Willkommen zum „IBM Informix Newsletter“

Inhaltsverzeichnis

Aktuelles

Liebe Leserinnen und Leser,

Frühlingsgefühle überall, die Natur blüht auf und auch INFORMIX startet mit CeBIT, Infobahn, IUG, … mit Vollgas in die Saison. Zudem ist seit wenigen Tagen die neue Version 12.10.FC5 verfügbar, die wieder einige neue Features und Bereinigungen mit sich bringt (wozu wir in der kommenden Ausgabe mehr berichten werden). Den Anfang macht diesmal die Nutzung von „derived tables“, was immer wieder sehr hilfreich sein kann bei komplexeren SQLs.

Wie immer haben wir für Sie eine Reihe an Tipps und Tricks zusammengestellt.
Viel Spaß mit den Tipps der aktuellen Ausgabe.

Ihr TechTeam

TechTipp: Derived Tables mit Spaltennamen

Die Ergebnisse von Funktionen und Prozeduren können bei Abfragen wiederum wie eine Tabelle angesprochen werden, diese mittels TABLE() als Tabelle deklariert wird. Gibt man zudem noch Namen an wie z.B. „... as h(f1,f2)“, so lassen sich die Rückgabewerte der Funktion wie Einträge in der Tabelle „h“ mit den Spalten „f1“ und „f2“ nutzen.
Beispiel: Eine Tabelle, die die Werte von 1 bis zur eingegebenen Zahl zurück gibt

```
drop procedure if exists generate_number(int);
create procedure generate_number(x_in int)
returning int as x_ret, char(12) as y_ret
define i int;
define t char(12);
let i=1;
while i <= x_in
let t = "Wert_"||i;
return i,t with resume;
let i = i+1;
end while
end procedure;
```

Ein kurzer Test der Funktion:

```
execute procedure generate_number(4);
```

Die Funktion kann nun im SQL genutzt werden:

```
select * from TABLE(generate_number(4)) as h(f1,f2);
```

Die Funktion kann als Tabelle im Join angesprochen werden:

```
select t.tabid, t.tabname[1,18], f1, f2
from systables t, TABLE(generate_number(8)) as h(f1,f2)
where t.tabid = f1
```

Ergebnis:

```
 tabid tabname                        f1 f2

     1 systables                       1 Wert_1
     2 syscolumns                      2 Wert_2
     3 sysindices                      3 Wert_3
     4 systabauth                      4 Wert_4
     5 syscolauth                      5 Wert_5
     6 sysviews                        6 Wert_6
     7 sysusers                        7 Wert_7
     8 sysdepend                       8 Wert_8
```

TechTipp: Informix Warehouse Accelerator: Trickle Feed und NONEXCLTRIG

Seit der Version 12.10.FC1 des Informix Servers und IWA gibt es die TrickleFeed-Funktionalität, mit der in Fakttabellen eingefügte Daten fortlaufend in einen bestehenden Data Mart eingepflegt werden. TrickleFeed basiert auf Triggers: INSERT Triggers werden benutzt, um die in Fakttabellen eingefügten Daten zu sammeln und sie dann in den entsprechenden Data Mart zu übertragen. Diese Methode verursacht geringe zusätzliche Belastung und funktioniert individuell für die betroffenen Fakttabellen. Wenn TrickleFeed für einen Data Mart aktiviert oder deaktiviert wird, dann werden die entsprechenden INSERT Triggers angelegt oder entfernt. Typische Situationen, die das Aktivieren oder Deaktivieren von TrickleFeed notwendig machen, sind z.B. RefreshMart oder das Neuladen der Daten - zur regelmässigen Ausführung empfohlene Operationen.

Nachteil dieser auf Triggers basierten Methode ist, dass das Anlegen oder Entfernen von Triggers Datendefinitionsoperationen sind, für die exklusiver Zugriff auf die jeweils betroffene Tabelle benötigt wird. Für den Informix Server bedeutet exklusiver Tabellenzugriff jedoch, dass zur selben Zeit keine andere Datenbanksitzung auf die Tabelle zugreifen darf, was sogar Dirty Read Lesezugriffe oder auch nur das Warten auf eine Sperre der Tabelle einschliesst. In der Praxis kann es daher schwierig sein, diesen exklusiven Tabellenzugriff zu erhalten, weil selbst eine exklusive Sperre auf die Tabelle nicht verhindert, dass eine andere Datenbanksitzung auf eine Sperre der Tabelle wartet oder Dirty Read durchführt. Das Aktivieren oder Deaktivieren von TrickleFeed kann fehlschlagen, weil der benötigte exklusive Tabellenzugriff nicht gegeben ist.

Eine mögliche Lösung für das Problem ist die Benutzung des undokumentierten NONEXCLTRIG. NONEXCLTRIG ist sowohl ein Konfigurationsparameter in der "onconfig"-Datei des Informix Servers, als auch eine Umgebungsvariable für die Verbindung eines Client zum Informix Server. NONEXCLTRIG wird folgendermassen benutzt:

> Setzen des Konfigurationsparameters **NONEXCLTRIG** in der "onconfig"-Datei des Informix Servers auf einen der folgenden drei Werte:
>
> > Wert 0:
> > Voreinstellung, Verhalten wie wenn der Parameter nicht vorhanden ist. Anlegen und Entfernen eines Triggers benötigt immer exklusiven Tabellenzugriff.

Wert 1:
Ob für das Anlegen oder Entfernen von Triggern exklusiver Tabellenzugriff benötigt wird, hängt von der Umgebungsvariablen NONEXCLTRIG der jeweiligen Client-Verbindung zum Datenbankserver ab.

Wert 2:
Anlegen oder Entfernen von Triggern benötigt keinen exklusiven Tabellenzugriff, unabhängig von der Umgebung einer Client-Verbindung zum Datenbankserver.

Damit die neue Konfiguration wirksam wird, ist ein Neustart der Informix Serverinstanz notwendig.

Wenn der Konfigurationsparameter NONEXCLTRIG auf den Wert 1 gesetzt ist, dann ist die Umgebungsvariable NONEXCLTRIG für den Client zu setzen, bevor die Verbindung zum Datenbankserver etabliert wird, mit der Triggern angelegt oder entfernt werden sollen. Zum Beispiel in der Bourne- oder Korn-Shell mit:

```
NONEXCLTRIG=1
export NONEXCLTRIG
```

Es ist wichtig, die Auswirkungen der Benutzung von NONEXCLTRIG zu verstehen:

Unterstützt ist:
INSERT CURSORS laden die modifizierte Triggerdefinition mit der nächsten PUT Operation.
UPDATE / DELETE WHERE CURRENT laden die Triggerdefinition vor dem EXECUTE.

Einschränkung:
EXECUTE anderer Befehle, deren PREPARE vor der Änderung des Triggers stattfand, erzeugen einen Fehler.
SELECT CURSORS laden die Triggerdefinition erst dann neu, wenn sie geschlossen und wieder geöffnet werden.

Der Informix Server versucht nicht zu prüfen (z.B. durch Aufspüren von Sperren der Transaktion), ob die laufende Transaktion die Tabelle schon verändert hat wenn die nächste DML Operation eine Änderung der Tabellendefinition feststellt. Somit können zwei Operationen innerhalb derselben Transaktion zwei verschiedene Triggerdefinitionen derselben Tabelle haben.

Zur Vermeidung unerwünschter Effekte der oben beschriebenen Einschränkungen empfehlen wir folgende Einstellungen, wenn die Benutzung von NONEXCLTRIG notwendig ist:

- Den Konfigurationsparameter NONEXCLTRIG in der "onconfig"-Datei auf den Wert 1 setzen.
- Die Umgebungsvariable NONEXCLTRIG nur für die Datenbankverbindung setzen, die TrickleFeed aktiviert oder deaktiviert.

Zur Wahrung der Datenkonsistenz im Data Mart und zur Vermeidung von Fehlern aufgrund geänderter Triggerdefinitionen empfehlen wir zudem:

- Die periodische Operation RefreshMart und das anschliessende Wiederaktivieren von TrickleFeed immer in derselben Datenbanksitzung ausführen,
- und in dieser Datenbanksitzung alle Tabellen, die am Data Mart beteiligt sind, sperren (mindestens im "share mode") bevor der Refresh Mart gestartet wird. Diese Sperren erst dann wieder freigeben, nachdem TrickleFeed erneut aktiviert ist.

Damit wird vermieden, dass im Data Mart Daten fehlen, die zwischen dem Beginn eines Refresh Mart und der nachfolgenden Wiederaktivierung von TrickleFeed in Fakttabellen eingefügt werden.

TechTipp: Sensordaten mit Informix erfassen - Beispiel (Teil 3)

Im Teil 3 erweitern wir die Sensordaten um den Datentyp BSON.
Zuerst erstellen wir wieder den Row Type für die TimeSeries. Diesmal mit dem Typ „BSON“ für die Werte:

```
create row type sensor_json_t(
     timestamp datetime year to fraction(5),
     values    bson
);
```

Analog dem Beispiel aus Teil 1 erstellen wir einen Container, die TimeSeries Tabelle und eine virtuelle Tabelle:

```
execute procedure TSContainerCreate (
        'sensor_json_cont',
        'rootdbs',
        'sensor_json_t',
        2048,
        2048);

create table sensor_json_ts (
        sensor_id char(40),
        sensor_type char(20),
        sensor_unit char(6),
        sensor_values timeseries(sensor_json_t),
        primary key (sensor_id)
) lock mode row;

execute procedure TSCreateVirtualTab (
        'sensor_json_data',
        'sensor_json_ts',
        'origin(2015-01-26
00:00:00.00000),calendar(ts_1min),container(sensor_json_cont)
,threshold(0),regular',0,'sensor_values');
```

Sobald die virtuelle Tabelle erstellt ist, können wir Werte einfügen:

```
insert into sensor_json_data values ("Sensor03", "Temp", "C",
     "2015-01-26 08:00"::datetime year to minute,
     '{ "temp1":21.5 }'::json);
insert into sensor_json_data values ("Sensor03", "Temp", "C",
     "2015-01-26 08:01"::datetime year to minute,
     '{ "temp1":23.1 }'::json);
insert into sensor_json_data values ("Sensor03", "Temp", "C",
```

```
        "2015-01-27 10:45"::datetime year to minute,
        '{ "temp1":22.8 }'::json);
    insert into sensor_json_data values ("Sensor04", "Temp", "C",
        "2015-01-26 12:02"::datetime year to minute,
        '{ "temp1":30.2, "temp2":10.3}'::json);
    insert into sensor_json_data values ("Sensor04", "Temp", "C",
        "2015-01-27 15:46"::datetime year to minute,
        '{ "temp1":34.0, "temp2":9.7}'::json);
    insert into sensor_json_data values ("Sensor04", "Temp", "C",
        "2015-01-27 15:47"::datetime year to minute,
        '{ "temp1":33.9, "temp2":9.2}'::json);
```

Die soeben eingefügten Werte lassen sich mittels SQL abfragen:

```
select sensor_id, sensor_type, sensor_unit, timestamp,
values::json values from sensor_json_data;
```

Das Ergebnis:

```
sensor_id    Sensor03
sensor_type  Temp
sensor_unit  C
timestamp    2015-01-26 08:00:00.00000
values       {"temp1":21.500000}

sensor_id    Sensor03
sensor_type  Temp
sensor_unit  C
timestamp    2015-01-26 08:01:00.00000
values       {"temp1":23.100000}

sensor_id    Sensor03
sensor_type  Temp
sensor_unit  C
timestamp    2015-01-27 10:45:00.00000
values       {"temp1":22.800000}

sensor_id    Sensor04
sensor_type  Temp
sensor_unit  C
timestamp    2015-01-26 12:02:00.00000
values       {"temp1":30.200000,"temp2":10.300000}
```

```
sensor_id    Sensor04
sensor_type  Temp
sensor_unit  C
timestamp    2015-01-27 15:46:00.00000
values       {"temp1":34.000000,"temp2":9.700000}

sensor_id    Sensor04
sensor_type  Temp
sensor_unit  C
timestamp    2015-01-27 15:47:00.00000
values       {"temp1":33.900000,"temp2":9.200000}
```

Obwohl die Werte im JSON-Format vorliegen, können die Funktionen der TimeSeries genutzt werden um z.B. die Summe je Stunde zu bilden:

```
SELECT sensor.timestamp::datetime year to hour timestamp,
sensor.value temp1 FROM TABLE (
TRANSPOSE ((
SELECT AggregateBy(
        'SUM($temp1)',
        'ts_1hour',
        sensor_values,
        0,
        '2015-01-26 00:00'::datetime year to minute,
        '2015-01-31 23:59'::datetime year to
minute)::TimeSeries(sensor_t)
FROM sensor_json_ts
WHERE sensor_id = "Sensor03"
))
) AS TAB (sensor);
```

Ergebnis:

```
timestamp              temp1
2015-01-26 08          44.60
2015-01-27 10          22.80
```

Im Beispiel wurden nun nur die Werte zu temp1 des Sensors 3 abgefragt. Sollen die Werte zu weiteren Einträgen wie z.B. temp2 mit berücksichtigt werden, so reicht es hierzu einen entsprechenden Row Type anzulegen:

```
create row type sensor_twovals_t
(
        timestamp       datetime year to fraction(5),
        value1           decimal(15,2),
        value2           decimal(15,2)
);
```

Dieser neue Row Type kann sofort in der Abfrage, hier die Werte von Sensor , eingebunden werden:

```
SELECT sensor.timestamp::datetime year to hour timestamp,
     sensor.value1 temp1, sensor.value2 temp2 FROM TABLE (
TRANSPOSE ((
     SELECT AggregateBy(
          'SUM($temp1),SUM($temp2)',
          'ts_1hour',
          sensor_values,
          0,
          '2015-01-26 00:00'::datetime year to minute,
          '2015-01-31 23:59'::datetime year to
     minute)::TimeSeries(sensor_twovals_t)
     FROM sensor_json_ts
     WHERE sensor_id = "Sensor04"
     ))
) AS TAB (sensor);
```

Mit dem Ergebnis:

```
timestamp               temp1           temp2
2015-01-26 12           30.20           10.30
2015-01-27 15           67.90           18.90
```

TechTipp: Sensordaten mit Informix erfassen - Beispiel (Teil 4)

Im Teil 4 wollen wir nun Sensordaten kontinuierlich auslesen und in die Datenbank einfügen.

Die Daten lesen wir in regelmässigen Abständen aus einer Datei aus. Hierzu erstellen wir eine Externe Tabelle, die auf die auszulesende Datei verweist. Im Beispiel ist das die interne Temperatur eines RaspberryPi, die ständig in der Datei

```
/sys/class/thermal/thermal_zone0/temp
```

aktualisiert wird. Wer dies z.B. auf Linux nachvollziehen will, der kann sich mittels „sensors“ die Temperatur der CPU formatiert ausgeben lassen:

```
sensors | grep temp1 | cut -d'+' -f2 | cut -d'C' -f 1 | tr -d
'°' |  awk '{printf "%06f\n", $0;}' | cut -c1-6 >
/tmp/sensor_temp
```

Das Beispiel für den RaspberryPi:

```
create external table rpi_temp
(
        temp1 integer external char(6)
)
using
(
        FORMAT "FIXED",
        DATAFILES
        (
                "DISK:/sys/class/thermal/thermal_zone0/temp"
        )
);
```

Für das Beispiel auf Linux sind die Datentypen der Externen Tabelle:

temp1 varchar(7) external char(7)

Ist die externe Tabelle erstellt, so können aus ihr die Werte gelesen werden:

```
insert into sensor_data
select "SensorRPiTemp", "Temperature", "C",
current::datetime year to minute,
(temp1)::DECIMAL(15,2) from rpi_temp;
```

Da dies für fortlaufende Werte jede Minute erfolgen sollte, erstellen wir einen Task in der Datenbank sysadmin, der uns die Werte ausliest:

```
database sysadmin;
INSERT INTO ph_task
(
        tk_name,
        tk_description,
        tk_type,
        tk_group,
        tk_execute,
        tk_start_time,
        tk_stop_time,
        tk_frequency,
        tk_dbs
)
VALUES
(
        "Read Out RPi Temp",
        "Reads out the RPi internal temp every 60 secs",
        "TASK",
        "MISC",
        "insert into sensor_db:sensor_data select
          'SensorRPiTemp', 'Temperature', 'C',
           current::datetime year to minute,
          (temp1)::DECIMAL(15,2)
          from sensor_db:rpi_temp",
        NULL,
        NULL,
        interval(1) minute to minute,
        "sensor_db"
)
```

Da Tasks nur zum Start des Schedulers eingelesen werden, muss dieser anschliessend neu gestartet werden:

```
database sysadmin;
EXECUTE FUNCTION task('scheduler shutdown');
EXECUTE FUNCTION task('scheduler start');
```

Zum Test können wir die nunmehr automatisch eingelesenen Werte ausgeben:

```
database sensor_db;
select first 5 * from sensor_data where sensor_id =
'SensorRPiTemp' order by timestamp desc
```

Um drei Werte für Min, Max und Avg auszugeben, erstellen wir einen RowType mit diesen Datentypen:

```
create row type sensor_threevals_t
(
        timestamp       datetime year to fraction(5),
        value1           decimal(15,2),
        value2           decimal(15,2),
        value3           decimal(15,2)
);
```

Nun kann über einen Zeitraum die minimale, maximale und durchschnittliche Temperatur ausgegeben werden:

```
SELECT sensor.timestamp::datetime year to hour timestamp,
        sensor.value1 min, sensor.value2 max, sensor.value3
avg
FROM TABLE (
TRANSPOSE ((
SELECT AggregateBy(
        'min($value), max($value), avg($value)',
        'ts_1hour',
        sensor_values,
        0,
        '2015-01-01 00:00'::datetime year to minute,
        '2015-12-31 23:59'::datetime year to
minute)::TimeSeries(sensor_threevals_t)
FROM sensor_ts
WHERE sensor_id = "SensorRPiTemp"
))
) AS TAB (sensor);
```

Ergebnis:

```
timestamp       min       max       avg
2015-02-04 16   46.00     50.00     47.82
2015-02-04 17   45.00     64.00     47.70
2015-02-04 18   46.00     46.00     46.00
```

Mit anderen Kalendern wie z.B. ts_1day, ts_1week, ts_1month, … kann hier das Intervall verändert werden.

TimeSeries bieten zudem die Möglichkeit, Virtuelle Tabellen anhand von Funktionen zu erstellen. So kann z.B. die Ausgabe der Stundenwerte auch direkt als Virtuelle Tabelle zur Verfügung gestellt werden:

```
EXECUTE PROCEDURE TSCreateExpressionVirtualTab(
       'sensor_data_min_max_avg', 'sensor_ts',
       'AggregateBy("min($value),max($value),avg($value)",
"ts_1hour", sensor_values, 0, $ts_begin_time, $ts_end_time)',
       'sensor_threevals_t', 0, 'sensor_values');

select sensor_id, sensor_type, sensor_unit,
        timestamp::datetime year to hour timestamp,
        value1 min, value2 max, value3 avg
from sensor_data_min_max_avg
where sensor_id = "SensorRPiTemp"
```

Als Vorlage für diesen Artikel diente der Blog-Beitrag von Alexander Körner im RaspberryPi-Forum.

Anmeldung / Abmeldung / Anmerkung

Der Newsletter wird ausschließlich an angemeldete Adressen verschickt. Die Anmeldung erfolgt, indem Sie eine Email mit dem Betreff „**ANMELDUNG**“ an **ifmxnews@de.ibm.com** senden.
Im Falle einer Abmeldung senden Sie „ABMELDUNG“ an diese Adresse.

Das Archiv der bisherigen Ausgaben finden Sie zum Beispiel unter:

http://www.iiug.org/intl/deu
http://www.iug.de/index.php?option=com_content&task=view&id=95&Itemid=149
http://www.informix-zone.com/informix-german-newsletter
http://www.drap.de/link/informix
http://www.nsi.de/informix/newsletter
http://www.cursor-distribution.de/index.php/aktuelles/informix-newsletter
http://www.listec.de/Newsletter/IBM-Informix-Newsletter/View-category.html
http://www.bereos.eu/software/informix/newsletter/

Die hier veröffentlichten Tipps&Tricks erheben keinen Anspruch auf Vollständigkeit. Da uns weder Tippfehler noch Irrtümer fremd sind, bitten wir hier um Nachsicht falls sich bei der Recherche einmal etwas eingeschlichen hat, was nicht wie beschrieben funktioniert.

Die Autoren dieser Ausgabe

Gerd Kaluzinski IT-Specialist Informix Dynamic Server und DB2 UDB
IBM Software Group, Information Management
gerd.kaluzinski@de.ibm.com +49-175-228-1983

Martin Fuerderer IBM Informix Entwicklung, München
IBM Software Group, Information Management
martinfu@de.ibm.com

Markus Holzbauer IBM Informix Advanced Support
IBM Software Group, Information Management Support
holzbauer@de.ibm.com

Die Versionsinfo stammt aus dem Versions-Newsletter der CURSOR Software AG
http://www.cursor-distribution.de/download/informix-vinfo

Sowie unterstützende Teams im Hintergrund.

Fotonachweis: Gerd Kaluzinski (Frühling im Redaktionsgarten)