МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

“Санкт-Петербургский национальный исследовательский университет информационных технологий механики и оптики”

Итоговый отчет

По дисциплине: “Хранилища данных”

Выполнил студент группы №M3314:

Стриженков Георгий Романович

САНКТ ПЕРЕРБУРГ
2025

Лабораторная работа №1. Разработка базы данных «Филиал» как источника данных.

В данной лабораторной работе требуется разработать прототип базы данных филиала торгового предприятия. База данных будет использована в дальнейшем как источник данных для наполнения хранилища данных.
Разрабатываемый прототип должен обеспечить хранение информации о следующих основных сущностях:

1. Покупатель;
2. Товар;
3. Сделка.

Информацию о покупателях можно ограничить уникальным идентификатором, а так же названием покупателя для обеспечения удобства работы с информацией.
Сущность Товар должна содержать следующие минимальные поля: уникальный номер товара, название товара, цена товара по каталогу. Следует отметить что каждый товар принадлежит к одной или нескольким категориям. Таким образом следует создать дополнительную сущность Категория, и обеспечить связь многие ко многим с сущностью Товар.

Сущность Сделка должна отражать информацию о факте покупки товара покупателем. Требуется отметить что покупатель за одну покупку может купить несколько наименований товара в разном количестве, а так же информацию о цене за которую был куплен товар (она может отличатся от заявленной цены, например с учетом скидки), а так же общую сумму сделки.

Целесообразно сделать связь между сущностями Сделка и Товар через промежуточную сущность содержащую информацию о количестве купленного товара и цена. Сущность Сделка будет содержать итоговую сумму покупки, а так же дополнительную информацию, например дату совершения сделки.

На основании имеющейся информации необходимо создать реляционную модель данных, и получить утвердить модель у преподавателя.
На основании имеющейся реляционной модели создать даталогическую модель, в соответствии с требованиями типов данных используемых на конкретной версии SQL Server, и так же утвердить у преподавателя.

На основании данных моделей создать скрипт создания базы данных, с учетом следующих требований:

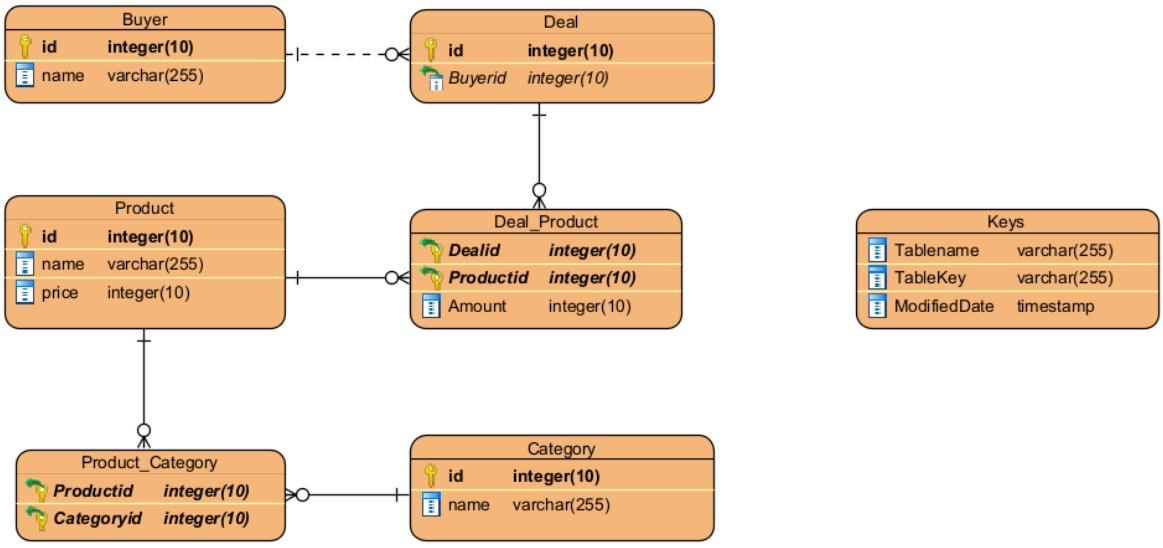
1. Каждая таблица должна иметь суррогатный целочисленный первичный ключ;
2. Каждая таблица, помимо атрибутов необходимых для хранения данных в соответствии с заданиями, должны содержать атрибут rowguid – уникальный идентификатор в рамках всей системы, и ModifiedDate - дата/время добавления или последней модификации строки;
3. Все связи должны быть реализованы как дополнительные изменения таблицы после их создания;
4. База данных должна содержать минимально необходимое количество ограничений и триггеров необходимых для стабильной работы;
5. Рекомендовано использовать английский язык для названий сущностей, атрибутов и других объектов БД.

Проверить работоспособность созданного скрипта и убедится в стабильности его работы.

Подготовить два комплекта операций вставки в базу данных обеспечивающих наполнение базы необходимыми для тестирования данными. Объем данных должен быть минимум двадцать пять строк для базовых сущностей, и минимум пятьдесят строк для сущностей реализующих связь.
Создать две базы данных по разработанному скрипту, и наполнить соответствующим набором данных. Созданные базы данных обозначить как «Филиал Запад» и «Филиал Восток».

Реляционная модель данных:

Тут представлены 4 основных таблиц — сущностный, 2 таблицы служат для связей многие ко многим. Еще 1 таблица для учета хранения даты изменения определенной таблицы и прочей информации.



Создание основных таблиц через скрипты:

```
CREATE TABLE Buyer(id INTEGER PRIMARY KEY, name VARCHAR(255) NOT NULL);

CREATE TABLE Deal(id INTEGER PRIMARY KEY, buyerId INTEGER);

ALTER TABLE Deal
ADD CONSTRAINT buyer_id FOREIGN KEY (buyerId)
REFERENCES Buyer(id);

CREATE TABLE Category(id INTEGER PRIMARY KEY, name VARCHAR(255) NOT NULL);

CREATE TABLE Product(id INTEGER PRIMARY KEY, name VARCHAR(255) NOT NULL, price INTEGER NOT NULL)

CREATE TABLE Deal_Product(dealId INTEGER, productId INTEGER, amount INTEGER NOT NULL);

ALTER TABLE Deal_Product
ADD CONSTRAINT deal_id FOREIGN KEY (dealId)
REFERENCES Deal(id);
ALTER TABLE Deal_Product
ADD CONSTRAINT product_id FOREIGN KEY (productId)
REFERENCES Product(id);

CREATE TABLE Product_Category(productId INTEGER, CategoryId INTEGER);

ALTER TABLE Product_Category
ADD CONSTRAINT product_id FOREIGN KEY (ProductId)
REFERENCES Product(id);
ALTER TABLE Product_Category
ADD CONSTRAINT category_id FOREIGN KEY (categoryId)
REFERENCES Category(id);
```

Создание таблицы Keys для учета изменений данных:

```
CREATE TABLE Keys(tablename VARCHAR(255) NOT NULL,
    rowGuid VARCHAR(255) NOT NULL DEFAULT uuid_generate_v4(),
    modifiedDate TIMESTAMP NOT NULL DEFAULT NOW()
);
INSERT INTO Keys(tablename) VALUES ('Buyer');
INSERT INTO Keys(tablename) VALUES ('Deal');
INSERT INTO Keys(tablename) VALUES ('Category');
INSERT INTO Keys(tablename) VALUES ('Product');
INSERT INTO Keys(tablename) VALUES ('Deal_Product');
INSERT INTO Keys(tablename) VALUES ('Product_Category');
```

	tablename character varying (255)	rowguid character varying (255)	modifieddate timestamp without time zone
1	Buyer	da2784da-3e7e-4e69-b1a4-32f07cb9505e	2024-09-25 09:14:35.033585
2	Deal	e93799e2-fc4b-4e28-a16d-8a9e27dd0874	2024-09-25 09:14:35.033585
3	Product	359f11aa-7391-4c3c-9a4d-8ae126067d89	2024-09-25 09:14:35.033585
4	Category	fd02776c-da39-400e-83b7-9c5eb8eca99f	2024-09-25 09:14:35.033585
5	Product_Category	b7a4ab2c-deeb-40a8-8583-aa3dd42aa9fc	2024-09-25 09:14:35.033585
6	Deal_Product	131b006b-b4cd-4338-8f5e-4b30d2752a43	2024-09-25 09:14:35.033585

Пример скрипта для учета изменения данных с одной таблицей:

Для всех 6 таблиц они выглядят схожим образом, различаются всего лишь названия колонок и таблиц.

```
CREATE OR REPLACE FUNCTION buyer_trigger_set_timestamp()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE Keys SET modifiedDate = Now() WHERE tablename = 'Buyer';
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER buyer_set_timestamp
AFTER UPDATE OR INSERT OR DELETE ON Buyer
FOR EACH ROW
EXECUTE PROCEDURE buyer_trigger_set_timestamp();
```

Добавление данных в таблицу:

В каждую из 4 основных таблиц минимум добавляется по 25 строк данных, а в таблицы реализующие связи минимум 50.

```
INSERT INTO Buyer(id, name) VALUES (1, 'Bill');
INSERT INTO Buyer(id, name) VALUES (2, 'Alex');
INSERT INTO Buyer(id, name) VALUES (3, 'Fred');
INSERT INTO Buyer(id, name) VALUES (4, 'Bob');
INSERT INTO Buyer(id, name) VALUES (5, 'Mike');
INSERT INTO Buyer(id, name) VALUES (6, 'Kate');
INSERT INTO Buyer(id, name) VALUES (7, 'Victor');
INSERT INTO Buyer(id, name) VALUES (8, 'Albert');
INSERT INTO Buyer(id, name) VALUES (9, 'Poly');
INSERT INTO Buyer(id, name) VALUES (10, 'Pol');
INSERT INTO Buyer(id, name) VALUES (11, 'Nick');
INSERT INTO Buyer(id, name) VALUES (12, 'Han');
INSERT INTO Buyer(id, name) VALUES (13, 'Robert');
INSERT INTO Buyer(id, name) VALUES (14, 'Joe');
INSERT INTO Buyer(id, name) VALUES (15, 'Donald');
INSERT INTO Buyer(id, name) VALUES (16, 'Dona');
INSERT INTO Buyer(id, name) VALUES (17, 'Ema');
INSERT INTO Buyer(id, name) VALUES (18, 'Emely');
INSERT INTO Buyer(id, name) VALUES (19, 'Tod');
INSERT INTO Buyer(id, name) VALUES (20, 'Bart');
INSERT INTO Buyer(id, name) VALUES (21, 'Molly');
INSERT INTO Buyer(id, name) VALUES (22, 'Andrew');
INSERT INTO Buyer(id, name) VALUES (23, 'John');
INSERT INTO Buyer(id, name) VALUES (24, 'Tyler');
INSERT INTO Buyer(id, name) VALUES (25, 'Pet');
```

Итог (Фрагмент данных из Buyer):

	id [PK] integer	name character varying (255)
1	1	Bill
2	2	Alex
3	3	Fred
4	4	Bob
5	5	Mike
6	6	Kate
7	7	Victor
8	8	Albert
9	9	Poly
10	10	Pol
11	11	Nick
12	12	Han
13	13	Robert
14	14	Joe
15	15	Donald
16	16	Dona

Лабораторная работа №2. Разработка модели и скрипта создания хранилища данных Хранилище данных будет реализовано с использованием реляционной СУБД. Наполнение будет производится в автоматизированном режиме из данных поступающих из источников разработанных в лабораторной работе №1. Для хранилища данных предусмотрена только операция добавления данных, операции модификации и удаления не допустимы.

Основными сущностями в хранилище данных будут:

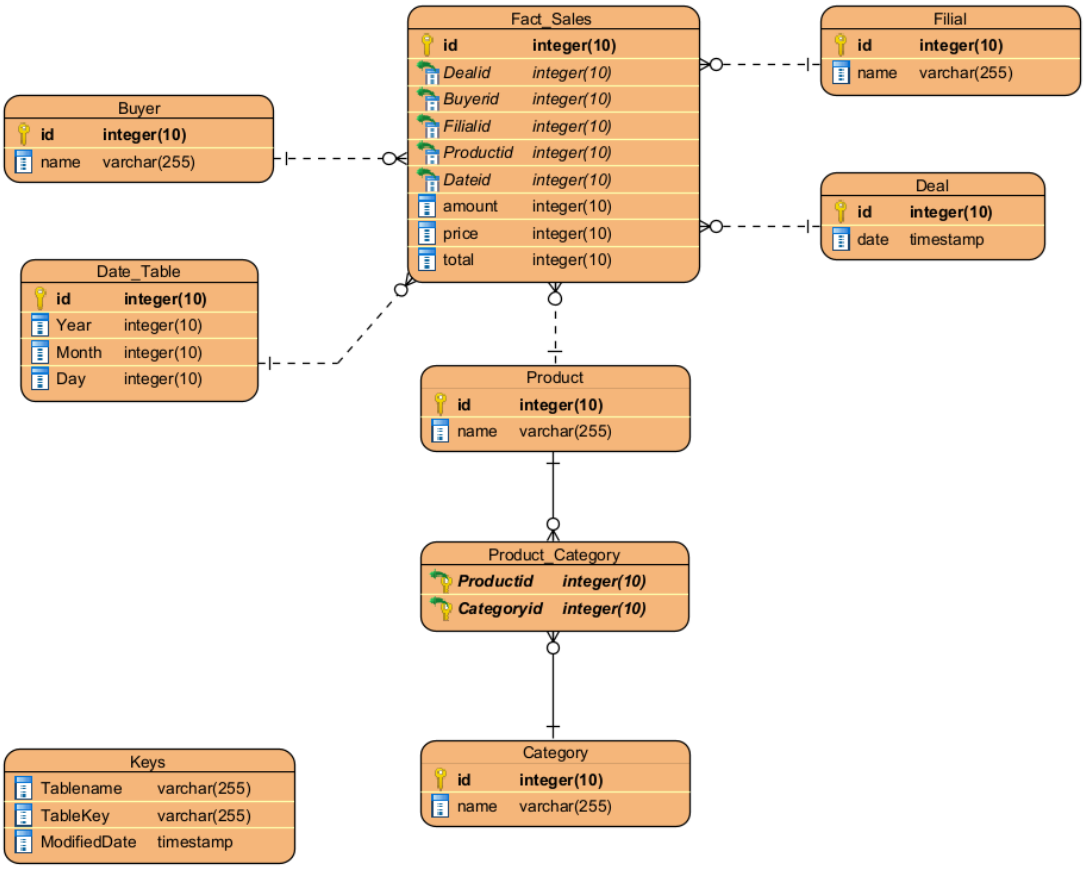
1. Филиал;
2. Покупатель;
3. Товар;
4. Сделки.

Сущность Филиал содержит информацию о филиале из которого поступили данные.
Сущность Товар содержит информацию о товаре идентичную информации содержащейся в базе данных Филиалы, а так же дополнительную информацию полученную в результате операции обогащения: номер филиала, дату поступления информации. Так как в существующих филиалах идентификация товаров осуществляется по внутренним уникальным ключам, необходимо хранить информацию об этих ключах, а так же дополнительные ключи, обеспечивающие уникальную идентификацию в Хранилище данных. Сущность Покупатель формируется аналогичным способом.
Сущность сделка содержит всю информацию о покупке конкретным покупателем конкретного филиала той или иной номенклатуры товаров в определенном количестве по определенной цене. Таким образом на каждый факт покупки должна хранится следующая информация: уникальный номер товара, по нумерации Хранилища данных, уникальный номер покупателя по нумерации Хранилища данных, количество товаров, цена по прайсу, цена сделки, дата сделки, номер сделки, номер филиала, дата поступления информации.

Так же может потребоваться хранить дополнительную информацию, такую как категория товаров, номера сделок, суммы по сделкам. Для хранения этой информации необходимо создать дополнительные сущности, с уникальными идентификационными номерами в рамках системы, а так же с уникальными номерами пришедшими из источников, дате поступления информации.
На основании имеющейся информации создать реляционную модель, утвердить ее у преподавателя. На основании утвержденной модели создать даталогическую модель, утвердить ее у преподавателя.

На основании созданных моделей создать скрипт создания хранилища данных, проверить его работоспособность. Создать тестовый (!) скрипт наполнения хранилища, проверить работоспособность.

Даталогическая модель:



Создание таблицы фактов продаж. Данная таблицы имеет связи с 5 таблицами измерений: Buyer, Deal, Product, Filial, Date_Table.

```
CREATE TABLE Fact_Sales(id INTEGER PRIMARY KEY, dealId INTEGER, buyerId INTEGER, filialId INTEGER, productId INTEGER, dateId

ALTER TABLE Fact_Sales
ADD CONSTRAINT deal_id FOREIGN KEY (dealId)
REFERENCES Deal(id);
ALTER TABLE Fact_Sales
ADD CONSTRAINT buyer_id FOREIGN KEY (buyerId)
REFERENCES Buyer(id);
ALTER TABLE Fact_Sales
ADD CONSTRAINT filial_id FOREIGN KEY (filialId)
REFERENCES Filial(id);
ALTER TABLE Fact_Sales
ADD CONSTRAINT product_id FOREIGN KEY (productId)
REFERENCES Product(id);
ALTER TABLE Fact_Sales
ADD CONSTRAINT date_id FOREIGN KEY (dateId)
REFERENCES Date_Table(id);

CREATE OR REPLACE FUNCTION fact_sales_trigger_set_timestamp()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE Keys SET modifiedDate = Now() WHERE tablename = 'Fact_Sales';
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER fact_sales_set_timestamp
AFTER UPDATE OR INSERT OR DELETE ON Fact_Sales
FOR EACH ROW
EXECUTE PROCEDURE fact_sales_trigger_set_timestamp();
```

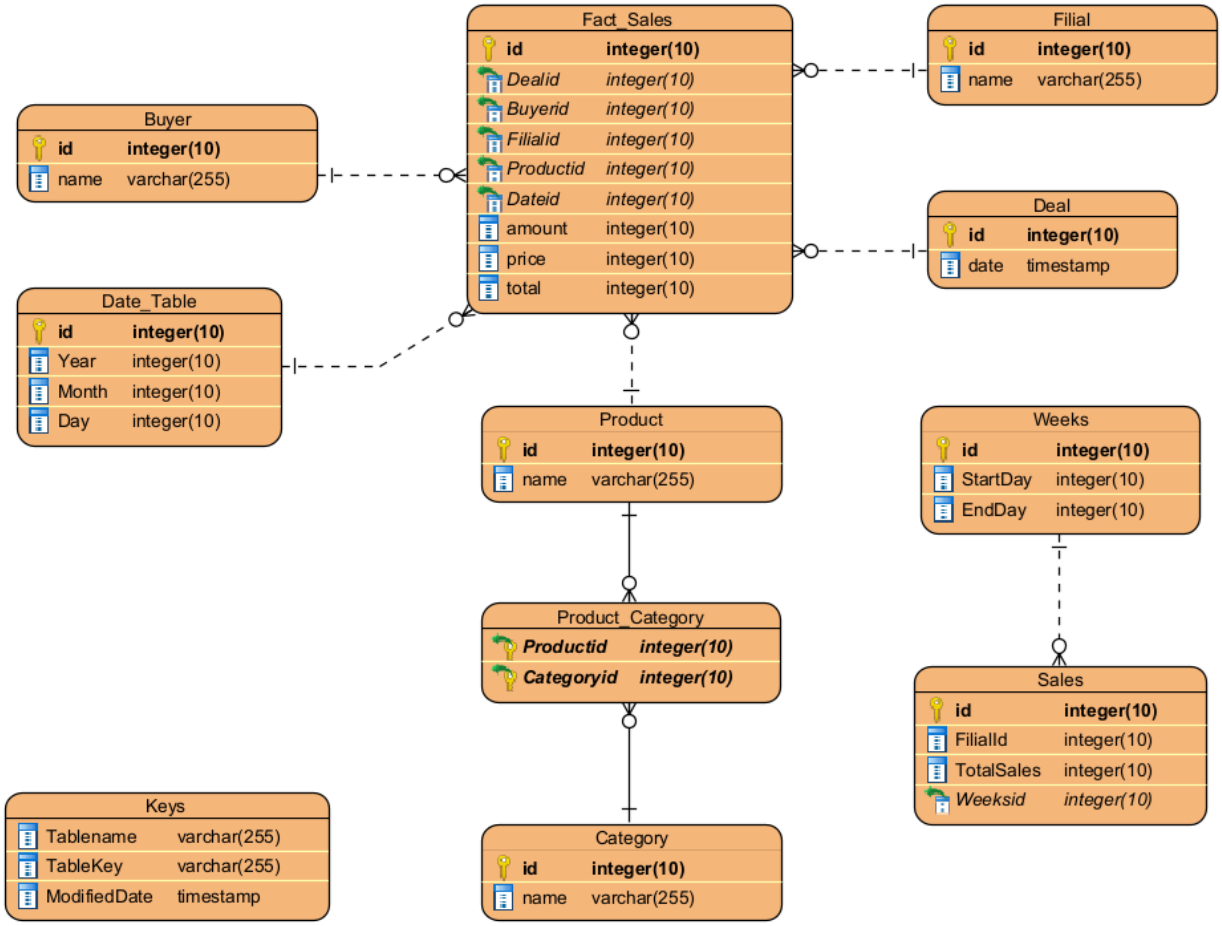
Результат наполнения таблицы фактов:

	id [PK] integer	dealid integer	buyerid integer	filialid integer	productid integer	dateid integer	amount integer	price integer	total integer
1	1	1	1	2	1	1	1	60	60
2	2	2	12	2	3	2	2	55	110
3	3	3	19	1	2	3	1	40	40
4	4	4	24	1	24	4	2	300	600
5	5	5	17	2	25	5	1	600	600

Лабораторная работа №3. Создание витрины данных.
Витрина данных представляет собой специализированное хранилище где данные консолидируются по определенному признаку. В данной работе требуется консолидировать результаты продаж по неделям. Это обеспечит уменьшение нагрузки на основное Хранилище данных.
Основными сущностями Витрины будут:

1. Недели;
2. Продажи.

Сущность недели хранит информацию о временных промежутках, за которые была произведена консолидация данных. Сущность Продажи обеспечивает хранение информации о результатах продаж в Филиале в соответствующую неделю.



```sql
CREATE TABLE Weeks (id INTEGER PRIMARY KEY, startDay INTEGER NOT NULL, endDay INTEGER NOT NULL);

CREATE TABLE Sales (id INTEGER PRIMARY KEY, weekId INTEGER NOT NULL, filialId INTEGER NOT NULL, totalSales INTEGER NOT NULL);
ALTER TABLE Sales
ADD CONSTRAINT week_id FOREIGN KEY (weekId)
REFERENCES Weeks(id);

INSERT INTO Weeks(id, startDay, endDay) VALUES (1, 1, 7);
INSERT INTO Weeks(id, startDay, endDay) VALUES (2, 8, 14);

INSERT INTO Sales(id, weekId, filialId, totalSales) VALUES (1, 1, 1, 500);
INSERT INTO Sales(id, weekId, filialId, totalSales) VALUES (2, 1, 2, 2500);
INSERT INTO Sales(id, weekId, filialId, totalSales) VALUES (3, 2, 1, 1000);

CREATE VIEW WeeklySales as
SELECT w.id, w.startDay, w.endDay, s.filialId, SUM(s.totalSales) as consolidatedSales FROM Weeks w
JOIN Sales s ON w.id = s.weekId
GROUP BY w.id, w.startDay, w.endDay, s.filialId;
```

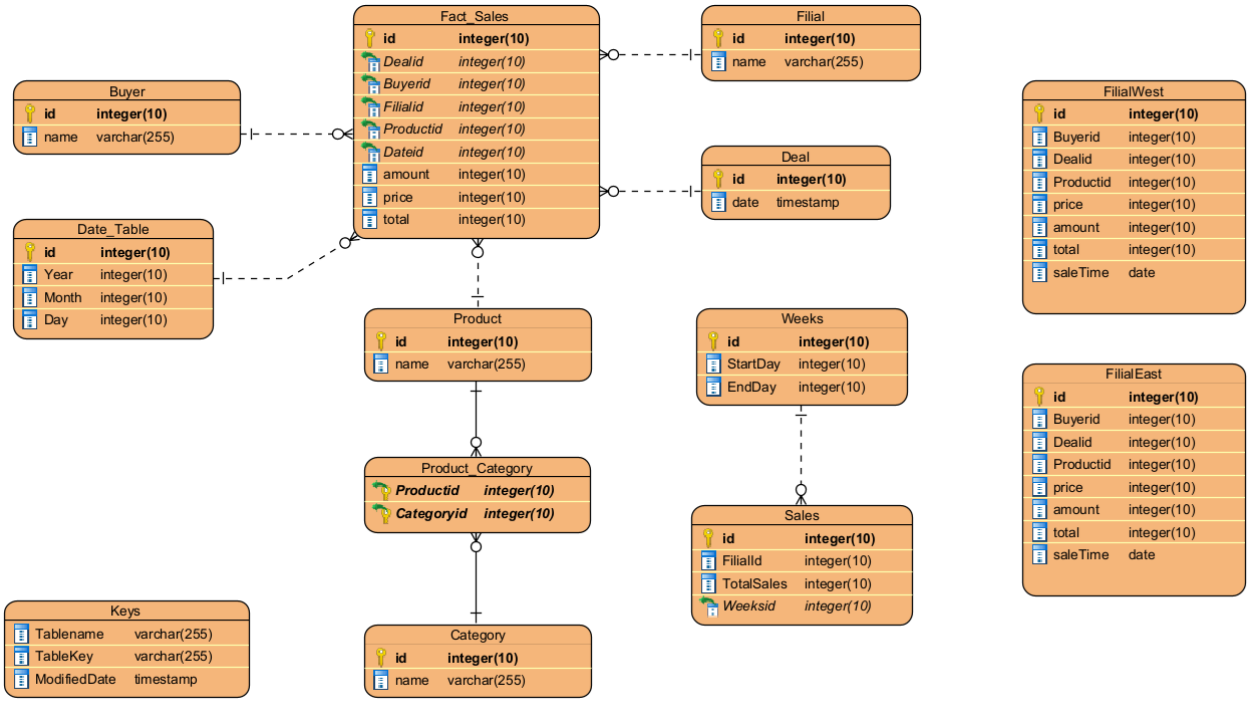
```
1  SELECT * FROM public.weeklysales
2
```

Data Output | Messages | Notifications

	id integer	startday integer	endday integer	filialid integer	consolidatedsales bigint
1	1	1	7	1	500
2	1	1	7	2	2500
3	2	8	14	1	1000

Лабораторная работа №4. Создания процедуры наполнения Хранилища данных.
Цель работы: создать хранимую процедуру обеспечивающую автоматизированный перенос данных из Филиалов в Хранилище. Процедура должна обеспечить добавление вновь поступившей информации из Филиала Запад, и Филиала Восток в центральное хранилище данных. Процедура должна исключить дублирование информации. Разработчик должен продумать механизм обеспечивающий минимализацию нагрузки на продуктивные сервера филиалов. В процедуре можно использовать операции множественной вставки.

Даталогическая модель:



Вызов функции переноса, которая переносит несколько последних строк данных из филиала, будет осуществляться при добавлении каждого набора данных в один из филиалов (то есть можно сделать множественную вставку или одиночную), что позволит минимизировать количество вызовов функции и сделает необходимое число вставок. Также осуществляется одновременная запись и парсинг набора данных времени в вспомогательную таблицу.

Все структуры/тригеры/функции/принципы поведения двух филиалов и связанных с ними функций одинаковы, но отличаются лишь названия.

Структура основных таблиц:

```sql
CREATE TABLE Filial_East (
    id SERIAL PRIMARY KEY,
    buyerId INT NOT NULL,
    dealId INT NOT NULL,
    productId INT NOT NULL,
    price INT NOT NULL,
    amount INT NOT NULL,
    total INT NOT NULL,
    saleTime DATE
);

CREATE TABLE Date_Table(
    id SERIAL PRIMARY KEY,
    parseYear INT,
    parseMonth INT,
    parseDay INT
);

CREATE TABLE Fact_Sales (
    Id SERIAL PRIMARY KEY,
    filialId INT NOT NULL,
    buyerId INT NOT NULL,
    dealId INT NOT NULL,
    productId INT NOT NULL,
    price INT NOT NULL,
    amount INT NOT NULL,
    saleTime Date,
    filialName VARCHAR(255),
    saleTimeId SERIAL,
    FOREIGN KEY (SaleTimeId) REFERENCES Date_Table(id)
);
```

Пример функции вставки для филиала запад:

```
CREATE OR REPLACE FUNCTION UpdateFactSalesWest()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Date_Table(parseYear, parseMonth, parseDay)
    SELECT parseYear, parseMonth, parseDay
    FROM (
        SELECT
            Filial_West.id as tempId,
            CAST(EXTRACT(YEAR FROM saleTime) AS INT) as parseYear,
            CAST(EXTRACT(MONTH FROM saleTime) AS INT) as parseMonth,
            CAST(EXTRACT(DAY FROM saleTime) AS INT) as parseDay
        FROM Filial_West ORDER BY id DESC
        LIMIT (SELECT id FROM Filial_West ORDER BY id DESC LIMIT 1) -
        (SELECT filialid FROM Fact_Sales WHERE filialName = 'West' ORDER BY filialid DESC LIMIT 1)
    ) AS subquery ORDER BY tempId ASC;

    INSERT INTO Fact_Sales(filialId, productId, buyerId, dealId, price, amount, saleTime, filialName)
    SELECT id, productId, buyerId, dealId, price, amount, saleTime, 'West'
    FROM (
        SELECT
            id, productId, buyerId, dealId, price, amount, saleTime, 'West'
        FROM Filial_West ORDER BY id DESC
        LIMIT (SELECT id FROM Filial_West ORDER BY id DESC LIMIT 1) -
        (SELECT filialid FROM Fact_Sales WHERE filialName = 'West' ORDER BY filialid DESC LIMIT 1)
    ) AS subquery ORDER BY id ASC;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Вставка происходит в 2 этапа:
1) Запись определенного числа строк (только новые) в таблицу времени предварительно отпарсив их.
2) Запись определенного числа строк (только новые) основных данных о подробностях определенных покупок в хранилище из филиала запад.
Нужное добавление количества строк считается следующим образом:
Берется разница между текущим количеством строк данных в филиале и номером самой последней строки данных из конкретного филиала, так как у нас есть двойная индексация в хранилище. Берется нужное количество строк с конца филиала и записывается в обратном порядке для соблюдения удобных вторых индексов в хранилище.

Тригер для вызова:

```
CREATE OR REPLACE TRIGGER FilialWestInsert
AFTER INSERT ON Filial_West
FOR EACH STATEMENT
EXECUTE FUNCTION UpdateFactSalesWest();
```

Вызов происходит единожды (FOR EACH STATEMENT) при добавлении новой строчки/набора строк в таблицу, что позволяет избежать множественных вызовов функции при добавлении целого набора данных множественной вставкой.

Пример работы с 3 операциями вставки:

```
INSERT INTO Filial_West (buyerId, dealId, productId, price, amount, total, saleTime)
VALUES (1, 1, 10, 100, 2, 200, '2023-01-01'),
       (8, 2, 20, 50, 1, 45, '2023-02-02'),
       (9, 3, 11, 40, 5, 200, '2023-02-05'),
       (21, 4, 20, 70, 3, 200, '2023-03-14'),
       (21, 4, 1, 65, 4, 260, '2023-04-11');

INSERT INTO Filial_East (buyerId, dealId, productId, price, amount, total, saleTime)
VALUES (4, 1, 1, 20, 2, 40, '2023-01-01'),
       (17, 2, 5, 80, 1, 80, '2023-02-02'),
       (19, 3, 8, 55, 1, 55, '2023-02-01'),
       (1, 4, 9, 90, 3, 250, '2023-02-17'),
       (20, 5, 13, 65, 3, 180, '2023-03-05');

INSERT INTO Filial_West (buyerId, dealId, productId, price, amount, total, saleTime) VALUES (20, 20, 1, 100, 2, 195, '2023-01
```

Содержимое хранилища и таблицы времени после работы функций соответственно:

Data Output | Messages | Notifications

	id [PK] integer	filialid integer	buyerid integer	dealid integer	productid integer	price integer	amount integer	saletime date	filialname character varying (255)	saletimeid integer
1	1	1	1	1	10	100	2	2023-01-01	West	1
2	2	2	8	2	20	50	1	2023-02-02	West	2
3	3	3	9	3	11	40	5	2023-02-05	West	3
4	4	4	21	4	20	70	3	2023-03-14	West	4
5	5	5	21	4	1	65	4	2023-04-11	West	5
6	6	1	4	1	1	20	2	2023-01-01	East	6
7	7	2	17	2	5	80	1	2023-02-02	East	7
8	8	3	19	3	8	55	1	2023-02-01	East	8
9	9	4	1	4	9	90	3	2023-02-17	East	9
10	10	5	20	5	13	65	3	2023-03-05	East	10
11	11	6	20	20	1	100	2	2023-01-01	West	11

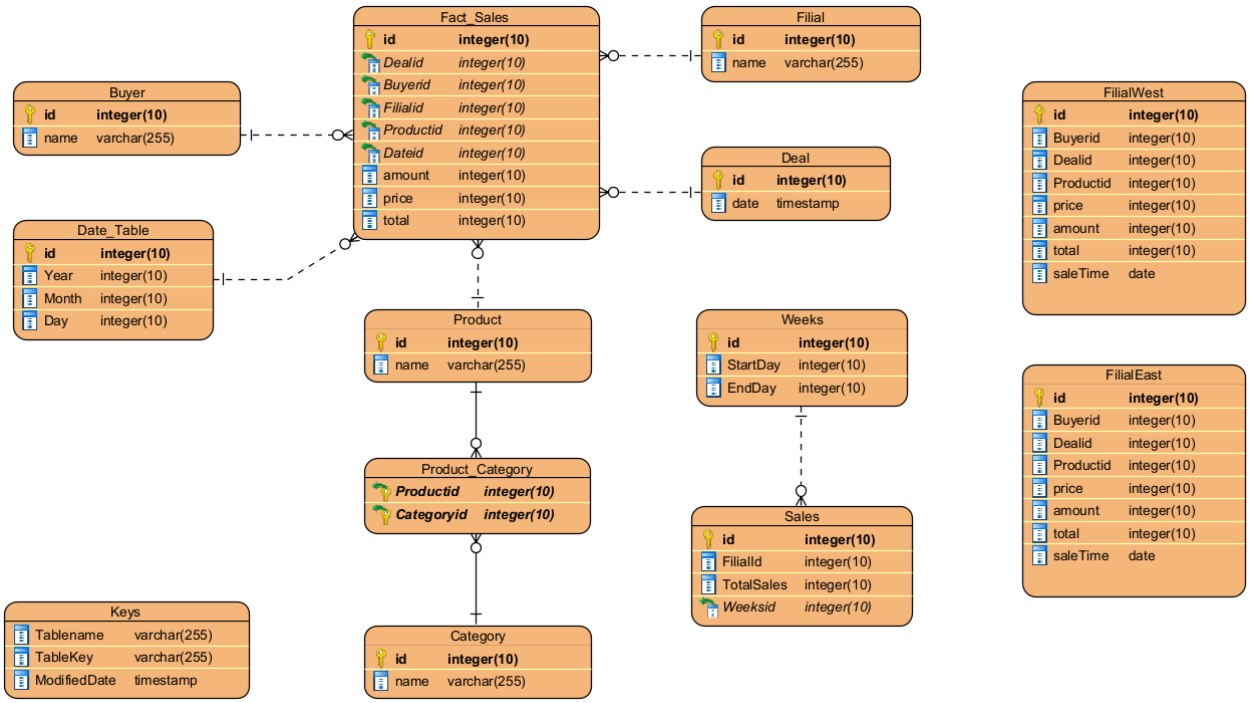
	id [PK] integer	parseyear integer	parsemonth integer	parseday integer
1	1	2023	1	1
2	2	2023	2	2
3	3	2023	2	5
4	4	2023	3	14
5	5	2023	4	11
6	6	2023	1	1
7	7	2023	2	2
8	8	2023	2	1
9	9	2023	2	17
10	10	2023	3	5
11	11	2023	1	1

По итого автоматическая вставка работает успешно, причем функция вызывается минимальное количество раз, в данном случае 3.

Лабораторная работа №5. Создание Процедуры наполнения Витрины данных.

Цель работы: разработать скрипт, или хранимую процедуру которая обеспечит наполнение Витрины данных на основании детальных данных имеющихся в Хранилище. Витрина данных наполняется данными за определенный период, например за пять недель выбранных пользователем.

Даталогическая модель:



Скрипт для создания витрины данных:

```sql
CREATE OR REPLACE VIEW SALES_FROM_FACT AS

WITH const AS (
    SELECT 2023 AS CurYear, 3 AS CurMonth, 10 AS CurDay
)

SELECT f.id, filialid, buyerid, dealid, productid, price, amount, saletime, filialname
FROM fact_sales as f
    JOIN date_table as d ON f.saletimeid = d.id
WHERE (parseyear < (SELECT CurYear from const))
OR (parseyear = (SELECT CurYear from const) AND parsemonth < (SELECT CurMonth from const))
OR (parseyear = (SELECT CurYear from const) AND parsemonth = (SELECT CurMonth from const) AND parseday < (SELECT CurDay from cor
ORDER BY f.id ASC
```

В верхней части скрипта создаются константы для выбора границы данных для переноса, в данном случае данные ограничены с одной стороны, но можно и сделать ограничение со второй.
В качестве данных записываются данные из хранилища, которые были выбраны путем слияния с таблицей конвертированных дат (деление на год/месяц/день). В витрине будет убрана последний столбик, так как он имеет лишние данные.

Содержимое хранилища:

	id [PK] integer	filialid integer	buyerid integer	dealid integer	productid integer	price integer	amount integer	saletime date	filialname character varying (255)	saletimeid integer
1	1	1	1	1	10	100	2	2023-01-01	West	1
2	2	2	8	2	20	50	1	2023-02-02	West	2
3	3	3	9	3	11	40	5	2023-02-05	West	3
4	4	4	21	4	20	70	3	2023-03-14	West	4
5	5	5	21	4	1	65	4	2023-04-11	West	5
6	6	1	4	1	1	20	2	2023-01-01	East	6
7	7	2	17	2	5	80	1	2023-02-02	East	7
8	8	3	19	3	8	55	1	2023-02-01	East	8
9	9	4	1	4	9	90	3	2023-02-17	East	9
10	10	5	20	5	13	65	3	2023-03-05	East	10
11	11	6	20	20	1	100	2	2023-01-01	West	11

Содержимое витрины (2 строчки данных не прошли фильтр):

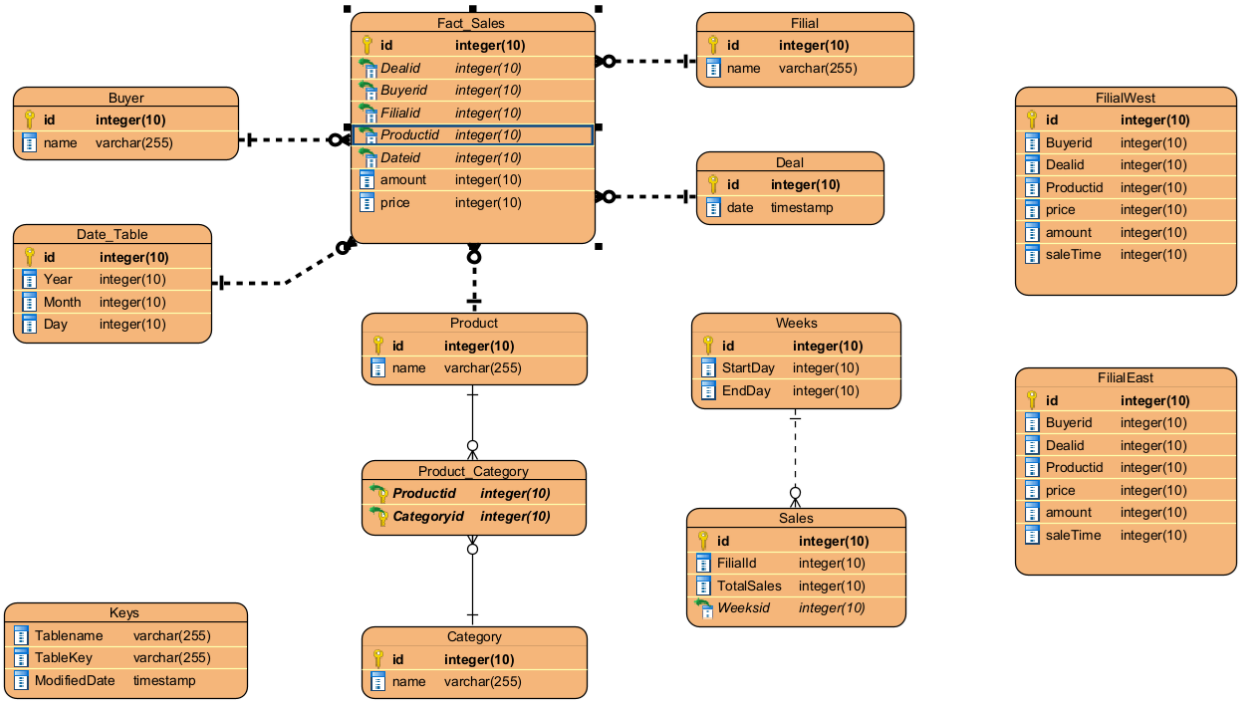
	id integer	filialid integer	buyerid integer	dealid integer	productid integer	price integer	amount integer	saletime date	filialname character varying (255)
1	1	1	1	1	10	100	2	2023-01-01	West
2	2	2	8	2	20	50	1	2023-02-02	West
3	3	3	9	3	11	40	5	2023-02-05	West
4	6	1	4	1	1	20	2	2023-01-01	East
5	7	2	17	2	5	80	1	2023-02-02	East
6	8	3	19	3	8	55	1	2023-02-01	East
7	9	4	1	4	9	90	3	2023-02-17	East
8	10	5	20	5	13	65	3	2023-03-05	East
9	11	6	20	20	1	100	2	2023-01-01	West

Лабораторная работа №6. Создание скрипта восстановления утраченных филиалом данных на основании детальных данных Хранилища.

Цель работы: Создание скрипта обеспечивающего обратный перенос очищенных данных из Хранилища в выбранный Филиал.
Скрипт можно оформить как набор SQL команд, или как хранимую процедуру.

Скрипт должен переносить данные за выбранный промежуток времени из Хранилища в Филиал. Скрипт должен работать исходя из того что данные в Филиале полностью отсутствуют.

Даталогическая модель:



Скрипт для восстановления данных в филиале, используя данные из хранилища:

```
INSERT INTO filial_west(id, buyerid, dealid, productid, price, amount, saletime)

WITH const AS (
    SELECT 2023 AS StartYear, 2 AS StartMonth, 1 AS StartDay,
    2023 AS EndYear, 3 AS EndMonth, 15 AS EndDay
)

SELECT filialid, buyerid, dealid, productid, price, amount, saletime
FROM fact_sales as f
    JOIN date_table as d ON f.saletimeid = d.id
WHERE (filialname = 'West') AND ((parseyear < (SELECT EndYear from const))
OR (parseyear = (SELECT EndYear from const) AND parsemonth < (SELECT EndMonth from const))
OR (parseyear = (SELECT EndYear from const) AND parsemonth = (SELECT EndMonth from const) AND parseday < (SELECT EndDay from co
AND ((parseyear > (SELECT StartYear from const))
OR (parseyear = (SELECT StartYear from const) AND parsemonth > (SELECT StartMonth from const))
OR (parseyear = (SELECT StartYear from const) AND parsemonth = (SELECT StartMonth from const) AND parseday > (SELECT StartDay f
ORDER BY f.id ASC
```

Задаем временной промежуток и выбираем строчки из хранилища данных удовлетворяющих условиям:
1) Требуемый промежуток времени
2) Требуемый филиал
Данный пример восстанавливает записи в филиале Запад данные за промежуток с 2023/02/01 по 2023/03/15 не включительно.

Содержимое хранилища:

	id [PK] integer	filialid integer	buyerid integer	dealid integer	productid integer	price integer	amount integer	saletime date	filialname character varying (255)	saletimeid integer
1	1	1	1	1	10	100	2	2023-01-01	West	1
2	2	2	8	2	20	50	1	2023-02-02	West	2
3	3	3	9	3	11	40	5	2023-02-05	West	3
4	4	4	21	4	20	70	3	2023-03-14	West	4
5	5	5	21	4	1	65	4	2023-04-11	West	5
6	6	1	4	1	1	20	2	2023-01-01	East	6
7	7	2	17	2	5	80	1	2023-02-02	East	7
8	8	3	19	3	8	55	1	2023-02-01	East	8
9	9	4	1	4	9	90	3	2023-02-17	East	9
10	10	5	20	5	13	65	3	2023-03-05	East	10
11	11	6	20	20	1	100	2	2023-01-01	West	11

Содержимое филиала Запад (3 строчки данных прошли фильтр по времени и филиалу из хранилища):

	id [PK] integer	buyerid integer	dealid integer	productid integer	price integer	amount integer	saletime date
1	2	8	2	20	50	1	2023-02-02
2	3	9	3	11	40	5	2023-02-05
3	4	21	4	20	70	3	2023-03-14