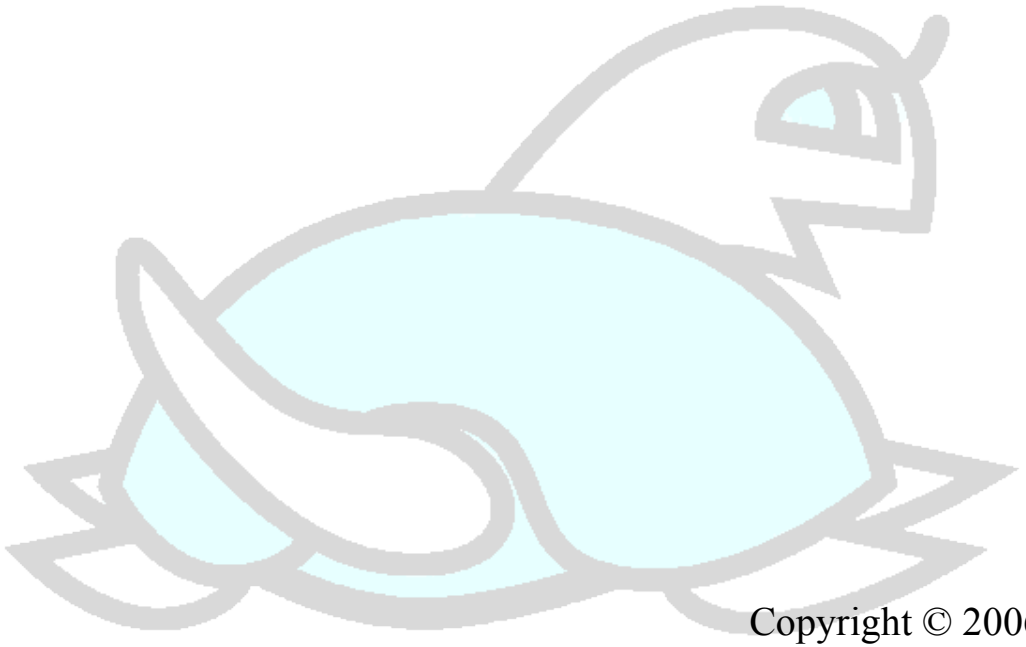
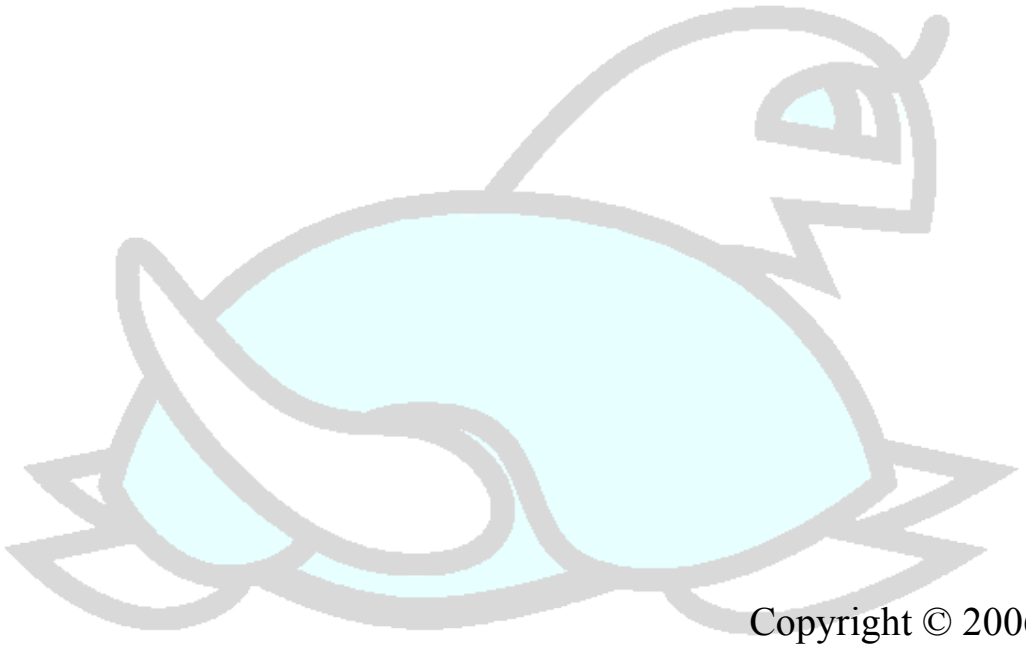
SQLチューニング



チューニングの手順

- 問題箇所はどこか？
  - ログの確認
- 何が悪いのか？
  - 問い合わせプランの確認



ログの確認

- log_statement
  - 実行したSQL文をログに出力する
  - 設定値:none、ddl、mod、all

```
LOG:  statement: SELECT * FROM t2 WHERE b < 20;
```

- log_duration
  - SQL文の実行時間をログに出力する
  - 設定値:off、on

```
LOG:  duration: 1018.215 ms
```

- log_min_duration_statement
  - 遅いSQL文をログに出力する
  - 設定値:-1、ミリ秒

```
LOG:  duration: 1018.215 ms  statement: SELECT * FROM t2 WHERE b < 20;
```

問い合わせプランの確認

- EXPLAIN ANALYZE
  - PostgreSQLが選んだプランの表示
  - 実行時間の報告
- 実行例

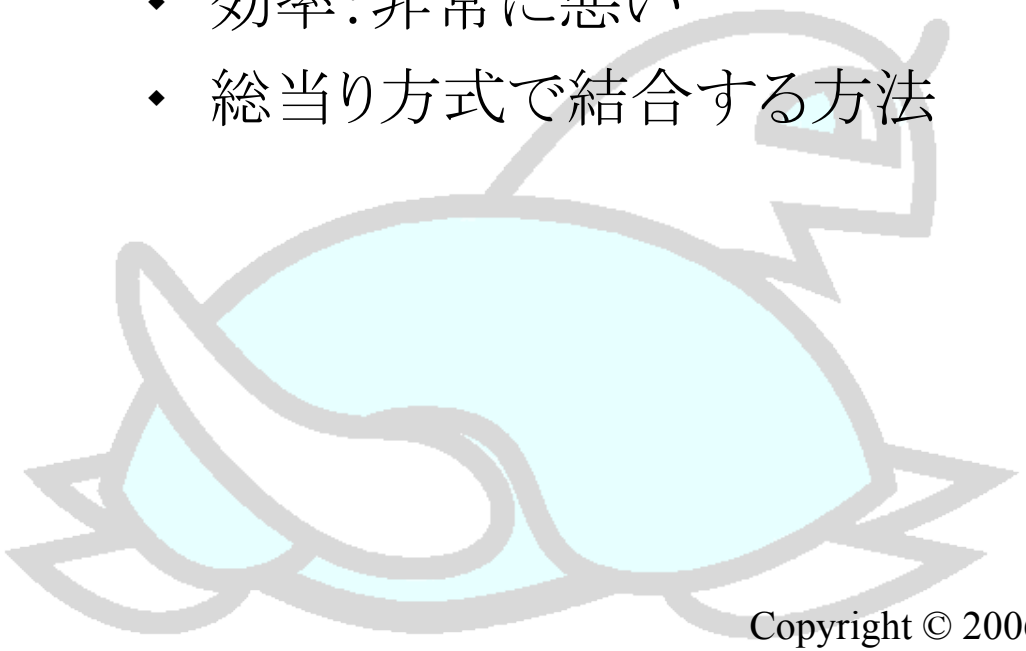
```
kataoka=# explain select * from t1, t2 where t1.a = t2.a;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Merge Join  (cost=0.00..53902.49 rows=1000000 width=24)
   Merge Cond: ("outer".a = "inner".a)
   ->  Index Scan using t1_a_idx on t1  (cost=0.00..19456.00 rows=1000000 width=12)
   ->  Index Scan using t2_a_idx on t2  (cost=0.00..19456.00 rows=1000000 width=12)
(4 rows)
```

スキャンの種類

- インデックススキャン
  - 効率：非常に良い
  - インデックス（索引）を使って高速に検索する方法
  - 検索対象レコードが少ない場合に効果的なアクセス方法
- ビットマップスキャン
  - 効率：割と良い
  - 以下の場合に効果的なアクセス方法
    - → 検索対象レコードが比較的多い場合
    - → 複数のインデックスを使って検索する場合
- シーケンシャルスキャン
  - 効率：悪い
  - テーブルをすべて読みこんで検索する方法

ジョインの種類

- mergejoin
  - 効率:もっとも良い(すぐに一件目が取り出せる)
  - ソートされたもの同士を高速に結合する方法
- hashjoin
  - 効率:良い(一方をすべて読み込むまで一件目が取り出せない)
  - 結合の一方が比較的小さい場合に効率の良い方法
- nestloop
  - 効率:非常に悪い
  - 総当り方式で結合する方法



インデックスとは

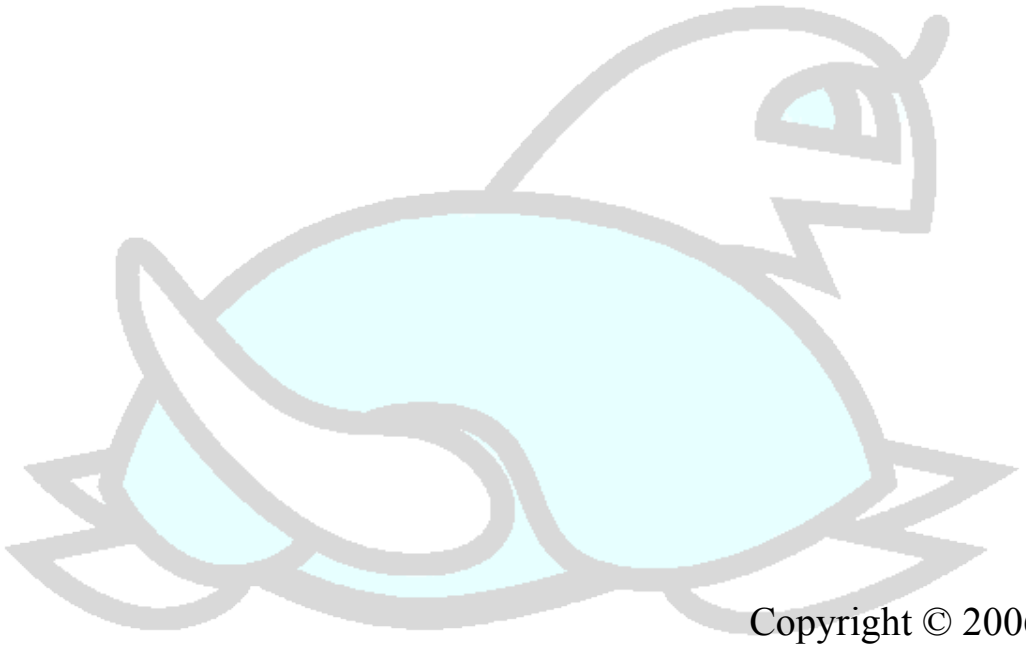
- インデックスとは
  - 効率よく検索するための「索引」
    - → 辞典の「索引」と同じもの
  - 必要に応じて作成する
    - → CREATE INDEX ～
- btreeインデックス
  - 以下の検索時に威力を発揮
    - → 完全一致による検索（a = 10など）
    - → 大小関係による検索（a < 10など）
- hashインデックス
  - 完全一致による検索時に威力を発揮（a = 10など）
  - 現在のPostgreSQLでは性能面で推奨されない

複数カラムでの検索を速くするには

- たとえば以下のような検索の場合
  - a = 10 AND b < 10
- マルチカラムインデックスを作成する
  - CREATE INDEX ～ (a, b)
  - PostgreSQL 8.0まで
    - → bのみを対象とした検索には使えない
  - PostgreSQL 8.1から
    - → bのみを対象とした検索でも使える
- それぞれのカラムに個別のインデックスを作成する
  - PostgreSQL 8.0まで
    - → どれか1つのインデックスのみが使われる
  - PostgreSQL 8.1から
    - → それぞれのインデックスが使われる

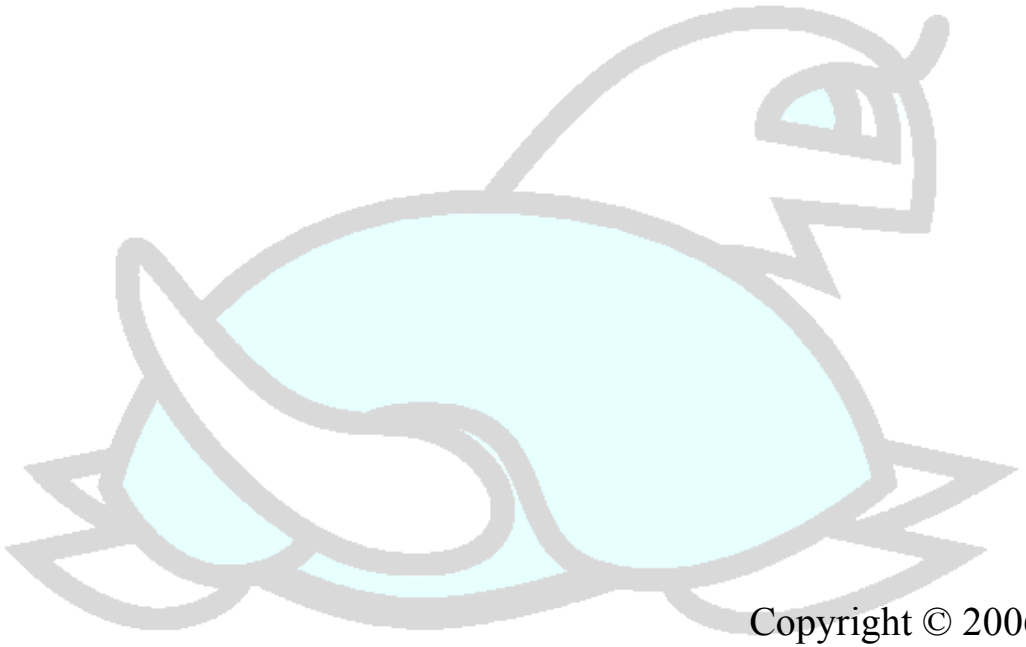
あいまいな検索を速くするには

- 以下のような場合はインデックスが使われない
  - a LIKE '%キーワード%'
  - a SIMILAR TO '%(キーワード|Keyword)%'
  - a ~ '.*(キーワード|keyword).*'
- 前方一致検索ならbtreeインデックスが使われる
  - a LIKE 'キーワード%'
  - a LIKE 'キーワード%です。'



演算を伴う検索を早くするには

- 以下のような場合はインデックスが使われない
  - to_char(a, 'YYYYMMDDHH24MISS') = '20060218130000'
  - to_char(a, 'YYYYMM') = '200602'
- ファンクショナルインデックスを作成する
  - カラム自体ではなく、演算結果でインデックスを作成する
    - ➔ CREATE INDEX ～ (to_char(a, 'YYYYMM'));



ソートを速くするには

- ソートカラムにbtreeインデックスを作成する
  - ORDER BY a
    - → CREATE INDEX ～ [USING btree] (a)
- 複数カラムでのソート
  - ORDER BY a, b, c
    - → CREATE INDEX ～ [USING btree] (a)
    - → CREATE INDEX ～ [USING btree] (a, b)
    - → CREATE INDEX ～ [USING btree] (a, b, c)
  - カラムの順序が違っているインデックスは使われない
    - → CREATE INDEX ～ [USING btree] (b)
    - → CREATE INDEX ～ [USING btree] (b, a)

MAX、MINを速くするには

- 対象カラムにbtreeインデックスを作成する
  - PostgreSQL 8.1から有効
    - ➔ SELECT MIN(a) FROM t1;

```
kataoka=# EXPLAIN SELECT MIN(a) FROM t1;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Result  (cost=0.02..0.03 rows=1 width=0)
   InitPlan
     ->  Limit  (cost=0.00..0.02 rows=1 width=4)
           ->  Index Scan using t1_a_idx on t1  (cost=0.00..19456.00 rows=1000000 width=4)
                 Filter: (a IS NOT NULL)
(5 rows)
```

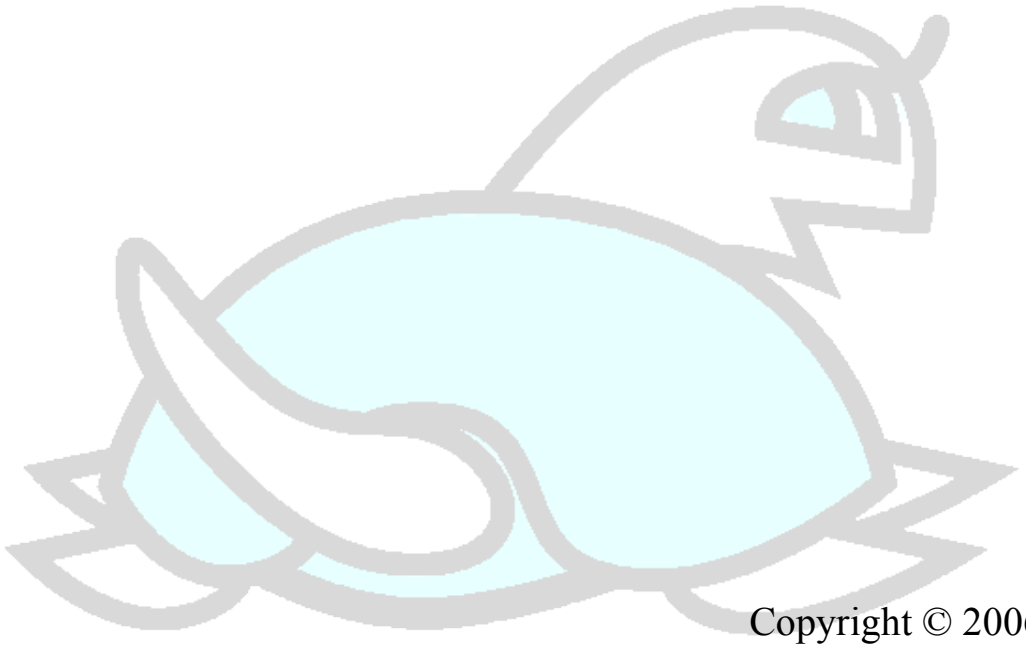
  - PostgreSQL 8.0までの代替方法
    - ➔ SELECT a FROM t1 ORDER BY a LIMIT 1;

結合を速くするには

- 結合のキーとなっているカラムにインデックスを作成する
  - 一方のカラムにのみインデックスを作成すると
    - ➔ プランに制限が発生するので気をつける必要がある
  - 両方のカラムにインデックスを作成すると
    - ➔ プランに自由度がある
- 3つ以上の結合の場合
  - JOIN句を使うと…
    - ➔ プランを制御できる
  - JOIN句を使わない（FROMとWHEREで表現する）と…
    - ➔ プランをPostgreSQLに任せられる

副問い合わせを速くするには

- FROM句の中かターゲットリスト内に置くようにする
- WHERE句に置くと…
  - 1レコードのアクセスごとに副問い合わせが実行される場合がある



チューニングの例(ステップ1)

- インデックスのない2つのテーブルのJOINの例
  - カラム a:1～1000000
  - カラム b:1～100
  - SELECT * FROM t1, t2 WHERE t1.a = t2.a AND t2.b = 1;
- PostgreSQLが選んだプラン

```
kataoka=# EXPLAIN SELECT * FROM t1, t2 WHERE t1.a = t2.a AND t2.b = 1;
                                QUERY PLAN
---------------------------------------------------------------------------
 Hash Join  (cost=17930.47..58434.34 rows=9787 width=24)
   Hash Cond: ("outer".a = "inner".a)
   ->  Seq Scan on t1  (cost=0.00..15406.00 rows=1000000 width=12)
   ->  Hash  (cost=17906.00..17906.00 rows=9787 width=12)
         ->  Seq Scan on t2  (cost=0.00..17906.00 rows=9787 width=12)
               Filter: (b = 1)
(6 rows)
```

チューニングの例(ステップ2)

- テーブルt2のカラムcにインデックスを作成する
  - CREATE INDEX t2_b_idx ON t2 (b);
- PostgreSQLが選んだプラン

```
kataoka=# EXPLAIN SELECT * FROM t1, t2 WHERE t1.a = t2.a AND t2.b = 1;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Hash Join  (cost=5729.37..46233.24 rows=9787 width=24)
   Hash Cond: ("outer".a = "inner".a)
   ->  Seq Scan on t1  (cost=0.00..15406.00 rows=1000000 width=12)
   ->  Hash  (cost=5704.91..5704.91 rows=9787 width=12)
         ->  Bitmap Heap Scan on t2  (cost=57.25..5704.91 rows=9787 width=12)
               Recheck Cond: (b = 1)
               ->  Bitmap Index Scan on t2_b_idx  (cost=0.00..57.25 rows=9787 width=0)
                     Index Cond: (b = 1)
(8 rows)
```

チューニングの例(ステップ3の失敗例)

- テーブルt2のカラムaにインデックスを作成する
  - CREATE INDEX t2_a_idx ON t2 (a);
- PostgreSQLが選んだプラン

```
kataoka=# EXPLAIN SELECT * FROM t1, t2 WHERE t1.a = t2.a AND t2.b = 1;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Hash Join  (cost=5729.37..46233.24 rows=9787 width=24)
   Hash Cond: ("outer".a = "inner".a)
   ->  Seq Scan on t1  (cost=0.00..15406.00 rows=1000000 width=12)
   ->  Hash  (cost=5704.91..5704.91 rows=9787 width=12)
         ->  Bitmap Heap Scan on t2  (cost=57.25..5704.91 rows=9787 width=12)
               Recheck Cond: (b = 1)
               ->  Bitmap Index Scan on t2_b_idx  (cost=0.00..57.25 rows=9787 width=0)
                     Index Cond: (b = 1)
(8 rows)
```

- 上記はコストを改善できなかった例

チューニングの例(ステップ3)

- テーブルt1のカラムaにインデックスを作成する
  - CREATE INDEX t1_a_idx ON t1 (a);
- PostgreSQLが選んだプラン

```
kataoka=# EXPLAIN SELECT * FROM t1, t2 WHERE t1.a = t2.a AND t2.b = 1;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Merge Join  (cost=6353.62..27590.29 rows=9787 width=24)
   Merge Cond: ("outer".a = "inner".a)
   ->  Index Scan using t1_a_idx on t1  (cost=0.00..18599.00 rows=1000000 width=12)
   ->  Sort  (cost=6353.62..6378.09 rows=9787 width=12)
         Sort Key: t2.a
         ->  Bitmap Heap Scan on t2  (cost=57.25..5704.91 rows=9787 width=12)
               Recheck Cond: (b = 1)
               ->  Bitmap Index Scan on t2_b_idx  (cost=0.00..57.25 rows=9787 width=0)
                     Index Cond: (b = 1)
(9 rows)
```

ご清聴ありがとうございました。