



トランザクションログを サポートした PostgreSQL 7.1について

(株)SRAオープンソースビジネス部

石井達夫

t-ishii@sra.co.jp



Contents

- PostgreSQLとは
 - PostgreSQLの歴史
 - PostgreSQLの特長
 - PostgreSQLの機能
- PostgreSQL 7.1の新機能
 - TOAST
 - OUTER JOIN
 - WAL
- 今後の予定



PostgreSQLとは

- カリフォルニア大学バークレー校(UCB)で開発、現在は世界中のボランティアの手によって維持
- 本格的なオープンソースデータベース
- Unix/Linux/Windowsなどで稼動
- <http://www.postgresql.org/>



PostgreSQLの歴史

- UCB(カリフォルニア大学バークレー校)
 - Ingres(1977-1985)
 - postgres(1986-1994)
- インターネットでの開発
 - postgres95(1994-1995)
 - Illustra
 - PostgreSQL(1996-)



PostgreSQLの歴史

6.1	1997/06	GEQOオプティマイザ、GROUP BY、SETコマンド
6.2	1997/10	JDBCドライバ、TRIGGER、多数のSQL92データ型
6.3	1998/03	subselect、マルチバイトサポート、ecpg、Pythonインターフェイス
6.4	1998/10	PL/PgSQL、PL/tcl、HAVING
6.5	1999/11	行ロック、MVCC、TRANSACTION ISOLATION LEVEL、LOCK TABLE IN...、SELECT FOR UPDATE、NUMERIC、LIMIT/OFFSET
7.0	2000/05	外部キー
7.1	2001/?	WAL、TOAST、OUTER JOIN



PostgreSQLの特徴

- 関係データベース+オブジェクト指向拡張
- 強力なトランザクション管理機能
 - 行ロック、MVCC、2レベルのtransaction isolation level
- 容易な管理
- 大規模データ・大規模ユーザの管理可能
- 多彩なAPIをサポート
 - C, C++, Java, Perl, Tcl/Tk, PHP...



PostgreSQLの機能

	PostgreSQL 7.1	商用DB	MySQL
トランザクション	○	○	△
MVCC	○	△	×
行ロック	○	○	×
外部キー	○	○	×
Subselect	○	○	×
外部結合	○	○	○
ユーザ定義データ型	○	△	×
Stored procedure	○	○	×
Trigger	○	○	×
Rule	○	×	×
分散データベース	×	○	×
Replication	×	△	×
トランザクションログ	○	○	×
マルチバイト対応	○	△	△
オープンソース/フリーソフト	○	×	○



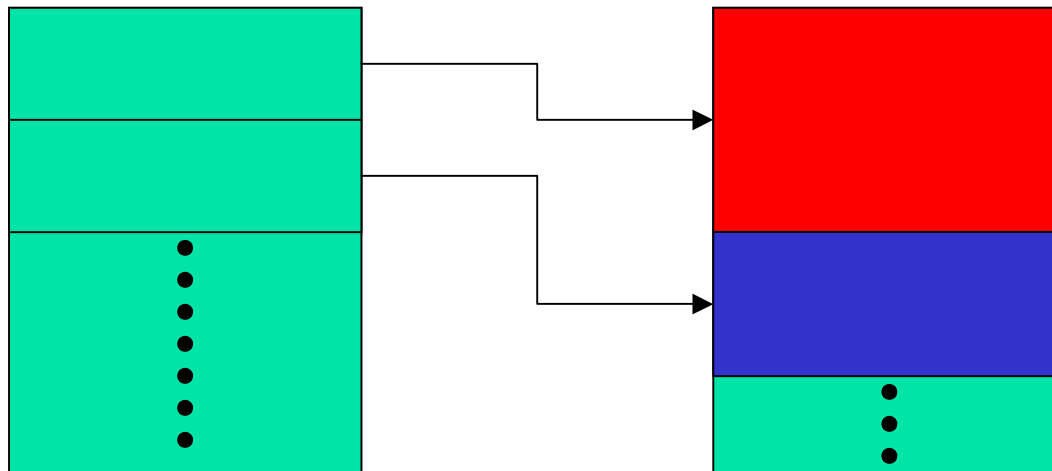
PostgreSQL 7.1の新機能

- TOAST (The Oversized-Attribute Storage Technique)
- OUTER JOIN
- WAL (Write-Ahead Logging)
- その他

TOAST (The Oversized-Atribute Storage Technique)



- レコードサイズ制限を撤廃
- 自動圧縮機能
- <http://www.postgresql.org/projects/devel-toast.html>



OUTER JOIN(外部結合)

- **SELECT * FROM t1 LEFT OUTER JOIN t2 on t1.i = t2.k;**

t1		t2		JOIN後の結果			
i	j	k	l	i	j	k	l
1	A	1	B	1	A	1	B
2	C	3	D	2	C	NULL	NULL

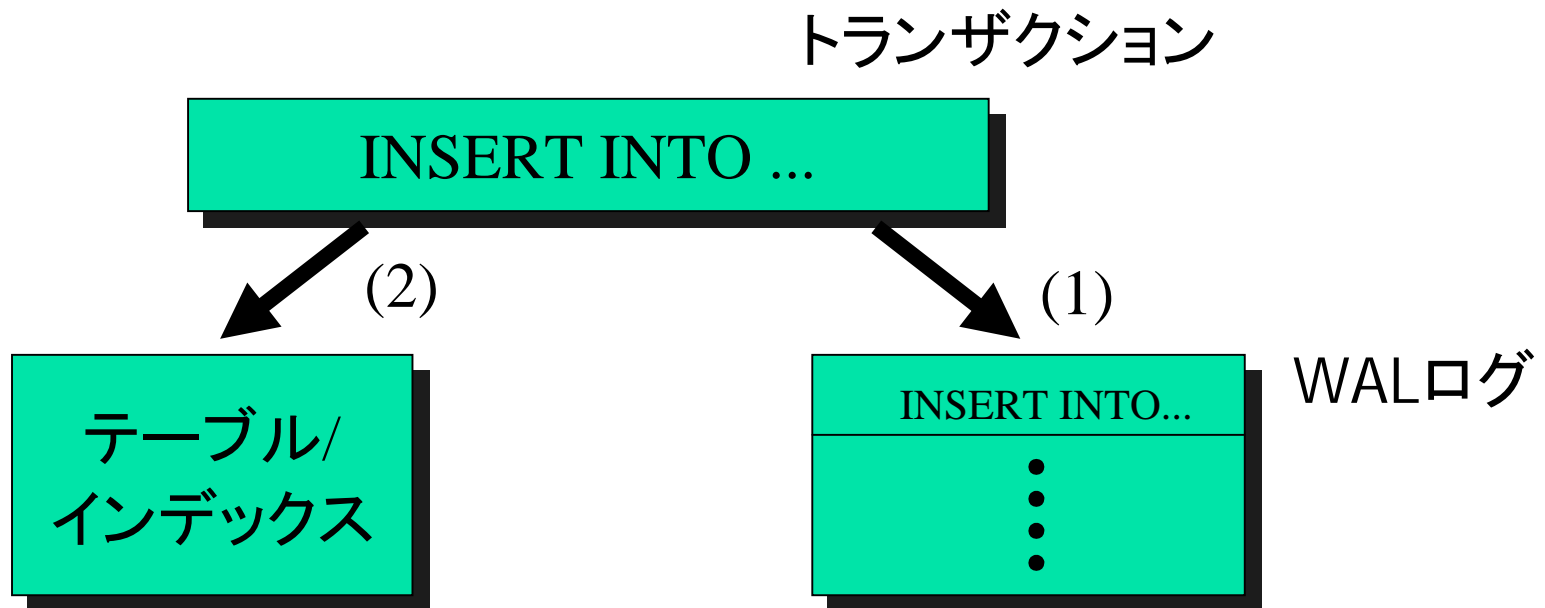


OUTER JOIN(外部結合)

- サポートされるJOIN構文
 - RIGHT OUTER JOIN
 - NATURAL JOIN など
- サポートされない構文
 - UNION JOIN

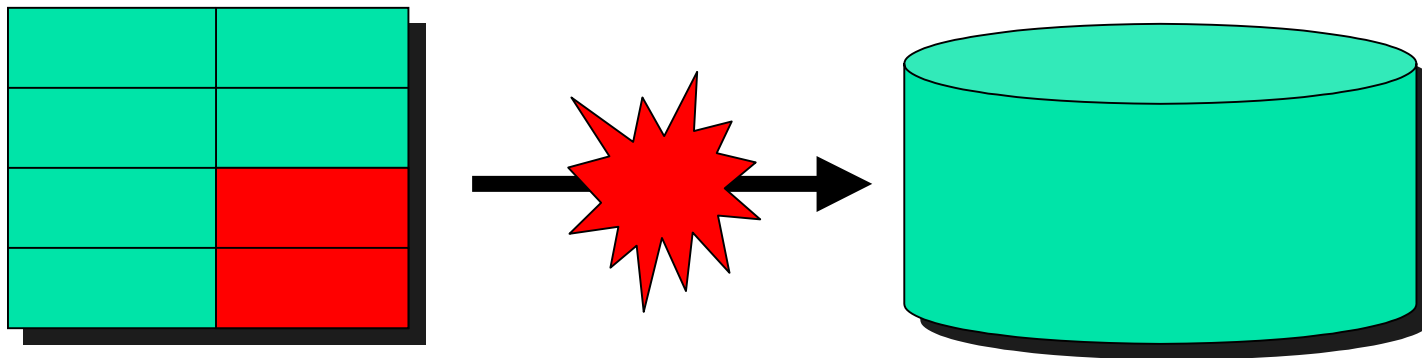
WAL (Write-Ahead Logging)

- ログを先に書いてからデータファイルを書く



WALの特長

- より高い信頼性
 - データページの不完全書き込みによる障害にも対応





postmasterの停止

```
pg_ctl stop
```

```
pg_ctl -m モード stop
```

モード: i: 即時停止

f: 高速停止

s: スマートシャットダウン



クラッシュリカバリ

- WALによりデータを回復
- データベースが異常終了→次回データベース立ち上げ時に自動回復

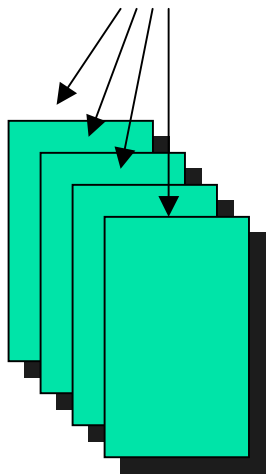
```
DEBUG: Data Base System was interrupted being in production at Mon Nov 13 14:02:04 2000
DEBUG: CheckPoint record at (0, 291388)
DEBUG: Redo record at (0, 291388); Undo record at (0, 0); Shutdown TRUE
DEBUG: NextTransactionId: 584; NextOid: 17891
DEBUG: The DataBase system was not properly shut down
      Automatic recovery is in progress...
DEBUG: Redo starts at (0, 291444)
DEBUG: Redo done at (0, 298540)
DEBUG: Data Base System is in production state at Mon Nov 13 14:02:17 2000
```

WALの特長

- 信頼性と性能の両立

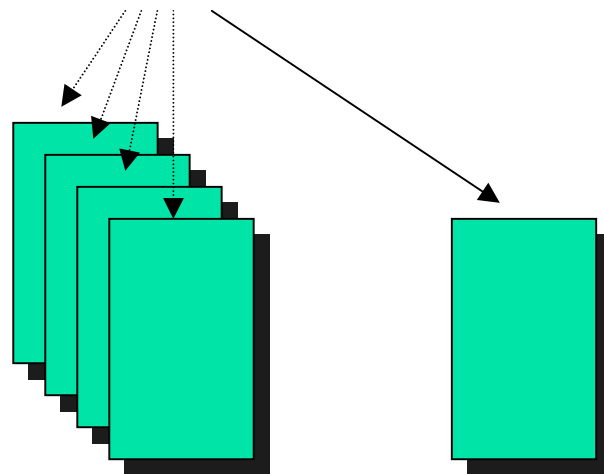
7.1以前

fsync()



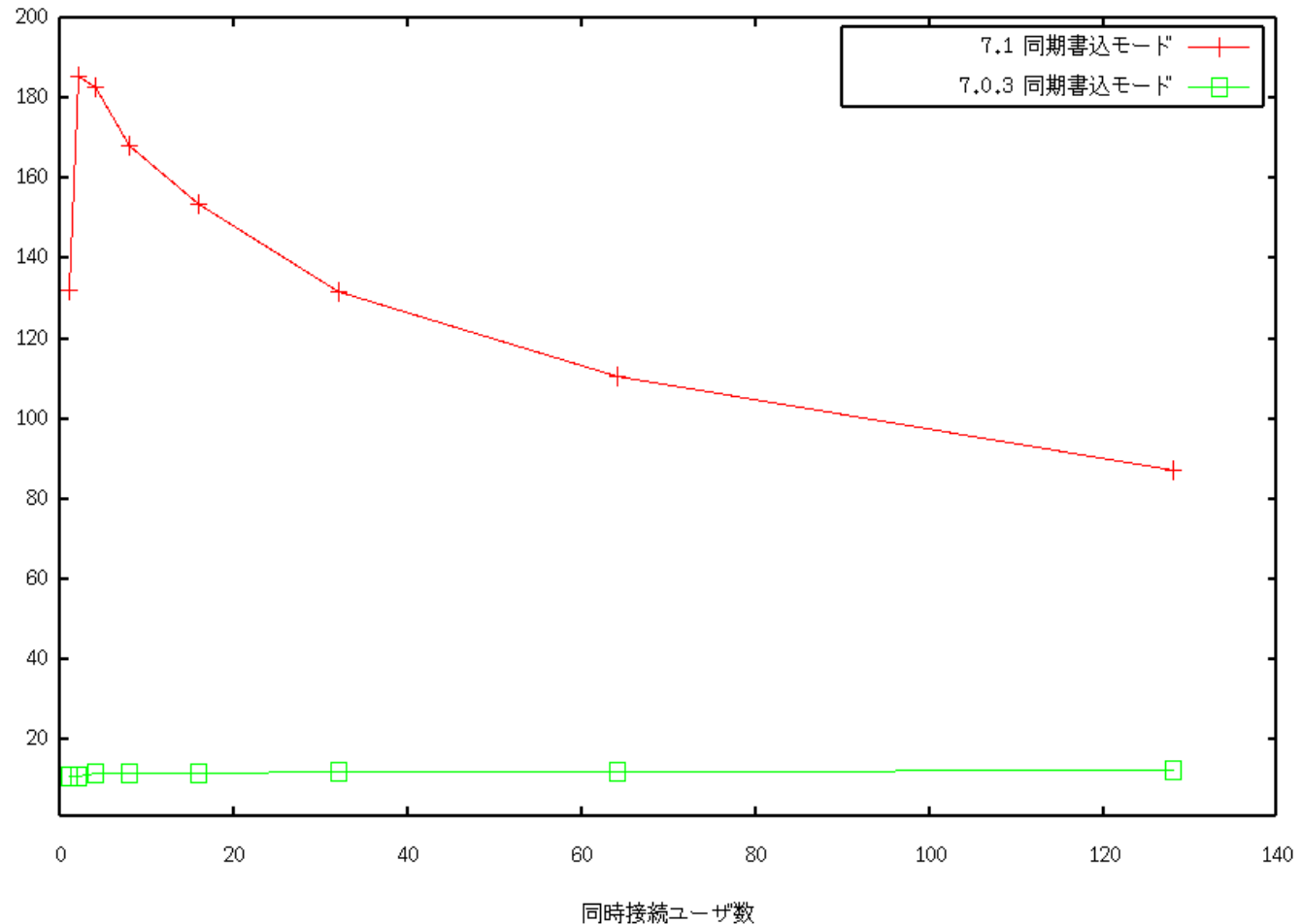
7.1以後

fsync()

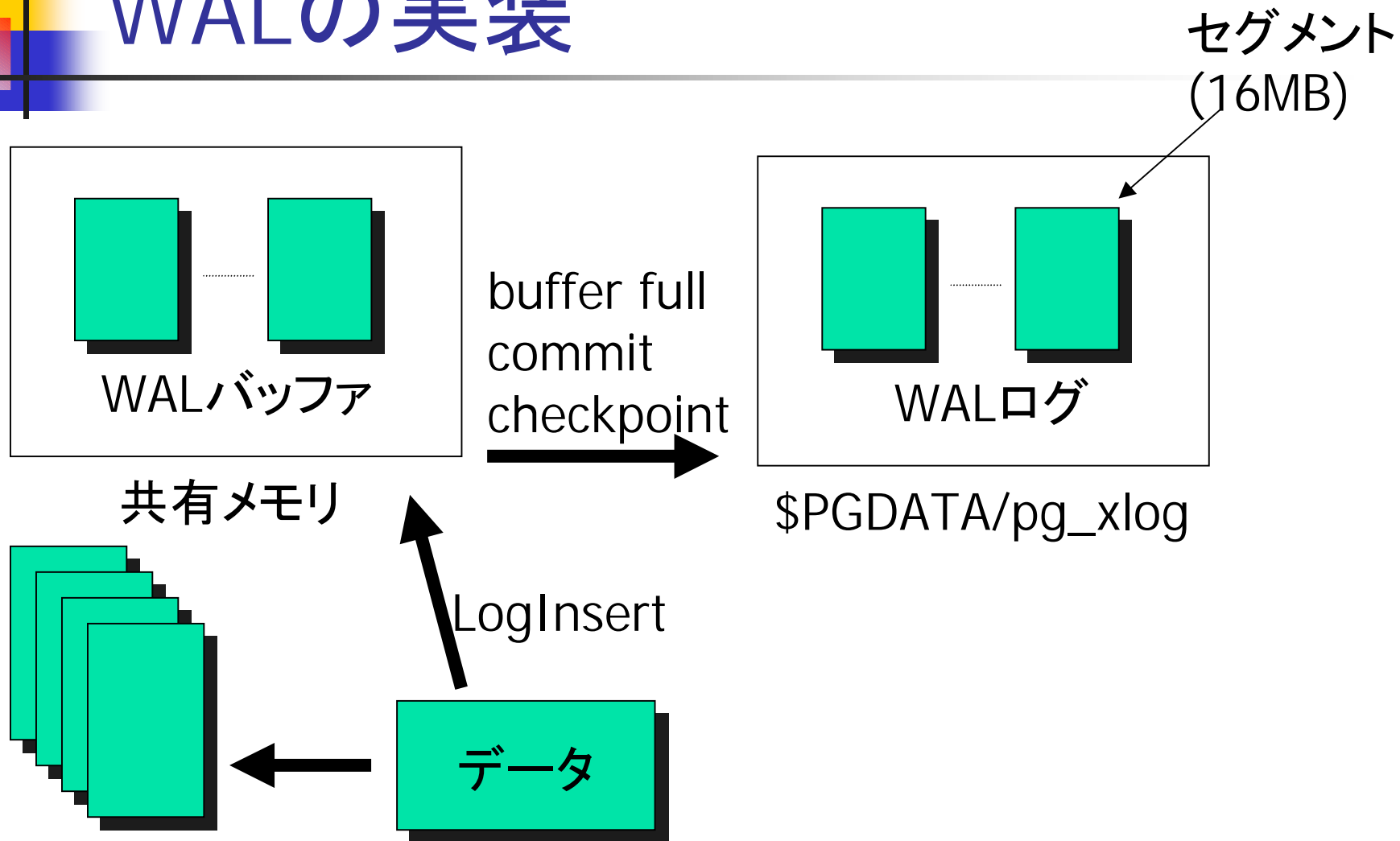


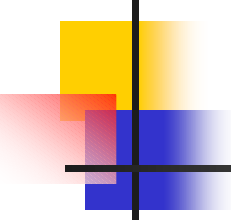
7.1と7.0.3の同期モードでの性能の違い

トランザクション数/秒 (TPS)



WALの実装





WALのチューニングポイント

- WAL_BUFFERS
 - WALバッファの数(デフォルト8)
- WAL_FILES
 - WALセグメントの数(デフォルト0)
- CHECKPOINT_TIMEOUT
 - checkpointの間隔(デフォルト300秒)
- COMMIT_DELAY
 - Logレコードを書いたからfsync()するまでの時間(デフォルト5マイクロセカンド)



WALの今後

- UNDOの実装
 - pg_logの縮小
 - “savepoint”の実装
- Backup and Restore(BAR)の実装
 - クラッシュリカバリー機能の向上



7.1 その他

- GUC(Grand Unified Configuration scheme)
- Unicode対応
- new pg_dump
- FROM句でのsubselect
- function managerの改良
- 関数呼び出しのメモリ使用量の削減
- large object改良
- LIKE/ILIKE



GUC(Grand Unified Configuration Scheme)

- ほとんどの設定ファイルを
\$PGDATA/postgresql.confに統合
- 今までは...
 - postmasterの引数
 - SETコマンド
 - pg_options



記述形式

```
# This is a comment  
log_connections = yes  
syslog = 2
```

* キーワードは大文字／小文字の区別をしない



GUCの設定例

- `postmaster -B 1024 -N 128 -S -o "-F" -i`
 - `shared_buffers = 1024`
 - `max_connections = 128`
 - `silent_mode = on`
 - `fsync = off`
 - `tcpip_socket = on`



設定データのタイプ

- Boolean
 - ON/OFF, TRUE/FALSE, YES/NO, 1/0(大文字／小文字の区別はない)
- integer
 - 整数
- floating point
 - 浮動小数点
- string
 - 整数



postmasterの引数でオプション を設定する

```
postmaster -c log_connections=yes -c syslog=2
```



環境変数によるオプション設定

```
env PGOPTIONS='-c geqo=off' psql
```

postmaster立ち上げ時のみ有効な主なオプション

MAX_CONNECTIONS	整数	最大接続数
PORT	整数	ポート番号
SHARED_BUFFERS	整数	共有バッファ数
SILENT_MODE	bool	デーモンモード
SORT_MEM	整数	ソートメモリバッファサイズ
SSL	bool	Secure TCP/IPコネクション
TCPIP_SOCKET	bool	INET dmainソケット有効
HOSTLOOKUP	bool	ホスト名のlookupを行う
LOG_CONNECTIONS	bool	クライアントの接続をログの対象にする
SYSLOG	整数	0: syslog無効、1:syslogにも出力、 2:syslogのみ



postmaster立ち上げ後も変更可能な主なオプション

DEBUG_LEVEL	整数	デバッグレベル
LOG_PID	bool	プロセスIDをログに付加する
LOG_TIMESTAMP	bool	タイムスタンプをログに付加する
show_source_port	bool	接続元のポート番号を表示する
log_connections	bool	クライアント接続ログを有効にする
debug_print_query	bool	クエリを表示する



設定値が有効になるタイミング

- postmaster起動時
- シグナル
 - HUPシグナルをpostmasterに送る
- セッション毎
 - `env PGOPTIONS="-c geqo=off" psql test`
- on the fly
 - `SET geqo TO off;`



Unicode対応

- Unicodeとは

- 世界中の文字を一つの文字集合で表す試み
 - Unicodeコンソーシアム
 - ISO/IEC 10646
 - JIS X 0221
 - ISO: International Organization for Standardization(国際標準化機構)
 - IEC: International Electrotechnical Commission(国際電気標準会議)



Unicodeのエンコーディング

- UCS-2
 - 16ビット固定
 - プログラムで扱いにくい
- UTF-8
 - ASCIIと互換性があるので、プログラムで扱いやすい
 - ASCIIは1バイト、ラテン文字は2バイト、日本語などは3バイトになる
 - PostgreSQLで採用



Unicodeのメリット

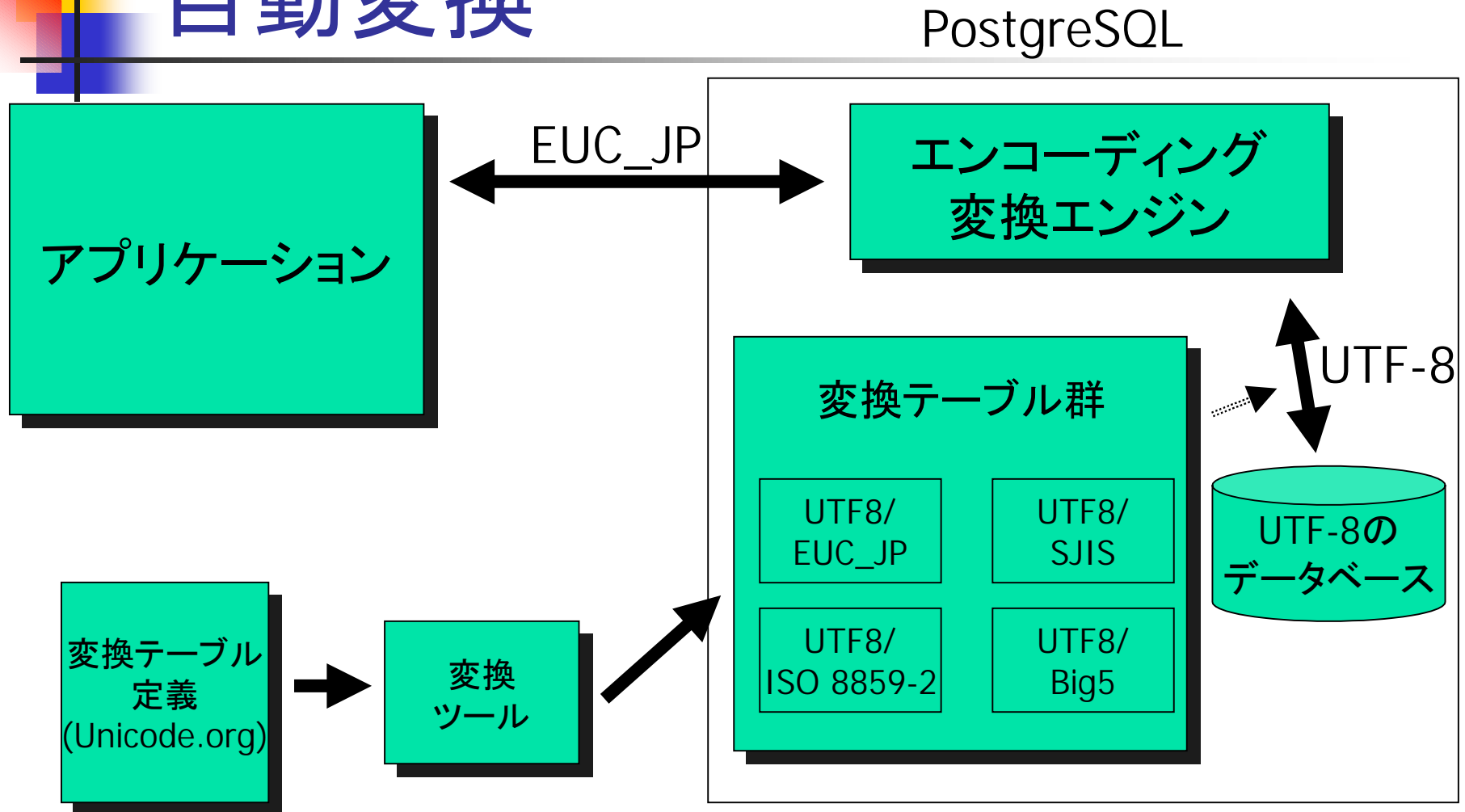
- 複数言語対応をアプリケーションを容易に構築できる



Unicodeの問題点

- 日本語・中国語・韓国語の漢字がごちゃまぜになっている
- Unicodeを扱うことのできるOSやソフトがまだ少ない
- Unicode用のフォントが少ない
- ほかの文字コードと相互変換が難しい
- そこで、データベースサーバで文字コード変換を行う
 - クライアントはUnicodeを意識する必要がない

PostgreSQLにおける文字コード 自動変換





Unicodeと相互変換可能な 文字コード

- EUC_JP
- SJIS
- EUC_KR
- EUC_CN
- BIG5
- EUC_TW
- ISO 8859-1から5まで

文字コード相互変換表

バックエンド

フロントエンド

	SQL_ASCII	EUC_JP	EUC_CN	EUC_KR	EUC_TW	Unicode	LATIN
SQL_ASCII		×	×	×	×	×	×
EUC_JP	×		×	×	×	○	×
EUC_CN	×	×		×	×	○	×
EUC_KR	×	×	×		×	○	×
EUC_TW	×	×	×	×		○	×
Unicode	○	○	○	○	○		○
LATIN	×	×	×	×	×	○	
SJIS	×	○	×	×	×	○	×
Big5	×	×	×	×	○	○	×



使い方

```
$ createdb -E UNICODE unicode
```

```
$psql
```

```
CREATE TABLE t1(id integer, japanese text, chinese text);
```

```
¥encoding EUC_JP
```

```
INSERT INTO t1 VALUES(1, '日本語');
```

```
¥encoding BIG5
```

```
UPDATE t1 SET chinese = '中文' WHERE id = 1;
```



今後の課題

- 変換テーブルの肥大化
 - → テーブルの動的ロード
- 変換可能文字コードの追加
 - → CREATE CHARACTER SETの実装



new pg_dump

- ラージオブジェクトのバックアップが可能に
- 自動圧縮バックアップ
- pg_restore



new pg_dumpの使い方

カスタムフォーマット によるバックアップと リストア

```
$ pg_dump -F c -b dbtest >  
/tmp/db.out
```

```
$ pg_restore /tmp/db.out --db=dbtest2
```

```
$ pg_restore -l /tmp/db.out  
;  
; Archive created at Sat Jan 13 14:13:55 2001  
;   dbname: dbtest  
;   TOC Entries: 4  
;   Compression: -1  
;   Dump Version: 1.4-22  
;   Format: CUSTOM  
;  
;  
; Selected TOC Entries:  
;  
2; 2208385 TABLE t1 t-ishii  
3; 2208385 TABLE DATA t1 t-ishii  
4; 0 BLOBS BLOBS
```



pg_dumpの主なオプション

<code>-a, --data-only</code>	dump out only the data, not the schema
<code>-b, --blobs</code>	dump out blob data
<code>-d, --inserts</code>	dump data as INSERT, rather than COPY, commands
<code>-F, --format {c f p}</code>	output file format (custom, files, plain text)
<code>-o, --oids</code>	dump object ids (oids)
<code>-s, --schema-only</code>	dump out only the schema, no data
<code>-S, --superuser <name></code>	specify the superuser username to use in plain text format
<code>-t, --table <table></code>	dump for this table only
<code>-Z, --compress {0-9}</code>	compression level for compressed formats

pgaccessと日本語メニュー

PostgreSQL access

データベース オブジェクト ヘルプ

新規 開く デザイン

テーブル
クエリ
ビュー
シーケンス
関数
レポート
フォーム
スクリプト
ユーザ
スキーマ

test2

ビジュアルクエリデザイナー

テーブル追加

SQL表示 SQL実行 クエリビルダに保存

顧客マスタ
顧客ID
顧客名
会社名
住所
電話番号
FAX番号

販売実績
顧客ID
商品ID
販売数量

商品マスタ
商品ID
商品名
単価

テーブル

フィールドでソート フィルター条件 再読込 閉じる

顧客名	会社名	商品名	売上
はなこ	はなこ株式会社	みくろう時計	300000
はなこ	はなこ株式会社	やまねこ時計	294000

フォームデザイン

問い合わせ実行

Choose color

Red: 0
Green: 0
Blue: 0

Selection: #000000

OK Cancel



そのほか

- BIT, BIT VARYING
- VARIANCE, STDDEV
- ESCAPE
 - SELECT * FROM t1 WHERE T LIKE 'A%'
ESCAPE '¥¥';
- ILIKE



そのほか

- サブクエリに関する制限の撤廃
 - `SELECT (SELECT MAX(j) FROM t2),j FROM t1;`
 - `SELECT t3.i, COUNT(t3.j) FROM (SELECT * FROM t1 UNION SELECT * FROM t2) AS t3 GROUP BY t3.i;`
 - `ORDER BY, LIMIT`の使用



そのほか

- ORDER BY, DISTINCT, UNION, LIMIT、集約関数を含むVIEW
- ALTER TABLE, DROP TABLEがロールバック可能



ONLYキーワード

```
test=# create table t2 (j int) inherits(t1);
```

```
CREATE
```

```
test=# insert into t1 values(1);
```

```
INSERT 2484814 1
```

```
test=# insert into t2 values(2,3);
```

```
INSERT 2484815 1
```

```
test=# select * from t1;
```

```
i
```

```
---
```

```
1
```

```
2
```

```
(2 rows)
```

```
test=# select * from only t1;
```

```
i
```

```
---
```

```
1
```

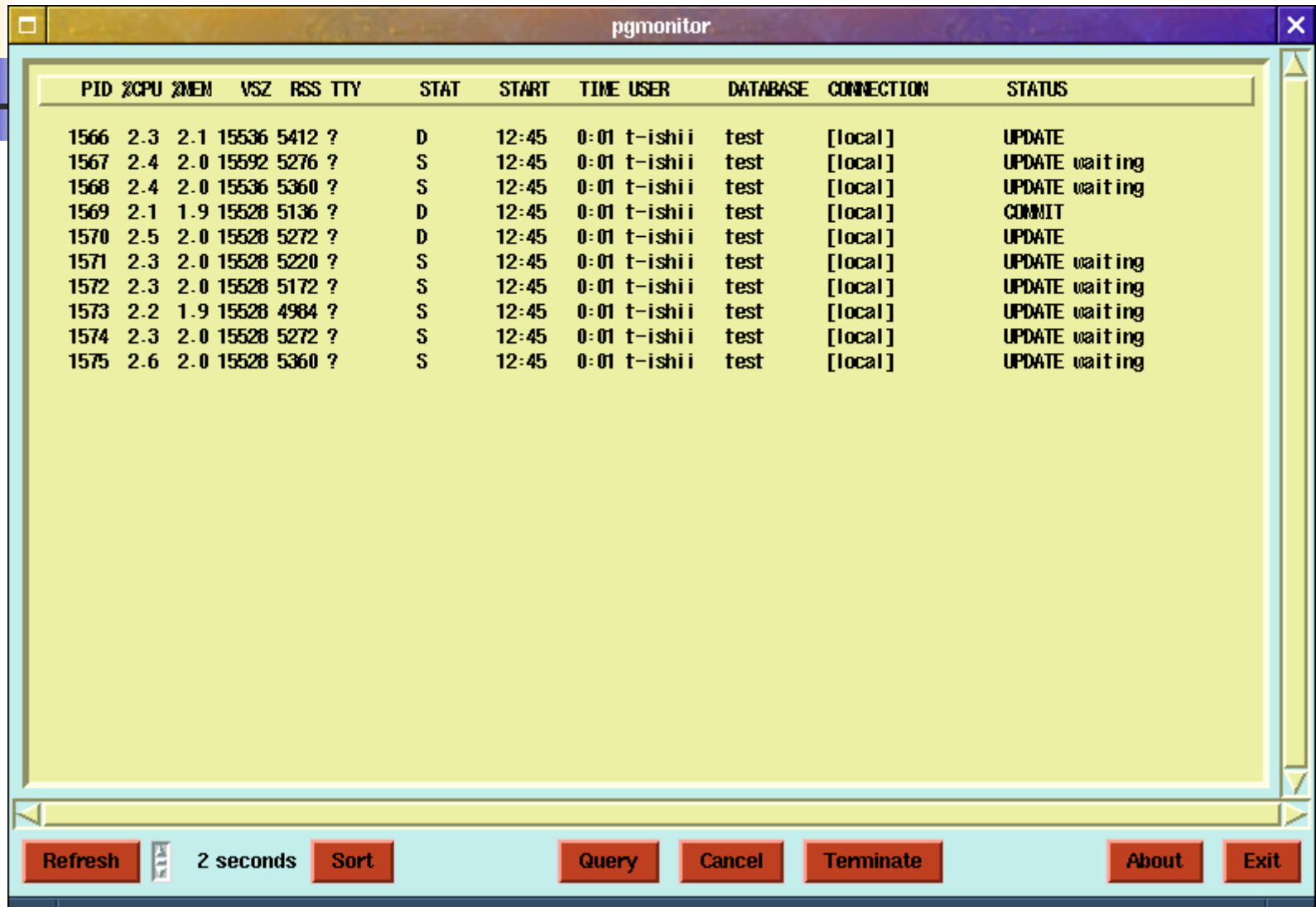
```
(1 row)
```



今後の予定

- レプリケーション
 - contrib/rserv
- vacuum頻度の軽減
 - 空きタプルの再利用
- BAR

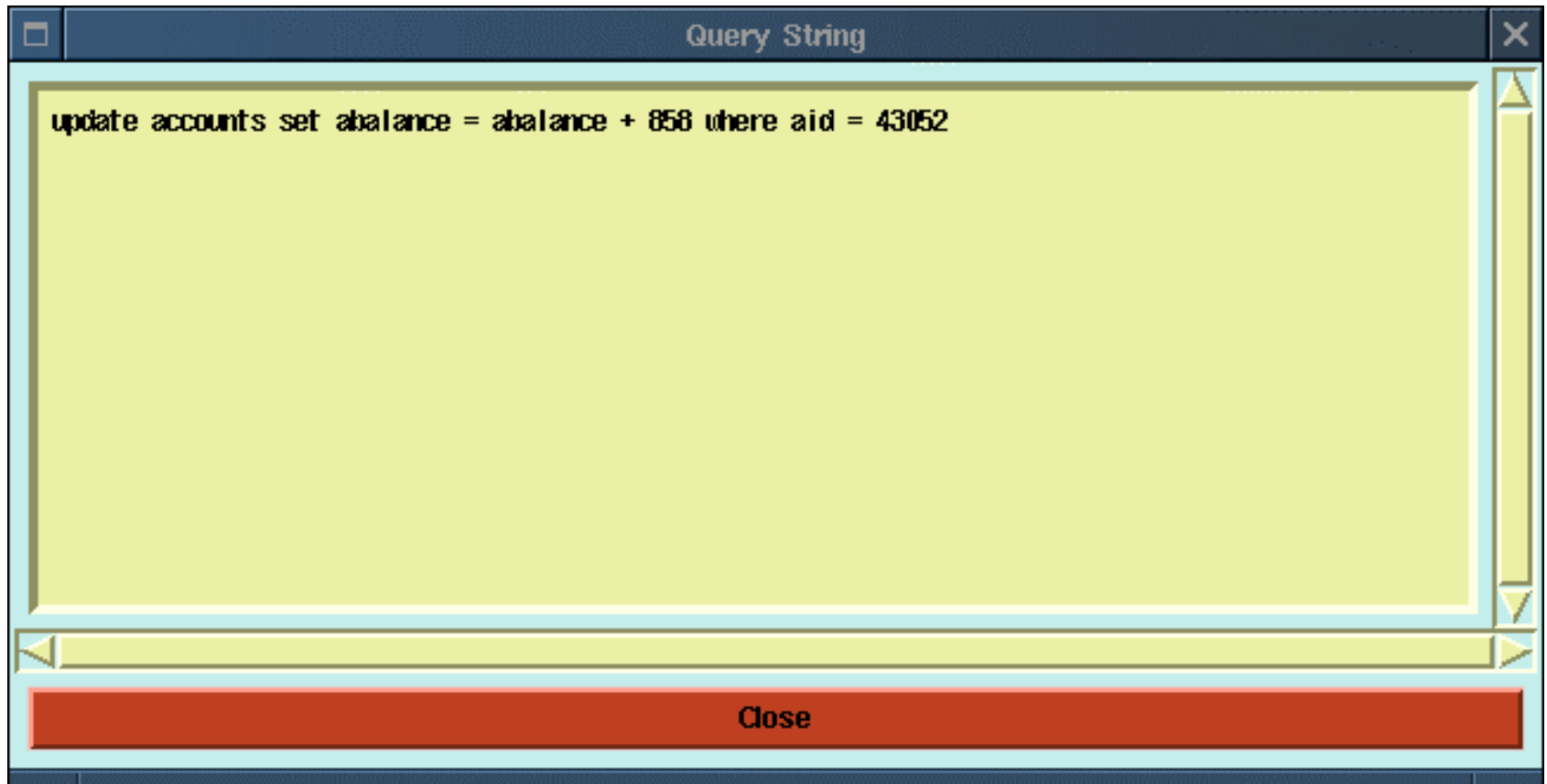
PostgreSQLモニタリングツール: pgmonitor



The screenshot shows the pgmonitor application window. The window title is "pgmonitor". The main content is a table with the following columns: PID, %CPU, %MEM, VSZ, RSS, TTY, STAT, START, TIME, USER, DATABASE, CONNECTION, and STATUS. The table contains 11 rows of data. At the bottom of the window, there is a control bar with buttons for Refresh, Sort, Query, Cancel, Terminate, About, and Exit. A refresh icon and the text "2 seconds" are also present.

PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	USER	DATABASE	CONNECTION	STATUS
1566	2.3	2.1	15536	5412	?	D	12:45	0:01	t-ishii	test	[local]	UPDATE
1567	2.4	2.0	15592	5276	?	S	12:45	0:01	t-ishii	test	[local]	UPDATE waiting
1568	2.4	2.0	15536	5360	?	S	12:45	0:01	t-ishii	test	[local]	UPDATE waiting
1569	2.1	1.9	15528	5136	?	D	12:45	0:01	t-ishii	test	[local]	COMMIT
1570	2.5	2.0	15528	5272	?	D	12:45	0:01	t-ishii	test	[local]	UPDATE
1571	2.3	2.0	15528	5220	?	S	12:45	0:01	t-ishii	test	[local]	UPDATE waiting
1572	2.3	2.0	15528	5172	?	S	12:45	0:01	t-ishii	test	[local]	UPDATE waiting
1573	2.2	1.9	15528	4984	?	S	12:45	0:01	t-ishii	test	[local]	UPDATE waiting
1574	2.3	2.0	15528	5272	?	S	12:45	0:01	t-ishii	test	[local]	UPDATE waiting
1575	2.6	2.0	15528	5360	?	S	12:45	0:01	t-ishii	test	[local]	UPDATE waiting

実行中のクエリの表示





おしまい

