【B2】DatabaseGear テクニカルセッション

EMBARCADERO TECHNOLOGIES®

DEVELOPER CAMP

効率的な「SQL」のすゝめ

日揮情報システム株式会社　システムマネジメント本部
ITマネジメント部 第2チーム　高井 寿宏

アジェンダ

- 会社案内
- 性能問題の発生現場にて
- SQLを生かすためのプログラム設計
- SQLを効率的に動かすには
- Demo
- まとめ

会社概要

- 社名　日揮情報システム株式会社
- 設立　1983年7月1日
- 資本金　4億円（日揮㈱100%出資）
- 売上高　87.8億円（2007年度実績）
- 従業員数　395人（2009年4月現在）
- 本社　横浜市西区みなとみらい3-6-3
- 拠点　上大岡オフィス（横浜市港南区）
  （日揮サポート部門の一部は日揮本社内）
- 業務内容　システムインテグレーション、システム運用、
  パッケージ製品販売
- 関連会社　J-SYS　Philippines（海外開発拠点）
  株式会社コア・システムデザイン（原価管理ツール販売）

EMBARCADERO TECHNOLOGIES®

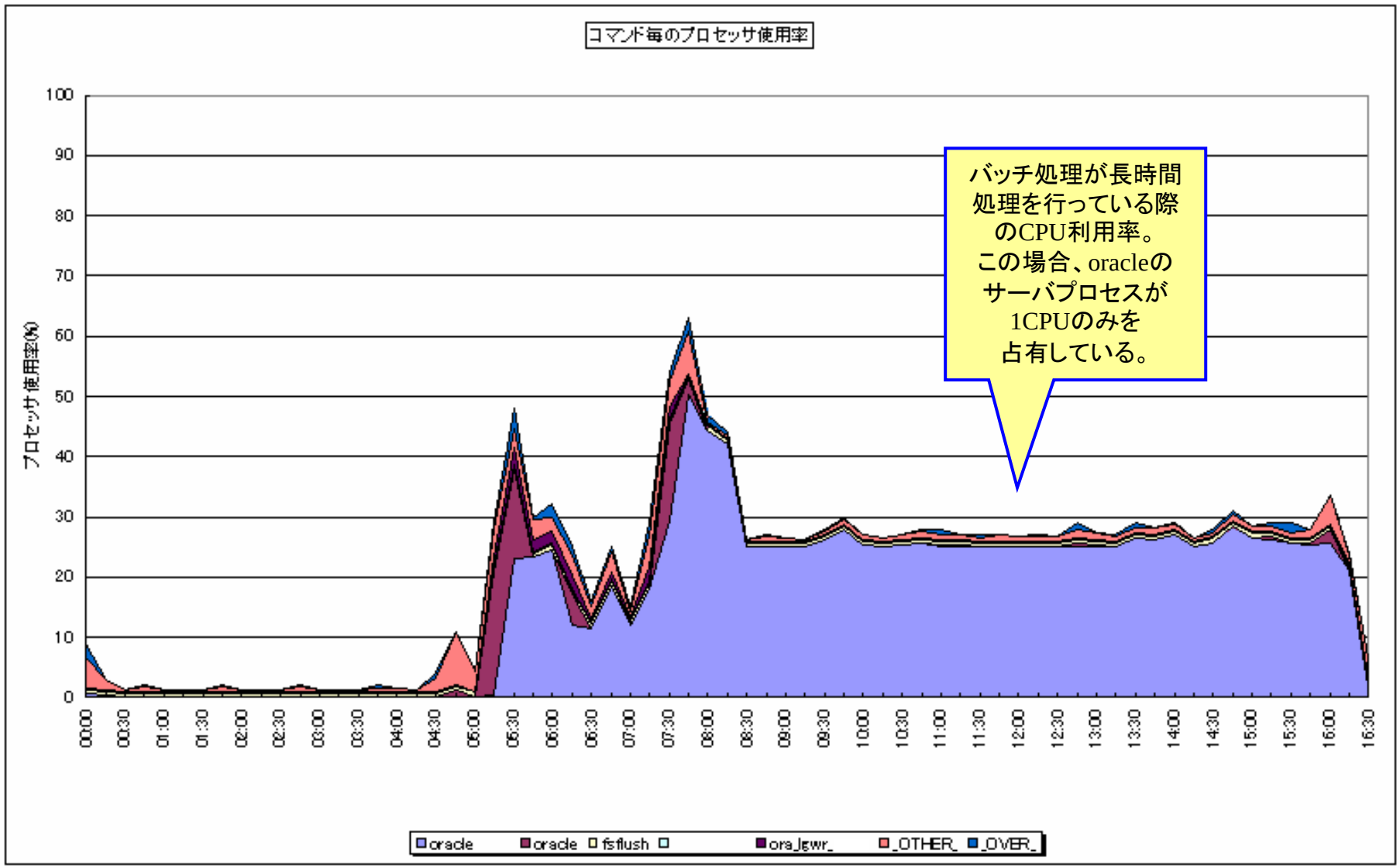
DEVELOPER CAMP

性能問題の発生現場にて

現場で耳にする話①

- CPU 利用率が100%なのでCPUを追加する
  - ▲必ずしも正解とは言えない。
  - 性能要件内であれば、現状は問題ない。
  - OLTP処理のセッション数が増加傾向であるというような場合には、CPUを追加することは、1つの有効な手段である。
  - CPU利用率を監視することには意味があるが、併せてAPのレスポンスタイム、バッチ処理時間等のエンドユーザからの視点で、APの処理時間も履歴的に監視・計測しておくことが重要。多くの性能要件は、エンドユーザからの視点によるものである。
  - CPU利用率が25%でも、性能要件を満たしていなければ、問題となる場合もある。
    - 例えば、4CPUのUNIX マシンで、vmstat コマンドを見ると、OS全体で、100%として表示するため、4CPUのうち、1つのCPUだけが100%利用されても、全体の利用率は、100/4＝25%という表示になる。この値を元に判断すると、思わぬ落とし穴にはまる。
      - 個々のCPU利用率を把握するには、mpstat コマンドで確認しなければならない。

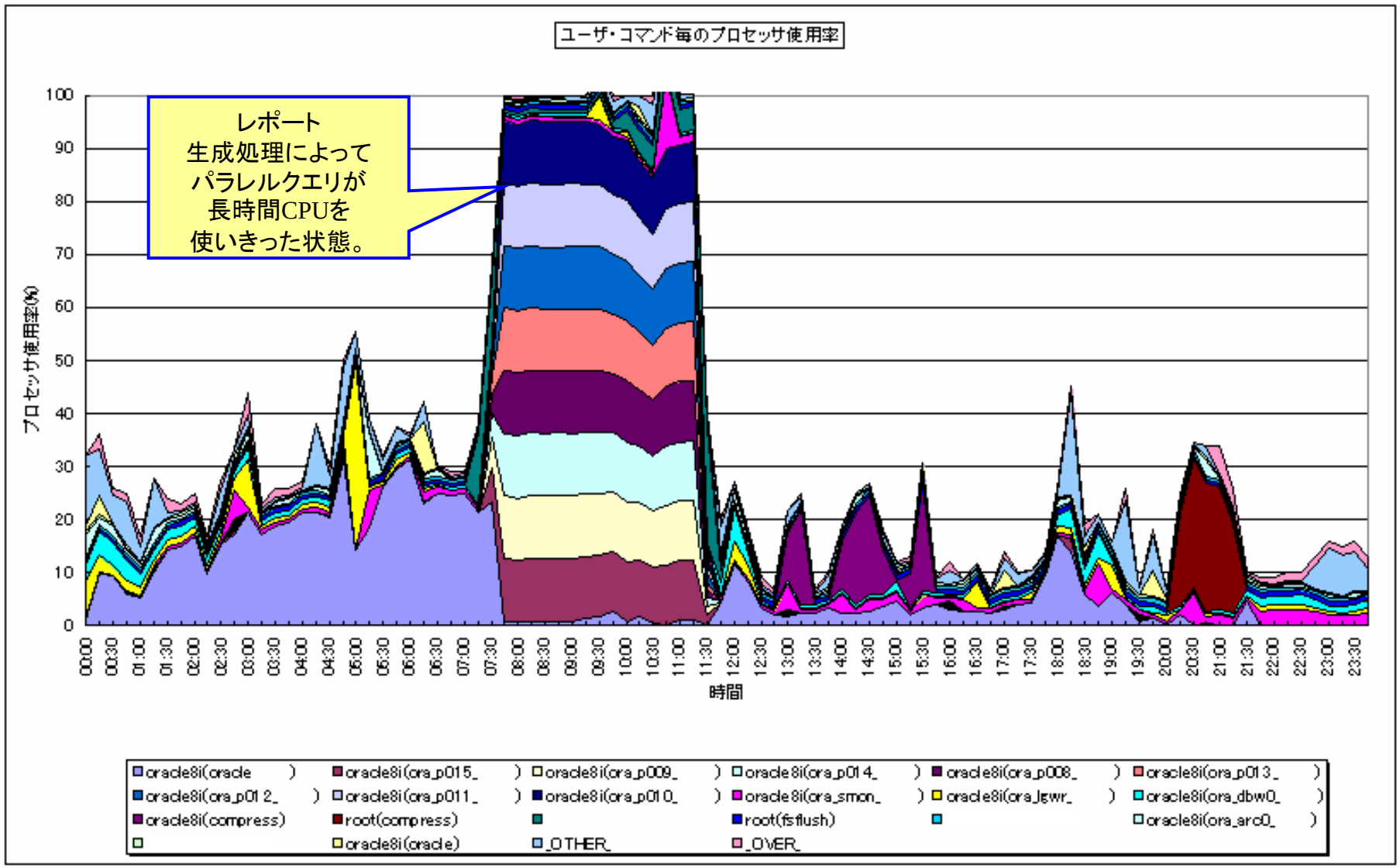
CPU利用率



現場で耳にする話②

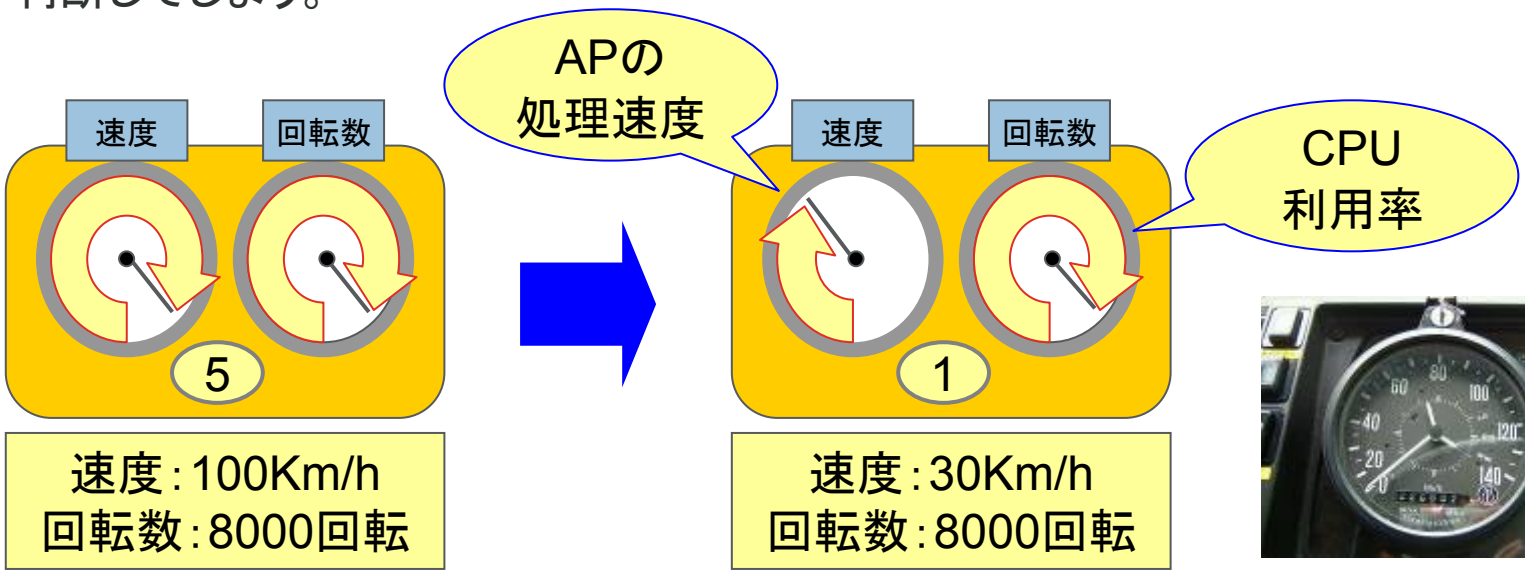
- CPUを追加して、DBにパラレル処理を設定すると速くなる
  - ▲必ずしも正解とは言えない。
  - 単件しか取得しないような処理には、意味がない。同時アクセスが多い処理では、逆にパラレル処理を設定することが大きな問題を発生させる要因にもなりえる。
  - パラレル処理に向く処理、シチュエーションがある。
    - FULL Table Scan が主体で、絞り込める条件がない処理。
    - パーティション化されているテーブルを対象とした処理。
    - etc

パラレル処理のCPU利用率

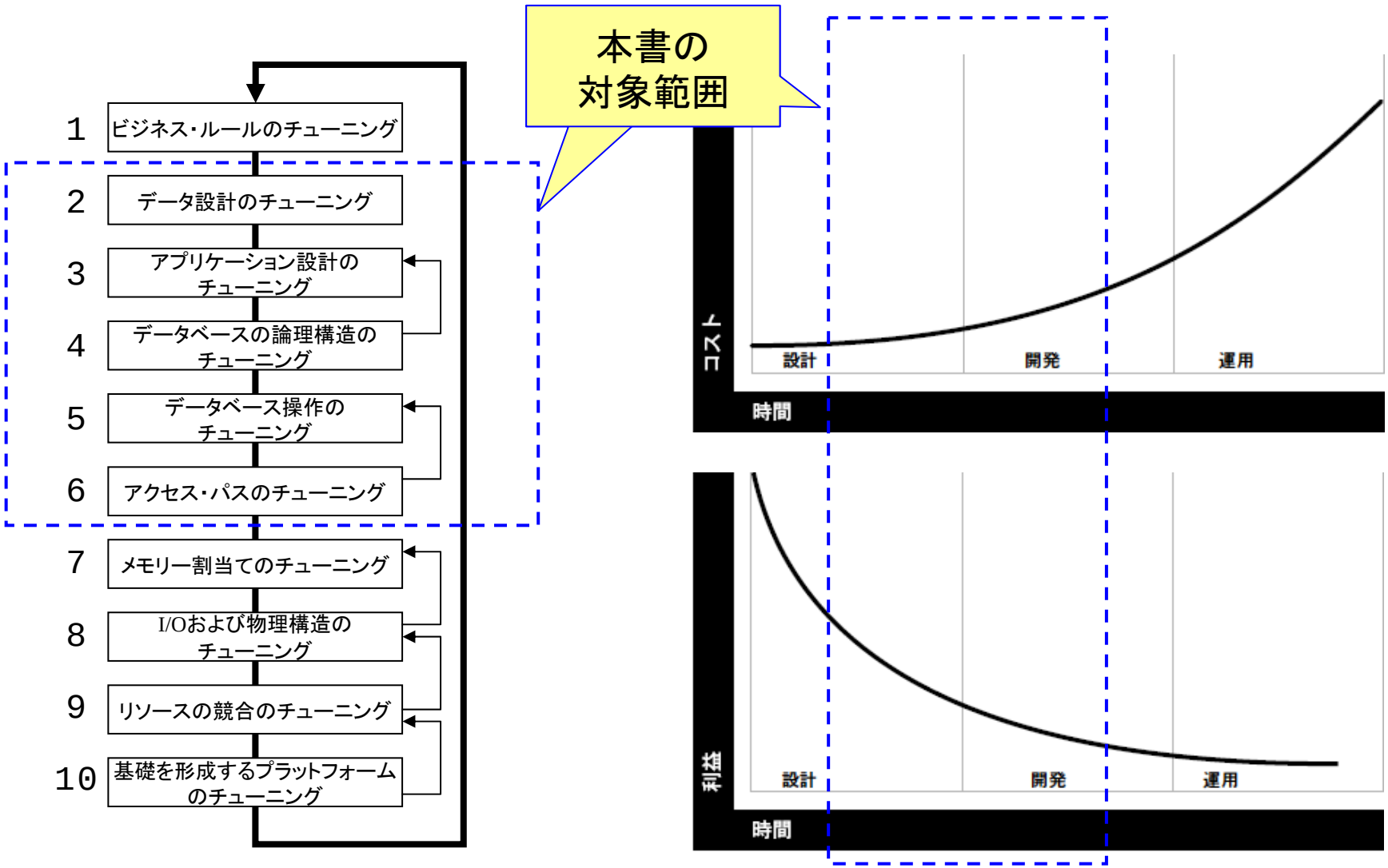


リソース消費量が表すものとは

- ハードウェア(以下H/Wと省略)・リソースの指標が表わす内容は、H/Wだけが原因ではない。
  - 5速のギアで、何とか100Km/h の速度で走ることができる車を運転手が、1速のギアを使って、アクセルを目一杯踏んでも、通常、100Km/hで走ることはできない。それは、運転手の運転の仕方が悪いのである。
    - この場合、車がH/W、運転手がAPと考えると、H/Wには問題がなく、APに問題があるわけだが、CPU利用率だけを見ていると、H/Wは限界にきていると判断してしまう。



チューニングの中での位置づけ



(参考: Oracle8i パフォーマンスのための設計およびチューニング、リリース8.1)

EMBARCADERO TECHNOLOGIES®

DEVELOPER CAMP

SQLを生かすためのプログラム設計

- 手続き型言語とは異なる。
- 必要なデータを要求する命令。データ取得方法を記述しなくても、データを取得することができる。
  - データの取得方法は、基本的にプログラマではなく、RDBMSが決める。
  - データ取得方法とは、どのテーブルをどの順番で参照して、テーブルに付与されているどのインデックスをどういうアクセス方法で参照して･･･という低水準な手順のことである。
    - この手順のことを実行計画、実行プランという。本書では実行計画と記述する。
- 手続き型言語に埋め込めるが、構造化プログラミングに向いているわけではない。

言語の違いを意識したAP(SQL)設計を考えなければならない

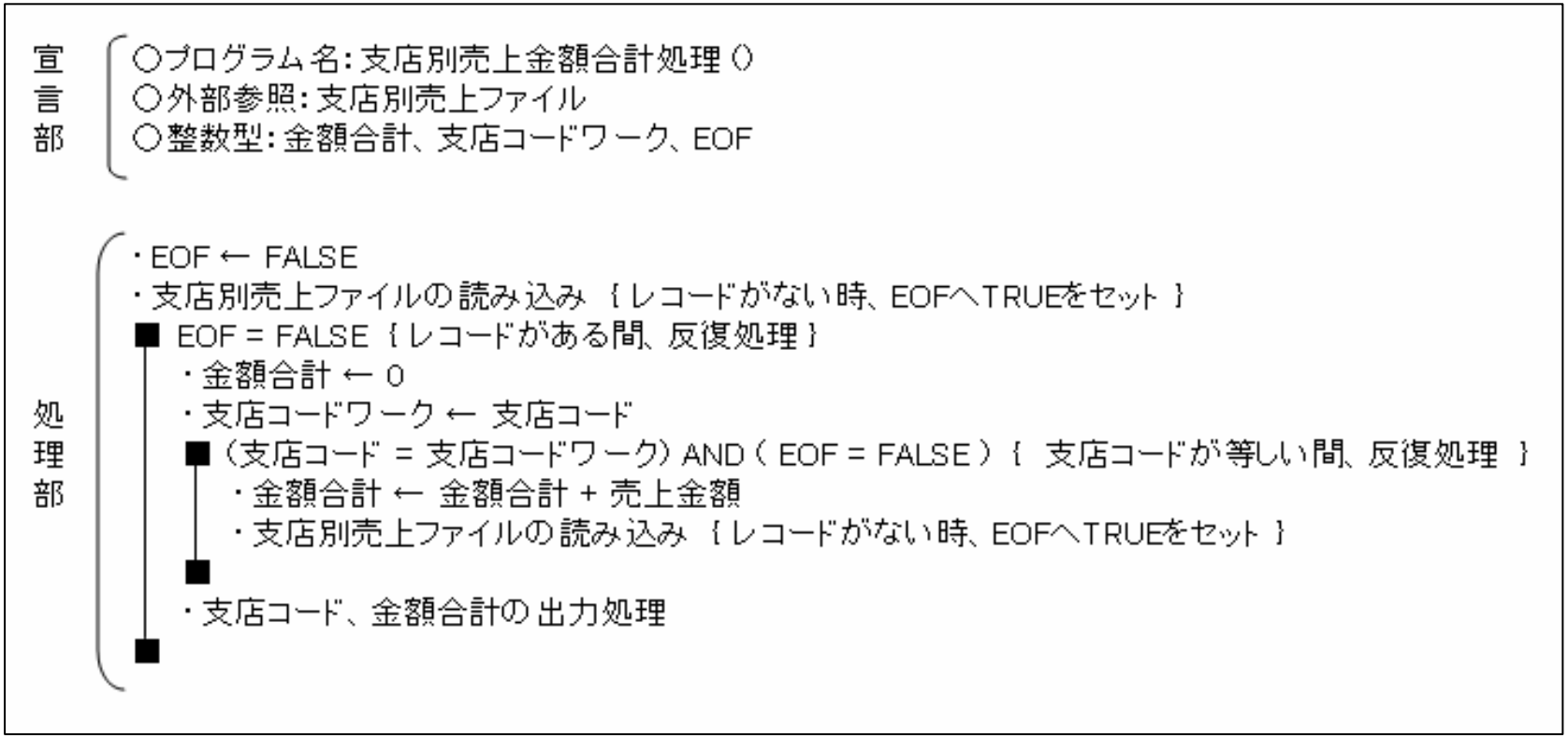
プログラム設計における注意点

- コントロール・ブレイク処理を基にしたプログラム設計はやめる。
  - SQLのGROUP BYを使った集計関数を利用する。
    - マッチング処理、エラー処理は、GROUP BYによる処理を前提に考える。
      - コントロール・ブレイク処理でよく発生するマッチしないデータ、エラーデータを探す処理に各々SQLを使うのは非効率。一致しないデータが必要ならば、差分を求めるようなSQLを書くことによって、一気に求めるとよい。
- SQLを記述するプログラムの構造化、共通化は程々にする。
  - 手続き型言語に埋め込めるが、共通関数のように頻繁な呼出がある関数にSQLを含めない。呼出元毎にJOIN等を行って取得する方法を考える。
  - Viewによる隠蔽化も程々にし、再利用はしない。
- 無駄なSQLの発行を抑制する。
  - 不要なORDER BYは排除する。
    - 特に、コントロール・ブレイク処理では必須の処理で、検索結果が大きいものでは、致命的である。
- 列定義は極力NOT NULL定義にする。

構造化、共通化が
性能を劣悪にする

コントロール・ブレイク処理(例)

- 集計処理を行うための代表的な処理方式



参考:http://www5f.biglobe.ne.jp/~pafu/kihonweb/gozen/01/1_3.htm

コントロールブレイク処理の書換え

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

この例では必要だが、この部分に不要な
ORDER BYが潜んでいる場合が多い。

●コントロール・ブレイク風のプログラム

```
CREATE OR REPLACE PROCEDURE CB IS
 CURSOR c_d IS
  SELECT deptno FROM d ORDER BY deptno;
 CURSOR c_e (d IN number) IS
  SELECT empno,sal FROM e where deptno = d
      ORDER BY empno;
  wk_empno number; v_empno number; v_sal    number;
  cnt number;      snum     number; v_deptno number;
BEGIN
    wk_empno := NULL; cnt := 0; snum := 0;
  OPEN c_d;
  LOOP
    FETCH c_d INTO v_deptno;
    EXIT WHEN c_d%NOTFOUND;
    OPEN c_e (v_deptno);
    LOOP
      FETCH c_e INTO v_empno,v_sal;
      IF c_e%NOTFOUND THEN
                CLOSE c_e;
                EXIT;
      END IF;
      IF cnt = 0 THEN
                wk_empno := v_empno;
      END IF;
    IF wk_empno <> v_empno THEN
      DBMS_OUTPUT.PUT_LINE('deptno='||v_deptno||
      ',empno='||wk_empno||',count='||cnt||',sum='||snum);
                cnt := 0;
                snum := 0;
      END IF;
      cnt := cnt + 1;
      snum := snum + v_sal;
      wk_empno := v_empno;
    END LOOP;
      DBMS_OUTPUT.PUT_LINE('deptno='||v_deptno||',empno='||
      wk_empno||',count='||cnt||',sum='||snum);
    cnt := 0;
    snum := 0;
  END LOOP;
END;
/
```

同一条件では、4分と
1.5秒程度の差がある。

●GROUP BYで一気に取得する場合

```
SQL> select e.deptno,empno,count(e.empno),sum(e.sal) from e,d
      where e.deptno = d.deptno
      group by e.deptno,e.empno order by e.deptno,e.empno;
```

共通関数の乱用

- カーソル内でSQL文を含んだファンクションを発行する場合は、カーソルの元となるSQL文との結合を検討する。

```
OPEN cursor_a;
  LOOP
    FETCH cursor_a INTO now_val_a;
    EXIT WHEN cursor_a%NOTFOUND;

    OPEN cursor_b;
    LOOP
      FETCH cursor_b INTO now_val_b;
      EXIT WHEN cursor_b%NOTFOUND;

      a_counter := a_counter + 1;

      IF get_rate(now_val_b.currency_val,
                  now_val_b.cur_date) = FALSE THEN
```

SQLを含んだストアドファンクションを呼出。カーソルcursor_bでFETCHしたレコード毎に当該ストアドファンクションの呼出が繰り返される。

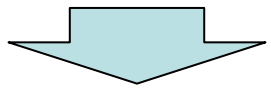
```
FUNCTION   get_rate( ...) RETURN booleanIS
CURSOR local_cursor IS
     SELECT
       ym.st_date
      ,new_ym.ed_date
     FROM
       act_ym       ym
      ,act_ym       rev_ym
     WHERE
        ym.rev BETWEEN ym.st_date
                   AND ym.ed_date
      AND ym.key_id = new_ym.key_id
      AND ym.status = 'ACTIVE'  ;

…
```

エラー処理は別途検討要

呼び出されるファンクションの内容。
当該ストアドファンクション内でSQLを発行しているため、呼び出された回数分、SQLが発行される。

NULLの使い方

- 列定義にNULLを定義するのは極力やめる。
  - NOT NULL定義にし、NULLに意味を持たせないようにする。
  - NULLに意味がある(データがないという意味も含む)と、NULLを使った検索が必要になり、IS NULL、IS NOT NULL検索や、NVL関数等を利用するため、インデックスを有効に利用することができなくなる。
- 対策
  - 列定義にDEFAULT句を指定することによって、値が未設定である場合の値を決めることができる。
  - NULLに該当する値は、DEFAULT句で定義した値(例えば、99999というようなコード値)とし、コード定義書にその意味を記しておく。



Where句に使う列、インデックスに使用する列はDEFAULT句、NOT NULL制約を設定する

NULL値の利用を極力避ける例

EMBARCADERO TECHNOLOGIES.
DEVELOPER CAMP

```
SELECT colA FROM TAB_A
 WHERE
   NVL(colB,'99999') = :parameter ;
```

※colB に NULL を 許可しているため 、:parameter で条件を指定する場合に、NULLの代わり'99999'で代替する例。これによって、colB列にインデックスが存在していても、インデックスが利用できなくなる。また、:parameterにNULLを利用する場合は、colB IS NULL という記述に変えなければならず、Where句の記述も2種類考慮しなければならない。この書き方の場合も、インデックスがあっても、インデックスは使われない。

```
SELECT colA FROM TAB_A
 WHERE
   colB = :parameter;
```

※colB列には、テーブル定義のDEFAULT句で設定した値が、レコード生成時に設定される。その上で、colB列にインデックスが存在していた場合、インデックスが利用でき、NULLを考慮した記述も不要になる。

ファンクション・インデックスという機能もあるが、IS NULL検索では利用されず、更新コストも大きい。

EMBARCADERO
TECHNOLOGIES®

DEVELOPER CAMP

SQLを効率的に動かすには

Where句を意識する

- インデックスを意識する。
  - Where句に出てくる列は、インデックスの作成対象
    - 特に、5～10%程度以下に絞り込まれる条件には有効。
    - 対象テーブルすべてのレコード件数が少なければ、インデックスを利用しないほうが速い場合もある。
- パーティションを意識する。
  - Where句に出てくる列で、テーブル全体をある程度、均等に分割できる条件は、パーティション化が有効。
    - ヒストリカルなデータの年月。
    - 分類コードのような、ある程度、複数にわかれるコード。
    - 多くのSQLのWhere句で利用される条件。
    - 検索結果が多いため、インデックスを使って絞り込むほどではない(絞り込めない)。少数データまで絞り込める条件は、インデックスが利用できる。

SQL発行数を削減する

- SQL文の発行数を減らす方向で設計し、1回のSQL発行で、できるだけ複数のレコードを取得するように意識して記述する。
  - SQLは、1回の発行で複数行となる結果集合が得られる性質を最大限利用する。
    - カーソルでのSELECT⇒INSERTではなく、INSERT...SELECTの利用推進
    - カーソルでのSELECT⇒DELETEではなく、DELETE...WHEREの利用推進
    - カーソルでのSELECT⇒UPDATEではなく、UPDATE...WHEREの利用推進
- 手続き型言語と併用する場合、手続き型言語とSQLとの切替を少なくすることを基本に考える。バッチ処理の場合に特に気をつける。
  - ループ構造をJOIN処理に変えて取得できる場合が多々ある。JOINは遅いと神話のように語り継がれているが、劇的に改善できる場合もある。

プログラム設計前から考慮し、方式設計等のプログラム・アーキテクチャを決定する段階で、上記方式を使った際のエラー処理等の共通の枠組みを決めておき、柔軟に対処できるようにしておく。

集合処理の推進<INSERT>例①

- 性能が出ないコーディング例

```
CREATE OR REPLACE PROCEDURE ins IS
CURSOR cc IS SELECT EMPNO,ENAME FROM E;
BEGIN
  FOR rc IN cc LOOP
    INSERT INTO A (A,B) VALUES (rc.EMPNO,rc.ENAME);
  END LOOP;
END;
/
```

●100万件をINSERTした場合の実行例

```
SQL> exec ins;

PL/SQLプロシージャが正常に完了しました。

経過: 00:01:00.90
```

- 性能を意識したコーディング例①

```
CREATE OR REPLACE PROCEDURE ins_only IS
BEGIN
INSERT INTO A (A,B)
       SELECT EMPNO,ENAME FROM E;
END;
/
```

●100万件をINSERTした場合の実行例

```
SQL> exec ins_only;

PL/SQLプロシージャが正常に完了しました。

経過: 00:00:02.11
```

この例では、100万件で、
処理時間が約1/30に短縮した

レコード毎のエラー処理が
必要な場合は利用不可

集合処理の推進<INSERT>例②

- 性能を意識したコーディング例②

```
CREATE OR REPLACE PROCEDURE ins IS
TYPE e_type_a is table of A.A%TYPE index by binary_integer;
e_tbl_a e_type_a;
TYPE e_type_b is table of A.B%TYPE index by binary_integer;
e_tbl_b e_type_b;
CURSOR cc IS SELECT EMPNO,ENAME FROM E;
BEGIN
open cc;
fetch cc bulk collect into e_tbl_a,e_tbl_b;
close cc;
FORALL i IN e_tbl_a.FIRST..e_tbl_a.LAST
  INSERT INTO A (A,B) VALUES
      (MOD(e_tbl_a(i),10),e_tbl_b(i));
END;
/
```

●100万件をINSERTした場合の実行例

```
SQL> exec ins;

PL/SQLプロシージャが正常に完了しました。

経過: 00:00:06.70
```

- 性能面では、「性能を意識したコーディング例①」を推奨する。可能な限り、この形で処理できるように設計するとよい。
- ただし、INSERT対象となるレコードを1行づつエラー処理が必要な場合は、「性能を意識したコーディング例②」を利用し、エラー処理を組み込むことも可能。

条件分岐のためにカーソルを利用することを回避

- 条件分岐
  - 条件分岐をして値を設定するためにSQL文を複数利用する場合もあり、その部分を DECODE 、CASE等のSQL関数に書換えし、単一SQL文内で判定するように変更する。

```
CREATE OR REPLACE PROCEDURE ins IS
VAL NUMBER;
CURSOR cc IS SELECT EMPNO,ENAME FROM E;
BEGIN
      FOR rc IN cc LOOP
         IF rc.EMPNO = 5000 THEN
                  VAL := 1;
         ELSE
                  VAL := 2;
         END IF;

         INSERT INTO A (A,B) VALUES (VAL,rc.ENAME);
      END LOOP;
END;
/
```

取得元の値から条件判定して値をセットするためだけにカーソルを使ってループを回し、1レコードづつ処理している。

```
CREATE OR REPLACE PROCEDURE ins IS
BEGIN
      INSERT INTO A (A,B) SELECT DECODE(EMPNO,5000,1,2) ,ENAME FROM E;
END;
/
```

INSERT~SELECT処理になり速い。
DECODE/CASE関数の詳細な利用方法はRDBMSのマニュアル等を参照の事

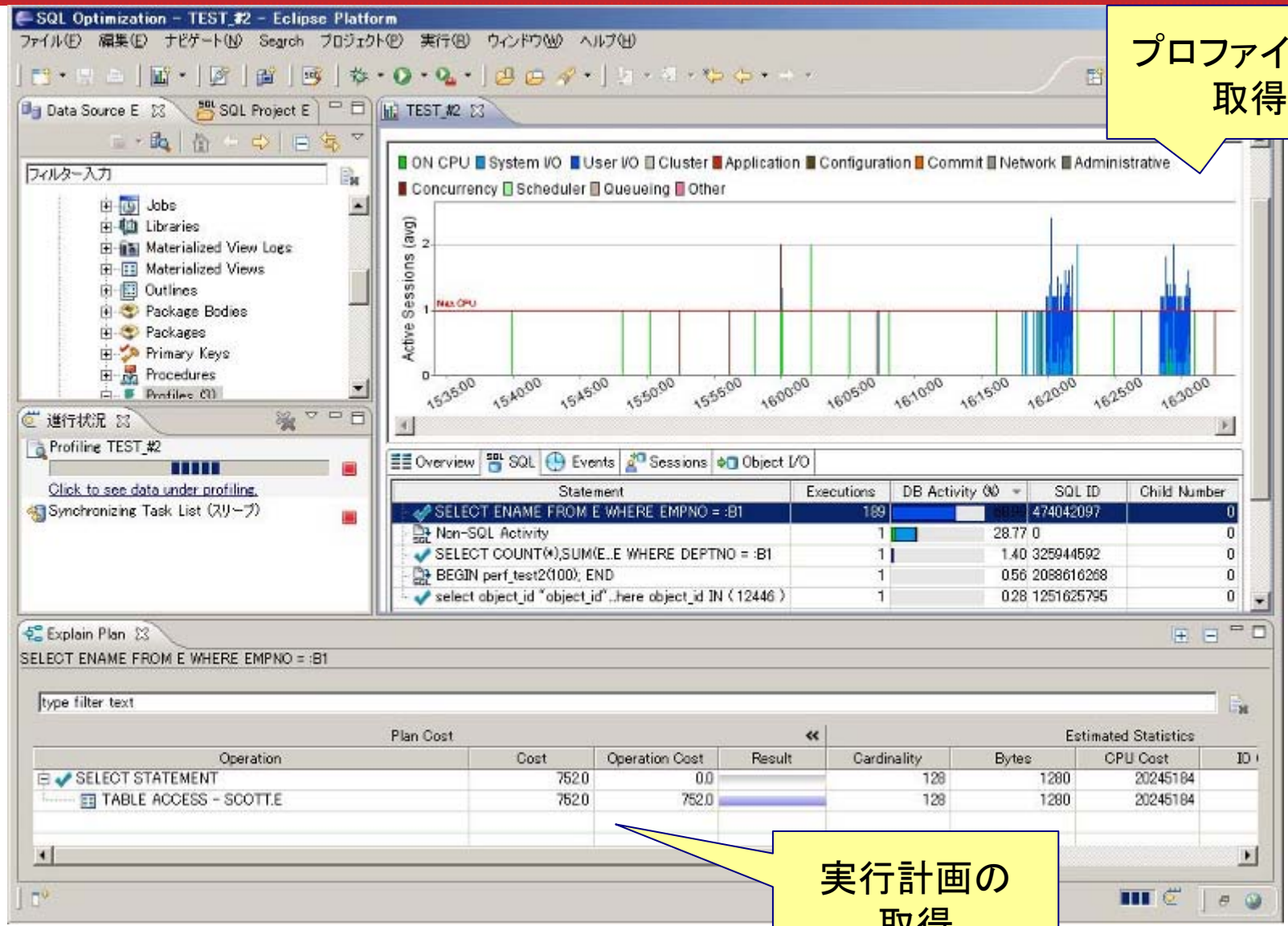
EMBARCADERO
TECHNOLOGIES®

DEVELOPER CAMP

Demo

DB Optimizer ～ プロファイリング

EMBARCADERO TECHNOLOGIES.
DEVELOPER CAMP

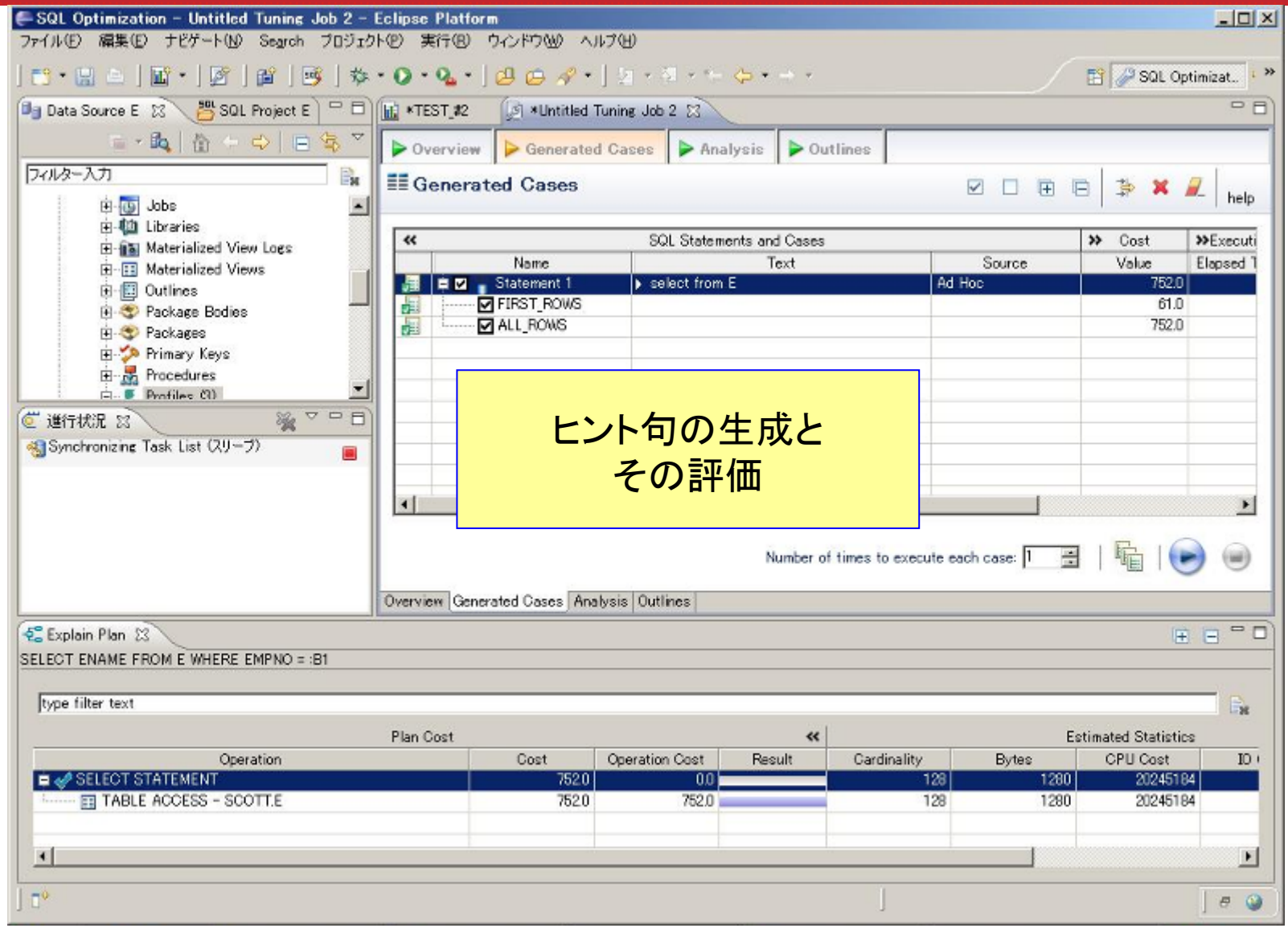


プロファイルの取得

実行計画の取得

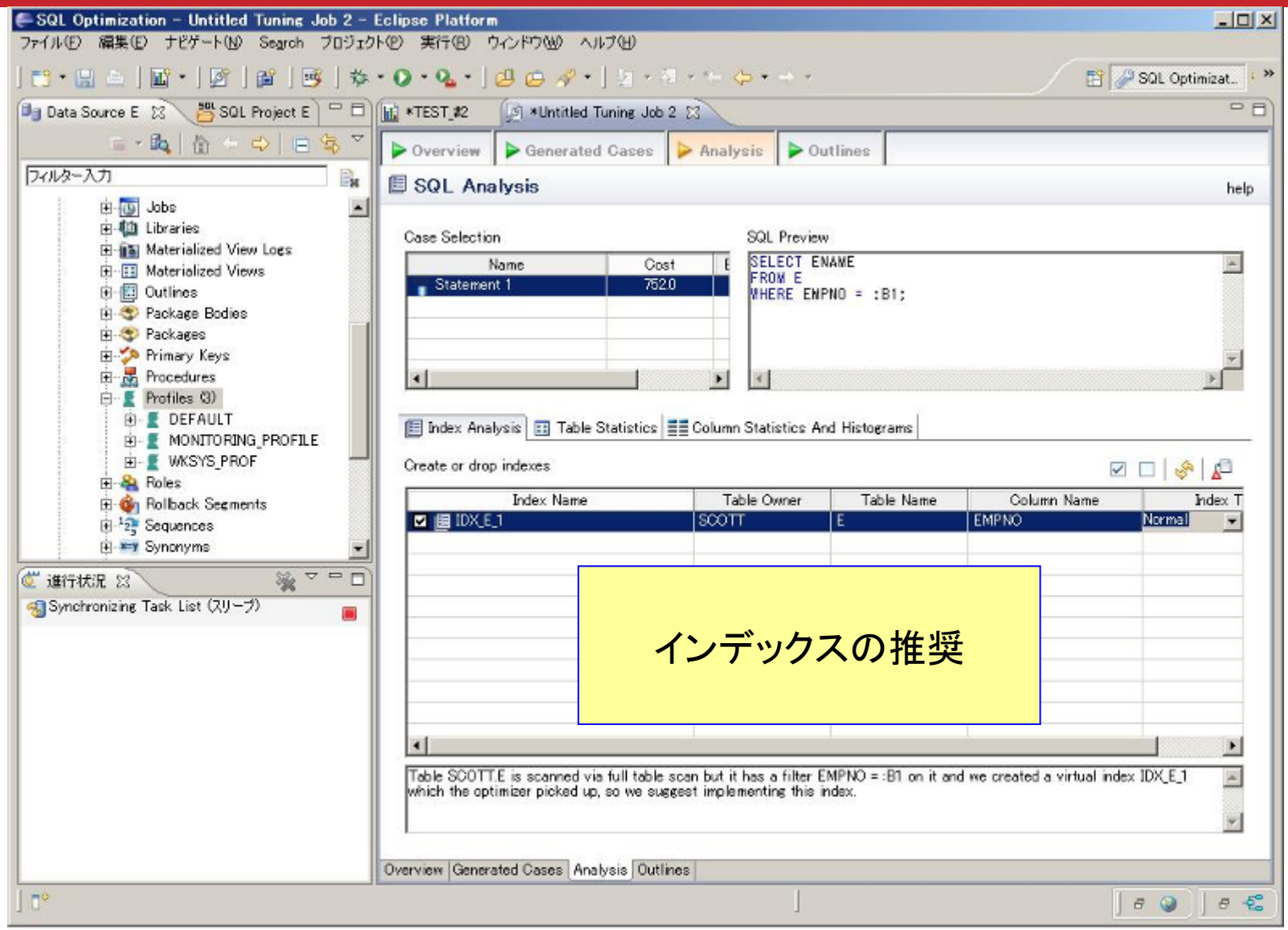
DB Optimizer ～ チューニング ～ ヒント句評価

EMBARCADERO TECHNOLOGIES.
DEVELOPER CAMP



DB Optimizer ～ チューニング ～ 推奨インデックス生成

EMBARCADERO TECHNOLOGIES.
DEVELOPER CAMP



EMBARCADERO TECHNOLOGIES®

DEVELOPER CAMP

まとめ

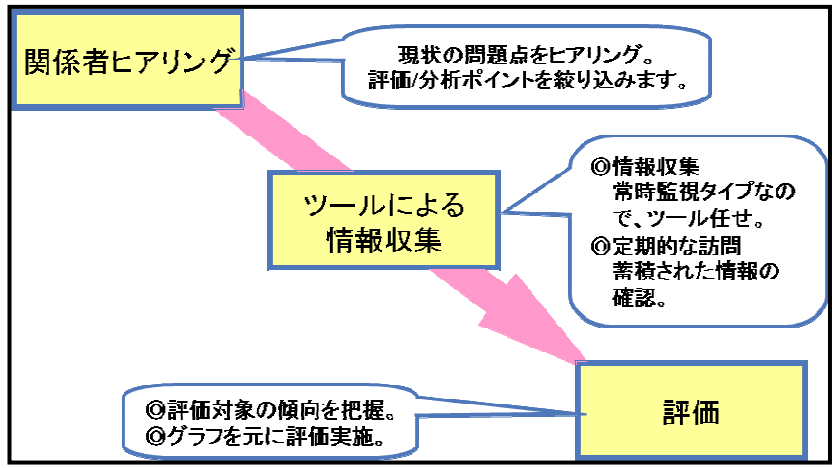
まとめ

- SQLは手続き型言語ではない。
- SQLを含んだロジックの共通化、構造化、隠蔽化は、性能向上の足かせになる場合が多いので設計段階から注意する。
- SQLの発行数は少なく、データ集合を一気に取得するように設計する。
- NULLは意識して、できる限り利用しないようにする。
- インデックス、パーティションを有効に利用する。
- 結果を監視し、様々な観点から評価する。

上記を遵守しても、性能が悪い場合もあり得るが、
後のチューニングで改善できる可能性が広がる。

DB性能診断サービス

- 概要
  - 対象システムのボトルネック発見及び改善提案。
- 対象範囲
  - 弊社指定ツールから収集されたデータを基に、ボトルネックの指摘及び対策案を提示します。
  - 評価対象期間は、別途ヒアリングの中で定める事とします。
- 対象DB
  - Oracle
  - IBM DB2 UDB
  - SQL Server
    - 対象バージョン等については別途お問合せください。
- 成果物
  - 報告書 (対象RDBMSの評価及び改善提案書)
  - 報告会 (1回(複数回実施の場合は、別途御見積))
- 期間パターン
  ① 一定期間集中型
  ② 定期訪問型
  ③ スポットコンサル型



- 注意点
  - 改善対策案に基づいたお客様プログラムの改造、DBサーバの設定変更等の作業は行いません。
  - 改善対策案に関しては、その効果を保証するものではありません。

- DBセキュリティ＆監査「Chakra」
  - 監視対象システムへ負荷をかけない
  - DBへのアクセスログを取得
  - 疑わしいアクセスを検知し通知

  個人情報保護・内部統制対策として、多くのユーザ様で使用されている製品です。

- 定例開催セミナー
  - 日揮情報システムが主催している定期開催セミナーです。

    Chakraの機能ご紹介はもちろんのこと、これまでの導入実績をもとにした事例のご紹介、関連サービスのご紹介、日揮情報システムオリジナルのユーティリティのご紹介など様々な有意義な情報をご担当者様 にご提供させていただいております。

    (各セミナーの詳細については、http://www.jsys-products.com/info/seminar/をご参照ください)

DBセキュリティ関連サービス

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 高インパクトSQL分析サービス
  - お客様環境にChakraを設置し、Chakraでログを収集させていただきます。
    収集したログからDBに対してインパクトがあるSQL文(ALTER文、DELETE文など)を抽出し、レポートを作成いたします。本サービスをご利用いただくことにより、DBの視点からも「可視化」することが可能となります。また、データ挿入件数等を把握し肥大化の予兆を察知することも可能です。
- レポート例
  - インパクトSQL 文の一覧表
  - Insert文の発行件数
  - 情報漏洩チェック一覧表
- サービスの流れ

価格

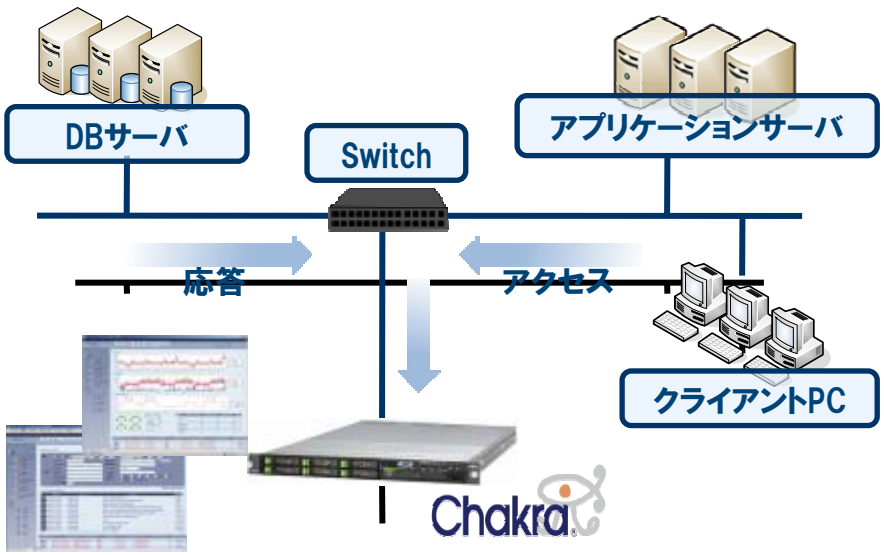
¥150,000

※税別、1データベースあたり

DBセキュリティ・スタートアップパッケージ

データベース セキュリティ＆監査 ソリューション「Chakra」を1パッケージ化!!
ネットワークに接続するだけで、アクセスログを取得することができます。

- こんな方に最適です
  - 簡単にDBアクセスログをとりたい
  - DBに負荷をかけたくない
  - 導入に手間をかけたくない
  - スモールスタートしたい



スペック表	
製品外観	
TYPE	1U
CPU	Intel Xeon(R) E5502 (1.86GHz/2コア/4MB)
メモリ	4GB
HDD	441GB(147GB x 3(RAID 5))
OS	Windows 2003 Server R2 Standerd Edition
監視対象DB数	1
監視対象サーバのCPU数	1～
パッケージ内容	ハードウェア、Chakraソフトウェア 導入設定サービス ほか
標準価格	~~¥3,800,000～~~

お問合せ

- URL：
  - Homepage　　：http://www.jsys.co.jp/
    - 弊社案内及び取扱製品のご案内
- メールアドレス
  - request@jsys-products.com
- 電話
  - (045)715-8707