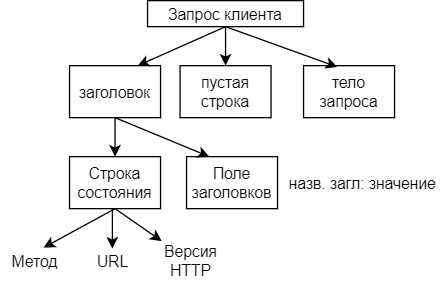
# **1. Опишите, структуру HTTP запроса клиента к серверу. Ответ проиллюстрируйте примерами.**

Клиент

web сервер

HTTP запрос

HTTP ответ



Формат заголовка:

<Название заголовка>: <значение>

Методы запросов:

GET- сервер отправляет что-то клиенту

POST – с клиента что-то на сервер  
PUT – обновить какие-то данные  
DELETE – удалить какие-то данные

…

OPTION

…

Клиент может отправлять вместе с запросом параметры, а сервер клиенту вместе с ответом – html-страницу. Если отправлять данные с помощью POST, то они должны быть в теле запроса. Если отправлять что-то на сервер методом GET (параметры будут за строкой состояния), то их будет видно на странице

**Примеры**:

1. GET [http://localhost/kunsov/index.html http/2.0](http://localhost/kunsov/index.html%20http/2.0)

Accept: text/html

Location: <URL>

1. GET <http://127.0.0.1:5000/api/v1/user/search?name=Anatoliy>

Accept: text/html

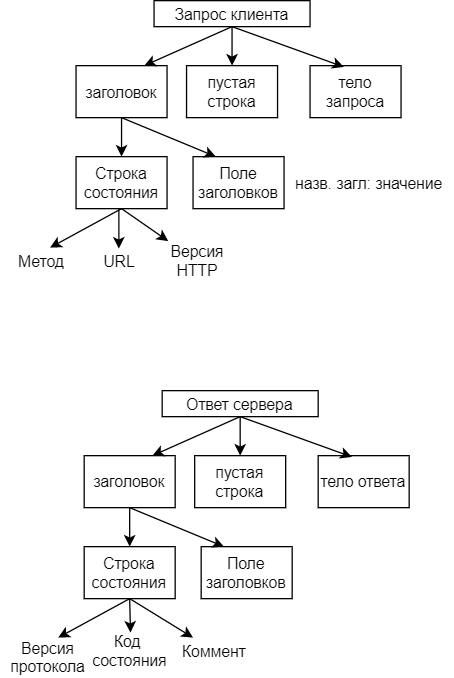
Location: <URL>

Заголовок Accept - это сообщение, какие типы данных он принимает

Заголовок Location - сообщает серверу, что надо прекратить выполнение текущего скрипты и надо перейти к этому адресу

# **2. Опишите структуру HTTP ответа сервера клиенту. Ответ проиллюстрируйте примерами.**

В ответе от сервера есть немало количество кодов состояния  
Как правило, они характеризуются так:

* Информационные (100-105)
* Успешные (200-226)
* Перенаправление (300-307)
* Ошибка клиента (400-499)
* Ошибка сервера (500-510)

**Пример**:  
HTTP/2.0 200 OK  
Content-Type: text/html

Content-Length: 243

Статус 200 – говорит, что всё успешно обработано  
Content-Type – тип сообщения, которое в ответе  
Content-Length -  указывает размер отправленного получателю тела объекта в байтах

# **3. Для чего при взаимодействии клиента с сервером нужен WSGI интерфейс.**

WSGI (Web Server Gateway Interface) – стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера и самим веб-сервером. WSGI – это API общения между собой серверных веб-библиотек для python.

WSGI появляется, потому что веб-серверу необходимо взаимодействовать с веб-приложением. WSGI определяет правила, которые должны быть реализованы на стороне веб-приложения и на стороне веб-сервера, чтобы они могли взаимодействовать друг с другом. Таким образом, сервер, совместимый с WSGI, сможет взаимодействовать с веб-приложением, совместимым с WSGI.

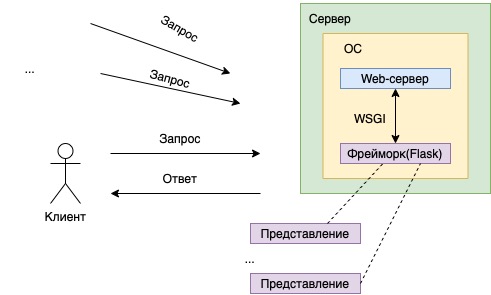
В архитектуре WSGI приложение WSGI должно быть вызываемым и должно передаваться на веб-сервер, чтобы веб-сервер мог вызывать веб-приложение всякий раз, когда сервер получает запрос.

# **4. Как фреймворк Flask обеспечивает поддержку WSGI интерфейса.**

Когда пользователь вводит что-либо в браузере (например, vk.com), от браузера отправляется запрос к серверу, который отвечает за работу этого сайта (идет по IP-адресу, который браузер узнал от DNS-сервера). Сервер постоянно находится в ожидании очередного запроса и, как только он приходит, формирует ответ клиенту в виде HTML-страницы. Эта страница возвращается в браузер клиента.

WSGI (Web Server Gateway Interface) – стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера и самим веб-сервером. То есть это фактически интерпретатор Python, который запускает WSGI – приложение, написанное на Flask.

При поступлении очередного запроса активизируется WSGI-приложение, выполняется определенный обработчик (или представление; как правило, выполнено в виде обычной Python-функции). Соответственно, когда приходит несколько запросов, то активизируется сразу несколько представлений. То есть в один момент времени отрабатывается сразу несколько функций (многопоточность).



По стандарту, WSGI-приложение должно удовлетворять следующим требованиям:

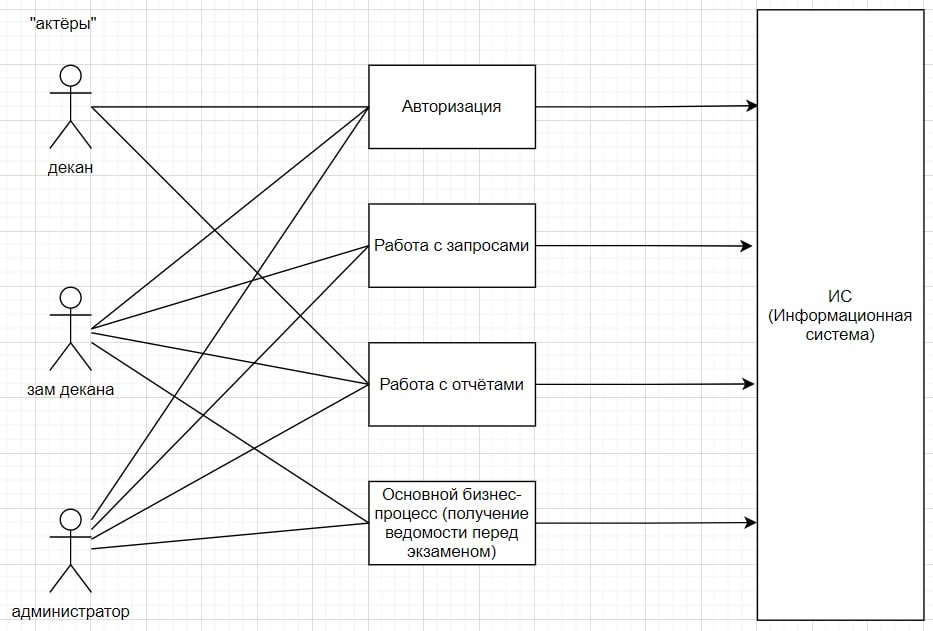
1. Должно быть вызываемым (callable) объектом (обычно это функция или метод)
2. Принимать два параметра:
3. словарь переменных окружения (environ)
4. обработчик запроса (start\_response)
5. Вызывать обработчик запроса с кодом HTTP-ответа и HTTP- заголовками
6. Возвращать итерируемый объект с телом ответа

# **5. Для чего составляется и как используется при проектировании системная UML диаграмма вариантов использования?**

UML (**U**nified **M**odeling **L**anguage) – стандартизированный язык моделирования при проектировании программ.

UML диаграмма вариантов использования – это диаграмма, которая описывает какой функционал разрабатываемой программной системы доступен каждой группе пользователей.

UML диаграммы строятся для того, чтобы не забывать, что нужно реализовывать в программе.

***Пример UML-диаграммы***

# **6. Для кого пишутся сценарии вариантов использования?**

Сценарий – это история, записанная по пунктам. В нем описаны варианты использование ИС по требованиям заказчика. Сценарии использования описывают то, «кто» и «что» может сделать с рассматриваемой системой. Сценарий создается для конечного пользователя, который не разбирается в технических особенностях ИС.

*Любой сценарий должен предусматривать:*

● предусловия,

● исключения,

● гарантии.

***Дополнительная инфа:***

***Пример:***

*Основной успешный сценарий (привёл ĸ ожидаемому результату):*

1. Пользователь начинает сценарий
2. Отсылка формы для ввода группы и экзамена
3. Пользователь вводит группу и экзамен
4. Выдать ведомость и закончить сценарий

Исключения:

4.a. Ошибка ввода группы → возврат ĸ пункту 2.

4.б. Ошибка ввода экзамена → возврат ĸ пункту 2.

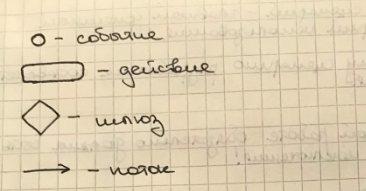
# **7. Для кого составляются и как используются BPMN диаграммы?**

*(Как я выяснил, препод сказала, что она составляется для разработчиков, что бред)*

BPMN-схемы верхних уровней рассчитаны на участников бизнес-процесса и других заинтересованных лиц, которые с их помощью могут получить доступную картину задействованных этапов. Схемы более детальных уровней предназначены для тех, кто непосредственно вовлечен во внедрение процесса, и в них содержится достаточно информации для выполнения этой задачи. BPMN-схемы описывают бизнес-процессы единым стандартизированным языком, который понятен всем участникам независимо от уровня их технических познаний, то есть бизнес-аналитикам, исполнителям процесса, менеджерам, разработчикам, а также внешним сотрудникам и консультантам. В идеале эти схемы должны представлять последовательность действий достаточно подробно и понятно, чтобы перекинуть удобный мостик от проекта бизнес-процесса до его внедрения.

Моделирование бизнес-процессов используется для донесения широкого спектра информации до различных категорий пользователей. Диаграммы бизнес-процессов позволяют описывать сквозные бизнес-процессы, но в то же время помогают читателям быстро понимать процесс и легко ориентироваться в его логике.

**Основные символы**



* Event – это то событие, которое произошло в описании процесса или хореографии (о ней я расскажу отдельно). Эти события могут быть начальными, конечными или промежуточными.
* Activity – это те действия (задачи), которые должны быть выполнены на определенном этапе бизнес-процесса. Их при моделировании обычно обозначают в виде прямоугольников, в которые вписывают суть действия.  
    
  Действия могут быть элементарными, т.е. неделимыми на какие-то более простые действия, так и не элементарными, т.е. такими, которые при детализации делятся на последовательность определенных более простых действий.
* Gateway – это контрольный узел, который появляется в случае условного ветвления бизнес-процесса. Графически изображается в виде ромба.  
    
  Также шлюзы необходимы в случаях, когда порядок действий зависит от тех или иных факторов. Например, при работе с заказчиками шлюз появляется на этапе принятия клиентом решения о покупке – «да или нет». При положительном решении необходимо оформить покупку, при отрицательном – выяснить возможные причины отказа, провести работу с «отказом» и т.д.
* Поток Flow – это последовательность действий, обозначается как стрелка, и показывает, какое действие после какого необходимо совершить.

# **8. Что представляет собой динамический шаблон, чем он отличается от статического шаблона. Приведите примеры динамических шаблонов.**

Динамический шаблон — это основная копия веб-страницы на основе HTML, которую можно создать, чтобы она содержала параметры, форматирование и такие элементы страницы, как текст, графика, макет страницы, стили и области веб-страницы, которые можно изменить.

Динамические веб-страницы имеют более широкий набор опций для обновления пользовательского интерфейса. Основное отличие динамического и статического шаблона в том, что в статический шаблон не подразумевает генерации содержания страницы от действий пользователя на сайте, на статических веб-страницах все, что вводится в коде HTML, отображается на веб-странице как есть. На динамических веб-страницах кодировка отличается, и в основном используются скриптовые языки программирования, отображающие контент, который запрашивается пользователем.

Пример

# **9.Что представляет собой статический шаблон? Приведите примеры статических шаблонов.**

Статическими шаблонами являются HTML страницы без использования в них функций шаблонизаторов. Например, в большинстве шаблонизаторов, например Jinja, используемого во Flask, Handlebars, Mustache, и.т.п., все управляющие команды пишутся внутри символьных комбинаций “{%”, “%}” и “{{“, “}}”, обозначающих границы блоков с командами шаблонизатора.

Примером подобной страницы будет любая HTML страница, отображаемая браузером.

**Например**:

<!DOCTYPE html>

<html lang=”ru”>

<head>

<meta charset="UTF-8">

<title>**Заголовок**</title>

</head>

<body>**Это простейшая html-страница**</body>

</html>

# **10. Как динамический шаблон преобразуется в HTML страницу? Как эта страница передается клиенту?**

В большинстве шаблонизаторов, например Jinja, используемого во Flask, Handlebars, Mustache, и.т.п., все управляющие команды пишутся внутри символьных комбинаций «{%», «%}» и «{{«, «}}», обозначающих границы блоков с командами шаблонизатора.

В блоках между «{%» и «%}» записываются логические команды шаблонизатора. Например, if или цикл for. В блоках между «{{« и «}}» записывается имя переменной, значение которой необходимо подставить на место этого блока.

Шаблонизатор, зачастую, подключается к программе, как библиотека. Для обработки им шаблона используется вызов функции шаблонизатора, в которую передаётся сам шаблон и переменные, используемые в шаблоне. Функция возвращает переменную строчного типа, являющуюся результатом шаблонизации. Если мы шаблонизировали html-страницу, то для отправки её клиенту, необходимо в заголовках ответа сервера кроме стандартного «HTTP/1.1 200 OK», прописать заголовок «Content-Type: text/html» для правильной интерпретации ответа браузером, после чего в теле ответа можно отправлять саму html-страницу.

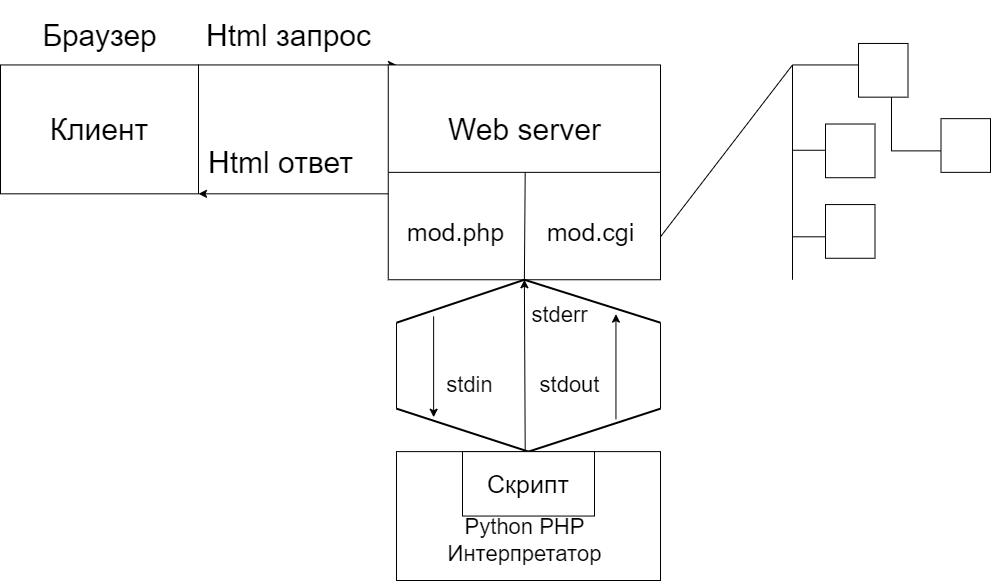
# **11. На основании чего составляются требования к динамическим шаблонам. Ответ проиллюстрируйте примером.**

Динамический шаблон — это HTML страница, в которую вставляется постоянно меняющееся информационное наполнение, которое обычно хранится в базе данных. Когда пользователь запрашивает страницу, соответствующая информация извлекается из базы, вставляется в шаблон, образуя новую страницу.

Требование к динамическому шаблону представляют собой данные, которые мы получаем из базы данных, передаваемые в HTML страницу. Так же обязательно нужно указать типа передаваемых значений, название переменных и указать для чего они будут использоваться на странице.

Требование элементов в динамическом шаблоне для пользователей могут быть следующее:

* Гиперссылки
* Элементный интерфейса
* Текст
* Материалы
* Списки
* Таблицы

Проиллюстрированный пример

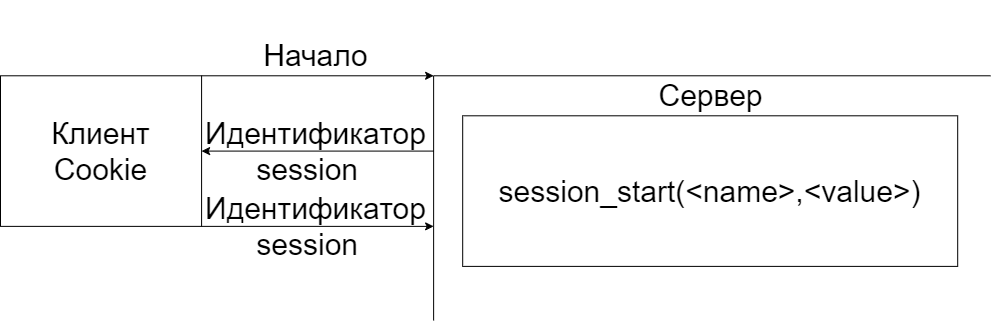
Пример:

*<!DOCTYPE* html*>  
<*html lang="en"*>  
 <*head*>* {% if title %}  
 *<*title*>*{{title}} - microblog*</*title*>* {% else %}  
 *<*title*>*Welcome to microblog*</*title*>* {% endif %}  
 *</*head*>  
 <*body*>  
 <*h1*>*Hello, {{user.nickname}}!*</*h1*>  
 </*body*>  
</*html*>*

# **12. Опишите, что представляет собой механизм сессий во flask. Приведите пример использования сессий.**

Сессии — способ хранить данные конкретных пользователей между запросами. Объект session из пакета flask используется для настройки и получения данных сессии. Объект session работает как словарь, но он также может отслеживать изменения.

Сессия Flask позволяет запоминать информацию от одного запроса к другому. Flask делает это с помощью подписанного файла cookie. Пользователь может просматривать содержимое сессии. Сессии работают по похожему с cookies принципу: они также хранятся в браузере в виде особых кук сессий.

Пример:

Код демонстрирует, как можно читать, записывать и удалять данные сессии.

from flask import session

def add\_to\_basket*(*item*)*:  
 basket = session.get*(*'basket', *[])* basket.append*(*item*)* session*[*'basket'*]* = basket  
  
  
def clear\_basket*()*:  
 if 'basket' in session:  
 session.pop*(*'basket'*)*

# **13. Опишите, как механизм сессий может использоваться для разделения прав доступа к информационной системе.**

Для реализации разделения прав доступа к информационной системе в базе данных существует отдельная таблица, в которую занесены логины и пароли пользователей, а также группа пользователей. Именно эта группа и определяет права пользователя. Данные о том какие именно права принадлежат пользователю прописываются в программе в файле permissions.json.

При авторизации пользователь вводит логин и пароль. Используя эти данные из базы данных определяется группа пользователя. Эта группа заносится в сессию. Уже впоследствии, при попытке выполнить какое-либо действие, происходит проверка, разрешено ли конкретное действие для пользователя. Для этого группа из сессии сравнивается с группами в файле permissions.json, и определяются права для данной группы, прописанные в этом файле.

# **14. Что представляют собой и как используются декораторы? Приведите примеры.**

Декоратор — это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. Внутри себя декоратор определяет функцию-"обёртку", которая дает возможность делать что-либо до и после того, что сделает декорируемая функция, не изменяя её.

Пример деĸоратора, проверяющего, авторизован ли пользователь:

def login\_required(func): #func-декорируемая функция

def wrapper(\*args, \*\* kwargs): #внутренняя функция

if user in session:

return func(\*args, \*\*kwargs) #запускает декорируемую функцию

else:

return ‘Вам необходимо авторизоваться’

return wrapper

Декоратор принимает в качестве аргумента функцию func. Затем расширяет функциональность func, создавая внутреннюю функцию wrapper и добавляя туда расширенное поведение. После чего возвращает внутреннюю функцию wrapper, которая становится новой версией функции func.

# **15. Для чего в состав системной архитектуры при разработке ИС включают фреймворки? Ответ проиллюстрируйте примерами.**

Фреймворк - это некий набор библиотек, который облегчает разработку любых продуктов. Они нужны чтобы делать код быстрее и качественнее, не задумываясь о некоторых мелочах.

Например, при использовании Flask, чтобы отправить ответ, нужно просто написать return. Вместо более низкой настройки. Или же при использовании Jinja, все управляющие команды пишутся внутри символьных комбинаций «{%», «%}» и «{{«, «}}», обозначающих границы блоков с командами шаблонизатора.

# **16. Перечислите основные требования к OLAP и OLTP системам. Дайте их сравнительную оценку.**

OLTP - производительность приоритетность, доступность

OLAP – гибкость, независимость работы пользователя

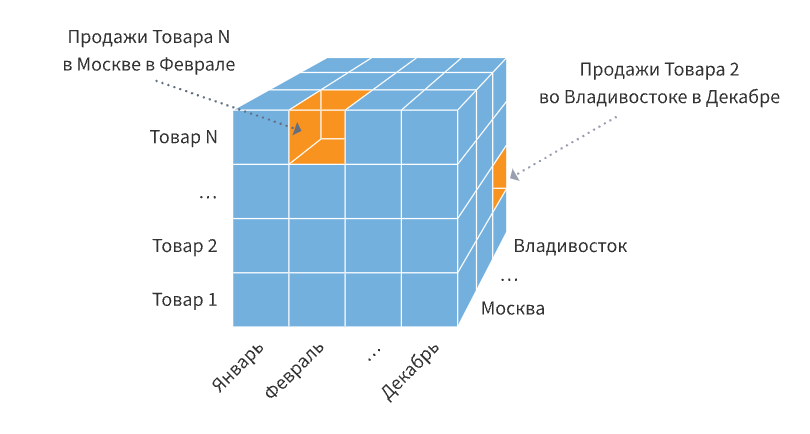
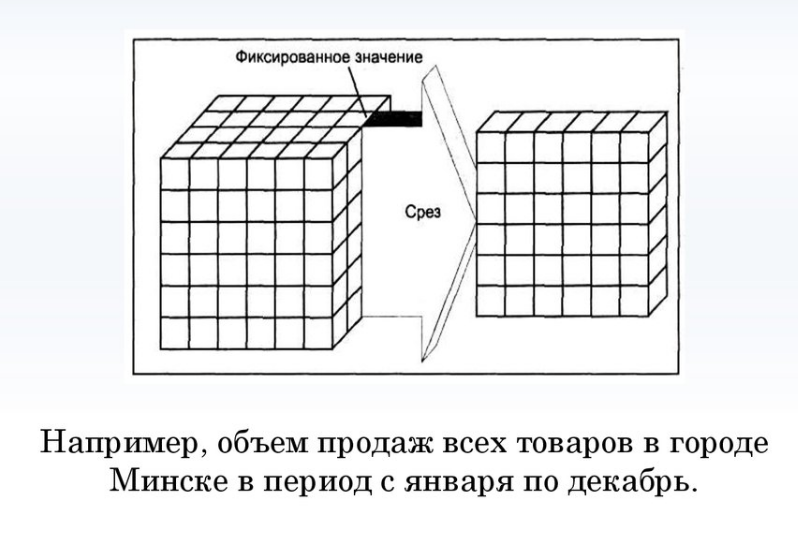
|  |  |  |
| --- | --- | --- |
|  | OLTP | OLAP |
| Степень детализации данных | Детальные данные за короткий отрезок времени | Детали за длительный срок и агрегированные данные |
| Качество данных | Допускаются ошибки из-за ошибок ввода | Не допускаются ошибки |
| Допущение избыточности | Минимальное, в соответствии с правилами нормализации | Допускается контролирование, денормализация (избыточность) |
| Операции над данными | Возможность редактирования | Только добавление |
| Приоритетность | Производительность и доступность | Гибкость и независимость |

# **17. Поясните, что представляет собой и для чего используются многомерные кубы в OLAP системах.**

Куб представляет собой [многомерный](https://wiki.loginom.ru/articles/multidimensional.html) массив данных, используемый в системах [оперативной аналитической обработки](https://wiki.loginom.ru/articles/online-analytical-processing.html) (OLAP). Кроме этого, куб можно рассматривать как модель данных, обеспечивающую функциональность OLAP.

В основе идеи построения куба лежит многомерная модель данных, предполагающая их разделение на [измерения](https://wiki.loginom.ru/articles/dimension.html) и [факты](https://wiki.loginom.ru/articles/fact.html). Куб, который содержит более чем три измерения, называется многомерным. Куб потенциально содержит всю информацию, которая может потребоваться для ответов на любые запросы.

Куб можно рассматривать как многомерное обобщение двумерной электронной таблицы. При этом измерения образуют оси многомерной модели данных (ребра куба), а факты — ячейки внутри куба, расположенные на пересечении соответствующих значений измерений.

На рисунке представлен пример 3-мерного куба, содержащего измерения «Товар», «Месяц» и «Город».

# **18. Опишите, что представляет собой логическая модель MOLAP. Приведите пример такой модели.**

MOLAP (матричная OLAP) — это классический OLAP, который облегчает анализ данных с помощью многомерного куба данных.

MOLAP лучше всего подходит для небольших наборов данных, он быстро рассчитывает агрегаты и возвращает ответы, но при этом генерируются огромные объёмы данных. Кратко можно назвать MOLAP кешированием всего объема данных.

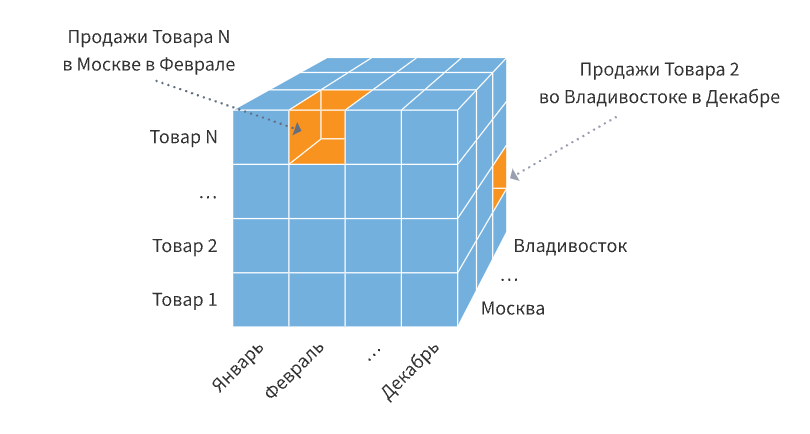
**Преимущества MOLAP.**

* Высокая производительность. Поиск и выборка данных осуществляется значительно быстрее, чем при многомерном концептуальном взгляде на реляционную базу данных.

**Недостатки MOLAP.**

* Гиперкуб сильно разрежен. Поскольку данные измерений упорядочены неопределенные значения удается удалить только за счет подбора оптимального порядка сортировки, позволяющего организовать факты в достаточно большие непрерывные группы. Но и в этом случае проблема решается только частично. Поэтому такой метод хранения используется только для хранения небольшого объема данных.
* MOLAP могут работать только со своими собственными многомерными БД и основываются на патентованных технологиях для многомерных СУБД, поэтому являются наиболее дорогими. Эти системы обеспечивают полный цикл OLAP-обработки и либо включают в себя, помимо серверного компонента, собственный интегрированный клиентский интерфейс, либо используют для связи с пользователем внешние программы работы с электронными таблицами.
* Отсутствуют единые стандарты на интерфейс, языки описания и манипулирования данными.

**Пример такой СУБД - EssBase**



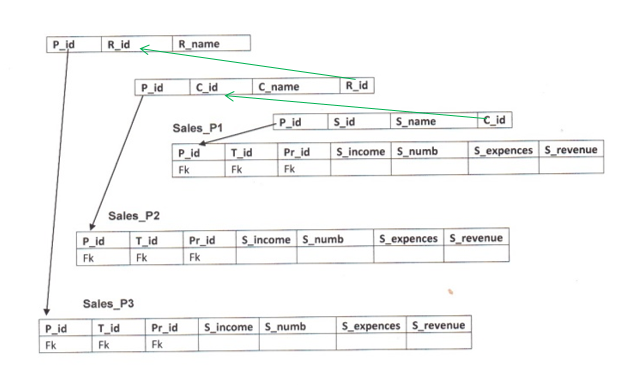
# **19. Опишите, что представляет собой логическая модель ROLAP типа «снежинка». Приведите пример такой модели.**

ROLAPэто OLAP система, в который данные хранятся в реляционной базе данных. Тип ROLAP модели зависит от того, как мы будем хранить данные.

Для модели типа “снежинка”, характерны следующие особенности:

* Таблицы фактов и таблицы измерений.
* В таблице фактов может находиться несколько величин (выручка, расходы, количество)
* Детальные и агрегированные данные хранятся отдельно
* Таблица фактов связана с несколькими таблицами измерений.
* Суррогатный ключ таблицы фактов состоит из совокупности ключей таблиц измерений.
* Таблицы измерений нормализованы.

**Пример (продажа товара в определённом магазине/городе/регионе):**



Сделаем 3 таблицы: для города, для региона, для магазина. Все они имеют связь один ко многим, следовательно, в таблицах города и магазина есть поле вторичного ключа. Для понятности назовём их “Таблицами мест”.

Так же сделаем ещё 3 таблицы, которые одинаковые по своей структуре (они хранят данные о продаже и id места). Id места продажи берём из таблиц сделанных ранее, при том в каждой таблице он должен быть только из 1 таблицы мест. В конечном итоге у нас получится 3 таблицы: продажи и их данные по региону, продажи и их данные по городу, продажи и их данные по магазину

# **20. Опишите, что представляет собой логическая модель ROLAP типа «звезда». Приведите пример такой модели.**

ROLAPэто OLAP система, в который данные хранятся в реляционной базе данных. Тип ROLAP модели зависит от того, как мы будем хранить данные.

Для модели типа “звезда”, характерны следующие особенности:

* В основе лежит таблицы фактов и таблицы измерений
* В таблице фактов может находится одновременно несколько величин.
* Детальные и агрегированные данные могут находится вместе.
* Таблицы фактов связаны с несколькими таблицами изменений
* Суррогатный ключ таблицы фактов состоит из совокупности ключей таблиц измерений (из-за этого факта модель получила своё название, т.к. все таблицы соединяются в одну общую)
* Таблицы измерений сильно денормализованы (чтобы ускорить создание отчёта)

**Пример (продажа товара в определённый сезон в определённом магазине/городе/регионе):**



Имеется 4 таблицы: товар, продажи, период и место Tovar, sales, period, place соответственно.

В таблице Place хранятся id и название региона, id и название города, id и название магазина. Так же есть поле level, которая характеризует уровень факта, на данном примере, если указан только регион, то уровень будет 1. Если указан регион и город, то второй. Если указано всё, то 3. Вместо цифр может быть что угодно, главное, чтобы можно разделить по уровням. И так же по аналогии для таблиц Tovar, Period.

В таблицу sales заносится id из всех этих таблиц и присваивается уровень.

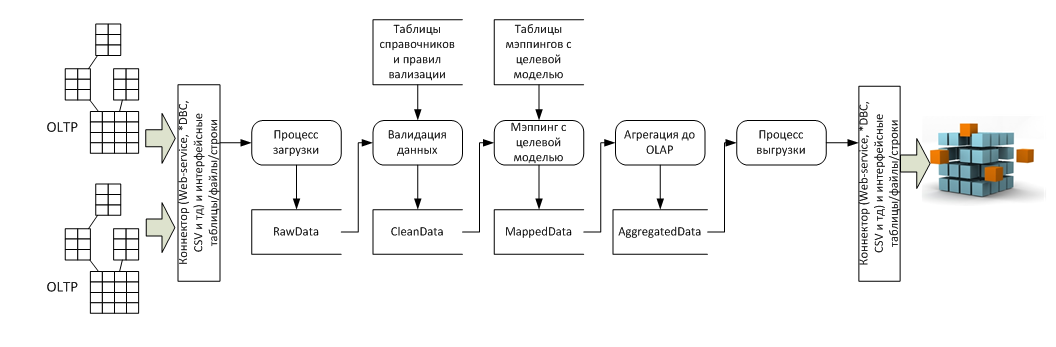
# **21. Для чего нужна и как работает система ETL. Приведите поясняющий пример.**

ETL – это система, которая перегружает данные из источника OLTP базы данных в OLAP базы данных, один из основных процессов в управлении хранилищами данных, который включает в себя:

− извлечение данных из внешних источников;

− их трансформация и очистка, чтобы они соответствовали потребностям бизнес-модели;

− и загрузка их в хранилище данных.



1. Процесс загрузки – Его задача затянуть в ETL данные произвольного качества для дальнейшей обработки, на этом этапе важно сверить суммы пришедших строк, если в исходной системе больше строк, чем в RawData то значит — загрузка прошла с ошибкой;
2. Процесс валидации данных – на этом этапе данные последовательно проверяются на корректность и полноту, составляется отчет об ошибках для исправления;
3. Процесс мэппинга данных с целевой моделью – на этом этапе к валидированной таблице пристраивается еще n-столбцов по количеству справочников целевой модели данных, а потом по таблицам мэппингов в каждой пристроенной ячейке, в каждой строке проставляются значения целевых справочников.
4. Процесс агрегации данных – этот процесс нужен из-за разности детализации данных в OLTP и OLAP системах. Поэтому сначала делается мэппинг на детализации OLTP, а потом в отдельной таблице данные «схлопывают» для загрузки в OLAP;
5. Выгрузка в целевую систему — это технический процесс использования коннектора и передачи данных в целевую систему.

Пример:

Для того чтобы понять какие товары чаще всего приобретаются, нам нужно из базы данных OLTP забрать требуемые значение, затем через систему ETR, чтобы не замедлять работу OLTP, занести их в базу данных OLAP. В итоге по полученным данным мы можем сделать вывод какой товар чаще всего покупали

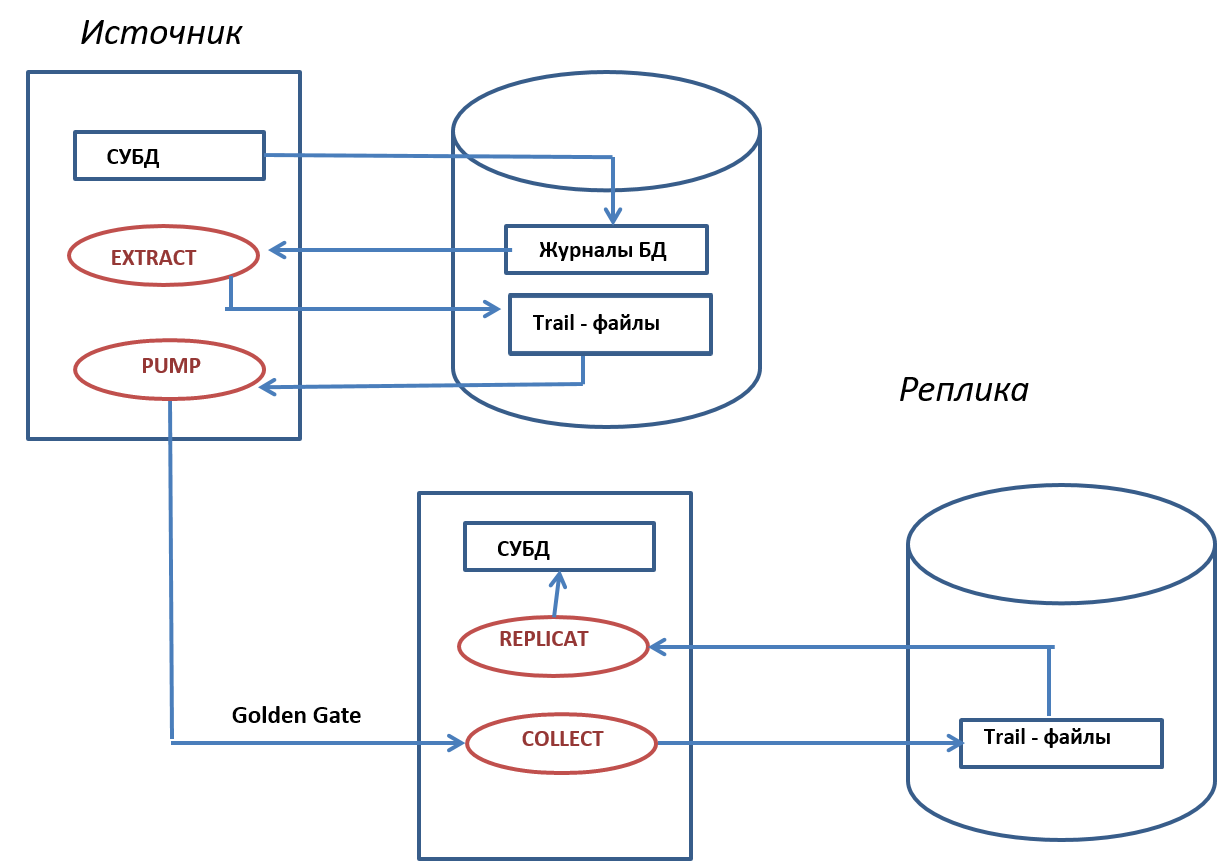
# **22. Для чего нужна и как используется технология Golden Gate.**

**GoldenGate** - это технология для захвата изменений данных, их преобразования и доставки, обеспечивающее двустороннюю репликацию на основе журналов СУБД. GoldenGate помогает гарантировать доступность критических систем в режиме 24/7, а также извлекать данные, необходимые для принятия решений, из различных источников.

технология Golden Gate:

1. **EXTRACT** -Захват изменений из журналов транзакций БД-источника и размещение их в специальных trail-файлах на локальной файловой системе или через процессы PUMP и COLLECT на одном или нескольких удалённых серверах.
2. **PUMP** -Передача данных из trail-файлов c сервера-источника на сервер-приёмник с возможностью сжатия (до 20 раз!) и шифрования. Есть возможность работы с ненадёжными сетями (повтор передачи, возобновление после обрыва связи и т. п.)
3. **COLLECT** -Получение и сохранение изменений, переданных удалённым процессом PUMP, в локальных trail-файлах на сервере приёмнике.
4. **REPLICAT** -Применение в целевой БД (реплике) изменений, извлекаемых из trail-файлов, с сохранением транзакционной целостности

Процесс выполнения технология Golden Gate:



# **23. Что такое Data Mining? Какие типичные задачи DM вам известны. Дайте характеристику этим задачам.**

Data mining - поиск скрытых закономерностей в данных (при том компьютером). За цель берется идентификация тенденций и паттернов, которая при обычном анализе невозможна. Для сегментации данных и оценки вероятности последующих событий используются сложные математические алгоритмы.

Задачи, решаемые методами data mining, принято разделять на описательные ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) descriptive) и предсказательные ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) predictive).

В описательных задачах самое главное — это дать наглядное описание имеющихся скрытых закономерностей, в то время как в предсказательных задачах на первом плане стоит вопрос о предсказании для тех случаев, для которых данных ещё нет.

К описательным задачам относятся:

* поиск ассоциативных правил или паттернов (образцов);
* группировка объектов, кластерный анализ;
* построение регрессионной модели.

К предсказательным задачам относятся:

* классификация объектов (для заранее заданных классов);
* [регрессионный анализ](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7), анализ [временных рядов](https://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B9_%D1%80%D1%8F%D0%B4).