Методическое пособие по теме:

«Изучение постреляционных особенностей на примере наследования таблиц и многомерных полей;»

Цель: изучение постреляционных особенностей на примере наследования таблиц и многомерных полей;.

Содержание:

Наследование

PostgreSQL реализует наследование таблиц, что может быть полезно для проектировщиков баз данных.

Начнём со следующего примера: предположим, что мы создаём модель данных для городов. В каждом штате есть множество городов, но лишь одна столица. Мы хотим иметь возможность быстро получать город-столицу для любого штата. Это можно сделать, создав две таблицы: одну для столиц штатов, а другую для городов, не являющихся столицами. Однако, что делать, если нам нужно получить информацию о любом городе, будь то столица штата или нет? В решении этой проблемы может помочь наследование. Мы определим таблицу capitalskак наследника cities:

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

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

В этом случае таблица capitals наследует все столбцы своей родительской таблицы, cities. Столицы штатов также имеют дополнительный столбец state, в котором будет указан штат.

В PostgreSQL таблица может наследоваться от нуля или нескольких других таблиц, а запросы могут выбирать все строки родительской таблицы или все строки родительской и всех дочерних таблиц. По умолчанию принят последний вариант. Например, следующий запрос найдёт названия всех городов, включая столицы штатов, расположенных выше 500 футов:

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

Для данных из введения он выдаст:

```
   name    | altitude
-----------+----------
 Las Vegas |     2174
 Mariposa  |     1953
 Madison   |      845
```

А следующий запрос находит все города, которые не являются столицами штатов, но также находятся на высоте выше 500 футов:

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

   name    | altitude
-----------+----------
 Las Vegas |     2174
 Mariposa  |     1953
```

Здесь ключевое слово ONLY указывает, что запрос должен применяться только к таблице cities, но не к таблицам, расположенным ниже cities в иерархии наследования. Многие операторы, которые мы уже обсудили, — SELECT, UPDATE и DELETE — поддерживают ключевое слово ONLY.

Вы также можете добавить после имени таблицы *, чтобы обрабатывались и все дочерние таблицы:

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

Указывать * не обязательно, так как теперь это поведение подразумевается по умолчанию (если только вы не измените параметр конфигурации sql_inheritance). Однако такая запись может быть полезна тем, что подчеркнёт использование дополнительных таблиц.

В некоторых ситуациях бывает необходимо узнать, из какой таблицы выбрана конкретная строка. Для этого вы можете воспользоваться системным столбцом tableoid, присутствующим в каждой таблице:

```
SELECT c.tableoid, c.name, c.altitude
FROM cities c
WHERE c.altitude > 500;
```

Этот запрос выдаст:

```
 tableoid |   name    | altitude
----------+-----------+----------
   139793 | Las Vegas |     2174
   139793 | Mariposa  |     1953
   139798 | Madison   |      845
```

Механизм наследования не способен автоматически распределять данные команд INSERT или COPY по таблицам в иерархии наследования. Поэтому в нашем примере этот оператор INSERT не выполнится:

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

Мы могли надеяться на то, что данные каким-то образом попадут в таблицу capitals, но этого не происходит: INSERT всегда вставляет данные непосредственно в указанную таблицу.

Дочерние таблицы автоматически наследуют от родительской таблицы ограничения-проверки и ограничения NOT NULL (если только для них не задано

явно NO INHERIT). Все остальные ограничения (уникальности, первичный ключ и внешние ключи) не наследуются.

Таблица может наследоваться от нескольких родительских таблиц, в этом случае она будет объединять в себе все столбцы этих таблиц, а также столбцы, описанные непосредственно в её определении. Если в определениях родительских и дочерней таблиц встретятся столбцы с одним именем, эти столбцы будут «объединены», так что в дочерней таблице окажется только один столбец. Чтобы такое объединение было возможно, столбцы должны иметь одинаковый тип данных, в противном случае произойдёт ошибка. Наследуемые ограничения-проверки и ограничения NOT NULL объединяются подобным образом. Так, например, объединяемый столбец получит свойство NOT NULL, если какое-либо из порождающих его определений имеет свойство NOT NULL. Ограничения-проверки объединяются, если они имеют одинаковые имена; но если их условия различаются, происходит ошибка.

Отношение наследования между таблицами обычно устанавливается при создании дочерней таблицы с использованием предложения INHERITS оператора CREATE TABLE. Другой способ добавить такое отношение для таблицы, определённой подходящим образом — использовать INHERIT с оператором ALTER TABLE. Для этого будущая дочерняя таблица должна уже включать те же столбцы (с совпадающими именами и типами), что и родительская таблица. Также она должна включать аналогичные ограничения-проверки (с теми же именами и выражениями). Удалить отношение наследования можно с помощью указания NO INHERIT оператора ALTER TABLE.

Для создания таблицы, которая затем может стать наследником другой, удобно воспользоваться предложением LIKE оператора CREATE TABLE. Такая команда создаст новую таблицу с теми же столбцами, что имеются в исходной. Если в исходной таблицы определены ограничения CHECK, для создания полностью совместимой таблицы их тоже нужно скопировать, и это можно сделать, добавив к предложению LIKE параметр INCLUDING CONSTRAINTS.

Родительскую таблицу нельзя удалить, пока существуют унаследованные от неё. Так же как в дочерних таблицах нельзя удалять или модифицировать столбцы или ограничения-проверки, унаследованные от родительских таблиц. Если вы

хотите удалить таблицу вместе со всеми её потомками, это легко сделать, добавив в команду удаления родительской таблицы параметр CASCADE.

При изменениях определений и ограничений столбцов команда ALTER TABLE распространяет эти изменения вниз в иерархии наследования. Однако удалить столбцы, унаследованные дочерними таблицами, можно только с помощью параметра CASCADE. При создании отношений наследования команда ALTER TABLE следует тем же правилам объединения дублирующихся столбцов, что и CREATE TABLE.

Гиперкубическая модель данных:

Многомерные базы данных рассматривают данные как кубы, которые являются обобщением электронных таблиц на любое число измерений. Кроме того, кубы поддерживают иерархию измерений и формул без дублирования их определений. Набор соответствующих кубов составляет многомерную базу данных (или хранилище данных).

Кубами легко управлять, добавляя новые значения измерений. В обычном обиходе этим термином обозначают фигуру с тремя измерениями, однако теоретически куб может иметь любое число измерений. На практике чаще всего кубы данных имеют от 4 до 12 измерений. Современный инструментарий часто сталкивается с нехваткой производительности, когда так называемый гиперкуб имеет свыше 10-15 измерений.

Комбинации значений измерений определяют ячейки куба. В зависимости от конкретного приложения ячейки в кубе могут располагаться как разрозненно, так и плотно. Кубы, как правило, становятся разрозненными по мере увеличения числа размерностей и степени детализации значений измерений.

На рис. 1 показан куб, содержащий данные по продажам в двух датских городах, указанных в таблице 1 с дополнительным измерением — «Время». В соответствующих ячейках хранятся данные об объеме продаж. В примере можно обнаружить «факт» — непустую ячейку, содержащую соответствующие числовые параметры — для каждой комбинации время, продукт и город, где была совершена, по крайней мере, одна продажа. В ячейке размещаются числовые значения, связанные с фактом — в данном случае, это объем продаж — единственный параметр.

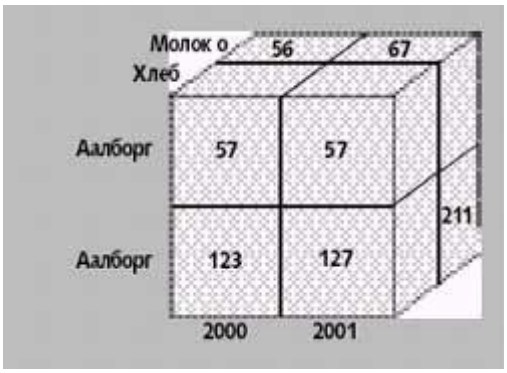

Рис. 1. Пример куба, содержащего данные о продажах.

В этом случае куб обобщает электронную таблицу из Таблицы 1, добавляя к ней третье измерение — время

В общем случае куб позволяет представить только два или три измерения одновременно, но можно показывать и больше за счет вложения одного измерения в другое. Таким образом, путем проецирования куба на двух- или трехмерное пространство можно уменьшить размерность куба, агрегировав некоторые размерности, что ведет к работе с более комплексными значениями параметров. К примеру, рассматривая продажи по городам и времени, мы агрегируем информацию для каждого сочетания город и время. Так, на рис. 1, сложив поля 127 и 211, получаем общий объем продаж для Копенгагена в 2001 году.