В качестве бизнес-области для проектной работы взята модель продаж.

Продажи товаров (product_id) осуществляются в магазинах (store_id) в течении всего года (date_id = день) в штуках (qty) и национальной валюте (РУБЛИ, sum_nv).

- Детализация: магазин - товар – день

Товар является элементом справочника Products.

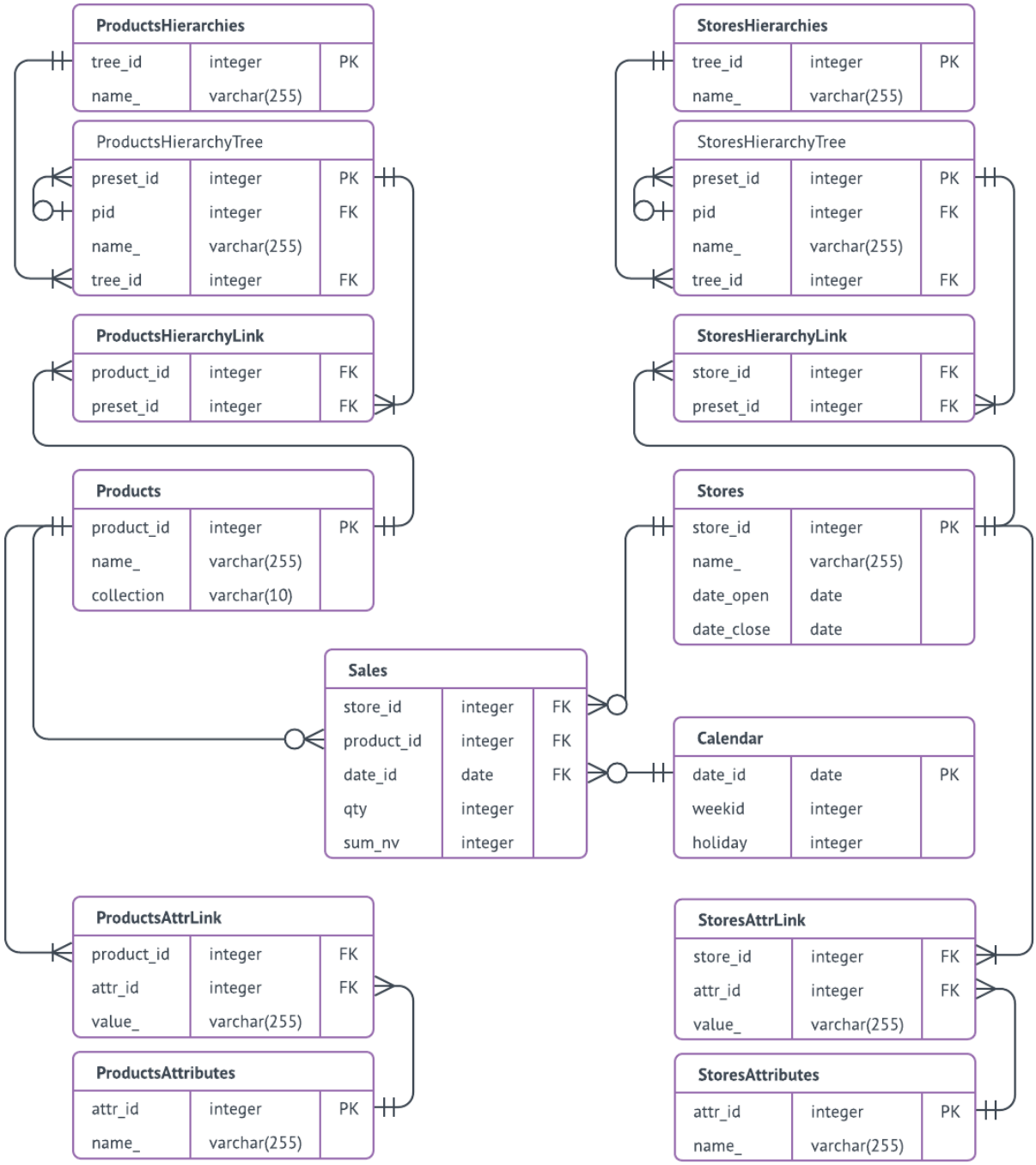
- У товара есть атрибуты (attr_id) ProductsAttributes (связь через link-таблицу ProductsAttrLink)
- Товарные иерархии ProductsHierarchyTree (справочник иерархий товара ProductsHierarchies) позволяют анализировать товар в различных преднастроенных деревьях

Магазин является элементом справочника Stores

- У магазина есть атрибуты StoresAttributes (связь через link-таблицу StoresAttrLink)
- Магазинные иерархии StoresHierarchyTree (справочник иерархий магазинов StoresHierarchies) позволяют анализировать магазины в различных преднастроенных деревьях

Дата является элементом календаря, простейший вариант которого включает лишь номер недели, год и признак выходного дня.

Проектная работа Нетология. SQL. [Никифоров Владимир]



Проектная работа Нетология. SQL. [Никифоров Владимир]

Скрипты создания таблиц и наполнения:

```
-- Проектная работа по модулю "SQL и получение данных"
-- ФИО: Никифоров Владимир
-- DDL&DML
DROP TABLE IF EXISTS stores CASCADE;
create table stores(
	store_id integer primary key,
	name_ varchar(255),
	date_open date,
	date_close date);
delete from stores;
insert into stores values(1, 'Central Universal Store', '2010-01-01', NULL);
insert into stores values(2, 'Universal State Store', '2000-01-01', NULL);
insert into stores values(3, 'Small Regional Store', '2001-01-01', '2010-12-31');
insert into stores values(4, 'Brand New Store by Elon Mask', '2016-01-01', '2019-01-01');
insert into stores values(5, 'Grandmother Zina`s Small Store', '1975-01-01', NULL);
commit;
-- select * from stores;
DROP TABLE IF EXISTS products CASCADE;
create table products(product_id integer primary key,
                                        name_ varchar(255),
                                        collection varchar(10));
insert into products values(1, 'BigFoot Boots Black', 'Winter');
insert into products values(2, 'Tuxedo', 'Summer');
insert into products values(3, 'Queen` Dress', 'Summer');
insert into products values(4, 'Heavy sneakers', 'Summer');
insert into products values(5, 'Nice slippers', 'Summer');
commit;
-- select * from products;
DROP TABLE IF EXISTS calendar CASCADE;
create table calendar(date_id date primary key,
                                        weekid integer,
                                        holiday integer);
insert into calendar
SELECT
	datum AS date_id,
	to_char(datum, 'IW')::integer AS weekid,
	CASE WHEN EXTRACT(isodow FROM datum) IN (6, 7) THEN 1 ELSE 0 END AS
holiday
FROM (
	-- There are 3 leap years in this range, so calculate 365 * 10 + 3 records
	SELECT '2010-01-01'::DATE + SEQUENCE.DAY AS datum
	FROM generate_series(0,3652) AS SEQUENCE(DAY)
	GROUP BY SEQUENCE.DAY
    ) DQ
ORDER BY 1;
commit;
-- select * from calendar;
DROP TABLE IF EXISTS sales CASCADE;
```

```
create table sales(
	store_id integer references stores(store_id),
	product_id integer references products(product_id),
	date_id date references calendar(date_id),
	qty integer,
	sum_nv integer);
insert into sales(store_id, product_id, date_id, qty, sum_nv)
select floor(random() * 5 + 1)::int store_id,  floor(random() * 5 + 1)::int product_id, date_id,
		floor(random() * 2 + 1)::int qty, floor(random() * 10 + 1)::int*1000-1 sum_nv
FROM (
	SELECT '2010-01-01'::DATE + 10*SEQUENCE.DAY AS date_id
	FROM generate_series(0,300) AS SEQUENCE(DAY)
	GROUP BY SEQUENCE.DAY
    ) DQ;
commit;
-- select * from sales;
DROP TABLE IF EXISTS ProductsHierarchies CASCADE;
create table ProductsHierarchies(tree_id integer primary key,
                                                        name_ varchar(255));
insert into ProductsHierarchies values(1, 'Main product hierarchy');
insert into ProductsHierarchies values(2, 'Alternative product hierarchy');
commit;
-- select * from ProductsHierarchies;
DROP TABLE IF EXISTS ProductsHierarchyTree CASCADE;
create table ProductsHierarchyTree(preset_id integer primary key,
                                                        pid integer references
ProductsHierarchyTree(preset_id),
                                                        name_ varchar(255),
                                                        tree_id integer references
ProductsHierarchies(tree_id));
insert into ProductsHierarchyTree values(101, null, 'Total', 1);
insert into ProductsHierarchyTree values(102, 101, 'Clothes', 1);
insert into ProductsHierarchyTree values(103, 102, 'Casual', 1);
insert into ProductsHierarchyTree values(104, 102, 'ForHolidays', 1);
insert into ProductsHierarchyTree values(105, 101, 'Shoes', 1);
insert into ProductsHierarchyTree values(106, 105, 'Casual', 1);
insert into ProductsHierarchyTree values(107, 105, 'ForHolidays', 1);
insert into ProductsHierarchyTree values(201, null, 'Total', 2);
insert into ProductsHierarchyTree values(202, 201, 'Man', 2);
insert into ProductsHierarchyTree values(203, 202, 'Clothes', 2);
insert into ProductsHierarchyTree values(204, 202, 'Shoes', 2);
insert into ProductsHierarchyTree values(205, 201, 'Woman', 2);
insert into ProductsHierarchyTree values(206, 205, 'Clothes', 2);
insert into ProductsHierarchyTree values(207, 205, 'Shoes', 2);
commit;
-- select * from ProductsHierarchyTree;
-- select * from products
DROP TABLE IF EXISTS ProductsHierarchyLink CASCADE;
create table ProductsHierarchyLink(product_id integer references products(product_id),
```

```
                                                    preset_id integer references
ProductsHierarchyTree(preset_id));
insert into ProductsHierarchyLink values(1,106);
insert into ProductsHierarchyLink values(1,204);
insert into ProductsHierarchyLink values(2,104);
insert into ProductsHierarchyLink values(2,203);
insert into ProductsHierarchyLink values(3,104);
insert into ProductsHierarchyLink values(3,206);
insert into ProductsHierarchyLink values(4,106);
insert into ProductsHierarchyLink values(4,204);
insert into ProductsHierarchyLink values(5,107);
insert into ProductsHierarchyLink values(5,207);
commit;
-- select * from ProductsHierarchyLink;
DROP TABLE IF EXISTS StoresHierarchies CASCADE;
create table StoresHierarchies(tree_id integer primary key,
                                                    name_ varchar(255));
insert into StoresHierarchies values(1, 'Main stores hierarchy');
insert into StoresHierarchies values(2, 'BusinessOwner` stores hierarchy');
commit;
-- select * from StoresHierarchies;
DROP TABLE IF EXISTS StoresHierarchyTree CASCADE;
create table StoresHierarchyTree(preset_id integer primary key,
                                                    pid integer references
StoresHierarchyTree(preset_id),
                                                    name_ varchar(255),
                                                    tree_id integer references
StoresHierarchies(tree_id));
insert into StoresHierarchyTree values(1, null, 'Total by stores', 1);
insert into StoresHierarchyTree values(2, 1, 'Moscow', 1);
insert into StoresHierarchyTree values(3, 1, 'Region', 1);
insert into StoresHierarchyTree values(11, null, 'All stores', 2);
insert into StoresHierarchyTree values(12, 11, 'Strong', 2);
insert into StoresHierarchyTree values(13, 11, 'Just funny', 2);
commit;
-- select * from StoresHierarchyTree;
-- select * from stores;
DROP TABLE IF EXISTS StoresHierarchyLink CASCADE;
create table StoresHierarchyLink(store_id integer references stores(store_id),
                                                    preset_id integer references
StoresHierarchyTree(preset_id));
insert into StoresHierarchyLink values(1,2);
insert into StoresHierarchyLink values(2,2);
insert into StoresHierarchyLink values(3,3);
insert into StoresHierarchyLink values(4,2);
insert into StoresHierarchyLink values(5,3);
insert into StoresHierarchyLink values(1,12);
insert into StoresHierarchyLink values(2,12);
insert into StoresHierarchyLink values(3,13);
insert into StoresHierarchyLink values(4,13);
```

```
insert into StoresHierarchyLink values(5,13);
commit;
-- select * from StoresHierarchyLink;
-- select * from products;
DROP TABLE IF EXISTS ProductsAttributes CASCADE;
create table ProductsAttributes(attr_id integer primary key,
                                                    name_ varchar(255));
insert into ProductsAttributes values(1, 'Color');
insert into ProductsAttributes values(2, 'Size');
commit;
-- select * from ProductsAttributes;
DROP TABLE IF EXISTS ProductsAttrLink CASCADE;
create table ProductsAttrLink(product_id integer references products(product_id),
                                                    attr_id integer references
ProductsAttributes(attr_id),
                                                    value_ varchar(255));
insert into ProductsAttrLink values(1, 1, 'Black');
insert into ProductsAttrLink values(2, 1, 'Black');
insert into ProductsAttrLink values(3, 1, 'Red');
insert into ProductsAttrLink values(4, 1, 'White');
insert into ProductsAttrLink values(5, 1, 'Red');
insert into ProductsAttrLink values(1, 2, '48');
insert into ProductsAttrLink values(2, 2, 'M');
insert into ProductsAttrLink values(3, 2, 'S');
insert into ProductsAttrLink values(4, 2, '48');
insert into ProductsAttrLink values(5, 2, '37');
commit;
-- select * from ProductsAttrLink;
-- select * from stores;
DROP TABLE IF EXISTS StoresAttributes CASCADE;
create table StoresAttributes(attr_id integer primary key,
                                                    name_ varchar(255));
insert into StoresAttributes values(1, 'City');
insert into StoresAttributes values(2, 'IsLegal');
commit;
-- select * from StoresAttributes;
DROP TABLE IF EXISTS StoresAttrLink CASCADE;
create table StoresAttrLink(store_id integer references stores(store_id),
                                                    attr_id integer references
StoresAttributes(attr_id),
                                                    value_ varchar(255));
insert into StoresAttrLink values(1, 1, 'Moscow');
insert into StoresAttrLink values(2, 1, 'Moscow');
insert into StoresAttrLink values(3, 1, 'Tula');
insert into StoresAttrLink values(4, 1, 'San Francisco');
insert into StoresAttrLink values(5, 1, 'Nyagan-city');
commit;
-- select * from StoresAttrLink;
```

Проектная работа Нетология. SQL. [Никифоров Владимир]

Запросы к таблицам:

```
-- Проектная работа по модулю "SQL и получение данных"
-- ФИО: Никифоров Владимир
-- SQL
-- Написать не менее 10 SQL запросов к базе данных.
-- В запросах должны быть отражены как базовые команды, так и аналитические функции (не менее 3 запросов).
-- Должно присутствовать описание того, что вы получаете путем каждого запроса.

select * from stores;
select * from products;
select * from calendar;
select * from sales;
select * from ProductsHierarchies;
select * from ProductsHierarchyTree;
select * from ProductsHierarchyLink;
select * from StoresHierarchies;
select * from StoresHierarchyTree;
select * from StoresHierarchyLink;
select * from ProductsAttributes;
select * from ProductsAttrLink;
select * from StoresAttributes;
select * from StoresAttrLink;

-- В какие дни больше покупают
select case when c.holiday = 0 then 'Рабочий день' else 'Выходной день' end day_type,
round(avg(qty),2) qty
  from sales s
  join calendar c on s.date_id = c.date_id
 group by c.holiday
 order by qty desc;
-- Сумма по каждому товару
select p.name_, sum(sum_nv) sum_nv
  from sales s
  join products p on s.product_id = p.product_id
 group by p.name_
 order by 2 desc;
-- Посмотрим на ранги магазинов по суммам продаж
select name_, sum_nv, dense_rank() over (order by sum_nv desc) rank_
  from (select distinct p.name_, sum(sum_nv) over (partition by p.name_) sum_nv
                  from sales s
                  join stores p on s.store_id = p.store_id
        ) t;
-- Какой цвет продается лучше всего? Группировка по атрибутам товара + сумма продаж.
select l.value_, sum(sum_nv) sum_nv
  from ProductsAttributes a
  join ProductsAttrLink l on a.attr_id = l.attr_id
  join sales p on l.product_id = p.product_id
 where a.attr_id = 1
```

```
 group by l.value_
 order by 2 desc;
-- Где продажи лучше всего? Группировка по атрибутам магазина + сумма продаж
select l.value_, sum(sum_nv) sum_nv
  from StoresAttributes a
  join StoresAttrLink l on a.attr_id = l.attr_id
  join sales p on l.store_id = p.store_id
 where a.attr_id = 1
 group by l.value_
 order by 2 desc;
-- Посмотрим на общие суммы продаж по номеру недели в году и определим на какой недели самый большой прирост относительно предыдущей недели
select weekid, sum_nv, sum_nv - lag(sum_nv) over (order by weekid) delta
  from (select c.weekid, sum(sum_nv) sum_nv
                  from sales p
                  join calendar c on c.date_id = p.date_id
                 group by c.weekid
             ) t
 order by 3 desc nulls last
 limit 1;
-- Определим товар bestseller (в штуках) для каждого магазина
with sums as (
select store_id, product_id, sum(qty) qty
  from sales
 group by store_id, product_id
 order by store_id, qty desc
), ranks as (select store_id, product_id, qty, rank() over (partition by store_id order by qty desc) rnk
                                 from sums
                               )
select s.name_ store_name, p.name_ product_name, qty
  from ranks r
  join products p on r.product_id = p.product_id
  join stores s on r.store_id = s.store_id
 where rnk = 1;
--================ Получим товарную иерархию с товарами ================
-- Построим дерево товаров с помощью рекурсивного CTE:
with recursive cte as (
   select pid, preset_id, name_, 1 lvl
     from ProductsHierarchyTree
    where tree_id = 1
            and pid is null
    union all
   Select t.pid, t.preset_id, t.name_, c.lvl + 1
     from cte c
     join ProductsHierarchyTree t ON t.pid = c.preset_id
   )
select preset_id, name_, lvl
  from cte
 order by lvl;
commit;
```

```
-- Выглядит ужасно (сначала идут элементы уровня 1, потом уровня 2, потом уровня 3)
-- Попробуем так:
WITH RECURSIVE cte AS (
SELECT pl.preset_id::varchar, pl.pid::varchar, ARRAY[name_]::varchar as path
 FROM ProductsHierarchyTree pl
 WHERE tree_id = 1 and pl.pid is null
 UNION ALL
 SELECT t2.preset_id::varchar, t2.pid::varchar, (cte.path || '/'|| t2.name_)::varchar
 FROM ProductsHierarchyTree as t2 inner join cte on (cte.preset_id=t2.pid::varchar)
)
SELECT preset_id, path as path
  FROM cte;
-- Уже лучше - подобие sys_connect_by_path, но всё-равно сортировка по уровням.
-- Можно было бы конечно отсортировать по preset_id, но это не наш подход => ставим
tablefunc
CREATE EXTENSION IF NOT EXISTS tablefunc;
-- Только сначала сделаем спеуциальную view с объеденненным товарным деревом и товаром
drop view if exists v_products_hierarchy;
create or replace view v_products_hierarchy as
select preset_id, pid, name_, tree_id, null product_id, name_ preset_name, null product_name
  from ProductsHierarchyTree
 union all
select (-1000)*l.preset_id + p.product_id preset_id, l.preset_id pid, p.name_, t.tree_id, p.product_id,
null preset_name, p.name_ product_name
  from ProductsHierarchyLink l
  join ProductsHierarchyTree t on l.preset_id = t.preset_id
  join products p on l.product_id = p.product_id;
commit;
-- А теперь повеселимся:
with recursive tree_full_names as (
select pl.preset_id::varchar, pl.pid::varchar, ARRAY[name_]::varchar as path
  from v_products_hierarchy pl
 where tree_id = 1
   and pl.pid is null
 union all
select t2.preset_id::varchar, t2.pid::varchar, (cte.path ||'/'|| t2.name_)::varchar
  from v_products_hierarchy as t2
  join tree_full_names cte on cte.preset_id = t2.pid::varchar
),
nice_tree as (select preset_id, pid, level lvl, row_number() over() order_
                                from connectby('v_products_hierarchy', 'preset_id', 'pid', '101', 0)
                                        as t(preset_id integer, pid integer, level int)
                             )
select t.*, v.product_id, preset_name, product_name, tfn.path product_full_name
  from nice_tree t
            join tree_full_names tfn on t.preset_id::varchar = tfn.preset_id
            join v_products_hierarchy v on t.preset_id = v.preset_id
 order by order_;
--================ Получим продажи товара через товарную иерархию
================
```

```
-- Круто! Посмотрим продажи товара через товарную иерархию
-- Чтобы не гонять весь код построения дерева - обернем во view:
drop view if exists v_product_tree;
create or replace view v_product_tree as
with recursive tree_full_names as (
select pl.preset_id::varchar, pl.pid::varchar, ARRAY[name_]::varchar as path
  from v_products_hierarchy pl
 where tree_id = 1
   and pl.pid is null
 union all
select t2.preset_id::varchar, t2.pid::varchar, (cte.path ||'/'|| t2.name_)::varchar
  from v_products_hierarchy as t2
  join tree_full_names cte on cte.preset_id = t2.pid::varchar
),
nice_tree as (select preset_id, pid, level lvl, row_number() over() order_
                                from connectby('v_products_hierarchy', 'preset_id', 'pid', '101', 0)
                                        as t(preset_id integer, pid integer, level int)
                              )
select t.*, v.product_id, preset_name, product_name, tfn.path product_full_name
  from nice_tree t
            join tree_full_names tfn on t.preset_id::varchar = tfn.preset_id
            join v_products_hierarchy v on t.preset_id = v.preset_id
 order by order_;
commit;
-- Итак, продажи товара через товарную иерархию:
select product_full_name, product_name, order_, sum(s.qty) qty
  from v_product_tree v
  left join sales s on v.product_id = s.product_id
 group by product_full_name, product_name, order_
 order by order_;
commit;
--================ Получим продажи товара по всей товарной иерархии
================
-- Да, продажи есть на товаре, но теперь уже хочется получить суммы и вверх по дереву
with tt as (select preset_id, pid, sum(s.qty) qty
                                 from v_product_tree v
                                 left join sales s on v.product_id = s.product_id
                                group by preset_id, pid
                               ),
sls as (
with recursive tree_sales as (
select pl.preset_id, pl.pid, qty
  from tt pl
 where qty is not null
 union all
select t2.preset_id, t2.pid, cte.qty qty
  from tt t2
  join tree_sales cte on cte.pid = t2.preset_id
)
select * from tree_sales),
```

```
presets_sum as (select preset_id, sum(qty) qty_sum from sls group by preset_id order by preset_id)
select product_full_name product, qty_sum total_sum
  from v_product_tree v
  join presets_sum p on v.preset_id = p.preset_id
 order by order_;

--================ Получим продажи в разрезе магазинной иерархии
===============
-- сначала подготовим вью - альтернативный вариант - параметризированная функция (чтобы
номер дерева подавать в функцию)
drop view if exists v_obj_hierarchy;
create or replace view v_obj_hierarchy as
select preset_id, pid, name_, tree_id, null store_id, name_ preset_name, null store_name
  from StoresHierarchyTree
 union all
select (-1000)*l.preset_id + p.store_id preset_id, l.preset_id pid, p.name_, t.tree_id, p.store_id, null
preset_name, p.name_ store_name
  from StoresHierarchyLink l
  join StoresHierarchyTree t on l.preset_id = t.preset_id
  join Stores p on l.store_id = p.store_id;
commit;
-- а теперь сами продажи по магазинам
with v_obj_tree as (
with recursive tree_full_names as (
select pl.preset_id::varchar, pl.pid::varchar, ARRAY[name_]::varchar as path
  from v_obj_hierarchy pl
 where tree_id = 1
   and pl.pid is null
 union all
select t2.preset_id::varchar, t2.pid::varchar, (cte.path ||'/'|| t2.name_)::varchar
  from v_obj_hierarchy as t2
  join tree_full_names cte on cte.preset_id = t2.pid::varchar
),
nice_tree as (select preset_id, pid, level lvl, row_number() over() order_
                                  from connectby('v_obj_hierarchy', 'preset_id', 'pid', '1', 0)
                                               as t(preset_id integer, pid integer, level int)
                                )
select t.*, v.store_id, preset_name, store_name, tfn.path store_full_name
  from nice_tree t
            join tree_full_names tfn on t.preset_id::varchar = tfn.preset_id
            join v_obj_hierarchy v on t.preset_id = v.preset_id
 order by order_
),
tt as (select preset_id, pid, sum(s.qty) qty
                                  from v_obj_tree v
                                  left join sales s on v.store_id = s.store_id
                                 group by preset_id, pid
                                ),
sls as (
with recursive tree_sales as (
```

```
select pl.preset_id, pl.pid, qty
  from tt pl
 where qty is not null
 union all
select t2.preset_id, t2.pid, cte.qty qty
  from tt t2
  join tree_sales cte on cte.pid = t2.preset_id
)
select * from tree_sales),
presets_sum as (select preset_id, sum(qty) qty_sum from sls group by preset_id order by preset_id)
select store_full_name product, qty_sum total_sum
  from v_obj_tree v
  join presets_sum p on v.preset_id = p.preset_id
 order by order_;
```