ER-МЕТОД

2025-04-22

ENTITY-RELATIONSHIP METHOD

- Entity – сущность
- Relationship – связь
- **Метод сущностей и связей**
- Описан Питером Ченом в 1976 году
- Выполняется в два этапа:
  1. построение модели предметной области (ER-модель)
  2. преобразование её в модель базы данных (например, в реляционную модель)

ПРЕИМУЩЕСТВА И НЕДОСТАТКИ

- ER-метод может применяться для больших и очень больших предметных областей с большим количеством атрибутов.
- Является более интуитивно понятным.
- В отличие от метода нормализации, ER-метод не гарантирует, что в получившихся отношениях нет нарушений нормальных форм.
- Лучше всего использовать эти два метода совместно.

УРОВНИ МОДЕЛЕЙ

1. **Концептуальная модель (инфологическая модель)** – описывает предметную область без учёта специфики БД (например, ER-модель).
2. **Логическая модель (даталогическая модель)** – учитывает вид БД, но без учёта конкретной СУБД (примеры моделей: реляционная, линейная, иерархическая, объектная, и т.д.)
3. **Физическая модель (полная атрибутивная модель)** – учитывает технические моменты конкретных СУБД, вплоть до типов атрибутов.

ЭТАПЫ ПРОЕКТИРОВАНИЯ РЕЛЯЦИОННОЙ БД ПРИ СОВМЕЩЕНИИ ER-МЕТОДА И НОРМАЛИЗАЦИИ

№	Этап	Уровень	Метод
1	Построение ER-модели	**Концептуальный** (инфологический)	ER-метод
2	Преобразование ER-модели в реляционную модель	**Логический** (даталогический)	ER-метод
3	Проверка нормальных форм для всех отношений	**Логический** (даталогический)	нормализация
4	Определение названий и типов для атрибутов и ключей	**Физический** (полный атрибутивный)	
5	Написание SQL-запросов для создания таблиц БД	**Физический** (полный атрибутивный)	

ИЗ ЧЕГО СОСТОИТ ER-МОДЕЛЬ?

- **Сущности**
- **Атрибуты сущностей**
- **Связи между сущностями**

1. **Сущность** – это некий общий класс информационных объектов, для которых характерен одинаковый набор свойств (параметров).
2. **Атрибут** – это свойство сущности, которое обязательно должно принимать ровно одно значение у каждого экземпляра сущности, и не должно быть связано с другими сущностями.
3. **Связь** – это информация, относящаяся сразу к двум сущностям.

ЧАСТЫЕ ОШИБКИ В ER-МОДЕЛЯХ

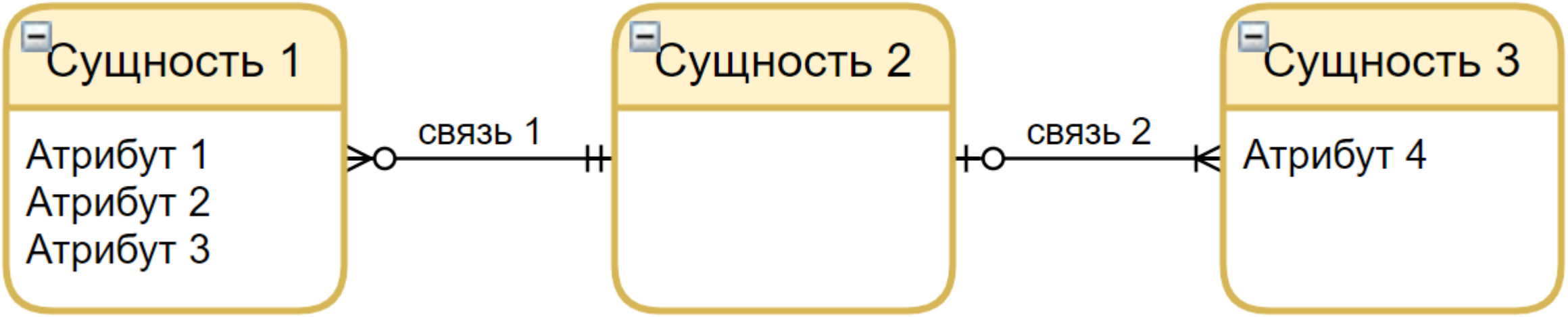
- В ER-модели не должно быть искусственной информации, которой нет в предметной области (не должно быть суррогатных id)!
- Информация, являющаяся сущностью, не может быть атрибутом другой сущности! Сущность не должна содержать в себе атрибуты чужих сущностей! Если какая-то информация касается сразу двух сущностей, её нужно представить не как атрибут, а как связь!
- Если атрибут может принимать сразу несколько значений у одного экземпляра сущности, то его нужно описать как отдельную сущность! Если нет уверенности, должна ли информация быть атрибутом, сущностью или связью, то лучше сделать её сущностью.
- Не нужно описывать в ER-модели вторичную информацию, которая может быть автоматически получена на основе первичной информации (например, кол-во).

ER-ДИАГРАММЫ

- ER-диаграмма используется для визуального представления ER-модели
- Существуют различные варианты записей (нотации), например:
- **Нотация Чена**:
  сущности – прямоугольники, атрибуты – овалы, связи – ромбы.
- **Нотация Crow's Foot** («воронья лапка»):
  сущности – прямоугольники со скруглёнными углами, внутри которых перечислены атрибуты; а связи обозначаются линиями.

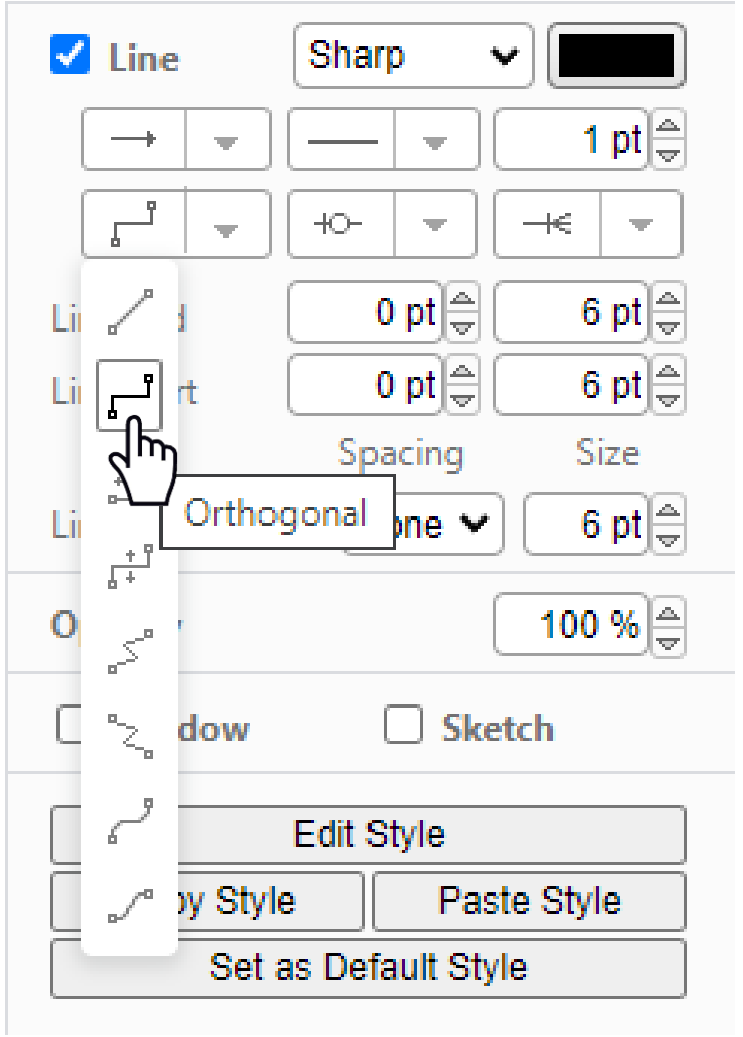
ЭЛЕМЕНТЫ ER-ДИАГРАММЫ CROW'S FOOT

- **Сущности**
- **Атрибуты сущностей**
- **Связи между сущностями**



РИСОВАНИЕ ER-ДИАГРАММ

- ER-диаграммы удобно создавать в онлайн-редакторе **draw.io** (diagrams.net)
- Сущности принято называть в единственном числе.
- Для **линий связей** рекомендуется выбирать тип **Orthogonal** на панели свойств, чтобы проще было настраивать положение линий так, чтобы они не пересекались.
- Сохраняйте результат в формате XML, чтобы можно было потом открыть схему и отредактировать её.



ЧАСТЫЕ ОШИБКИ В ER-МОДЕЛЯХ

- В ER-модели не должно быть искусственной информации, которой нет в предметной области (не должно быть суррогатных id)!
- Информация, являющаяся сущностью, не может быть атрибутом другой сущности! Сущность не должна содержать в себе атрибуты чужих сущностей! Если какая-то информация касается сразу двух сущностей, её нужно представить не как атрибут, а как связь!
- Если атрибут может принимать сразу несколько значений у одного экземпляра сущности, то его нужно описать как отдельную сущность! Если нет уверенности, должна ли информация быть атрибутом, сущностью или связью, то лучше сделать её сущностью.

СТЕПЕНИ СВЯЗЕЙ

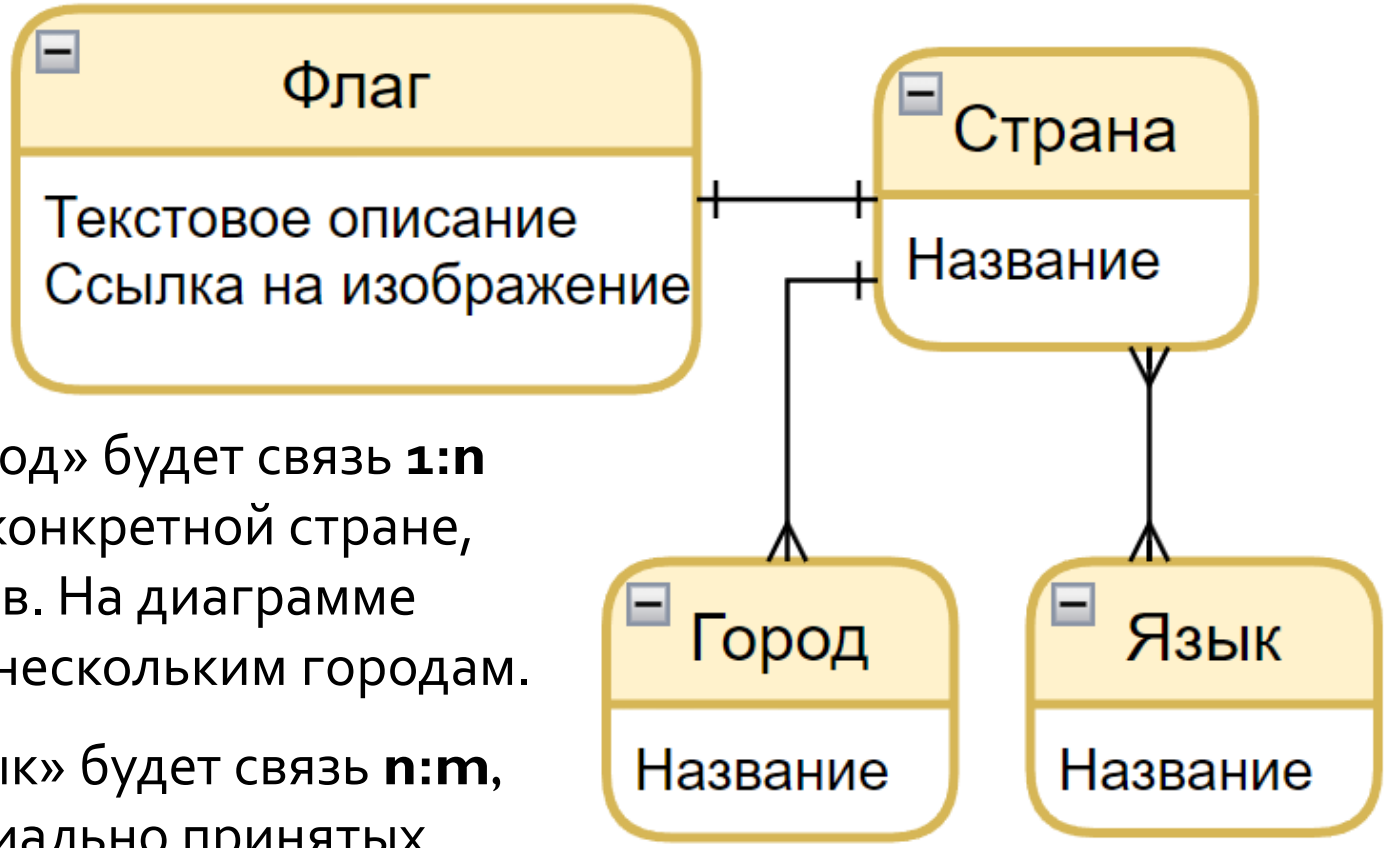
- **Один к одному – 1:1**

- **Один ко многим – 1:n**

- **Многие ко многим – m:n**

ПРИМЕРЫ СТЕПЕНЕЙ СВЯЗИ

- Между сущностями «Флаг» и «Страна» будет связь **1:1**, если хранить информацию об официальных флагах стран на текущий момент времени.



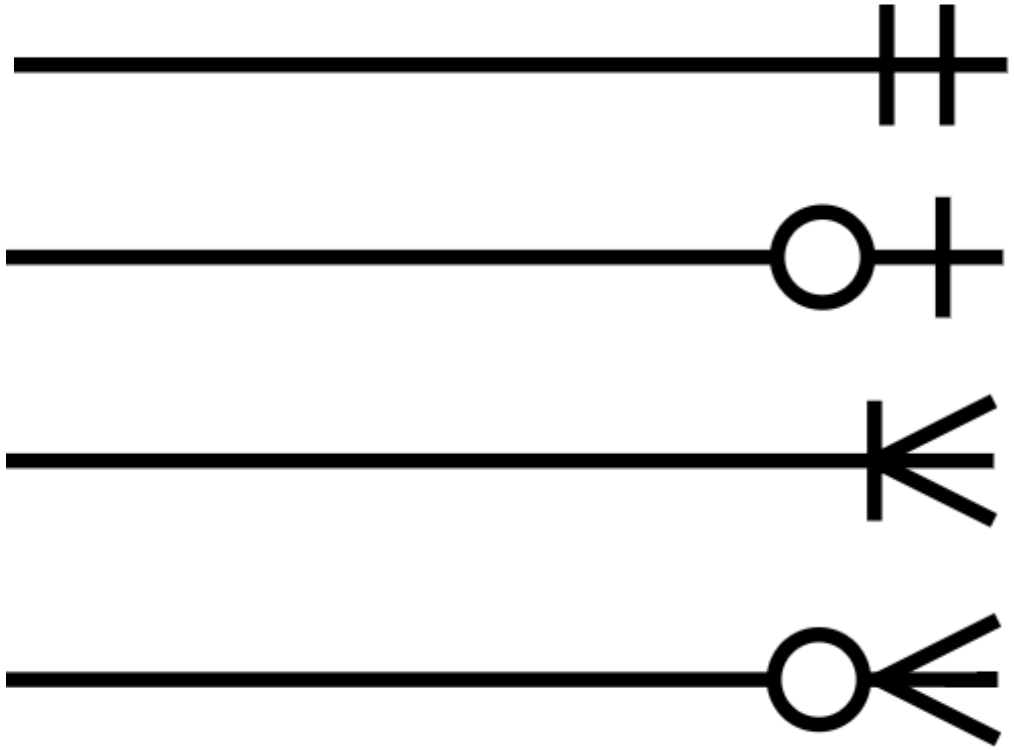
- Между сущностями «Страна» и «Город» будет связь **1:n** – каждый город находится в одной конкретной стране, а в стране может быть много городов. На диаграмме конец линии как бы «расходится» к нескольким городам.
- Между сущностями «Страна» и «Язык» будет связь **n:m**, если хранить информацию об официально принятых государственных языках стран (в стране может быть несколько официальных языков, и один и тот же язык может быть государственным в нескольких странах).

КЛАСС ПРИНАДЛЕЖНОСТИ

Каждый конец связи может иметь класс принадлежности «**обязательный**» (обозначается чертой) или «**необязательный**» (обозначается кружком).

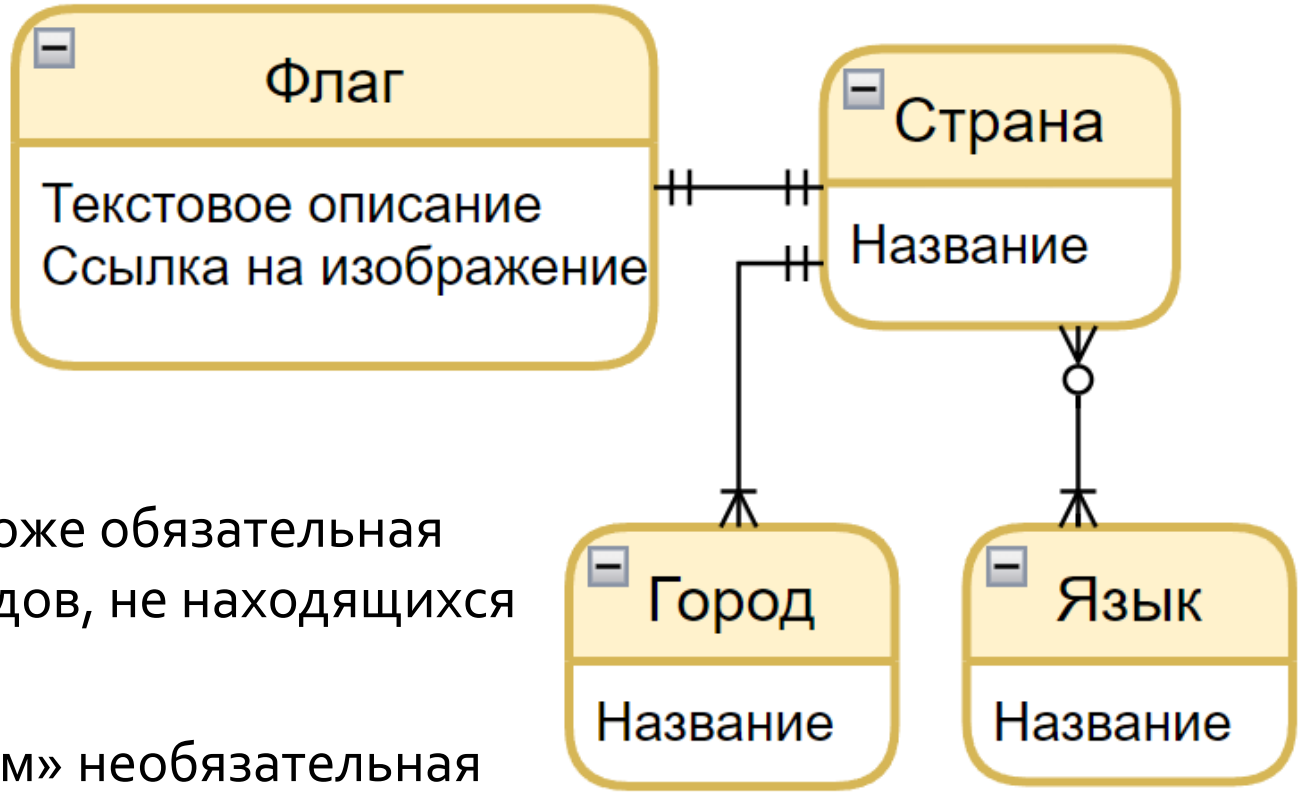
Возможные концы связей:

- **«обязательный к одному»**
  (означает, что это связь ровно с одним объектом – ни больше, ни меньше).
- **«необязательный к одному»**
  (означает связь с одним объектом или с нулём объектов).
- **«обязательный ко многим»**
  (означает связь минимум с одним объектом, но можно и больше).
- **«необязательный ко многим»**
  (количество связанных объектов может быть любым начиная от нуля).



ПРИМЕРЫ КЛАССОВ ПРИНАДЛЕЖНОСТИ

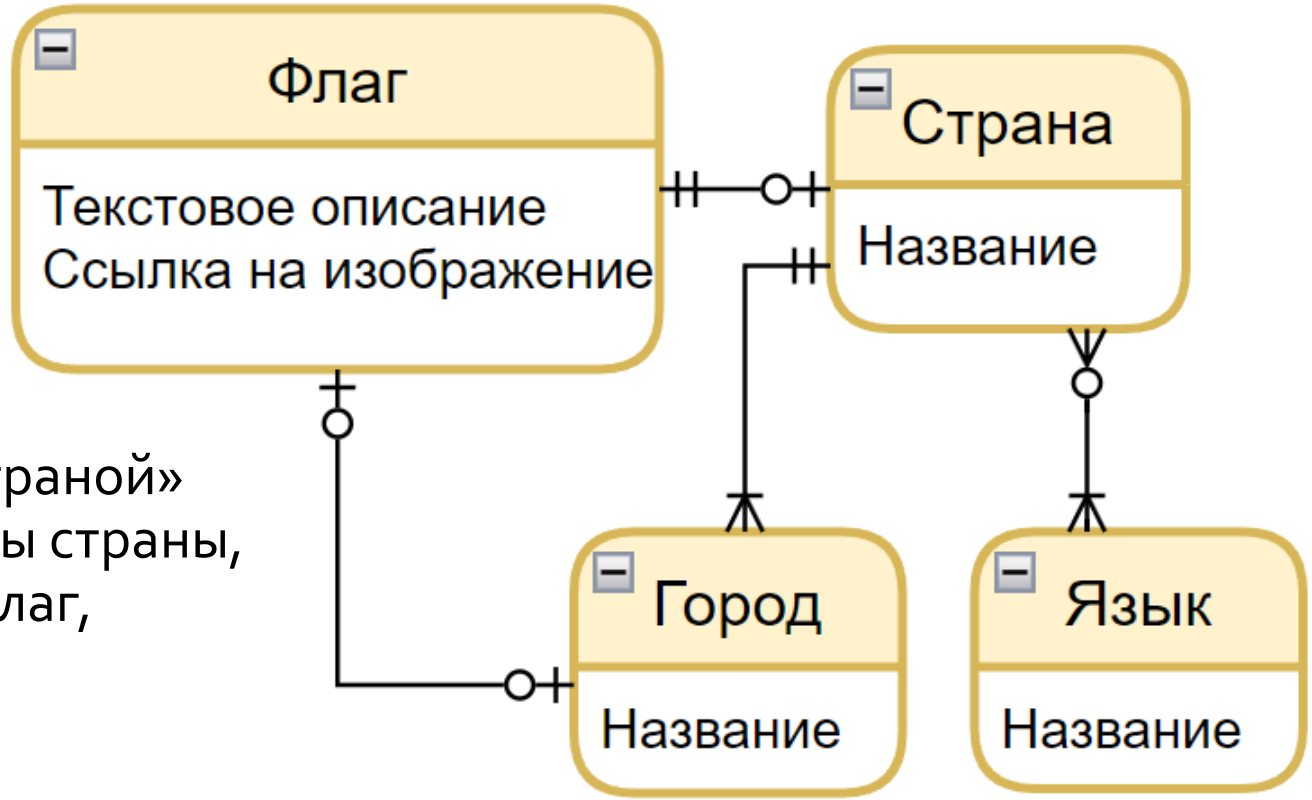
- Если в нашей предметной области рассматриваются только флаги стран, то каждый флаг обязательно должен принадлежать какой-то стране. И каждая страна имеет флаг. Поэтому связь «Флага» и «Страны» обязательная с обеих сторон.



- Связь между «Страной» и «Городом» тоже обязательная с обеих сторон, так как не бывает городов, не находящихся в странах, и нет стран без городов.
- А вот связь между «Страной» и «Языком» необязательная с одной из сторон: страна обязательно имеет хотя бы один государственный язык (рисуем палочку рядом с сущностью «Язык»), но могут быть языки, не являющиеся государственными ни в одной стране (рисуем кружок рядом с сущностью «Страна»).

ИЗМЕНЕНИЕ КЛАССА ПРИНАДЛЕЖНОСТИ

- Если же в нашей предметной области нужно рассматривать флаги не только стран, но и флаги городов, тогда мы добавляем связь между сущностями «Флаг» и «Город».

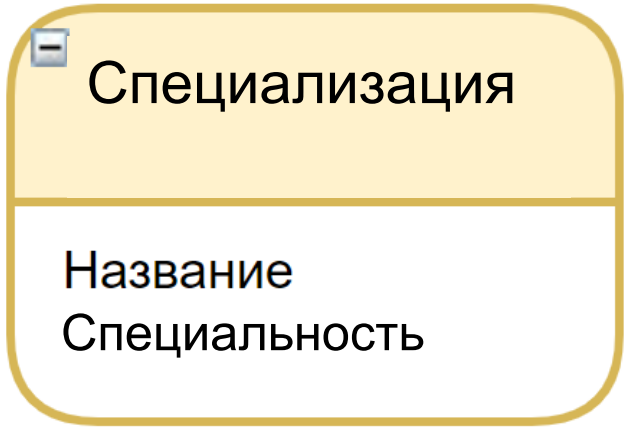


- При этом связь между «Флагом» и «Страной» становится необязательной со стороны страны, потому что теперь у нас может быть флаг, который связан с нулём стран (если это флаг города).
- Связь между «Флагом» и «Городом» будет необязательной с обеих сторон, потому что флаг может быть не связан ни с одним городом (если это флаг страны), а некоторые города не имеют собственного флага.

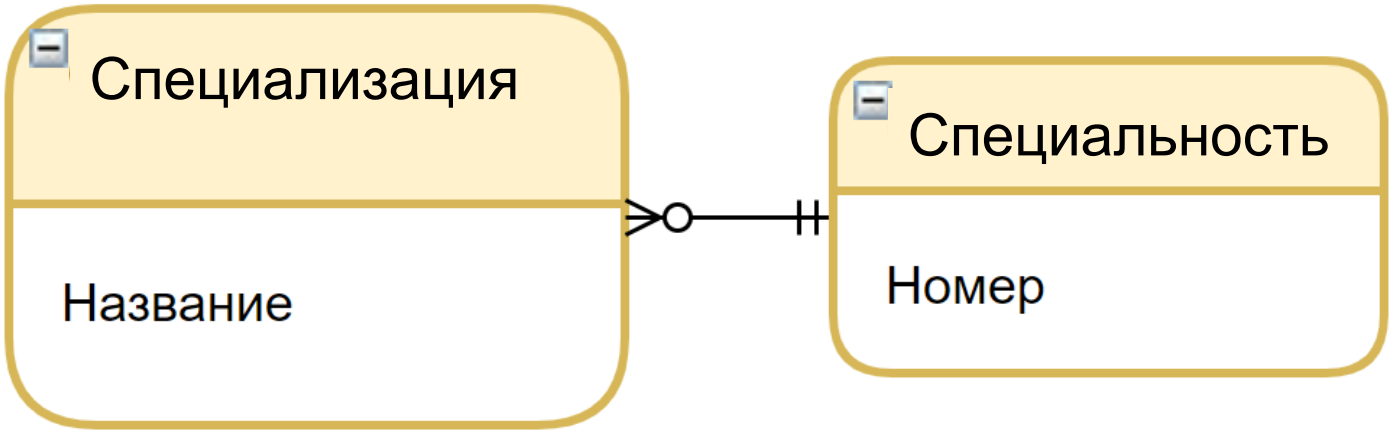
ПРЕОБРАЗОВАНИЕ АТРИБУТА В СУЩНОСТЬ

- Атрибут сущности при необходимости можно сделать отдельной сущностью, связанной с данной сущностью.

Было:



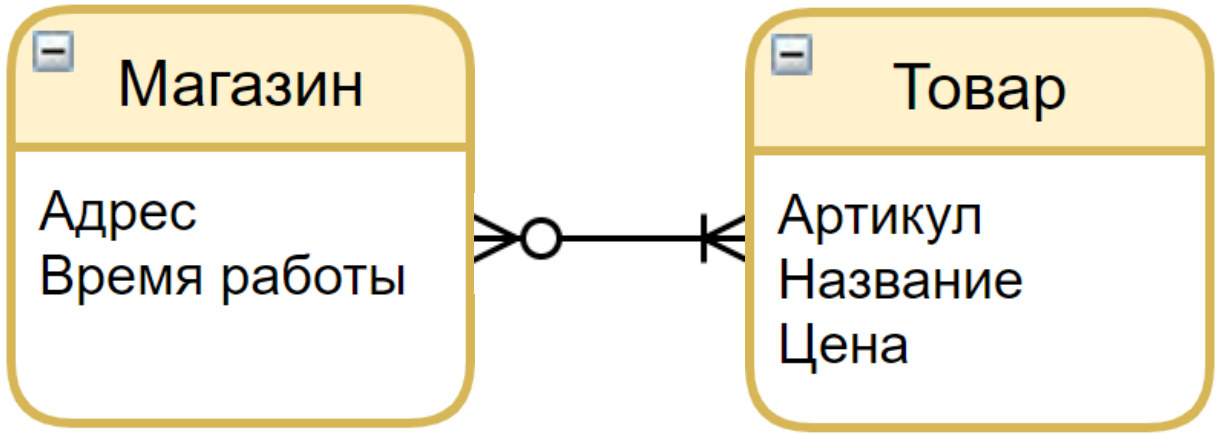
Стало:



ПРИМЕР: СВЯЗЬ КАК СУЩНОСТЬ

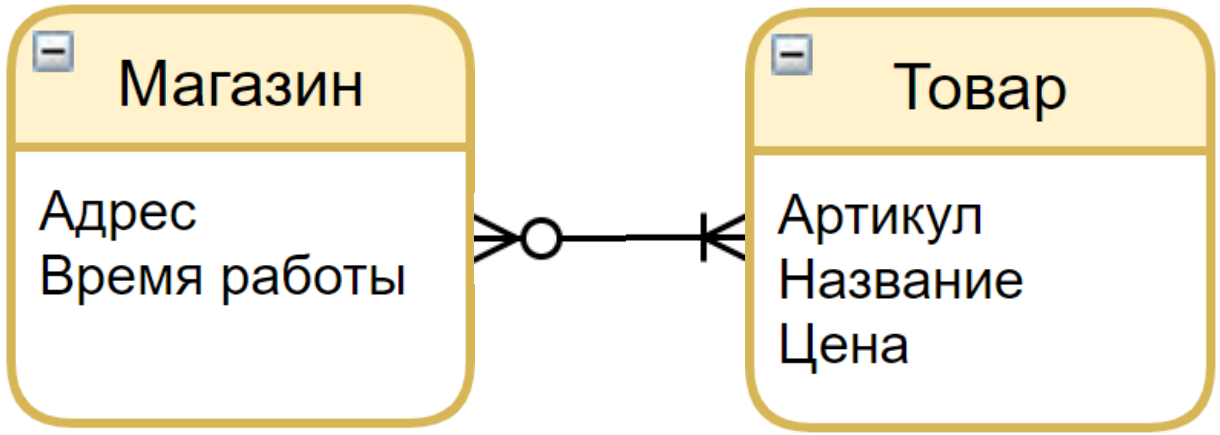
- Связь между двумя сущностями при необходимости можно сделать отдельной сущностью, связанной с этими двумя сущностями.

Было:



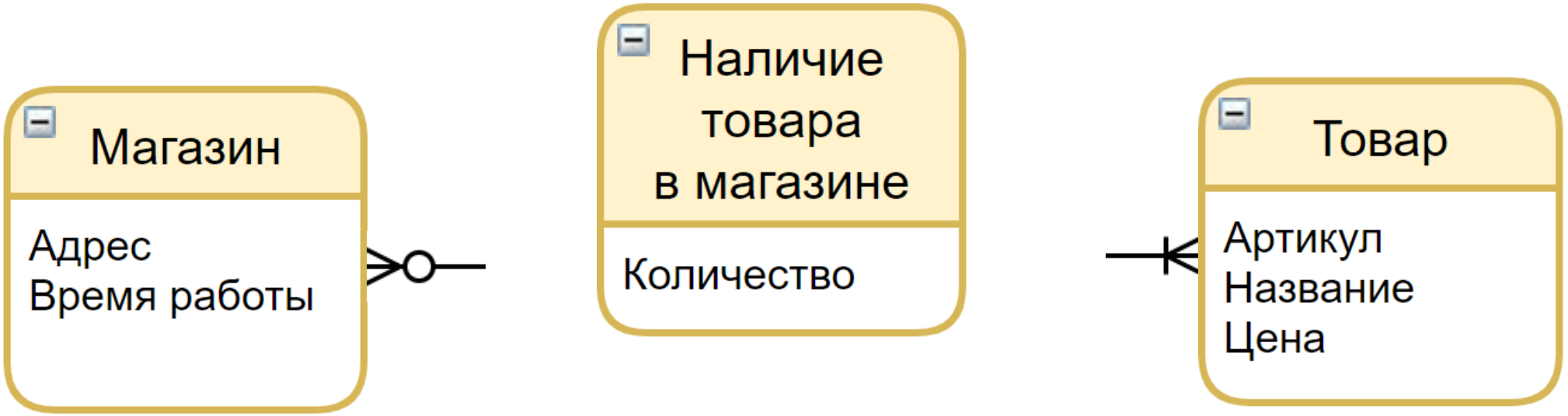
ПРИМЕР: СВЯЗЬ КАК СУЩНОСТЬ

1. Разрываем связь:



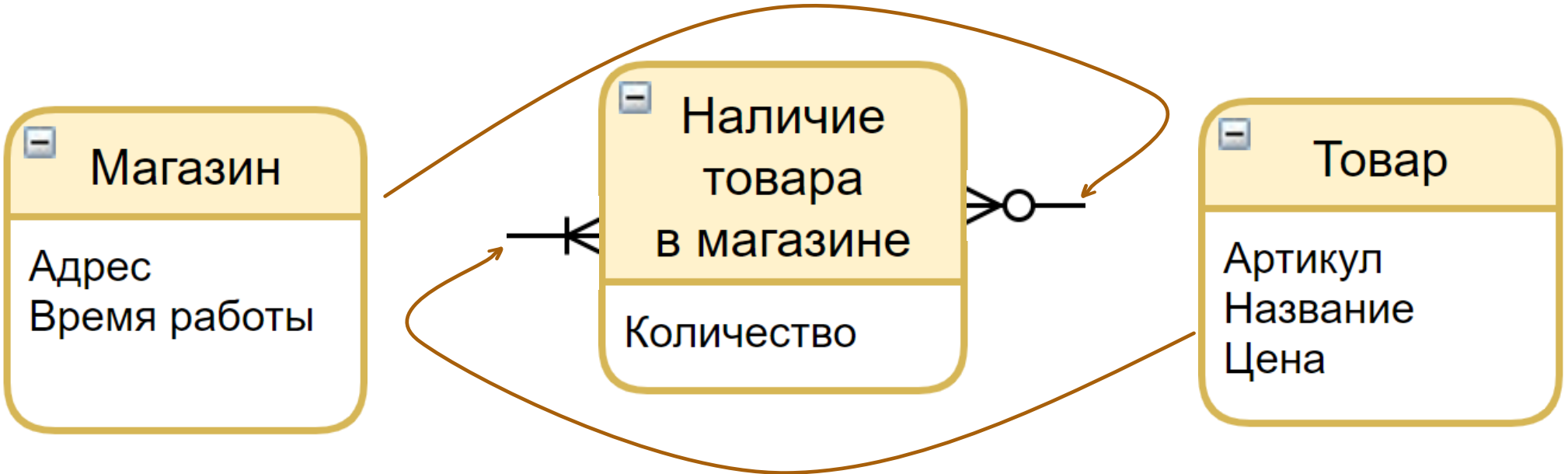
ПРИМЕР: СВЯЗЬ КАК СУЩНОСТЬ

2. Создаём новую сущность:



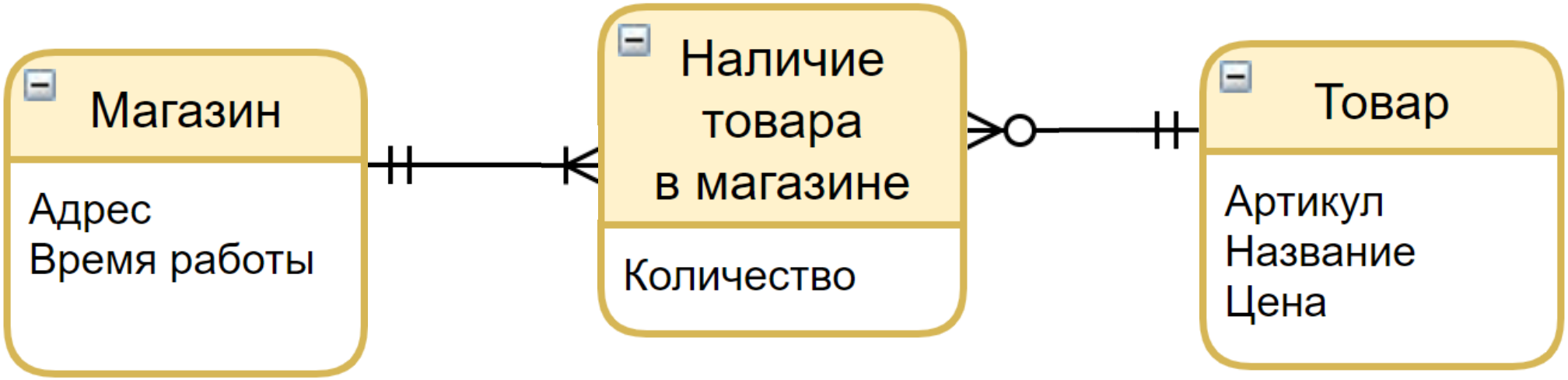
ПРИМЕР: СВЯЗЬ КАК СУЩНОСТЬ

3. Перемещаем концы связей, не поворачивая их:



ПРИМЕР: СВЯЗЬ КАК СУЩНОСТЬ

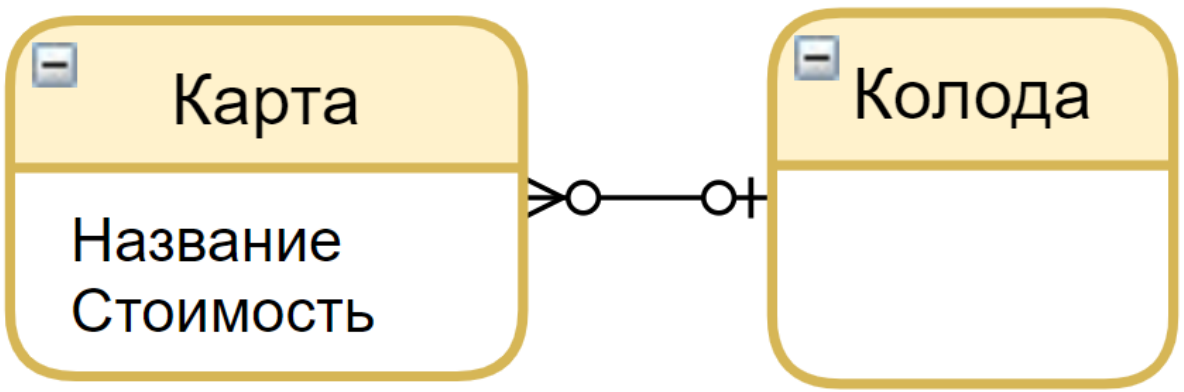
4. Дополняем концы новых связей двумя черточками:



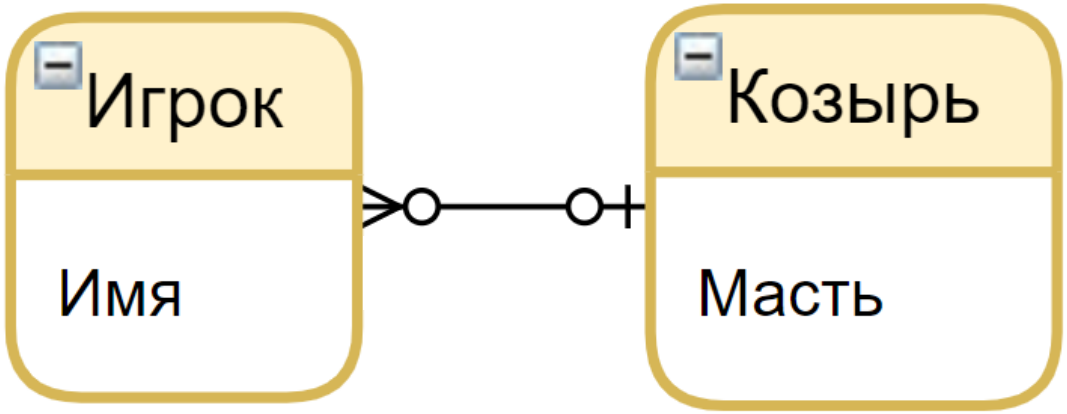
ОСОБЫЕ СЛУЧАИ: НАЗВАНИЯ СВЯЗЕЙ

- Обычно именовать связи не требуется,
  но иногда без названий непонятно, что это за связь.

Пример очевидной связи:



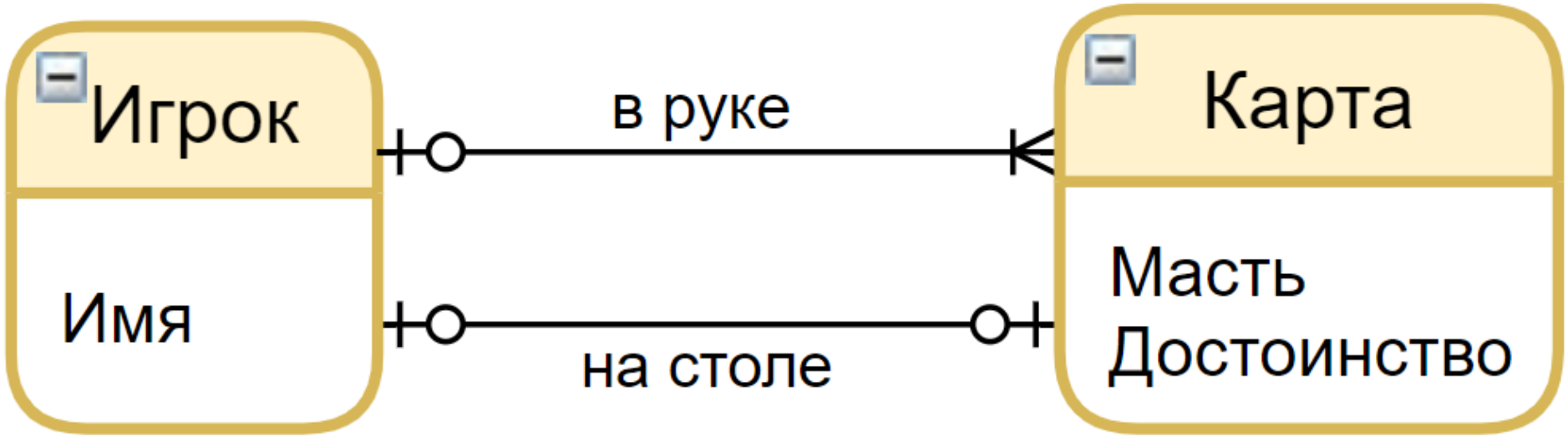
Пример неочевидной связи:



ОСОБЫЕ СЛУЧАИ: НЕСКОЛЬКО СВЯЗЕЙ МЕЖДУ ДВУМЯ СУЩНОСТЯМИ

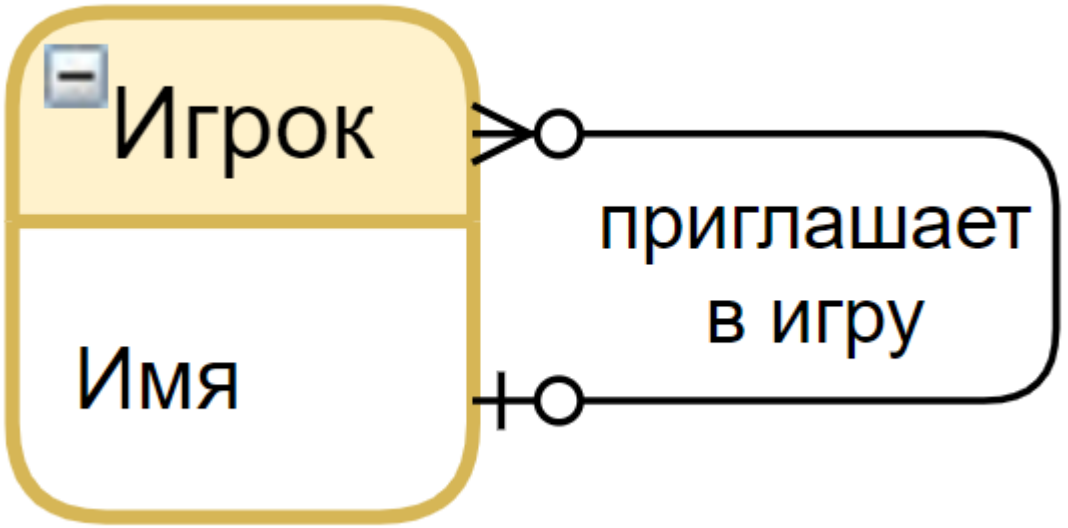
- Две сущности могут быть связаны несколькими разными связями.
- В этом случае обязательно нужно указать названия этих связей.

Пример: принадлежащие игроку карты могут быть у него в руке (скрыты от других игроков) или перед ним на столе (открыты).



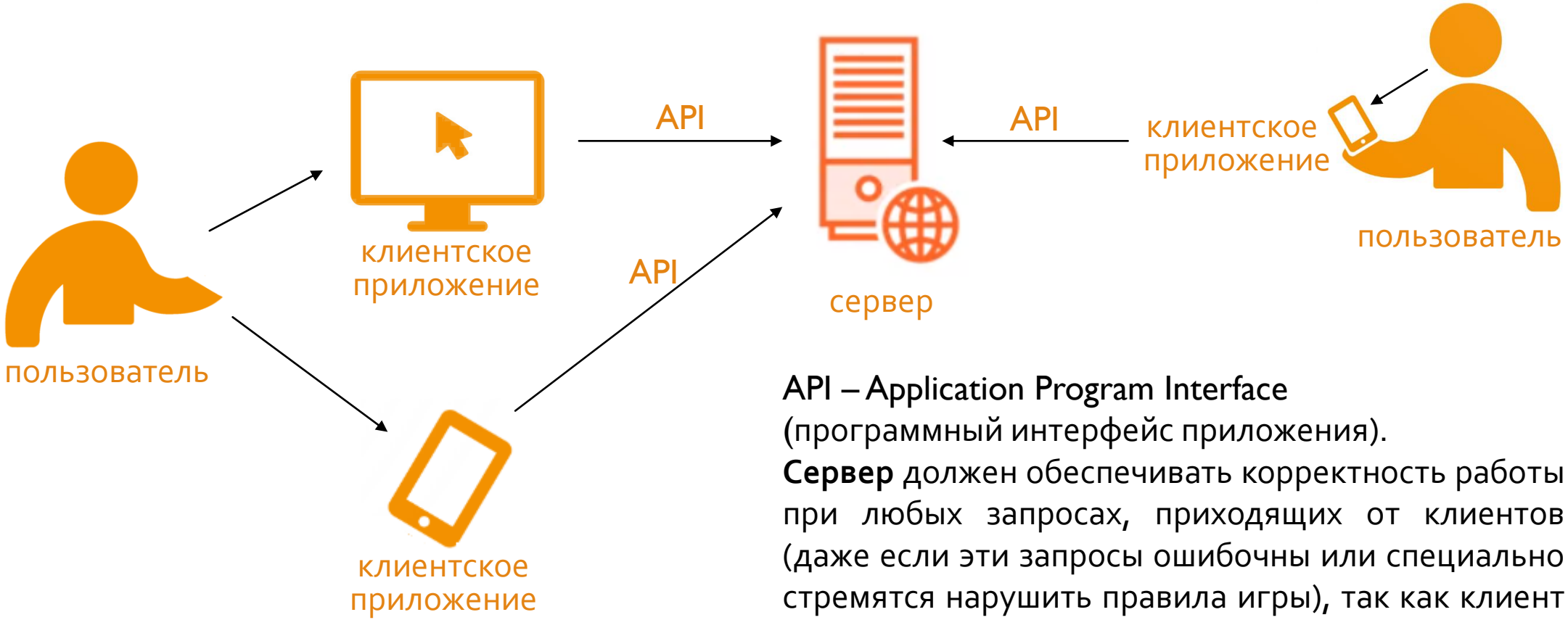
ОСОБЫЕ СЛУЧАИ: СВЯЗЬ СУЩНОСТИ С САМОЙ СОБОЙ

Пример:



- В этом случае может потребоваться не только указать название связи, но и подписать каждый из концов этой связи.
- Но лучше, чтобы не запутаться, рассмотреть эту связь как сущность.

КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА



API – Application Program Interface (программный интерфейс приложения).

Сервер должен обеспечивать корректность работы при любых запросах, приходящих от клиентов (даже если эти запросы ошибочны или специально стремятся нарушить правила игры), так как клиент может быть создан кем угодно.

UML-ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

- Для удобства можно сначала изобразить последовательность взаимодействия пользователя, клиента и сервера СУБД с помощью UML-диаграммы, называемой диаграммой последовательности взаимодействия (Sequence Diagram).
- Важно учитывать, что SQL-сервер не может посылать запросы клиенту. Он может только отвечать на запросы клиента. Поэтому, чтобы пользователь мог узнавать об изменениях в игре, произошедших без его участия (например, о ходах других игроков), клиент должен автоматически отправлять запросы на сервер через какие-то промежутки времени (например, каждые 5 секунд).
- Чтобы соперники не могли подглядеть скрытую информацию других пользователей, у каждого пользователя должен быть пароль, чтобы сервер предоставлял ему только доступную ему информацию.

ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

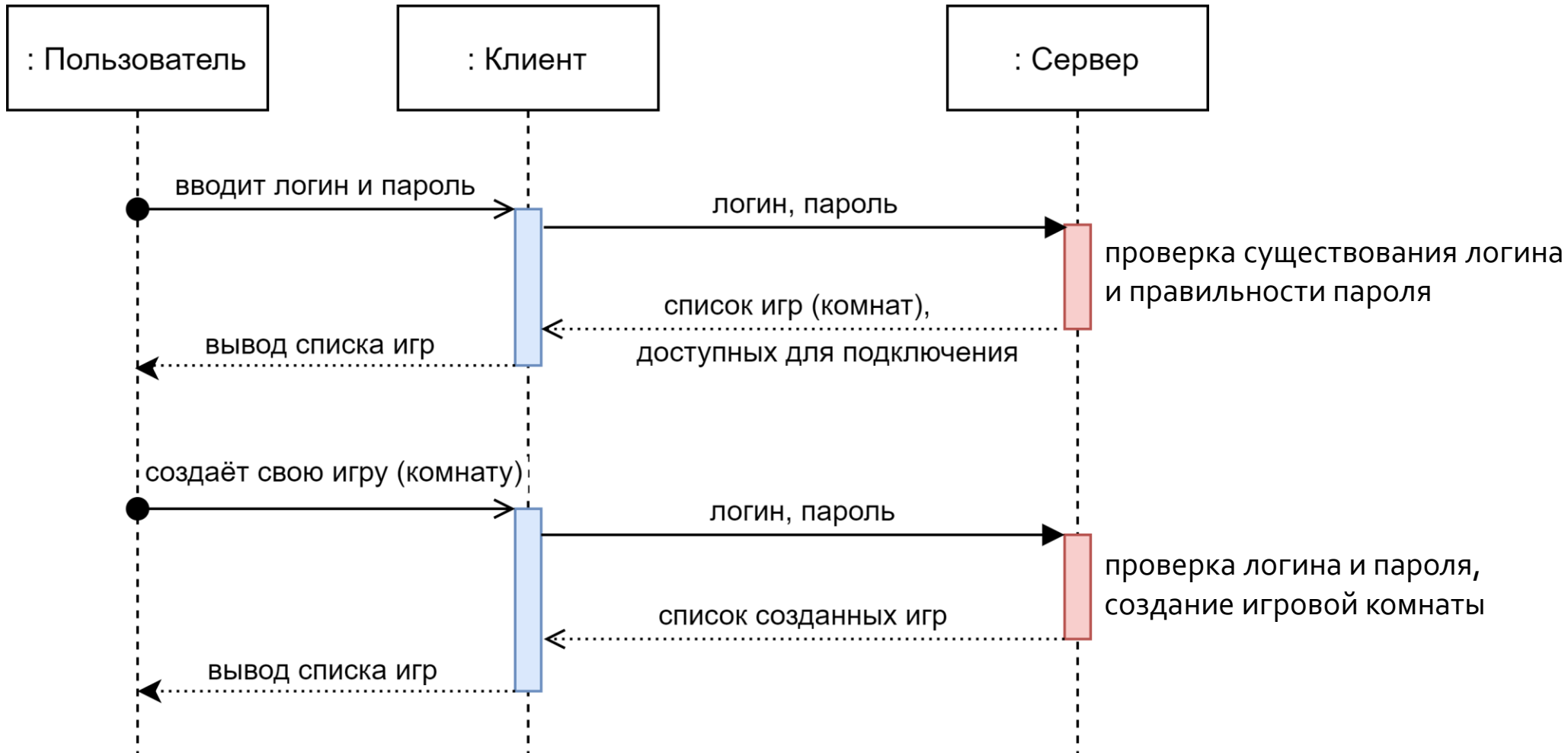


ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

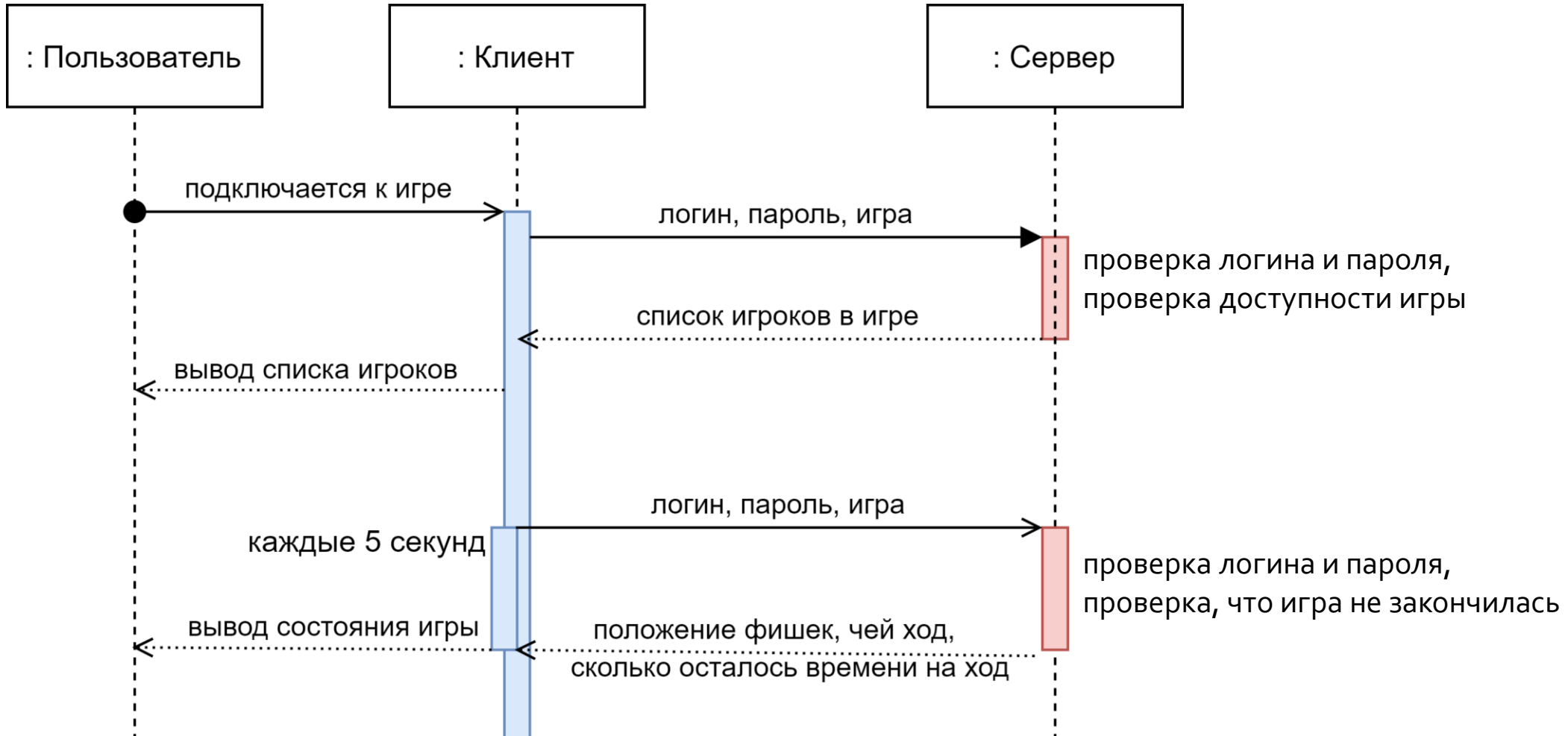
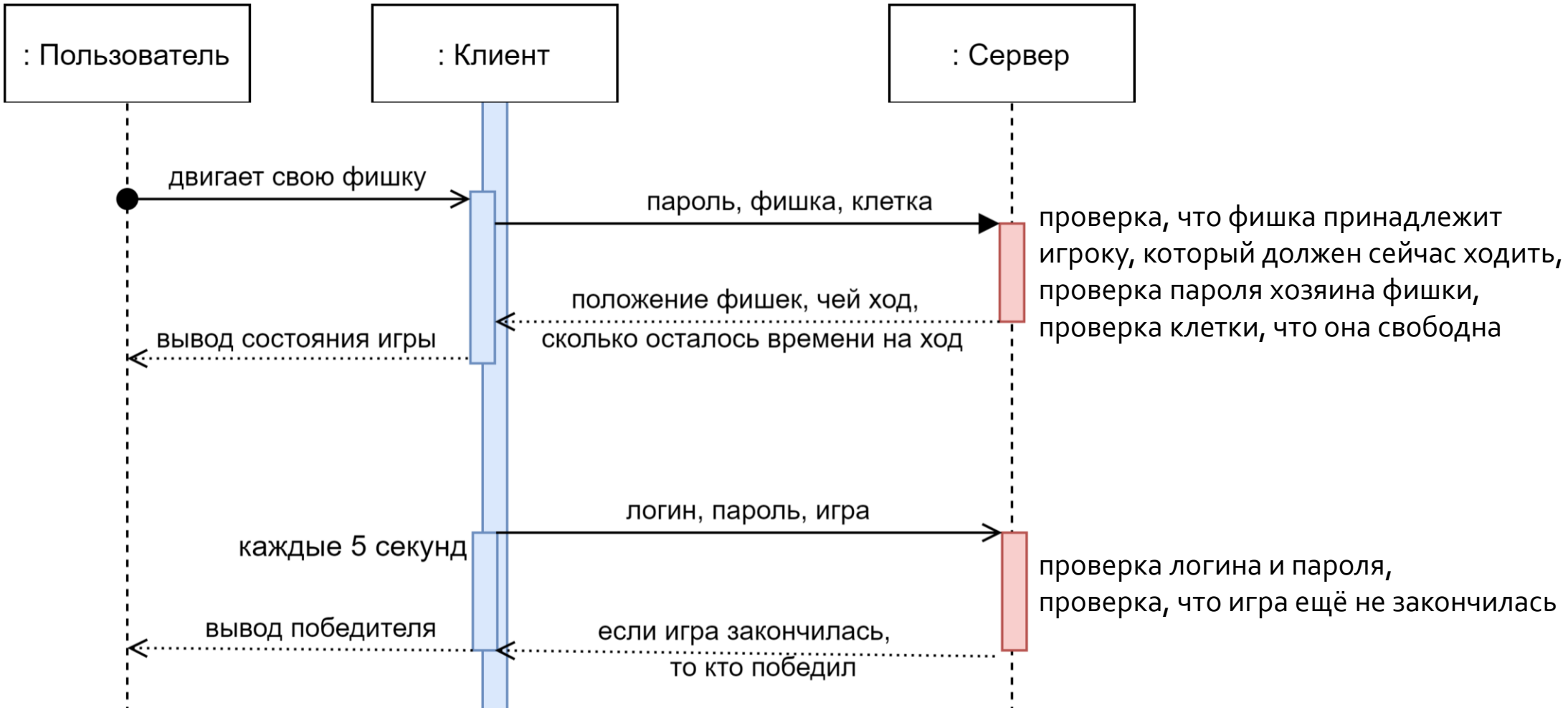


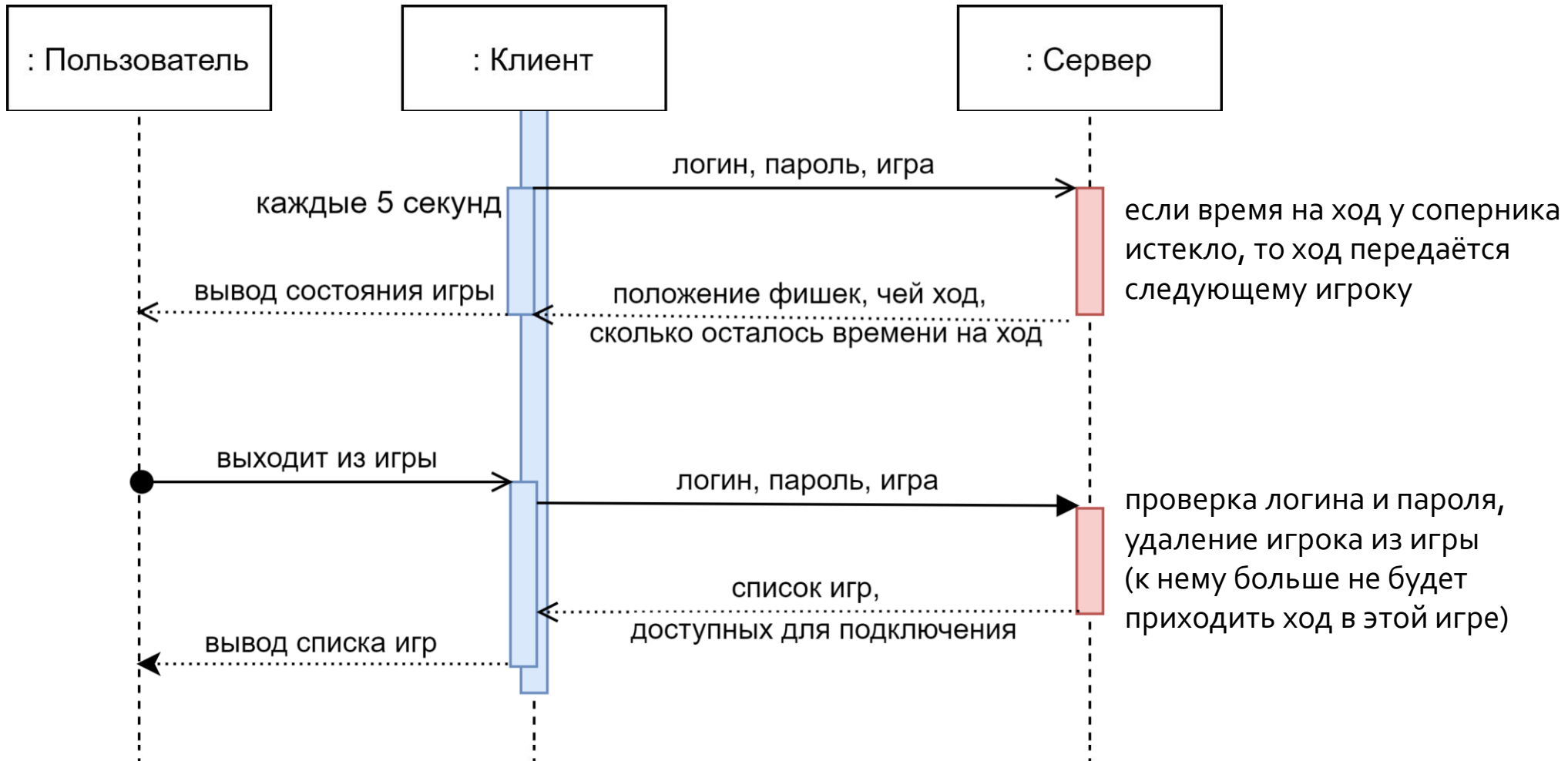
ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ



проверка, что фишка принадлежит игроку, который должен сейчас ходить,
проверка пароля хозяина фишки,
проверка клетки, что она свободна

проверка логина и пароля,
проверка, что игра ещё не закончилась

ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ



ПРИМЕР: КРЕСТИКИ-НОЛИКИ

- Пользователь видит список других пользователей и может приглашать их сыграть.
- Если приглашение принимается (или если два пользователя пригласили друг друга), то начинается игра.
- Кто играет крестиками, а кто ноликами, определяется случайно в момент начала игры (на сервере).
- Первый ход делают крестики.
- Если игрок не успел сделать свой ход за 120 секунд, то ход переходит к сопернику.

ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

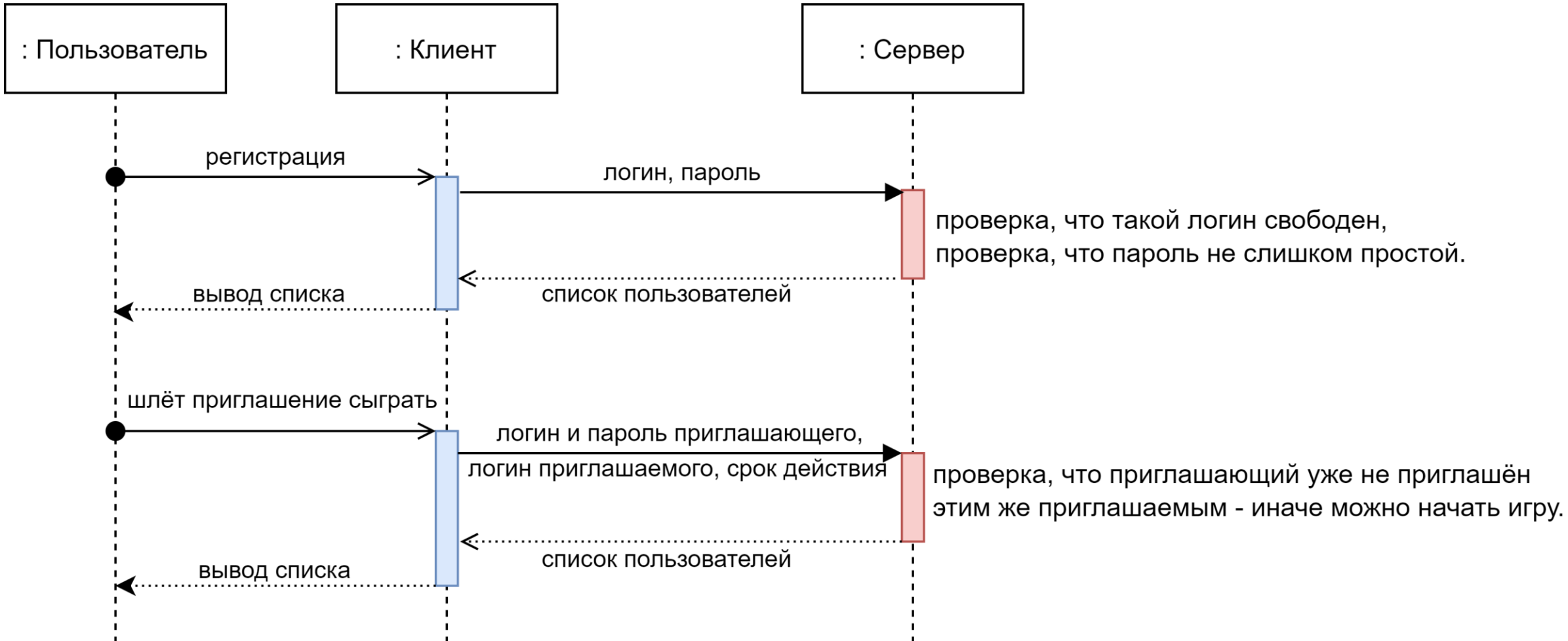
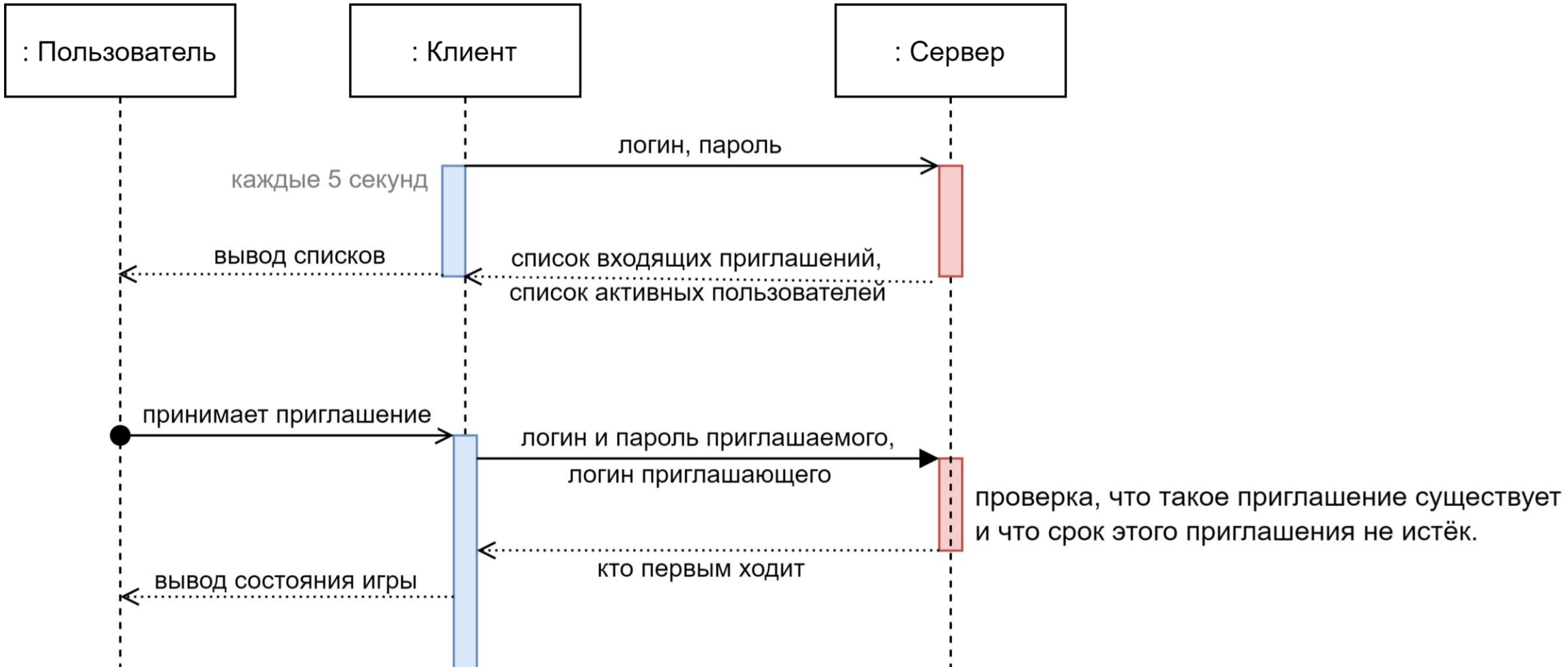
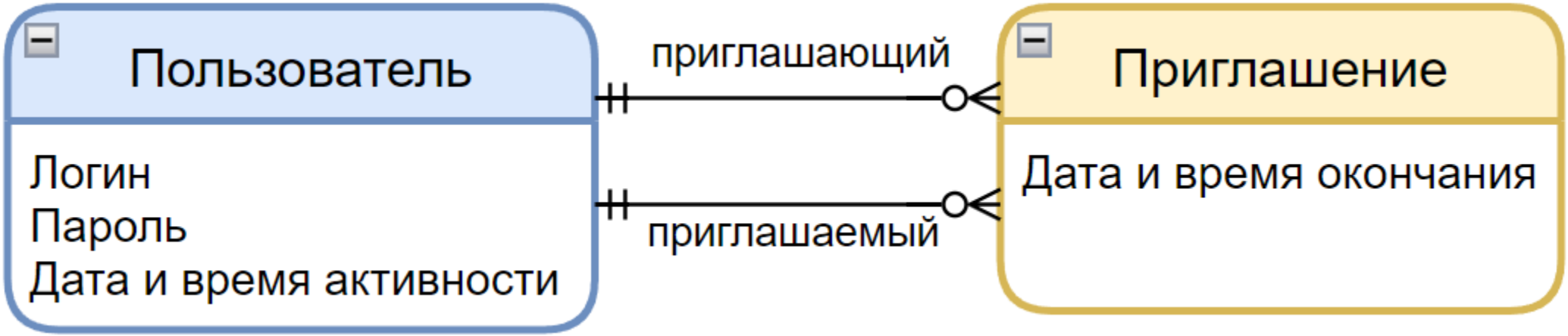


ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ



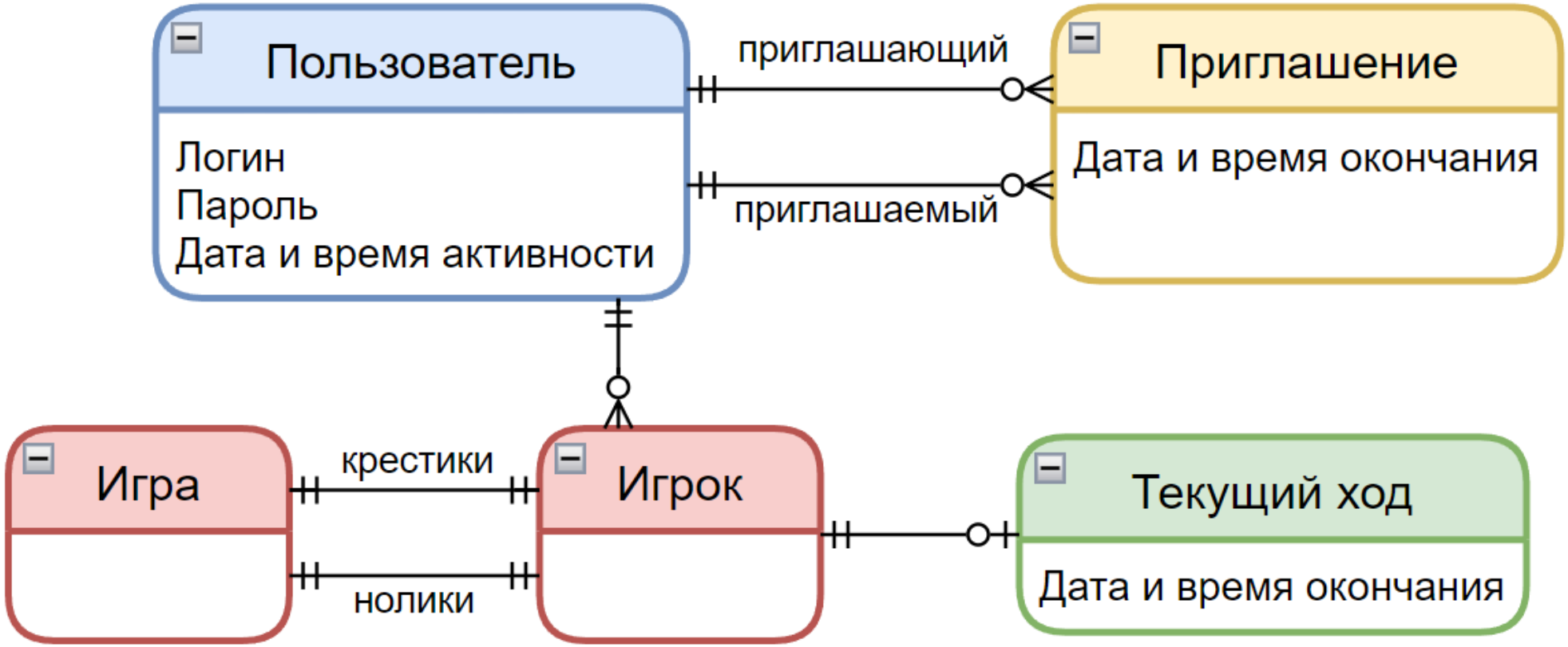
ПРИМЕР ER-ДИАГРАММЫ

- Чтобы выводить в списке пользователей для приглашения не всех зарегистрированных пользователей, а только тех, кто находится «онлайн» (то есть от чьих клиентов приходили запросы в последние несколько минут, например), нужно для каждого пользователя хранить, когда он был в последний раз активен.
- Чтобы хранить срок окончания действия приглашения, нужно изобразить приглашение не как связь, а как отдельную сущность.
- Пользователь может одновременно приглашать много пользователей (и в дальнейшем он сможет играть параллельно несколько игр).



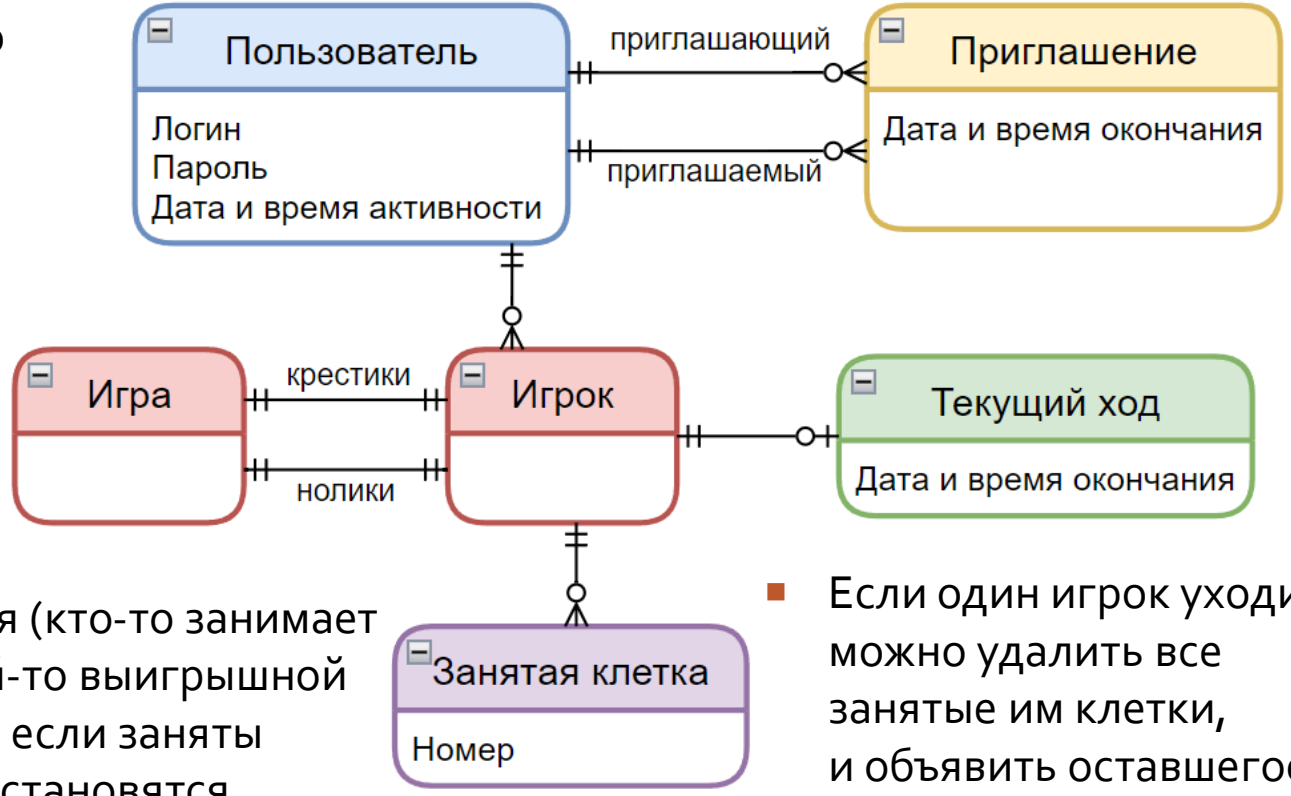
ПРИМЕР ER-ДИАГРАММЫ

- Если приглашение принято, то создаётся игра, и для обоих пользователей создаётся по игроку в этой игре: один играет крестиками, а другой ноликами. Изначально текущий ход привязан к играющему крестиками.



ПРИМЕР ER-ДИАГРАММЫ

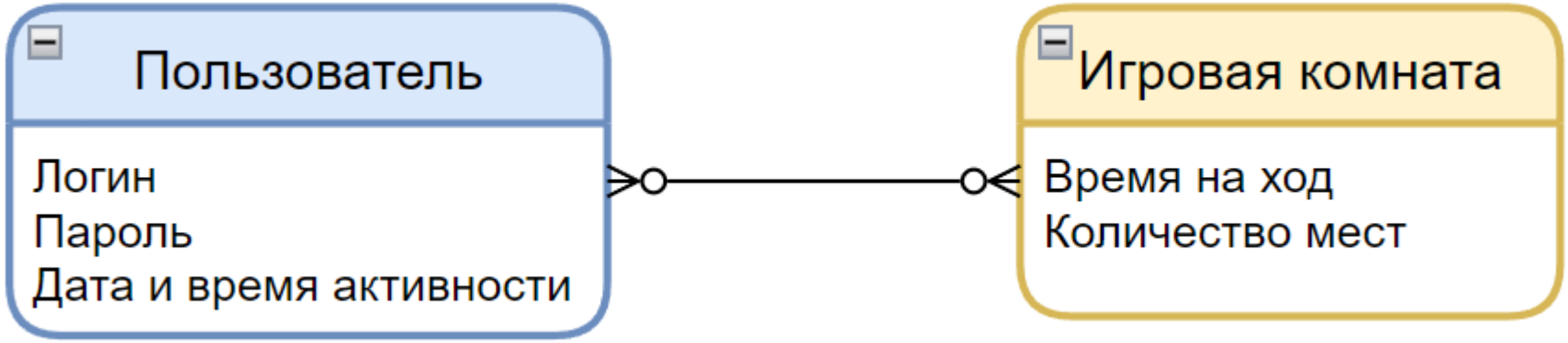
- Нам не нужно хранить информацию обо всех ходах в игре – достаточно знать, кто должен ходить сейчас, и какие клетки на поле уже заняты какими игроками.
- Клетки поля в крестиках-ноликах можно просто пронумеровать от 1 до 9.
- Связав клетку с игроком, её уже не нужно связывать с игрой, так как по игроку будет понятно, к какой игре она относится.
- Хранить свободные клетки не нужно, так как понятно, что свободные клетки – это клетки с номерами от 1 до 9, которые не встречаются среди клеток, связанных с игроками этой игры.
- Если после очередного хода игра заканчивается (кто-то занимает клетки с номерами 1-2-3 или 3-5-7 или ещё какой-то выигрышной комбинацией, либо игра заканчивается вничью если заняты 9 клеток и никто не победил), тогда оба игрока становятся не связаны с текущим ходом, и никто больше не сможет ходить.



- Если один игрок уходит, можно удалить все занятые им клетки, и объявить оставшегося игрока победителем.

ПРИМЕР ER-ДИАГРАММЫ

- Если количество игроков в игре может быть больше двух, то вместо приглашений можно предусмотреть создание игровых комнат пользователями. Пользователь может создать игровую комнату, устанавливая для неё параметры игры (длительность хода в секундах, количество мест и т.п.), а другие пользователи могут подключаться к комнатам, в которых есть свободные места.



- Если количество пользователей в комнате станет равно количеству мест, будет создана игра, и для каждого пользователя будут созданы игроки в этой игре.

ДОМАШНЕЕ ЗАДАНИЕ

Составить ER-модель игрового процесса для выбранной настольной игры.

ER-модель должна содержать всю необходимую информацию из предметной области игры, которая потребуется для клиент-серверной реализации этой игры с контролем соблюдения правил на стороне СУБД.

Пользователь должен иметь возможность играть одновременно в несколько игр (в каждой игре он будет представлять отдельного игрока).

Предусмотреть возможность ограничения времени на ход.