УДК 004.652.4

ОРГАНИЗАЦИЯ ХРАНЕНИЯ ХРОНОЛОГИЧЕСКИХ ДАННЫХ В БАЗАХ ДАННЫХ СИСТЕМ МОНИТОРИНГА И ПРОГНОЗИРОВАНИЯ

Фишер Антон Владиславович
аспирант
ООО «СофтПро», Краснодар, Россия

Дьяченко Роман Александрович
к.т.н
Кубанский государственный технологический университет, Краснодар, Россия

Лоба Инна Сергеевна
аспирант
Армавирский государственный педагогический университет, Краснодар, Россия

В статье рассматриваются различные способы хранения хронологических данных. Построены математические модели каждого из способов. В качестве одного из решения приводится методика организации партицирования данных на конкретной СУБД

Ключевые слова: БАЗА ДАННЫХ, НАСЛЕДОВАНИЕ, ПАРТИЦИРОВАНИЕ, СУБД, ТРИГГЕР, ХРОНОЛОГИЧЕСКИЕ ДАННЫЕ

UDC 004.652.4

STORAGE ORGANIZATION OF CHRONOLOGICAL DATA IN DATABASES OF MONITORING AND FORECASTING SYSTEMS

Fisher Anton Vladislavovich
graduate student
SoftPro, Krasnodar, Russia

Dyachenko Roman Aleksandrovich
Cand.Tech.Sci.
Kuban State Technological University, Krasnodar, Russia

Loba Inna Sergeevna
graduate student
Armavir State Pedagogical University, Krasnodar, Russia

The article discusses various ways to store chronological data. The mathematical models were constructed for each method. As one of the solutions is the method of data of organization partitioning of a particular DBMS

Keywords: DATABASE, INHERITANCE, PARTITIONING, DBMS, TRIGGER, CHRONOLOGICAL DATA

1. Постановка задачи

В процессе проектирования информационных систем сбора информации, мониторинга и прогнозирования возникает проблема организации хранения хронологических данных, где показания, полученные с датчиков необходимо сохранить, дополнив информацией о времени их поступления. В большинстве случаев для решения указанной проблемы используются различные приемы при проектировании базы данных.

С течением времени могут возникнуть проблемы снижения быстродействия информационной системы по причине большого количества хронологических данных (например, может увеличиться время выполнения запросов, использующих сортировку). По этой причине задача рациональной организации хранения хронологических данных с

обеспечением допустимого уровня быстродействия информационной системы является актуальной. Целью данной работы является поиск наилучшего способа хранения хронологических данных среди типизированных решений.

2. Варианты решения

Для формализации поставленной задачи далее будем считать, что данные поступающие в систему мониторинга и прогнозирования представляют набор пар “значение - дата создания”.

Возможными решениями задачи хранения хронологической информации являются:

- Использование одной таблицы (вся информация хранится в одной таблице);
- Разбиение таблицы на архивные периоды с использованием дополнительного поля;

Для представления реляционной базы данных используется математическая модель.

Первый вариант решения имеет вид:

$$S_1=<A_1:D_1;A_2:D_2;A_3:D_3;>, \quad (1)$$

где A_1 - атрибут “уникальный номер”,

D_1 - множество целых чисел,

A_2 - атрибут “значение”,

D_2 - множество действительных чисел,

A_3 - атрибут “дата создания”,

D_3 - домен “дата-время”.

Второй вариант решения:

$$S_1=<A_1:D_1;A_2:D_2;A_3:D_3;A_4:D_4;>, \quad (2)$$

где A_1 - атрибут “уникальный номер”,

D_1 - множество целых чисел,

A_2 - атрибут “значение”,

D_2 - множество действительных чисел,

A_3 - атрибут “дата создания”,

D_3 - домен “дата-время”,

A_4 - атрибут “период”,

D_4 - множество целых чисел.

Как видно из описанных моделей с накоплением данных будет увеличиваться объем таблицы, следовательно, для их обработки потребуется больше ресурсов.

Во втором варианте дополнительно вводится поле “период”, что позволяет упростить запросы выборок наборов данных по определенному периоду, используя индекс по этому атрибуту. В качестве периода используется наиболее пригодный период для построения будущих запросов к базе.

Оба описанных варианта имеют общий недостаток - это растущий размер одной используемой таблицы. Что создает проблемы при работе с ней, кроме того это создает дополнительные проблемы при создании полных резервных копий.

Тривиальным способом хранения хронологических данных является использование одной таблицы. Но в большинстве случаев данные полученные от системы мониторинга обрабатываются определенными периодами, на основе чего возможно оптимизировать способ их хранения. В качестве одного из решений по организации и хранению хронологических данных можно предложить вариант использования

объектно-ориентированных возможностей современных баз данных - “разбиение” и “наследование” таблиц.

Разбиение таблиц можно реализовать с помощью использования встроенного механизма наследования и специальных триггеров.

Механизм наследования таблиц позволяет создавать копии таблицы-шаблона одинаковые по структуре. Таблицы-наследники могут иметь общий первичный ключ (в СУБД *Postgresql* реализуется с помощью последовательностей), что гарантирует совокупную уникальность записей хранимых в них. Наследование в СУБД позволяет выполнять запросы как к отдельной таблице-наследнику, при этом получая данные содержащиеся непосредственно в ней, так и к таблице-шаблону. При обращении к таблице-шаблону будут обработаны данные содержащиеся во всех таблицах-наследниках. Следовательно, если разделить таблицы таким образом, что для наиболее частых запросов выполняемых в вычислительной системе будет достаточным обратиться к одной таблице-наследнику, можно затратить меньшее количество ресурсов, следовательно, получит выигрыш производительности всей системы в целом.

Для организации распределения данных по таблицам-потомкам при вставке используются триггеры (в *Postgresql* триггерные функции). Все запросы вставки данных используют таблицу-шаблон, а триггерные функции в соответствии с принятым периодом разделения данных записывают данные в нужную таблицу-потомка. По истечению периода эти функции обновляются, меняя при этом правило записи. Модель имеет вид (1).

В качестве примера реализации рассмотрим случай простого разделения данных по временным периодам, где длина периода равна календарному месяцу.

3. Техническая реализация по этапам

3.1 Этап 1. Создание таблиц (шаблонов)

На первом этапе создается таблица-шаблон согласно модели 3 (реализация на *Postgresql* см. листинг 1):

```
CREATE TABLE "table_c"
(
    "id" bigserial NOT NULL,
    "value" double precision NOT NULL,
    "time" timestamp with time zone NOT NULL DEFAULT now(),
    CONSTRAINT "table_c_pkey" PRIMARY KEY ("id")
);
```

Листинг 1. Пример скрипта создающего таблицу шаблон.

3.2 Этап 2. Создание таблиц наследников

На основе таблицы созданной скриптом (листинг 1) можно создавать таблицы-наследники. Так как ограничение первичного ключа (*table_c_1_pkey*) не наследуется, то его необходимо явно указать в скрипте создания таблицы (листинг 2).

```
CREATE TABLE table_c_1
(
  CONSTRAINT table_c_1_pkey PRIMARY KEY (id)
)
INHERITS (table_c);
```

Листинг 2. Пример скрипта, создающего таблицу-наследника.

Для обеспечения целостности первичного ключа в СУБД *Postgresql* применяются последовательности.

Последовательность объектов (также называется последовательностью генераторов или просто последовательностью) являются специальной однорядной таблицей, созданной с помощью команды *CREATE SEQUENCE*. Последовательность объектов обычно используется для генерации уникальных идентификаторов для строк таблицы. Например, если в создаваемой таблице будет указанно поле с типом *serial* (*bigserial*), то последовательность будет создана автоматически (вида *ИмяТаблицы_ИмяПоля_seq*). При вставке данных в эту таблицу, каждая новая запись будет уникальное значение из

последовательности возвращаемое функцией *nextval('ИмяПоследовательности')*. Для просмотра текущего значения последовательности можно использовать код листинга 3 (в приведенном запросе используется встроенная функция получения значения последовательности).

```
SELECT currval('ИмяПоследовательности');
```

Листинг 3. Пример скрипта получения текущего значения последовательности.

В таблице созданной в листинге 2 все наследуемые таблицы будут использовать одну последовательность в качестве первичного ключа (*table_c_id_seq*), что гарантирует сквозную уникальность его значения (каждая таблица-наследник хранит уникальные данные).

3.3 Этап 3. Реализация функции создания нового периода

Данная функция необходима для создания нового периода (новой таблицы-наследника и создания триггерной функции отвечающей за перенаправление записи данных в новую таблицу, при этом запись в таблицу предыдущего периода блокируется).

Пример функции представлен на листинге 4.

```
CREATE OR REPLACE FUNCTION create_period()
RETURNS void AS
$BODY$
DECLARE
    now_date Date;
    now_year char(4);
    now_month char(2);
BEGIN
    RAISE NOTICE 'create_period(): start';

    now_date = now();
    now_year := to_char(now_date, 'YYYY');
    now_month := to_char(now_date, 'MM');

    -- create new table
```

```
            RAISE NOTICE 'create_period(): create new table';
            EXECUTE '
                 CREATE TABLE "table_c_' || now_year || '_' ||
now_month || '" '
                 || '(PRIMARY KEY ("id")) '
                 || 'INHERITS ("table_c")';

            -- create trigger function
            RAISE NOTICE 'create_period(): create trigger function';
            EXECUTE '
                 CREATE OR REPLACE FUNCTION
table_c_i_trigger_function() '
                 || 'RETURNS TRIGGER AS $$
                      BEGIN
                      INSERT INTO "table_c_' || now_year || '_' ||
now_month || '" VALUES (NEW.*);
                      RETURN NULL;
                      END; '
                 || '$$ LANGUAGE plpgsql;';

            RAISE NOTICE 'create_period(): end';
       END;
       $BODY$
       LANGUAGE plpgsql;
```

Листинг 4. Скрипт, реализующий функцию создания нового периода

3.4 Этап 4. Реализация триггера на событие INSERT

Созданную функцию необходимо вызвать для создания первого периода, после чего будет создана триггерная функция, которую нужно вызвать в триггере таблицы-родителя, пример вызова представлен в листинге 5.

```
       SELECT create_period();
       CREATE TRIGGER table_c_i_trigger BEFORE INSERT
                 ON "table_c" FOR EACH ROW EXECUTE PROCEDURE
table_c_i_trigger_function();
```

Листинг 5. Скрипт вызова функции создания первого периода и создания триггера на событие INSERT

4. Результаты работы

1. Вставка данных в первый созданный период (используется таблица *table_c*):

```
# INSERT INTO table_c(value) VALUES (1),(2),(3);
# SELECT * FROM table_c;
```

Результат представлен в таблице 1.

Таблица 1 - данные первого периода, таблица *table_c*

id	*value*	*time*
1	*1*	*2011-*
2	*2*	*2011-*
3	*3*	*2011-*

2. Создание второго периода:

```
# SELECT create_period();
```

Вставка данных во второй период (используется таблица table_c):

```
# INSERT INTO table_c(value) VALUES (4),(5),(6);
# SELECT * FROM table_c;
```

Результат представлен в таблице 2.

Таблица 2 - данные первого периода, таблица *table_c*

id	*value*	*time*
1	*1*	*2011-09-11 21:06:03.621682+04*
2	*2*	*2011-09-11 21:06:04.621682+04*
3	*3*	*2011-09-11 21:06:05.621682+04*
4	*4*	*2011-09-11 21:06:06.621682+04*
5	*5*	*2011-09-11 21:06:07.621682+04*

6	*6*	*2011-09-11 21:06:08.621682+04*

В таблицах наследниках данные распределены следующим образом:

```
# SELECT * FROM table_c_2011_09;
```

Результат представлен в таблице 3.

Таблица 3 - данные хранящиеся в таблица *table_c_2011_09*

id	*value*	*time*
1	*1*	*2011-09-11 21:06:03.621682+04*
2	*2*	*2011-09-11 21:06:04.621682+04*
3	*3*	*2011-09-11 21:06:05.621682+04*

```
# SELECT * FROM table_c_2011_10;
```

Результат представлен в таблице 4.

Таблица 4 - данные хранящиеся в таблица *table_c_2011_10*

id	*value*	*time*
4	*4*	*2011-09-11 21:06:06.621682+04*
5	*5*	*2011-09-11 21:06:07.621682+04*
6	*6*	*2011-09-11 21:06:08.621682+04*

Заключение

В результате исследования были построены математические модели способов хранения хронологических данных, а также предложено решение организации хранения хронологических данных по средствам использования методов разделения массивов данных в различные таблицы. Что позволило оптимизировать типовые запросы, наиболее часто выполняемые в системах мониторинга и прогнозирования.

Приложение

Листинг 6. Пример скрипта для создания одной таблицы с хронологическими данными.

```
CREATE TABLE "table_a"
(
    "id" bigserial NOT NULL,
    "value" double precision NOT NULL,
    "time" timestamp with time zone NOT NULL DEFAULT now(),
    CONSTRAINT "table_a_pkey" PRIMARY KEY ("id")
);
```

Листинг 7. Пример скрипта для создания таблицы с разбиением на архивные периоды и использованием дополнительного поля. Дополнительное поле хранит порядковый номер периода разбиения.

```
CREATE TABLE "table_b"
(
    "id" bigserial NOT NULL,
    "value" double precision NOT NULL,
    "time" timestamp with time zone NOT NULL DEFAULT now(),
    "period" int NOT NULL,
    CONSTRAINT "table_b_pkey" PRIMARY KEY ("id")
);
```

Литература

1. Дж. Уорсли, Дж. Дрейк, *PostgreSQL*. Для профессионалов. - Питер, 2003. - 496 стр.

2. *www.postgresql.org* [электронный ресурс]

3. Simon Riggs, Hannu Krosing, PostgreSQL 9 Admin Cookbook. - PacktPublishing, 2010. - 360 стр.