Club Db2 第146回

達人が語る こんなデータベース設計はヤダ！

講演者:ミック

プロフィール

経歴：SIer勤務。性能設計やチューニングを専門にしています
　　サイジング、ベンチマーク、性能試験、火消し

著書：『達人に学ぶ DB設計徹底指南書』
　　『プログラミング学習シリーズ SQL』
　　『達人に学ぶ SQL徹底指南書』

Twitterアカウント：copinemickmack

データベース設計とは何か

データベース設計の三つのレイヤー：

1. 物理設計
   いわゆる「基盤（インフラ）」に関わる設計
   サーバ、ストレージ、アーキテクチャ、サイジング

2. 論理設計
   データモデルに関わる設計
   ER図、モデリング、処理方式

3. 実装設計
   実装特有の制限や機能を意識した設計

今日取り上げるのは主にここ

今日のキーワード

他は全て忘れても、これだけは覚えて帰ってください：

- トレードオフ
  - うまい話には裏がある。
  - 物理 vs 論理
- 手続き型の呪い
  - 「魂を重力に引かれている」的なアレ
  - ファイル ≠ テーブル
- ストレージに触る者は不幸になる
  - ディスクに触ったら負けかなと思ってる

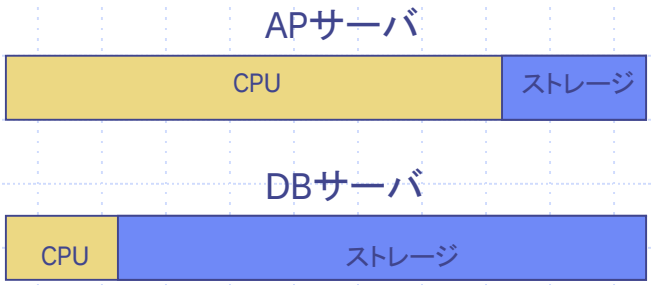
ダメな物理設計

DBサーバとサイジング

サイジングでCPUしか見ない奴、一歩前に出ろ

・サーバサイジングの比重はCPUに置かれている。
⇒主なベンチマーク指標でもCPUの性能しか見ることができない。
ex. SPECint、TPC-C

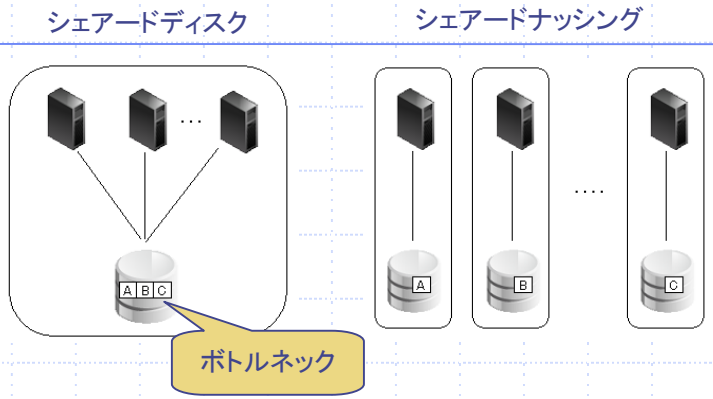
・しかしDBサーバの性能を決めるのは、相対的にストレージ。



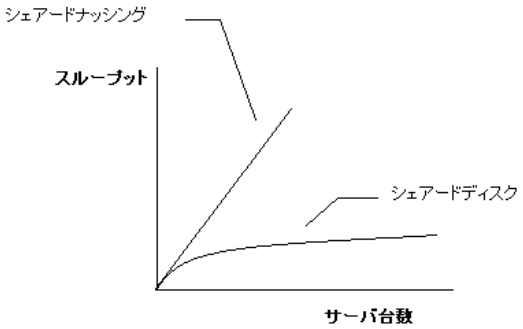
DBの性能はスケールするか？

DBはWeb/APに比べ性能のスケールアウトが困難

・シェアードディスクではストレージがボトルネックポイント

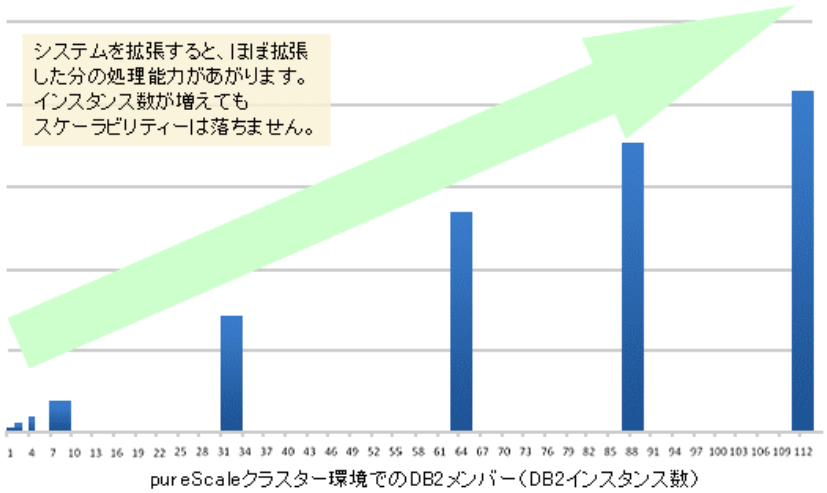


シェアードディスクはスケールしない



参考：PureScaleの場合

・シェアードディスクがスケールするための条件は？

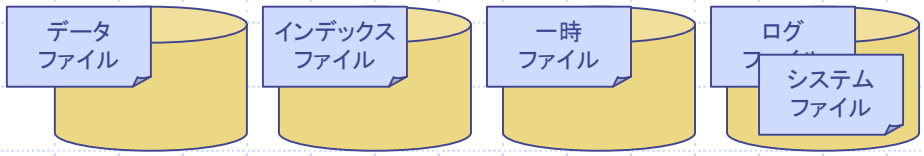


DB2 pureScale ～ オンライン・トランザクションのデータ処理量が増加しても、容易に確実に拡張できます

（http://www-06.ibm.com/software/jp/data/db2/linux-unix-windows/purescale/）

ストレージネックを解消しましょう

・かつてはファイルとストレージの分散が重要でした
⇒RAID、データファイルとインデックスファイルの分散、などなど。



もちろん今でもストレージとファイルの分散は基本ではあるのだが・・・

ストレージの（性能面での）サイジングというのは難しい。

全部メモリに乗っけちまった方がはやくねえか・・・？

オンメモリの時代

- もうディスクは速くならない。
  ⇒1500万回転のディスクが登場する可能性はゼロに近い
- メモリが安価になり、SSDなども実用化されてきた
  ⇒最近はTBクラスのメモリを搭載できるサーバも登場
  大金を出さなくても性能が手に入る時代へ
- シェアードナッシングはやはり一部にしか適用できない
  ⇒当分はDWH専用

“ディスクに触ったら負けかなと思ってる”

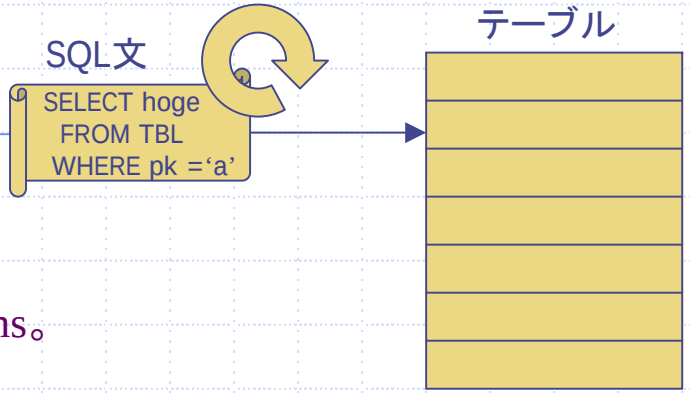
32bit OSはオワコン

- Linux/Unix OSはもうほとんど64 bit だが、Windows Server にはまだ 32bit が現役。
  ⇒通称「2GBの壁」によってメモリを使えない。

ダメな論理設計

手続き型の呪い

ループが便利すぎて困る



“カーソル”の功罪

利点：手続き型言語と同じ発想なので
　　プログラミングが簡単

欠点：パフォーマンスが悪い。
　　SQLの実行速度は速くても0.1～1ms。

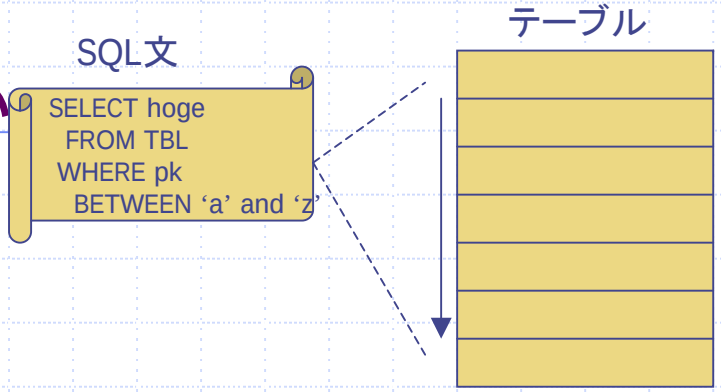
“データベースにおいて、カーソルのパフォーマンスを改善する最高の方法を教えよう。カーソルを使わないことだ。SQL エンジンは、集合操作のために設計されており、個々の行を扱うよりも行の集合をひとまとまりに扱う方が得意だ。”

（J.Celko “SQL for Smarties” 4th ed. Ch.5.）

テーブルはガツンと一発で操作

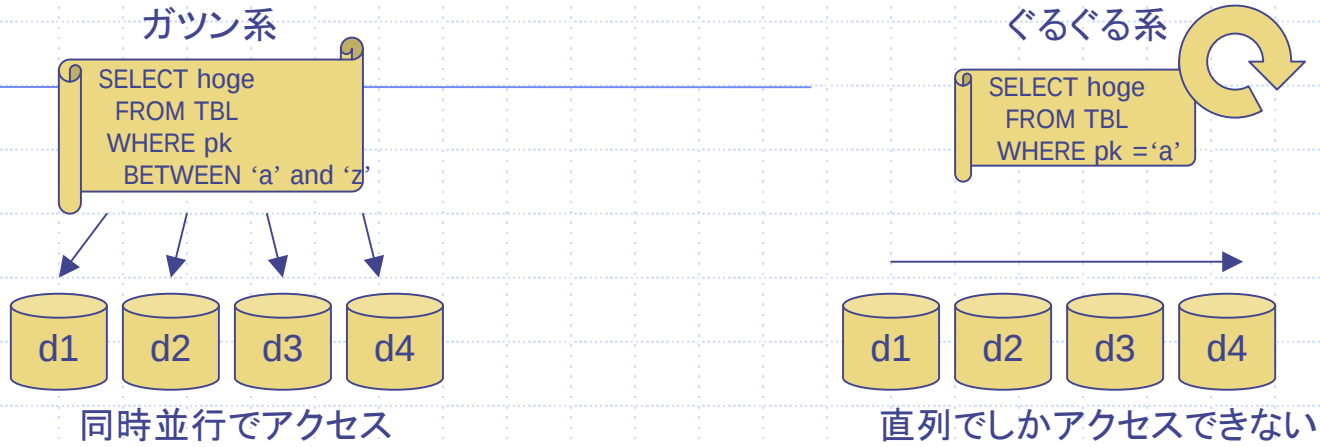
常に複数行を一度に操作する意識を持つ

テーブルはファイルではない



なぜループは遅いのか

最大の原因：せっかくの並列アーキテクチャ(ex. RAID、シェアードナッシング)を敢えて直列化してしまうから



⇒その結果、ガツン系の処理時間は（理想的には）$O(1)$
だが、ぐるぐる系はデータ量nに対して$O(n)$になってしまう。

これ以外にも、ぐるぐる系は、マルチブロック読み込み、プリフェッチキャッシュといったミドルやハードの機能を利用できないことも不利な条件。

ぐるぐる系は“詰む”

「ぐるぐる系のAPの性能が出ない」という相談をよく受けるが・・・

・ミック「じゃあループの粒度をもっと大きくしましょう」
　PJ「今さらAPを変更できない」
　PJ「SQLを速くしてよ」
ミック「SQLはもう極限まで速いです」
　PJ「・・・」
ミック「・・・」
　PJ「え、じゃあ、どうすりゃいいの？」

さて、どうすればよいでしょう？

テーブルはファイルじゃねえんだよ

人はなぜSQLをループさせたがるか

・テーブルをファイルだと思っている。
⇒ファイルは手続き型言語でループさせるもの
⇒ならばテーブルもループで一行ずつ処理すればよい

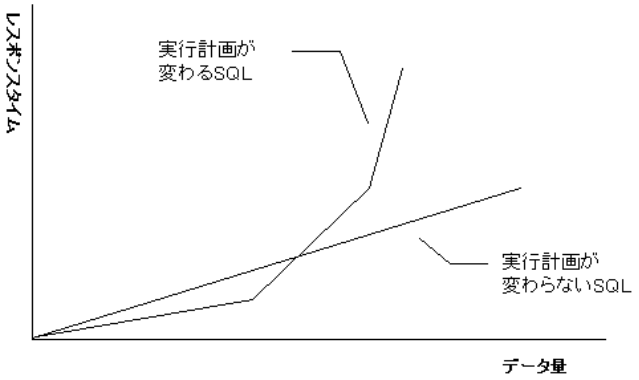
“ぐるぐる系はDBMSとハードウェアの
進歩に敢えて背を向ける時代錯誤”

ぐるぐる系からの反論

「でもさ、SQLがシンプルであることには利点もあるんじゃないの？」

この意見には一理ある。

- 結合を使用したSQLの実行計画「揺れ」
- オプティマイザが最適な実行計画を選択してくれない



- 派生する議論に「統計情報を凍結するべきか」

オプティマイザと統計情報

最近は結構頭がよくなってきているのだが・・・

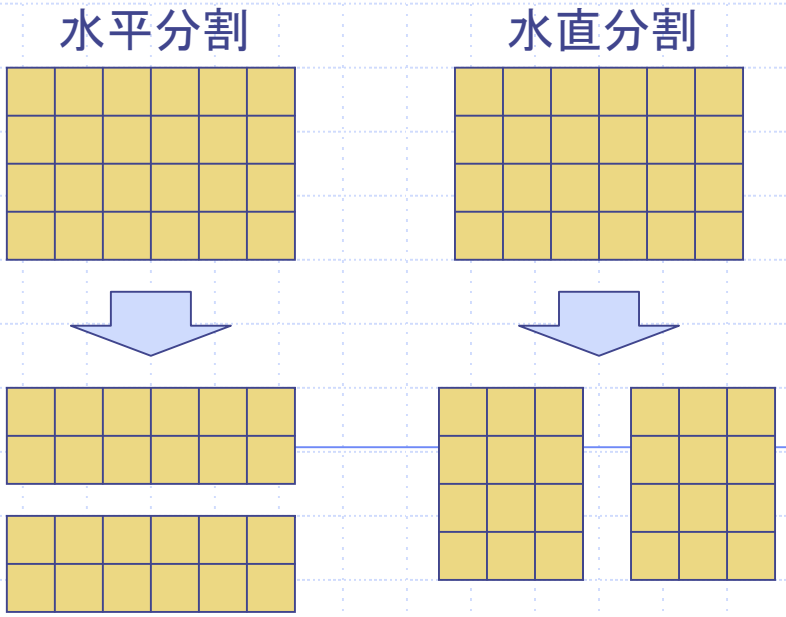
・色々細かい情報まで加味して判断するがゆえに
逆に揺れてしまうという問題も。

・もともとRDBはバッチのような大量データ処理をさせる
のに向いたアーキテクチャではない。

垂直分割と水平分割

物理と論理のトレードオフ

- 論理レベルにおいて、垂直分割と水平分割を擁護する理由は一つもない。



“ファイルは物理的存在だが、
テーブルは論理的存在である”

（J.Celko “SQL for Smarties”
4th ed. Ch.2）

物理の犠牲になる論理

物理と論理が喧嘩すると、物理が勝つ

・ディスクI/Oを減らすための手段として、論理が犠牲になることは多い。
しかし本来は、それは物理レベルで解決されるべき問題。

正規形はいつ崩すべきか

論理モデルの変更は手戻りのコストが大きい

・「原則として正規化すべし」
　⇒正規化して性能が出なかった場合、すでに
　　設計を変更するのは手遅れではないのか。

・SQLチューニングとデータマートによる解決が可能か。

関係モデルと非親和的な設計

ナチュラルキー VS サロゲートキー

・サロゲートキーが許される局面

・オートナンバリングとSQLの親和性（または非親和性）
- IDENTITY列
- シーケンスオブジェクト
- ROW_NUMBER関数

・・・手続き型への回帰

RDBの理想と現実

DBエンジニアはどこまで物理を意識するべきか？

・ビュー ⇔ テーブル ⇔ ファイル ⇔ ディスク(ストレージ)
- 「ビューまでしか意識しない」という人はさすがにいない
- 一般的にはファイル／ストレージで断絶が起こる
（実際、ここが責任分界点というシステムは多い）
- 「こんなところで責任分界できるか」というのが性能屋としての本音

・かといって妙に物理を意識した設計をされるのも困り者
- 単純に実装の力不足による問題