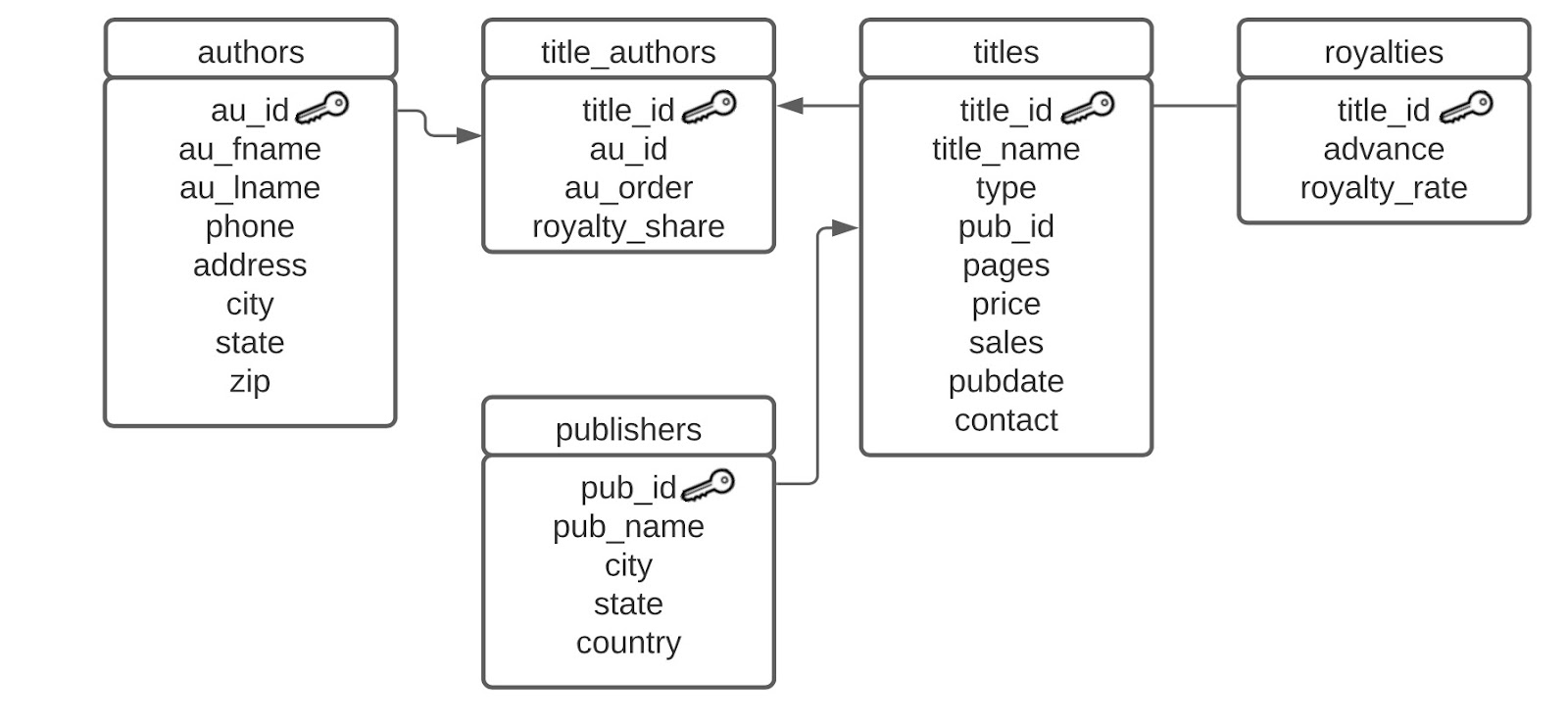
**Повторение SQL**

База данных книжное издательство



**Таблица titles**

В поле «type» могут содержаться следующие значения: биография, бизнес, кулинария

Sales-общее количество проданных экземпляров

Contract- признак подписания контракта автором(0- контракт не подписан 1- контракт подписан)

**Таблица Title\_authors**-стыковочная таблица. С помощью нее реализуется связь многие ко многим между таблицами авторы и книги(один автор может написать несколько книг и одну книгу может писать несколько авторов)

Au\_order-порядок следования авторов

Royality\_share-доля автора в общем гонораре (число от 0 до 1)

**Таблица** **Royalities**-данные о проценте отчислений от продажи книги в виде авторского гонорара, который выплачивается всем авторам

Advance-предоплата

Royality\_rate-процент с продажи книг

**Фильтрация групп записей с помощью предложения HAVING**

Очередность с которой СУБД применяет предложения WHERE,GROUP BY и HAVING:

1. WHERE фильтрует строки, которые являются результатом работы операция предложений FROM и JOIN
2. GROUP BY группирует выход предложения WHERE (результат работы WHERE)
3. HAVING фильтрует строки сгруппированного результата, то есть условие HAVING применяет условие фильтра к каждой группе строк

**Подзапросы**

Подзапрос- команда SELECT, встроенная в другую команду SQL. Вы можете располагать подзапрос в:

* Предложении SELECT,FROM,WHERE или HAVING другой команды SELECT
* Другом подзапросе
* Команде INSERT,UPDATE или DELETE

Вывести список издательств, которые издают тип биография

SELECT pub\_id

FROM publishers

WHERE pub\_id IN (SELECT pub\_id FROM titles WHERE type=”biography”);

Отобразить список всех авторов, живущих в городе где находится издательство

SELECT au\_id,city

FROM authors

WHERE city IN (SELECT city FROM publishers )

**Соединения таблиц**

Соединения таблиц можно рассматривать как процесс сравнения данных в указанных столбцах этих таблиц, и формирование новой таблицы из строк исходных таблиц, которые дают положительный результат при сравнении.

Наиболее часто встречаются соединения с равенством.

При выполнении соединения сначала образуется декартово произведение таблиц, т.е. всевозможные комбинации строк этих таблиц друг с другом.

Само декартово произведение не содержит полезной информации, по этой причине в соединении обязательно должно быть предложение WHERE (Отбирает связанные между собой строки и указывает как именно они должны быть связаны)

**Задание**

Найти имена авторов и издателей, живущих в одном и том же городе

SELECT au\_fname,au\_Iname,pub\_name

FROM authors,publishers

WHERE authors.city=publishers.city

Найти названия всех книг типа биография и имена их авторов

SELECT title\_name,au\_fname

FROM title\_authors,titles,authors

WHERE (authors.au\_id=title\_authors.au\_id) AND (title\_authors.title\_id=titles.title\_id) AND titles.type=”biography”;

**Виды соединений типа JOIN**

CROSS JOIN-это декартово произведение (в результирующей выборке показаны всевозможные комбинации строк)

INNER JOIN-внутреннее соединение (соединяются только те строки где найдены совпадающие значения столбца)

LEFT (OUTER) JOIN-левое внешнее соединение. Включает в себя все строки из таблицы А (совпадающие и несовпадающие) + совпадающие значения из таблицы В. Для строк из таблицы А, к которым не найдено соответствие, значение NULL заносится в столбцы, извлекаемые из таблицы В

RIGHT (OUTER) JOIN-правое внешнее соединение. Обратное соединение предыдущему, т.е. все строки из таблицы В (правая таблица) представлены в соединении и дополнены совпадающими строками из таблицы А

FULL (OUTER) JOIN-полное внешнее соединение. Комбинация левого и правого соединения, т.е. присутствуют все строки из обеих таблиц (в несовпадающих строках значения столбцов заполняются NULL)

UNION-обратное соединение по отношению к INNER JOIN, т.е. включает только те строки из таблиц, для которых не найдено совпадение.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TABLE A   |  |  | | --- | --- | | ID\_NUM | NAME | | 1334 | Иванов | | 1399 | Петров | | 1208 | Сидоров | | TABLE B   |  |  | | --- | --- | | NAME | PHONE | | Иванов | 555-85-54 | | Петров | 555-52-23 | | Петров | 555--1191 | | Крылов | 555-78-94 | | |
| Таблицы 2.1.1 и 2.1.2 Две исходные таблицы для соединения | | |
| **Перекрестное соединение**  **A CROSS JOIN B**   |  |  |  |  | | --- | --- | --- | --- | | ID\_NUM | A.NAME | B.NAME | PHONE | | 1334 | Иванов | Иванов | 555-8554 | | 1399 | Петров | Иванов | 555-8554 | | 1208 | Сидоров | Иванов | 555-8554 | | 1334 | Иванов | Петров | 555-5223 | | 1399 | Петров | Петров | 555-5223 | | 1208 | Сидоров | Петров | 555-5223 | | 1334 | Иванов | Петров | 555-9111 | | 1399 | Петров | Петров | 555-9111 | | 1208 | Сидоров | Петров | 555-9111 | | 1334 | Иванов | Крылов | 555-7894 | | 1399 | Петров | Крылов | 555-7894 | | 1208 | Сидоров | Крылов | 555-7894 |   *Таблица 2.2* | | **Внутреннее соединение**  **A INNER JOIN В**   |  |  |  | | --- | --- | --- | | ID\_NUM | NAME | PHONE | | 1334 | Иванов | 555-85-54 | | 1399 | Петров | 555-52-23 | | 1399 | Петров | 555--1191 |   *Таблица 2.3*  **Левое внешнее соединение**  **A LEFT OUTER JOIN В**   |  |  |  | | --- | --- | --- | | ID\_NUM | NAME | PHONE | | 1334 | Иванов | 555-85-54 | | 1399 | Петров | 555-52-23 | | 1208 | Сидоров | NULL | | 1399 | Петров | 555--1191 |   *Таблица 2.4* |
| **Правое внешнее соединение**  **A RIGHT OUTER JOINВ**   |  |  |  | | --- | --- | --- | | ID\_NUM | NAME | PHONE | | 1334 | Иванов | 555-85-54 | | 1399 | Петров | 555-52-23 | | 1399 | Петров | 555--1191 | | NULL | Крылов | 555-78-94 |   Таблица 2.5 | | **Полное внешнее соединение**  **A FULL OUTER JOIN В**   |  |  |  | | --- | --- | --- | | ID\_NUM | NAME | PHONE | | 1334 | Иванов | 555-85-54 | | 1399 | Петров | 555-52-23 | | 1208 | Сидоров | NULL | | 1399 | Петров | 555--1191 | | NULL | Крылов | 555-78-94 |   Таблица 2.6 |

**Соединение типа объединение**

**A UNION JOIN В**

|  |  |  |
| --- | --- | --- |
| ID\_NUM | NAME | PHONE |
| 1208 | Сидоров | NULL |
| NULL | Крылов | 555-78-94 |

Таблица2.7

**Использование JOIN в БД «Книготорговая компания»**

table\_A [тип\_соединени] JOIN table\_B ON (предикат)

Предикат в этой конструкции определят условиях соединения строк из разных таблиц

**Пример 1**: Найти авторов, живущих в одном городе с издателями

SELECT au\_id,city

FROM authors INNER JOIN publishers ON authors.city=publishers.city

**Пример 2**: Вывести для каждого издательства количество изданных книг

SELECT pub\_name,COUNT(title\_id)

FROM publishers INNER JOIN titles ON publishers.pub\_id=titles.pub\_id

GROUP BY pub\_name

**Подзапросы с квантором EXISTS**

Подзапросы, которым предшествует ключевое слово EXISTS. Осуществляют проверку существования хотя бы одной строки удовлетворяющей подзапросу.

**Особенности:**

1. Перед этим ключевым словом не должно быть названий столбцов, констант или других выражений
2. Подзапрос с квантором существования возвращает значение true или false и не возвращает никаких данных из таблицы
3. Список выбора такого подзапроса часто состоит из одной \*, так как у нас нет необходимости указывать названия столбцов, если осуществляется проверка существования строк , удовлетворяющих условиям, указанным в подзапросе.

**Задание 1.**

Найти название всех издательств, которые публиковали книги по бизнесу.

SELECT PUB\_NAME

FROM PUBLISHERS

WHERE EXISTS (SELECT \*

FROM TITLES

WHERE TITLES.PUB\_ID=PUBLISHERS.PUB\_ID

AND TYPE=’BUSINESS’)

**Задание 2.** Найти авторов которые живут в одном городе с издательством

SELECT au\_fname,au\_lname

FROM authors a

WHERE EXISTS (SELECT \*

FROM publishers p

WHERE a.city=p.city)

**Индекс**

Избыточная структура, предназначенная для ускорения поиска

**Основные предназначения:**

* Увеличения скорости доступа к данным
* Поддержка уникальности данных
* Автоматическое упорядочение записей при выборке

**Поиск с помощью индекса**

* Точное значение атрибута
* Интервал значения атрибута
* Значение нескольких атрибутов

**Способы определения индекса:**

* Автоматическое создание индекса при создании первичного ключа **PRIMARY\_KEY**
* Автоматическое создание индекса при определении ограничения целостности **UNIQUE**
* Создание индекса с помощью команды **CREATE\_INDEX**

**Создание индекса**

Create index **имя индекса** on **имя таблицы (столбец)**

**Удаление индекса**

Drop index **имя индекса** on **имя таблицы**

**Архитектура удаленных БД**

1. **Локальные базы данных (SQLite)**

На одном ПК размещается и сама база данных и ее СУБД

Работа организована в однопользовательском режиме

На приложение клиента возложены следующие функции

* Обеспечение достоверности и целостности данных
* Формирование запросов к базе данных
* Интерпретация полученных ответов

**Достоинства:**

За счет работы единственного пользователя обеспечивается высокая степень безопасности

**Недостатки:**

Возможность работы только одного пользователя

1. **Архитектура файл-сервер(Access)**

БД располагается на сетевом сервере, а приложение клиента на пользовательском ПК.

Количество пользователей не больше 10-15

Каждый пользователь переписывает на свой ПК копию БД, которая периодически обновляется с сервера

Имеется дополнительная трудность по блокировке записей, которые используются другими пользователями

Поэтому в некоторые моменты времени пользователь может видеть недостоверные сведения в БД

**Достоинства**:

Возможность работы с БД нескольких пользователей

**Недостатки**:

* Перегрузка сети
* Сложно обеспечить секретность данных
* Необходимость систематического обновления на всех ПК пользователей
* Блокировка данных

1. **Архитектура клиент-сервер**

В ее основе лежит распределенная модель вычисления. В самом общем случае под клиентом и сервером понимается 2 взаимодействующих процесса:

Сервер-логический процесс, который обеспечивает некоторый сервис по запросу от клиента. Обычно сервер не только выполняет запрос, но и управляет очередностью запросов, буфером обмена, а также извещает клиентов о выполнении запроса.

Клиент-процесс, который запрашивает обслуживание от сервера.

При данном взаимодействии инициатором диалога является клиент.

Например, сервером может выступать СУБД (back-end), а клиентом-приложение, созданное в среде разработки, которое использует данные с сервера (front-end)

**Достоинства**:

* Резко уменьшается сетевой трафик
* Единые правила для всех (высокая степень защиты)
* Практически всегда расчеты выполняются на сервере , что обеспечивает быстродействие

**Компонентный подход в программировании**

Компонентный подход предполагает построение программного обеспечения из отдельных компонентов, то есть физически отдельно существующих частей ПО, которые взаимодействуют между собой через стандартные интерфейсы

**ActiveX Data Object** (**ADO**) – программный интерфейс для доступа к данным через приложения

**Провайдер данных** – компонент манипулирующий данными. Он располагается между потребителем данных и базой данных. Как правило, все провайдеры предоставляют данные в табличном формате

**Функции** провайдера:

* Получение от потребителя запросов на получение/модификацию данных
* Получение данных из базы
* Возвращение данных потребителю

**SQLite**

Это встраиваемая кроссплатформенная СУБ

Особенности

В СУБД , которые представляют собой клиент-серверное приложение есть выделенное приложение- сервер, которое принимает запросу от клиентов и выполняет их , оно непосредственно занимается записью/ чтением данных не диск и представляет различные функции

SQLite не использует парадигму клиент —сервер. Выстраиваемая СУБД не имеет выделенного сервера, ее движок включается в приложение в виде библиотеки и предоставляй доступ к базе через специализированный API

SQLite хранит всю базу данных (включая определения ,таблицы индексы и данные) в единственном стандартном файле на том компьютере , на котором выполняется программа.

Применение:

На сегодняшний день SQLite является одной из самых популярных резаний такого типа . Она используется:

1. В музыкальных плеерах с функцией органайзера
2. В мобильных телефонах и играх
3. Браузер Google Chrome использует SQLite для хранения Cookies
4. В Skype (хранение истории сообщений)
5. Системы Airbus A350

SQLite обладает феноменальной надежностью. Несмотря на то, что она является открытой, разработчики практически не принимают сообщения об ошибках.

Значение внутри БД может принадлежать к одному из следующих типов хранения (storage class)

* NULL пустое значение
* INTEGER целое число
* REAL дробное число
* TEXT строка
* BLOB для изображений и бинарных файлов

SQLite не имеет отдельного логического класса хранения. Вместо этого, логические значения хранятся как целые числа 0 (false ) и 1 (true)

SQLite не имеют классов , предназначенных для хранения дат и/или времени . Вместо этого,встроенные функции даты и времени в SQLite способны работать с хатами и временем , сохранёнными а виде значений TEXT ,REAL и INTEGER

Строковое представлен е даты и времени в SQLite:

Для того чтобы SQLite правильно понимал и работал с датой (сортировал,сравнивал и тд), строка содержащая дату и время должна быть в одном из следующих форматов:

- YYYY-MM-DD

- YYYY-MM-DD HH:MM

- YYYY-MM-DD HH:MM:SS

- YYYY-MM-DD HH:MM:SS:SSS

- HH:MM

- HH:MM:SS

-HH:MM:SS:SSS

В SQLite3 по умолчанию отключена поддержка внешних ключей, команде PRAGMA позволяет включить внешние ключи в базах данных  SQLite.

PRAGMA foreign\_keys=on

CHECK ограничение

CHECK ограничение даёт возможность проверить значение, введённое в запись.

Если условие принимает значение false ,то это ограничение  нарушено и значение не вводится в таблицу.

CREATE TABLE COMPANY(

…

SALARY REAL CHECK(SALARY>0)

);

**Работа с данными . Технология ADO.NET**

ADO.NET нацелена на автономную работу с помощью объектов DataSet. Эти типы представляют локальные копии любого количество взаимосвязанных таблиц данных , каждая из которых содержит набор строк и столбцов.

Объекты DataSet позволяют работать с содержимым DataSet, изменять его, не требуя подключения к источнику данных. Также можно отправлять обратно блоки измененных данных, с помощью соответубщего адаптера данных.

 В ADO. NET имеются различные поставщики данных (data provider) , каждый из которых оптимизирован для взаимодействия с конкретной СУБД.

Каждый поставщик данных определяет набор классов, обеспечивающих основную функциональность .

Конкретные имена этих основных классов различаются к различных поставщиков (например SQLConnection,OracleConnection, OledConnection, MySQLConnection), но все эти объекты порождены от одного и того же базового класса (в случае объектов подключения это DbConnection)

Дистрибутив .Net ,поставляемый Microsoft , содержит множество различных поставщиков данных



Если необходимо взаимодействие с базой данных MS Access это можно сделать с помощью поставщика данных OLE DB или ODBC.

Основу интерфейса взаимодействия с базами данных в ADO.NET  представляет следующий круг объектов :

* Connection
* Command
* DataReader
* DataSet
* DataAdapter

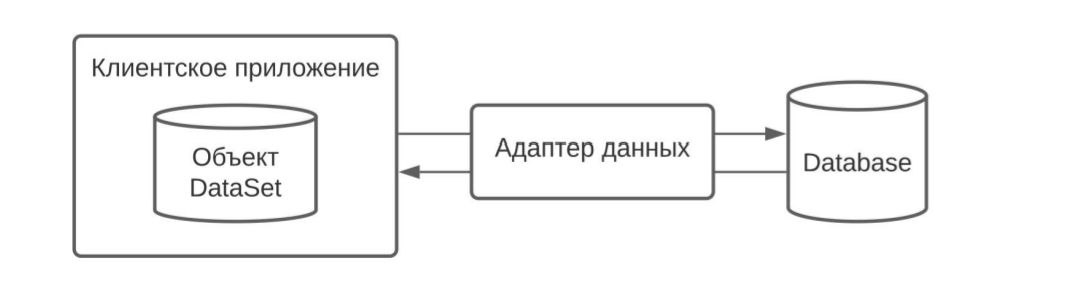
С помощью объекта Connection происходит установка подключения к источнику данных.

Объект Command позволяет выполнять операции с данными из БД.

Объект DataReader считывает полученные в результате запроса данные.

Объект DataSet предназначен для хранения данных из БД и позволяет работать с ними независимо от БД.

Объект DataAdapter является посредником между DataSet и источником данных. Главным образом, через эти объекты и будет идти работа с базой данных.



DataAdapter содержит метод Fill() для обновления данных из базы и заполнения DataSet.

Объекты адаптеров пересылают данные между вызывающи процессом и источником данных с помощью объектов DataSet.

Тип DataSet представляет собой контейнер для любого количества объектов DataTable

Объекты DataTable используются для представления таблиц в DataSet. DataTable представляет одну таблицу из базы данных.

Как только получен объект DataSet, происходит отключение от базы данных, и работа ведется с локальной копией удаленных данных.

Теперь в этом наборе данных можно вставлять, удалять или изменять строки различных объектов DataTable, но физически база данных не обновляется, пока вызывающий процесс явно не передаст набор DataSet адаптеру данных для обновления.

По сути объекты DataSet имитируют постоянное подключение клиентов.

Функционально классы ADO.NET можно разбить на два уровня: подключенный и отключенный. Каждый провайдер данных .NET реализует свои версии объектов Connection, Command, DataReader, DataAdapter и ряда других, который составляют подключенный уровень. То есть, с помощью них устанавливается подключение к БД и выполняется взаимодействие с ней.

Другие классы, такие как DataSet, DataTable, DataRow, DataColumn и ряд других составляют отключенный уровень, так как после извлечения данных в DataSet мы можем работать с этими данными независимо от того , установлено подключение или нет. То есть после получения данных из БД приложение может быть отключено от источника данных.

Компонент **Connection** имеет свойство **ConnectionString**, в котором фиксируется вся необходимая для установления соединения с БД информация. Кроме того, поддерживается ряд методов, позволяющих обрабатывать данные с применением транзакций.

Свойства объекта Connection позволяют:

* задавать реквизиты пользователя
* указывать расположение источника данных

**Параметры строки подключения к БД**

Свойство **ConnectionString** получает ссылку на строку символов, содержащую необходимую для установления соединения информацию:

**Provider** – имя OLE DB провайдера

**Database** – имя текущей базы данных;

**DataSource** – имя сервера или имя файла-источника данных

Представитель класса **Command** поддерживает следующие методы:

* **ExecuteNonQuery** – обеспечивает выполнение команд, не возвращающих данные, например INSERT, UPDATE, DELETE;

* **ExecuteScalar** – исполняет запросы к БД, возвращающие единственное значение;
* **ExecuteReader** – возвращает результирующий набор через объект **DataReader**

**Entity Framework**

Представляет собой специальную объектно-ориентированную технологию на базе Фреймворка для работы с данными

**ORM-object relational mapping**-технология программирования, которая связывает базы данных с концепциями объектно-ориентированных ЯП, создавая виртуальную объектную базу данных.

Если традиционные средства ADO.NET позволяют создавать подключения, команды для взаимодействия с БД, то **Entity Framework** это более высокий уровень абстракции, который позволяет абстрагироваться от самой БД и позволяет работать с данными независимо от типа хранилища.

Если на физическом уровне мы оперирует таблицами, первичными и внешними ключами, то на концептуальном уровне, который предлагает EF работа ведется уже с объектами.

Центральными объектов является понятие сущности Entity, которое представляет собой набор данных, ассоциированных с определенным объектом.

Отличительной особенностью является использование запросов LINQ для выборки данных из базы данных.

**LINQ-language integrared query(запросы, интегрированные в язык)**

**Entity Data Model**

Эта модель сопоставляет классы сущностей с реальными таблицами в базе данных.

Состоит из трех уровней

-концептуальный

-уровень хранилища

-уровень сопоставления(маппинга)

На концептуальном уровне происходит определение классов сущностей, используемых в приложении.

Уровень хранилища определяет таблицы, столбцы, отношения между таблицами и типы данных с которыми сопоставляется используемая база данных.

Уровень сопоставления служит посредником между первыми двумя, определяя сопоставления между свойствами класса сущности и столбцами таблиц.

Entity Framework предполагает три возможных способа взаимодействия с БД

-DataBase First(Создает набор классов, которые отражают модель конкретной БД)

-Model First(Сначала разработчик создает модель базы данных, по которой затем с помощью Entity Framework cсоздаётся реальная БД на сервере)

-Code First(Разработчик создает класс модели, который будет хранится в БД, а затем по этой модели генерируется БД и ее таблица)

Стоит учитывать, что Еntity Framework выступает прослойкой между приложением и базой данных, поэтому может ухудшаться производительность при взаимодействии с БД.

**Формат JSON**

Переменные, массивы и объекты – привычная и удобная форма представления данных

Формат JSON позволяет обобщить их в одно целое и не делать акцент на языке программирования.

При этом данные превращаются в пары: "**имя = значение**". Значение в каждой из них также может быть совокупностью таких пар.

**JSON(Object Notation JavaScript)-простой, основанный на использовании текста, способ хранить и передавать структурированные данные.**

Наиболее частое распространенное использование JSON – пересылка данных от сервера к браузеру

**Есть несколько основных правил для создания строки JSON:**

Строка JSON содержит либо массив значений, либо объект (ассоциативный массив пар имя/значение)

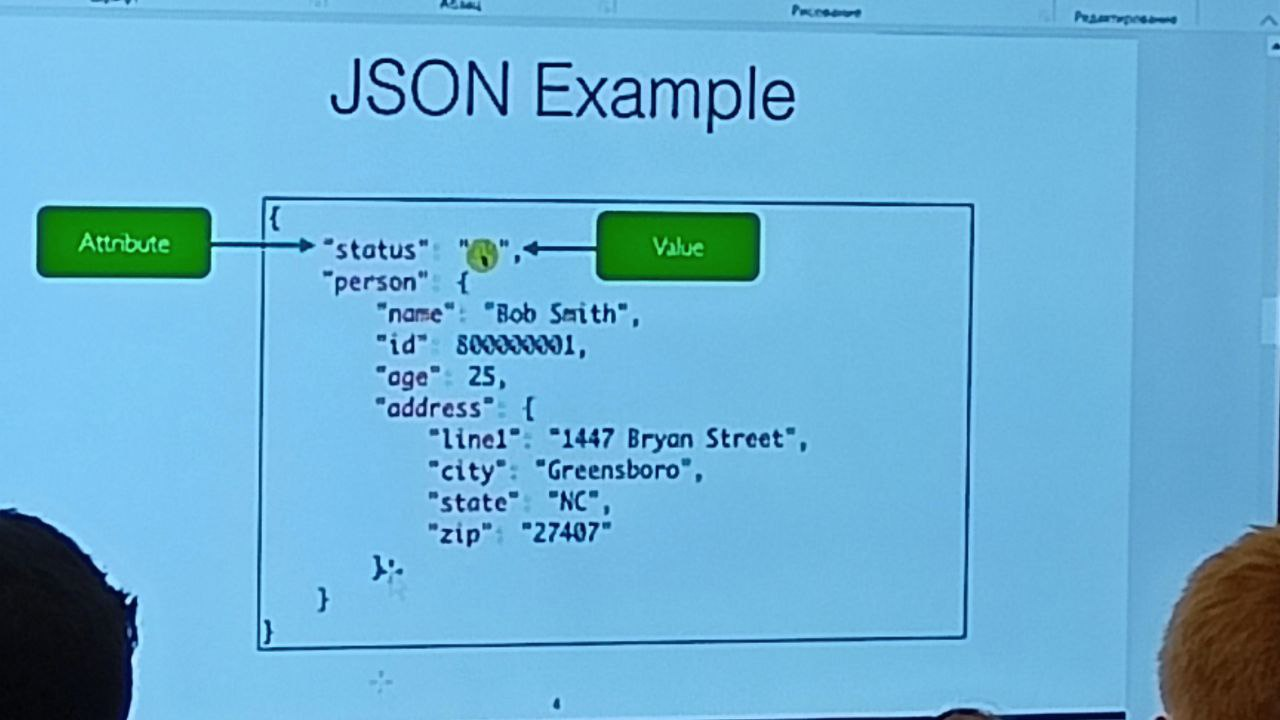
**Массив** заключается в квадратные скобки **[ , ]** и содержит разделенный запятой список значений

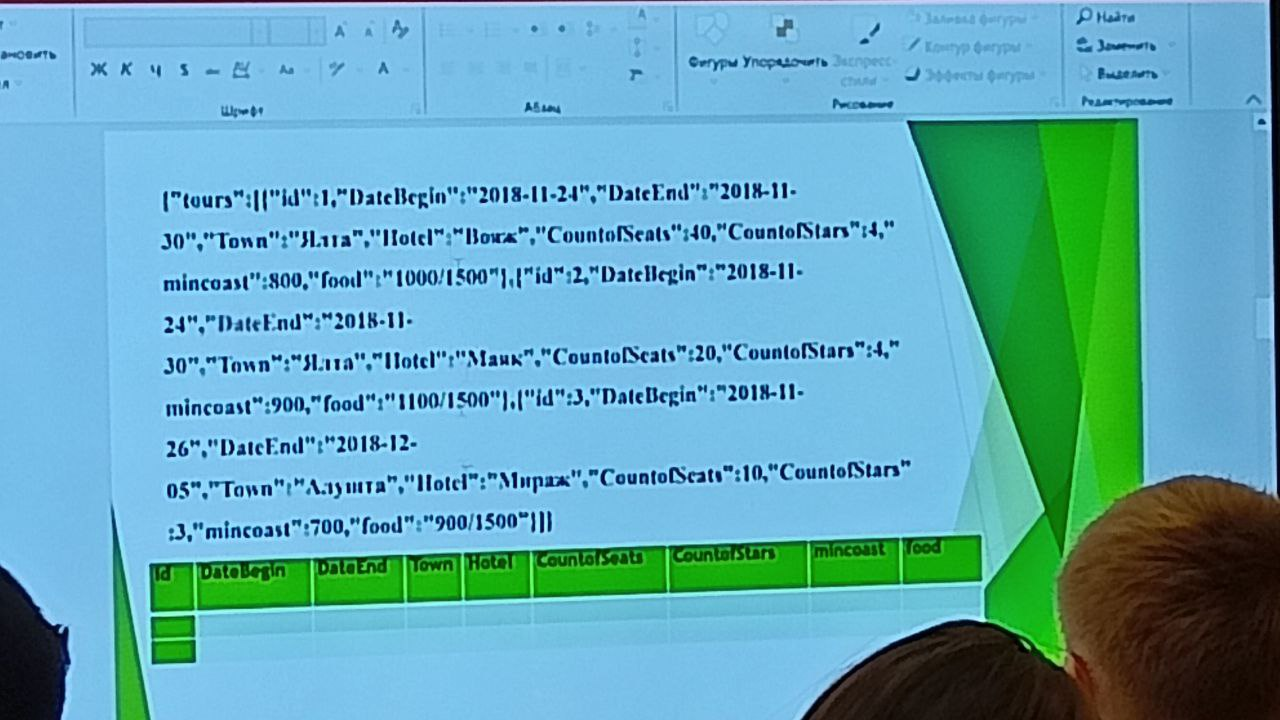
**Объект** заключается в фигурные скобки **{ , }** и содержит разделенный запятой список пар имя/значение  
Пара **имя/значение** состоит из имени поля, заключенного в двойные кавычки, за которым следует двоеточие **( : )** и значение поля

Значение в массиве или объекте может быть:

* Числом (целым или с плавающей точкой)
* Строкой (в двойных кавычках)
* Логическим значением (true или false)
* Другим массивом (заключенным в квадратные скобки)
* Другой объект (заключенный в фигурные скобки)
* Значение null

Чтобы включить двойные кавычки в строку, нужно использовать экранирование: **\"**





**Сериализация** – это процесс перевода структуры данных в последовательность битов, или в другую структуру данных, которую удобно хранить и передавать

**Десериализация** – это обратный процесс, преобразование сериализованных данных в структуру данных

Библиотека «Newtonsoft.Json» для работы с JSON

Опишем класс, который будем сериализовать:

class MyMusic {

public Track[] Tracks;

}

class Track {

public string Artist;

public string Album ;

public string Title ;

public string Year ;

}