負荷分散と仕様変更に耐えるための*DB*設計

2014/4/12

第3回中国地方DB勉強会in福山

よくある仕様変更

- 「この機能がないと動かないから…」
  - 要求定義漏れ、設計漏れ、データの漏れ
- 「こういう使い方もしたいから…」
  - 便利機能の追加
  - 鶴の一声で現場が逆らえないことも
- 「イメージと違う…」
  - 究極のちゃぶ台返し

*DB*の仕様変更

- DBはシステムの根幹
  - 仕様変更は影響範囲が広く、致命的結果を伴う
- 安全策がとられる
  - 不要なものを消さずに追加で誤魔化す
  - 歴史的背景を持った理解不能なテーブルが作られる
    - 分からないので、安全策がとられる
    - （以下、再帰呼出）
- 誰にもメンテできなくなる

*DB*の仕様変更を防ぐには

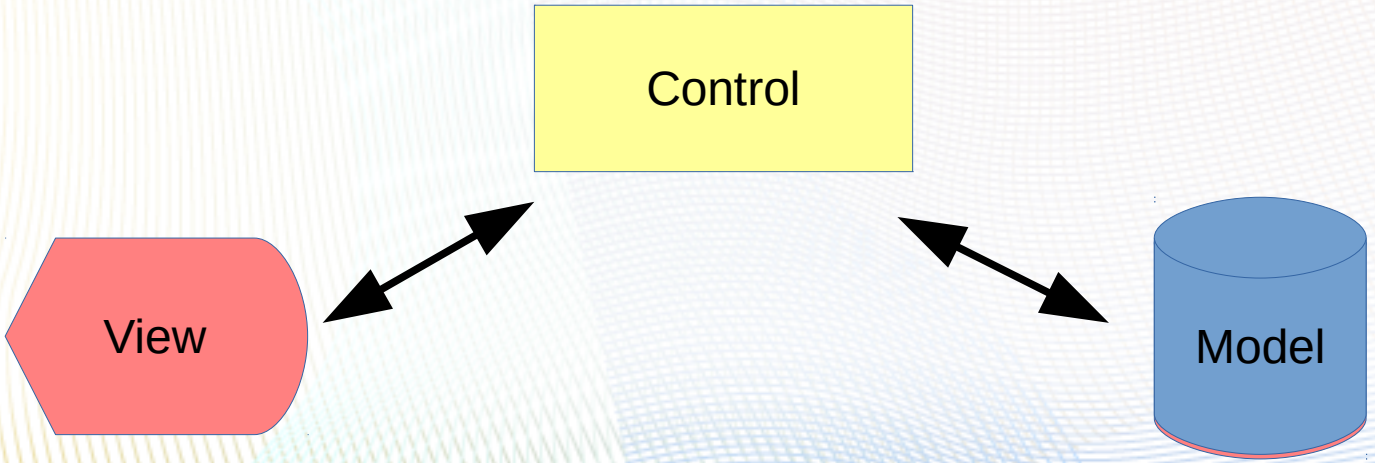
- 無理
  - 使われているソフトは変化する
    - 仕様変更のないソフトは使われていないソフト
- 仕様変更の影響を受けにくい設計はできる
  - 仕様変更が起こりやすい箇所に注意して設計する

対策：データ項目の冗長性

- 文字列型
  - char(n), vchar(n)
  - text
- 日付型
  - date, time, timestamp
  - timestamp with timezone

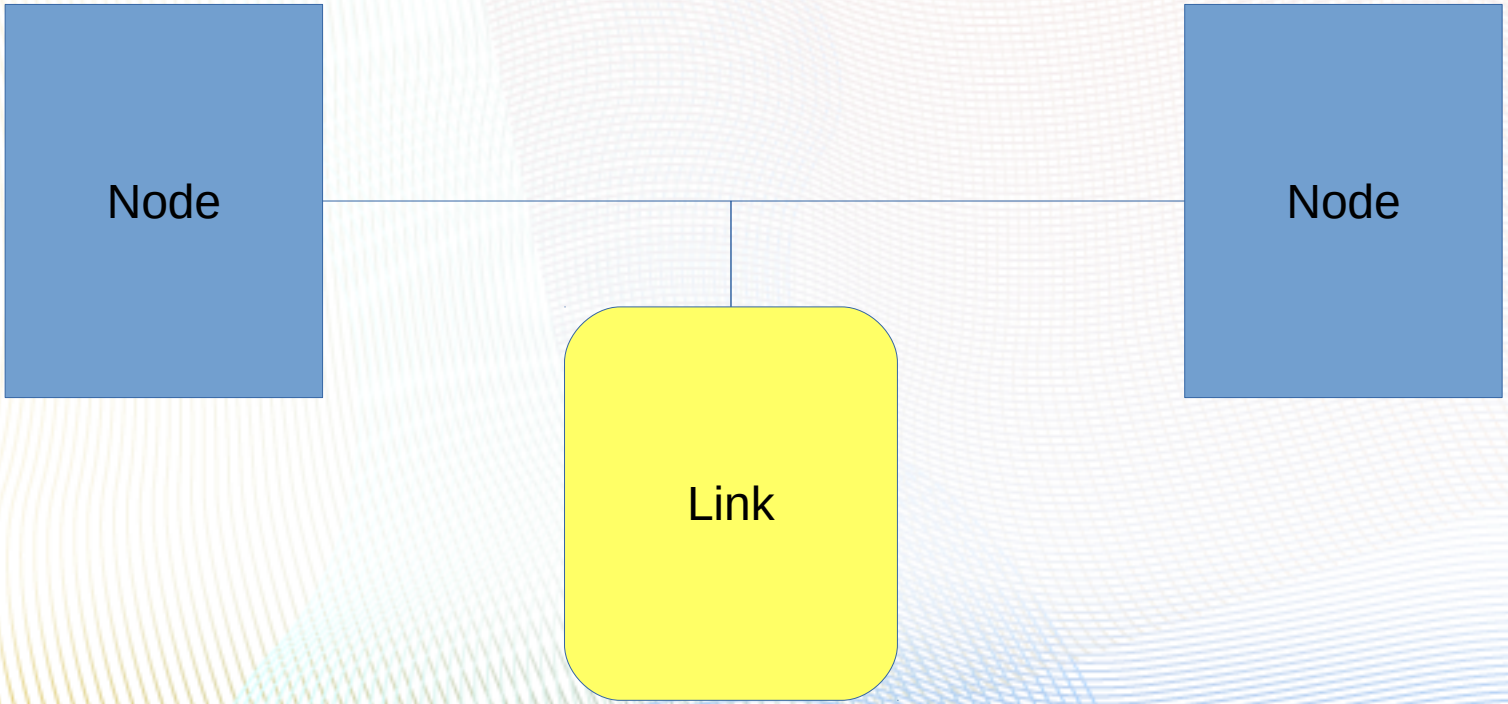
対策：テーブル定義

- よくあるテーブル定義
  - Excelシートのようなテーブル定義
  - 画面の表示項目がテーブルになっている
    - MVCに分けている意味が無い
      - Viewが最も仕様変更を受け易い



対策：テーブル定義

- オブジェクトをテーブルにする
  - ノードとリンクに分けて設計する
    - 仕様変更が発生するのはリンク



隠れた課題：パフォーマンス

- 正規化のし過ぎ
  - JOINが多くなる
- データの集中化し過ぎ
  - 大きなテーブルはボトルネックになる
  - JOINは掛け算
    - 10 × 10,000,000 × 500

対策：負荷分散

- データの肥大化
  - 大きくなるテーブルは小分けする
- セッション負荷の増加
  - 参照負荷
    - スケールアウト
  - 更新負荷
    - スケールアップ
      - 予算が必要

スケールアウトの方法

- レプリケーション
- テーブルスペース
- パーティショニング
- パラレルクエリ
  - PostgreSQLには(まだ)無い

方法１：レプリケーション

- クエリーベース
  - Pgpool,PGCluster,Usogresなど
- トリガーベース
  - slony
- ログベース
  - PostgreSQL9.0から実装

Streaming Replication

- WALを使ったレプリケーション
  - 手動で取得
  - 自動的にストリーミングに流す
- シングルマスタ + マルチスタンバイ
- 非同期
- 同期
  - 9.2から

使い方：マスタサーバの設定

- pg_hba.confにreplication権限の追加
  - host replication user IP/mask trust(md5)
- postgresql.confにレプリケーション設定
  - wal_level = hot_standby
  - max_wal_sender = 2　　#スタンバイ機＋１
  - wal_keep_segments = 8 #8-32

使い方：スタンバイサーバの設定

- 基本的にマスタサーバと同じ
- recovery.confにスタンバイの設定
  - standby_mode = 'on'
  - primary_conninfo =
    - 'host = マスタDBホスト名
    - port=マスタDBポート番号
    - user=replication権限を持つユーザ
    - password=上記ユーザのパスワード
  - recovery_target_timeline='latest'
    - フェイルオーバー用

使い方：初回データ同期

- pg_basebackup コマンド
  - -h マスタDBのホスト名
  - -p マスタDBのポート番号
  - -U マスタDBのユーザ名
  - -D スタンバイDBのデータクラスタ
  - --xlog バックアップにWALを含む
  - --checkpoint=fast チェックポイントモード
  - --progress 進行状況表示

使い方：フェイルオーバー

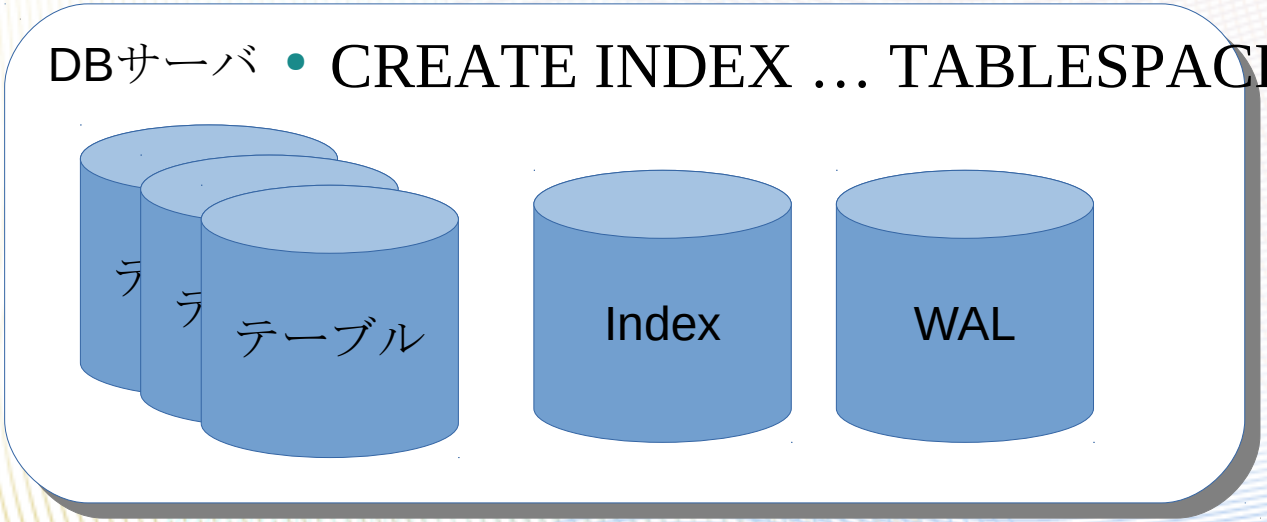
- pg_ctl promote
  - 9.1以降
- recovery.conf
  - trigger_file=トリガーファイル
  - トリガーファイルができると、マスタに昇格

使い方：状態確認

- pg_stat_replication
  - SELECT state FROM pg_stat_replication
    - startup : 接続の確立中
    - backup : pg_basebackup によるバックアップの実施中
    - catchup : 過去の更新を反映中
    - streaming : 更新をリアルタイムに反映中

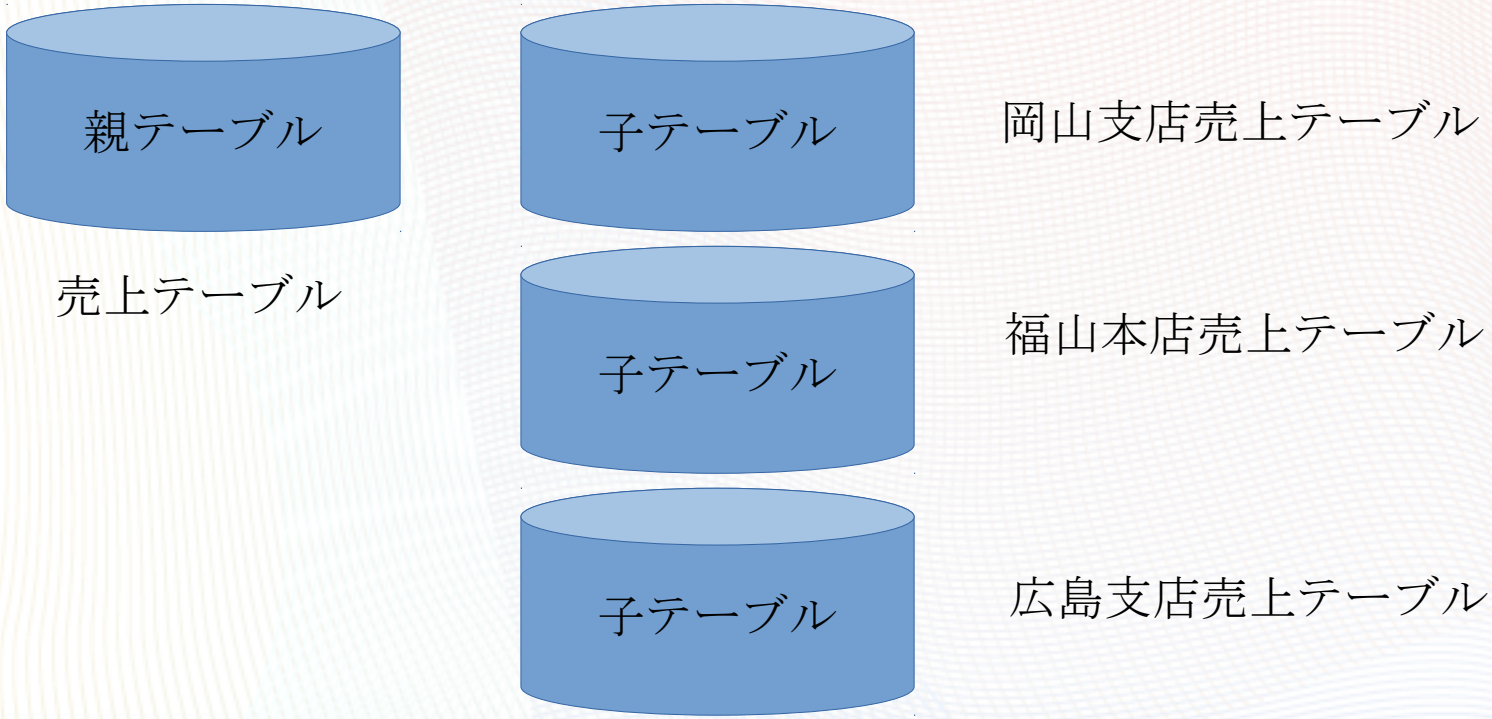
方法２：テーブルスペース

- DB,Table,Indexの物理的な格納場所指定
  - CREATE TABLESPACE 名前 LOCATION '場所'
    - CREATE DB … TABLESPACE 名前
    - CREATE TABLE … TABLESPACE 名前
    - CREATE INDEX … TABLESPACE 名前



方法３：パーティショニング

- DBサーバ内で論理的にテーブルを分割する
- テーブル継承と制約によりテーブルを分割



使い方：親テーブル

- 親テーブルは通常のテーブル
  - マスタテーブルの作成
    - CREATE TABLE sales (
      blanch_id text not null,
      sales int,
      udate timestamp with timezone
      );

使い方：子テーブル

- 子テーブルへのSELECT,UPDATE,DELETEは自動的に分割してくれる
- ただしオーバーヘッドあり
  - 子テーブルの作成（CHECK制約付き）
    - CREATE TABLE sales_001 (
      LIKE sales INCLUDING INDEXES
      INCLUDING DEFAULTS
      INCLUDING CONSTRAINTS,
      CHECK (city_id == '001')
) INHERITS (sales);

使い方：トリガー

- INSERTはトリガで行う
  - CREATE FUNCTION sales_insert_trigger()
    RETURNS TRIGGER AS

    $$

    DECLARE

      part text; -- 子テーブルの名前

    BEGIN

      part := 'sales_' || new.blanch_id; -- キー値から計算 : sales_branch_id

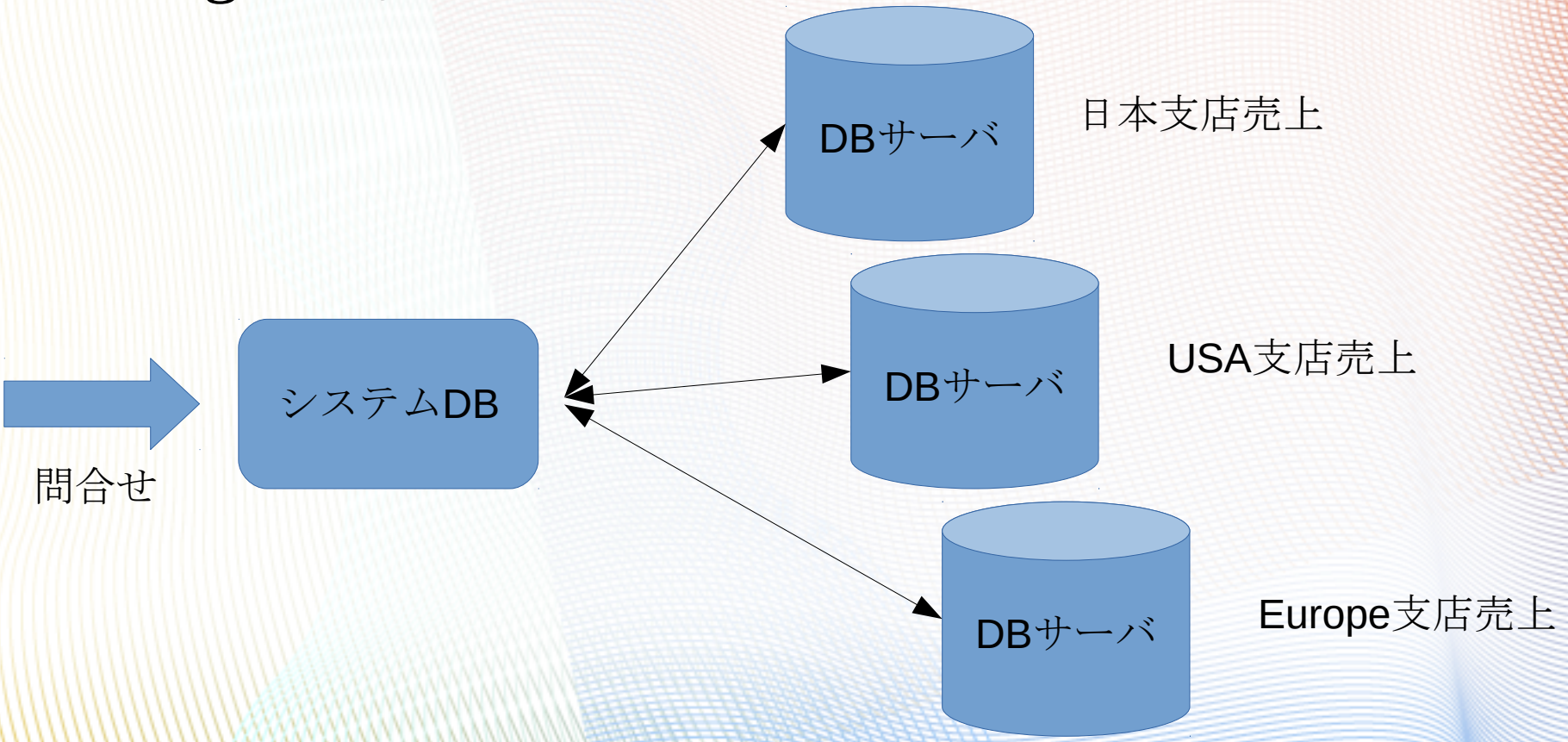
      EXECUTE 'INSERT INTO ' || part || ' VALUES(($1).*)' USING new;

パーティショニングの注意点

- プライマリキーは継承されない
- 分割しすぎるとオーバーヘッドの方が大きくなる
  - コア数/スレッド数が目安
- 分割キーの変更を動的にできない

方法４：パラレルクエリ

- DBを分散（物理的なパーティショニング）
- PostgreSQLには未実装



使い方：パラレルクエリ

- Pgpool-II + DBLinkを使用
  - システムデータベースを作成
    - pgpool_catalog.dist_defの定義
      - 分散ルールを格納するテーブル
      - 分散ルールは関数で定義、登録
    - pgpool_catalog.replicate_defの定義
      - レプリケーションを行うテーブルの情報(複製ルール)
- 動的な変更はできない

まとめ

- 仕様変更に強い設計
  - オブジェクト指向に基づいたDB設計
- パフォーマンス対策
  - レプリケーション
  - テーブルスペース
  - パーティショニング
  - パラレルクエリ
  - SQLのブラッシュアップ