessとExcelの連携機能を活用し、入力データのチェックや集計処理を効率化する。

SQL Server への移行を考慮

3年以上のデータが蓄積されることを想定し、Microsoft Accessでの運用を最適化。

大規模データ処理の効率化のため、SQL Serverへのスムーズな移行を可能とするデータ設計を行う。

履歴データの管理

履歴DB（過去データを保存する仕組み）の構築 により、過去のデータを保管し、必要に応じて参照・検索できる仕組みを整備。

データ削除を極力回避し、過去の取引情報や業務記録を適切に保持。

長期間のデータ蓄積によるパフォーマンス劣化（処理速度の低下）を防ぐため、定期的なデータアーカイブ処理（古いデータを移動・整理する作業）を検討。

1.3.データ増加への対応

本システムは、データ量の増加に対応しつつ、業務運用に影響を与えないパフォーマンスを維持するため、

履歴データの適切な管理・負荷試験の実施・SQL Server（Microsoftのリレーショナルデータベース管理システム） へのスムーズな移行 を計画的に実施します。

1.3.1 履歴データ管理

1年以上前のデータは履歴DBへ移行し、必要に応じて検索可能な環境を整備。

運用中のデータと履歴データを分離することで、メインDBのパフォーマンスを維持。

履歴DBの検索インターフェースを設け、適切なユーザー権限管理を行う。

履歴データ移行方式

1. 定期的なバッチ処理で移行を実施

月次で移行を実行し、過去データを履歴DBへ保存。

移行時はデータの整合性をチェックし、エラーが発生した場合はログに記録。

2. 履歴DBの検索最適化

必要なインデックスを設定し、検索クエリの負荷を軽減。

履歴データの取得速度を 3 秒以内に維持するようチューニング。

1.3.2 負荷試験の実施

将来的な SQL Server（Microsoftのリレーショナルデータベース管理システム） への移行 を視野に入れ、以下の 負荷テスト（システムの性能評価テスト） を実施。

負荷テスト基準

同時10名が検索・登録を実行 した際に、主要なクエリ（データ取得・更新・削除）の平均応答時間を1秒以内に抑えることを目標とする。

主要クエリには、以下の業務処理を含む：

検索処理：警備計画、勤務履歴、請求情報の検索

登録処理 ： 新規警備契約、シフト変更、請求書発行

更新処理 ： 警備員の勤務記録の修正、契約情報の変更

削除処理 ： 不要なデータの整理（※削除は限定的に実施）

負荷テストの測定方法

シミュレーションツールを使用 し、実際のユーザー操作を模倣した同時アクセスを検証。

クエリごとの応答時間を測定 し、1秒以内に収まらない場合はインデックス調整やクエリ最適化を実施。

CPU使用率、メモリ消費量、ディスクI/Oを監視 し、リソース負荷を分析。

1.3.3 SQL Server（Microsoftのリレーショナルデータベース管理システム） への移行計画

SQL Server への移行を円滑に進めるため、以下のデータ移行試験 を段階的に実施。

データ移行チェックポイント

移行前後のテーブルのレコード総数が一致することを検証

プライマリキー・外部キーの整合性を確認

データのランダムサンプリング（全体の10%）を実施し、レコードの整合性を検証

履歴データを含めたクエリの実行結果が一致することをテスト

パフォーマンスの比較（Access vs SQL Server）

SQL Server移行の技術検討

クラウド型SQL Serverの選択肢を検討（Microsoft Azure SQL Database など）

ストアドプロシージャを活用し、クエリ処理の最適化

スケーラビリティを考慮したテーブル設計の見直し

データアーカイブ機能を導入し、古いデータを適切に管理

2. システム構成

2.1 運用環境

本システムは Microsoft Access（分散管理） を利用し、警備保障会社の業務統合管理（会社全体の業務データを統一して管理すること）を効率化します。

運用環境として以下の方式を採用し、各部門の業務負荷やデータ量に応じた適切な管理を実現します。

2.1.1 運用方式

1 分散管理方式（フロントエンド・バックエンド構成）

各部門ごとに Access のデータベースファイル（.accdb）を分割 し、管理を最適化。

メインのバックエンドデータベース（.accdb）をファイルサーバー上に配置 し、各部門がリンクテーブルを通じてデータを参照・更新する形式。

各ユーザーのPC上でフロントエンド（.accdb）を動作 させ、サーバー負荷を軽減。

メリット

同時アクセスの衝突を防止（各部門のアクセスを独立管理）

データ管理の柔軟性（必要に応じて特定部門のデータ構造を変更可能）

セキュリティの向上（不要なデータにアクセスできない設計）

課題

複数部門を横断するデータ分析の難易度が増す

→ 解決策：定期的なデータ統合処理を導入（バッチ処理またはリアルタイム同期）

2 SQL Server（Microsoftのリレーショナルデータベース管理システム） への移行を視野に入れた設計

Access の運用を最適化しつつ、将来的な SQL Server（Microsoftのリレーショナルデータベース管理システム） への移行を前提としたデータ管理方式を採用。

Access のリンクテーブルを活用し、SQL Server（Microsoftのリレーショナルデータベース管理システム） との統合が容易にできるよう管理を強化。

アクセス負荷を分散するため、重要データは SQL Server（Microsoftのリレーショナルデータベース管理システム） へ移行し、検索・分析用のクエリを最適化。

移行戦略

1. 段階的な SQL Server（Microsoftのリレーショナルデータベース管理システム）への移行

まずは大容量データ（履歴データやログ情報）を SQL Server（Microsoftのリレーショナルデータベース管理システム）へ移行。

その後、主要テーブルを SQL Server（Microsoftのリレーショナルデータベース管理システム）に統合し、Access からリンクテーブルで接続。

2. リンクテーブル管理の強化

外部データベースとの接続を厳格に管理 し、不要なテーブルリンクを制限。

ODBC での接続パフォーマンスを最適化（インデックスの適用、ビューの活用）。

3. データ統合・同期処理

SQL Server（Microsoftのリレーショナルデータベース管理システム）とのハイブリッド運用を実施

（一部のデータは Access、主要データは SQL Server（Microsoftのリレーショナルデータベース管理システム））。

Access 内の一時データを定期的に SQL Server（Microsoftのリレーショナルデータベース管理システム）へ同期 し、データの一貫性を保持。

2.1.2 運用環境の構成

項目	内容
フロントエンド	各ユーザーPC上の Access (.accdb)
バックエンド	ファイルサーバー上の Access (.accdb) or SQL Server（Microsoftのリレーショナルデータベース管理システム）
サーバー環境	Windows Server 2019 以降（SQL Server（Microsoftのリレーショナルデータベース管理システム）配置予定）
ユーザー数	最大10名が同時アクセス
データ保存期間	3年（履歴データは別DBへ移行）
アクセス管理	ユーザーごとに権限設定

2.2 SQL Server（Microsoftのリレーショナルデータベース管理システム）移行のタイムライン

本システムでは、Microsoft Access から SQL Server（Microsoftのリレーショナルデータベース管理システム）への移行を4年間かけて段階的に実施 し、

システムの安定運用を確保しながら移行リスクを最小限に抑える。

2.2.1タイムライン

年数	実施内容
1年目	Access の最適化・運用開始、負荷テスト実施（最大10名）
	Access のテーブル設計を見直し、SQL Server（Microsoftのリレーショナルデータベース管理システム）移行を考慮したデータ構造へ最適化
	負荷テスト（同時10名の検索・登録・更新処理を実施し、レスポンスタイム 1秒以内を目標）
	データクレンジングの実施（重複データや不整合の解消）
	Access の運用ルール確立（データ入力基準、定期バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）の設定）
2年目	SQL Server（Microsoftのリレーショナルデータベース管理システム）のテスト環境構築、勤務記録データの移行、履歴検索の動作確認
	SQL Server（Microsoftのリレーショナルデータベース管理システム）のテスト環境を構築（ODBC接続の検証、セキュリティ設定）
	勤務記録データの移行（履歴データ約3年分）を試験的に実施
	履歴検索クエリの最適化（インデックス作成・検索速度向上）
	データ整合性チェック（Access との比較検証）
3年目	人事・契約情報の SQL Server（Microsoftのリレーショナルデータベース管理システム）移行
	人事・契約情報の主要テーブルを SQL Server（Microsoftのリレーショナルデータベース管理システム）へ移行
	SQL Server（Microsoftのリレーショナルデータベース管理システム）のパフォーマンス監視と負荷分散の調整
	Access から SQL Server（Microsoftのリレーショナルデータベース管理システム）へのリンクテーブル運用開始（ODBC 経由での接続最適化）
	Access 上のクエリを SQL Server（Microsoftのリレーショナルデータベース管理システム）のストアドプロシージャに移行

4年目	Access と SQL Server（Microsoftのリレーショナルデータベース管理システム）のハイブリッド運用を6ヶ月実施、移行リスクの低減
	主要データは SQL Server（Microsoftのリレーショナルデータベース管理システム）で運用し、Access はフロントエンドとして継続
	6ヶ月間の運用テストを実施し、SQL Server（Microsoftのリレーショナルデータベース管理システム）への完全移行を検討
	Access の役割を最小化し、SQL Server（Microsoftのリレーショナルデータベース管理システム）をメインDBとする構成へ移行
	バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）・障害対策の確立（定期的なリストアテストを実施）

2.2.2 移行リスクと対策

SQL Server（Microsoftのリレーショナルデータベース管理システム）への移行に伴うリスクと、それに対する対策を以下の通り整理する。

リスク	対策
データ整合性の欠如	データ移行前後の整合性チェックを徹底 （総レコード数、プライマリキー・外部キーのチェック）
SQL Server（Microsoftのリレーショナルデータベース管理システム）の負荷増加	インデックス最適化、ストアドプロシージャの活用、負荷分散の実施
Access との接続トラブル	リンクテーブルの運用テストを実施、ODBC接続の安定化
ユーザー操作の混乱	運用マニュアル作成、定期的な操作トレーニングの実施

2.3 負荷テストのシナリオ詳細

本システムでは、同時アクセスによるパフォーマンス評価を目的とした負荷テスト を実施し、データ検索・更新・競合時の応答速度を測定します。

特に、SQL Server（Microsoftのリレーショナルデータベース管理システム）への移行を視野に入れたパフォーマンステスト を行い、システムのスケーラビリティを確認します。

2.3.1 負荷テストの目的

同時アクセス時のレスポンスタイムを測定し、検索・更新処理の最適化 を図る。

競合発生時のデータ整合性を確認 し、同時更新時のエラーを検証。

将来的な SQL Server（Microsoftのリレーショナルデータベース管理システム）への移行を見据えた負荷許容範囲の把握。

2.3.2 負荷テストシナリオ

シナリオ	内容	計測項目	許容範囲
①単純検索テスト	10名のユーザーが同時に特定の 検索クエリを実行	検索応答時間	1秒以内
②データ更新テスト	5名が検索、5名がデータを同時に更新	更新処理時間、データ競合発生率	1.5秒以内
③テーブル競合テスト	複数のユーザーが異なるテーブルに 対して、同時に検索・登録・更新 を実行	応答時間、デッドロック発生頻度	2秒以内、デッドロック 発生率5%未満
④大量データ処理テスト	1万件以上のレコードを検索・更新	クエリの処理時間、CPU負荷	3秒以内、CPU使用率70%未満
⑤同時ログインテスト	10名が同時ログインし、異なる操作を実施	ログイン時間、認証エラー発生率	3秒以内、認証エラーなし

2.3.3 負荷テストの実施方法

1 シミュレーションツールの使用

JMeter または Access VBA マクロを利用 し、疑似的な同時アクセス環境を作成。

実際の業務データを使用 し、リアルな負荷状態を再現。

2 テスト環境の設定

テスト用データセット：50万件のレコード

テスト対象テーブル：勤務記録、契約データ、警備シフト情報

ネットワーク環境：社内LAN（1000Mbps）

サーバースペック：Core i7 / 16GB RAM / SSD 1TB

3 パフォーマンス測定

CPU負荷、メモリ使用率、ディスクI/Oを監視

エラーログを記録し、異常が発生した場合の原因を特定

負荷分散対策として、クエリ最適化とインデックス適用を検討

2.3.4 負荷テストの合格基準

SQL クエリの応答時間が目標値以内であること

データ競合（デッドロック）発生率が 5% 以下であること

10名同時アクセス時でもシステムが安定稼働すること

SQL Server（Microsoftのリレーショナルデータベース管理システム） 移行後も同等以上のパフォーマンスを維持できること

3. データ管理

3.1 データ構造

本システムでは、SQL Server（Microsoftのリレーショナルデータベース管理システム） 移行後のデータ整合性を維持し、

データの一貫性・保全性を強化するために、以下の設計を採用します。

3.1.1 履歴専用テーブルの導入

目的

主要な業務データの変更履歴を保持し、監査証跡（Audit Trail） を確保。

データの削除・更新が行われた際、過去データを履歴テーブルに自動保存 し、必要に応じて参照・復元が可能。

履歴DBを分離 し、メインデータベースのパフォーマンスを最適化。

対象テーブル

契約情報	顧客データ	シフト管理データ
勤務記録	警備員情報	請求履歴

履歴テーブルの設計例

```
CREATE TABLE Contract_History (
    HistoryID INT IDENTITY(1,1) PRIMARY KEY,
    ContractID INT NOT NULL,  -元の契約ID
    OperationType VARCHAR(10) NOT NULL,  -INSERT / UPDATE / DELETE
    ChangedBy NVARCHAR(50) NOT NULL,  -変更者
    ChangedAt DATETIME DEFAULT GETDATE(),  -変更日時
    OldData NVARCHAR(MAX),  -変更前データ（JSON形式）
    NewData NVARCHAR(MAX)  -変更後データ（JSON形式）
);
```

OldData / NewData に JSON 形式でデータを保存し、変更内容を完全記録。　　変更履歴は契約IDごとに管理し、過去のデータを素早く参照可能。

3.1.2 削除時のログ管理

削除データの完全保存を実施し、不正削除を防止

削除されたデータは 即時削除せず、論理削除（フラグ管理） を適用

IsDeleted フラグ（0 = 有効, 1 = 削除済み）を付与し、検索時に除外

実際の物理削除は 定期的なアーカイブ処理 により実施

削除ログを別テーブルに保存し、削除前後のデータを記録

削除ログの設計

```
    LogID INT IDENTITY(1,1) PRIMARY KEY,
    TableName NVARCHAR(50) NOT NULL,  -削除対象のテーブル名
    RecordID INT NOT NULL,  -削除されたレコードID
    DeletedBy NVARCHAR(50) NOT NULL,  -削除実行者
    DeletedAt DATETIME DEFAULT GETDATE(),  -削除実行日時
    DeletedData NVARCHAR(MAX)  -削除されたデータ（JSON 形式）
);
```

削除データを JSON 形式で記録し、復元可能な状態を維持

削除操作は管理者のみ実行できるように制限

一定期間（例: 1年）経過後、削除ログを履歴DBにアーカイブ

3.1.3 データ整合性維持のための対策

課題	解決策
データの不整合リスク	トランザクション処理※を適用 し、一貫性を保証 ※一連のデータベース処理を「ひとまとまりの操作」として扱う仕組み
履歴データが増加しすぎる	圧縮技術・アーカイブ機能を導入（1年以上のデータは履歴DBへ）
削除データの復元要求	削除データをログとして保持し、管理者が復元可能にする
SQL Server（Microsoftのリレーショナルデータベース管理システム） への移行後のデータアクセス遅延	インデックス最適化・検索クエリのチューニングを実施

3.2 データ競合防止策

本システムでは、データの同時編集による競合（データの上書きや不整合）の発生を防止するため、以下の対策を導入 します。

3.2.1 更新専用フォームの導入

1 直接編集の制限

テーブルを直接編集する機能を無効化 し、すべてのデータ更新は 専用フォーム経由でのみ実行可能 にする。

データ変更時は必ず「確定」ボタンを押して反映（即時反映ではなく、明示的な承認ステップを設ける）。

フォームでは、編集中のデータをロック し、他ユーザーの同時編集を防止。

2 更新履歴の記録

データ更新前後の情報をログに保存 し、変更履歴を追跡可能にする。

変更内容をJSON形式で保存 し、前回の値と現在の値を比較可能にする。

更新履歴テーブルの例

```
CREATE TABLE Update_Log (
    LogID INT IDENTITY(1,1) PRIMARY KEY,
    TableName NVARCHAR(50) NOT NULL,  -更新対象のテーブル名
    RecordID INT NOT NULL,  -更新されたレコードID
    UpdatedBy NVARCHAR(50) NOT NULL,  -更新者
    UpdatedAt DATETIME DEFAULT GETDATE(),  -更新日時
    OldData NVARCHAR(MAX),  -変更前データ（JSON形式）
    NewData NVARCHAR(MAX)  -変更後データ（JSON形式）
);
```

3.2.2 「確定」ボタンの導入

1 変更内容の確定手順

「確定」ボタンを押さない限り、データは保存されない 仕組みを導入。

確定時に、データの変更前・変更後のログを記録 し、追跡可能にする。

確定前に「プレビュー」機能を設け、変更内容を事前確認できるようにする。

2 同時編集時の競合回避

確定時に、対象レコードの最終更新時間をチェック し、他のユーザーが変更済みの場合は警告を表示。

変更が競合する場合は「手動マージ」機能を提供（ユーザーが変更内容を比較し、選択できる仕組み）。

3.2.3 同一レコードの同時編集禁止

1 レコードロック機能の導入

ユーザーがレコードを編集中は、

　　他のユーザーがそのレコードを編集できないようロックをかける。

ロックは 一定時間（例: 10分）操作がなければ自動解除 される。

レコードを開いた時点で Record_Lock テーブルに記録

編集完了後、または一定時間後にロック解除

ロックされたレコードを他ユーザーが開こうとした場合、警告を表示

レコードロック管理テーブルの例

```
CREATE TABLE Record_Lock (
    TableName NVARCHAR(50) NOT NULL,  -ロック対象のテーブル
    RecordID INT NOT NULL,  -ロックされたレコードID
    LockedBy NVARCHAR(50) NOT NULL,  -ロックしたユーザー
    LockedAt DATETIME DEFAULT GETDATE(),  -ロック開始時間
    PRIMARY KEY (TableName, RecordID)
);
```

3.2.4 競合発生時の警告と対策

シナリオ	対応策	警告メッセージ
別のユーザーが同じレコードを編集中	レコードロックを適用し、二重編集を防ぐ	「このデータは別のユーザーが編集中です。編集を続けるには、編集者に確認してください。」
確定ボタン押下時にデータが更新済み	変更内容を比較し、マージする仕組みを提供	「他のユーザーがこのデータを更新しました。変更内容を確認してください。」
通信エラー等でロックが解除されない	一定時間後にロックを自動解除	「編集ロックが解除されませんでした。管理者に問い合わせてください。」

3.3 リレーションシップとリアル値の使い分け

本システムでは、データ管理の効率化と柔軟性を両立するために、リレーションシップとリアル値の使い分けを明確に定めています。

1 リレーションシップを利用するデータ

変更の確率が低いマスタデータ（例：部門マスタ、職種マスタ、役職マスタなど）については、リレーションシップを用いて管理します。

テーブル名	主な項目	用途	設定方法
部門マスタ	部門ID, 部門名	従業員の所属を管理	リレーションシップ
職種マスタ	職種ID, 職種名	従業員の役割を管理	リレーションシップ
役職マスタ	役職ID, 役職名	従業員の役職を管理	リレーションシップ
従業員テーブル	従業員ID, 氏名, 部門ID, 職種ID	各従業員情報を管理	部門ID・職種IDとリレーション

2 リアル値をコピーするデータ

変更の可能性が高いデータ（例：顧客名、取引先名、施設名など）については、登録時にリアル値をコピーする方式を採用します。

テーブル名	主な項目	用途	設定方法
顧客マスタ	顧客ID, 顧客名	最新の顧客情報を管理	マスタ参照
取引履歴	取引ID, 顧客名, 取引日, 金額	過去の取引データを保持	登録時にリアル値コピー
施設マスタ	施設ID, 施設名	施設情報を管理	マスタ参照
契約履歴	契約ID, 施設名, 契約日, 契約内容	過去の契約データを保持	登録時にリアル値コピー

3 使い分けの方針

種類	方式	用途	例
変更が少ない情報	リレーションシップ	データ整合性を維持	部門、職種、役職
変更が多い情報	リアル値コピー	過去データを維持	顧客名、取引先名、施設名

運用のポイント:

1. 過去データの整合性が崩れる

例）顧客マスタの「〇〇株式会社」が「△△株式会社」に変更された場合、過去の契約情報まで全て新名称に変わってしまう。

2. リレーションシップを強制するとエラーのリスクが増える

例）担当者が異動や退職で削除された際に、過去の取引データと整合性が取れなくなる。

3. データ履歴管理が難しくなる

例）過去の契約を参照する際に「当時の契約名」と「現在の契約名」が異なると混乱が生じる。

4. 運用上の注意点

リアル値をコピーする方式を採用する場合でも、最新情報との整合性を保つために「マスタの参照ボタン」や「検索機能」を設ける。

リレーションシップ方式のテーブルについては、削除時の影響を事前にシミュレーションし、不要データの削除基準を明確化する。

リアル値コピー方式のデータについては、「登録時のタイムスタンプ」や「元の名称」などを記録し、変更履歴を確認できるようにする。

3.4 データ削除と復元

削除時のログを記録し、「削除申請→管理者確認→ログ記録→削除実行」 のプロセスを適用。

削除データは「削除ログテーブル」に記録し、管理者専用ツール（VBAスクリプト）を利用して復元可能にしました。

削除後30日以内なら、管理者が削除ログから直接復元可能。30日を超えた場合はバックアップデータから復元。

このように、システム全体のデータ管理において「変更可能性の高いデータはリアル値コピー方式」「変更可能性の低いデータはリレーションシップ方式」を適用し、

業務要件に応じた柔軟な設計を実施する。

4. システム構成

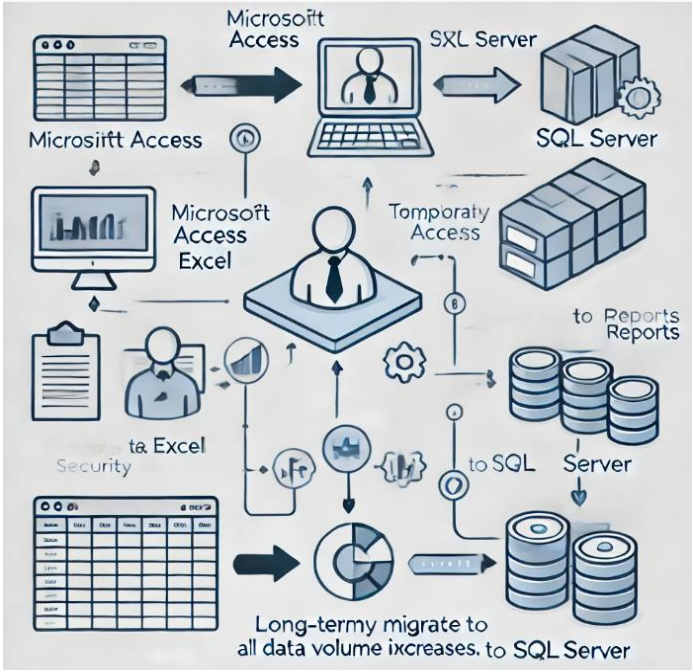
4.1 システムの全体像

1. システム構成図（データフローの視覚化）

このシステム運用初期では AccessとExcelが連携して運用 、将来的にSQL Serverがメインのデータ管理ツールへ移行する予定です。

データの流れは以下のようになります。

データの流れ



警備業務の流れ（顧客登録 → 契約管理 → 勤務スケジュール → 給与計算）

[顧客登録]
- 契約書作成

[契約管理]
- 契約条件の確認
- 警備業務内容の整理

[勤務スケジュール]
- スタッフ割り当て
- 勤務計画の作成
- 当日の業務チェック

[給与計算]
- 勤務実績の集計
- 給与支払い処理

運用時のデータフロー

1. **データ入力（Access）**

ユーザーが Access のフォームを使って、顧客情報・契約情報・警備スケジュールなどを入力

入力データは Access のテーブルに保存

2. **データの利用（Access → Excel）**

必要に応じて、Access のデータを Excel にエクスポート

Excel での分析・集計・帳票作成に活用（例：警備員シフト表、売上報告書）

3. **データの共有・レポート（Excel）**

Excel で作成された帳票・レポートを社内で共有

管理者が Excel のデータを基に意思決定

将来的なデータフロー（SQL Server 移行後）

1. **データ入力（Access）**

Access はフロントエンドの役割として残り、データの入力フォームを担当

直接 SQL Server にデータを書き込むよう変更

2. **データの保存・管理（SQL Server）**

レコード数増加に対応し、SQL Server でデータを一元管理

クエリ最適化やインデックス設定によるパフォーマンス向上

3. **データの活用（Excel・BIツール）**

SQL Server からデータを取得し、

Excel や BIツール（Power BI など）で分析

帳票出力・データの可視化を強化

まとめ

運用初期では Access を中心としたデータ管理

Excel は一時的なデータ処理・分析に利用

将来的に SQL Server へ完全移行し、スムーズなデータ管理を実現

2. システムの主な役割

Access: データの管理・入力フォーム　　Excel: レポート作成・データ分析　　SQL Server（移行計画）: クエリ最適化・負荷テスト

4.2 主要な機能と技術

1. データ管理機能

リアルタイム更新（変更履歴を記録）　マスタ管理（顧客・施設・契約）　アクセス権限の制御（一般・管理者・監査）

2. セキュリティとログ管理

アクセス権限管理　削除・更新履歴の保存　定期バックアップと復元手順　SQL Server のアクセス制限と暗号化通信

3. 業務自動化

定型レポートの自動生成　VBA を活用したデータ入力補助　エラー検出と通知システム

4.3 システムの使い方（操作フロー）

1. ユーザー別の操作フロー

一般ユーザー: データ入力・検索　管理者: データ修正・アクセス管理　監査ユーザー: 閲覧のみ（削除不可）

2. データ登録・更新時の注意点

入力ミス防止のためのチェック機能　変更履歴の自動保存　削除は管理者のみ可能（30日間復元可能）

4.4 データの安全管理（バックアップ・トラブル対策）

1. バックアップポリシー

デイリー: ローカルバックアップ　ウィークリー: SQL Server の自動バックアップ　月次: フルバックアップ（外部ストレージ）

2. トラブル発生時の対応

システム障害発生時の手順　データ破損時の復旧方法　サポート窓口（社内 IT 担当者の連絡先）

5. マルチユーザー環境

本システムでは、複数のユーザーが同時にデータを操作することを前提に設計 し、Access 環境と SQL Server（Microsoftのリレーショナルデータベース管理システム）環境で適切なパフォーマンスを確保できるようにする。

5.1 同時アクセス想定

環境	最大同時アクセス数	負荷試験・調整内容	想定される課題と対策
Access 環境	最大10名	負荷テスト後に調整	同時アクセス時のクエリ遅延
			競合発生時のデータロック管理
SQL Server 環境	最大50名 （将来的には100名以上も想定）	負荷テストを実施し、最適なサーバースペックを選定	接続数増加時の負荷分散
			クエリ最適化とインデックス管理

5.1.1 Access 環境での同時アクセス対策

Access では、データベースファイル (.accdb) を分割し、フロントエンド（ユーザー用）とバックエンド（データ管理用）に分離 することで、同時アクセス時の競合を最小限に抑える。

対策

1. 分散管理方式を採用

各ユーザーにフロントエンド (.accdb) を配布し、サーバー上のバックエンド (.accdb) とリンク。

直接テーブルを操作せず、クエリを最適化 してパフォーマンスを向上。

2. レコードロックの適用

編集ロック（悲観的ロック）を設定 し、同時編集によるデータ競合を回避。

クエリの最適化 により、データ取得速度を向上。

3. 同時検索の負荷軽減

SQL パススルークエリを利用 し、処理を SQL Server（Microsoftのリレーショナルデータベース管理システム） にオフロード。

検索用ビューを作成 し、Access からのクエリを高速化。

5.1.2 SQL Server（Microsoftのリレーショナルデータベース管理システム） 環境での同時アクセス対策

SQL Server（Microsoftのリレーショナルデータベース管理システム） に移行後は、最大50名（将来的には100名以上）が同時アクセスすることを想定し、

サーバー負荷の分散とスケーラビリティの確保 を行う。

対策

1. 負荷分散の実施

複数の SQL Server（Microsoftのリレーショナルデータベース管理システム） インスタンスを活用し、負荷を分散

クエリ実行の最適化（INDEX の適用、WITH (NOLOCK) の活用）

2. 接続管理

ユーザーごとの接続管理を行い、不要な接続を切断してリソースを節約。

アイドル状態の接続を監視し、タイムアウト設定を適用。

3. データキャッシュの活用

頻繁に参照されるデータは、メモリキャッシュを活用し応答速度を向上。

履歴データは別のデータベース（履歴DB）へ移行し、メインDBの負荷を軽減。

4. クラウド環境の検討

Microsoft Azure SQL Database の活用 も視野に入れ、負荷が増大した場合のスケールアップを容易にする。

5.1.3 負荷テスト計画

SQL Server（Microsoftのリレーショナルデータベース管理システム） 移行後、想定アクセス数に応じて 負荷テスト を実施し、パフォーマンスを検証。

負荷テスト項目	内容	目標
同時検索	50名が同時に勤務記録を検索	1秒以内
同時更新	10名が契約情報を同時に更新	1.5秒以内
大規模データ処理	100万件のレコード検索	3秒以内
ネットワーク負荷	VPN経由でのアクセス時のレスポンスタイムを測定	快適な操作性を確保

5.1.4 まとめ

Access 環境は最大10名の同時アクセスを想定し、負荷テスト後に調整。

SQL Server（Microsoftのリレーショナルデータベース管理システム） 環境では最大50名、将来的には100名以上のアクセスに対応できるよう設計。

負荷テストを段階的に実施し、パフォーマンスとスケーラビリティを最適化。

5.2 同時編集時のデータ競合対策

本システムでは、複数ユーザーが同じレコードを同時に編集する際のデータ競合を防止するため、

「楽観ロック方式（Optimistic Locking）」を採用 し、以下の対策を導入します。

5.2.1 楽観ロック方式の採用

楽観ロック方式とは？

データの競合が発生しないことを前提とし、ユーザーがデータ編集を確定するタイミングで変更の衝突を検出 する方式。

他ユーザーが同じレコードを先に更新した場合、再入力を促すメッセージを表示 し、データの上書きを防止。

5.2.2 競合検出の仕組み

各レコードに「更新タイムスタンプ」（LastUpdated カラム）を追加。

ユーザーがレコードを開いた際に LastUpdated の値を取得し、編集開始。

「確定」ボタン押下時に、現在の LastUpdated とデータベースの LastUpdated を比較。

一致している場合　→ 正常に更新処理を実施。

一致していない場合 → 競合が発生 したとみなし、エラーを表示。

テーブル設計例

5.2.3 競合発生時の対応選択肢

競合が発生した場合、ユーザーに以下の対応を選択させる

1.「ロック解除を待つ」

一定時間後に最新データを自動取得し、再編集を許可（更新間隔を5秒などに設定可能）。

管理者が手動でロック解除する機能も提供。

2.「別のレコードに切り替える」

現在のレコードの編集を中断し、他のレコードの編集に切り替え可能。

3.「変更を上書き」

管理者権限を持つユーザーのみ、強制的に変更を適用できる（ただしログに記録）。

5.2.4 競合発生時のメッセージ表示

シナリオ	表示メッセージ	ユーザー対応
他ユーザーが先に更新した場合	「このデータは他のユーザーによって変更されました。 最新のデータを取得してください。」	最新データを取得し、再入力
同じレコードを複数人が編集中	「このデータは現在、別のユーザーが編集中です。編集を続けますか？」	継続 or 他のレコードに切り替え
強制上書き可能な場合 （管理者向け）	「データの競合が発生しました。管理者権限で変更を確定しますか？」	強制確定（ログに記録）

5.2.5 物理ロック（悲観ロック）との違い

項目	楽観ロック	悲観ロック
仕組み	変更時に競合を検出	編集時にロックを取得
同時編集	可能（ただし競合時は再入力）	不可（他ユーザーは編集できない）
パフォーマンス	高い（ロックを使用しないため）	低い（ロックの管理が必要）
適用シナリオ	多数のユーザーが編集するデータ	重要なデータ（請求書・契約情報）

本システムでは、楽観ロック方式を基本とし、契約情報や請求データなどの重要データのみ悲観ロックを導入 する。

5.2.6 競合防止の追加対策

編集開始時にレコードの「一時コピー」を作成 し、競合が発生してもユーザーが変更を保持できるようにする。

5秒ごとにバックグラウンドで最新データを自動取得し、競合発生前に警告を表示。

「編集中」ユーザー名を表示し、どのユーザーが作業中か確認可能にする。

5.2.7 まとめ

楽観ロック方式を採用し、同時編集時の競合を最小限に抑える。

競合が発生した場合は「ロック解除を待つ」「別のレコードに切り替える」「変更を上書き」の選択肢を提供。

重要データのみ悲観ロックを適用し、安全性を確保。

6. データ改ざん・削除リスクの対策

6.1 直接編集によるリスク回避

本システムでは、データの改ざん・削除リスクを最小限に抑えるため、直接編集を制限し、すべてのデータ変更・削除に承認プロセスとログ管理を適用 します。

6.1.1 「確定」ボタンによる登録制御

ユーザーが データを入力・修正しても、「確定」ボタンを押さない限り、本登録されない仕様 を採用。

確定前にプレビュー機能を導入 し、変更内容を確認可能にする。

変更を一時保存できる仕組みを実装 し、誤操作時に元のデータへ戻せる。

変更プロセス

1. データ入力・修正
2. 確定ボタンを押す
3. 変更履歴テーブルへ記録
4. 本登録としてデータベースに反映
5. 管理者が変更ログを監視

6.1.2 ログ管理の強化

管理者がすべてのデータ変更履歴を追跡できるよう、詳細なログを記録 し、改ざんリスクを軽減。

1 変更履歴ログ

すべてのデータ更新は、履歴テーブルに記録 し、元のデータを復元可能にする。

変更履歴をJSON形式で保存 し、差分を明確化。

```
CREATE TABLE Update_Log (
    LogID INT IDENTITY(1,1) PRIMARY KEY,
    TableName NVARCHAR(50) NOT NULL,  -更新対象のテーブル
    RecordID INT NOT NULL,  -更新されたレコードID
    UpdatedBy NVARCHAR(50) NOT NULL,  -更新者
    UpdatedAt DATETIME DEFAULT GETDATE(),  -更新日時
    OldData NVARCHAR(MAX),  -変更前データ（JSON形式）
    NewData NVARCHAR(MAX)  -変更後データ（JSON形式）
);
```

2 管理者用の変更履歴閲覧機能

管理者が特定のレコードの変更履歴を確認可能 な専用画面を提供。

変更が承認されていない場合、元のデータに戻せる「ロールバック機能」 を実装。

6.1.3 削除時のリスク回避策

削除時は、即時削除を禁止し、「削除申請 → 承認 → ログ記録 → 削除実行」 のプロセスを適用 する。

1 削除申請ワークフロー

1. ユーザーが削除申請を実行
2. 管理者が申請を確認
3. ログ記録後、削除を実行
4. 削除ログを別テーブルへ保存

2 削除ログの保存

削除されたデータを保持し、復元可能な仕組みを構築。

```
CREATE TABLE Deletion_Log (
    LogID INT IDENTITY(1,1) PRIMARY KEY,
    TableName NVARCHAR(50) NOT NULL, -削除対象のテーブル
    RecordID INT NOT NULL, -削除されたレコードID
    DeletedBy NVARCHAR(50) NOT NULL, -削除実行者
    DeletedAt DATETIME DEFAULT GETDATE(), -削除実行日時
    DeletedData NVARCHAR(MAX) -削除されたデータ（JSON 形式）
);
```

3 削除プロセスの自動化

一定期間（例：1年）経過後に削除ログをアーカイブ。

管理者が定期的に削除データを確認し、不正な削除がないかチェック。

6.1.4 まとめ

対策	詳細
確定ボタンによる変更確定	確定ボタンを押さない限りデータが本登録されない
ログ管理の強化	変更履歴をログに保存し、管理者が追跡可能
削除ワークフローの導入	削除は即時実行せず、申請・承認プロセスを適用
削除データの保持	削除されたデータはログとして保存し、復元可能に

6.2 削除ルールの明確化

本システムでは、データ削除に関する明確なルールを設け、誤削除や不正削除を防止し、適切な復元手順を確立 します。

6.2.1 削除のフロー

削除は即時実行されるのではなく、以下の手順を経ることで安全性を確保する。

1. 削除申請

ユーザーが削除を希望するデータを「削除申請」として登録。

申請内容には 削除理由を必須入力 させる。

2. 管理者の確認

管理者が 削除申請を承認または却下 する。

却下された場合はデータはそのまま保持される。

3. 削除ログの記録

承認された削除データは 削除ログテーブルに保存（削除前のデータ内容も保持）。

4. ソフトデリート（論理削除）

データは即時削除されず、IsDeleted フラグを 1 に変更

削除後30日間は復元可能（リストア機能を提供）

5. 物理削除（完全削除）

90日経過後、バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）データを削除

物理削除を実行し、データを完全に削除。

6.2.2 削除データの管理

1 削除ログの保存

削除されたデータは 専用の削除ログテーブル に記録し、追跡可能な形で保存。

```
CREATE TABLE Deletion_Log (
    LogID INT IDENTITY(1,1) PRIMARY KEY,
    TableName NVARCHAR(50) NOT NULL,  -削除対象のテーブル
    RecordID INT NOT NULL,  -削除されたレコードID
    DeletedBy NVARCHAR(50) NOT NULL,  -削除実行者
    DeletedAt DATETIME DEFAULT GETDATE(),  -削除実行日時
    DeletedData NVARCHAR(MAX)  -削除されたデータ（JSON 形式）
);
```

2 ソフトデリート方式の採用

すぐにデータを削除せず、「削除フラグ（IsDeleted）」を追加し、データを非表示にする方式 を採用。

削除されたデータは30日間 「復元可能データ」として保持 し、必要に応じて復旧できるようにする。

テーブル構造（ソフトデリート対応）

通常の検索では IsDeleted = 0 のデータのみを取得する。

6.2.3 バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）と復元ルール

項目	内容
バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）タイミング	削除前に自動バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）を取得
バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）保存期間	90日間
削除データの復元期間	30日以内
復元方法	管理者が削除ログから復元処理を実行
完全削除の条件	90日経過後、バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）を削除

6.2.4 削除ルールのまとめ

ルール	内容
削除申請制	ユーザーは管理者の承認なしに削除できない
ソフトデリート採用	すぐに削除せず、IsDeleted フラグで管理
削除後30日間復元可能	誤削除の場合、30日以内であれば復元可能
削除ログの記録	削除されたデータは JSON 形式でログに保存
完全削除は90日後	90日経過後、物理削除を実施

6.2.5 削除時の警告とユーザー通知

シナリオ	表示メッセージ	ユーザー対応
削除申請時	「このデータを削除申請しますか？ 管理者の承認が必要です。」	承認を待つ
削除後30日以内	「このデータは削除済みですが、30日以内なら復元可能です。」	管理者に復元を依頼
完全削除実行前	「このデータは90日間バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）されています。削除後は復元できません。」	確認の上、完全削除

6.2.6 まとめ

即時削除を禁止し、削除申請・承認プロセスを導入

削除後30日間は復元可能なソフトデリート方式を採用

90日間バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）を保持し、完全削除時にデータが消去

削除ログを詳細に記録し、管理者が追跡可能にする

6.3 データ削除の「復元30日間」の方法

本システムでは、データ削除後30日以内であれば即時復元が可能、それ以降もバックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）から復元可能な仕組み を採用し、誤削除リスクを低減します。

6.3.1 削除データの保存方法

削除されたデータは、即時削除されず「削除ログテーブル」に記録される。

削除ログには削除されたデータのすべての情報を保持 し、復元時に元の状態に戻せるようにする。

30日経過後、自動的にバックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）サーバーへ移動 し、本番環境からは削除される。

6.3.2 削除ログテーブルの設計

削除データを保存し、必要に応じて復元できるよう、以下のテーブルを用意する。

```
CREATE TABLE Deletion_Log (
    LogID INT IDENTITY(1,1) PRIMARY KEY,
    TableName NVARCHAR(50) NOT NULL,  -削除対象のテーブル
    RecordID INT NOT NULL,  -削除されたレコードID
    DeletedBy NVARCHAR(50) NOT NULL,  -削除実行者
    DeletedAt DATETIME DEFAULT GETDATE(),  -削除実行日時
    DeletedData NVARCHAR(MAX)  -削除されたデータ（JSON 形式）
);
```

TableName … 削除されたデータの元のテーブル名

RecordID … 削除されたレコードの元のID

DeletedBy … 削除を実行したユーザー

DeletedAt … 削除実行日時

DeletedData … 削除されたデータの内容（JSON 形式で保存）

6.3.3 削除後30日間の復元方法

削除後30日以内なら、管理者が削除ログから即時復元できる。

1 VBAスクリプトを利用した復元ツール

管理者専用のVBAスクリプトを用意 し、削除データのリストから選択して復元可能にする。

削除データを削除ログテーブルから取得し、元のテーブルに挿入。

復元処理のログを記録 し、復元の正当性を証明。

```
Sub RestoreDeletedRecord(TableName As String, RecordID As Integer)
    Dim db As DAO.Database
    Dim rs As DAO.Recordset
    Dim sql As String
    Set db = CurrentDb()
    ' 削除ログからデータ取得
    sql = "SELECT DeletedData FROM Deletion_Log WHERE TableName='" & TableName & "' AND RecordID=" & RecordID
    Set rs = db.OpenRecordset(sql)
    If Not rs.EOF Then
        Dim jsonData As String
        jsonData = rs!DeletedData
        ' JSONデータを元のテーブルに復元（パース処理は別途実装）
        Call InsertRecordFromJSON(TableName, jsonData)
        ' 削除ログから該当データを削除
        sql = "DELETE FROM Deletion_Log WHERE TableName='" & TableName & "' AND RecordID=" & RecordID
        db.Execute sql
    End If
    rs.Close
    Set rs = Nothing
    Set db = Nothing
End Sub
```

6.3.4 30日を超えた場合の復元

削除後30日を超えると、データは削除ログテーブルからバックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）サーバーへ移行する。

1 バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）データの復元

管理者がバックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）データをリストア可能

30日経過後は、SQL Server（Microsoftのリレーショナルデータベース管理システム） の バックアップ（システム障害時にデータを復旧できるよう、

定期的に保存すること）ファイル（bak 形式）から手動で復元

バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）からの復元手順：

1. バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）ファイルを取得
2. SQL Server（Microsoftのリレーショナルデータベース管理システム） Management Studio (SSMS) を使用し、該当データを抽出
3. 復元対象のレコードを手動で挿入

2 90日経過後のデータ完全削除

90日を超えたデータは完全削除（物理削除）

管理者のみが削除できるスクリプトを実行

6.3.5 削除・復元のワークフロー

ステップ	期間	内容
削除ログに記録	即時	削除データは削除ログテーブルに保存
削除後30日以内	30日間	管理者が削除ログから即時復元可能
削除後30日経過	30～90日	データはバックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）に移動、手動復元可能
削除後90日経過	90日後	バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）も削除、完全削除（復元不可）

6.3.6 削除・復元時の警告メッセージ

シナリオ	表示メッセージ	ユーザー対応
削除申請時	「このデータを削除申請しますか？ 削除後30日間は復元可能です。」	承認待ち
削除後30日以内	「削除済みのデータですが、復元できます。」	管理者に復元依頼
削除後30日超え	「このデータはバックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）から復元できますが、管理者の作業が必要です。」	管理者にバックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）リストア依頼
削除後90日超え	「このデータは完全削除され、復元できません。」	復元不可

6.3.7 まとめ

削除データは削除ログに保存され、30日以内なら即時復元可能

30日経過後はバックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）から復元可能、90日を超えると完全削除

VBAスクリプトを活用し、管理者が簡単に削除データを復元可能

削除前に必ずバックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）を取得し、誤削除時のリカバリー対応を強化

7. セキュリティ仕様

7.1 アクセス制御

本システムでは、不正アクセスを防止し、データの機密性を確保するために、VBA を活用したユーザー認証システムを導入 します。

また、認証システムの異常時に管理者が緊急対応できる仕組み を整備し、安全性と運用性の両立を図ります。

7.1.1 ユーザー認証の実装

Access の VBA を活用し、起動時にログイン認証を必須化 する。

1 認証方式

ユーザーIDとパスワードの入力を必須化

入力情報は暗号化し、DB内に保存

認証成功後にアクセス権限を割り当て、操作範囲を制限

認証情報をセッション管理し、一定時間操作がない場合は再認証を求める

Windows の Active Directory 連携も検討

2 ユーザーロールと権限管理

ユーザーロール	説明	許可される操作
管理者	システム設定・データ管理	すべての機能にアクセス
一般ユーザー	業務データの閲覧・入力	データの閲覧・一部更新可能（削除不可）
閲覧専用ユーザー	閲覧のみ許可	データ変更不可

7.1.2 VBA を活用した認証システム

VBA を活用し、Access 起動時にログインフォームを表示。

```
Private Sub btnLogin_Click()
    Dim db As DAO.Database
    Dim rs As DAO.Recordset
    Dim sql As String
    Dim enteredUsername As String
    Dim enteredPassword As String
    enteredUsername = Me.txtUsername.Value
    enteredPassword = Me.txtPassword.Value
    ' ハッシュ化されたパスワードと比較
    sql = "SELECT * FROM Users WHERE Username='" & enteredUsername & "'"
    Set db = CurrentDb
    Set rs = db.OpenRecordset(sql)
    If Not rs.EOF Then
        If rs!PasswordHash = HashFunction(enteredPassword) Then
            MsgBox "ログイン成功", vbInformation
            DoCmd.OpenForm "MainMenu"
            DoCmd.Close acForm, "LoginForm"
        Else
            MsgBox "パスワードが違います", vbCritical
        End If
    Else
        MsgBox "ユーザーが存在しません", vbCritical
    End If
    rs.Close
    Set rs = Nothing
    Set db = Nothing
End Sub
```

ログインフォームでユーザー名・パスワードを入力

入力されたパスワードをハッシュ化し、保存済みの値と比較

認証成功時にメイン画面を開き、失敗時は警告を表示

7.1.3 認証システムの異常時対応

認証システムが異常発生し、通常のログインができなくなった場合に備え、管理者用の緊急アクセスを提供。

1 緊急ログインモード

管理者専用の「緊急アクセスコード」を事前発行

通常のログインが失敗した際、管理者コードを入力することでログイン可能

緊急ログインの実行履歴を記録し、不正利用を防止

vba

コピーする編集する

```
If enteredUsername = "admin" And enteredPassword = EmergencyCode Then
    MsgBox "緊急ログイン成功", vbInformation
    DoCmd.OpenForm "AdminPanel"
End If
```

2 ログインエラー時の対応

シナリオ	メッセージ	管理者対応
パスワード忘れ	「パスワードをリセットしてください。」	管理者がリセット処理
複数回ログイン失敗	「アカウントがロックされました。」	管理者がロック解除
DB接続エラー	「認証サーバーに接続できません。」	DB接続をチェック

3 不正アクセス検知

ログイン試行回数を記録し、一定回数超過でロック

異常アクセスが検知された場合、管理者へアラート通知

ログイン履歴テーブルを作成し、アクセス履歴を監視

```
CREATE TABLE Login_History (
    LogID INT IDENTITY(1,1) PRIMARY KEY,
    Username NVARCHAR(50) NOT NULL,
    LoginTime DATETIME DEFAULT GETDATE(),
    LoginStatus NVARCHAR(20)  -成功 / 失敗
);
```

7.1.4 セキュリティ強化のための追加対策

対策	内容
多要素認証（MFA）の導入	管理者アカウントはメール認証などを追加
一定時間経過後の自動ログアウト	15分間操作がない場合はログアウト
定期的なパスワード変更	90日ごとにパスワード変更を促す
アカウントロック機能	5回ログイン失敗で一時ロック

7.1.5 まとめ

VBAを活用し、Access起動時にユーザー認証を必須化

管理者・一般ユーザー・閲覧専用ユーザーの3階層でアクセス制御

緊急アクセスコードを導入し、認証システム異常時の対応を明確化

ログイン履歴を記録し、不正アクセスを監視

多要素認証・自動ログアウト・パスワード変更の運用を確立

8. SQL Server への移行計画

本システムでは、Microsoft Access から SQL Server（Microsoftのリレーショナルデータベース管理システム） への移行を 段階的に実施し、データの整合性・パフォーマンスを確保 しま

移行プロセスごとに チェックリストを策定 し、各フェーズで負荷テストを実施（最大50名の同時アクセスを想定）して、移行の品質を確保します。

8.1 移行の全体スケジュール

フェーズ	期間	内容
フェーズ1	1～6ヶ月	SQL Server（Microsoftのリレーショナルデータベース管理システム） の テスト環境構築、初期負荷テスト実施
フェーズ2	7～12ヶ月	データ移行の試験運用（勤務記録など低リスクデータから）
フェーズ3	13～18ヶ月	主要データ（契約・人事情報）の移行と SQL Server（Microsoftのリレーショナルデータベース 管理システム） での運用開始
フェーズ4	19～24ヶ月	完全移行後の最適化・パフォーマンスチューニング

8.2 各移行段階ごとのチェックリスト

移行の各段階で、以下の 品質チェックリスト を適用し、移行品質を確保する。

1 移行前の準備（フェーズ1）

SQL Server（Microsoftのリレーショナルデータベース管理システム） の テスト環境構築

既存 Access データベースの スキーマ（テーブル構造）を分析

SQL Server（Microsoftのリレーショナルデータベース管理システム） で データ型の互換性を確認

インデックスと外部キーの設計

セキュリティ要件の明確化（アクセス権限・認証方式）

2 データ移行試験（フェーズ2）

一部のテーブルを SQL Server（Microsoftのリレーショナルデータベース管理システム） に移行（勤務記録データなど）

データ整合性チェック（Access のデータと比較）

テストユーザーによる動作確認

Access から SQL Server（Microsoftのリレーショナルデータベース管理システム） へのリンクテーブルのテスト

初回負荷テスト（最大10名）

3 本番移行（フェーズ3）

主要テーブル（契約・人事情報）の SQL Server（Microsoftのリレーショナルデータベース管理システム） への移行

クエリ・ストアドプロシージャの最適化

バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）ポリシーの確立

同時アクセス50名での負荷テスト

システム全体の動作テスト（結合テスト）

4 完全移行後の最適化（フェーズ4）

運用開始後のエラー監視・ログ分析

パフォーマンス改善（インデックス調整・クエリ最適化）

SQL Server（Microsoftのリレーショナルデータベース管理システム） の負荷分散検討

最終的な Access の役割整理（補助ツールとして運用）

8.3 負荷テストの詳細

移行フェーズごとに、負荷テストを実施 し、SQL Server（Microsoftのリレーショナルデータベース管理システム） のパフォーマンスを評価。

テスト内容	実施フェーズ	許容範囲
同時検索テスト（10万件データ検索）	フェーズ2	1秒以内
同時更新テスト（10名が一斉に勤務データ更新）	フェーズ2	1.5秒以内
アクセス集中時の応答速度（50名同時ログイン）	フェーズ3	2秒以内
データ整合性チェック（移行前後のデータ比較）	フェーズ3	100%一致

負荷テストの結果をもとに、SQL Server（Microsoftのリレーショナルデータベース管理システム） のパフォーマンスチューニング

（インデックスの適用・ストアドプロシージャの最適化）を実施。フェーズ4では、運用開始後のトラフィックを監視し、必要に応じてサーバースペックを調整。

8.4 移行時のデータ整合性チェック

SQL Server（Microsoftのリレーショナルデータベース管理システム）へ移行後、データの整合性を確保するために、以下の方法でチェックを実施。

1 移行前後のレコード数比較

```
SELECT COUNT(*) FROM AccessTable
UNION ALL
SELECT COUNT(*) FROM SQLServerTable;
```

Access と SQL Server（Microsoftのリレーショナルデータベース管理システム）のレコード数が一致しているか確認。

2 プライマリキー・外部キーの整合性

```
SELECT COUNT(*) FROM SQLServerTable WHERE ForeignKey IS NULL;
```

外部キーが適切に設定されているかチェック。

3 ランダムデータ検証

```
SELECT * FROM SQLServerTable
ORDER BY NEWID()
FETCH FIRST 10 PERCENT ROWS ONLY;
```

ランダムに抽出したデータを Access 版と比較し、データの欠損や不整合がないか確認。

8.5 SQL Server（Microsoftのリレーショナルデータベース管理システム）への移行時のリスクと対策

リスク	対策
データ整合性の欠如	移行前後でデータ比較を実施（レコード数・内容の検証）
パフォーマンス低下	負荷テスト結果に基づきインデックス最適化
Access とのリンクトラブル	ODBC 接続テストを繰り返し実施
SQL Server（Microsoftのリレーショナルデータベース管理システム）停止時の	

8.6 移行計画の最終目標

Access から SQL Server（Microsoftのリレーショナルデータベース管理システム）へ段階的に移行し、データの品質を確保

負荷テストを通じて、最大50名の同時アクセスに耐えうる環境を構築

移行後も定期的な監視を行い、運用の最適化を継続

SQL Server（Microsoftのリレーショナルデータベース管理システム）移行後のパフォーマンス監視

本システムでは、SQL Server（Microsoftのリレーショナルデータベース管理システム）へ移行後のパフォーマンスを継続的に監視し、安定した運用を確保 するため、

定期的なレスポンスチェックとパフォーマンス最適化を実施します。

1. パフォーマンス監視スケジュール

期間	監視頻度	実施内容
移行後 6か月間	週1回	クエリの応答時間・CPU負荷を測定、最適化
移行後 6か月以降	月1回	定期的なパフォーマンス分析と負荷分散の見直し
異常発生時	即時	アラート通知に基づきトラブルシューティング

2. レスポンス監視

移行後、SQL Server（Microsoftのリレーショナルデータベース管理システム）のパフォーマンスをリアルタイムで監視し、

レスポンス時間が2秒以上のクエリを抽出し、最適化 します。

1 スロークエリの抽出

SQL Server（Microsoftのリレーショナルデータベース管理システム）の sys.dm_exec_requests や sys.dm_exec_query_stats を利用し、実行時間の長いクエリを特定。

```
SELECT TOP 20
    total_worker_time/execution_count AS AvgCPUTime,
    total_elapsed_time/execution_count AS AvgElapsedTime,
    execution_count,
    query_text
FROM sys.dm_exec_query_stats
CROSS APPLY sys.dm_exec_sql_text(sql_handle)
WHERE total_elapsed_time/execution_count > 2000  -2秒以上のクエリ
ORDER BY AvgElapsedTime DESC;
```

最適化の対象となるクエリ

2秒以上の応答時間が発生するクエリ

高頻度で実行され、CPU使用率が高いクエリ

インデックス不足によりスキャンが発生しているクエリ

3. パフォーマンス最適化の実施

スロークエリの特定後、以下の手順でパフォーマンスを改善。

1 インデックスの最適化

頻繁に検索される列に適切なインデックスを作成

未使用のインデックスを削除し、書き込みパフォーマンスを向上

CREATE INDEX IDX_EmployeeName ON Employee (Name);

2 クエリの書き換え

不要な SELECT * を避け、必要なカラムのみ取得

テーブル結合を見直し、サブクエリの使用を最小限に

ストアドプロシージャ化し、パフォーマンスを向上

```
CREATE PROCEDURE GetEmployeeByID
@EmpID INT
AS
BEGIN
    SELECT Name, Position FROM Employee WHERE EmployeeID = @EmpID;
END
```

3 キャッシュの活用

頻繁に実行されるクエリの結果をキャッシュ

SQL Server（Microsoftのリレーショナルデータベース管理システム） の Query Store を有効化し、最適なクエリプランを選択

4. ダッシュボードによるリアルタイム監視

管理者向けに SQL Server（Microsoftのリレーショナルデータベース管理システム）のパフォーマンスを可視化するダッシュボードを設置 し、リアルタイム監視を可能に。

1 ダッシュボードの監視項目

監視項目	内容
CPU 使用率	SQL Server（Microsoftのリレーショナルデータベース管理システム）の CPU 負荷状況
メモリ使用率	使用メモリとスワップ発生状況
ディスクI/O	読み書き速度とボトルネックの検出
スロークエリリスト	2秒以上のクエリをリアルタイム表示
接続ユーザー数	現在接続しているユーザー数
トランザクションのロック状態	ロックが発生しているテーブル

2 Power BI や Grafana との連携

SQL Server（Microsoftのリレーショナルデータベース管理システム）のデータを

Power BI に連携し、管理者向けにレポートを提供Grafana でリアルタイムモニタリングを実施し、異常検知

5. 異常検知とアラート通知

システム異常が発生した際、管理者へ即時通知を送る仕組みを導入。

1 異常発生時のアラート設定

レスポンスタイムが3秒を超えた場合、アラート通知

CPU 使用率が80%を超えた場合、アラート通知

ディスク I/O 遅延が発生した場合、警告ログを出力

```
EXEC msdb.dbo.sp_send_dbmail
    @profile_name = 'AdminNotification',
    @recipients = 'admin@example.com',
    @subject = 'SQL Server（Microsoftのリレーショナルデータベース管理システム） Performance Alert',
    @body = 'CPU Usage exceeded 80%. Check the server performance.';
```

2 アラート対応のフロー

発ﾋ 自動対応　　管理者対応

スﾛ クエリプランﾍ インデックス調整

CPI 不要なプロｾ サーバーリソースの増強

ディー時キャッシ ストレージ拡張

6. まとめ

移行後6か月は週1回、その後は月1回のパフォーマンス分析を実施

レスポンス時間が2秒以上のクエリを抽出し、最適化

ダッシュボードを活用し、リアルタイムでCPU・メモリ・スロークエリを監視

異常検知時は自動アラートを送信し、管理者が即時対応可能に

9. Access × Excel 連携で業務を効率化

Access と Excel をシームレスに連携させることで、データ検索・集計・帳票作成の業務を大幅に効率化 します。

また、データ保護機能を実装し、不正なデータ持ち出しや改変を防止 しながら、安全かつ利便性の高い運用を実現します。

9.1 データ抽出の最適化（ボタン1つで簡単にエクスポート！）

1 高速検索＆ワンクリック抽出

Access の 検索フォーム で条件を入力し、「データ抽出ボタン」 を押すだけで、Excel にデータが自動出力されます。

ユーザーがExcelで手動入力する手間をゼロに！

検索条件を変更すれば、自由自在に抽出データを変更可能！

複数のテーブルを結合し、必要な情報を1つのExcelファイルに統合！

vba

コピーする編集する

```
Sub ExportToExcel()
    Dim db As DAO.Database
    Dim rs As DAO.Recordset
    Dim xlApp As Object
    Dim xlBook As Object
    Dim xlSheet As Object
    ' Access のクエリを開く
    Set db = CurrentDb()
    Set rs = db.OpenRecordset("SELECT * FROM ExportQuery") ' クエリ名を指定
    ' Excel を起動
    Set xlApp = CreateObject("Excel.Application")
    xlApp.Visible = True
    Set xlBook = xlApp.Workbooks.Add
    Set xlSheet = xlBook.Sheets(1)
    ' フィールド名をExcelの1行目に出力
    Dim i As Integer
    For i = 0 To rs.Fields.Count 1
        xlSheet.Cells(1, i + 1).Value = rs.Fields(i).Name
    Next i
    ' データをExcelへ転送
    xlSheet.Range("A2").CopyFromRecordset rs
    rs.Close
    Set rs = Nothing
    Set db = Nothing
End Sub
```

◈ これにより、Access のデータをExcelに瞬時に出力し、手作業を不要に！

◈ 出力フォーマットは事前に設定可能！ たとえば、売上集計表、契約一覧表、警備員シフト表など用途別にテンプレートを作成できる！

2 Excelの自動フォーマット適用

Access から出力したデータに対し、Excel 側で「見やすいフォーマット」を自動適用！

・列の幅を自動調整

・フィルター機能を有効化

・金額欄は「通貨表示」、日付欄は「YYYY/MM/DD」形式に統一！

・グラフやピボットテーブルを自動生成！（売上推移グラフ、勤務実績表など）

```
Sub FormatExcelSheet()
    Dim xlApp As Object
    Dim xlBook As Object
    Dim xlSheet As Object
    ' Excel を取得
    Set xlApp = GetObject(, "Excel.Application")
    Set xlBook = xlApp.ActiveWorkbook
    Set xlSheet = xlBook.Sheets(1)
    ' 列幅自動調整
    xlSheet.Columns("A:Z").AutoFit
    ' ヘッダーの装飾
    xlSheet.Rows(1).Font.Bold = True
    xlSheet.Rows(1).Interior.ColorIndex = 15  ' 灰色背景
    ' フィルター適用
    xlSheet.Cells(1, 1).AutoFilter
    ' ピボットテーブル作成（売上データの場合）
    xlSheet.Cells(1, 1).Select
    xlApp.ActiveWorkbook.PivotTableWizard SourceType:=xlDatabase, SourceData:=xlSheet.UsedRange, _
    TableDestination:=xlSheet.Cells(5, 10), TableName:="PivotTable1"
    Set xlSheet = Nothing
    Set xlBook = Nothing
    Set xlApp = Nothing
End Sub
```

9.2 Excel 連携の制限（セキュリティを万全に！）

Excel 連携機能は便利な一方で、情報漏洩のリスクもあるため、以下の対策を導入！

1 機密情報のExcel出力を制限

氏名・住所・電話番号・契約情報を含むデータは 役員の承認がなければ出力不可 にする。

承認された場合のみ、特定の管理者がパスワードを入力すると出力可能！

承認履歴をログとして記録。

```
Function IsAuthorized(user As String) As Boolean
    ' ユーザー権限をDBから取得
    Dim db As DAO.Database
    Dim rs As DAO.Recordset
    Set db = CurrentDb()
    Set rs = db.OpenRecordset("SELECT Permission FROM Users WHERE Username='" & user & "'")
    If Not rs.EOF Then
        IsAuthorized = (rs!Permission = "Admin" Or rs!Permission = "Manager")
    Else
        IsAuthorized = False
    End If
    rs.Close
    Set rs = Nothing
    Set db = Nothing
End Function
```

◈ これにより、機密情報の流出を防止！

◈ 特定の管理者のみ出力可能にし、情報漏洩リスクを最小限に！

2 Excel 出力のログ監査

誰が、いつ、どのデータを Excel に出力したのか？ を記録し、不正持ち出しを防止。

Excel 出力時に、自動でログテーブルに保存。

```
CREATE TABLE Export_Log (
    LogID INT IDENTITY(1,1) PRIMARY KEY,
    ExportedBy NVARCHAR(50),
    ExportedAt DATETIME DEFAULT GETDATE(),
    ExportedData NVARCHAR(MAX)
);
```

◈ 管理者は「誰が何を出力したか？」をいつでも確認できる！

◈ 不正なデータ持ち出しがあれば即座に検知可能！

3 Excelファイルの改変防止

Excel ファイルを 読み取り専用に設定し、不正改変を防止！

Access からExcelに送信したデータに対して、変更時に警告メッセージを表示！

Excel 側で「シート保護」を適用し、重要なデータが編集できないようにする！

```
Sub ProtectExcel()
    Dim xlApp As Object
    Dim xlBook As Object
    Dim xlSheet As Object
    Set xlApp = GetObject(, "Excel.Application")
    Set xlBook = xlApp.ActiveWorkbook
    Set xlSheet = xlBook.Sheets(1)
    ' シート保護を適用（パスワード付き）
    xlSheet.Protect Password:="securepass", UserInterFaceOnly:=True
    Set xlSheet = Nothing
    Set xlBook = Nothing
    Set xlApp = Nothing
End Sub
```

まとめ

ワンクリックで Access のデータを Excel に出力！

Excel 側で自動フォーマット＆集計！

機密情報の持ち出しを防ぐ強力なセキュリティ対策！

10. 今後のスケジュール

本プロジェクトでは、Access から SQL Server（Microsoftのリレーショナルデータベース管理システム） への移行を4年間のフェーズで段階的に実施 し、

データの最適化・負荷テスト・移行作業・本運用の順に進めます。

各フェーズごとに具体的なタスクを明確にし、計画的に移行を行います。

10.1 移行スケジュール（詳細版）

フェーズ	実施内容	期間
フェーズ1	Access の分散運用開始、データ最適化	1年目
フェーズ2	SQL Server（Microsoftのリレーショナルデータベース管理システム） のテスト運用開始	2年目
フェーズ3	人事・契約情報の SQL Server（Microsoftのリレーショナルデータベース管理システム） 移行	3年目
フェーズ4	SQL Server（Microsoftのリレーショナルデータベース管理システム） での本運用開始	4年目

10.2 フェーズごとの詳細作業

フェーズ1：Access 分散運用開始、データ最適化（1年目）

目的：Access のデータ構造を整理し、SQL Server（Microsoftのリレーショナルデータベース管理システム） へのスムーズな移行に備える。

タスク一覧

- Access データベースのテーブル設計見直し
- 業務ごとにデータの分割・整理（重複排除、正規化）
- Access フロントエンドとバックエンド（データ保存用）を分離
- 各部門のユーザーにAccessの運用研修を実施
- 分散管理方式（部門ごとの accdb）を導入
- 負荷テスト（最大10名の同時アクセス）を実施
- Access のパフォーマンス最適化（クエリ・インデックス調整）

成果物

正規化された Access データベース

ユーザー操作マニュアル

Access 分散運用マップ

フェーズ2：SQL Server（Microsoftのリレーショナルデータベース管理システム） のテスト運用開始（2年目）

目的：SQL Server（Microsoftのリレーショナルデータベース管理システム） 環境の構築とテスト運用を実施し、移行準備を進める。

タスク一覧

SQL Server（Microsoftのリレーショナルデータベース管理システム） のテスト環境構築

Access から SQL Server（Microsoftのリレーショナルデータベース管理システム） への接続テスト（ODBCリンクテーブル設定）

SQL Server（Microsoftのリレーショナルデータベース管理システム） のテーブル設計（Access との互換性を考慮）

勤務記録データを SQL Server（Microsoftのリレーショナルデータベース管理システム） に移行し、動作確認

スロークエリを特定し、パフォーマンス改善

同時アクセステスト（最大20名）を実施

バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）・復元の手順確立

成果物

SQL Server（Microsoftのリレーショナルデータベース管理システム） の初期環境

移行用スクリプト（Access → SQL Server（Microsoftのリレーショナルデータベース管理システム））

負荷テスト結果レポート

フェーズ3：人事・契約情報の SQL Server（Microsoftのリレーショナルデータベース管理システム） 移行（3年目）

目的：主要なデータ（人事・契約情報）を SQL Server（Microsoftのリレーショナルデータベース管理システム） に移行し、本運用に近い形で動作を確認。

タスク一覧

人事・契約情報のデータクレンジング（重複・不整合のチェック）

SQL Server（Microsoftのリレーショナルデータベース管理システム） へ人事・契約情報テーブルの移行

Access から SQL Server（Microsoftのリレーショナルデータベース管理システム） へのリンクテーブル更新

Access のクエリを SQL Server（Microsoftのリレーショナルデータベース管理システム） のストアドプロシージャに置き換え

SQL Server（Microsoftのリレーショナルデータベース管理システム） の負荷テスト（最大50名）を実施

アクセス管理（ユーザー権限設定）を強化

ユーザー向け研修を実施（SQL Server（Microsoftのリレーショナルデータベース管理システム） 版 Access の利用方法）

成果物

SQL Server（Microsoftのリレーショナルデータベース管理システム） へのデータ移行完了報告書

SQL Server（Microsoftのリレーショナルデータベース管理システム） のセキュリティポリシー

ユーザーマニュアル（SQL Server（Microsoftのリレーショナルデータベース管理システム） 版）

フェーズ4：SQL Server（Microsoftのリレーショナルデータベース管理システム） での本運用開始（4年目）

目的：SQL Server（Microsoftのリレーショナルデータベース管理システム） へ完全移行し、最適化・運用保守を開始。

タスク一覧

全データを SQL Server（Microsoftのリレーショナルデータベース管理システム） に移行

Access をフロントエンド（データ入力）として利用

Access のフォーム・レポートを SQL Server（Microsoftのリレーショナルデータベース管理システム） 対応に変更

SQL Server（Microsoftのリレーショナルデータベース管理システム） の監視システム導入（パフォーマンス監視・ログ管理）

定期バックアップ（システム障害時にデータを復旧できるよう、定期的に保存すること）・リカバリーテストの実施

本番環境でのトラブル対応ルール確立

移行後のサポート体制の構築

成果物

SQL Server（Microsoftのリレーショナルデータベース管理システム） の本番運用開始

運用・保守マニュアル

緊急対応フロー

10.3 移行に伴うリスクと対応策

リスク	対応策
データ移行時の不整合	移行前後のデータ検証を徹底（レコード数チェック・キー整合性確認）
SQL Server（Microsoftのリレーショナルデータベース管理システム） のパフォ	事前に負荷テストを実施し、インデックス最適化
Access から SQL Server（Microsoftのリレーショナルデータベース管理システ	ODBC 設定を最適化し、エラー発生時の対応手順を確立
ユーザーの操作ミス	研修を実施し、新しい操作フローを習得

10.4 まとめ

1年目：Access のデータ最適化と分散管理を導入

2年目：SQL Server（Microsoftのリレーショナルデータベース管理システム） のテスト運用を開始し、接続性を確認

3年目：人事・契約データを SQL Server（Microsoftのリレーショナルデータベース管理システム） に移行し、本格運用の準備

4年目：SQL Server（Microsoftのリレーショナルデータベース管理システム） へ完全移行し、本番運用を開始！

SQL Server 移行の技術検討について

1. SQL Server とは？

SQL Server は、Microsoft が開発した**リレーショナルデータベース管理システム（RDBMS）**です。

企業内のデータを管理・保存し、必要な情報を迅速に取得できるシステムを構築できます。

2. なぜ SQL Server の移行を検討するのか？

現在のデータベース環境におけるパフォーマンス向上・運用コスト削減・セキュリティ強化を目的に、移行を検討しています。

移行の技術ポイント

1 クラウド型 SQL Server の選択

選択肢: Microsoft Azure SQL Database など

メリット:

初期コストの削減（サーバーの物理設置が不要）

運用負担の軽減（自動バックアップ・セキュリティ強化）

スケーラビリティの向上（データ量の増加に柔軟対応）

2 ストアドプロシージャを活用したクエリ処理の最適化

ストアドプロシージャとは、データベースにあらかじめ保存した SQL コードのこと。

メリット:

クエリの処理速度を向上　　システム負荷の分散

セキュリティ強化（SQL インジェクション対策）

3 スケーラビリティを考慮したテーブル設計の見直し

スケーラビリティ: 将来的なデータ増加にも対応できる柔軟な設計

対応策:

適切なインデックス設計　　パーティション分割（大容量データを効率よく管理）

4 データアーカイブ機能の導入

目的: 古いデータを適切に管理し、データベースのパフォーマンスを維持

方法:

アクセス頻度の低いデータをアーカイブ領域へ移動　　必要に応じて復元できる仕組みを構築

まとめ

SQL Server の移行により、コスト削減・運用負担軽減・パフォーマンス向上が見込めます。

特に クラウド化 により、最新のセキュリティ対策とスケーラビリティの強化が可能となります。

この方針に基づき、詳細な技術検討を進めていきます。

「クラウド型 SQL Server」と「Microsoft Azure SQL Database」の関係について

1. SQL Server とは？

SQL Server とは、Microsoft が開発したデータベース管理システムです。

企業のシステム内で、データを保存・管理・検索するためのソフトウェアです。

顧客情報を保存し、検索・更新する

売上データを分析する

システム内のデータを整理・管理する

2. クラウド型 SQL Server とは？

SQL Server には、「オンプレミス型」（社内サーバー設置型）と**「クラウド型」**（インターネット経由で利用する型）があります。

方式	特徴
オンプレミス型（社内サーバー）	会社内に物理サーバーを設置し、管理・運用する。
クラウド型 SQL Server	インターネット上のサーバー（クラウド）で SQL Server を利用する。

3. Microsoft Azure SQL Database とは？

「Microsoft Azure SQL Database」は、Microsoft が提供するクラウド型 SQL Server の一種です。

簡単にいうと…

SQL Server をクラウドで利用できるサービス

Microsoft がすべてのサーバー管理をしてくれる

初期コストを抑えられ、運用も楽になる

4. クラウド型 SQL Server と Microsoft Azure SQL Database の関係

クラウド型 SQL Server ＝「SQL Server をクラウド上で使うこと」

Microsoft Azure SQL Database は、その代表的なサービスのひとつ

身近な例で例えると…

「クラウド型 SQL Server」= 車（クラウドで使えるデータベース全体）

「Microsoft Azure SQL Database」= トヨタの車（Microsoft が提供する特定のクラウド SQL）

「Amazon RDS for SQL Server」= 日産の車（Amazon が提供するクラウド SQL）

つまり、「クラウド型 SQL Server」は大きな概念であり、

その中に「Microsoft Azure SQL Database」**という具体的な選択肢がある、という関係です。**

5. なぜ Microsoft Azure SQL Database を検討するのか？

コスト削減: 自社サーバーを持たなくても利用可能

管理の手間が減る: Microsoft が運用・バックアップを管理

セキュリティ強化: 最新のセキュリティ対策が適用される

スケーラビリティ: データ量が増えても、簡単に対応できる

このため、企業のデータ管理の効率を向上させる目的で、「Microsoft Azure SQL Database」を選択肢として検討しています。

結論

「クラウド型 SQL Server」とは、SQL Server をクラウド上で使うこと全般を指す。

「Microsoft Azure SQL Database」は、その中でも Microsoft が提供する代表的なサービスの一つ。

選択する理由は、コスト削減・管理の簡略化・セキュリティ強化などのメリットがあるため。

パーティション分割（大容量データを効率よく管理）

必要に応じて復元できる仕組みを構築

	特徴
	会社内に物理サーバーを設置し、管理・運用する。
	インターネット上のサーバー（クラウド）で SQL Server を利用する。

Access 統合情報管理システム

本計画は 17部門の業務情報を統合・最適化 するための Microsoft Access と Excel を活用したシステム構築 に関する詳細な手順を示します。

1. 各部門のヒアリングと改善策の策定

目的： 17部門の業務内容・管理対象データを把握　　現行業務の課題と改善策を抽出　　業務フローの整理と統合ポイントの特定

ヒアリング項目　例

部門	ヒアリング内容	改善策
総務情報部門	勤怠管理、給与計算	勤怠データのリアルタイム入力
経理情報部門	経費精算、請求書発行	経理処理の自動化
営業情報部門	顧客管理、売上管理	顧客データの一元化
運用現場情報部門	シフト管理、出動記録	シフト最適化機能の追加
採用情報部門	応募者管理、面接スケジュール	採用試験結果のデータ管理
管制情報部門	指令・緊急対応	緊急指令のデータ連携
機械情報部門	設備点検、保守管理	保守履歴の統合管理
交通情報部門	交通誘導計画、規制情報	規制情報のデータ統合
施設情報部門	巡回管理、設備監視	巡回記録のデジタル化
集配情報部門	現金輸送管理	警備員割当の最適化
教育情報部門	研修・資格取得	資格更新の管理
営業所情報部門	各営業所の業務記録	営業所ごとのデータ統合
制服情報部門	制服貸与・在庫管理	在庫のリアルタイム管理
車両情報部門	車両管理・点検	車両の最適運用
プロジェクト情報部門	進捗管理・予算	プロジェクト進行状況の可視化
連絡情報部門	重要通達・連絡網	連絡情報の履歴保存
入札情報部門	入札案件・契約履歴	入札データの一元管理

2. 各部門の情報整理とテーブル設計

目的 ： 各部門の管理対象データを整理　　テーブル設計とフィールドの確定　　データ正規化の実施

共通ルール

リアル値 vs マスタ参照

変更の少ないデータ（例：部門、職種）→ マスタテーブルで管理

頻繁に変更されるデータ（例：顧客名、警備員名）→ 登録時にリアル値を保存

テーブル分割の原則

業務ごとに主テーブルを作成　　履歴データを分離　　検索用インデックスを適用

アクセス権の設定

部門ごとに閲覧・編集制限を設定

テーブル設計例（総務情報部門）

フィールド	データ型	説明
社員ID	AUTOINCREMENT	プライマリキー
氏名	TEXT	社員の氏名
部門ID	INTEGER	部門マスタ参照
勤務開始日	DATE	入社日
勤怠ID	AUTOINCREMENT	プライマリキー
社員ID	INTEGER	社員情報テーブル参照
出勤日時	DATETIME	出勤時間
退勤日時	DATETIME	退勤時間

全17部門でテーブル設計を統一し、正規化を適用する。

3. クエリ・フォーム・レポート・モジュール計画

テーブル確定後に実装計画を策定する

目的： システムの実装計画を策定 データの抽出・入力・出力機能を設計　　クエリ・フォーム・レポート・VBAモジュールの実装計画を立案

1 クエリ計画

クエリ名	目的
Q_社員情報検索	部門・氏名で社員情報を検索
Q_勤怠集計	社員ごとの勤怠データを月次集計
Q_売上分析	顧客ごとの売上データを集計

2 フォーム計画仕様

フォーム名	目的
F_社員管理	社員情報の登録・編集
F_勤怠入力	日次の勤怠データを入力
F_売上管理	営業案件の登録・更新

3 レポート計画仕様

レポート名	目的
R_勤怠一覧	社員ごとの月次勤怠レポート
R_売上推移	期間ごとの売上推移レポート

4 VBAモジュール計画仕様

モジュール	機能
M_入力制御	フォームの入力チェック
M_エクスポート	データをExcelに出力
M_インポート	ExcelデータをAccessに取り込み

4. 実施計画スケジュール

フェーズ	実施内容	期間
フェーズ1	ヒアリング・業務分析	1ヶ月
フェーズ2	テーブル設計・正規化	2ヶ月
フェーズ3	クエリ・フォーム設計	3ヶ月
フェーズ4	実装・テスト	3ヶ月
フェーズ5	本運用開始	1ヶ月

5. 具体的な実行内容

1 ユーザー権限設定

総務部門：給与データは総務担当のみ閲覧可

営業部門：営業成績は営業マネージャーのみ編集可

2 Excel連携

月次レポートをExcel自動出力

データインポート機能を実装

3 セキュリティ対策

データ改ざん防止 → 変更履歴をログに記録

アクセス制限 → VBAでログイン認証を実装

結論

1. 全17部門の業務情報を整理・統合
2. データの正規化を徹底し、最適なテーブル設計を実施
3. Access のクエリ・フォーム・レポートを設計
4. Excel との連携機能を構築
5. セキュリティ強化を考慮しながら、実装計画を立案

6. 主要な機能と技術 - 具体的な実装方法

1. データ管理機能

○ リアルタイム更新（変更履歴を記録）

【方法①】Access の変更履歴を記録する

Accessのテーブルに変更履歴を記録する方法として、「履歴テーブル」と「トリガーマクロ」を活用します。

1. 履歴テーブルの作成

変更履歴テーブル（ChangeLog）	旧値（テキスト型）
ID（主キー, AUTOINCREMENT）	新値（テキスト型）
変更対象テーブル名（テキスト型）	変更日時（DATETIME）
変更対象フィールド名（テキスト型）	変更者（テキスト型）

2. トリガーマクロの作成

Accessのマクロ機能 を利用して、データ更新時に ChangeLog に変更履歴を記録する。

BeforeUpdate イベントを使用し、変更前の値を保存。

【方法②】SQL Server のトリガーを利用

SQL Server にデータを移行した場合は、AFTER UPDATE トリガー を利用して履歴を保存できます。

sql

```
CREATE TRIGGER trg_UpdateLog
ON 顧客テーブル
AFTER UPDATE
AS
BEGIN
    INSERT INTO ChangeLog（変更対象テーブル名, 変更対象フィールド名, 旧値, 新値, 変更日時, 変更者）
    SELECT '顧客テーブル', '顧客名', d.顧客名, i.顧客名, GETDATE(), SUSER_NAME()
    FROM deleted d
    INNER JOIN inserted i ON d.ID = i.ID
    WHERE d.顧客名 <> i.顧客名;
END;
```

○ マスタ管理（顧客・施設・契約）

【方法】リレーションシップを利用したデータ管理 Access では リレーショナルデータベース を活用し、各マスタを適切に関連付ける。

1. マスタテーブル

顧客マスタ　　契約マスタ

施設マスタ　　スタッフマスタ

2. リレーション設定

顧客マスタ（1）→（多）契約マスタ

契約マスタ（1）→（多）勤務スケジュール

3. データの一貫性を保つ方法

入力制限（ルックアップウィザード）

参照整合性の強制（リレーションシップの制約）

2. セキュリティとログ管理

アクセス権限管理

SQL Server でユーザーごとに適切な権限を付与。

sql

```
CREATE LOGIN ユーザー名 WITH PASSWORD = 'パスワード';
CREATE USER ユーザー名 FOR LOGIN ユーザー名;
GRANT SELECT ON 顧客テーブル TO ユーザー名;
```

削除・更新履歴の保存

【方法】削除前に履歴テーブルへバックアップ 削除ボタンを押した際に、削除データを履歴テーブルにコピーする。

sql

```
CREATE TRIGGER trg_DeleteLog
ON 契約テーブル
AFTER DELETE
AS
BEGIN
    INSERT INTO DeleteLog（契約ID, 契約者, 削除日時）
    SELECT ID, 契約者, GETDATE() FROM deleted;
END;
```

定期バックアップと復元手順

【方法】SQL Server のバックアップ自動化

sql

コピーする編集する

```
BACKUP DATABASE 警備管理システム
TO DISK = 'C:¥Backups¥警備管理_バックアップ.bak'
WITH FORMAT, MEDIANAME = '警備管理バックアップ', NAME = 'フルバックアップ';
```

sql

```
RESTORE DATABASE 警備管理システム
FROM DISK = 'C:¥Backups¥警備管理_バックアップ.bak'
WITH REPLACE;
```

3. 業務自動化

定型レポートの自動生成

【方法①】Access のマクロを利用

マクロの「エクスポート」機能を利用し、レポートをPDFやExcelに出力。

VBAを活用し、指定した日時に自動生成。

vba

```
DoCmd.OutputTo acOutputReport, "契約一覧レポート", acFormatPDF, "C:¥レポート¥契約一覧.pdf"
```

VBA を活用したデータ入力補助

【方法①】Access のデータ入力フォームに VBA を追加

フォームの「未入力チェック」

フォームの「オートコンプリート機能」

vba

コピーする編集する

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    If IsNull(Me.顧客名) Then
        MsgBox "顧客名を入力してください", vbExclamation
        Cancel = True
    End If
End Sub
```

エラー検出と通知システム

【方法①】SQL Server のエラーログを監視

sql

```
SELECT * FROM sys.messages WHERE severity >= 16;
```

【方法②】Access VBA でエラーメッセージを送信

vba

```
Private Sub Form_Error(DataErr As Integer, Response As Integer)
    MsgBox "エラーが発生しました。エラーコード:" & DataErr, vbCritical
    Response = acDataErrContinue
End Sub
```

まとめ

機能	具体的な実装方法
リアルタイム更新	Access マクロ / SQL Server トリガー
マスタ管理	リレーションシップ設定、ルックアップ
アクセス権限	SQL Server ログイン管理、VBA
削除履歴の保存	履歴テーブル、AFTER DELETE トリガー
定期バックアップ	SQL Server のスケジュールバックアップ
定型レポート	Access VBA / SQL クエリ
エラー検出	SQL ログ監視 / VBA メッセージボックス

7. システムの使い方（操作フロー） – 具体的なイメージ

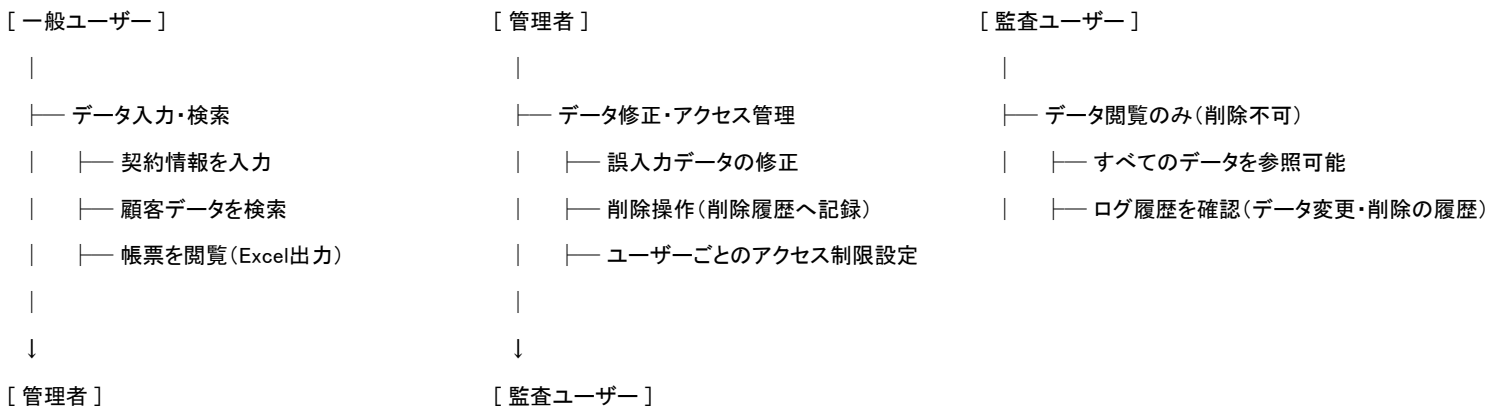
1. ユーザー別の操作フロー

操作イメージ

以下の3つのユーザーグループに分かれ、それぞれ権限が異なります。

操作可能な内容	具体的な操作イメージ
データ入力・検索	データ入力フォームで新規登録
	検索フィールドで契約情報を検索
	帳票（Excel出力）を閲覧
データ修正・アクセス管理	データの編集・更新（誤入力修正）
	削除操作（削除履歴へ記録）
	ユーザーごとのアクセス制限を設定
データ閲覧のみ（削除不可）	すべてのデータを参照可能
	ログ履歴を確認（データ変更・削除の履歴）

操作フロー図（イメージ）



2. データ登録・更新時の注意点

データの正確性を保つため、入力時のチェック機能 を導入します。

入力ミス防止のためのチェック機能

数値のみ許可する入力制限（例: 電話番号）

必須項目の未入力防止

入力データの形式チェック

例: YYYY-MM-DD 形式の日付チェック

例: 顧客ID は6桁の数値のみ

データ重複チェック

例: 既に存在する顧客名で登録できない

VBA を活用したエラーチェック（Access）

vba

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    If IsNull(Me.顧客名) Then
        MsgBox "顧客名を入力してください", vbExclamation
        Cancel = True
    End If
End Sub
```

変更履歴の自動保存

Access のマクロ/SQL Server のトリガーを利用

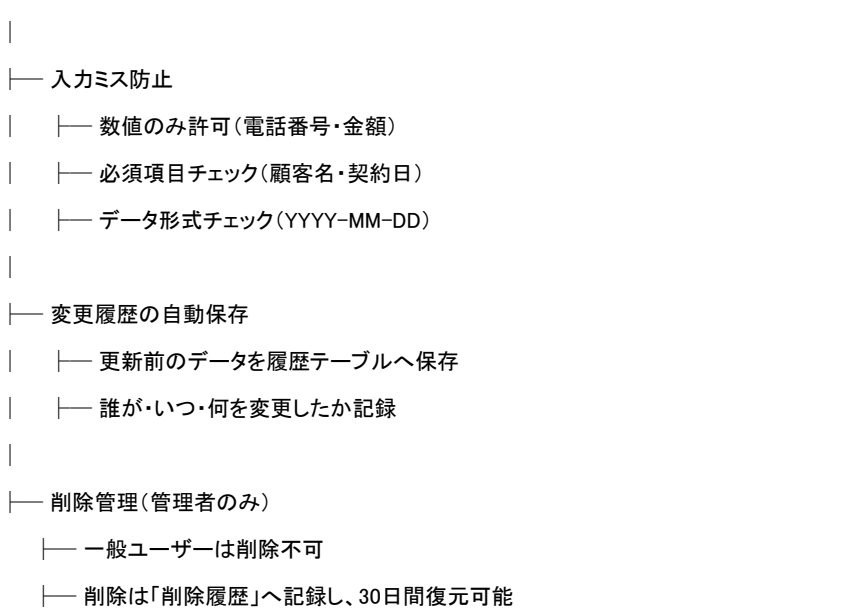
データ変更時に旧データを履歴テーブルへ保存

誰が・いつ・どのデータを変更したか記録

SQL Server の AFTER UPDATE トリガー

sql

```
CREATE TRIGGER trg_UpdateLog
ON 顧客テーブル
AFTER UPDATE
AS
BEGIN
    INSERT INTO ChangeLog (変更対象テーブル名, 変更対象フィールド名, 旧値, 新値, 変更日時, 変更者)
    SELECT '顧客テーブル', '顧客名', d.顧客名, i.顧客名, GETDATE(), SUSER_NAME()
    FROM deleted d
    INNER JOIN inserted i ON d.ID = i.ID
    WHERE d.顧客名 <> i.顧客名;
END;
```



削除は管理者のみ可能（30日間復元可能）

削除時のフロー

1. 一般ユーザーは削除不可
2. 管理者が削除ボタンを押す
3. データを「削除済みリスト」に移動（実際には削除しない）
4. 30日経過後に完全削除

SQL Server の削除トリガー（削除データをバックアップ）

```
sql
CREATE TRIGGER trg_DeleteLog
ON 契約テーブル
AFTER DELETE
AS
BEGIN
    INSERT INTO DeleteLog (契約ID, 契約者, 削除日時)
    SELECT ID, 契約者, GETDATE() FROM deleted;
END;
```

VBA を活用した削除防止

```
vba
Private Sub btnDelete_Click()
    If CurrentUser <> "管理者" Then
        MsgBox "削除権限がありません", vbCritical
        Exit Sub
    End If
    ' 削除処理の実行
End Sub
```

まとめ

機能	具体的な操作方法
一般ユーザーの操作	データ入力・検索・帳票閲覧
管理者の操作	データ編集・削除・ユーザー管理
監査ユーザーの操作	データ閲覧・ログ履歴の確認
入力ミス防止	数値チェック・必須項目の確認・データ重複防止
変更履歴の保存	SQL Server の AFTER UPDATE トリガー
削除管理	30日間削除ログ保存・管理者のみ削除可能

8.データの安全管理（バックアップ・トラブル対策） – 具体的なイメージ

1. バックアップポリシー

データのバックアップは「3階層」で管理

バックアップは、データ損失を防ぐために デイリー・ウィークリー・月次の3段階 で実施します。

バックアップの流れ

[デイリーバックアップ]（毎日）

|

├── ローカルPC / サーバー上で自動バックアップ

|

[ウィークリーバックアップ]（毎週）

|

├── SQL Serverのバックアップスケジュール設定

|

├── バックアップファイルを専用のサーバーに転送

|

[月次フルバックアップ]（毎月）

|

├── 外部ストレージ（クラウド / NAS）へ保存

|

├── 物理媒体（HDD / テープ）にも定期保存

具体的な設定方法

1. **SQL Server の自動バックアップ**

   SQL Server で自動バックアップをスケジュール

   sql

   BACKUP DATABASE 警備管理システム

   TO DISK = 'C:¥Backups¥警備管理_バックアップ.bak'

   WITH FORMAT, MEDIANAME = '警備管理バックアップ', NAME = 'フルバックアップ';

2. **バックアップファイルの自動転送（バッチ処理）**

   定期的にサーバーへコピーするバッチスクリプト

   batch

   xcopy "C:¥Backups¥*.bak" "¥¥BackupServer¥SQL_Backups¥" /D /E /C /Y

2. トラブル発生時の対応

システム障害発生時の手順

障害が発生した際の対応をフロー化し、迅速な復旧ができるようにします。

システム障害時の対応フロー

[システム障害発生]

|

├── 影響範囲を確認（特定のPCか、全体の問題か）

|

├── エラーログを確認（SQL Server / Access のログ）　　├── 改善しない場合、IT担当者に連絡

|　　|

├── 一時対応（再起動・ネットワークチェック）　　├── 必要に応じてバックアップから復旧

|

データ破損時の復旧方法

データが破損した場合は、以下の手順で復旧を行います。

データ復旧手順

[データ破損検出]

|

├── SQL Server のログを確認（エラーメッセージ分析）

|

├── Access ファイルが破損している場合「修復ツール」を実行

|

├── 最新のバックアップからデータを復元

|

├── 復旧後、データの整合性チェック

SQL Server でデータ復旧コマンド

```
RESTORE DATABASE 警備管理システム
FROM DISK = 'C:¥Backups¥警備管理_バックアップ.bak'
WITH REPLACE;
```

サポート窓口（社内 IT 担当者の連絡先）

障害が発生した際に迅速に対応できるよう、**IT担当者の連絡先を明確に**しておきます。

障害発生時の連絡フロー

[システムトラブル発生]

|

├── 社内 IT 担当者に連絡

|

├── 影響範囲を確認し、対応方針を決定

|

├── 必要に応じて外部ベンダー（サポート会社）に問い合わせ

まとめ

項目	具体的な対策
デイリーバックアップ	ローカルサーバー / 自動バックアップ
ウィークリーバックアップ	SQL Server のスケジュールバックアップ
月次フルバックアップ	クラウド / 外部ストレージ保存
システム障害対応	影響範囲の確認・エラーログ解析・IT担当者へ連絡
データ破損復旧	バックアップ復元 / Access 修復ツール
サポート窓口	IT担当者・外部ベンダーの連絡先管理

Access とExcel 連携

1. Access × Excel で実現できること

機能	内容
データの自動集計	Access でデータを蓄積し、Excel でリアルタイム分析
入力作業の自動化	Excel のフォームを使い、Access に直接データ入力
帳票作成の自動化	Access のデータを元に Excel でレポートを瞬時に作成
ワークフローの自動化	Access のデータを基に Excel で業務プロセスを自動処理
予測・シミュレーション	Access で蓄積したデータを Excel でAI分析
グラフ・可視化	Access のデータを Excel でダッシュボード化

2. Access × Excel 連携の革新的な活用例

1 リアルタイム「勤怠管理システム」

出勤・退勤の入力をリアルタイムで管理し、給与計算まで自動化！

具体的な仕組み

Access に社員の勤怠データを蓄積

Excel に「出勤・退勤ボタン」を配置（VBAで Access に直接登録）

Excel で残業時間を自動計算し、給与計算表を作成

Access のデータをもとに、月次勤怠報告書を自動作成

ポイント！

ICカード・QRコード での打刻データを Access に記録し、Excel に自動連携

「勤怠異常検知システム」→ 一定時間以上の勤務を検出して警告

月間の勤怠データをPowerPointへ自動出力 し、役員会議に即時活用

2 顧客分析 × AI予測システム

売上データを Access で蓄積し、Excel でAI分析を行い未来予測！

具体的な仕組み

Access に顧客データ（売上・取引履歴）を保存

Excel の「Power Query」で Access のデータをリアルタイム取得

Excel の「予測シート機能」で 次月の売上をAI分析

「この顧客はリピート率が高い」と自動判定し、営業部に通知

ポイント！

売上予測 AI モデル を Excel に組み込み、未来の売上トレンドを可視化

過去の取引履歴を分析 し、VIP顧客を自動抽出し、特別割引を提案

顧客の購買パターンを検出し、離反リスクを自動通知

3 パトロール最適化 × Googleマップ連携

Access で警備ルートを管理し、Excel × Google マップ で最適ルートを提案！

具体的な仕組み

Access に警備現場の位置情報を登録

Excel で「パトロールルート最適化ボタン」を押すと 最短ルートを自動計算

GoogleマップAPIを利用し、リアルタイムで地図を表示

最適ルートを Excel の「地図グラフ」で可視化

ポイント！

交通渋滞を考慮したルート最適化 により移動時間を短縮

異常発生エリアのリアルタイム警告（Access に警備記録を登録 → Excel で異常検知）

Excel でルートを編集し、Access に自動反映（ルート変更が即座に記録される）

4 制服管理 × 自動発注システム

Access で貸与管理し、Excel で不足分をAI分析して自動発注！

具体的な仕組み

Access に「制服の貸出状況」を記録

Excel で「在庫不足アラート」を自動表示

AI が「発注量」を予測し、仕入れ担当者へメール送信

Excel で貸与履歴を可視化し、「貸し出し回数が多い制服は買い替え推奨」

ワクワクポイント！

貸与期限が切れる前に自動アラート を出し、回収漏れを防止

発注データを Excel から Access に自動入力（発注ミス削減）

貸与データをPower BI で可視化し、コスト削減の分析

5 車両管理 × ドライブレコーダー分析

車両の走行データを Access に蓄積し、Excel で運転傾向を分析！

具体的な仕組み

Access に「車両の走行データ・燃費データ」を保存

Excel で「危険運転判定」を自動実行

急ブレーキ・急加速を検出し、ドライバー評価を作成

事故率の高いルートを特定し、回避ルートを提案

ワクワクポイント！

運転の安全評価レポートを自動作成し、研修に活用

事故発生地点をマップで可視化し、安全ルートを提案

燃費の悪い車両を自動検出し、整備スケジュールを提案

3. 未来の活用

機能	内容
音声認識×データ入力	音声で指示を出すと Access にデータを入力
Excel Chatbot 連携	AIが Access のデータを元に質問に回答
Access × LINE 連携	Access から LINE へ業務通知を自動送信
リアルタイムBI分析	Power BI で業務データをダッシュボード化
IoTデバイス連携	センサーからデータを Access に蓄積し、異常検知

まとめ

Microsoft Access と Excel を連携させることで、業務の効率化、自動化、分析、予測、可視化 まで幅広い活用が可能です！

単なるデータ管理ではなく、ワクワクするシステムを構築し、未来の業務を変革 することができます。

これらの活用例をベースに、具体的な業務要件に合わせて さらにワクワクする機能 を追加していくことができます！🚀✨

4. Access × Excel 連携による警備保障会社向け革新的システム開発目標

本仕様書は、警備保障会社の現状を正確に把握し、それをどのように改善するかを明記する ことを目的としています。

そのため、Excel との連携機能については、具体的な実装手法に縛られるのではなく、

警備業界に必要な内容、および革新的なシステム開発の方向性を示す形で掲載します。

1. システム開発の基本方針

本システムは、Microsoft Access を中心にデータを統合し、Excel を補助ツールとして活用 することで、以下の3つの柱を軸に開発を進めます。

1. 現状の問題を解決

情報の分散管理による非効率化を改善（データを一元管理）

手作業の多い業務を自動化（Excel の自動処理を活用）

リアルタイムな業務状況の把握（ダッシュボード化）

2. 革新的な機能を開発

警備業界特有の課題を解決するシステムを構築

データの有効活用（AI分析、BIツール連携）

IoTデバイス・GPS・クラウドとの連携を視野に入れる

3. 未来を見据えたシステム設計

SQL Server への移行を視野に入れたスケーラブルな構築

業務の拡張に柔軟に対応できるデータ構造を採用

全社的なデジタル化を進めるための基盤を整備

2. 各部門での改革目標と開発方針

本システムは 警備業界の特性に最適化された機能を実装 することで、業務を効率化し、組織の生産性を向上させることを目指します。

以下に、各部門での改革目標と開発方針 を示します。

1. 総務情報部門

改革目標

勤怠管理の自動化（出勤・退勤のリアルタイム管理）

給与計算の最適化（勤怠データと給与計算の連携）

開発方針

ICカードやスマホ打刻 を利用した勤怠記録システム

勤務時間・残業時間のリアルタイム自動集計

給与データを自動算出し、Excel で帳票作成

2. 警備管制情報部門

改革目標

緊急対応の迅速化（警備員配置の最適化）

出動履歴のデジタル管理

開発方針

リアルタイムGPS追跡による警備員の位置管理

異常発生時に最適な警備員を自動割当て

出動ログを自動記録し、緊急報告書をExcel に自動出力

3. 機械警備情報部門

改革目標

設備点検・保守管理の効率化

保守履歴のデジタル化

開発方針

設備点検のスケジュールを自動作成し、Excel に通知

点検報告書をスマホから入力し、Access に自動保存

故障発生時に修理手配を自動化（担当者に通知）

4. 集配警備情報部門

改革目標

現金輸送ルートの最適化

輸送記録のデジタル化

開発方針

輸送ルートをAIで最適化し、Googleマップに自動反映

輸送記録をタブレット入力し、Access でリアルタイム更新

輸送中の異常検知時に即座に本部へ警告

5. 教育管理部門

改革目標

資格取得・研修管理の自動化

教育履歴のデジタル管理

開発方針

資格の有効期限を自動管理し、更新アラートをExcel に通知

研修の出欠記録をリアルタイム登録し、履歴を可視化

教育データをAccess で管理し、レポートをExcel に出力

6. 車両管理部門

改革目標

警備車両の利用最適化

運転履歴のデータ分析

開発方針

車両の運行履歴をGPSと連携して自動記録

急ブレーキ・急加速を検出し、安全運転レポートを作成

燃費や整備履歴を分析し、適切な車両管理を実施

7. 革命的なシステム開発目標

警備業界のDX（デジタルトランスフォーメーション）を促進するために、次のようなシステムを目標に開発を進めます。

目標	開発方針
リアルタイム警備管理	GPSと連携し、警備員の位置と勤務状況を即時確認
AIを活用した警備ルート最適化	交通情報や警備記録を分析し、最適なルートを提案
異常発生時の自動警告システム	緊急時に担当者へ即時アラートを発信
予測分析で業務の最適化	売上・出動履歴・勤怠データをAI分析し、未来予測
IoT機器との連携	防犯カメラ・センサーとシステムを統合し、異常検知
スマホアプリ連携	現場警備員がモバイルから報告できるシステムを構築

4. まとめ

本システムは、Access を中核に据え、Excel を活用しながら警備業務を効率化・最適化 することを目的としています。

また、AI・IoT・クラウド技術との連携を視野に入れ、警備業界に革新をもたらすシステム を構築します。

運用・管理

1. **システムのメンテナンス手順**

   定期バックアップのスケジュール（SQL Server, Access, Excel）

   データアーカイブの実施方法（履歴テーブルの活用）

   データ整理の基準（3年以上のデータはアーカイブへ移動）

2. **アクセス権限管理**

   ユーザー別のアクセス制限（一般ユーザー・管理者・監査）

   閲覧専用ユーザーと編集可能ユーザーの区別

   監査ログの管理（変更履歴の保持期間）

3. **ファイルサーバーとデータ連携**

   契約書・報告書の格納ルール

   保存フォルダの命名規則

   権限管理（特定部門のみ閲覧可能にする設定）

総務管理部門のデータベース設計

1. 計画概要

総務管理部門では、従業員情報の管理を中心に、勤怠、福利厚生、文書管理を統合的に扱います。

データの整合性を確保し、過去の情報を適切に追跡するため、履歴管理を導入し、固定情報はマスタ化します。

また、リアル値を直接格納することで、直感的なデータ管理を実現します。

2. 設計方針

1. リアル値の管理

部署名、役職名、雇用形態などはIDではなく、直接リアル値を格納、外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

勤怠情報や福利厚生情報も履歴として保持し、過去データの参照を可能にする

3. 固定情報のマスタ化

役職や部署、雇用形態など、頻繁に変更されない情報はマスタテーブルで統一

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

履歴情報はメインフォームのサブフォームとして可視化し、変更履歴の確認を容易にする

3. 仕様

データ管理

従業員情報は「基本情報」と「履歴情報」に分類　　文書データはバージョン管理を適用

勤怠データや福利厚生データは履歴テーブルを利用　　リアル値を直接格納し、外部キーによる参照を不要とする

運用方法　　マスタ情報を導入し、データの統一性を担保

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

4 メイン台帳テーブル（リアル値仕様）

従業員マスタ

従業員ID	数値型	各従業員を識別する一意のID
氏名	文字列型	フルネーム
フリガナ	文字列型	ふりがな
生年月日	日付型	生年月日
性別	文字列型	男性 / 女性 / その他
退職日	日付型	退職者のみ記録
部署名	文字列型	所属する部署名（リアル値）
役職名	文字列型	役職名（リアル値）
雇用形態	文字列型	正社員 / 契約社員 / パート など（リアル値）

5 サブ情報明細テーブル（履歴情報の管理）

勤怠履歴テーブル

項目名	データ型	説明
勤怠ID	数値型	勤怠データの一意ID
従業員ID	数値型	従業員マスタと関連（外部キー）
出勤日	日付型	出勤日
出勤時間	時間型	出勤した時間
退勤時間	時間型	退勤した時間
シフト	文字列型	シフト名（リアル値）
休憩時間	数値型	休憩時間（分）
実働時間	数値型	勤務時間 ー 休憩時間

福利厚生履歴テーブル

項目名	データ型	説明
福利厚生ID	数値型	福利厚生データの一意ID
従業員ID	数値型	従業員マスタと関連（外部キー）
健康診断日	日付型	受診日
健康診断結果	文字列型	異常なし / 要精密検査
退職金制度	文字列型	加入 / 未加入
福利厚生利用状況	文字列型	ジム利用 / 保養所利用 など

6 正規化後のテーブル生成SQL

```
CREATE TABLE 従業員マスタ (
    従業員ID AUTOINCREMENT PRIMARY KEY,
    氏名 TEXT NOT NULL,
    フリガナ TEXT NOT NULL,
    生年月日 DATETIME,
    性別 TEXT,
    入社日 DATETIME,
    退職日 DATETIME,
    部署名 TEXT NOT NULL,
);
```

```
CREATE TABLE 勤怠履歴 (
    勤怠ID AUTOINCREMENT PRIMARY KEY,
    従業員ID INTEGER NOT NULL,
    出勤日 DATETIME NOT NULL,
    出勤時間 DATETIME,
    退勤時間 DATETIME,
    シフト TEXT,
    残業時間 INTEGER,
    休憩時間 INTEGER,
    実働時間 INTEGER
);
```

```
CREATE TABLE 福利厚生履歴 (
    福利厚生ID AUTOINCREMENT PRIMARY KEY,
    従業員ID INTEGER NOT NULL,
    健康診断日 DATETIME,
    健康診断結果 TEXT,
    退職金制度 TEXT,
    福利厚生利用状況 TEXT
);
```

7 注意点およびポイント

リアル値（部署名、役職名、雇用形態）を従業員マスタに格　Accessのリレーションシップは使用せず、VBAやクエリで関連付け

履歴情報（勤怠、福利厚生など）は履歴テーブルとして管理　変更履歴は履歴テーブルに記録し、サブフォームで可視化

固定情報（役職、部署など）はマスタ化

8 クエリ設計

9 フォーム設計

10 リポート設計

11 モジュール設計

経理管理部門のデータベース設計

1 計画概要

経理管理部門では、財務管理、支払管理、経費精算、税務申告、資産管理を統合的に管理します。

データの正確性と整合性を確保し、履歴管理とマスタ管理を適用することで、過去の財務データを適切に追跡できるように設計します。

また、リアル値を直接格納し、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

勘定科目や支払方法などのマスタ情報はリアル値で管理し、ID参照を最小限に外部キーの使用を減らし、シンプルなデータ管理を実現

2. 履歴管理の導入

月次ごとの財務データを履歴として保持し、過去の財務状況を追跡可能に

3. 固定情報のマスタ化

取引先、勘定科目、資産種類などの固定情報はマスタ化し、統一されたデータを管理

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用　　財務データや経費精算履歴は、メインフォームとサブフォームで可視化

3 仕様

データ管理

月次ごとの財務情報は履歴テーブルを活用し、過去データの参照を可能に税務申告データは、年度ごとに履歴化し、過去の申告情報を保持

データ構造

主要な財務情報は、財務台帳テーブルに格納

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携　　サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

財務管理テーブル

項目名	データ型	説明
財務ID	数値型	各財務データの一意ID
年度	数値型	会計年度（例：2025）
収入総額	数値型	その月の全収入
支出総額	数値型	その月の全支出
粗利益	数値型	収入総額 － 支出総額
貸借対照表URL	文字列型	B/S（貸借対照表）のリンク

5 サブ情報明細テーブル（履歴情報の管理）

支払管理テーブル

項目名	データ型	説明
支払ID	数値型	支払データの一意ID
従業員名	文字列型	支払対象の従業員名（リアル値）
支払年月	数値型	支払年・月（例：202501）
基本給	数値型	基本給金額
残業代	数値型	残業手当
控除額	数値型	社会保険料、税金などの控除
振込口座番号	文字列型	給与振込口座（暗号化推奨）
支払日	日付型	支払日

経費精算履歴テーブル

項目名	データ型	説明
経費ID	数値型	経費精算データの一意ID
申請者名	文字列型	申請した従業員名（リアル値）
使用日	日付型	経費を使用した日
経費金額	数値型	経費精算額
経費種類	文字列型	交通費 / 接待費 / 備品購入 など（リアル値）
領収書URL	文字列型	領収書データのリンク
承認者名	文字列型	承認担当者名（リアル値）

6 正規化後のテーブル生成SQL

```
CREATE TABLE 財務管理 (
    財務ID AUTOINCREMENT PF
    年度 INTEGER NOT NULL,
    月次 INTEGER NOT NULL,
    収入総額 INTEGER NOT NUL
    支出総額 INTEGER NOT NUL
    粗利益 INTEGER NOT NULL,
    貸借対照表URL TEXT,
    損益計算書URL TEXT
);
```

```
CREATE TABLE 支払管理 (
    支払ID AUTOINCREMENT PRIMARY KE
    従業員名 TEXT NOT NULL,
    支払年月 INTEGER NOT NULL,
    基本給 INTEGER NOT NULL,
    残業代 INTEGER,
    交通費 INTEGER,
    控除額 INTEGER,
    振込口座番号 TEXT,
    支払日 DATETIME
);
```

```
CREATE TABLE 経費精算履歴 (
    経費ID AUTOINCREMENT PRIMARY KEY,
    申請者名 TEXT NOT NULL,
    使用日 DATETIME NOT NULL,
    経費金額 INTEGER NOT NULL,
    経費種類 TEXT NOT NULL,
    領収書URL TEXT,
    精算日 DATETIME,
    承認者名 TEXT NOT NULL
);
```

7 注意点およびポイント

リアル値（従業員名、経費種類など）をテーブルに直接格納

財務データや経費精算履歴は履歴テーブルとして管理

Accessのリレーションシップは使用せず、VBAやクエリで関連付け

変更履歴は履歴テーブルに記録し、サブフォームで可視化

固定情報（勘定科目、支払方法など）はマスタ化

8 クエリ設計

9 フォーム設計

10 リポート設計

11 モジュール設計

営業管理部門のデータベース設計

1 計画概要

営業管理部門では、顧客情報、案件管理、見積作成、契約管理 を統合的に管理し、営業活動の効率化とデータの一元管理を実現します。

特に、見積作成は必須機能とし、過去の見積履歴や承認フローを適切に管理できるように設計します。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

顧客名、取引先名、営業担当者名、見積項目名などはID管理ではなく、リアル値で格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

案件進捗や見積履歴を履歴テーブルで管理し、過去の取引情報を追跡可能に見積書のバージョン管理を導入し、修正履歴を保持

3. 固定情報のマスタ化

営業ステータス、契約種別、サービス種別などの固定情報はマスタ化

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

重要な履歴情報（見積履歴、契約履歴）はメインフォームのサブフォームとして可視化

3 仕様

データ管理

顧客情報は最新データを保持し、過去の取引履歴は履歴テーブルで管理

案件情報、見積履歴、契約履歴を別々のテーブルとして管理し、履歴管理を徹底

データ構造

営業活動の基盤となるデータを営業管理台帳に格納

見積書、案件進捗、契約履歴などをサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

営業管理台帳

項目名	データ型	説明
顧客名	文字列型	取引先の顧客名（リアル値）
営業担当者名	文字列型	担当営業の氏名（リアル値）
案件名	文字列型	案件の名称
案件終了日	日付型	案件の完了日
案件ステータス	文字列型	進行中 / 受注 / 失注 など（リアル値）
見積書番号	文字列型	最新の見積書番号（リアル値）

5 サブ情報明細テーブル（履歴情報の管理）

見積履歴テーブル

項目名	データ型	説明
見積ID	数値型	見積データの一意ID
見積書番号	文字列型	見積書の識別番号（バージョン管理用）
作成日	日付型	見積書の作成日
有効期限	日付型	見積の有効期限
見積金額	数値型	見積書の合計金額
承認担当者	文字列型	見積書の承認者（リアル値）
ステータス	文字列型	承認待ち / 承認済み / 失注 など

契約履歴テーブル

項目名	データ型	説明
契約ID	数値型	契約データの一意ID
案件ID	数値型	案件管理台帳と関連（外部キー）
契約書番号	文字列型	契約書の識別番号
契約開始日	日付型	契約の開始日
契約金額	数値型	契約総額
支払条件	文字列型	一括 / 分割 など
契約担当者	文字列型	契約処理を担当した社員名（リアル値）

6 正規化後のテーブル生成SQL

```
CREATE TABLE 営業管理 (
    案件ID AUTOINCREMENT PRIMARY KEY,
    顧客名 TEXT NOT NULL,
    営業担当者名 TEXT NOT NU
    案件名 TEXT NOT NULL,
    案件開始日 DATETIME,
    案件終了日 DATETIME,
    案件ステータス TEXT,
    契約状況 TEXT
);
```

```
CREATE TABLE 見積履歴 (
    見積ID AUTOINCREMENT PRIMARY KEY,
    案件ID INTEGER NOT NULL,
    見積書番号 TEXT NOT NULL,
    作成日 DATETIME NOT NULL,
    有効期限 DATETIME NOT NULL,
    見積金額 INTEGER NOT NULL,
    見積詳細 TEXT,
    承認担当者 TEXT,
    ステータス TEXT
);
```

```
CREATE TABLE 契約履歴 (
    契約ID AUTOINCREMENT PRIMARY KEY,
    案件ID INTEGER NOT NULL,
    契約書番号 TEXT NOT NULL,
    契約開始日 DATETIME NOT NULL,
    契約終了日 DATETIME NOT NULL,
    契約金額 INTEGER NOT NULL,
    支払条件 TEXT,
    契約担当者 TEXT
);
```

7 注意点およびポイント

リアル値（顧客名、営業担当者名、案件ステータスなど）を
見積履歴はバージョン管理を導入し、過去の見積を追跡
契約履歴は契約締結後もデータが残るように履歴テーブ

固定情報（案件ステータス、契約種別、支払条件など）はマスタ化
Accessのリレーションシップは使用せず、VBAやクエリで関連付け
変更履歴は履歴テーブルに記録し、サブフォームで可視化

8 クエリ設計

9 フォーム設計

10 リポート設計

11 モジュール設計

運用現場管理部門のデータベース設計

1 計画概要

運用現場管理部門では、警備配置、業務報告、緊急対応 などの業務を統合的に管理します。

警備員の配置、出勤・退勤、業務内容の記録、異常発生時の対応履歴を適切に管理し、業務の透明性を向上させます。

リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

警備員名、担当現場名、業務内容などはIDではなく、直接リアル値を格納し外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

警備員の勤務履歴、異常発生履歴、巡回報告などは履歴テーブルで管理

異常発生時の対応内容は詳細に記録し、後の検証に活用できるようにする

3. 固定情報のマスタ化

警備業務種別、異常種別、対応ステータスなどはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

警備員の勤務履歴、巡回報告、異常発生履歴は、メインフォームのサブフォームとして可視化

3 仕様

データ管理

警備員の勤務履歴、業務内容、異常発生記録を履歴テーブルで管理

異常発生時の対応履歴を記録し、過去の対応履歴を追跡可能に

業務報告を電子化し、業務内容をリアルタイムで記録できるようにする

データ構造

現場ごとの警備業務を「運用現場管理台帳」に記録

警備員の勤務履歴、異常発生履歴、業務報告をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

運用現場管理台帳

現場ID	数値型	各警備現場を識別する一意のID
現場名	文字列型	警備を担当する現場名（リアル値）
顧客名	文字列型	取引先の顧客名（リアル値）
担当警備員	文字列型	担当する警備員の氏名（リアル値）
勤務日	日付型	勤務日
勤務時間	時間型	勤務時間（開始～終了）
業務内容	文字列型	巡回 / 受付 / 監視 など（リアル値）
異常発生	文字列型	あり / なし
報告書URL	文字列型	業務報告書のリンク

5 サブ情報明細テーブル（履歴情報の管理）

勤務履歴テーブル

項目名	データ型	説明
勤務ID	数値型	勤務データの一意ID
現場ID	数値型	運用現場管理台帳と関連（外部キー）
警備員名	文字列型	勤務した警備員名（リアル値）
出勤時間	時間型	出勤した時間
退勤時間	時間型	退勤した時間
業務内容	文字列型	巡回 / 受付 / 監視 など（リアル値）

異常発生履歴テーブル

項目名	データ型	説明
異常ID	数値型	異常発生データの一意ID
現場ID	数値型	運用現場管理台帳と関連（外部キー）
発生日時	日付型	異常が発生した日時
異常内容	文字列型	侵入検知 / 火災警報 / 設備異常 など（リアル値）
対応内容	文字列型	警察連絡 / 消火対応 など（リアル値）
担当警備員	文字列型	異常対応を担当した警備員名（リアル値）

6 正規化後のテーブル生成SQL

```
CREATE TABLE 運用現場管理 (
    現場ID AUTOINCREMENT PRIMARY KEY,
    現場名 TEXT NOT NULL,
    顧客名 TEXT NOT NULL,
    担当警備員 TEXT NOT NULL
    勤務日 DATETIME NOT NULL
    勤務時間 TEXT NOT NULL,
    業務内容 TEXT NOT NULL,
    異常発生 TEXT,
    報告書URL TEXT
);
```

```
CREATE TABLE 勤務履歴 (
    勤務ID AUTOINCREMENT PRIMARY KEY,
    現場ID INTEGER NOT NULL,
    警備員名 TEXT NOT NULL,
    出勤時間 DATETIME NOT NULL,
    退勤時間 DATETIME NOT NULL,
    業務内容 TEXT NOT NULL
);
```

```
CREATE TABLE 異常発生履歴 (
    異常ID AUTOINCREMENT PRIMARY KEY,
    現場ID INTEGER NOT NULL,
    発生日時 DATETIME NOT NULL,
    異常内容 TEXT NOT NULL,
    対応内容 TEXT NOT NULL,
    担当警備員 TEXT NOT NULL
);
```

7 注意点およびポイント

リアル値（警備員名、現場名、業務内容など）を直接格納　　固定情報（異常種別、対応内容など）はマスタ化

警備員の勤務履歴、異常発生履歴を履歴テーブルで管理　　Accessのリレーションシップは使用せず、VBAやクエリで関連付け

変更履歴は履歴テーブルに記録し、サブフォームで可視化

8 クエリ設計

9 フォーム設計

10 リポート設計

11 モジュール設計

人材採用管理部門のデータベース設計

1 計画概要

人材採用管理部門では、応募者管理、試用期間評価、採用計画、離職分析 を統合的に管理します。

応募者の基本情報、選考プロセスの履歴、試用期間の評価、採用後の離職分析を適切に管理し、採用戦略の改善に活用できるように設計します。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

応募者名、採用担当者名、選考ステータスなどはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

応募者の選考プロセス（書類審査、面接、内定）を履歴として記録

試用期間中の評価を記録し、採用の適正性を分析できるようにする

3. 固定情報のマスタ化

選考ステータス、職種、面接官情報などの固定情報はマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

応募者履歴、試用期間評価、離職分析データはメインフォームのサブフォームとして可視化

3 仕様

データ管理

応募者情報、選考ステータス、試用期間の評価を履歴テーブルで管理

選考プロセスの段階ごとに履歴を記録し、合格・不合格の判断を明確に

採用後の試用期間評価を記録し、長期的な人材定着率を分析

データ構造

応募者情報を「人材採用管理台帳」に記録

選考履歴、試用期間評価、離職分析をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

人材採用管理台帳

項目名	データ型	説明
応募ID	数値型	各応募者を識別する一意のID
応募者名	文字列型	応募者の氏名（リアル値）
応募職種	文字列型	応募した職種（リアル値）
応募日	日付型	応募を受け付けた日
選考ステータス	文字列型	書類審査 / 一次面接 / 二次面接 / 内定 など（リアル値）
採用担当者	文字列型	応募を担当する人事担当者（リアル値）
最終結果	文字列型	採用 / 不採用

5 サブ情報明細テーブル（履歴情報の管理）

選考履歴テーブル

項目名	データ型	説明
選考ID	数値型	選考履歴の一意ID
応募ID	数値型	人材採用管理台帳と関連（外部キー）
選考日	日付型	選考を実施した日
選考フェーズ	文字列型	書類審査 / 一次面接 / 二次面接 / 最終面接 など（リアル値）
面接官	文字列型	面接担当者の氏名（リアル値）
評価	文字列型	合格 / 不合格 / 保留
コメント	文字列型	面接の評価コメント

試用期間評価テーブル

項目名	データ型	説明
評価ID	数値型	試用期間評価の一意ID
応募ID	数値型	人材採用管理台帳と関連（外部キー）
試用開始日	日付型	試用期間の開始日
試用終了日	日付型	試用期間の終了日
業務適応度	文字列型	高 / 中 / 低
成果評価	文字列型	優秀 / 普通 / 不適格
最終判断	文字列型	本採用 / 解雇 / 再試用

7 正規化後のテーブル生成SQL

```
CREATE TABLE 人材採用管理 (
    応募ID AUTOINCREMENT PRIMARY KEY,
    応募者名 TEXT NOT NULL,
    応募職種 TEXT NOT NULL,
    応募日 DATETIME NOT NULL
    選考ステータス TEXT NOT NU
    採用担当者 TEXT NOT NULL
    最終結果 TEXT
);
```

```
CREATE TABLE 選考履歴 (
    選考ID AUTOINCREMENT PRIMARY KEY,
    応募ID INTEGER NOT NULL,
    選考日 DATETIME NOT NULL,
    選考フェーズ TEXT NOT NULL,
    面接官 TEXT NOT NULL,
    評価 TEXT NOT NULL,
    コメント TEXT
);
```

```
CREATE TABLE 試用期間評価 (
    評価ID AUTOINCREMENT PRIMARY KEY,
    応募ID INTEGER NOT NULL,
    試用開始日 DATETIME NOT NULL,
    試用終了日 DATETIME NOT NULL,
    業務適応度 TEXT NOT NULL,
    成果評価 TEXT NOT NULL,
    最終判断 TEXT NOT NULL
);
```

8 注意点およびポイント

リアル値（応募者名、職種、選考フェーズなど）を直接格納

選考履歴を記録し、応募者の選考過程を追跡可能に

試用期間の評価を記録し、本採用の適切性を分析できる

固定情報（職種、評価基準など）はマスタ化

Accessのリレーションシップは使用せず、VBAやクエリで関連付け

変更履歴は履歴テーブルに記録し、サブフォームで可視化

9 クエリ設計

10 フォーム設計

11 リポート設計

12 モジュール設計

管制管理部門のデータベース設計

1 計画概要

管制管理部門では、警備員の配置、シフト管理、緊急対応、巡回報告 などを統合的に管理します。

警備員の勤務シフトや配置状況を把握し、緊急時の指示や対応を迅速に行えるようにデータを適切に管理します。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

警備員名、配置現場、シフト時間、対応内容などはIDではなく、直接リアル値を格外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

警備員のシフト履歴、緊急対応履歴、巡回報告を履歴テーブルで管理　　異常発生時の対応状況を記録し、後の分析に活用できるようにする

3. 固定情報のマスタ化

シフトパターン、緊急対応種別、異常報告ステータスなどはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

警備員のシフト管理、巡回報告、異常発生履歴はメインフォームのサブフォームとして可視化

3 仕様

データ管理

警備員の勤務シフト、巡回報告、異常発生履歴を履歴テーブルで管理

緊急対応時の指示内容や対応履歴を記録し、過去の対応を分析可能に

シフト管理を効率化し、適切な人員配置を実現する

データ構造

シフトごとの警備業務を「管制管理台帳」に記録

警備員のシフト履歴、巡回報告、異常発生履歴をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

管制管理台帳

項目名	データ型	説明
管制ID	数値型	各シフトを識別する一意のID
配置現場	文字列型	警備を担当する現場名（リアル値）
担当警備員	文字列型	担当する警備員の氏名（リアル値）
シフト時間	文字列型	早番 / 遅番 / 夜勤 など（リアル値）
勤務日	日付型	勤務日
緊急対応	文字列型	あり / なし
巡回報告	文字列型	巡回報告のステータス（完了 / 異常あり）
指示内容	文字列型	管制室からの指示内容

6 サブ情報明細テーブル（履歴情報の管理）

シフト履歴テーブル

項目名	データ型	説明
シフトID	数値型	シフト履歴の一意ID
管制ID	数値型	管制管理台帳と関連（外部キー）
警備員名	文字列型	勤務した警備員名（リアル値）
出勤時間	時間型	出勤した時間
退勤時間	時間型	退勤した時間
業務内容	文字列型	巡回 / 受付 / 監視 など（リアル値）

異常発生履歴テーブル

項目名	データ型	説明
異常ID	数値型	異常発生データの一意ID
管制ID	数値型	管制管理台帳と関連（外部キー）
発生日時	日付型	異常が発生した日時
異常内容	文字列型	侵入検知 / 火災警報 / 設備異常 など（リアル値）
対応内容	文字列型	警察連絡 / 消火対応 など（リアル値）
担当警備員	文字列型	異常対応を担当した警備員名（リアル値）

7 正規化後のテーブル生成SQL

```
CREATE TABLE 管制管理 (
    管制ID AUTOINCREMENT PRIMARY KEY,
    配置現場 TEXT NOT NULL,
    担当警備員 TEXT NOT NULL
    シフト時間 TEXT NOT NULL,
    勤務日 DATETIME NOT NULL
    緊急対応 TEXT,
    巡回報告 TEXT,
    指示内容 TEXT
);
```

```
CREATE TABLE シフト履歴 (
    シフトID AUTOINCREMENT PRIMARY KEY,
    管制ID INTEGER NOT NULL,
    警備員名 TEXT NOT NULL,
    出勤時間 DATETIME NOT NULL,
    退勤時間 DATETIME NOT NULL,
    業務内容 TEXT NOT NULL
);
```

```
CREATE TABLE 異常発生履歴 (
    異常ID AUTOINCREMENT PRIMARY KEY,
    管制ID INTEGER NOT NULL,
    発生日時 DATETIME NOT NULL,
    異常内容 TEXT NOT NULL,
    対応内容 TEXT NOT NULL,
    担当警備員 TEXT NOT NULL
);
```

8 注意点およびポイント

リアル値（警備員名、配置現場、シフト時間など）を直接格納

警備員のシフト履歴、異常発生履歴を履歴テーブルで管理

固定情報（シフトパターン、対応内容など）はマスタ化

Accessのリレーションシップは使用せず、VBAやクエリで関連付け

変更履歴は履歴テーブルに記録し、サブフォームで可視化

9 クエリ設計

10 フォーム設計

11 リポート設計

12 モジュール設計

機械警備管理部門のデータベース設計

1 計画概要

機械警備管理部門では、警備機材の管理、メンテナンス履歴、異常発生、警報対応 などを統合的に管理します。

警備機材の設置状況、故障や修理履歴、異常発生時の対応履歴を適切に記録し、設備の稼働状況を最適化するためのデータ管理を行います。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

機材名、設置現場、異常種別、対応内容などはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

警備機材のメンテナンス履歴、異常発生履歴、対応履歴を履歴テーブルで管理

異常発生時の対応状況を記録し、後の分析に活用できるようにする

3. 固定情報のマスタ化

機材種別、異常種別、対応内容などはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

警備機材のメンテナンス履歴、異常発生履歴はメインフォームのサブフォームとして可視化

3 仕様

データ管理

警備機材の設置情報、メンテナンス履歴、異常発生履歴を履歴テーブルで管理

機材の故障・修理履歴を記録し、交換や修理のタイミングを最適化

異常発生時の対応履歴を記録し、対応フローの改善に活用

データ構造

機材ごとの管理情報を「機械警備管理台帳」に記録

メンテナンス履歴、異常発生履歴、対応履歴をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

機械警備管理台帳

項目名	データ型	説明
機材ID	数値型	各警備機材を識別する一意のID
機材名	文字列型	機材の名称（リアル値）
設置現場	文字列型	設置場所（リアル値）
設置日	日付型	設置した日付
メーカー名	文字列型	機材のメーカー名（リアル値）
状態	文字列型	正常 / 故障中 / 修理中 など（リアル値）
最終メンテナンス日	日付型	最新のメンテナンス実施日

5 サブ情報明細テーブル（履歴情報の管理）

メンテナンス履歴テーブル

項目名	データ型	説明
メンテナンスID	数値型	メンテナンス履歴の一意ID
機材ID	数値型	機械警備管理台帳と関連（外部キー）
実施日	日付型	メンテナンスを実施した日
作業担当者	文字列型	メンテナンスを担当した技術者名（リアル値）
作業内容	文字列型	点検 / 修理 / 部品交換 など（リアル値）
コメント	文字列型	作業の詳細メモ

異常発生履歴テーブル

項目名	データ型	説明
異常ID	数値型	異常発生データの一意ID
機材ID	数値型	機械警備管理台帳と関連（外部キー）
発生日時	日付型	異常が発生した日時
異常内容	文字列型	通信エラー / 機器故障 / センサー異常 など（リアル値）
対応内容	文字列型	再起動 / 修理依頼 / 機材交換 など（リアル値）
担当者名	文字列型	異常対応を行った担当者名（リアル値）

6 正規化後のテーブル生成SQL

```
CREATE TABLE 機械警備管理 (
    機材ID AUTOINCREMENT PRIMARY KEY,
    機材名 TEXT NOT NULL,
    設置現場 TEXT NOT NULL,
    設置日 DATETIME NOT NULL,
    メーカー名 TEXT NOT NULL,
    状態 TEXT NOT NULL,
    最終メンテナンス日 DATETIME
);
```

```
CREATE TABLE メンテナンス履歴 (
    メンテナンスID AUTOINCREMENT PRIMARY KEY,
    機材ID INTEGER NOT NULL,
    実施日 DATETIME NOT NULL,
    作業担当者 TEXT NOT NULL,
    作業内容 TEXT NOT NULL,
    コメント TEXT
);
```

```
CREATE TABLE 異常発生履歴 (
    異常ID AUTOINCREMENT PRIMARY KEY,
    機材ID INTEGER NOT NULL,
    発生日時 DATETIME NOT NULL,
    異常内容 TEXT NOT NULL,
    対応内容 TEXT NOT NULL,
    担当者名 TEXT NOT NULL
);
```

7 注意点およびポイント

リアル値（機材名、設置現場、異常内容など）を直接格納

警備機材のメンテナンス履歴、異常発生履歴を履歴テーブルで管理

固定情報（異常種別、対応内容など）はマスタ化

Accessのリレーションシップは使用せず、VBAやクエリで関連付け

変更履歴は履歴テーブルに記録し、サブフォームで可視化

8 クエリ設計

9 フォーム設計

10 リポート設計

11 モジュール設計

交通警備部門のデータベース設計

1 計画概要

交通警備部門では、交通整理業務、警備員配置、出動記録、異常発生、シフト管理、勤怠管理、業務報告 などを統合的に管理します。

交通警備員の配置状況、出動履歴、異常発生時の対応履歴を適切に記録し、業務の最適化を図ります。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現 します。

2 設計方針

1.リアル値の管理

警備員名、配置場所、異常種別、対応内容などはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2.履歴管理の導入

出動履歴、異常発生履歴、業務報告を履歴テーブルで管理

異常発生時の対応状況を記録し、後の分析に活用できるようにする

3.固定情報のマスタ化

「取引先名」は「現場名」と統合し、現場マスタで一元管理

異常種別、対応内容、シフトパターンなどはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

出動履歴、異常発生履歴はメインフォームのサブフォームとして可視化

3 仕様

データ管理

警備員の基本情報を一元管理

交通警備の配置情報、シフト管理、勤怠管理を統合

出動履歴、異常発生履歴、業務報告を履歴テーブルで管理

異常発生時の対応履歴を記録し、対応フローの改善に活用

4 メイン台帳テーブル（リアル値仕様）

交通警備管理台帳

項目名	データ型	説明
現場ID	オートナンバー	各現場を識別する一意のID
現場名	短いテキスト	交通警備現場の名称（リアル値）
所在地	長いテキスト	交通警備の実施場所（リアル値）
交通整理種別	短いテキスト	工事 / イベント / 事故処理 など（リアル値）
契約状況	短いテキスト	継続中 / 契約終了 / 一時休止 など

5 サブ情報明細テーブル（履歴情報の管理）

出動履歴テーブル

項目名	データ型	説明
出動ID	オートナンバー	出動履歴の一意ID
現場ID	数値型	交通警備管理台帳と関連（外部キー）
出動日時	日時型	出動した日時
警備員名	短いテキスト	出動した警備員名（リアル値）
出動結果	長いテキスト	問題なし / 交通整理完了 / 異常対応 など（リアル値）

異常発生履歴テーブル

項目名	データ型	説明
異常ID	オートナンバー	異常発生履歴の一意ID
現場ID	数値型	交通警備管理台帳と関連（外部キー）
発生日時	日時型	異常が発生した日時
異常内容	長いテキスト	事故発生 / 交通渋滞 / 信号機故障 など（リアル値）
対応内容	長いテキスト	迂回指示 / 管理者報告 / 交通誘導継続 など（リアル値）
担当者名	短いテキスト	異常対応を行った警備員名（リアル値）

業務報告テーブル

項目名	データ型	説明
報告ID	オートナンバー	業務報告の一意ID
現場ID	数値型	交通警備管理台帳と関連（外部キー）
報告日時	日時型	報告した日時
報告者名	短いテキスト	報告を行った警備員名（リアル値）
報告内容	長いテキスト	交通整理状況、異常発生、トラブル対応など

6 シフト・勤怠管理テーブル

シフト管理テーブル

項目名	データ型	説明
シフトID	オートナンバー	一意のシフトID
現場ID	数値型	交通警備管理台帳と関連（外部キー）
警備員名	短いテキスト	担当警備員（リアル値）
勤務日	日時型	勤務日
勤務時間	短いテキスト	09:00–18:00 など

勤怠管理テーブル

項目名	データ型	説明
勤怠ID	オートナンバー	一意の勤怠ID
現場ID	数値型	交通警備管理台帳と関連（外部キー）
警備員名	短いテキスト	出勤した警備員名（リアル値）
出勤時間	日時型	出勤時間
退勤時間	日時型	退勤時間

7 正規化後のテーブル生成SQL

```
CREATE TABLE 交通警備管理
    現場ID AUTOINCREMENT PF
    現場名 TEXT NOT NULL,
    所在地 TEXT NOT NULL,
    交通整理種別 TEXT NOT NU
    契約状況 TEXT NOT NULL
);
```

```
CREATE TABLE 出動履歴 (
    出動ID AUTOINCREMENT PRIMARY KEY,
    現場ID INTEGER NOT NULL,
    出動日時 DATETIME NOT NULL,
    警備員名 TEXT NOT NULL,
    出動結果 TEXT NOT NULL
);
```

```
CREATE TABLE 異常発生履歴 (
    異常ID AUTOINCREMENT PRIMARY KEY,
    現場ID INTEGER NOT NULL,
    発生日時 DATETIME NOT NULL,
    異常内容 TEXT NOT NULL,
    対応内容 TEXT NOT NULL,
    担当者名 TEXT NOT NULL
);
```

8 注意点およびポイント

リアル値（現場名、警備員名、異常内容など）を直接格納

出動履歴、異常発生、業務報告を履歴テーブルで管理

固定情報（異常種別、対応内容など）はマスタ化

リレーションシップなしで、VBAやクエリで関連付け

9 クエリ設計

10 フォーム設計

11 リポート設計

12 モジュール設計

施設警備部門のデータベース設計

1 計画概要

施設警備部門では、施設の管理、巡回警備、異常発生対応、シフト管理、勤怠管理、業務報告 などを統合的に管理します。

警備員の配置状況、巡回警備履歴、異常発生時の対応履歴を適切に記録し、業務の最適化を図ります。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現 します。

2 設計方針

1.リアル値の管理

施設名、所在地、異常種別、対応内容などはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2.履歴管理の導入

巡回警備履歴、異常発生履歴、業務報告を履歴テーブルで管理

異常発生時の対応状況を記録し、後の分析に活用できるようにする

3.固定情報のマスタ化

取引先名は「施設名」と統合し、施設マスタで一元管理

異常種別、対応内容、シフトパターンなどはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

施設警備の巡回履歴、異常発生履歴はメインフォームのサブフォームとして可視化

3 仕様

データ管理

施設の基本情報、警備員の基本情報を一元管理

施設ごとの警備員配置情報、シフト管理、勤怠管理を統合

巡回警備履歴、異常発生履歴、業務報告を履歴テーブルで管理

異常発生時の対応履歴を記録し、対応フローの改善に活用

4 メイン台帳テーブル（リアル値仕様）

施設警備管理台帳

項目名	データ型	説明
施設ID	オートナンバー	各施設を識別する一意のID
施設名	短いテキスト	施設の名称（リアル値）
所在地	長いテキスト	施設の所在地（リアル値）
施設種別	短いテキスト	オフィス / 商業施設 / 病院 など（リアル値）
契約状況	短いテキスト	継続中 / 契約終了 / 一時休止 など

5 サブ情報明細テーブル（履歴情報の管理）

巡回警備履歴テーブル

項目名	データ型	説明
巡回ID	オートナンバー	巡回履歴の一意ID
施設ID	数値型	施設マスタと関連（外部キー）
巡回日時	日時型	巡回した日時
警備員名	短いテキスト	巡回を実施した警備員（リアル値）
巡回結果	長いテキスト	異常なし / 施錠確認 / 設備確認 など（リアル値）

異常発生履歴テーブル

項目名	データ型	説明
異常ID	オートナンバー	異常発生履歴の一意ID
施設ID	数値型	施設マスタと関連（外部キー）
発生日時	日時型	異常が発生した日時
異常内容	長いテキスト	侵入検知 / 火災報知器作動 / 施錠ミス など（リアル値）
対応内容	長いテキスト	再巡回 / 管理者報告 / 修理依頼 など（リアル値）
担当者名	短いテキスト	異常対応を行った警備員名（リアル値）

業務報告テーブル

項目名	データ型	説明
報告ID	オートナンバー	業務報告の一意ID
施設ID	数値型	施設マスタと関連（外部キー）
報告日時	日時型	報告した日時
報告者名	短いテキスト	報告を行った警備員名（リアル値）
報告内容	長いテキスト	巡回状況、設備異常、トラブル発生など

6 シフト・勤怠管理テーブル

シフト管理テーブル

項目名	データ型	説明
シフトID	オートナンバー	一意のシフトID
施設ID	数値型	施設マスタと関連（外部キー）
警備員名	短いテキスト	担当警備員（リアル値）
勤務日	日時型	勤務日
勤務時間	短いテキスト	09:00–18:00 など

勤怠管理テーブル

項目名	データ型	説明
勤怠ID	オートナンバー	一意の勤怠ID
施設ID	数値型	施設マスタと関連（外部キー）
警備員名	短いテキスト	出勤した警備員名（リアル値）
出勤時間	日時型	出勤時間
退勤時間	日時型	退勤時間

7 正規化後のテーブル生成SQL

```
CREATE TABLE 施設警備管理 (
    施設ID AUTOINCREMENT PRIM
    施設名 TEXT NOT NULL,
    所在地 TEXT NOT NULL,
    施設種別 TEXT NOT NULL,
    契約状況 TEXT NOT NULL
);
```

```
CREATE TABLE 巡回警備履歴 (
    巡回ID AUTOINCREMENT PRIMARY KE
    施設ID INTEGER NOT NULL,
    巡回日時 DATETIME NOT NULL,
    警備員名 TEXT NOT NULL,
    巡回結果 TEXT NOT NULL
);
```

```
CREATE TABLE 異常発生履歴 (
    異常ID AUTOINCREMENT PRIMARY KEY,
    施設ID INTEGER NOT NULL,
    発生日時 DATETIME NOT NULL,
    異常内容 TEXT NOT NULL,
    対応内容 TEXT NOT NULL,
    担当者名 TEXT NOT NULL
);
CREATE TABLE シフト管理 (
    シフトID AUTOINCREMENT PRIMARY KEY,
    施設ID INTEGER NOT NULL,
    警備員名 TEXT NOT NULL,
    勤務日 DATETIME NOT NULL,
    勤務時間 TEXT NOT NULL
);
```

8 注意点およびポイント

リアル値（施設名、警備員名、異常内容など）を直接格納

巡回警備、異常発生、業務報告を履歴テーブルで管理

固定情報（異常種別、対応内容など）はマスタ化

リレーションシップなしで、VBAやクエリで関連付け

9 クエリ設計

10 フォーム設計

11 リポート設計

12 モジュール設計

集配金管理部門のデータベース設計

1 計画概要

集配金管理部門では、集配ルート管理、現金輸送管理、集配報告 を統合的に管理します。

輸送ルートごとの現金取扱状況、警備員の配置、異常発生時の対応履歴を適切に記録し、安全かつ効率的な現金輸送を実現します。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

輸送ルート名、顧客名、輸送担当者名、異常内容などはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

集配金業務の履歴、異常発生履歴、輸送報告を履歴テーブルで管理

異常発生時の対応状況を記録し、後の分析に活用できるようにする

3. 固定情報のマスタ化

ルート種別、輸送手段、異常種別、対応内容などはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

集配履歴、異常発生履歴はメインフォームのサブフォームとして可視化

3 仕様

データ管理

現金輸送のルート管理、輸送履歴、異常発生履歴を履歴テーブルで管理

警備員の輸送実績を記録し、安全性向上のためのデータを蓄積

異常発生時の対応履歴を記録し、適切な対応策を検討できるようにする

データ構造

輸送ごとの管理情報を「集配金管理台帳」に記録　　集配履歴、異常発生履歴、輸送報告をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

集配金管理台帳

項目名	データ型	説明
輸送ID	数値型	各輸送業務を識別する一意のID
ルート名	文字列型	集配ルート名（リアル値）
顧客名	文字列型	取引先の顧客名（リアル値）
担当警備員	文字列型	担当する警備員の氏名（リアル値）
輸送日	日付型	輸送日
開始時間	時間型	集配開始時間
終了時間	時間型	集配終了時間
輸送額	数値型	輸送した金額
異常発生	文字列型	あり / なし

5 サブ情報明細テーブル（履歴情報の管理）

集配履歴テーブル

項目名	データ型	説明
履歴ID	数値型	集配履歴の一意ID
輸送ID	数値型	集配金管理台帳と関連（外部キー）
担当警備員	文字列型	輸送を担当した警備員名（リアル値）
集配地点	文字列型	集配先の名称（リアル値）
集配時間	時間型	集配を実施した時間
集配額	数値型	その地点での集配金額

異常発生履歴テーブル

項目名	データ型	説明
異常ID	数値型	異常発生データの一意ID
輸送ID	数値型	集配金管理台帳と関連（外部キー）
発生日時	日付型	異常が発生した日時
異常内容	文字列型	強盗未遂 / 通信エラー / 設備故障 など（リアル値）
対応内容	文字列型	警察通報 / 現場確認 / 緊急連絡 など（リアル値）
担当者名	文字列型	異常対応を行った担当者名（リアル値）

7 正規化後のテーブル生成SQL

```
CREATE TABLE 集配金管理 (
    輸送ID AUTOINCREMENT PRIMARY KEY,
    ルート名 TEXT NOT NULL,
    顧客名 TEXT NOT NULL,
    担当警備員 TEXT NOT NULL
    輸送日 DATETIME NOT NULL
    開始時間 DATETIME NOT NU
    終了時間 DATETIME NOT NU
    輸送額 INTEGER NOT NULL,
    異常発生 TEXT
);
```

```
CREATE TABLE 集配履歴 (
    履歴ID AUTOINCREMENT PRIMARY KEY,
    輸送ID INTEGER NOT NULL,
    担当警備員 TEXT NOT NULL,
    集配地点 TEXT NOT NULL,
    集配時間 DATETIME NOT NULL,
    集配額 INTEGER NOT NULL
);
```

```
CREATE TABLE 異常発生履歴 (
    異常ID AUTOINCREMENT PRIMARY KEY,
    輸送ID INTEGER NOT NULL,
    発生日時 DATETIME NOT NULL,
    異常内容 TEXT NOT NULL,
    対応内容 TEXT NOT NULL,
    担当者名 TEXT NOT NULL
);
```

8 注意点およびポイント

リアル値（ルート名、顧客名、警備員名、異常内容など）を直接格納

集配履歴、異常発生履歴を履歴テーブルで管理

固定情報（異常種別、対応内容など）はマスタ化

Accessのリレーションシップは使用せず、VBAやクエリで関連付け

変更履歴は履歴テーブルに記録し、サブフォームで可視化

9 クエリ設計

10 フォーム設計

11 リポート設計

12 モジュール設計

教育管理部門のデータベース設計

1 計画概要

教育管理部門では、研修プログラム、受講履歴、資格取得、教育評価 などを統合的に管理します。

警備員や従業員の教育履歴を記録し、研修の受講状況や資格取得状況を適切に管理することで、教育の効果測定や適正な人材配置を実現します。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

研修名、受講者名、資格名、評価内容などはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

受講履歴、試験結果、資格取得履歴を履歴テーブルで管理

研修受講状況や資格更新状況を記録し、教育の進捗を管理

3. 固定情報のマスタ化

研修カテゴリ、資格種別、評価基準などはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

研修受講履歴、資格取得履歴はメインフォームのサブフォームとして可視化

3 仕様

データ管理

研修プログラム、受講履歴、資格取得状況を履歴テーブルで管理

受講者の試験結果や教育評価を記録し、適切な人材育成を支援

資格更新の期限を管理し、適切なタイミングで再試験を実施

データ構造

研修ごとの管理情報を「教育管理台帳」に記録　　受講履歴、資格取得履歴、試験結果をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

教育管理台帳

項目名	データ型	説明
研修ID	数値型	各研修プログラムを識別する一意のID
研修名	文字列型	研修プログラム名（リアル値）
研修カテゴリ	文字列型	法定研修 / 社内研修 / 外部研修 など（リアル値）
研修日	日付型	研修実施日
講師名	文字列型	研修講師の氏名（リアル値）
会場	文字列型	研修会場の名称（リアル値）
定員	数値型	研修の定員人数
状態	文字列型	実施済み / 予定 など（リアル値）

5 サブ情報明細テーブル（履歴情報の管理）

受講履歴テーブル

項目名	データ型	説明
受講ID	数値型	受講履歴の一意ID
研修ID	数値型	教育管理台帳と関連（外部キー）
受講者名	文字列型	研修を受講した従業員名（リアル値）
受講日	日付型	受講した日
受講状況	文字列型	受講済み / 欠席 / 途中退席 など（リアル値）
試験結果	文字列型	合格 / 不合格 など（リアル値）
評価	文字列型	優秀 / 普通 / 要改善 など（リアル値）

資格取得履歴テーブル

項目名	データ型	説明
資格ID	数値型	資格取得履歴の一意ID
受講者名	文字列型	資格を取得した従業員名（リアル値）
資格名	文字列型	取得した資格の名称（リアル値）
取得日	日付型	資格取得日
有効期限	日付型	資格の有効期限
更新状況	文字列型	更新済み / 更新予定 / 失効 など（リアル値）

6 正規化後のテーブル生成SQL

```
CREATE TABLE 教育管理 (
    研修ID AUTOINCREMENT PRIMARY KEY,
    研修名 TEXT NOT NULL,
    研修カテゴリ TEXT NOT NULL,
    研修日 DATETIME NOT NULL,
    講師名 TEXT NOT NULL,
    会場 TEXT NOT NULL,
    定員 INTEGER NOT NULL,
    状態 TEXT NOT NULL
);
```

```
CREATE TABLE 受講履歴 (
    受講ID AUTOINCREMENT PRIMARY KEY,
    研修ID INTEGER NOT NULL,
    受講者名 TEXT NOT NULL,
    受講日 DATETIME NOT NULL,
    受講状況 TEXT NOT NULL,
    試験結果 TEXT,
    評価 TEXT
);
```

```
CREATE TABLE 資格取得履歴 (
    資格ID AUTOINCREMENT PRIMARY KEY,
    受講者名 TEXT NOT NULL,
    資格名 TEXT NOT NULL,
    取得日 DATETIME NOT NULL,
    有効期限 DATETIME NOT NULL,
    更新状況 TEXT NOT NULL
);
```

7 注意点およびポイント

リアル値（研修名、受講者名、資格名など）を直接格納

受講履歴、資格取得履歴を履歴テーブルで管理

固定情報（研修カテゴリ、評価基準など）はマスタ化

Accessのリレーションシップは使用せず、VBAやクエリで関連付け

変更履歴は履歴テーブルに記録し、サブフォームで可視化

8 クエリ設計

9 フォーム設計

10 リポート設計

11 モジュール設計

営業所管理部門のデータベース設計

1 計画概要

営業所管理部門では、営業所の運営管理、備品管理、業績管理、スタッフ管理 などを統合的に管理します。

営業所ごとの基本情報や設備、備品の管理状況、業績データ、スタッフの配置状況を適切に管理し、営業所運営の効率化を実現します。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

営業所名、所在地、責任者名、備品名などはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

営業所の売上・経費履歴、備品の購入履歴、スタッフの配置履歴を履歴テーブルで管理

営業所の異動や責任者変更の履歴を記録し、長期的な分析を可能にする

3. 固定情報のマスタ化

営業所種別、備品種別、業績評価項目などはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

売上履歴、備品管理履歴、スタッフ配置履歴はメインフォームのサブフォームとして可視化

3 仕様

データ管理

営業所の基本情報、売上履歴、備品管理履歴を履歴テーブルで管理

スタッフの異動履歴を記録し、配置管理を最適化

営業所ごとの業績データを記録し、経営分析を実施可能にする

データ構造

営業所ごとの管理情報を「営業所管理台帳」に記録

売上履歴、備品管理履歴、スタッフ配置履歴をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

項目名	データ型	説明
営業所ID	数値型	各営業所を識別する一意のID
営業所名	文字列型	営業所の名称（リアル値）
所在地	文字列型	営業所の住所（リアル値）
責任者名	文字列型	営業所責任者の氏名（リアル値）
開設日	日付型	営業所開設日
営業所種別	文字列型	本社 / 支社 / 営業所 など（リアル値）
状態	文字列型	営業中 / 閉鎖 など（リアル値）

5 サブ情報明細テーブル（履歴情報の管理）

売上履歴テーブル

項目名	データ型	説明
売上ID	数値型	売上履歴の一意ID
営業所ID	数値型	営業所管理台帳と関連（外部キー）
売上日	日付型	売上発生日
売上額	数値型	売上金額
経費	数値型	その月の経費
純利益	数値型	売上額 − 経費

備品管理履歴テーブル

項目名	データ型	説明
備品ID	数値型	備品管理履歴の一意ID
営業所ID	数値型	営業所管理台帳と関連（外部キー）
備品名	文字列型	管理する備品の名称（リアル値）
購入日	日付型	備品購入日
数量	数値型	購入数量
状態	文字列型	良好 / 修理中 / 廃棄 など（リアル値）

スタッフ配置履歴テーブル

項目名	データ型	説明
配置ID	数値型	スタッフ配置履歴の一意ID
営業所ID	数値型	営業所管理台帳と関連（外部キー）
従業員名	文字列型	配置された従業員名（リアル値）
配置開始日	日付型	配置開始日
配置終了日	日付型	配置終了日（異動・退職時のみ）
役職	文字列型	配置時の役職（リアル値）

7 正規化後のテーブル生成SQL

```
CREATE TABLE 営業所管理 (
    営業所ID AUTOINCREMENT PRIMARY KEY,
    営業所名 TEXT NOT NULL,
    所在地 TEXT NOT NULL,
    責任者名 TEXT NOT NULL,
    開設日 DATETIME NOT NULL,
    営業所種別 TEXT NOT NULL,
    状態 TEXT NOT NULL
);
CREATE TABLE 売上履歴 (
    売上ID AUTOINCREMENT PRIMARY KEY,
    営業所ID INTEGER NOT NULL,
    売上日 DATETIME NOT NULL,
    売上額 INTEGER NOT NULL,
    経費 INTEGER NOT NULL,
    純利益 INTEGER NOT NULL
);
CREATE TABLE 備品管理履歴 (
    備品ID AUTOINCREMENT PRIMARY KEY,
    営業所ID INTEGER NOT NULL,
    備品名 TEXT NOT NULL,
    購入日 DATETIME NOT NULL,
    数量 INTEGER NOT NULL,
    状態 TEXT NOT NULL
);
CREATE TABLE スタッフ配置履歴 (
    配置ID AUTOINCREMENT PRIMARY KEY,
    営業所ID INTEGER NOT NULL,
    従業員名 TEXT NOT NULL,
    配置開始日 DATETIME NOT NULL,
    配置終了日 DATETIME,
    役職 TEXT NOT NULL
);
```

8 注意点およびポイント

リアル値（営業所名、所在地、責任者名など）を直接格納
売上履歴、備品管理履歴、スタッフ配置履歴を履歴テーブ
固定情報（営業所種別、備品状態など）はマスタ化
Accessのリレーションシップは使用せず、VBAやクエリで関連付け
変更履歴は履歴テーブルに記録し、サブフォームで可視化

9 クエリ設計

10 フォーム設計

11 リポート設計

12 モジュール設計

制服管理部門のデータベース設計

1 計画概要

制服管理部門では、制服の在庫管理、仕入・支給・返却・クリーニング履歴 を統合的に管理します。

警備員ごとの制服支給状況、返却履歴、クリーニング回数、在庫の変動を記録し、適切な制服管理を実現します。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

制服種別、サイズ、警備員名、状態などはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

制服の支給履歴、返却履歴、クリーニング履歴を履歴テーブルで管理

支給日や返却日の記録により、適切な在庫管理を可能にする

3. 固定情報のマスタ化

制服種別（夏服・冬服・雨具）、サイズ、状態（良好・汚損・廃棄済み）などはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

支給履歴、返却履歴、クリーニング履歴はメインフォームのサブフォームとして可視化

3 仕様

データ管理

制服の支給履歴、返却履歴、クリーニング履歴を履歴テーブルで管理

警備員ごとの支給数や返却状況を記録し、適切な在庫管理を可能にする

クリーニング頻度を記録し、管理状況を可視化

データ構造

制服ごとの管理情報を「制服管理台帳」に記録

支給履歴、返却履歴、クリーニング履歴をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

制服管理台帳

項目名	データ型	説明
制服ID	数値型	各制服を識別する一意のID
制服種別	文字列型	夏服 / 冬服 / 雨具 など（リアル値）
サイズ	文字列型	S / M / L / LL など（リアル値）
在庫数	数値型	現在の在庫数
状態	文字列型	良好 / 汚損 / 廃棄済み など（リアル値）

5 サブ情報明細テーブル（履歴情報の管理）

支給履歴テーブル

項目名	データ型	説明
支給ID	数値型	支給履歴の一意ID
制服ID	数値型	制服管理台帳と関連（外部キー）
支給日	日付型	制服支給日
支給者名	文字列型	支給された警備員の氏名（リアル値）
数量	数値型	支給された枚数

返却履歴テーブル

項目名	データ型	説明
返却ID	数値型	返却履歴の一意ID
制服ID	数値型	制服管理台帳と関連（外部キー）
返却日	日付型	制服返却日
返却者名	文字列型	返却した警備員の氏名（リアル値）
状態	文字列型	良好 / 汚損 / 廃棄済み など（リアル値）

クリーニング履歴テーブル

項目名	データ型	説明
クリーニングID	数値型	クリーニング履歴の一意ID
制服ID	数値型	制服管理台帳と関連（外部キー）
クリーニング日	日付型	クリーニングを行った日
クリーニング回数	数値型	クリーニングの累計回数

7 正規化後のテーブル生成SQL

```
CREATE TABLE 制服管理 (
    制服ID AUTOINCREMENT PRIMARY KEY,
    制服種別 TEXT NOT NULL,
    サイズ TEXT NOT NULL,
    在庫数 INTEGER NOT NULL,
    状態 TEXT NOT NULL
);
```

```
CREATE TABLE 支給履歴 (
    支給ID AUTOINCREMENT PRIMARY KEY,
    制服ID INTEGER NOT NULL,
    支給日 DATETIME NOT NULL,
    支給者名 TEXT NOT NULL,
    数量 INTEGER NOT NULL
);
```

```
CREATE TABLE 返却履歴 (
    返却ID AUTOINCREMENT PRIMARY KEY,
    制服ID INTEGER NOT NULL,
    返却日 DATETIME NOT NULL,
    返却者名 TEXT NOT NULL,
    状態 TEXT NOT NULL
);
```

```
CREATE TABLE クリーニング履歴 (
    クリーニングID AUTOINCREMENT PRIMARY KEY,
    制服ID INTEGER NOT NULL,
    クリーニング日 DATETIME NOT NULL,
    クリーニング回数 INTEGER NOT NULL
);
```

8 注意点およびポイント

リアル値（制服種別、サイズ、支給者名など）を直接格納

支給履歴、返却履歴、クリーニング履歴を履歴テーブルで管理

固定情報（状態、サイズなど）はマスタ化

Accessのリレーションシップは使用せず、VBAやクエリで関連付け

変更履歴は履歴テーブルに記録し、サブフォームで可視化

9 クエリ設計

10 フォーム設計

11 リポート設計

12 モジュール設計

車両管理部門のデータベース設計

1 計画概要

車両管理部門では、車両の台帳管理、運行履歴、メンテナンス履歴、燃料管理 を統合的に管理します。

各車両の登録情報、運行履歴、点検・修理履歴を記録し、車両の安全運用と効率的な管理を実現します。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

車両名、車両番号、運転者名、メンテナンス項目などはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

運行履歴、メンテナンス履歴、燃料管理履歴を履歴テーブルで管理

点検や修理履歴を記録し、車両の安全性を確保

3. 固定情報のマスタ化

車両種別、燃料種別、メンテナンス項目などはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

運行履歴、メンテナンス履歴、燃料管理履歴はメインフォームのサブフォームとして可視化

3 仕様

データ管理

車両の基本情報、運行履歴、メンテナンス履歴を履歴テーブルで管理

燃料消費履歴を記録し、燃費分析を行い、コスト管理を強化

車両ごとの利用状況を記録し、適切な車両配備を可能にする

データ構造

車両ごとの管理情報を「車両管理台帳」に記録

運行履歴、メンテナンス履歴、燃料管理履歴をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

車両管理台帳

項目名	データ型	説明
車両ID	数値型	各車両を識別する一意のID
車両番号	文字列型	車両の登録番号（リアル値）
車両名	文字列型	車両の名称（リアル値）
車両種別	文字列型	軽自動車 / 普通車 / トラック など（リアル値）
燃料種別	文字列型	ガソリン / 軽油 / 電動 など（リアル値）
登録日	日付型	車両の登録日
状態	文字列型	使用中 / 整備中 / 廃車 など（リアル値）

5 サブ情報明細テーブル（履歴情報の管理）

運行履歴テーブル

項目名	データ型	説明
運行ID	数値型	運行履歴の一意ID
車両ID	数値型	車両管理台帳と関連（外部キー）
運転者名	文字列型	運行を担当した運転者名（リアル値）
運行日	日付型	運行日
出発地	文字列型	出発した場所（リアル値）
到着地	文字列型	到着した場所（リアル値）
走行距離	数値型	走行距離（km）

メンテナンス履歴テーブル

項目名	データ型	説明
メンテナンスID	数値型	メンテナンス履歴の一意ID
車両ID	数値型	車両管理台帳と関連（外部キー）
実施日	日付型	メンテナンスを実施した日
作業担当者	文字列型	メンテナンスを担当した技術者名（リアル値）
作業内容	文字列型	オイル交換 / タイヤ交換 / 車検 など（リアル値）
コメント	文字列型	作業の詳細メモ

燃料管理履歴テーブル

項目名	データ型	説明
燃料ID	数値型	燃料管理履歴の一意ID
車両ID	数値型	車両管理台帳と関連（外部キー）
給油日	日付型	給油した日
給油量	数値型	給油した燃料量（リットル）
燃費	数値型	走行距離 / 給油量（km/L）

6 正規化後のテーブル生成SQL

```
CREATE TABLE 車両管理 (
    車両ID AUTOINCREMENT PRIMARY KEY,
    車両番号 TEXT NOT NULL,
    車両名 TEXT NOT NULL,
    車両種別 TEXT NOT NULL,
    燃料種別 TEXT NOT NULL,
    登録日 DATETIME NOT NULL,
    状態 TEXT NOT NULL
);
```

```
CREATE TABLE 運行履歴 (
    運行ID AUTOINCREMENT PRIMARY KEY,
    車両ID INTEGER NOT NULL,
    運転者名 TEXT NOT NULL,
    運行日 DATETIME NOT NULL,
    出発地 TEXT NOT NULL,
    到着地 TEXT NOT NULL,
    走行距離 INTEGER NOT NULL
);
```

```
CREATE TABLE メンテナンス履歴 (
    メンテナンスID AUTOINCREMENT PRIMARY KEY,
    車両ID INTEGER NOT NULL,
    実施日 DATETIME NOT NULL,
    作業担当者 TEXT NOT NULL,
    作業内容 TEXT NOT NULL,
    コメント TEXT
);
```

```
CREATE TABLE 燃料管理履歴 (
    燃料ID AUTOINCREMENT PRIMARY KEY,
    車両ID INTEGER NOT NULL,
    給油日 DATETIME NOT NULL,
    給油量 INTEGER NOT NULL,
    燃費 REAL NOT NULL
);
```

7 注意点およびポイント

リアル値（車両番号、車両名、運転者名など）を直接格納

運行履歴、メンテナンス履歴、燃料管理履歴を履歴テーブルで管理

固定情報（車両種別、燃料種別など）はマスタ化

Accessのリレーションシップは使用せず、VBAやクエリで関連付け

変更履歴は履歴テーブルに記録し、サブフォームで可視化

車両管理部門のデータベース設計が完了しました。

8 クエリ設計

9 フォーム設計

10 リポート設計

11 モジュール設計

プロジェクト管理部門のデータベース設計

1 計画概要

プロジェクト管理部門では、プロジェクトマスタ、進捗管理、リソース管理、コスト管理 を統合的に管理します。

各プロジェクトの基本情報、進捗状況、リソース（人員・設備・資金）管理、コストの記録を行い、プロジェクトの円滑な運営を実現します。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

プロジェクト名、リーダー名、フェーズ名、タスク名などはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

プロジェクト進捗履歴、タスク管理履歴、リソース使用履歴、コスト管理履歴を履歴テーブルで管理

進捗状況やタスクの変更履歴を記録し、適切なプロジェクト運営を可能にする

3. 固定情報のマスタ化

プロジェクト種別、フェーズ分類、リソース種別などはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

進捗履歴、タスク管理履歴、コスト管理履歴はメインフォームのサブフォームとして可視化

3 仕様

データ管理

プロジェクトの基本情報、進捗履歴、タスク管理履歴を履歴テーブルで管理

リソース（人員・設備・資金）の管理とコストの記録を行い、経費分析を実施

プロジェクトのフェーズごとに進捗を記録し、適切なリソース配分を支援

データ構造

プロジェクトごとの管理情報を「プロジェクト管理台帳」に記録

進捗履歴、タスク管理履歴、リソース使用履歴、コスト管理履歴をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

プロジェクト管理台帳

項目名	データ型	説明
プロジェクトID	数値型	各プロジェクトを識別する一意のID
プロジェクト名	文字列型	プロジェクトの名称（リアル値）
リーダー名	文字列型	プロジェクトリーダーの氏名（リアル値）
開始日	日付型	プロジェクトの開始日
終了予定日	日付型	プロジェクトの終了予定日
状態	文字列型	進行中 / 完了 / 保留 など（リアル値）

5 サブ情報明細テーブル（履歴情報の管理）

進捗履歴テーブル

項目名	データ型	説明
進捗ID	数値型	進捗履歴の一意ID
プロジェクトID	数値型	プロジェクト管理台帳と関連（外部キー）
更新日	日付型	進捗更新日
フェーズ	文字列型	企画 / 設計 / 開発 / テスト / 完了 など（リアル値）
進捗率	数値型	現在の進捗率（%）
担当者名	文字列型	進捗を報告した担当者の氏名（リアル値）

タスク管理履歴テーブル

項目名	データ型	説明
タスクID	数値型	タスク履歴の一意ID
プロジェクトID	数値型	プロジェクト管理台帳と関連（外部キー）
タスク名	文字列型	実施タスクの名称（リアル値）
担当者名	文字列型	タスク担当者の氏名（リアル値）
期限	日付型	タスクの期限
状態	文字列型	未着手 / 進行中 / 完了 など（リアル値）

リソース使用履歴テーブル

項目名	データ型	説明
リソースID	数値型	リソース使用履歴の一意ID
プロジェクトID	数値型	プロジェクト管理台帳と関連（外部キー）
リソース名	文字列型	利用リソースの名称（リアル値）
使用日	日付型	使用した日
使用時間	数値型	使用時間（時間）

コスト管理履歴テーブル

項目名	データ型	説明
コストID	数値型	コスト管理履歴の一意ID
プロジェクトID	数値型	プロジェクト管理台帳と関連（外部キー）
費目	文字列型	人件費 / 設備費 / 交通費 など（リアル値）
発生日時	日付型	コストが発生した日
金額	数値型	発生した費用の金額

6 正規化後のテーブル生成SQL

```
CREATE TABLE プロジェクト管理 (
    プロジェクトID AUTOINCREMENT PRIMARY KEY,
    プロジェクト名 TEXT NOT NULL,
    リーダー名 TEXT NOT NULL,
    開始日 DATETIME NOT NULL,
    終了予定日 DATETIME NOT NULL,
    状態 TEXT NOT NULL
);
```

```
CREATE TABLE タスク管理履歴 (
    タスクID AUTOINCREMENT PRIMARY KEY,
    プロジェクトID INTEGER NOT NULL,
    タスク名 TEXT NOT NULL,
    担当者名 TEXT NOT NULL,
    期限 DATETIME NOT NULL,
    状態 TEXT NOT NULL
);
```

```
CREATE TABLE 進捗履歴 (
    進捗ID AUTOINCREMENT PRIMARY KEY,
    プロジェクトID INTEGER NOT NULL,
    更新日 DATETIME NOT NULL,
    フェーズ TEXT NOT NULL,
    進捗率 INTEGER NOT NULL,
    担当者名 TEXT NOT NULL
);
```

```
CREATE TABLE リソース使用履歴 (
    リソースID AUTOINCREMENT PRIMARY KEY,
    プロジェクトID INTEGER NOT NULL,
    リソース名 TEXT NOT NULL,
    使用日 DATETIME NOT NULL,
    使用時間 INTEGER NOT NULL
);
```

```
CREATE TABLE コスト管理履歴 (
    コストID AUTOINCREMENT PRIMARY KEY,
    プロジェクトID INTEGER NOT NULL,
    費目 TEXT NOT NULL,
    発生日時 DATETIME NOT NULL,
    金額 INTEGER NOT NULL
);
```

8 クエリ設計

9 フォーム設計

10 リポート設計

11 モジュール設計

連絡情報管理部門のデータベース設計

1 計画概要

連絡情報管理部門では、連絡先マスタ、連絡履歴、緊急連絡網、通知管理 を統合的に管理します。

社員や取引先の連絡先情報、社内外の連絡履歴、緊急時の連絡対応記録、通知の履歴を適切に記録し、情報共有の効率化と連絡体制の強化を実現します。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

氏名、役職、部署名、取引先名、連絡方法などはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

連絡履歴、緊急連絡履歴、通知履歴を履歴テーブルで管理

緊急連絡の対応状況や通知送信記録を保持し、連絡体制の改善に活用

3. 固定情報のマスタ化

部署、役職、取引先種別、通知方法などはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

連絡履歴、緊急連絡履歴、通知履歴はメインフォームのサブフォームとして可視化

3 仕様

データ管理

連絡先マスタを作成し、社員・取引先の連絡先を管理

通知履歴を記録し、確実な情報伝達を確保

社内外の連絡履歴、緊急連絡履歴、通知履歴を履歴テーブルで管理

データ構造

連絡情報を「連絡情報管理台帳」に記録

連絡履歴、緊急連絡履歴、通知履歴をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

連絡情報管理台帳

項目名	データ型	説明
連絡先ID	数値型	各連絡先を識別する一意のID
氏名	文字列型	連絡先の氏名（リアル値）
役職	文字列型	役職（リアル値）
部署名	文字列型	所属部署（リアル値）
取引先名	文字列型	取引先の名称（リアル値）
連絡先種別	文字列型	社員 / 取引先 / 顧客 など（リアル値）
電話番号	文字列型	電話番号
メールアドレス	文字列型	メールアドレス
住所	文字列型	連絡先の住所

5 サブ情報明細テーブル（履歴情報の管理）

連絡履歴テーブル

項目名	データ型	説明
連絡ID	数値型	連絡履歴の一意ID
連絡先ID	数値型	連絡情報管理台帳と関連（外部キー）
連絡日	日付型	連絡を行った日
連絡方法	文字列型	電話 / メール / 訪問 など（リアル値）
連絡内容	文字列型	連絡の内容

緊急連絡履歴テーブル

項目名	データ型	説明
緊急連絡ID	数値型	緊急連絡履歴の一意ID
連絡先ID	数値型	連絡情報管理台帳と関連（外部キー）
連絡日時	日付型	緊急連絡を行った日時
連絡内容	文字列型	緊急連絡の内容
対応状況	文字列型	確認済み / 未対応 / 再連絡 など（リアル値）

通知履歴テーブル

項目名	データ型	説明
通知ID	数値型	通知履歴の一意ID
連絡先ID	数値型	連絡情報管理台帳と関連（外部キー）
通知日	日付型	通知を送信した日
通知方法	文字列型	メール / SMS / 書面 など（リアル値）
通知内容	文字列型	通知の内容
確認状況	文字列型	確認済み / 未確認 など（リアル値）

6 正規化後のテーブル生成SQL

```
CREATE TABLE 連絡情報管理 (
    連絡先ID AUTOINCREMENT PRIMARY KEY,
    氏名 TEXT NOT NULL,
    役職 TEXT NOT NULL,
    部署名 TEXT NOT NULL,
    取引先名 TEXT,
    連絡先種別 TEXT NOT NULL
    電話番号 TEXT,
    メールアドレス TEXT,
    住所 TEXT
);
```

```
CREATE TABLE 連絡履歴 (
    連絡ID AUTOINCREMENT PRIMARY KEY,
    連絡先ID INTEGER NOT NULL,
    連絡日 DATETIME NOT NULL,
    連絡方法 TEXT NOT NULL,
    連絡内容 TEXT NOT NULL
);
```

```
CREATE TABLE 緊急連絡履歴 (
    緊急連絡ID AUTOINCREMENT PRIMARY KEY,
    連絡先ID INTEGER NOT NULL,
    連絡日時 DATETIME NOT NULL,
    連絡内容 TEXT NOT NULL,
    対応状況 TEXT NOT NULL
);
```

```
CREATE TABLE 通知履歴 (
    通知ID AUTOINCREMENT PRIMARY KEY,
    連絡先ID INTEGER NOT NULL,
    通知日 DATETIME NOT NULL,
    通知方法 TEXT NOT NULL,
    通知内容 TEXT NOT NULL,
    確認状況 TEXT NOT NULL
);
```

7 注意点およびポイント

リアル値（氏名、役職、部署名など）を直接格納

連絡履歴、緊急連絡履歴、通知履歴を履歴テーブルで管理

固定情報（連絡先種別、通知方法、対応状況など）はマスタ化

Accessのリレーションシップは使用せず、VBAやクエリで関連付け

変更履歴は履歴テーブルに記録し、サブフォームで可視化

8 クエリ設計

9 フォーム設計

10 リポート設計

11 モジュール設計

入札管理部門のデータベース設計

1 計画概要

入札管理部門では、入札資格管理、案件管理、証明書管理 を統合的に管理します。

企業や官公庁との契約に関する入札案件、必要な資格・証明書の管理、入札の結果を記録し、透明性の高い入札プロセスを実現します。

また、リアル値を直接格納し、ID参照を最小限にすることで、データの直感的な管理を実現します。

2 設計方針

1. リアル値の管理

入札案件名、取引先名、担当者名、証明書名などはIDではなく、直接リアル値を格納

外部キーの使用を最小限に抑え、データの直感的な管理を実現

2. 履歴管理の導入

入札案件の履歴、証明書の更新履歴、契約履歴を履歴テーブルで管理

入札結果や契約状況を記録し、適切な入札戦略を支援

3. 固定情報のマスタ化

入札種別、契約形態、証明書種別などはマスタ化し、データの統一性を確保

マスタの更新により、関連データの整合性を確保

4. Accessの運用ルール

リレーションシップは設定せず、VBAやクエリを活用

入札案件履歴、証明書管理履歴、契約履歴はメインフォームのサブフォームとして可視化

4 仕様

データ管理

入札資格情報、入札案件情報、契約履歴を履歴テーブルで管理

入札案件ごとに必要な資格・証明書の管理を徹底

入札の結果を記録し、次回の入札に向けた戦略を策定

データ構造

入札案件の管理情報を「入札管理台帳」に記録

入札履歴、証明書管理履歴、契約履歴をサブテーブルとして管理

運用方法

Accessのリレーションシップは使用せず、クエリやVBAでデータを連携

サブフォームを利用して履歴データを可視化

4 メイン台帳テーブル（リアル値仕様）

入札管理台帳

項目名	データ型	説明
入札案件ID	数値型	各入札案件を識別する一意のID
入札案件名	文字列型	入札案件の名称（リアル値）
取引先名	文字列型	入札元の企業・官公庁名（リアル値）
担当者名	文字列型	担当営業・管理者の氏名（リアル値）
入札日	日付型	入札実施日
入札種別	文字列型	公開入札 / 指名入札 など（リアル値）
状態	文字列型	進行中 / 落札 / 失注 など（リアル値）

5 サブ情報明細テーブル（履歴情報の管理）

入札履歴テーブル

項目名	データ型	説明
入札履歴ID	数値型	入札履歴の一意ID
入札案件ID	数値型	入札管理台帳と関連（外部キー）
入札日	日付型	入札を行った日
入札金額	数値型	入札額
入札結果	文字列型	落札 / 失注 など（リアル値）
コメント	文字列型	入札結果に関する補足情報

証明書管理履歴テーブル

項目名	データ型	説明
証明書ID	数値型	証明書管理履歴の一意ID
入札案件ID	数値型	入札管理台帳と関連（外部キー）
証明書名	文字列型	提出した証明書の名称（リアル値）
取得日	日付型	証明書取得日
有効期限	日付型	証明書の有効期限
更新状況	文字列型	更新済み / 更新予定 / 失効 など（リアル値）

契約履歴テーブル

項目名	データ型	説明
契約ID	数値型	契約履歴の一意ID
入札案件ID	数値型	入札管理台帳と関連（外部キー）
契約日	日付型	契約締結日
契約金額	数値型	契約総額
契約形態	文字列型	一括契約 / 分割契約 など（リアル値）
契約担当者	文字列型	契約を管理する担当者名（リアル値）

6 正規化後のテーブル生成SQL

```
CREATE TABLE 入札管理 (
    入札案件ID AUTOINCREMENT PRIMARY KEY,
    入札案件名 TEXT NOT NULL,
    取引先名 TEXT NOT NULL,
    担当者名 TEXT NOT NULL,
    入札日 DATETIME NOT NULL,
    入札種別 TEXT NOT NULL,
    状態 TEXT NOT NULL
);
CREATE TABLE 入札履歴 (
    入札履歴ID AUTOINCREMENT PRIMARY KEY,
    入札案件ID INTEGER NOT NULL,
    入札日 DATETIME NOT NULL,
    入札金額 INTEGER NOT NULL,
    入札結果 TEXT NOT NULL,
    コメント TEXT
);
CREATE TABLE 証明書管理履歴 (
    証明書ID AUTOINCREMENT PRIMARY KEY,
    入札案件ID INTEGER NOT NULL,
    証明書名 TEXT NOT NULL,
    取得日 DATETIME NOT NULL,
    有効期限 DATETIME NOT NULL,
    更新状況 TEXT NOT NULL
);
CREATE TABLE 契約履歴 (
    契約ID AUTOINCREMENT PRIMARY KEY,
    入札案件ID INTEGER NOT NULL,
    契約日 DATETIME NOT NULL,
    契約金額 INTEGER NOT NULL,
    契約形態 TEXT NOT NULL,
    契約担当者 TEXT NOT NULL
);
```

7 注意点およびポイント

リアル値（入札案件名、取引先名、担当者名など）を直接

入札履歴、証明書管理履歴、契約履歴を履歴テーブルで

固定情報（入札種別、契約形態、証明書種別など）はマスタ化

Accessのリレーションシップは使用せず、VBAやクエリで関連付け

変更履歴は履歴テーブルに記録し、サブフォームで可視化

8 クエリ設計

9 フォーム設計

10 リポート設計

11 モジュール設計

トラブルシューティング

1. **システム障害時の対応フロー**

   Accessが開かない場合 → 修復ツールを使用

   SQL Serverのデータが取得できない場合 → 接続テスト方法

   Excelの帳票が正しく出力されない場合 → マクロのチェック方法

2. **バックアップ復元の具体的な手順**

   削除されたデータの復旧方法（削除ログテーブルから復旧）

   30日を超えたデータの復元方法（バックアップサーバーから復旧）

   Accessの復旧手順（ファイル破損時の対処）

3. **よくあるエラーメッセージと対策**

   「クエリが遅い」→ インデックス再構築

   「ファイルが開けない」→ ODBC接続の確認

   「データが反映されない」→ リンクテーブルの再接続