DBのパフォーマンスについて

DBのパフォーマンスを決める要因

- 正規化
- インデックス
- 統計情報

DBのパフォーマンスを決める要因

- 正規化
- インデックス
- 統計情報

正規化

- データの重複を排除し、「矛盾」の発生を「設計レベル」で防ぐこと
- テーブルを分割することで、データの整合性、管理しやすくする

https://qiita.com/k_kind/items/d2651cd2035ec3589df1

正規化のステップ

第１正規化：
テーブル中に複数の値をもつようなデータ項目を含まないという条件を満たすように整理する。

第２正規化：
第１正規化後のテーブルに、主キーの一部の項目だけに従属するような項目を含めないように整理する。

第３正規化：
第２正規化後のテーブルに、主キー以外の項目に従属するような項目がなくなるように整理する。

https://ssaits.jp/promapedia/technology/normalization.html

正規化のデメリット

- 結合が発生するのでパフォーマンスは下がる

https://qiita.com/trail0738/items/d7ac5cafb9e4d7971033

非正規化のメリット・デメリット

- インデックスがより機能し検索のパフォーマンスが上がる
- 更新処理を行う際には、非正規化した分、負荷がかかるようになる
- **第３正規化までは行う**

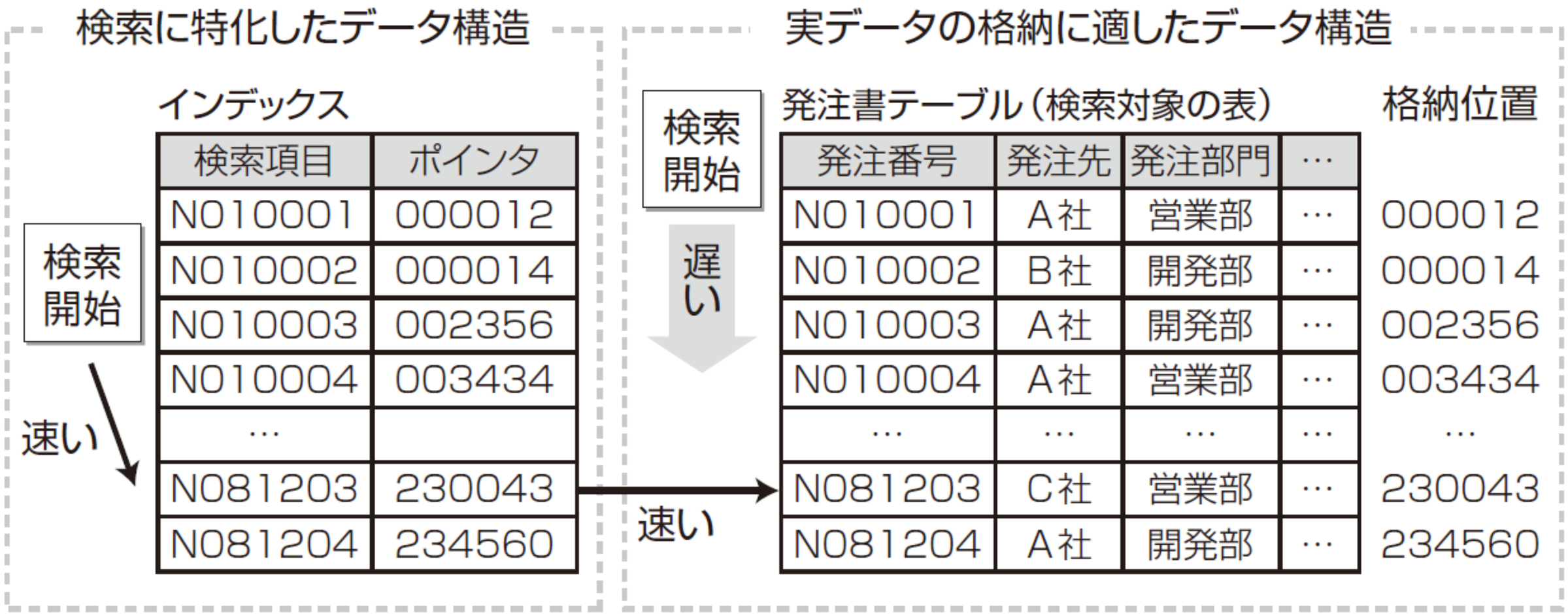
DBのパフォーマンスを決める要因

- 正規化
- インデックス
- 統計情報

インデックス

- 目次のようなもの
- キー値と実データ、実データへのポインタ
- データの格納位置を特定し、その位置を直接アクセスする事で、表の検索速度を上げることができる

インデックスによる性能向上のイメージ



https://atmarkit.itmedia.co.jp/ait/articles/1703/01/news199.html

インデックスのメリット

- テーブルのデータに影響を与えない
- インデックスを作ることでデータの中身が影響を受けることはない
- テーブルの構造を変える必要もない

インデックスのメリット

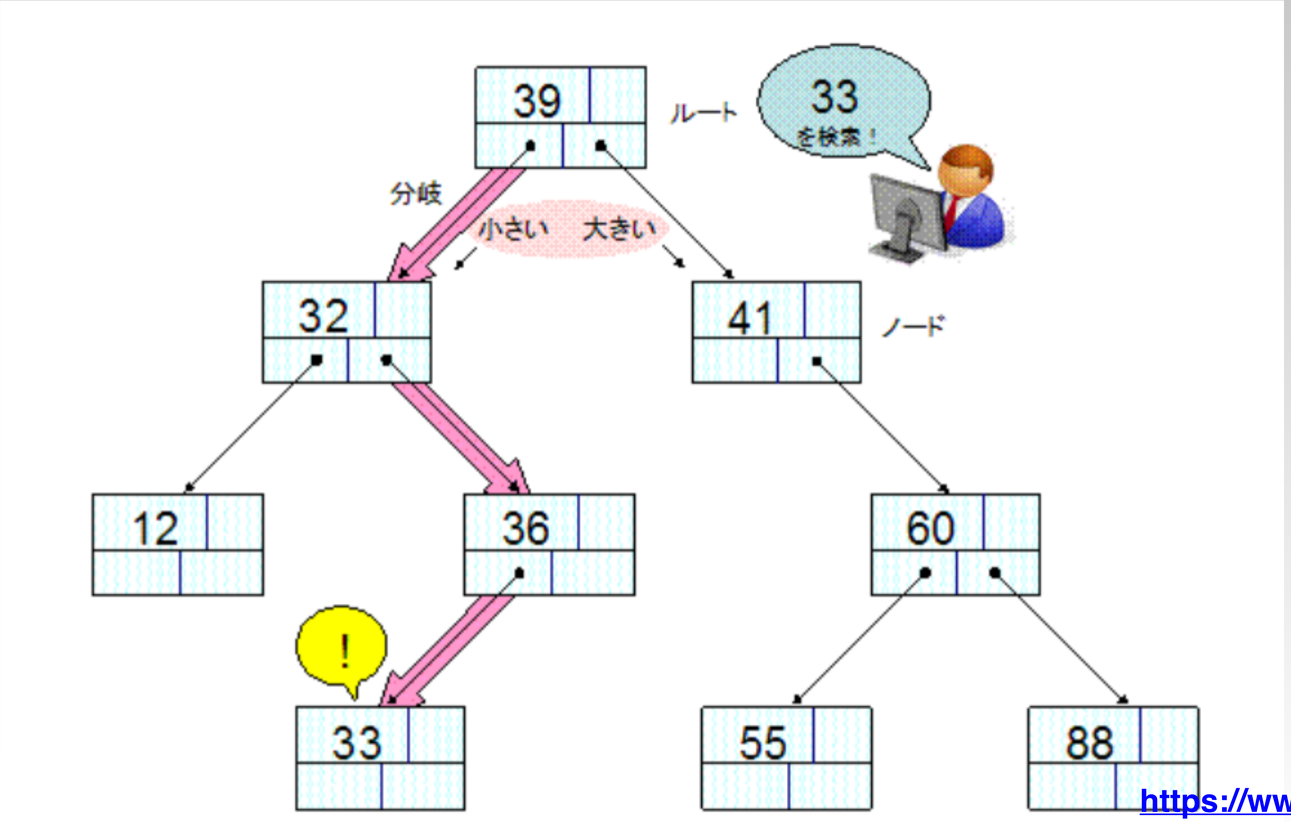
- 大きな性能改善効果
- インデックスの性能がデータ量に対して劣化しにくい

インデックスの種類

- B-treeインデックス
- ビットマップインデックス
- ハッシュインデックス

B-treeインデックスの仕組み

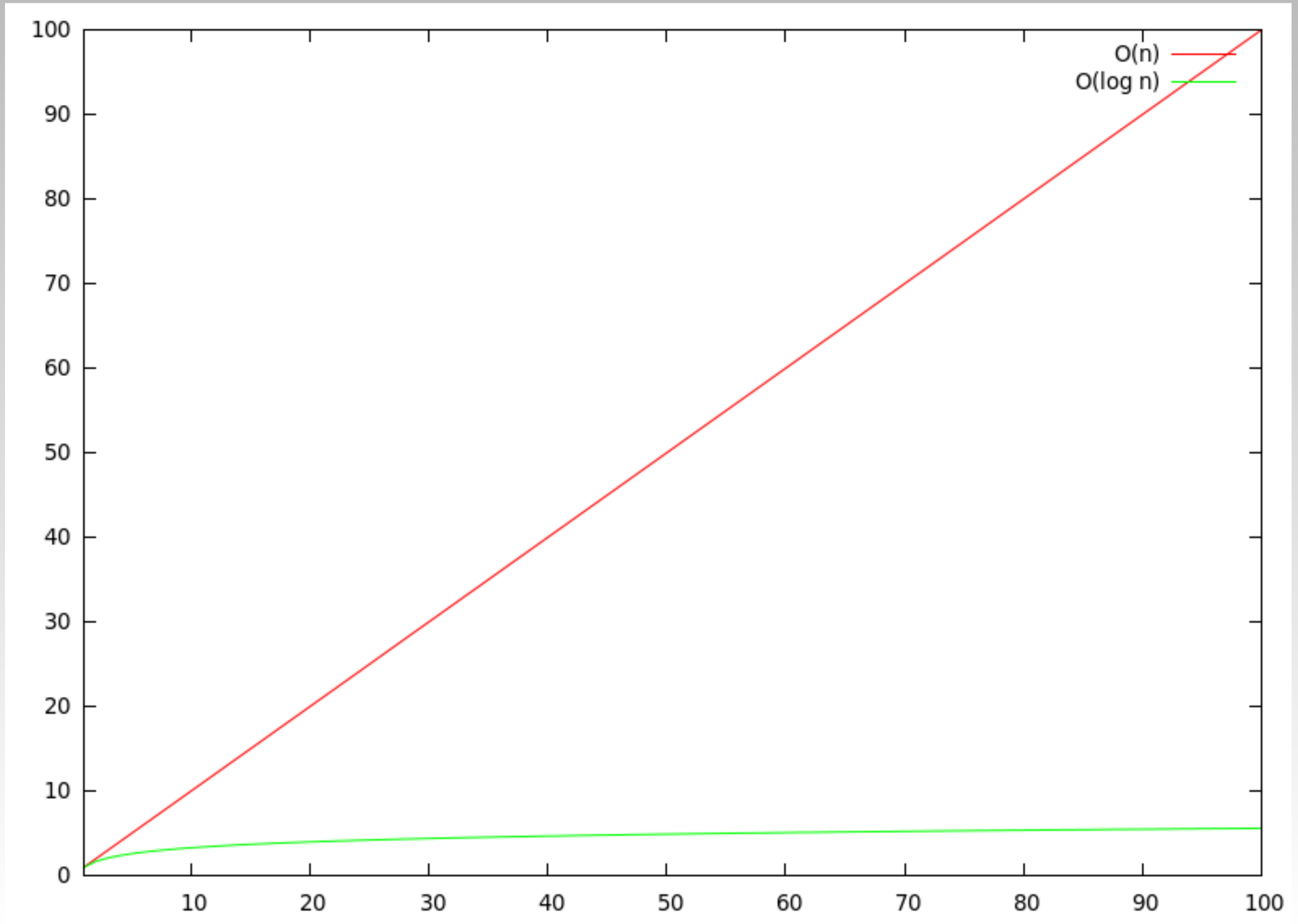
- tree=木構造でデータを保持する



https://www.mtioutput.com/entry/what-is-btreeplus

om/kiyodori/items/f66a545a47dc59dd8839

- フルスキャン＝O(n)
- B-treeインデックス=O(log n)

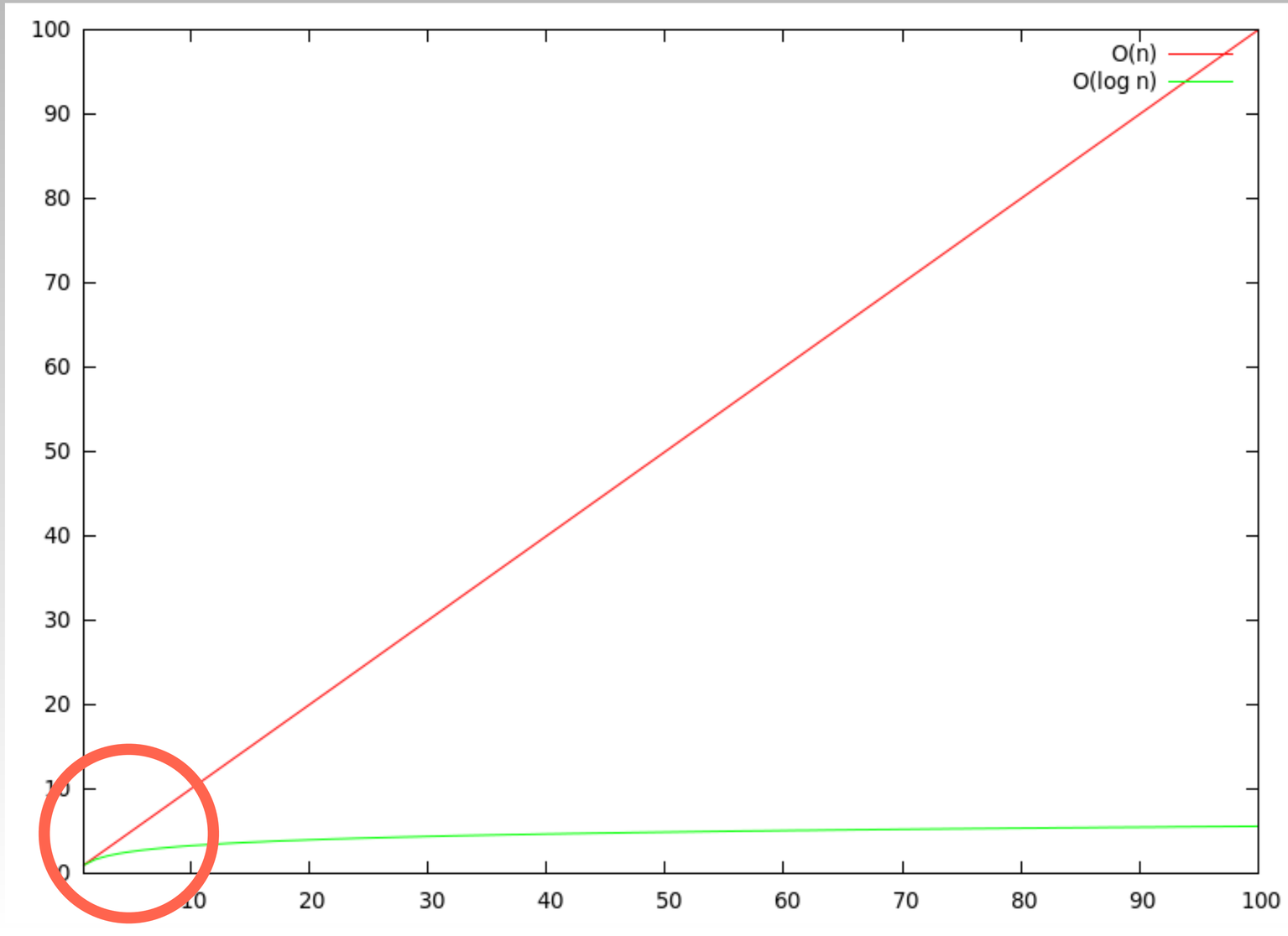


インデックスの設計方針

- 大規模なテーブルに対して作成する
- カーディナリティの高い列に作成する
- 選択条件、結合条件に使用されている列に作成する

インデックスの設計方針

大規模なテーブルに対して作成する



インデックスの設計方針
カーディナリティの高い列に作成する

- カーディナリティとは「どのくらいの種類の多さをもつか」の概念

インデックスの設計方針
カーディナリティの高い列に作成する

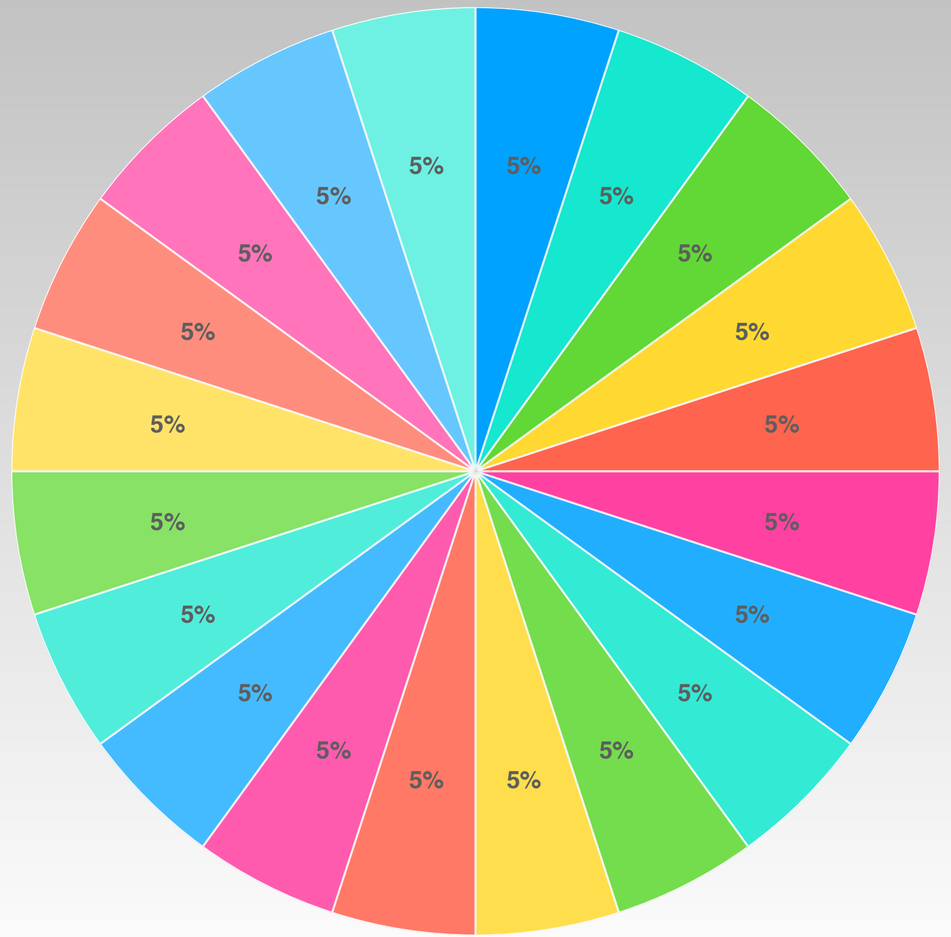
- 性別
- 男性・女性・不詳
- 種類が少ない=カーディナリティは低い

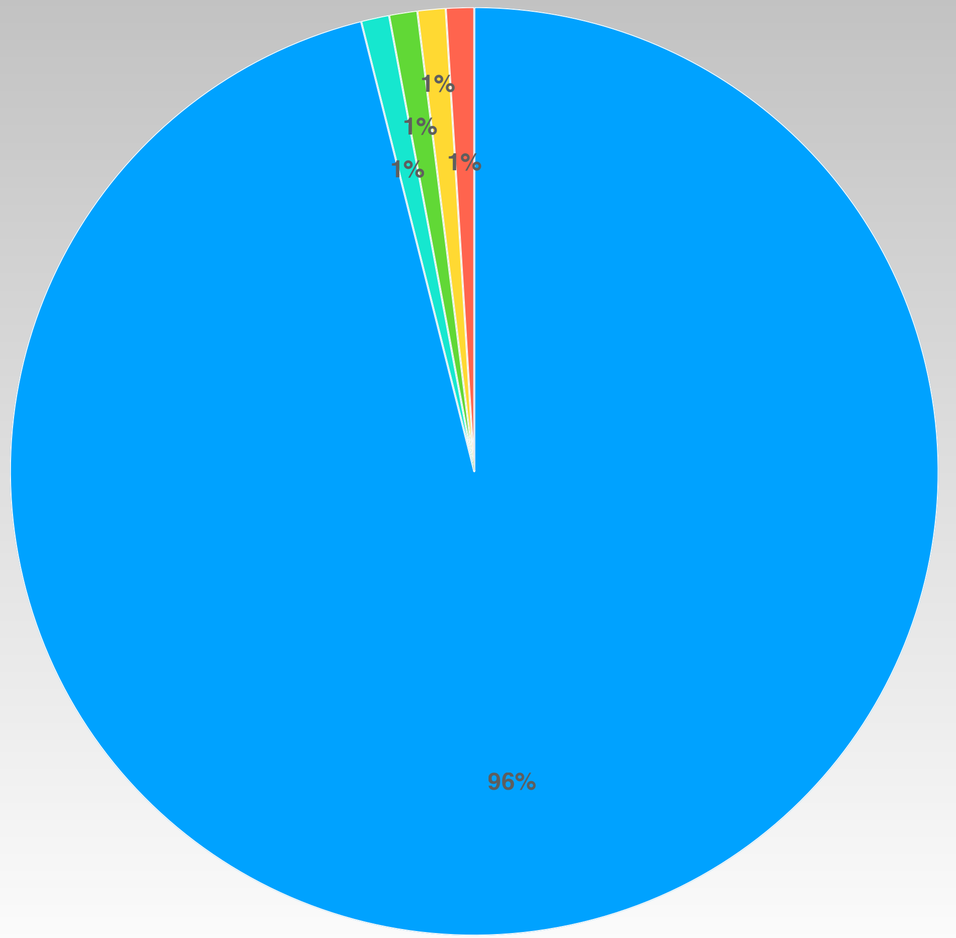
インデックスの設計方針
カーディナリティの高い列に作成する

- 口座番号
- 受付日
- たくさんの種類がある=カーディナリティは高い

インデックスの設計方針

カーディナリティの高い列に作成する





インデックスの設計方針
カーディナリティの高い列に作成する

- 特定の値にデータが集中している列はNG
- ある程度値が平均的に分散しているのがベスト

インデックスの設計方針

選択条件、結合条件に使用されている列に作成する

- 検索条件、結合条件として使用されない列に作っても意味なし

インデックスの設計方針
選択条件、結合条件に使用されている列に作成する

- インデックス列に演算を行っている(WHERE col * 1.1 > 100)
- 否定系を用いている(WHERE col <> 100)
- ORを使う(WHERE col = 99 OR col = 100)
- 後方一致、中間一致のLIKEを使っている(WHERE col LIKE ‘%a’)

インデックスの設計方針

選択条件、結合条件に使用されている列に作成する

- インデックスを作成した列は「そのままの値」で用いる
- 否定系を使わない
- 複数検索したいときはINを使う
- LIKEを使うときは「前方一致」

インデックスの注意点
主キー、一意制約の列には作成不要

- 内部的にインデックスを作成しているため

インデックスの注意点
インデックスは更新性能を劣化させる

- インデックス列の値が更新されると、インデックス内の値も更新されるため
- 作成すればするほど、更新性能が落ちる=トレードオフ

データベースのパフォーマンスを決める要因

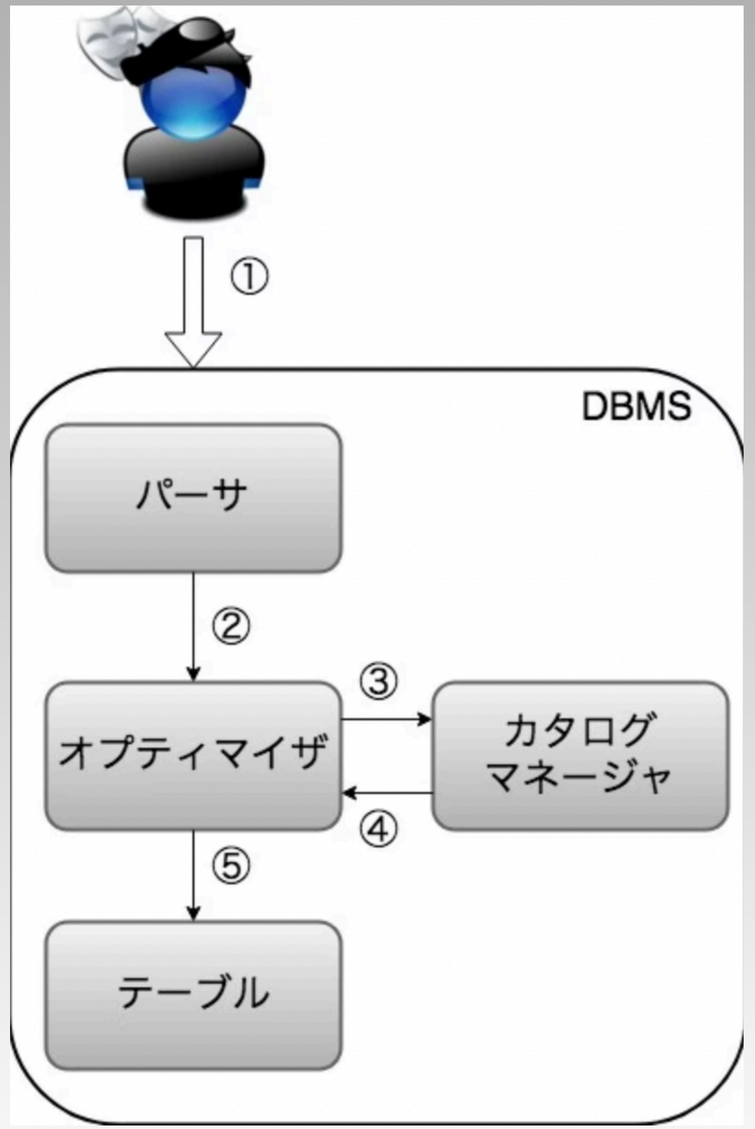
- 正規化
- インデックス
- 統計情報

統計情報

- テーブルやインデックスなど「データ」についてのデータ（メタデータ）
- DBMSはこのメタデータを頼りにSQLのアクセスパスを決定する

統計情報

- ①SQL文がパーサ（構文チェック）に渡される
- ②オプティマイザに送る（実行計画を決める）
- ③カタログマネージャ（統計情報管理）に照会
- ④最短経路を決定
- ⑤テーブルにアクセス



統計情報は何から作られるか

- 各テーブルのレコード数
- 各テーブルの列数と列のサイズ
- 列のカーディナリティ
- 列の値の分散具合
- 列内のNULLの数
- インデックス情報

統計情報

- SQLのパフォーマンスを上げるには
- 統計情報の情報を常に最新の正しい情報にするor凍結する
- 統計情報をもとに作られた実行計画を理解し、間違った道を教えていないか確認する
- 実行計画をもとに正しいインデックス情報が作られているか確認する

https://qiita.com/yoshinori_hisakawa/items/f5d608d302ede8f9883d

まとめ

- 正規化にはトレードオフがあるが、第３正規化までは必ず行う
- インデックスはパフォーマンスを上げるために有効な手段だが、使い方に注意が必要
- 定期的に実行計画を見て、正しい列にインデックスが当てられているか確認する
- 設計段階での正規化、実装・運用段階でのSQLチューニングがデータベースのパフォーマンスの鍵を握る！

END