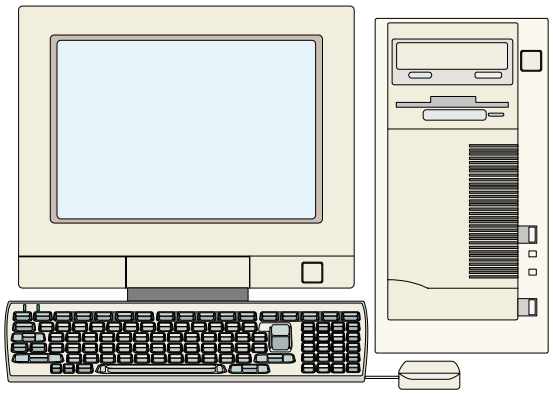
第３章　【機能検証に於けるVacuumの効果】

当章では、検証に使用したシステムの構成やソフトウエアについて述べ、具体的な検証方法について述べたあと、検証結果の提示とそれに対する解説を論ずる。

3−1　システム構成とインストール手順

1）　システム構成

① 検証用マシン



PC　：　ＤＥＬＬDimension 2100
ＣＰＵ：Celeron 1GHz
メモリ　：２５６MByte
HDD：３０ＧＢｙｔｅ

図 3-1　マシンイメージ

② ＯＳ

RedHat Linux 7.1

Linux は、事実上フリーの OS であり、低価格で高性能なサーバの構築が可能になるため、現在では幅広い分野で使用されている。また、Linux と PostgreSQL でシステムを構築した事例も多く、比較的低スペックのマシンでも安定して動作することから、今回の検証システムの OS として採用した。

③ ＤＢ

PostgreSQL 7.2.1

当時の最新版をインストールした（2002/12/25 現在の最新版は 7.3）。

Oracle 9.0.1 (試用版)

商用データベースのなかでも、最も普及率の高いＯｒａｃｌｅの試用版を検証の比較用データベースとして使用した。

2）　Redhat Linux 7.1 のインストール

Linux のインストールは GUI のインストールプログラムに従って、以下の設定で行った。

表 3-1　ＯＳインストール時の設定

番号	項目	設定値
1	言語の選択	Japanese
2	キーボードの設定 モデル レイアウト デッドキー	 Japanese 106 key Japanese 有効にする
3	マウスの設定	2 Button Mouse (P/S2)
4	インストールの種類の選択	サーバシステム
5	パーティション設定	パーティションの自動設定
6	タイムゾーンの選択	アジア/東京
7	言語サポートの選択 デフォルトの言語 インストールする言語	 Japanese 選択なし
8	アカウントの設定 root パスワード アカウント名 パスワード	 oisaoisa oisa oisaoisa
	パッケージグループの選択	デフォルト（未指定）

3）　PostgreSQL 7.2 のインストール

PostgreSQL のインストール手順を以下に示す。

① ユーザ名「postgres」でアカウントを作成

```
# adduser postgres
```

② インストールするディレクトリを作成。所有者権限の変更を行う

```
# mkdir /usr/local/
# chown postgres /usr/local/
# mkdir /usr/local/pgsql
# chown postgres /usr/local/pgsql
```

③ postgres ユーザに変更

```
# su - postgres
```

④ インストールの実行

```
$ cd /usr/local/src
$ tar xfz postgresql-7.2.tar.gz
$ cd postgresql-7.2
$ make
$ make check          # インストールテスト
$ make install        # インストール実行
```

⑤ 環境設定

~/.tcshrc に以下の記述を追加

```
setenv POSTGRES_HOME "/usr/local/pgsql"
setenv PGDATA "$POSTGRES_HOME/data"
setenv PGLIB "$POSTGRES_HOME/lib"
setenv PATH "$POSTGRES_HOME/bin:$PATH"
```

次に、データベースのセットアップ手順を以下に示す。

⑥ データベースクラスタの作成

```
$ initdb
```

⑦ デーモン(postmaster)の起動

```
$ pg_ctl start
```

⑧ データベースの作成

```
$ createdb oisadb
CREATE DATABASE
```

⑨ データベースの使用

```
$ psql oisadb
Welcome to psql, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

oisadb=#
```

※これ以降、SQL 文による問い合わせでデータベースを操作できる。

4）　Oracle9.0.1 のインストール

インストール時に利用する各種設定値を以下のようにしておく。

Oracle インストール用アカウントoracle（デフォルト）

Oracle インストール用グループoinstall（デフォルト）

DB 管理者用グループdba（デフォルト）
SID orcl（デフォルト）
GlobalDatabaseName orcl.oracle.co.jp

Root ユーザでログインし、自動インストーラを起動する。
これ以降のインストール手順については、リリースノートに従って行った。
参考URL：
http://otndnld.oracle.co.jp/tech/linux_win/pdf/0712_oracle9idatabaseforlinuxstartguide.pdf

※RedHat7.1にOracle9.0.1をインストールする場合の注意
RedHat7.1へOracle9.0.1をインストールする場合、ldコマンドのバージョンが古いため、インストール中のクライアント・ライブラリ作成時にエラーが発生することがわかっているため、以下の手順でエラーを回避する必要がある。

① oracle ユーザーで接続する。
② 環境変数 ORA_HOME を設定する。
③ $ORA_HOME/bin にある genclntsh スクリプトの 147 行目の
　LD_SELF_CONTAINED=”-z defs”
　の-z defsの記述を削除する。
④ 修正した genclntsh スクリプトを実行する。
⑤ エラーウインドウに戻り、「再試行」ボタンをクリックしてインストールを続ける。

3−2　検証方法

1）　方針

（１） postgreSQL の VACUUM 機能を検証することを目的とする。
（２） VACUUM 機能の特性を鑑み、応答速度・データ格納領域のリソースを検証する。
（３） 他の商用 DB でも同様の検証を実施し、相対的な検証結果を得る。
※相対比較を行う商用 DB には ORACLE（Oracle9i 試用版）を選定する。
※Oracle は、試用版なので結果は公表しない。

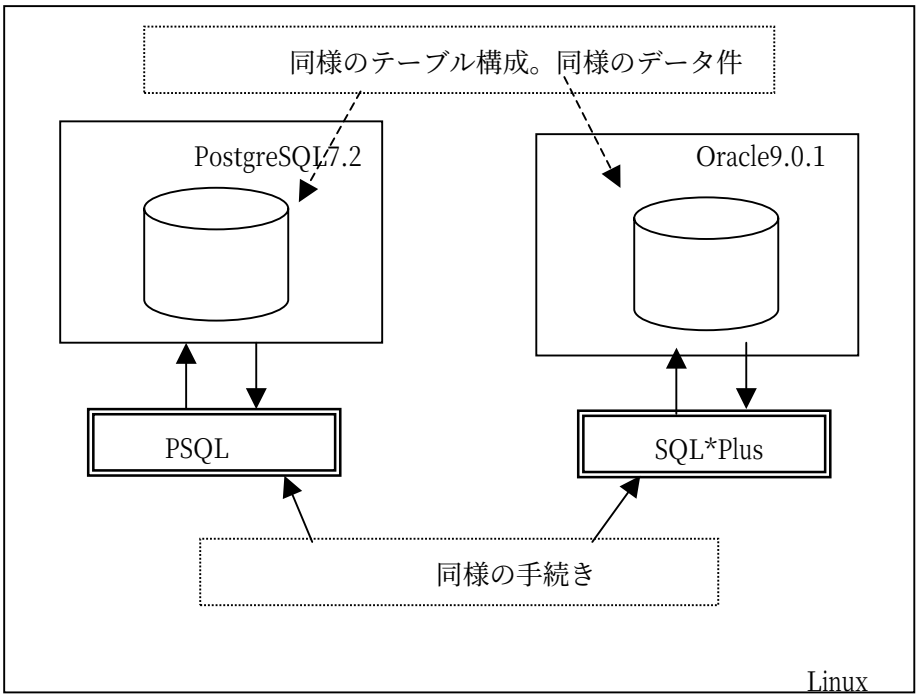


図 3-2　検証の概要図

2）　準備

（１） データ

郵政省から無償配布されている郵便番号データの全国版を使用する。
（関連サイト：http://www.post.yusei.go.jp/newnumber/down_2.htm）
このデータは約１２万件のレコード数を保有し CSV 形式で配布されている。

表 3-2　郵便番号データ CSV ファイルの構成

No	項目	形式	備考
1	全国地方公共団体コード	半角数字	
2	(旧)郵便番号(5 桁)	半角数字	
3	郵便番号(7 桁)	半角数字	
4	都道府県名	半角カタカナ	
5	市区町村名	半角カタカナ	

6	町域名	半角カタカナ	
7	都道府県名	漢字	
8	市区町村名	漢字	
9	町域名漢字	漢字	
10	一町域が二以上の郵便番号で表される場合の表示	半角数字	1:該当 0:該当せず
11	小字毎に番地が起番されている町域の表示	半角数字	1:該当 0:該当せず
12	丁目を有する町域の場合の表示	半角数字	1:該当 0:該当せず
13	一つの郵便番号で二以上の町域を表す場合の表示	半角数字	1:該当 0:該当せず
14	更新の表示	半角数字	0:変更なし 1:変更あり 2:廃止
15	変更理由	半角数字	0:変更なし 1:市政･区政･町政･分区･政令指定都市施行 2:住居表示の実施 3:区画整理 4:郵便区調整、集配局新設 5:訂正 6:廃止

（２） テーブル

郵便データの CSV 形式から、PostgreSQL・ORACLE それぞれのテーブル構成を作成する。

① postgreSQL 版　郵便便番号データ（テーブル名：postlist）

表 3-3　テーブルレイアウト（postgreSQL 版）

No	項目名	属性	キー	備考
1	JISCODE	text		
2	OLD_ID	varchar(5)		
3	POSTID	char(7)		
4	KEN_K	text		
5	SI_K	text		
6	TYOU_K	text		
7	KEN	text		
8	SI	text		
9	TYOU	text		
10	SOMEZIPS	Int4		
11	AZA	Int4		
12	CHOUME	Int4		
13	SOMETOWNS	Int4		
14	CHANGED	Int4		
15	CHANGEDWHY	Int4		

- テーブル生成スクリプト

```
create table postlist
        (
                jiscode         text,
                old_id          varchar(5),
                post_id         char(7),
                ken_k           text,
                si_k            text,
                tyou_k          text,
                ken             text,
                si              text,
                tyou            text,
                somezips        int4,
                aza             int4,
                choume          int4,
                sometowns       int4,
                changed         int4,
                changed_why     int4
        )
```

（３）　ORACLE 版　郵便便番号データ（テーブル名：postlist）

表 3-4　テーブルレイアウト（oracle 版）

No	項目名	属性	キー	備考
1	JISCODE	varchar2(5)		
2	OLD_ID	varchar2(5)		
3	POSTID	varchar2(7)		
4	KEN_K	varchar2(16)		
5	SI_K	varchar2(64)		
6	TYOU_K	varchar2(192)		
7	KEN	varchar2(8)		
8	SI	varchar2(32)		
9	TYOU	varchar2(96)		
10	SOMEZIPS	varchar2(1)		
11	AZA	varchar2(1)		
12	CHOUME	varchar2(1)		
13	SOMETOWNS	varchar2(1)		
14	CHANGED	varchar2(1)		
15	CHANGEDWHY	varchar2(1)		

- テーブル生成スクリプト

```
create table postlist
        (
```

```
    jiscode          varchar2(5),
    old_id           varchar2(5),
    post_id          varchar2(7),
    ken_k            varchar2(16),

    si_k             varchar2(64),
    tyou_k           varchar2(192),
    ken              varchar2(8),
    si               varchar2(32),
    tyou             varchar2(96),
    somezips         varchar2(1),
    aza              varchar2(1),
    choume           varchar2(1),
    sometowns        varchar2(1),
    changed          varchar2(1),
    changed_why      varchar2(1)
)
```

（４） 検証用スクリプト

① インデックスの作成

検証には、大きくインデックス有り検索とインデックス無し検索の 2 種類に分けて行う。
POST_ID をキーとしてインデックスを作成する。

- PostgreSQL 版　インデックス生成スクリプト

```
create index postlist_index on postlist (post_id)
```

- ORACLE 版　インデックス生成スクリプト

```
create index postlist_index on postlist (post_id)
```

② 現在時刻の取得

検索の応答速度の測定には、検索の直前と直後にシステム時刻を取得し、その差異を測定値として用いる。

- PostgreSQL 版　現在時刻取得スクリプト

```
select ‘now’::timestamp
```

- ORACLE 版　現在時刻取得スクリプト

```
time sqlplus ログイン名/パスワード @検索スクリプト>null
```

（５） データ検索

データの検索は、「完全一致検索」「前方一致検索」「副問合せ」の 3 種類を行う。

① 完全一致検索

- PostgreSQL 版　完全一致検索スクリプト

```
SELECT * FROM POSTLIST WHERE POST_ID = '0600000'
```

- ORACLE 版　完全一致検索スクリプト

```
SELECT * FROM POSTLIST WHERE POST_ID = '0600000'
```

② 前方一致検索

- PostgreSQL 版　前方一致検索スクリプト

```
SELECT * FROM POSTLIST WHERE POST_ID LIKE '060%'
```

- ORACLE 版　前方一致検索スクリプト

```
SELECT * FROM POSTLIST WHERE POST_ID LIKE '060%'
```

③ 副問合せ

- PostgreSQL 版　副問合せスクリプト

```
SELECT * FROM POSTLIST WHERE POST_ID IN
    (SELECT POST_ID FROM POSTLIST WHERE POST_ID = '0600000')
```

- ORACLE 版　副問合せスクリプト

```
SELECT * FROM POSTLIST WHERE POST_ID IN
    (SELECT POST_ID FROM POSTLIST WHERE POST_ID = '0600000')
```

（６） VACUUM

VACUUM にはいくつかパラメータが用意されているが、ここでは完全な VACUUM を実施するため「FULL」パラメータを使用する。

- VACUUM 実行スクリプト（PostgreSQL のみ）

```
VACUUM FULL
```

３）　具体的方法

（１）　検証ポイント

データ件数による違いも検証するため、以下のレコード件数を検証ポイントとする。

- 30,000 件
- 60,000 件
- 90,000 件
- 121,208 件　（全件）

（２）　検証マトリクス

様々な条件を組み合わせて行うため、下記のような検証マトリクスを作成する。

検証の組合せは、Ｉｎｄｅｘ・ＶＡＣＣＵＭ・データの虫食いの有無により、全件数（データ数４パターン）の合計３２パターンである。

各検証パターンごとに、３つの計測値とＤＢファイルの測定を行う。

表 3-5　検証マトリクス

Index 有無	VACUUM 有無	データの虫食い有無	
		無	有
有	有	ア	ア
		イ	イ
		ウ	ウ
		エ	エ
	無	ア	ア
		イ	イ
		ウ	ウ
		エ	エ
無	有	ア	ア
		イ	イ
		ウ	ウ
		エ	エ
	無	ア	ア
		イ	イ
		ウ	ウ
		エ	エ

① Index 有無

検証対象となるテーブルにインデックスを設定したかどうかの有無

② VACUUM 有無

検証中に VACUUM を実行したかどうかの有無

③ データ虫食いの有無

ＤＢファイル内のデータ格納状態に虫食いがあるかどうかの有無

※虫食いには、データの挿入･削除を繰り返し意図的に状態を作り出す。

④ 計測値

ア）　完全一致検索の計測値

イ）　前方一致検索の計測値

ウ）　副問合せ検索の計測値

エ）　ＤＢファイルのサイズ

４） 手順

（１）検証フロー

各検証ポイントにおいて、検証マトリクスに従い検証を進める。以下に検証フローを示す。

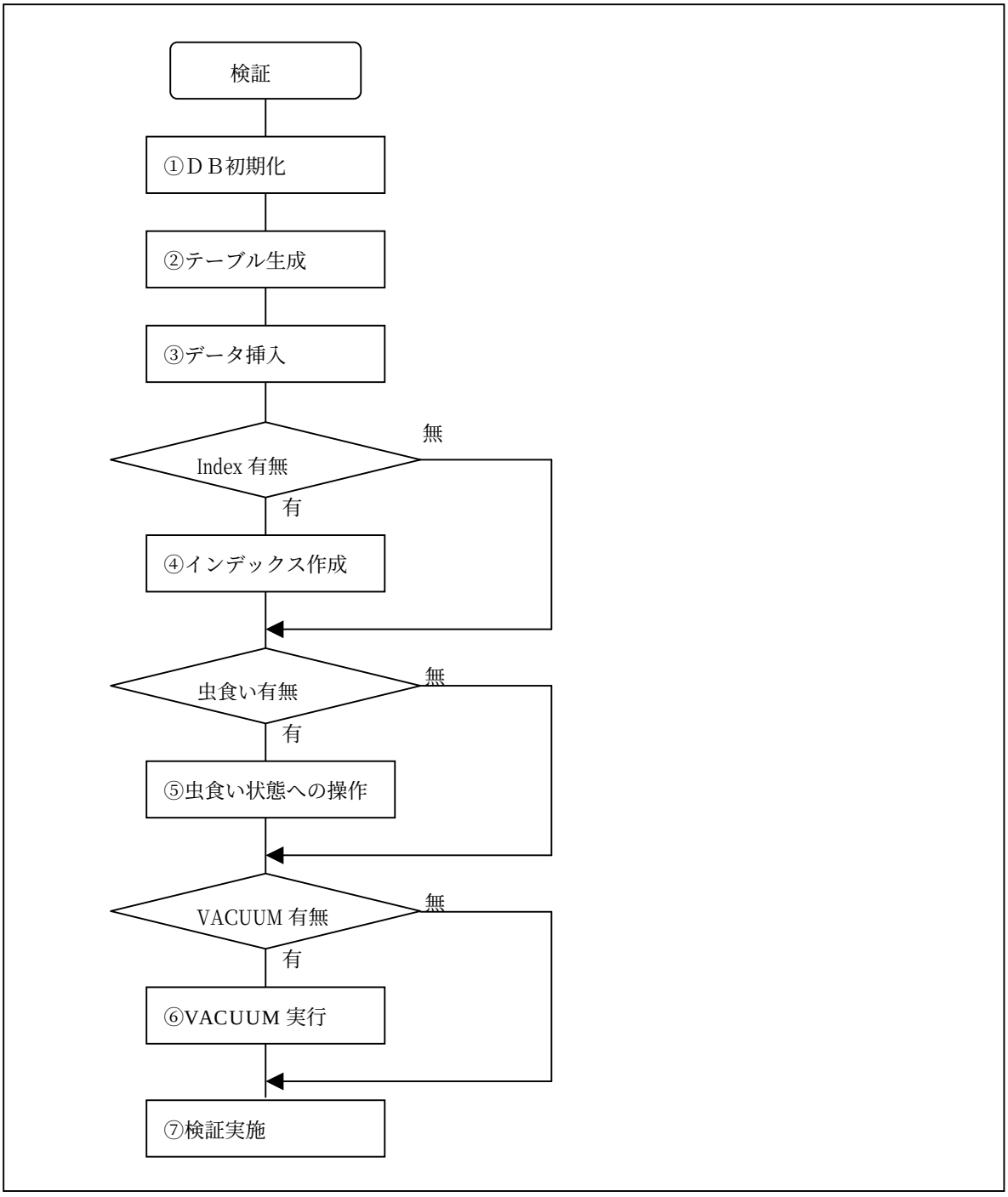


図 3-3　検証フロー

（２）フロー説明

①　DB 初期化

A)　環境変数＄PGDATA に設定されているデータファイル格納ディレクトリ以下を全削除する。

postgres> rm －r $PGDATA

B)　DB 初期化を行う。

postgres> initdb

C)　DB の作成を行う。

postgres> createdb oisadb

②テーブル作成

A)　郵便番号データテーブルの作成を行う。

postgres> psql oisadb

oisadb=# \i *テーブル生成スクリプトのファイル名*

③　データ挿入

A)　検証ポイントのデータ件数分のデータファイルを用意する。

B)　インポート

oisadb=# copy postlist from *データファイル名* using delimiters ‘,’

④　インデックス作成

A)　インデックス作成スクリプトを実行する。

oisadb=# \i *インデックス作成スクリプトファイル名*

⑤　虫食い状態への操作

A)　全件削除する。

oisadb=# delete from postlist

B)　インポート

oisadb=# copy postlist from *データファイル名* using delimiters ‘,’

⑥　VACUUM 実行

A)　VACUUM 実行スクリプトを実行する。

oisadb=# \i *VACUUM 実行スクリプトファイル名*

⑦　検証実施

A)　データ検索スクリプトを実行する。

oisadb=# \i *データ検索スクリプトファイル名*

5） スクリプト

① ＰｏｓｔｇｒｅＳＱＬテーブル生成スクリプト

```
create table postlist
(
--    id            serial,
    jiscode         text,
    old_id          varchar(5),
    post_id         char(7),
    ken_k           text,
    si_k            text,
    tyou_k          text,
    ken             text,
    si              text,
    tyou            text,
    somezips        int4,
    aza             int4,
    choume          int4,
    sometowns       int4,
    changed         int4,
    changed_why     int4
)
;
```

② Ｏｒａｃｌｅテーブル生成スクリプト

```
drop table postlist;
create table postlist
(
--    id              number(12) default 0,
    jiscode         varchar2(5),
    old_id          varchar2(5),
    post_id         varchar2(7),
    ken_k           varchar2(256),
    si_k            varchar2(256),
    tyou_k          varchar2(256),
    ken             varchar2(256),
    si              varchar2(256),
    tyou            varchar2(256),
    somezips        number(1),
    aza             number(1),
    choume          number(1),
    sometowns       number(1),
    changed         number(1),
    changed_why     number(1)
)
;
```

③　インポートファイル・ＣＯＰＹスクリプトの生成を行うスクリプト

```
#!/usr/local/bin/perl

#
# 現在のディレクトリを取得
#
$dir = `pwd`;
chomp($dir);

#
# 問い合わせを発行
#
print "filename---";
$filename = <STDIN>;
print "insert---";
$insert = <STDIN>;
exit if($insert == "");
print "delete---";
$delete = <STDIN>;
#exit if($delete == "");
while($vacuum ne "y" && $vacuum ne "n"){
print "vacuum(y/n)---";
$vacuum = <STDIN>;
chomp($vacuum);
}
while($index ne "y" && $index ne "n"){
print "index(y/n)---";
$index = <STDIN>;
chomp($index);
}

#
# CSV ファイルをオープン
#
open(FILE, "../DATA/ken_all_euc.txt");
@file = <FILE>;
close(FILE);

#
# copy 用ファイルの作成
#
chomp(@file);
```

```
        open(COPYFILE,">copyfile");
        for($i=0;$i<$insert;$i++){
        print COPYFILE "$file[$i]\n";
                push(@add,$file[$i]);
        }
        close(COPYFILE);

        #
        # ランダムに削除
        #
        for($i=0;$i<$delete;$i++){
        $delete_line = splice(@add,rand @add,1);
        @koumoku = split(/,/,$delete_line);
        # delete 文作成
        $delete_zip = splice(@zip,rand @zip,1);
        #       push(@delete,"delete from postlist where post_id =
'".$koumoku[2]."' and tyou = '".$koumoku[8]."';\n");
                push(@delete,"delete from postlist where post_id =
'.$koumoku[2].';\n");
        }

        #
        # 新しいファイルに出力
        #
        open(NEW_FILE,">$filename");
        print NEW_FILE "--$filename";
        print NEW_FILE "--insert -> $insert";
        print NEW_FILE "--delete -> $delete";
        print NEW_FILE "--vacuume-> $vacuum";
        print NEW_FILE "--index  -> $index\n";
        print NEW_FILE "copy postlist from '".$dir."/copyfile' delimiters ',';\n" ;
        print NEW_FILE @delete;
        print NEW_FILE "vacuum postlist;\n" if($vacuum eq "y");
        print NEW_FILE "create index post_id_index on postlist(post_id);\n"
if($index eq
         "y");
        print NEW_FILE "drop index post_id_index;\n" if($index eq "n");
        close(NEW_FILE);
```

④　データ検索スクリプト

```
\echo
##########################################################################
\echo ******** 最小の OID を算出 ********
\i /home/oisa/ura/now.sql
select min(oid) from postlist;
\i /home/oisa/ura/now.sql
\echo
##########################################################################
\echo ******** 最大の OID を算出 ********
\i /home/oisa/ura/now.sql
select max(oid) from postlist;
\i /home/oisa/ura/now.sql



\echo
##########################################################################
\echo ******* = *******
-- No1
\echo
==========================================================================
\i /home/oisa/ura/now.sql
select * from postlist where post_id='0600000' ;
\i /home/oisa/ura/now.sql

\echo
##########################################################################
\echo ******* LIKE *******
-- No1
\echo
==========================================================================
\i /home/oisa/ura/now.sql
select * from postlist where post_id like '060%'  ;
\i /home/oisa/ura/now.sql

\echo
##########################################################################
\echo ******* sub select *******
-- No1
\echo
==========================================================================
\i /home/oisa/ura/now.sql
select * from postlist where post_id in
```

```
(select post_id from postlist where post_id='0600000')   ;
\i /home/oisa/ura/now.sql
```

3－3　検証の結果

１）検証結果

件数で比較した結果を以下に示す。

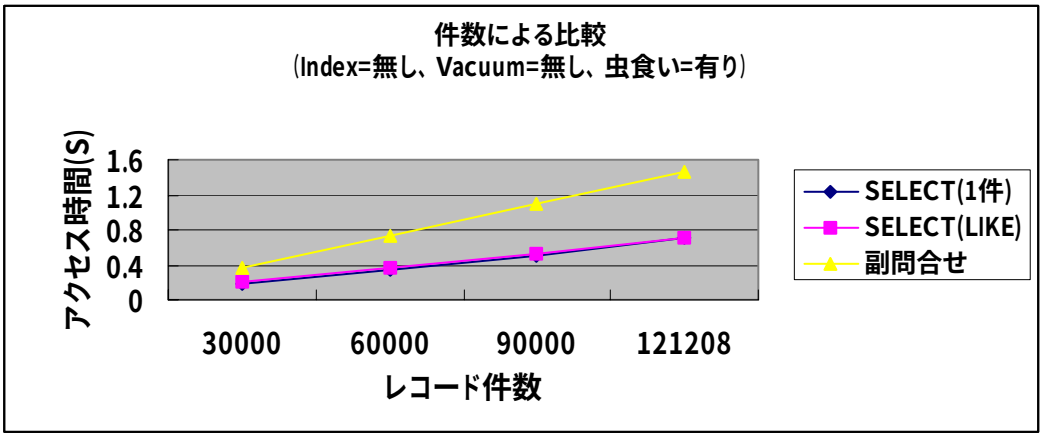


図 3-4　件数による比較 (Index=無し、Vacuum=無し、虫食い=有り)

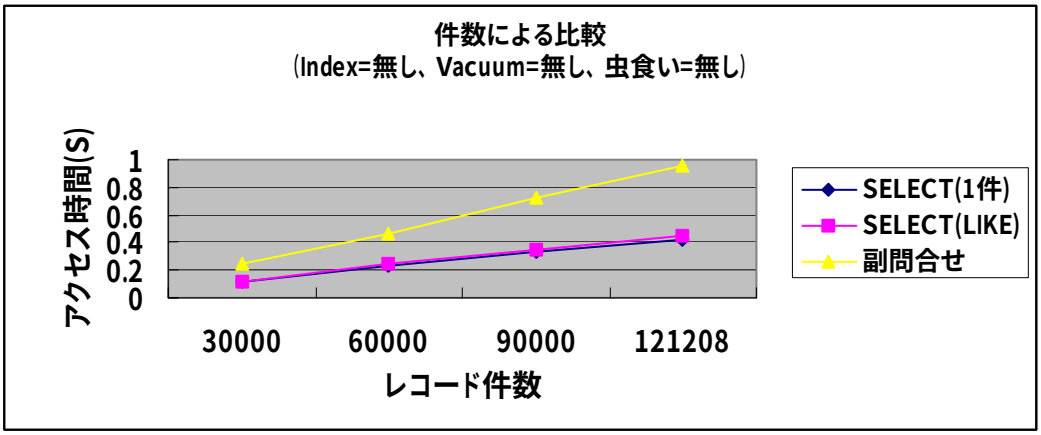


図 3-5　件数による比較 (Index=無し、Vacuum=無し、虫食い=無し)

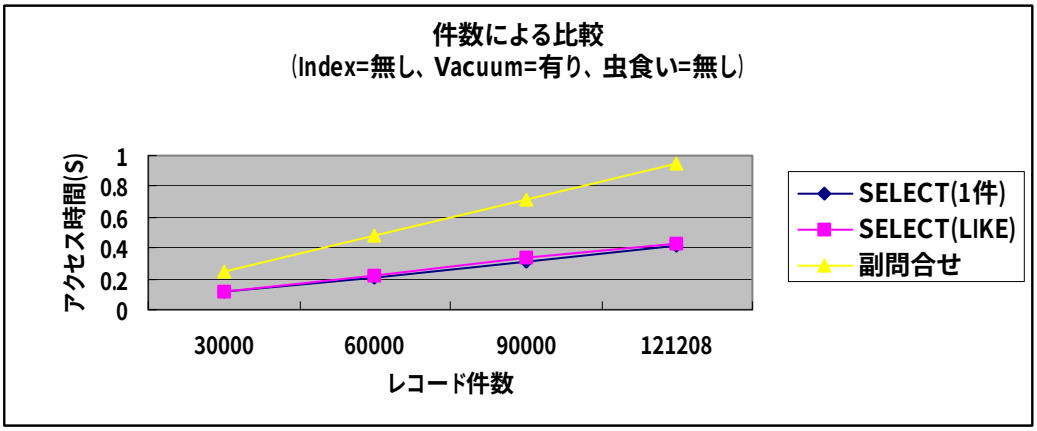


図 3-6　件数による比較 (Index=無し、Vacuum=有り、虫食い=無し)

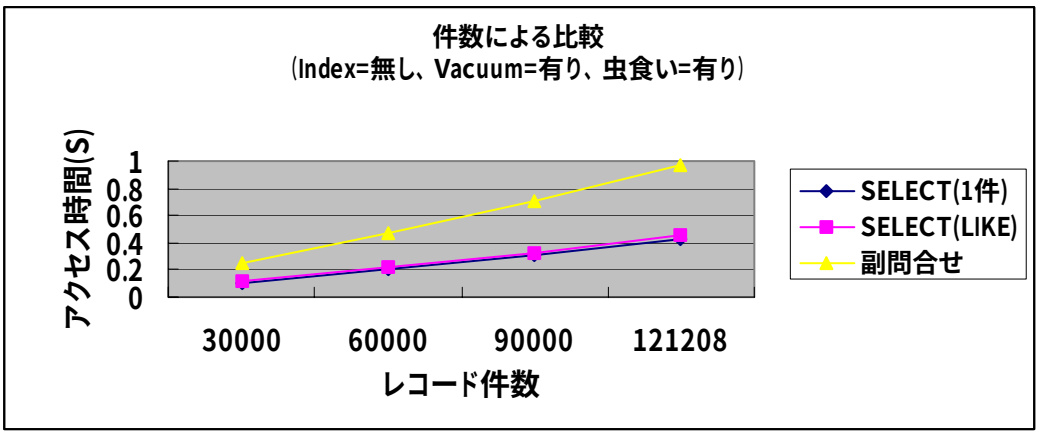


図 3-7　件数による比較 (Index=無し、Vacuum=有り、虫食い=有り)

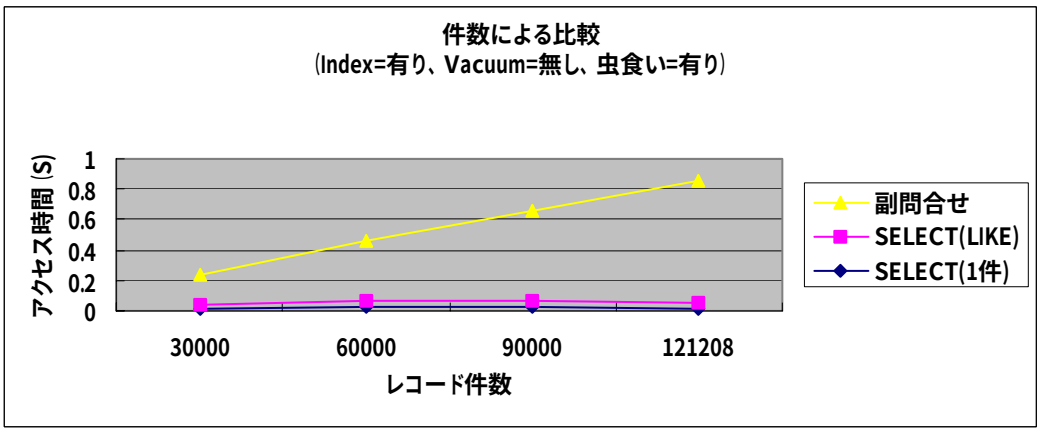


図 3-8　件数による比較 (Index=有り、Vacuum=無し、虫食い=有り)

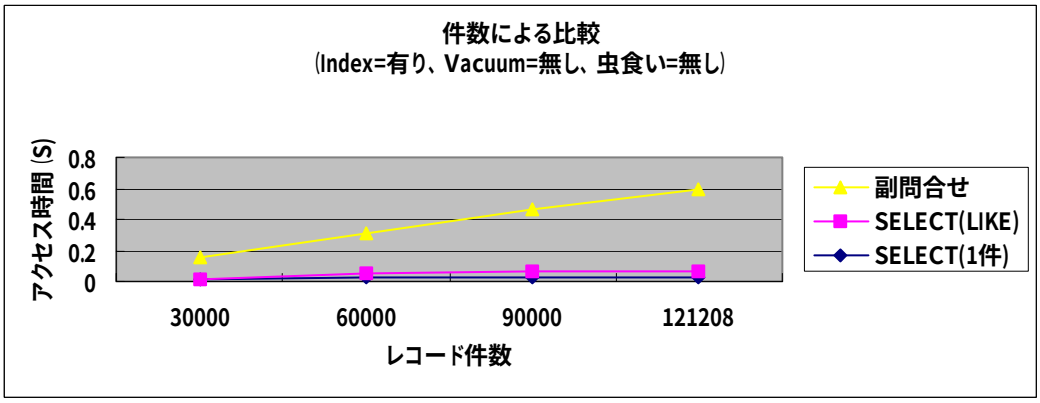


図 3-9　件数による比較 (Index=有り、Vacuum=無し、虫食い=無し)

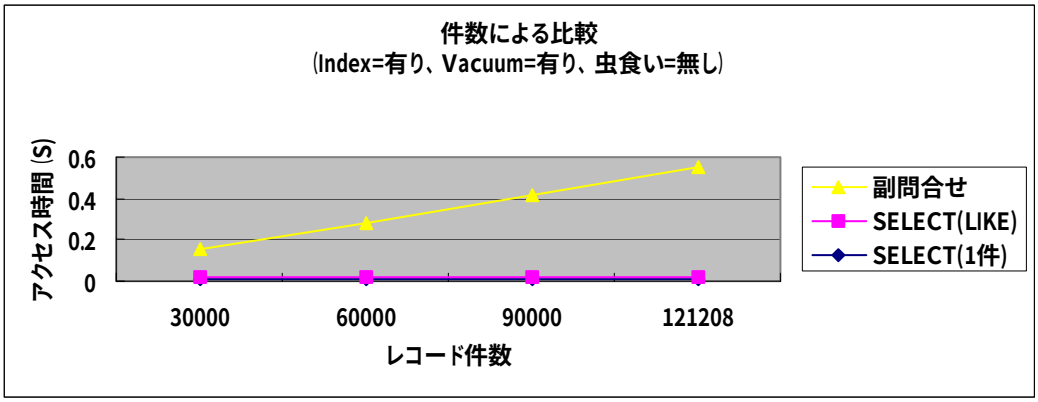


図 3-10　件数による比較 (Index=有り、Vacuum=有り、虫食い=無し)

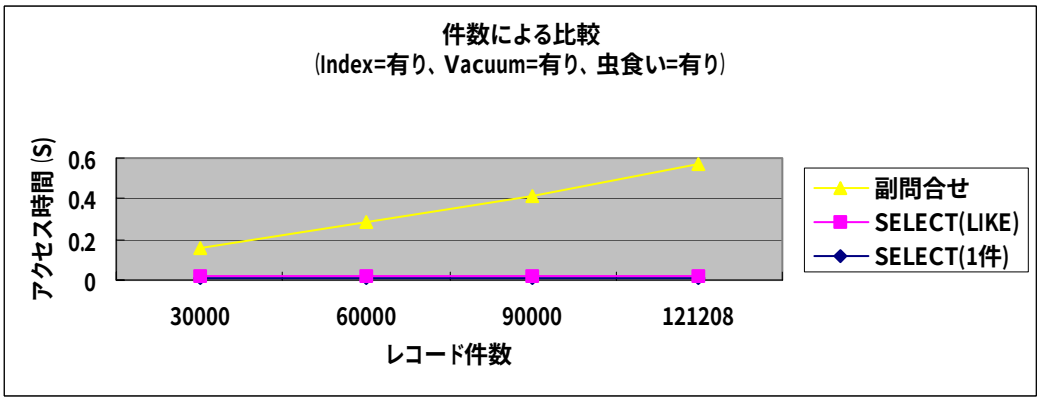


図 3-11　件数による比較 (Index=有り、Vacuum=有り、虫食い=有り)

以下の項目にまとめられる。

① 予想通り件数と比例してアクセス時間も増加している。しかし、Index をつけると SELECT 1 件、LIKE 検索の差は殆ど出なかった。

② Index 無し時の虫食い状態の有無を比べると、アクセス速度（一件あたりのアクセス時間）の違いから、vacuum の効果が伺える。運用時は Index の付加と vacuum 使用頻度に注意が必要。

③ Index 無し時の虫食い状態の有無を比べると、アクセス速度の違いから vacuum の効果が伺える。運用時は Index の付加と vacuum 使用頻度に注意が必要。

④ 副問い合わせより、前方一致検索の方が良いレスポンスを示した。

2）　まとめ

検証結果をまとめると、今回検証した如何なる条件においてもレコード件数に対するアクセス時間は比例関係にあり、傾きは有効桁数２桁に揃えると（n.m）×10^{-6} となることから、レコード件数が 10^{6}(100 万)件以内であれば約 10 秒以内で処理が可能であること予想される。また、インデックス項目からの検索ではインデックスが無い場合に比べ約 3 倍以上アクセス速度が速いことがわかり、一件検索に対して副問い合わせを用いて行う場合とそうでない場合とを比べると約 3 倍以上アクセス速度が速いことがわかる。虫食いが起こった場合には vacuum を行うとファイルサイズが減少し、アクセス速度は約 1.5 倍向上することがわかった。