




ОНЛАЙН-ОБРАЗОВАНИЕ

# Онлайн-образование





# Меня хорошо видно && слышно?

Ставьте , если все хорошо  
Напишите в чат, если есть проблемы



# Защита проекта

## Тема: Подготовка инфраструктуры для разработки хранилищ и витрин данных



Щербаков Игорь Борисович

Должность: ведущий разработчик

# Введение

Разработка выполняется на общих серверах, но часто требуется развернуть базы данных локально на своей рабочей станции.

Это может потребоваться:

- на начальных стадиях проекта, когда полноценные серверы еще не выделены,
- для проверки алгоритмов,
- для изучения возможностей средств разработки и т.д.

Важно выполнять эту работу быстро, поэтому к ней нужно подготовиться.

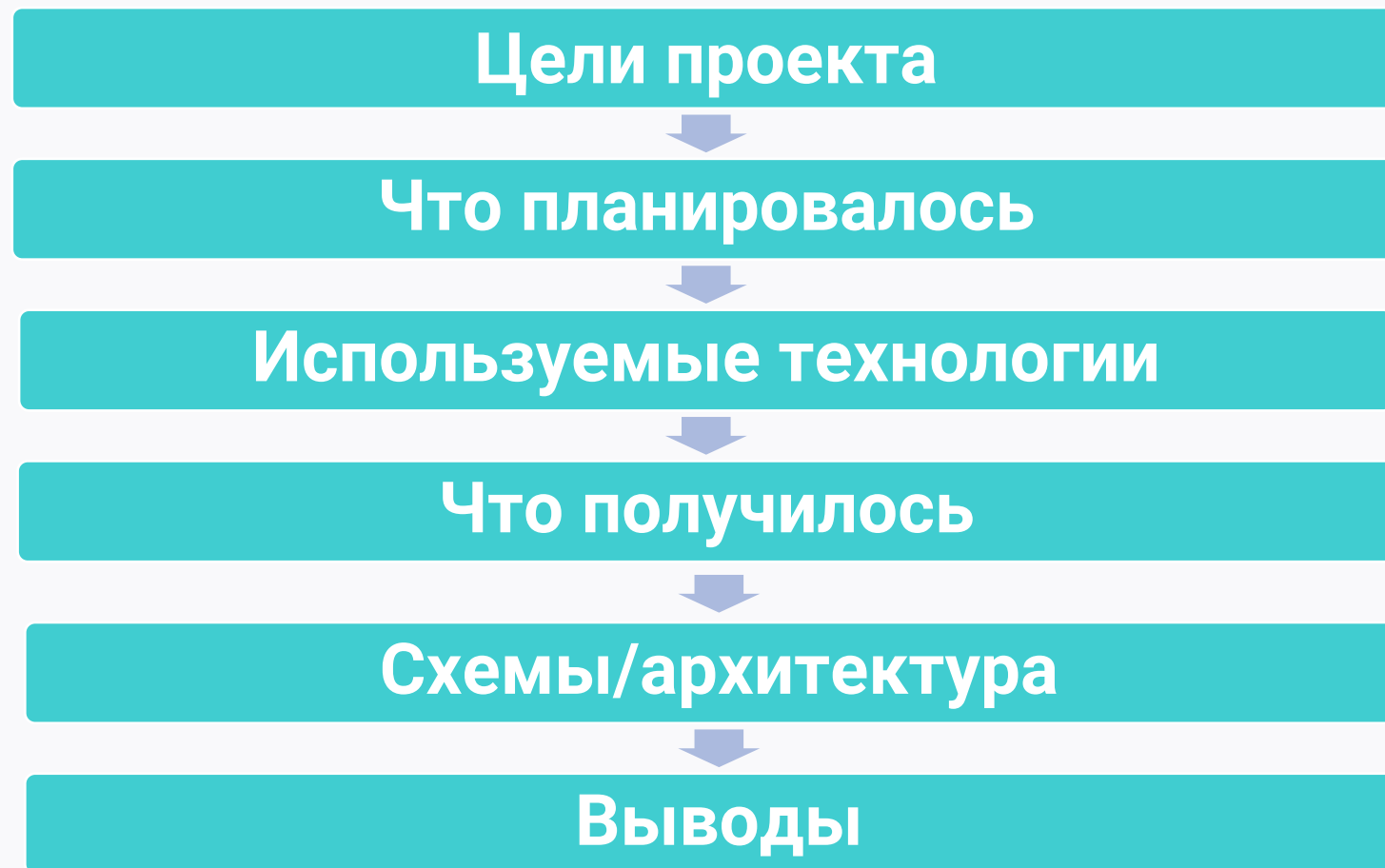
Я развернул на своей рабочей станции инфраструктуру для разработки хранилищ и витрин данных, поскольку последнее время участвовал именно в таких проектах.

Выполнил упрощенную разработку хранилища и витрин для проверки того, что все настройки выполнены правильно.

Составил подробные шпаргалки по настройкам.



# План защиты



# Цели проекта

1

Закрепление основных навыков полученных на текущем курсе

2

Создание инфраструктуры для разработки хранилищ и витрин данных, настройка отдельных компонентов

3

Создание объектов БД, заполнение таблиц тестовыми данными

# Что планировалось

- 1 Развернуть БД PostgreSQL в виртуальных машинах и docker-ax
- 2 Обеспечить взаимодействие баз данных и средств разработки
- 3 Создать упрощенные БД регистрирующих систем и хранилища данных
- 4 Написать подробные шпаргалки по настройке

без картинок

0-9



# Используемые технологии

**1** Виртуализация и виртуальные машины - VirtualBox

**2** Контейнеризация - Docker

**3** ETL - Extract, Transform, Load - извлечение, преобразование, загрузка

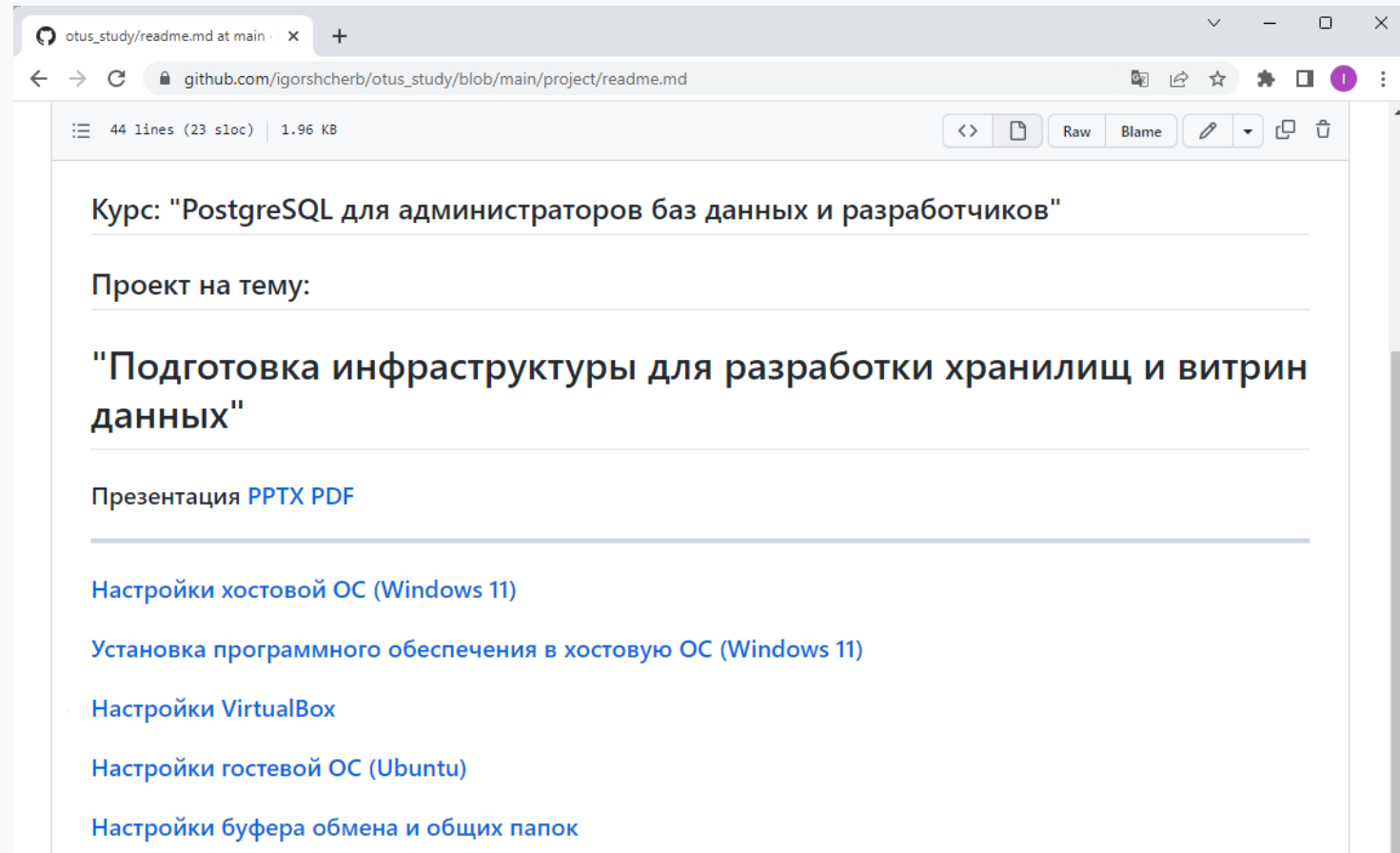




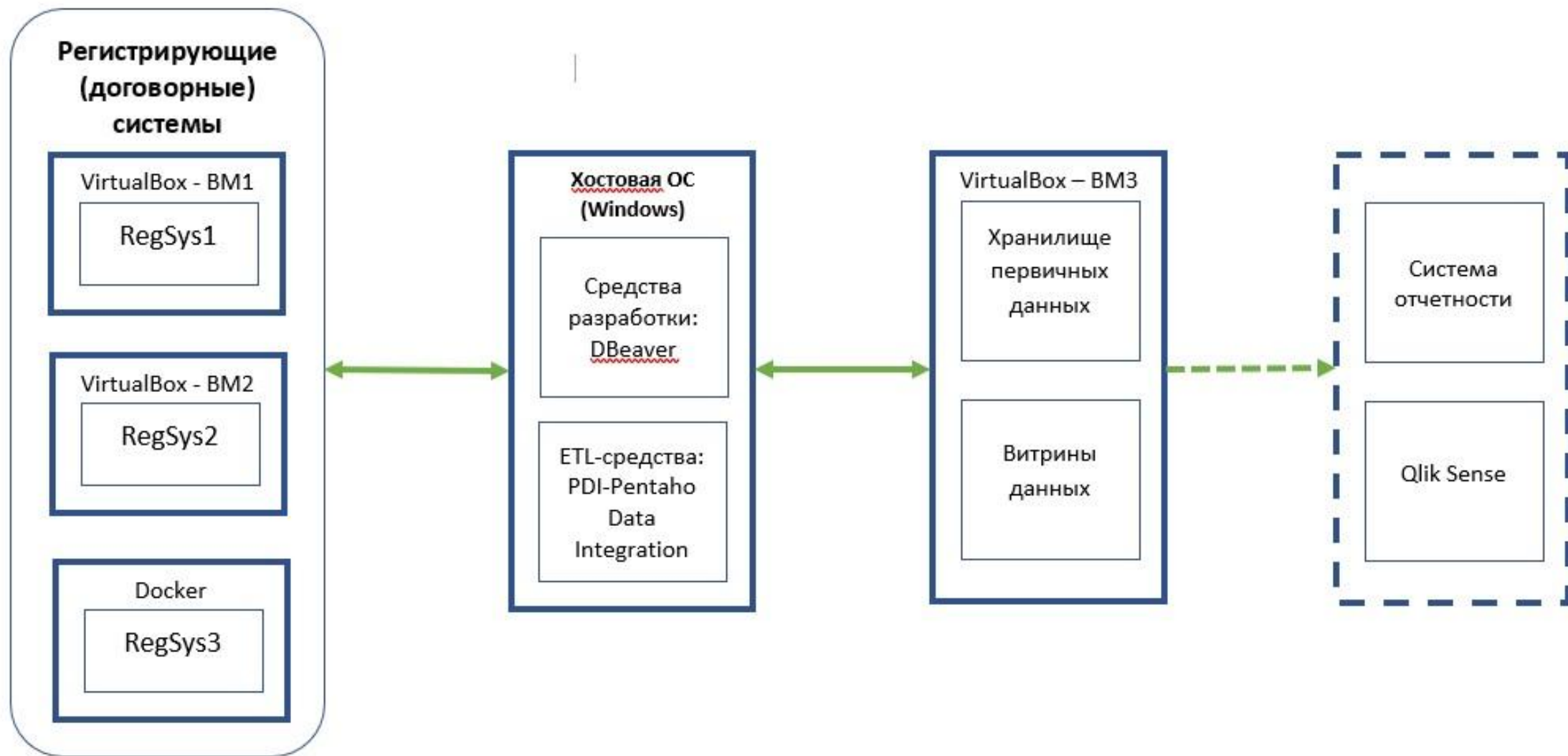
# Что получилось

Репозиторий с текущей версией проекта

[https://github.com/igorshcherb/otus\\_study/blob/main/project/readme.md](https://github.com/igorshcherb/otus_study/blob/main/project/readme.md)



# Архитектура






# Мои действия

- Выполнил настройки хостовой ОС, установил программное обеспечение: Virtual Box, средства разработки, ETL-средства.
- Выполнил настройки VirtualBox.
- Создал виртуальную машину, установил Ubuntu, выполнил настройки.
- Запустил PostgreSQL в гостевой ОС.
- Определил оптимальные параметры PostgreSQL, применил их.
- Установил программное обеспечение в гостевую ОС.
- Клонировал виртуальную машину, выполнил дополнительные настройки.
- Установил Docker с PostgreSQL, выполнил настройки.
- Выполнил настройки средств разработки и ETL-средства в хостовой ОС.
- Написал скрипты и создал объекты баз данных регистрирующих (договорных) систем.
- Написал скрипты и заполнил таблицы регистрирующих систем тестовыми данными.
- Написал скрипты и создал объекты хранилища данных.
- Написал скрипты и создал таблицы витрин.
- Сделал трансформации Pentaho и заполнил данными таблицы хранилища.
- Сделал трансформации Pentaho и заполнил данными витрины.

Подробное описание всех этих действий можно посмотреть на [github](#). Некоторые действия представлены на следующих слайдах.

# Настройка параметров БД - OLTP

 **PGTune**

### Parameters of your system

DB version

what is this?

14

OS Type

what is this?

Linux

DB Type

what is this?

Online transaction processing system

Total Memory (RAM)

what is this?

4

GB

Number of CPUs

what is this?

2

Number of Connections

what is this?

20

Data Storage

what is this?

SSD storage

Generate

postgresql.conf

ALTER SYSTEM

Add/modify this settings in **postgresql.conf** and restart database

```
# DB Version: 14
# OS Type: linux
# DB Type: oltp
# Total Memory (RAM): 4 GB
# CPUs num: 2
# Connections num: 20
# Data Storage: ssd

max_connections = 20
shared_buffers = 1GB
effective_cache_size = 3GB
maintenance_work_mem = 256MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1
effective_io_concurrency = 200
work_mem = 26214kB
min_wal_size = 2GB
max_wal_size = 8GB
```

Copy configuration



# Настройка параметров БД - DWH



## PGTune

### Parameters of your system

DB version what is this?

OS Type what is this?

DB Type what is this?

Total Memory (RAM) what is this?

Number of CPUs what is this?

Number of Connections what is this?

Data Storage what is this?

Generate

postgresql.conf

ALTER SYSTEM

Add/modify this settings in **postgresql.conf** and restart database

```
# DB Version: 14
# OS Type: linux
# DB Type: dw
# Total Memory (RAM): 4 GB
# CPUs num: 2
# Connections num: 20
# Data Storage: ssd

max_connections = 20
shared_buffers = 1GB
effective_cache_size = 3GB
maintenance_work_mem = 512MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 500
random_page_cost = 1.1
effective_io_concurrency = 200
work_mem = 13107kB
min_wal_size = 4GB
max_wal_size = 16GB
```

Copy configuration

# Настройка соединения DBeaver

Конфигурация соединения "postgres\_regsys2"

**Настройки соединения**  
Свойства соединения с PostgreSQL

PostgreSQL

▼ Настройки соединения

- Инициализация
- Команды ОС
- Идентификация кли
- Transactions
- Общее
- Метаданные
- Обработка ошибок
- > Редактор данных
- > SQL редактор

Главное PostgreSQL Свойства драйвера SSH Proxy SSL

Server

Connect by: ☒ Host ☐ URL

URL: jdbc:postgresql://192.168.1.9:5433/postgres

Хост: 192.168.1.9 Порт: 5433

База данных: postgres

Аутентификация

Аутентификация: Database Native

Пользователь: postgres

Пароль: • ☒ Сохранять пароль локально

Advanced

Роль сессии: Локальный клиент: PostgreSQL Binaries

Вы можете использовать системные переменные в параметрах.

Драйвер: PostgreSQL

Тест соединения ...

OK Отмена



# Соединения DBeaver

The screenshot shows the DBeaver 22.3.1 interface. The title bar indicates the current connection is 'postgres\_regsys2' and the active script is 'Script-7'. The menu bar includes 'Файл', 'Редактирование', 'Навигация', 'Поиск', 'Редактор SQL', 'База данных', 'Окна', and 'Справка'. The toolbar contains icons for file operations, SQL execution, and connection management.

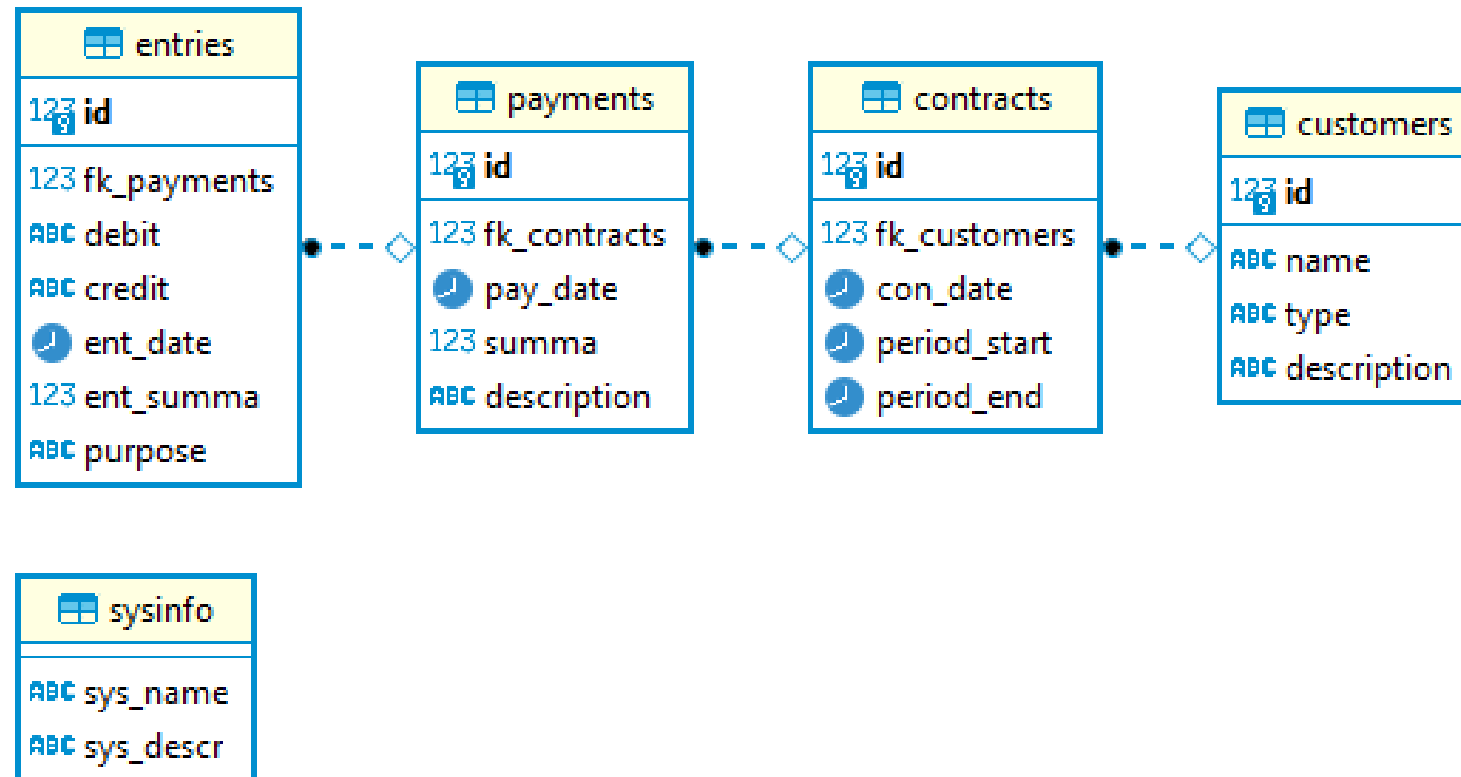
On the left sidebar, the 'Базы данных' (Databases) section is expanded, showing a list of connections. The connection 'postgres\_regsys2 - 192.168.1.9:5433' is highlighted with a red box. Other connections listed include 'postgres\_dwh - 192.168.1.10:5434', 'postgres\_regsys1 - 192.168.1.7:5432', and 'postgres\_regsys3\_docker - localhost:49153'.

The main editor area displays the SQL query: `select * from customers;`. Below the query editor, the 'customers 1' table is shown with 8 rows of data. The table has columns: 'id', 'name', 'type', and 'description'.

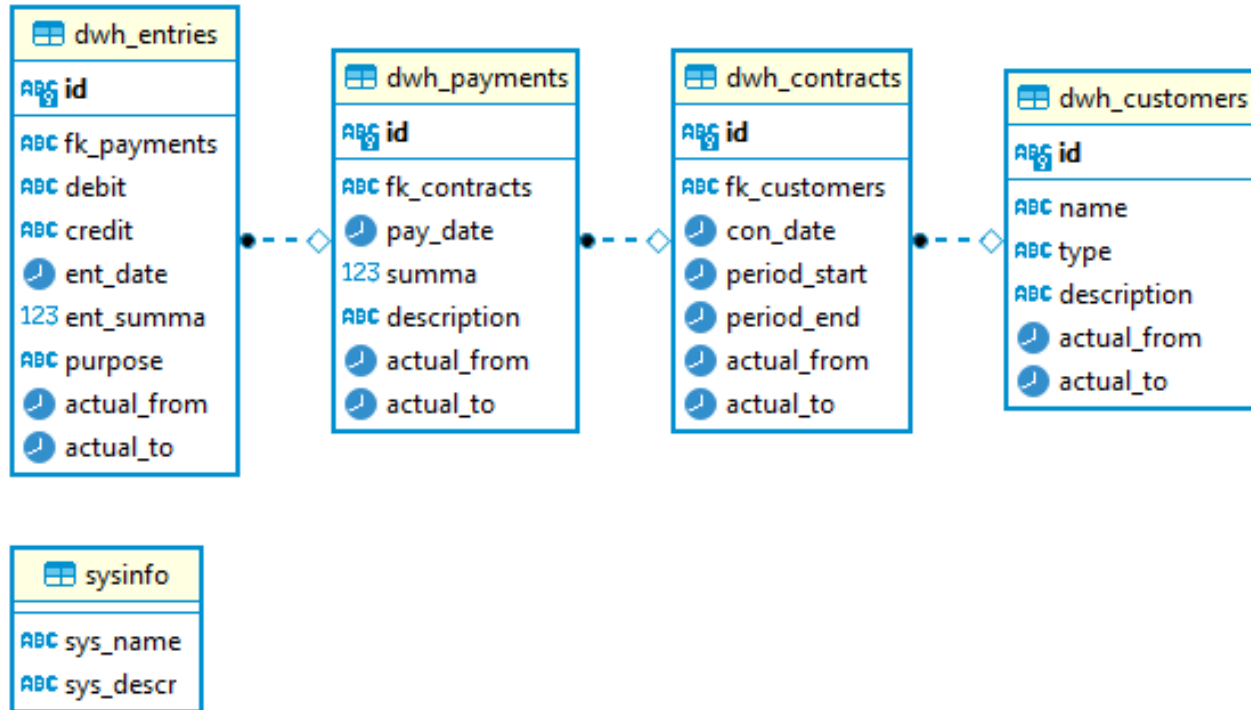
	id	name	type	description
1	1	b779ab7e75	2	c2cd7165067653516b3c
2	2	10c51b059f	3	363cd6784e2403224203
3	3	a659afad77	1	38d70efe13ef02ed91c9
4	4	5174544398	1	e130c43eb22b0a928284
5	5	1ea3607804	3	90d88a992ab6ac1b1a31
6	6	dd6cbf66bf	2	62ff9e9c36203e6714fa
7	7	8a2a7d1921	2	acce1e12d4656a57f765
8	8	0a128b9362	3	b484cf0ea80c2aab1189

At the bottom, the status bar shows '200 строк получено - 4ms, 2022-12-31 в 14:56:54'.

# Таблицы регистрирующих систем



# Таблицы хранилища данных





# Таблицы витрин

dm_contract_payments
ABC id
ABC fk_customers
🕒 con_date
🕒 period_start
🕒 period_end
ABC customer_name
ABC customer_type
ABC customer_desc
123 payment_cnt
123 payment_sum
🕒 load_date

dm_entries_customers
ABC id
ABC fk_payments
ABC debit
ABC credit
🕒 ent_date
123 ent_summa
ABC purpose
🕒 pay_date
ABC fk_contracts
ABC fk_customers
🕒 con_date
🕒 period_start
🕒 period_end
ABC customer_name
ABC customer_type
ABC customer_desc
🕒 load_date

# Соединение Pentaho

The screenshot shows the 'Database Connection' dialog box in Pentaho. The 'General' tab is selected in the left sidebar. The 'Connection name' field is set to 'regsys2'. The 'Connection type' list has 'PostgreSQL' selected. The 'Access' list has 'Native (JDBC)' selected. The 'Settings' section on the right contains the following fields: 'Host Name' (192.168.1.9), 'Database Name' (postgres), 'Port Number' (5433), 'Username' (postgres), and 'Password' (masked with a dot). At the bottom, there are buttons for 'Test', 'Feature List', 'Explore', 'OK', and 'Cancel'.

Database Connection

General  
Advanced  
Options  
Pooling  
Clustering

Connection name:  
regsys2

Connection type:  
PostgreSQL  
Redshift  
Remedy Action Request System  
SAP ERP System  
SQLite  
Snowflake  
SparkSQL  
Sybase  
SybaseIQ  
Teradata  
UniVerse database

Access:  
Native (JDBC)  
ODBC  
JNDI

Settings  
Host Name:  
192.168.1.9  
Database Name:  
postgres  
Port Number:  
5433  
Username:  
postgres  
Password:  
•

Test Feature List Explore

OK Cancel

# Загрузка клиентов в хранилище (1)

Table input

Step name

Connection

SQL

```
select
('sys2|' || id) as id
,name
,type
,description
,current_date as actual_from
,'9999-12-31'::date as actual_to
from customers
```

Line 1 Column 0

Store column info in step meta data ☐

Enable lazy conversion ☐

Replace variables in script? ☐

Insert data from step

Execute for each row? ☐

Limit size



# Загрузка клиентов в хранилище (2)

Table output

Step name:

Connection:

Target schema:

Target table:

Commit size:

Truncate table: ☐

Ignore insert errors: ☐

Specify database fields: ☒

Main options Database fields

Fields to insert:

#	Table field	Stream field	
1	id	id	
2	name	name	
3	type	type	
4	description	description	
5	actual_from	actual_from	
6	actual_to	actual_to	

# Загрузка клиентов в хранилище (3)

The screenshot displays the Spoon - load\_customers window. The left sidebar shows a tree view of folders: Input, Output, Streaming, Transform, Utility, Flow, Scripting, Pentaho Server, Lookup, Joins, Data Warehouse, Validation, Statistics, Big Data, Agile, Cryptography, and Job. The main canvas shows a data flow diagram with two steps: 'get customers' and 'load dwh\_customers', both marked with green checkmarks. The 'Execution Results' panel at the bottom shows a log of the transformation execution.

**Execution Results**

- Logging
- Execution History
- Step Metrics
- Performance Graph
- Metrics
- Preview data

2023/01/01 20:38:13 - Spoon - Running transformation using the Kettle execution engine  
2023/01/01 20:38:13 - Spoon - Transformation opened.  
2023/01/01 20:38:13 - Spoon - Launching transformation [load\_customers]...  
2023/01/01 20:38:13 - Spoon - Started the transformation execution.  
2023/01/01 20:38:13 - load\_customers - Dispatching started for transformation [load\_customers]  
2023/01/01 20:38:13 - load dwh\_customers.0 - Connected to database [dwh] (commit=1000)  
2023/01/01 20:38:14 - get customers.0 - Finished reading query, closing connection  
2023/01/01 20:38:14 - get customers.0 - Finished processing (I=10000, O=0, R=0, W=10000, U=0, E=0)  
2023/01/01 20:38:14 - load dwh\_customers.0 - Finished processing (I=0, O=10000, R=10000, W=10000, U=0, E=0)  
2023/01/01 20:38:14 - Spoon - The transformation has finished!!

# Получение данных для витрины

Table input

Step name:

Connection:

SQL

```
with q1 as
(
select
  con.id
,con.fk_customers
,con.con_date
,con.period_start
,con.period_end
,cust.name as customer_name
,cust.type as customer_type
,cust.description as customer_desc
,(select json_build_object(
  'payment_cnt', count(*)::varchar,
  'payment_sum', sum(summa)::varchar)
 from dwh_payments pay where pay.fk_contracts = con.id
 and current_date between pay.actual_from
 and pay.actual_to) pay_json
,current_date as load_date
from
dwh_contracts con
left join dwh_customers cust on cust.id = con.fk_customers
and current_date between cust.actual_from and cust.actual_to
where
current_date between con.actual_from and con.actual_to
)
select
  q1.id
,q1.fk_customers
,q1.con_date
,q1.period_start
,q1.period_end
,q1.customer_name
,q1.customer_type
,q1.customer_desc
,(q1.pay_json ->> 'payment_cnt')::int as payment_cnt
,(q1.pay_json ->> 'payment_sum')::numeric(30,2) as payment_sum
,load_date
from q1
```

Line 1 Column 0

☐ Store column info in step meta data

☐ Enable lazy conversion

☐ Replace variables in script?

Insert data from step

☐ Execute for each row?

Limit size



# Выводы

Если выполнить предварительную подготовку, то разворачивание сложной инфраструктуры на рабочей станции не занимает много времени.

Если есть предварительно настроенная виртуальная машина, то после клонирования требуются лишь незначительные дополнительные настройки.

Поскольку для ведения разработки нужно, чтобы одновременно было доступно не более двух БД, то даже для сложных инфраструктур не требуется слишком много оперативной памяти на рабочей станции.

Для разворачивания таких инфраструктур виртуальные машины предпочтительнее Docker-а, поскольку желательна установка дополнительного программного обеспечения в гостевые ОС, в частности, средств разработки.

С незначительными доработками представленная инфраструктура может быть использована для других задач.

The background of the slide is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue gradient. A network of thin, light blue lines connects various points across the blue area, creating a digital or technological feel. The text "Спасибо за внимание!" is centered in the middle of the slide in a white, sans-serif font.

Спасибо за внимание!