Dipl. Inf. Eric Winter

Entwicklungsleiter
PTC GPS-Services GmbH

PostgreSQL als Huge Data Storage

Ein Erfahrungsbericht

PTC GPS
Planning · Tracking · Controlling

Inhalt

1. **Problembeschreibung**
2. **Partielle Indexierung**
3. **Partitionierung**
    1. Vererbung in PostgreSQL
    2. Partitionierung in PostgreSQL
    3. Probleme durch Partitionierung
4. [**Asynchrone Datenverarbeitung mit `NOTIFY` und `LISTEN`**]

1. Problembeschreibung

PTC GPS-Services GmbH

= Anbieter von Lösungen im Bereich Fahrzeugortung

- **Echtzeitortung**
  - Täglich 4.000.000 Wegpunkte (ca. 650 MB)
  - müssen nicht nur aufgezeichnet sondern auch validiert und verarbeitet werden
- **Automatisiertes Schreiben von Fahrtenbüchern**
  - Hoher Aufwand zur Korrektur und Nachverarbeitung
- **Live-Zugriff auf alle Daten**
  - Daten müssen vertragsgemäß 3 Jahre gehalten werden
  - Aktuell ca. 1.2 Mrd. Wegpunkte
  - Datenbankgröße: 640 GB (davon 30% Wegpunkte)

1. Problembeschreibung

Struktur der Wegpunkt Tabelle

```
CREATE TABLE waypoints (
  id bigserial PRIMARY KEY,
  box_id integer,
  lat double precision,
  lng double precision,
  status integer,
  logged_at timestamp,
  ...
);
```

Indizes

```
CREATE INDEX ON waypoints (box_id);
CREATE INDEX ON waypoints (status);
CREATE INDEX ON waypoints (logged_at);
```

1. Problembeschreibung

Problem:

- Indizes wachsen proportional zur Tabelle.
- Indexsuche zwar schneller als sequenzieller Scan der Tabelle ABER ab bestimmter Datenmenge auch langsam (1,2 Mrd Datensätze!?).

2. Partielle Indexierung

= Begrenzung eines Index auf best. Wertebereich.

- Kann Anfragezeiten erheblich reduzieren, wenn
  - viele Anfragen nur diesen Wertebereich erfordern.
  - Wertebereich möglichst selten vorkommt.

```
SELECT * FROM waypoints
WHERE status = 5 OR status = 15;
```

- Indizierung über `status = 5` und `status = 15`

```
CREATE INDEX ON waypoints (box_id)
WHERE status = 5 OR status = 15;
```

- Indizierung über `logged_at`

```
CREATE INDEX ON waypoints (logged_at)
WHERE logged_at BETWEEN '2013-01-01' AND '2013-12-31';
```

3. Partitionierung

= Spezielle Form von Vererbung

Vererbung in PostgreSQL

```
CREATE TABLE cities (
  name text,
  population float,
  altitude int
);

CREATE TABLE capitals (
  state char(2)
) INHERITS (cities);
```

Quelle: PostgreSQL 9.1 Dokumentation

3. Partitionierung

= Spezielle Form von Vererbung

Vererbung in PostgreSQL

- Selektion über Vererbungshierarchie

```
SELECT name, altitude
FROM cities
WHERE altitude > 500;
```

Quelle: PostgreSQL 9.1 Dokumentation

- Selektion nur (**ONLY**) aus bestimmter Tabelle

```
SELECT name, altitude
FROM ONLY cities
WHERE altitude > 500;
```

Quelle: PostgreSQL 9.1 Dokumentation

3. Partitionierung

= Spezielle Form von Vererbung

Vererbung in PostgreSQL

- Einfügen (**`INSERT`**, **`COPY`**) neuer Datensätze

```
INSERT INTO cities (name, population, altitude, state)
VALUES ('New York', NULL, NULL, 'NY');
```

Quelle: PostgreSQL 9.1 Dokumentation

```
INSERT INTO capitals (name, population, altitude, state)
VALUES ('New York', NULL, NULL, 'NY');
```

3. Partitionierung

```
CREATE TABLE waypoints (
  id bigserial PRIMARY KEY,
  box_id integer,
  lat double precision,
  lng double precision,
  status integer,
  logged_at timestamp,
  ...
);
```

```
CREATE TABLE waypoints_2013_02_04 (
  CHECK (
    logged_at >= '2013-02-04'::timestamp AND
    logged_at < '2013-02-05'::timestamp
  )
) INHERITS waypoints;
```

```
CREATE TABLE waypoints_2013_02_05 ...
```

```
...
```

3. Partitionierung

```
CREATE TABLE waypoints (
  id bigserial PRIMARY KEY,
  box_id integer,
  lat double precision,
  lng double precision,
  status integer,
  logged_at timestamp,
  ...
);
```

```
CREATE TABLE waypoints_2013_02_04 (
  PRIMARY KEY id,
  CHECK (
    logged_at >= '2013-02-04'::timestamp AND
    logged_at < '2013-02-05'::timestamp
  )
) INHERITS waypoints;
```

3. Partitionierung

Vorteile:

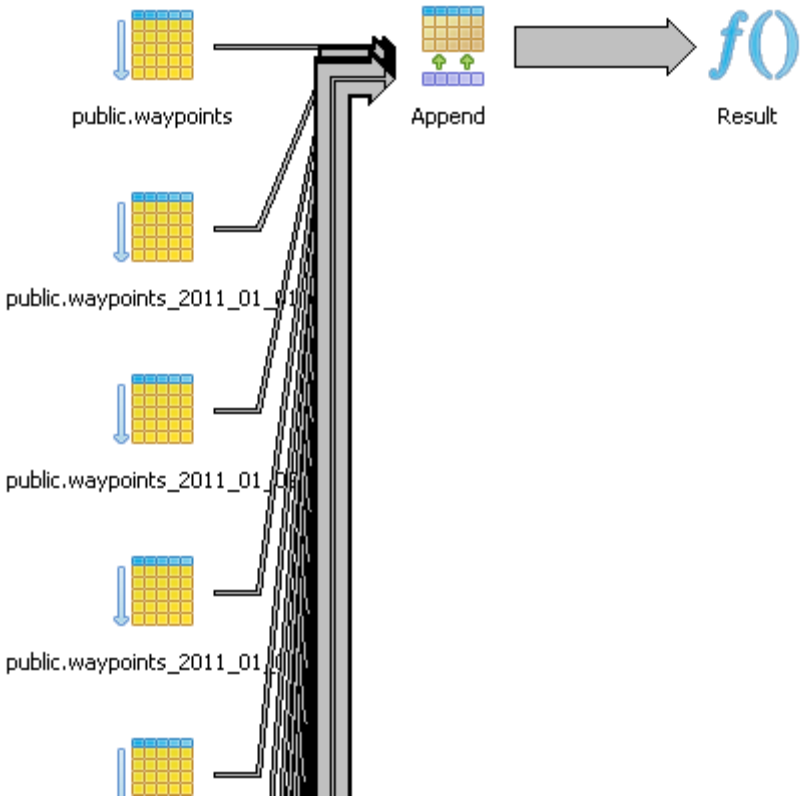
- Einzeltabellen enthalten nur noch max. 4.000.000 Datensätze (650MB)
  → Passen wieder komplett in Speicher.
- Wenn angefragte Daten sich nur auf best. Zeitintervall beziehen, können betroffene Partitionen dauerhaft im Speicher gehalten werden.
- Partitionen können einzeln indiziert werden.

3. Partitionierung

Nachteile:

- Anfragen müssen immer Selektion über partitioniertes Feld enthalten.

```
SELECT * FROM waypoints;
```

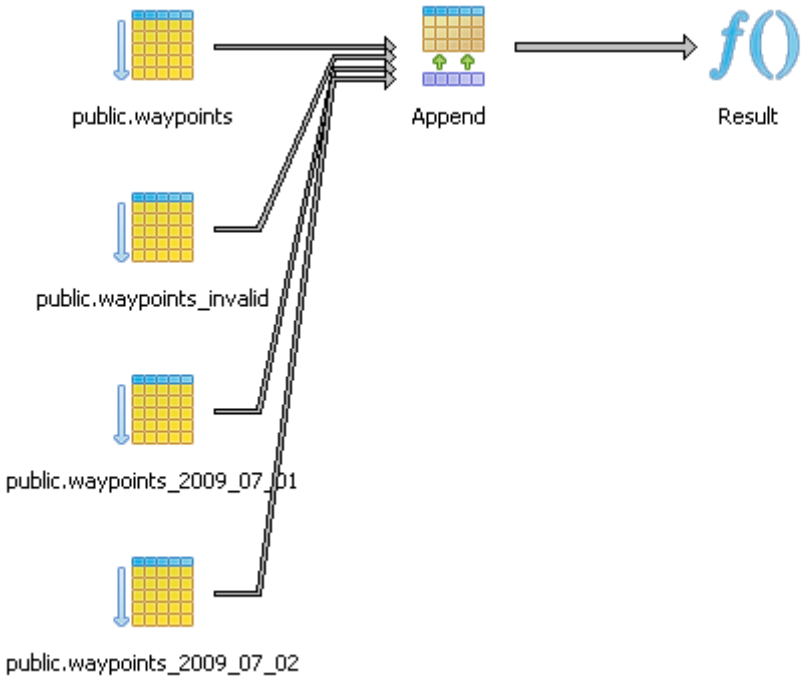


3. Partitionierung

Nachteile:

- Anfragen müssen immer Selektion über partitioniertes Feld enthalten.

```
SELECT * FROM waypoints WHERE logged_at < '2009-07-03';
```



3. Partitionierung

Nachteile:

- Einfügen neuer Datensätze erschwert, da immer zuerst die entsprechende Tabelle ermittelt werden muss.

→ **Lösung mittels Trigger**

```
CREATE TRIGGER trigger_waypoints_insert
BEFORE INSERT ON waypoints
FOR EACH ROW EXECUTE PROCEDURE waypoints_insert_trigger();
```

3. Partitionierung

```
CREATE FUNCTION waypoints_insert_trigger()
  RETURNS trigger AS $BODY$
DECLARE
  tablename text;
BEGIN
  tablename := 'waypoints_' || to_char(NEW.logged_at, 'yyyy_mm_dd');

  IF (NOT (SELECT EXISTS (
    SELECT 1 FROM pg_class WHERE relname = tablename))
  ) THEN
    tablename := 'waypoints_invalid';
  END IF;

  EXECUTE
    'INSERT INTO ' || tablename || ' (id, box_id, lat, lng, logged_at)' ||
    'VALUES ($1, $2, $3, $4, $5)'
  USING
    NEW.id, NEW.box_id, NEW.lat, NEW.lng, NEW.logged_at;

  RETURN NULL;
END;
$BODY$ LANGUAGE plpgsql;
```

4. Asynchrone Datenverarbeitung

NOTIFY und LISTEN

- Ermöglichen asynchrone Benachrichtigung über benutzerdefinierte Ereignisse.
- **NOTIFY** (**pg_notify**) bereits in Version 7 verfügbar
  → Jedoch konnten keine Daten mitgesendet werden.
- Seit Version 9.0:

```
NOTIFY channel [ , payload ];
```

- Bsp.:

```
LISTEN foo;
NOTIFY foo, 'Some important information';
>> Asynchronous notification "foo" with payload "Some
important information" received from server process with
PID 8448.
```

4. Asynchrone Datenverarbeitung

Was bringt das im Bezug auf Problemstellung?

- Daten müssen in „Echtzeit“ dem Kunden angezeigt werden
  - Traditionell wird gepollt. (schlecht)
- Durch Notifications ist es möglich, über Veränderungen benachrichtigt zu werden.
  - Daten werden nur dann übertragen, wenn sich wirklich etwas geändert hat.
  - Es können nur die Daten übertragen werden, die sich geändert haben.

→ **Enorme Reduzierung von Anfragen und damit Last auf der Datenbank.**

4. Asynchrone Datenverarbeitung

```
CREATE FUNCTION waypoints_insert_trigger()
...
  PERFORM pg_notify(
    TG_TABLE_SCHEMA || '.' || TG_TABLE_NAME,
    '{' ||
      'action:"' || TG_OP || '",' ||
      'context:"' || TG_WHEN || '",' ||
      'data:{' ||
        format(
          'id:%s, box_id:%s, lat:%s, lng:%s, logged_at:"%s"',
          NEW.id, NEW.box_id, NEW.lat, NEW.lng, NEW.logged_at
        ) ||
      '}' ||
    '}'
  );

  RETURN NULL;
END;
$BODY$ LANGUAGE plpgsql;
```

```
LISTEN "public.waypoints";
>> Asynchronous notification "public.waypoints" with
payload "{action:..., context:..., data:...}" received ...
```

Vielen Dank für Ihre Aufmerksamkeit