

# PostgreSQL 8.3 - Ein Überblick

Bernd Helmle

26. August 2007

# Agenda

- Neue Funktionen im Bereich SQL, XML und Volltextsuche
- Geschwindigkeitsverbesserungen
- Neue Erweiterungen (contrib)

# XML (1)

- Neuer Datentyp `xml` (“well-formedness”)
- Viele Funktionen zur Verarbeitung von Daten zu XML
- XML-Export

## XML (2)

```
SELECT XMLPARSE(CONTENT '<name>Bernd Helmle</name><town>Mönchengladbach</town>');
```

```
-----
                xmlparse
```

```
-----
<name>Bernd Helmle</name><town>Mönchengladbach</town>
```

```
SELECT
    xmlroot(xmlagg(xmlelement(NAME id, xmlattributes(gender),
                id,
                xmlelement(NAME name, name))),
            VERSION '1.0',
            STANDALONE NO)
FROM foo;
```

```
-----
<?xml version="1.0" standalone="no"?><id gender="female">7<name>Jennifer</name></id><id gender="female">
```

cursor\_to\_xml()  
cursor\_to\_xmlschema()  
database\_to\_xml()  
database\_to\_xml\_and\_xmlschema()  
database\_to\_xmlschema()  
query\_to\_xml()  
query\_to\_xml\_and\_xmlschema()  
query\_to\_xmlschema()  
schema\_to\_xml()  
schema\_to\_xml\_and\_xmlschema()  
schema\_to\_xmlschema()  
table\_to\_xml()  
table\_to\_xml\_and\_xmlschema()  
table\_to\_xmlschema()  
xmlcomment()

# Volltextsuche

Integrierte Infrastruktur für Volltext-Indizierung (Text Search, ehemals tsearch2)

```
CREATE TABLE foo(id INTEGER, body TEXT);

INSERT INTO foo VALUES(1, 'The House Of The Rising Sun');

CREATE INDEX foo_txt_idx ON foo
  USING gin(to_tsvector('english', body));

SELECT * FROM foo
  WHERE body @@ to_tsquery('english', 'House | Sun');
```

# Table Functions

Gibt das Ergebnis einer Abfrage innerhalb einer pl/pgsql-Funktion zurück

```
CREATE TABLE foo(id SERIAL PRIMARY KEY, name TEXT NOT NULL);
INSERT INTO foo VALUES(DEFAULT, 'Bernd Helmle');
```

```
CREATE OR REPLACE FUNCTION f_get_foo(p_name IN TEXT)
RETURNS SETOF foo AS
$$
BEGIN
    RETURN QUERY
        SELECT * FROM foo WHERE name = p_name;
    RETURN;
END;
$$
LANGUAGE plpgsql STRICT STABLE;
```

```
SELECT * FROM f_get_foo('Bernd Helmle');
```

```
id |      name
----+-----
  1 | Bernd Helmle
```

# Updatable Cursor (1)

- Aktualisieren einer Tabelle an der Position eines darauf verweisenden Cursor
- Nur innerhalb derselben Transaktion
- Keine JOINS, keine Aggregate
- WHERE CURRENT OF nicht mit weiteren Bedingungen kombinierbar

# Updatable Cursor (2)

```

BEGIN;

DECLARE cur CURSOR FOR
    SELECT * FROM foo WHERE body @@ to_tsquery('english', 'House | Walk');

FETCH 1 FROM cur;
  id |          body
-----+-----
  1 | The House Of The Rising Sun

FETCH 1 FROM cur;
  id |          body
-----+-----
  1 | Egyptian Walk

UPDATE foo SET body = 'Walk Like An Egyptian' WHERE CURRENT OF cur;

SELECT * FROM foo;
  id |          body
-----+-----
  1 | The House Of The Rising Sun
  1 | Unchained Melody
  1 | Bitter Sweet Symphony
  1 | Dream A Little Dream
  1 | Walk Like An Egyptian

COMMIT;

```

# Neue Datentypen

- uuid: Global eindeutige Identifikationsnummer nach RFC 4122
- enum: Ein echter Aufzählungstyp

```
CREATE TYPE color AS enum('Red', 'Blue', 'Green');
```

- money: Überarbeitet für Locale-Support, 64-Bit
- Arrayunterstützung für benutzerdefinierte Typen

```
CREATE TYPE address_type AS (v1 text, v2 text, v3 text);  
INSERT INTO addresses  
VALUES(ARRAY[('a', 'b', 'c')::address_type,  
            ('b', 'e', 'f')::address_type]);
```

# Zu guter Letzt....

- Plan-Invalidierung
- Autovacuum mit konfigurierbaren “Workern”
- Bessere Unterstützung von Replikations-Lösungen
- Schnittstellen für erweiterbare Planer-Funktionen (Index Advisor, Algorithmen etc.)

# Synchronized Sequential Scans

- Synchronisierter Ablauf konkurrierender Scans von Tabellen
- Page wird idealerweise nur einmal in den Buffer Pool gelesen
- Prozesse “teilen” sich Leseaufwand
- Reduziert I/O

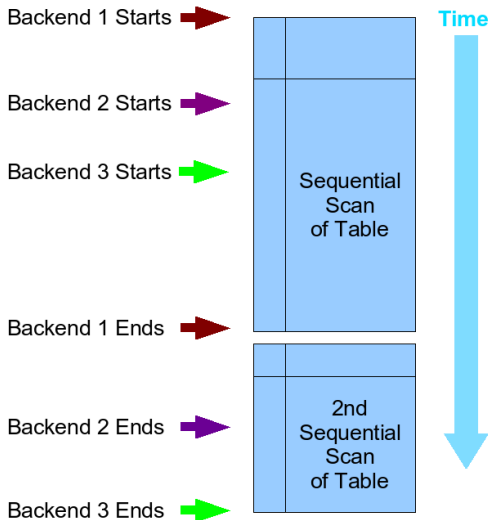


Abbildung: Synchronized Sequential Scans (by Josh Berkus, <http://www.powerpostgresql.com>)

# HOT-Updates

- “Update-In-Place” Verfahren bei UPDATE-Operationen
- Noch recht umstritten, da relativ komplex

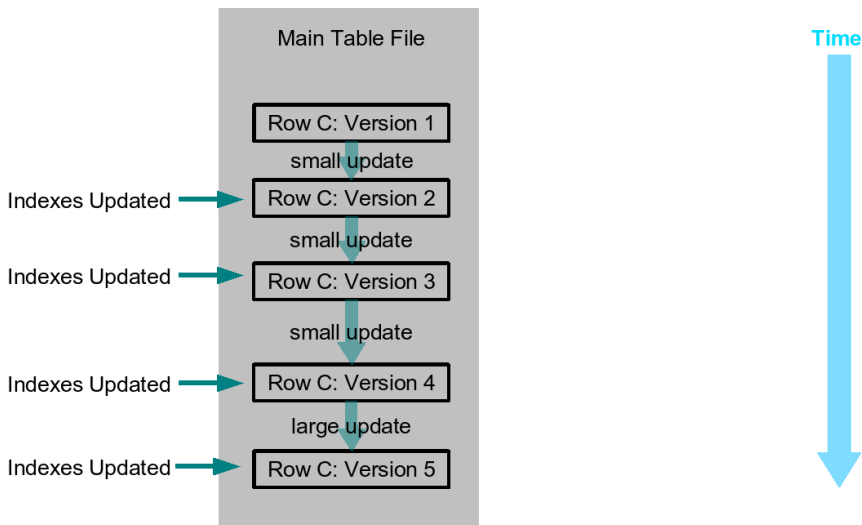


Abbildung: Herkömmliches Update Verfahren (by Josh Berkus, <http://www.powerpostgresql.com>)

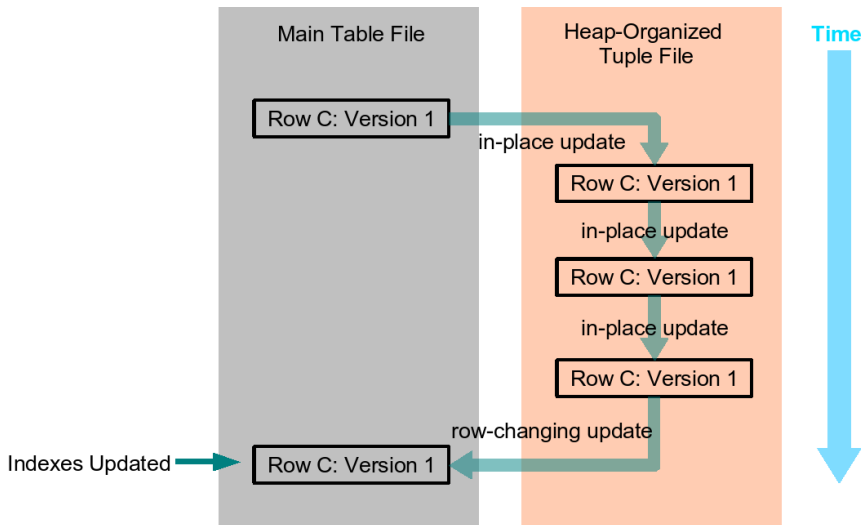


Abbildung: HOT Update Verfahren (by Josh Berkus, <http://www.powerpostgresql.com>)

# Kostenparameter für Stored Procedures

- COST Konstante
- ROWS Zeilenanzahl

```
CREATE OR REPLACE FUNCTION f_get_foo(p_name IN TEXT)
RETURNS SETOF foo AS
$$
BEGIN
    RETURN QUERY
        SELECT * FROM foo WHERE name = p_name;
    RETURN; END;
$$
LANGUAGE plpgsql STRICT STABLE COST 1.0 ROWS 100;
```

# Weitere Optimierungen

- Verbesserte Sortieralgorithmen (ORDER BY ... LIMIT, Merge Joins)
- Weitere Reduzierungen von WAL-Daten (temporäre Tabellen, COPY)
- Asynchrones COMMIT
- Distributed Checkpoints
- Schlankere Speicherstrukturen für Felder variabler Länge (TEXT, kombinierte CMIN/CMAX)

# Neues in contrib

- `pg_standby`: Hilfsprogramm für Warm-Standby-Setups
- `pageinspect`: Analysieren von Datenbankseiten
- `uuid-ossdp`: Hilfsprogramme für das Erzeugen von `uuid`-Werten

PQfinish(lecture);