Практическая работа №4.
Основы аналитики данных

Цель работы: научиться пользоваться базовыми инструментами анализа данных, встроенных в СУБД Postgres.

Теоретический материал

При разработки систем поддержки принятия решений используются модели, которые строятся по результатам анализа данных. В ходе анализа данных изучается, как одни данные влияют на другие, например, как зависит прибыль от того, в каком регионе производятся продажи.

Предиктор – это набор данных, который при анализе рассматривается как то, что влияет на изучаемое явление

Отклик – это набор данных, который описывает результат влияния предиктора на изучаемое явление. В некоторых случаях анализ возможен без учёта отклика.

Например, в примере с влиянием региона продаж на прибыль регион является предиктором, а прибыль – откликом.

Для того чтобы выполнить анализ данных, необходимо, чтобы часть данных выражалась естественными числами, либо предиктор, либо отклик, либо и предиктор, и отклик одновременно. Если предиктор или отклик не выражается вещественными числами, он должен выражаться дискретными значениями, не подразумевающими применение к этим значениям каких-либо операций. Если предиктор или отклик выражается вещественными числами, то это называется количество. Если предиктор или отклик выражается дискретными значениями, то это называется качество. В соответствии с тем, что является количеством, а что качеством, принципиально возможно четыре вида анализа:

1. Дисперсионный анализ изучает влияние качественных предикторов на количественный отклик. Пример с влиянием региона продаж на прибыль как раз относится именно к этому виду анализа.
2. Регрессионный анализ изучает влияние количественных предикторов на количественный отклик. Например, может изучаться влияние средней зарплаты сотрудников на прибыль отдела, в котором эти сотрудники работают
3. Дискриминантный анализ изучает влияние количественных предикторов на качественный отклик. Например, может изучаться, по каким измеряемым показателям клинического анализа крови предсказывать болезни.
4. Кластерный анализ изучает многомерное пространство количественных предикторов и позволяет выделить в нём облака точек. Обычно это используется при анализе социальной информации,

например, поиск групп товаров, которые покупатели часто заказывают совместно.

Любой из видов анализа в качестве исходных данных требует плоской модели данных, т.е. одной большой таблицы, в которой каждая колонка отвечает за определённый признак, который может быть предиктором или откликом в рамках конкретного исследования данных, а каждая строка отвечает за одно из множества наблюдений, содержащих по одному значению на каждый признак. Однако в целом плоская модель данных в чистом виде очень неэффективна. Кроме того, при выполнении анализа данных требуется решать задачи с высокой вычислительной сложностью, поэтому перед анализом они должны быть предварительно агрегированы. Поэтому существуют специализированные технологии, позволяющие автоматизировать предварительную подготовку данных к анализу с высоким уровнем оптимизации.

Для того чтобы вынести анализ из рабочей базы данных вовне, аналитика производится в другой базе данных. Чтобы обеспечить поддержку актуальности данных, необходимо использовать инструментарий для подключения внешних таблиц к базе данных, с которым вы познакомились в практической работе номер 2. Кроме того, для подготовки данных к анализу PostgreSQL предоставляет специальные инструменты: возможность создания своих агрегатных функций, а также операторы **GROUPING SETS**, **ROLLUP** и **CUBE**. Оператор **GROUPING SETS** позволяет создавать объединения запросов с несколькими вариантами группировки с одной и той же агрегатной функцией, оператор **ROLLUP** автоматически создаёт наборы группировок для иерархических структур, а оператор **CUBE** создаёт набор группировок со всеми подмножествами указанного множества атрибутов. Все эти операторы позволяют строить простейшие OLAP-кубы – эффективные структуры данных, предназначенные для первичной агрегации данных перед анализом, а также позволяющие делать простейший анализ с использованием агрегатных функций.

Расширение для анализа данных

Для выполнения анализа данных Postgres содержит стандартное расширение table_func, в котором можно решать три основные задачи:

1. Генерировать случайные числа с нормальным распределением;
2. Обрабатывать таблицы с иерархическими структурами;
3. Преобразовывать отношения в матрицы для более удобной визуализации анализируемых данных.

Для генерации случайных чисел используется функция **normal_rand**. Она имеет три параметра: сколько чисел нужно сгенерировать, среднее значение и стандартное отклонение. Пример использования:

```
SELECT normal_rand(10, 25, 5) AS "values";
```

Для обработки таблиц с иерархическими структурами используется функция **connectby**, имеющая следующую сигнатуру:

```
connectby
(
   relname text,
   keyid_fld text,
   parent_keyid_fld text
   [, orderby_fld text],
   start_with text,
   max_depth integer
   [, branch_delim text]
) → setoff record.
```

Первый параметр этой функции – это имя таблицы, содержащей унарную связь «многие к одному». Для реализации этой связи у таблицы должен быть простой ключ; имя соответствующей колонки – это второй параметр. На него ссылается внешний ключ из этой же таблицы; имя соответствующей колонки – это третий параметр. Остальные параметры определяются так: *parent_keyid_fld* - имя поля, содержащего ключ родителя, *orderby_fld* - имя поля, по которому сортируются потомки (необязательно), *start_with* - значение ключа отправной строки, *max_depth* - максимальная глубина, на которую можно погрузиться, либо ноль для неограниченного погружения, *branch_delim* - строка, разделяющая ключи в выводе ветви (необязательно). Если в таблице, устроенной таким образом, обнаружились петли, то функция завершится с ошибкой.

Функция **crosstab** предназначена для преобразования отношений в матрицы. В качестве параметра ей нужно передать два SQL-запроса. Первый должен возвращать таблицу с тремя колонками: значения первой колонки должны стать строками результирующей матрицы, значения второй колонки должны стать колонками результирующей матрицы, а значения третьей колонки – ячейками матрицы. После вызова этой функции необходимо перечислить получившиеся колонки, например, так:

```
SELECT * FROM crosstab(
  $$ SELECT year, c.name, ROUND(AVG(yield), 2)
     FROM experiments e
     JOIN cultures c ON e.culture_id = c.id
     GROUP BY year, c.name
     ORDER BY 1, 2 $$,
  $$ SELECT name FROM cultures ORDER BY name $$
) AS result (year INT, "Горох" NUMERIC, "Овёс"
NUMERIC, "Пшеница" NUMERIC, "Ячмень" NUMERIC);
```

Эта функция требует того, чтобы последняя строчка запроса, в которой перечисляются результирующие колонки, соответствовала данным. Это может быть не очень удобно, но в рамках реляционной модели данных необходимо.

Создание агрегатной функции

После завершения интеграции серверов всё готово к началу собственно анализа данных. В ходе анализа данные группируются и агрегируются. Сама СУБД PostgreSQL предоставляет довольно много готовых агрегатных функций, однако иногда может потребоваться разрабатывать свои. Агрегатные функции состоят из трёх частей: инициализации, аккумуляции и завершения. Инициализация задаёт начальное значение переменным, которые обрабатываются агрегатной функцией, аккумуляция обрабатывает каждое из входных значений, а завершение выполняет дополнительные действия после того, как все входные значения обработаны, и возвращает окончательный результат вычисления агрегатной функции. При разработке агрегатных функций, инициализация должна быть разработана последней.

Рассмотрим создание агрегатной функции на примере функции расчёта среднего геометрического.

1. Функция аккумуляции должна иметь те же самые параметры, что и создаваемая агрегатная функция. Кроме того, у неё добавляется дополнительный первый параметр, содержащий состояние, сохраняемое между обработками отдельных входных значений. Возвращать эта функция должна значение этого же параметра после изменения состояния. В случае среднего геометрического для хранения состояния нужен массив из двух вещественных чисел. Первый элемент массива будет хранить произведение всех входных значений, а второй – их количество. Учитывая всё сказанное, функция будет выглядеть так:

```
CREATE OR REPLACE FUNCTION g_accum
(
    state numeric[],
    value numeric
)
RETURNS numeric[] AS $$
BEGIN
    state[1] := state[1] * value;   -- Произведение
    state[2] := state[2] + 1;       -- Количество

    RETURN state;
END;
$$ LANGUAGE plpgsql;
```

Обратите внимание, что нумерация элементов массива в plpgsql начинается с единицы.

2. Функция завершения должна в качестве параметра принимать состояние, которое принимает и возвращает функция аккумуляции, и возвращать итоговое значение агрегатной функции. В случае среднего геометрического функция завершения должна накопленное произведение значений возвести в степень, обратную количеству обработанных значений.

Для безопасного выполнения она должна проверить это количество, чтобы не делить на 0. Учитывая всё сказанное, функция будет выглядеть так:

```
CREATE OR REPLACE FUNCTION g_final(state numeric[])
RETURNS numeric AS $$
BEGIN
    IF state[2] = 0 THEN
        RETURN 0;  -- Если нет данных, возвращаем 0
    END IF;

    -- Корень степени count
    RETURN power(state[1], 1.0 / state[2]);
END;
$$ LANGUAGE plpgsql;
```

3. Функция аккумуляции и функция завершения – это обычные функции SQL. Функция инициализации создаёт собственно агрегатную функцию, и синтаксис у неё другой. Необходимо написать команду **CREATE AGGREGATE**, после чего указать имя создаваемой агрегатной функции, тип возвращаемого значения в скобках и задать у неё четыре параметра. **SFUNC** задаёт функцию аккумуляции, **STYPE** задаёт тип состояния, **FINALFUNC** задаёт функцию завершения, а **INITCOND** в апострофах задаёт начальное состояние. Для расчёта среднего геометрического начальное произведение должно быть равно 1, а начальный счётчик 0.

Учитывая всё сказанное, функция будет выглядеть так:

```
CREATE AGGREGATE g_mean (numeric)
(
    SFUNC = g_accum,       -- Функция агрегации

    -- Тип состояния (массив из двух элементов)
    STYPE = numeric[],

    FINALFUNC = g_final,   -- Функция завершения
    INITCOND = '{1, 0}'    -- Начальное состояние
);
```

4. Напишите простой запрос с созданной агрегатной функции, чтобы убедиться в её работоспособности.

Создание куба

Для создания куба в PostgreSQL необходимо написать запрос с группировкой, в которой используется оператор **GROUPING SETS**, **ROLLUP** или **CUBE**. После любого из этих операторов в скобках следует указать список колонок, по которым необходимо сделать группировку. С помощью **GROUPING SETS** можно явно задать несколько наборов колонок, по

которым требуется сделать группировку. Они перечисляются в общих круглых скобках через запятую, и каждый из них представляет свой собственный список колонок через запятую в круглых скобках. Например:

```
SELECT carname, carregnumber,
g_mean(personid) FROM personcars
GROUP BY GROUPING SETS
(
     (carname, carregnumber),
     (carregnumber)
)
```

При использовании операторов ROLLUP или CUBE колонки просто перечисляются через запятую:

```
SELECT carname,carregnumber,
g_mean(personid) FROM personcars
GROUP BY CUBE(carname, carregnumber);
```

Наборы колонок для группировки эти операторы создают сами.

Для того чтобы куб не загружал основную базу данных, следует сделать его в виде материализованного представления:

```
CREATE MATERIALIZED VIEW имя_представления AS
SELECT carname, carregnumber,
g_mean(personid) FROM personcars
GROUP BY CUBE(carname, carregnumber);
```

Для того, чтобы загрузить данные в материализованное представление в «аналитической» базе данных из «рабочей» базы данных, необходимо выполнить следующую команду:

```
REFRESH MATERIALIZED VIEW имя_представления;
```

Задание

1. Создать два сервера в pgAdmin на разных портах.
2. На первом сервере развернуть учебную базу данных Canteen Dishes.
3. На втором сервере создать БД для анализа данных.
4. Создать в «рабочей» базе данных пользователя с правами на чтение из любых таблиц.
5. Создать в базе данных для анализа данных внешние таблицы для таблиц Canteen Dishes.
6. Написать на «аналитической» базе данных агрегатную функцию. Саму функцию нужно согласовать с преподавателем, желательно брать что-нибудь максимально близкое к индивидуальному заданию по ознакомительной практике.

7. Используя эту функцию, создать в «аналитической» базе данных на основе внешних данных, получаемых из базы данных Canteen Dishes, куб.
8. Внести в базу данных Canteen Dishes новые данные.
9. Обновить куб в «аналитической» базе данных.
10. Создать в «аналитической» базе данных пользователя, который имеет право только читать данные из куба.
11. Создать в «рабочей» базе данных внешнюю таблицу, которая будет брать данные из куба, находящегося в «аналитической» базе данных.

Оформление отчёта

Отчёт должен иметь титульный лист с указанием ФИО и группы студента, а так же темы практической работы. После этого должны быть указаны порты и настройки внешних подключений к созданным серверам (выписки из конфигурационных файлов). Затем должны быть указаны имена баз данных и скрипты создания нового пользователя в «рабочей» базе данных, подключения внешних таблиц в «аналитической» базе данных, создания в ней агрегатной функции и куба. Затем должны быть скриншоты и скрипты, показывающие, какие изменения были внесены в базу данных Canteen Dishes, и скрипт обновления куба. Затем должны быть скрипты создания пользователя «аналитической» базы данных для чтения куба, внешнего подключения к кубу и скриншот, показывающий, что пользователь в «рабочей» базе данных может просматривать данные из куба.