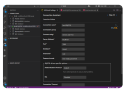
PGミニセミナー「データベース」

- 今日のお題
  - データベースとはなにか
  - なぜ使われるのか
- 対象
  - データベースをちゃんと理解したい方
  - データベースを利用したWebアプリケーションを構築したい方
- お断り
  - データベースに関する共通の知識を得るための内容で、WordPressに特化したものではありません
- 環境
  - 参考
    - これから学ぶ技術の全体像 https://zero1-pg.com/topic/01-%e3%81%af%e3%81%98%e3%82%81%e3%81%ab-2/
    - PHP/DB(SQL) https://zero1-pg.com/topic/03-php/
  - データベースの環境
    - XAMMP/MAMP https://www.apachefriends.org/jp/index.html
      - データベース
        - MySQL
      - 
      - XAMPP
    - Visual Studio Code https://code.visualstudio.com/download
      - Microsoft謹製のテキストエディター
    - SQLTools
      - 
      - VSCodeからMySQLを使う拡張機能
      - 
      - 設定
- 内容
  - データベースとは
    - データベースとは
      - さまざまな目的を達成するために蓄積されたデータの集合体
      - データをアプリケーションプログラムとは分離独立し、維持・管理を可能とする
    - データベースの種類
      - インハウスデータベース 企業や学校など組織内で作成し利用するデータベース
      - オンラインデータベース ネットワークを経由して情報の検索に利用するデータベース
    - データベースモデルの種類
      - 関係型データベース（Relational Data Base：RDB）
        - データを2次元の表形式で表す
        - 表間に関係を持たせる
      - Key-Value型データベース（NOSQL型データベース）
        - 「Key（キー）」と「Value（値）」のペアで管理する
    - DBMSとは
      - DBMS（Data Base Management System）
        - データベース管理システム
      - データベースを実現するためのミドルウェア Oracle・SQL Server・MySQL・PostgreSQL・DB2など
      - RDBMSが現在の主流
  - DBMSの機能
    - データベース言語機能
      - データの定義と操作を行うための言語（SQL）
      - ANSIやJISで標準化されている
      - SQL（Structured Query Language）
        - データの定義 SQL-DDL（Data Definition Language：データ定義言語）
        - データの操作 SQL-DML（Data Manipulation Language：データ操作言語）
    - 整合性制約機能
      - アプリケーションが意識しなくとも、データの整合性を保つ機能
      - SQL-DDLで指定
      - 制約の例
        - 一意（UNIQUE）制約 データの値が一意（ユニーク）であること
        - 参照制約（または、外部キー制約） あるデータの値が他のデータの値として存在すること
        - 非NULL制約 データにNULL値（空）を持てない
        - 形式制約 データ型（桁数、型）に関する制約
        - 検査（CHECK）制約 データ登録時のチェックに関する制約
    - 機密保護機能（セキュリティ）
      - データベースの機密性を保持する機能
      - データベースのオブジェクト単位・ユーザー単位でアクセス権が設定できる
        - データベース
        - テーブル
        - 行
        - 列
        - ビュー
        - ストアドプロシージャ
      - 権限の種類
        - ALL 以下のすべての権限
        - SELECT 参照
        - INSERT 追加
        - DELETE 削除
        - UPDATE 更新
    - トランザクション管理機能（同時実行制御機能）
      - トランザクション処理 DBの更新・参照を伴う処理
      - 同時実行制御機能
        - 複数のトランザクション処理が同時に発生しても、データに矛盾を含まないようにする
        - ロストアップデート 同じデータを同時に更新すると、一方の更新が失われる
        - 同時実行制御
      - ACID特性
        - ランザクションを安全かつ確実に処理するための4つの重要な特性
        - 原子性（Atomicity）
          - 「すべて実行されるか、まったく実行されないか」
          - トランザクション内の処理は 不可分な単位（アトミック）であり、一部だけが実行されることはない。
          - 途中でエラーが発生すると、すべての処理がロールバック（ROLLBACK）され、データの整合性が保たれる。
          - 例
            - 銀行口座の振込処理
            - Aさんの口座から1万円を引き出す
            - Bさんの口座に1万円を振り込む
            - → どちらか一方だけが実行されると整合性が崩れるため、両方が成功する場合のみ確定し（COMMIT）、途中で失敗した場合はすべて取り消す（ROLLBACK）。
        - 一貫性（Consistency）
        - 独立性（Isolation）
        - 耐久性（Durability）
      - トランザクション管理の種類
        - コミット（COMMIT） 変更を確定し、永続化する
        - ロールバック（ROLLBACK） 変更を取り消し、トランザクション開始前の状態に戻す
        - ロールフォワード（ROLLFORWARD） 障害復旧時に、ジャーナル（ログ）を使ってデータベースを最新の整合性がある状態に回復する
    - 障害回復機能
      - 障害回復機能とは データベースの障害回復機能は、システム障害（ハードウェア故障、ソフトウェアバグ、停電など）が発生した際に、データを整合性のある状態に戻すための機能
      - 障害への備え
        - バックアップファイル（アーカイブ）
          - 「データベースのある時点の完全なコピー」
          - データベースを特定の時点の状態に復元するためのファイル。
          - 通常、定期的に取得 し、障害発生時に最新のバックアップをもとに復旧を行う。
          - 「フルバックアップ」（すべてのデータを保存）や「差分バックアップ」（変更部分のみ保存）などの種類がある。
          - 例
            - 1日1回深夜にデータベースのバックアップを取得。
            - システム障害が発生したら、最新のバックアップを使って復元し、その後ログを適用して最新状態に戻す
        - ログファイル
        - チェックポイント
      - 
      - まとめ
  - SQL
- 注意点・その他
  - データベースは概念を掴もう！
  - SQLはとにかく手を動かして使ってみるのが大事！

PGミニセミナー 「データベース」

1. 環境

1.1. 参考

1.1.1. これから学ぶ技術の全体像

1.1.1.1. https://zero1-pg.com/topic/01-%e3%81%af%e3%81%98%e3%82%81%e3%81%ab-2/

1.1.2. PHP/DB(SQL)

1.1.2.1. https://zero1-pg.com/topic/03-php/

1.2. データベースの環境

1.2.1. XAMMP/MAMP

1.2.1.1. https://www.apachefriends.org/jp/index.html

1.2.1.2. データベース

1.2.1.2.1. MySQL

1.2.1.2.2. XAMPP

1.2.2. Visual Studio Code

1.2.2.1. https://code.visualstudio.com/download

1.2.2.2. Microsoft謹製のテキストエディター

1.2.3. SQLTools

1.2.3.1. VSCodeからMySQLを使う拡張機能

1.2.3.2. 設定

2. 対象

2.1. データベースをちゃんと理解したい方

2.2. データベースを利用したWebアプリケーションを構築したい方

3. お断り

3.1. データベースに関する共通の知識を得るための内容で、WordPressに特化したものではありません

4. 今日のお題

4.1. データベースとはなにか

4.2. なぜ使われるのか

5. 内容

5.1. データベースとは

5.1.1. データベースとは

5.1.1.1. さまざまな目的を達成するために蓄積されたデータの集合体

5.1.1.2. データをアプリケーションプログラムとは分離独立し、維持・管理を可能とする

5.1.2. データベースの種類

5.1.2.1. インハウスデータベース

5.1.2.1.1. 企業や学校など組織内で作成し利用するデータベース

5.1.2.2. オンラインデータベース

5.1.2.2.1. ネットワークを経由して情報の検索に利用するデータベース

5.1.3. データベースモデルの種類

5.1.3.1. 関係型データベース（Relational Data Base：RDB）

5.1.3.1.1. データを2次元の表形式で表す

5.1.3.1.2. 表間に関係を持たせる

5.1.3.2. Key-Value型データベース（NO SQL型データベース）

5.1.3.2.1.「Key（キー）」と「Value（値）」のペアで管理する

5.1.4. DBMSとは

5.1.4.1. DBMS（Data Base Management System）

5.1.4.1.1. データベース管理システム

5.1.4.2. データベースを実現するためのミドルウェア

5.1.4.2.1. Oracle、SQL Server、MySQL、PostgreSQL、DB2など

5.1.4.3. RDBMSが現在の主流

5.2. DBMSの機能

5.2.1. データベース言語機能

5.2.1.1. データの定義と操作を行うための言語（SQL）

5.2.1.2. ANSIやJISで標準化されている

5.2.1.3. SQL（Structured Query Language）

5.2.1.3.1. データの定義

5.2.1.3.1.1. SQL-DDL（Data Definition Language：データ定義言語）

5.2.1.3.2. データの操作

5.2.1.3.2.1. SQL-DML（Data Manipulation Language：データ操作言語）

5.2.2. 整合性制約機能

5.2.2.1. アプリケーションが意識しなくとも、データの整合性を保つ機能

5.2.2.2. SQL-DDLで指定

5.2.2.3. 制約の例

5.2.2.3.1. 一意（UNIQUE）制約

5.2.2.3.1.1. データの値が一意（ユニーク）であること

5.2.2.3.2. 参照制約（または、外部キー制約）

5.2.2.3.2.1. あるデータの値が他のデータの値として存在すること

5.2.2.3.3. 非NULL制約

5.2.2.3.3.1. データにNULL値（空）を持てない

5.2.2.3.4. 形式制約

5.2.2.3.4.1. データ型（桁数、型）に関する制約

5.2.2.3.5. 検査（CHECK）制約

5.2.2.3.5.1. データ登録時のチェックに関する制約

5.2.3. 機密保護機能（セキュリティ）

5.2.3.1. データベースの機密性を保持する機能

5.2.3.2. データベースのオブジェクト単位・ユーザー単位でアクセス権が設定できる

5.2.3.2.1. データベース

5.2.3.2.2. テーブル

5.2.3.2.3. 行

5.2.3.2.4. 列

5.2.3.2.5. ビュー

5.2.3.2.6. ストアドプロシージャ

5.2.3.3. 権限の種類

5.2.3.3.1. ALL

5.2.3.3.1.1. 以下のすべての権限

5.2.3.3.2. SELECT

5.2.3.3.2.1. 参照

5.2.3.3.3. INSERT

5.2.3.3.3.1. 追加

5.2.3.3.4. DELETE

5.2.3.3.4.1. 削除

5.2.3.3.5. UPDATE

5.2.3.3.5.1. 更新

5.2.4. トランザクション管理機能（同時実行制御機能）

5.2.4.1. トランザクション処理

5.2.4.1.1. DBの更新・参照を伴う処理

5.2.4.2. 同時実行制御機能

5.2.4.2.1. 複数のトランザクション処理が同時に発生しても、データに矛盾を含まないようにする

5.2.4.2.2. ロストアップデート

5.2.4.2.2.1. 同じデータを同時に更新すると、一方の更新が失われる

5.2.4.2.3. 同時実行制御

5.2.4.2.3.1. 複数のトランザクションが同時に実行される際に、データの整合性を維持するための仕組みです。

5.2.4.2.3.2. 排他制御（Exclusive Control）

5.2.4.2.3.2.1. 「複数のトランザクションが同じデータを同時に操作しないようにする仕組み」

5.2.4.2.3.2.2. 排他制御とは、データの整合性を保つために、あるトランザクションがデータを更新中は、他のトランザクションがそのデータにアクセスできないようにする 制御方法の総称です。

5.2.4.2.3.2.3. ロック（Locking）

5.2.4.2.3.2.3.1. 排他ロック（Exclusive Lock）

5.2.4.2.3.2.3.1.1. 他のトランザクションが読み書きできない。

5.2.4.2.3.2.3.2. 共有ロック（Shared Lock）

5.2.4.2.3.2.3.2.1. 読み取りは許可されるが、書き込みは不可。

5.2.4.2.3.2.3.3. 例：

5.2.4.2.3.2.3.3.1. 銀行口座の更新時に、ある人の残高を更新する間は、他のトランザクションが同じ口座のデータを変更できないようにする。

5.2.4.2.3.2.4. デッドロック（Deadlock）

5.2.4.2.3.2.4.1. 「複数のトランザクションが互いにロックを保持したまま、相手のロックが解除されるのを待ち続ける状態」

5.2.4.2.3.2.4.2. デッドロックが発生すると、どのトランザクションも処理を進めることができず、永久に待機状態になる ため、適切な対策が必要

5.2.4.2.3.2.4.3. 例

5.2.4.2.3.2.4.3.1. トランザクションA（T1）

5.2.4.2.3.2.4.3.1.1. 「口座X」をロック（先に取得）

5.2.4.2.3.2.4.3.1.2. 「口座Y」へのロックを要求（しかし、すでにT2がロックしているため待機）

5.2.4.2.3.2.4.3.2. トランザクションB（T2）

5.2.4.2.3.2.4.3.2.1. 「口座Y」をロック（先に取得）

5.2.4.2.3.2.4.3.2.2. 「口座X」へのロックを要求（しかし、すでにT1がロックしているため待機）

5.2.4.2.3.2.4.3.3. この状態では、T1は T2のロックが解除されるのを待ち、T2はT1のロックが解除されるのを待っているため、永久に待ち続ける

5.2.4.2.3.2.4.4. デッドロックの対策

5.2.4.2.3.2.4.4.1. ロックの順序を統一する（順序付け）

5.2.4.2.3.2.4.4.2. タイムアウトを設定する

5.2.4.2.3.2.4.4.3. デッドロック検出と回避

5.2.4.2.3.2.4.4.4. 楽観的排他制御（OCC）を使う

5.2.4.2.3.2.5. 悲観的排他制御（Pessimistic Concurrency Control, PCC）

5.2.4.2.3.2.5.1. 「データの競合が発生する可能性を前提に、あらかじめロックをかけておく方式」

5.2.4.2.3.2.5.2. 特徴

5.2.4.2.3.2.5.2.1. データを操作する前に、他のトランザクションがそのデータにアクセスできないようにロックをかける。

5.2.4.2.3.2.5.2.2. 競合を未然に防ぐため、安全だがロックによる待ち時間が発生しやすい。

5.2.4.2.3.2.5.3. メリット

5.2.4.2.3.2.5.3.1. 競合が発生しやすい環境（例：大量のユーザーが同じデータを更新するシステム） ではデータ整合性を高く保てる。

5.2.4.2.3.2.5.4. デメリット

5.2.4.2.3.2.5.4.1. ロックの取得・解放のオーバーヘッドが発生するため、スループット（処理性能）が低下 しやすい。

5.2.4.2.3.2.5.5. 例

5.2.4.2.3.2.5.5.1. 在庫管理システム Aさんが商品Aの在庫数を更新する前にロックを取得 → BさんはAさんの処理が終わるまで待つ。

5.2.4.2.3.2.6. 楽観的排他制御（Optimistic Concurrency Control, OCC）

5.2.4.2.3.2.6.1. 「データの競合は発生しないと仮定し、最終的に競合した場合のみ対処する方式」

5.2.4.2.3.2.6.2. 特徴

5.2.4.2.3.2.6.2.1. データの更新時に ロックを使用せず、最後の更新タイミングで競合があった場合のみ対処（例：リトライやエラー） する。

5.2.4.2.3.2.6.2.2. データの更新頻度が低い 場合に有効。

5.2.4.2.3.2.6.3. メリット

5.2.4.2.3.2.6.3.1. ロックによる待ち時間がないため、スループットが高い（ロック待ちの影響を受けない）。

5.2.4.2.3.2.6.3.2. リソース消費が少なく、スケーラビリティに優れる。

5.2.4.2.3.2.6.4. デメリット

5.2.4.2.3.2.6.4.1. 競合が発生した場合、再試行が必要になり、処理の手間が増える。

5.2.4.2.3.2.6.5. 例

5.2.4.2.3.2.6.5.1. ブログ記事の編集 AさんとBさんが同じ記事を編集していた場合、Bさんが保存する際にAさんの更新後のデータと競合があれば、Bさんに「変更がありました」と通知し、編集をやり直してもらう。

5.2.4.2.3.2.7. まとめ

5.2.4.2.3.2.7.1. まとめ

5.2.4.3. ACID特性

5.2.4.3.1. ランザクションを安全かつ確実に処理するための4つの重要な特性

5.2.4.3.2. 原子性（Atomicity）

5.2.4.3.2.1. 「すべて実行されるか、まったく実行されないか」

5.2.4.3.2.2. トランザクション内の処理は 不可分な単位（アトミック）であり、一部だけが実行されることはない。

5.2.4.3.2.3. 途中でエラーが発生すると、すべての処理がロールバック（ROLLBACK） され、データの整合性が保たれる。

5.2.4.3.2.4. 例

5.2.4.3.2.4.1. 銀行口座の振込処理 ・Aさんの口座から1万円を引き出す ・Bさんの口座に1万円を振り込む → どちらか一方だけが実行されると整合性が崩れるため、両方が成功する場合のみ確定し（COMMIT）、途中で失敗した場合はすべて取り消す（ROLLBACK）。

5.2.4.3.3. 一貫性（Consistency）

5.2.4.3.3.1. 「トランザクション前後でデータの整合性が保たれる」

5.2.4.3.3.2. トランザクションの実行前後で、データベースのルールや制約（外部キー制約、一意性制約など）が維持される。

5.2.4.3.3.3. トランザクションが成功しても、データが矛盾する状態にならない。

5.2.4.3.3.4. 例

5.2.4.3.3.4.1. 銀行の口座残高が負の値にならないようにする（預金額 < 0 を許容しない）。

5.2.4.3.3.4.2. 商品の在庫数が注文数を下回らないようにする。

5.2.4.3.4. 独立性（Isolation）

5.2.4.3.4.1. 「同時実行されるトランザクションが互いに影響を与えない」

5.2.4.3.4.2. 複数のトランザクションが同時に実行されても、互いに干渉せず、順番に実行されたかのような結果になる。

5.2.4.3.4.3. 例

5.2.4.3.4.3.1. Aさんがオンラインショップで商品を購入する処理を実行中に、Bさんが同じ商品の在庫数を参照する。 独立性が確保されていないと、Aさんが購入処理中の未確定データ（未コミットデータ）をBさんが見てしまう可能性がある。

5.2.4.3.4.4. トランザクションの分離レベル（Isolation Levels）

5.2.4.3.4.4.1. 独立性を調整するためのレベル

5.2.4.3.4.4.2. Read Uncommitted

5.2.4.3.4.4.2.1. 未コミットデータを読める → データの不整合が起こりやすい

5.2.4.3.4.4.3. Read Committed

5.2.4.3.4.4.3.1. コミット済みのデータのみ読める

5.2.4.3.4.4.4. Repeatable Read

5.2.4.3.4.4.4.1. 同じトランザクション内でデータの読み取り結果が一貫

5.2.4.3.4.4.5. Serializable

5.2.4.3.4.4.5.1. 完全な独立性を確保、同時実行トランザクションの影響を受けない

5.2.4.3.5. 耐久性（Durability）

5.2.4.3.5.1. 「トランザクションの結果が確定したら、システム障害が発生してもデータが失われない」

5.2.4.3.5.2. COMMIT されたデータは、ディスクに書き込まれ、電源障害やシステムクラッシュ後も保持される。

5.2.4.3.5.3. データベースは、WAL（Write-Ahead Logging） や レプリケーションなどを活用してデータを保護する。

5.2.4.3.5.4. 例

5.2.4.3.5.4.1. ATMでお金を引き出した後、電源が落ちても引き出し処理が取り消されずに確実に残る

5.2.4.4. トランザクション管理の種類

5.2.4.4.1. コミット（COMMIT）

5.2.4.4.1.1. 変更を確定し、永続化する

5.2.4.4.2. ロールバック（ROLLBACK）

5.2.4.4.2.1. 変更を取り消し、トランザクション開始前の状態に戻す

5.2.4.4.3. ロールフォワード（ROLLFORWARD）

5.2.4.4.3.1. 障害復旧時に、ジャーナル（ログ）を使ってデータベースを最新の整合性がある状態に回復する

5.2.5. 障害回復機能

5.2.5.1. 障害回復機能とは

5.2.5.1.1. データベースの障害回復機能は、システム障害（ハードウェア故障、ソフトウェアバグ、停電など）が発生した際に、データを整合性のある状態に戻すための機能

5.2.5.2. 障害への備え

5.2.5.2.1. バックアップファイル（アーカイブ）

5.2.5.2.1.1. 「データベースのある時点の完全なコピー」

5.2.5.2.1.2. データベースを特定の時点の状態に復元するためのファイル。

5.2.5.2.1.3. 通常、定期的に取得 し、障害発生時に最新のバックアップをもとに復旧を行う。

5.2.5.2.1.4. 「フルバックアップ」（すべてのデータを保存）や「差分バックアップ」（変更部分のみ保存）などの種類がある。

5.2.5.2.1.5. 例

5.2.5.2.1.5.1. 1日1回深夜にデータベースのバックアップを取得。

5.2.5.2.1.5.2. システム障害が発生したら、最新のバックアップを使って復元し、その後ログを適用して最新状態に戻す

5.2.5.2.2. ログファイル

5.2.5.2.2.1. 「トランザクションの実行履歴を記録するファイル」

5.2.5.2.2.2. ログファイルには、データベースの変更履歴が記録されており、障害時の復旧に使用される。

5.2.5.2.2.3. （1）更新前ログ（Before Image）

5.2.5.2.2.3.1. 「データが変更される前の状態を記録するログ」

5.2.5.2.2.3.2. ロールバック（ROLLBACK） を実行する際に使用される。

5.2.5.2.2.3.3. トランザクションの途中で障害が発生した場合、更新前のデータに戻して整合性を保つ。

5.2.5.2.2.3.4. 例

5.2.5.2.2.3.4.1. 口座残高 10000円 → 8000円 に更新したとき、更新前の10000円を記録。 トランザクションが失敗した場合、ログを参照して10000円に戻す（ロールバック）。

5.2.5.2.2.4. （2）更新後ログ（After Image）

5.2.5.2.2.4.1. 「データが変更された後の状態を記録するログ」

5.2.5.2.2.4.2. ロールフォワード（ROLLFORWARD）（障害復旧）に使用される。

5.2.5.2.2.4.3. システム障害後、バックアップファイルを復元した後、更新後ログを適用して最新状態に戻す。

5.2.5.2.2.4.4. 例

5.2.5.2.2.4.4.1. 口座残高 10000円 → 8000円 に更新したとき、更新後の8000円を記録。 システム障害から復旧後、ログを参照して8000円に更新（ロールフォワード）。

5.2.5.2.2.5. 更新前ログと更新後ログの関係

5.2.5.2.3. チェックポイント

5.2.5.2.3.1. 「データベースが整合性のある状態であることを保証するためのタイミング」

5.2.5.2.3.2. 一定間隔ごとにデータベースの状態を保存し、障害時の復旧をスムーズにする仕組み。

5.2.5.2.3.3. チェックポイントより 前 のトランザクションは確実にディスクに書き込まれているため、障害時には チェックポイント以降のログのみを使って復旧すればよい。

5.2.5.2.3.4. 例

5.2.5.2.3.4.1. チェックポイントが午前2時に記録され、障害が午前3時に発生。 データベースは 午前2時時点のバックアップをロードし、午前2時以降のログを適用して最新の状態に復元 する。

5.2.5.2.4. まとめ

5.3. SQL

6. 注意点・その他

6.1. データベースは概念を掴もう！

6.2. SQLはとにかく手を動かして使ってみるのが大事！