Экзаменационный проект   
по дисциплине  
Проектирование баз данных

студента

**Р”СѓР»Р°РµРІР° Р”РјРёС‚СЂРёСЏ РђР»РµРєСЃР°РЅРґСЂРѕРІРёС‡Р°**

по теме

**РњРµРґРёС†РёРЅР°**

# Описание проекта

База данных онлайн ресурса по соревнованию в игре «шахматы».

На ресурсе существуют зарегистрированные пользователи со своей личной информацией и их рейтингом в различных типах игр.

Также на ресурсе присутствуют игры между парами пользователей. Сами игры бывают разных типов (изначальное время в минутах, которое отводится каждому пользователю и добавочного время в секундах, которое прибавляется ко времени пользователя после его хода) и описываются как последовательность ходов пользователей, а результат игры — ее тип (соперник побежден, произошла ничья или противник сдался), а также дополнительная информация о игре.

Игры бывают, как свободные, так и в составе некоторого турнира. Турнир создается по мотивам определенного типа игры и для определенного дивизиона. В таком случае, в нем могут участвовать только пользователи с рейтингом по данному типу игры не ниже, требуемого для получения данного дивизиона. Сам же турнир — это множество игроков, играющих между собой множество игр. Промежуточный результат хранится в таблице либеров в виде счета пользователей, участвующих в турнире.

Помимо всего, пользователям предоставлена возможность оставлять комментарии к той или иной партии, отправлять личные сообщения и писать сообщения на форуме. Каждое сообщение может являться ответом на другое сообщение в той же игре, переписке, переписке на форуме соответственно.

В ходе проектирования базы данных было принято решение представления временны́х данных в виде полувременны́х так как во всех отношениях, время ставится один раз в момент создания кортежа и больше никогда не изменяется.

Также было выявлено, что при проектировании потребуется использовать наследование, так как прямые сообщения, комментарии у игры и сообщения на форуме имеют одинаковую общую структуру сообщения, от которого они наследуются. В результате было принято решение использовать “таблицы конкретных классов”, так как, во-первых, нам обычно интересно загружать объекты одного типа, а загрузка разных объектов — это вырожденный случай, а значит таких запросов почти не будет. Во-вторых, данный способ избавляет нас от работы с nullable атрибутами, что уменьшит количество ошибок в будущих запросах.

Стоит отметить, что в ходе проектирования было принято решение избавляться от nullable столбцов в итоговой базе данных, дабы избежать проблем с запросами в будущем и сделать базу данных более человеко-читаемой, в том плане, что значение null может трактоваться по-разному, а присутствие или отсутствие записи в таблице — однозначно.

# Построение отношений

В результате предварительного проектирования были выделены следующие отношения:

* User — зарегистрированный пользователь с его личной информацией.
* GameType — разрешенные на ресурсе типы игр (см. описание проекта). Создаем данное отношение отдельно, чтобы хранить в одном месте все разрешенные типы игр.
* Rating — рейтинг пользователей по играм различного типа. Рейтинг не является частью отношения User так как игрок может не иметь рейтинга по какому-то типу игр (например, если он только что зарегистрировался и еще не сыграл ни одной игры), может иметь несколько рейтингов по разным типам игр, и в таком случае в теле отношения происходило бы ненужное дублирование информации о самом пользователе.
* Division — дивизионы с границей минимального требуемого рейтинга для их получения, представленные на ресурсе. Если у пользователя по некоторому типы игры рейтинг будет не меньше границы для дивизиона, считается, что данный пользователь входит в этот дивизион по данному типу игры.
* Games — информация о игре между парой пользователей. Содержит всю необходимую информацию о самой игре, и ее начале. Информацию об окончании игры не содержит.
* GameResults — отношение, содержащее информацию об окончании игры: время окончания и тип конца игры. Данное отношение не является частью отношения Games, чтобы в итоге избавиться от nullable атрибутов.
* GameSteps — в данном отношении хранятся ходы, сделанные в определенных играх. Данная информация хранится отдельно от отношения Games, так как один сделанный ход — это один кортеж в теле данного отношения, а при переносе этой информации в отношение Game, информация о самой игре, которая, должна быть уникальна и неизменна, начинает дублироваться, что может привести систему в несогласованное состояние.
* Tournaments — информация о турнирах. В данном отношение хранится информация о том, что это за турнир, когда начался и закончился. У каждого турнира есть тип игры — тип игр которые будут играться в данном турнире. А также тип дивизиона, который показывает, сколько минимум требуется пользователям рейтинга для участия в данном турнире.
* TournamentParticipants — отношение, предоставляющее информацию о том, какие пользователи принимали/принимают участие в турнире, а также их текущий рейтинг в данном турнире. Данное отношение не является частью отношения Tournament, так как при добавлении в отношении Tournaments, на каждого пользователя будет дублироваться копия информации о турнире, что в свою очередь номет привести систему в несогласованное состояние.
* TournamentGames — отношение, дающее понять, какие игры были начаты/сыграны в том или ином турнире. TournamentGames является отдельным отношением, так как если, например, сделать его частью отношения Games, то появится nullable атрибут (т.к. игры могут, как входить в турнир, так и нет, а TournamentId по игру хранить придется), но было принято решение избавляться от nullable атрибутов, а если сделать его частью отношения Tournaments, то из-за того, что в одном турнире может быть много игр, то данное действие приведет к дублированию данных о турнире на каждую игру, что в свою очередь может привести систему в несогласованное состояние.
* Forums — отношение, задающее форумы с различными темами для обсуждений.
* FurumMessages — отношение, задающее информацию о сообщениях, отправленных на форуме. Данное отношение является по сути своей классом, наследуемом от некого класса Message. Объяснение выбранного способа представления наследований расписано в описании проекта.
* ReplyFormMessages — отношение, хранящее информацию об ответах одних сообщений на другие в пределах одного форума. Данное отношение не является частью отношения ForumMessages, так как в таком случае атрибут, являющийся идентификатором ответного сообщения был бы nullable, но было принято решение избавляться от таких атрибутов.
* DirectMessages — отношение, задающее информацию о сообщениях, отправленных напрямую между парой пользователей (личные сообщения). Данное отношение является по сути своей классом, наследуемом от некого класса Message. Объяснение выбранного способа представления наследований расписано в описании проекта.
* ReplyDirectMessages — отношение, хранящее информацию об ответах одних сообщений на другие в пределах одной переписки. Данное отношение не является частью отношения DirectMessages, так как в таком случае атрибут, являющийся идентификатором ответного сообщения был бы nullable, но было принято решение избавляться от таких атрибутов.
* GameComments — отношение, задающее информацию о коментариях, оставленных под какой-то игрой. Данное отношение является по сути своей классом, наследуемом от некого класса Message. Объяснение выбранного способа представления наследований расписано в описании проекта.
* ReplyGameComments — отношение, хранящее информацию об ответах одних сообщений на другие в пределах комментарий под одной игрой. Данное отношение не является частью отношения GameComments, так как в таком случае атрибут, являющийся идентификатором ответного сообщения был бы nullable, но было принято решение избавляться от таких атрибутов.

## Отношение Users

Атрибуты:

* UserId — уникальный идентификатор пользователей.
* Login — уникальное имя пользователя, под которым он заходит в систему.
* PasswordSha — зашифрованный пароль пользователя, просоленный при помощи сгенерированной соли.

Функциональные зависимости:

* UserId → Login
* UserId → PasswordSha
* Login → UserId

Ключи:

* {UserId}
* {Login}

### Нормализация

1 НФ:

* В отношении нет повторяющихся групп.
* Все атрибуты атомарны.
* Как можно заметить из функциональных зависимостей, замыкамие атрибута {UserId} над множеством ФЗ, дает в результате множество всех атрибутов, так как для любого атрибута из отношения существует ФЗ из UserId в этот атрибут. Исходя из этого, UserId — надключ. Но так как UserId — множество из одного элемента, то оно уже минимально по включению, следовательно множество {UserId} — ключ. Также если рассмотреть замыкание атрибута Login, то по ФЗ Login → UserId, понимаем, что в нашем замыкание будет UserId, а значит, исходя из всего вышесказанного, и все остальные атрибуты. Следовательно {Login} — тоже (над)ключ из тех же соображений. Заметим, что ни один из оставшихся атрибутов не присутствует в левой части какой-либо ФЗ, а значит из них никак не получится вывести остальные атрибуты. Следовательно, они не являются частью какого-либо ключа. Как мы поняли, в нашем отношении есть ключ.
* Таким образом, отношение находится в 1 НФ.

2 НФ:

* Отношение находится в 1 НФ.
* В нашем отношении, атрибут PasswordSha — единственный неключевой атрибут. Но, так как {UserId} — ключ и есть ФЗ UserId → PasswordSha, и больше нигде в ФЗ PasswordSha не входит, то делаем вывод, что нет неключевого атрибута, который бы зависел от части ключа.
* Таким образом, отношение находится во 2 НФ.

3 НФ:

* Отношение находится во 2 НФ.
* Как мы уже знаем, неключевой атрибут у нас единственный (PasswordSha), и есть ФЗ, в которой PasswordSha непосредственно зависит от ключа UserId.
* Таким образом, отношение находится в 3 НФ.

НФБК:

* Как мы уже знаем, в нашем отношении ключами являются {UserId} и {Login}. Также можно заметить, что во всех приведенных нетривиальных ФЗ, слева стоит ключ, а значит и надключ.
* Таким образом, отношение находится в НФБК.

4 НФ:

* Покажем, что в нашем отношении нет нетривиальных МЗ, где слева стоял бы не надключ. От противного. Пусть такая МЗ есть. Значит слева не надключ, следовательно там не может быть ключевых атрибутов (так как по следствию из определения, любое надмножество ключ — его надключ). Значит слева могут стоять только неключевые атрибуты. Но у нас только один неключевой атрибут PasswordSha. Тогда для правой части остаются 2 оставшихся атрибута-ключа. Но как бы их не расставили в правой части, чтобы получилось нетривиальная МЗ, всегда будут получаться эквивалентные МЗ (по следствию из теоремы Фейгина). Рассмотрим данную МЗ: PasswordSha ↠ UserId | Login. Это многозначная зависимость не выполняется, так как при фиксированном PasswordSha, для разных Login не могут быть одинаковое множество UserId, так как UserId — ключ и уникален для каждого кортежа. А при разных значениях атрибута Login, мы, очевидно, получаем 2 разных кортежа. Противоречие. Следовательно в данном отношении не существует таких МЗ.
* Таким образом, отношение находится в 4 НФ.

5 НФ:

* Покажем, что в нашем отношении нет ЗС, у которых одно из составляющих было бы не надключем. От противного пусть такое есть. Тогда в нем есть некое X не являющееся надлюче. Как мы уже знаем, такое множество X единственное и является множеством {PasswordSha}. Рассмотрим оставляется множества из данной ЗС. Их естественное соединения (по ассоциативности и коммутативности) можно запиать, так множесто Y. Заметим, что множество Y должно содержать атрибуты UserId и Login, иначе в итоговом естественном соединении нельзя было бы их ни откуда получить, а значит такое естественно соединение не равнялось бы исходному отношению. Значит для разных кортежей множества Y различны и отличаются как минимум в UserId и Login. Пусть также наше отношение состоят только из двух кортежей, где, как мы уже знаем множества Y — различны, но также и различны множества X. Тогда, во-первых, если X ∩ Y = ∅, то естественное соединение — декартого произведение в данном случае и мы получим 4 различных кортежа, вместо двух. Во-вторых, если X ∩ Y ≠ ∅, то Y содержит UserId, Login и PasswordSha. Следовательно это тривиальная ЗС, что мы не рассматриваем. Следовательно мы рассмотрели все возможные ЗС, которые нам могли бы мешать. Противоречие. Таких ЗС нет в нашем отношении.
* Таким образом, отношение находится в 5 НФ.

## Отношение GameTypes

Атрибуты:

* InitialGameTypeTime — начальное время в минутах некого типа игры (например, если тип игры — это X:Y, то InitialTime соотносится с X).
* AdditionalGameTypeTime — добавочное время в секундах после каждого хода некого типа игры (например, если тип игры — это X:Y, то AdditionalTime соотносится с Y).
* GameTypeName — иназвание типа игры, соотносящееся с ее начальным и добавочным временем.

Функциональные зависимости:

* InitialGameTypeTime, AdditionalGameTypeTime → GameTypeName

Ключи:

* {InitialGameTypeTime, AdditionalGameTypeTime}

### Нормализация

1 НФ:

* В отношении нет повторяющихся групп.
* Все атрибуты атомарны.
* Заметим, что в нашем отношение только одна ФЗ, содержащая все атрибуты из отношения. Следовательно в левой части — ключ, так как от него зависит все остальное и нет других правил, ограничивающих это.
* Таким образом, отношение находится в 1 НФ.

2 НФ:

* Отношение находится в 1 НФ.
* Как мы знаем, все неключевые атрибуты в нашем отношении находятся в правой части нашей ФЗ. А так как в левой части — ключ, следовательно все неключевые атрибуты зависят от ключа в целом.
* Таким образом, отношение находится во 2 НФ.

3 НФ:

* Отношение находится во 2 НФ.
* Как мы уже знаем, неключевой атрибут у нас единственный, и есть ФЗ, в которой данный атрибут зависит от ключа. Значит все неключевые атрибуты зависят от ключей непосредственно.
* Таким образом, отношение находится в 3 НФ.

НФБК:

* Как уже было сказано ранее, во всех наших нетривиальных ФЗ слева стоит ключ, а значит и надключ. А неключевой атрибут у нас один и уже есть ФЗ, где он зависит от надключа.
* Таким образом, отношение находится в НФБК.

4 НФ:

* Покажем, что в нашем отношении не существует нетривиальных МЗ, где слева — не надключ. От противного. Пусть есть такая МЗ. Значит слева не надключ. Следовательно слева может быть любой из атрибутов множества, но только в количестве одной штуки, так как, во-первых, каждый из атрибутов по-отдельности не является ключем, а значит и надключем не является, а, во-вторых, если слева будет 2 атрибута или больше, то МЗ будет тривиальной. Зафиксируем 3 множества X, Y, Z. В каждое из них положим один из атрибутов из нашего отношения, причем в каждый, различные атрибуты. Заметм, что как бы мы не положили эти атрибуты по трем множествам, любая пара множеств, зависит друг от друга. Следовательно множество значений Y зависит от R \ X \ Y = Z. Противоречие. Следовательно не существует таких МЗ.
* Следовательно отношение находится в 4 НФ.

5 НФ:

* Покажем, что в нашем отношении нет нетривиальных ЗС, в которых содержался бы такой X, что X — не надключ. От противного. Путь существует такая ЗС. Тогда есть такой X, что не надключ. Как мы уже знаем, множество X мощности 1 и содержит любой из атрибутов данного отношения. Возьмем оставшуются часть ЗС и применить все естественные соединения, воспользуясь его коммутативностью и ассоциативностью. В итоге получим множество Y: X ⋈ Y = R. Для начала Очевидно, что Y должно содержать все атрибуты из R \ X, так как иначе в итоговом естественном соединение не были бы все атрибуты, а значит данное соединение не равнялось бы исходному множеству. Во-первых, рассмотрим случай, когда атрибут из X принадлежит Y. Y содержит все атрибуты из отношения. А это уже является тривиальной ЗС, следовательно не рассматриваем данный случай. Остается случай, когда X ∩ Y = ∅, причем X ∪ Y ≠ ∅. Тогда пусть наше отношение состоит из двух кортежей, таких, что значения X и Y между кортежей различны между собой. Значения атрибутов, соответствующих Y точно различны, так как Y содержит ключ, а его значение между кортежами не повторяется. X же зависит только от ключа, а так как ключи в наших кортежах различные, то значения X двух кортежей можно сделать различными. В таком случае X ⋈ Y = X × Y, причем в итоге получается 4 различных кортежа, вместо 2 изначальных. Противоречие. Следовательно не существует таких ЗС.
* Следовательно наше отношение находится в 5 НФ.

## Отношение Ratings

Атрибуты:

* UserId — уникальный идентификатор пользователя, у которого есть рейтинг.
* InitialGameTypeTime — начальное время некого типа игры (см. GameType для более подробного описания).
* AdditionalGameTypeTime — добавочное время после каждого хода некого типа игры (см. GameType для более подробного описания).
* Score — рейтинг игрока в некотором типе игры.

Функциональные зависимости:

* UserId, InitialGameTypeTime, AdditionalGameTypeTime → Score

Ключи:

* {UserId, InitialGameTypeTime, AdditionalGameTypeTime}

## Отношение Divisions

Атрибуты:

* DivisionId — уникальный идентификатор дивизиона.
* DivisionName — название дивизиона.
* DivisionScoreBound — минимальная граница рейтинга, требуемая пользователям для достижения данного дивизиона по некоторому типу игры.

Функциональные зависимости:

* DivisionId → DivisionName, DivisionScoreBound

Ключи:

* {DivisionId}

## Отношение Games

Атрибуты:

* GameId — уникальный идентификатор игры.
* WhitePlayerId — уникальный идентификатор пользователя, играющего белыми фигурами.
* BlackPlayerId — уникальный идентификатор пользователя, играющего черными фигурами.
* InitialGameTypeTime — начальное время некого типа игры (см. GameType для более подробного описания).
* AdditionalGameTypeTime — добавочное время после каждого хода некого типа игры (см. GameType для более подробного описания).
* StartGameTime — время начала игры.

Функциональные зависимости:

* GameId → WhitePlayerId
* GameId → BlackPlayerId
* GameId → InitialGameTypeTime
* GameId → AdditionalGameTypeTime
* GameId → StartGameTime

Ключи:

* {GameId}

## Отношение GameResults

Атрибуты:

* GameId — уникальный идентификатор закочившейся игры.
* EndGameTime — время окончания игры.
* EndGameType — тип окончания игры (победа одного из игроков, ничья, один из игроков сдался).

Функциональные зависимости:

* GameId → EndGameTime
* GameId → EndGameType

Ключи:

* {GameId}

## Отношение GameSteps

Атрибуты:

* GameId — уникальный идентификатор игры, где был сделан ход.
* StepId — уникальный номер хода по порядку.
* FromCell — клетка шахматного поля, из которой был сделан ход.
* ToCell — клетка шахматного поля, в которую был сделан ход.
* SetpTime — время, в которое был сделан ход.

Функциональные зависимости:

* GameId, StepNumber → FromCell
* GameId, StepNumber → ToCell
* GameId, StepNumber → StepTime

Ключи:

* {GameId, StepNumber}

## Отношение Tournaments

Атрибуты:

* TournamentId — уникальный идентификатор турнира.
* TournamentName — название турнира, заданное его организаторами.
* TournamentDescription — описание турнира, заданное его организаторами
* InitialGameTypeTime — начальное время некого типа игры (см. GameType для более подробного описания) — тип игр, которые будут играться в данном турнире.
* AdditionalGameTypeTime — добавочное время после каждого хода некого типа игры (см. GameType для более подробного описания) — тип игр, которые будут играться в данном турнире.
* DivisionId — уникальный идентификатор дивизиона, показывающий, пользователи с каким минимальным рейтингом по данному типу игры могут принять участие в турнире.
* StartTournamentTime — время начала турнира. Данный атрибут является обязательным полем, обычно данное время определено в будущем для запланированных турниров, но можно также указать текущее время.
* EndTournamentTime — время окончания турнира. Данный атрибут является обязательным, задается в процессе создания турнира и обозначает время, когда турнир должен закончется. Значение данного атрибута обязано быть больше, чем значение атрибута StartTournamentTime + minTournamentDurationConst, где minTournamentDurationConst — некая константа, обозначающая, сколько минимум по времени должен идти любой турнир.

Функциональные зависимости:

* TournamentId → TournamentName
* TournamentId → TournamentDescription
* TournamentId → InitialGameTypeTime
* TournamentId → AdditionalGameTypeTime
* TournamentId → DivisionId
* TournamentId → StartTournamentTime
* TournamentId → EndTournamentTime

Ключи:

* {TournamentId}

### Нормализация

1 НФ:

* В отношении нет повторяющихся групп.
* Все атрибуты атомарны.
* У отношения есть ключ {TournamentId} т.к. от него есть ФЗ, по которой все остальные атрибуты зависят от него. Значит замыкание множества {TournamentId} будет равно {TournamentId, TournamentName, TournamentDescription, InitialGameTypeTime, AdditionalGameTypeTime, DivisionId, StartTournamentTime, EndTournamentTime}. Что означает, что {TournamentId} — надключ. Но так как этот надключ и множество из одного элемента, то данное множество минимально по включению. Следовательно, это ключ. Заметим, что других ключей нет т.к. в левой части всех ФЗ можно встретить только TournamentId. Следовательно по любому другому атрибуты нельзя не зная TournamentId получить в замыкание какой-то другой атрибут. А для всех множеств, являющихся надмножеством {TournamentId}, их замыкание — надключ, но, как мы уже видели, эти множества ну будут минимальны по включению.
* Таким образом, отношение находится в 1 НФ.

2 НФ:

* Отношение находится в 1 НФ.
* Каждый из атрибутов, кроме TournamentId — неключевой. Также для каждого такого атрибута есть ФЗ из TournamentId в данный атрибут. А так как {TournamentId} — ключ, причем единственный, то каждый из неключевых атрибутов зависит от ключа в целом.
* Таким образом, отношение находится во 2 НФ.

3 НФ:

* Отношение находится во 2 НФ.
* Как мы знаем из доказательства выше, {TournamentId} — ключ, а также каждый из оставшихся атрибутов — неключевые и существует ФЗ, по которой они непосредственно зависят от TournamentId.
* Таким образом, отношение находится в 3 НФ.

НФБК:

* Как мы уже знаем, во всех нетривиальных ФЗ, представленных выше, слева стоит всегда TournamentId. Также мы знаем, что {TournamentId} — ключ, а значит и надключ. Следовательно, в каждой нетривиальной ФЗ, которые представлены выше, слева стоит надключ. Также мы знаем, что ФЗ можно получать при помощи правил вывода. Заметим, что левую часть расщепить нельзя, так в каждой левой части уже стоит один атрибут, а при добавлении элементов в левую (и правую) часть, левая часть будет являться надключем, так как любое надмножество ключа — надключ по определению.
* Таким образом, отношение находится в НФБК.

4 НФ:

* Как мы знаем, все ФЗ являются МЗ. Но так как каждая левая часть этих МЗ — ключ, а следовательно и надключ, то данные МЗ могу присутствовать в 4 НФ.
* Рассмотрим остальные нетривиальные МЗ. Покажем, что больше никаких нетривиальных не существует. Возьмем 2 корежа из нашего отношения и будем постепенно присваивать значения их атрибутам. Заметим, что так как TournamentId — ключ, то данный атрибут разный у этих кортежей. Поставим в качестве значения TournamentId 2 разных значения. Затем зафиксируем произвольное подмножество других атрибутов и назовем его X. Заметим, что так как атрибуты X зависят только от TournamentId, а между кортежами они разные, то значения этих не зависит от других кортежей, а значит могут быть любыми. Пусть в каждом из двух кортежей атрибуты из множества X принимают одинаковые значения. Теперь зафиксируем любое подмножество из оставшихся атрибутов и назовем его Y. Заметим, что если в Y входит TournamentId, то множества Y уже разные. Если же данный атрибут не входит в Y, то по аналогии с вышеописанными утверждениями можно сделать, чтобы множества Y принимали разные значения. Сделаем это. Теперь возьмем оставшиеся атрибуты, сформируем из них множество Z, и по аналогии зафиксируем в этих множествах разные значения. Получается, что при фиксированном X для разных Z множество Y разные, что противоречит определению множественной зависимости. Таким образом в данном отношении больше нет нетривиальных МЗ.
* Исходя из всего вышесказанного, в данном отношении для всех нетривиальных МЗ, слева — надключ.
* Следовательно отношение находится в 4 НФ.

5 НФ:

* Покажем, что в нашем отношении нет нетривиальных ЗС, в которых могли бы быть не надключами. От противного. Пусть есть такая ЗС. Тогда в нее входит некое множество атрибутов X, такое, что X — не надключ. По определению ЗС, мы должны при помощи естественного соединения всех множеств, входящих в ЗС, получить исходное отношение. Таким образом, так как ЗС нетривиальная, существует некое Y, что X ⋈ Y = R (по коммутативности и ассоциативности естественного соединения). Теперь рассмотрим 2 кортежа из отношения. И разберем 2 случая. Для начала пусть X ∩ Y = ∅. Следовательно X ∪ Y = R, иначе бы существовал Z ≠ ∅: R = X ∪ Y ∪ Z. Следовательно X ⋈ Y не мог бы включать никак атрибуты Z в себе, а значит X ⋈ Y не равнялось бы R. Тогда пусть наше отношение состоит только из этих двух коржей и попарно их множества X и Y — различны (для Y это очевидно, так как в нем содержатся ключи, а они различны, а для X, как было описано в пункте 4 НФ, неключевые атрибуты зависят только от ключа, а значит никак не влияют на значения друг друга, что означает, для конкретного значения ключа можно выбрать любое значение неключевых атрибутов). Тогда X ⋈ Y = X × Y, следовательно появится 4 различных кортежа, вместо двух изначальных. Теперь разберем случай, когда X ∩ Y ≠ ∅. Более того, X ∪ Y = R, так как иначе при естественном соединении, мы бы никак не смогли получить атрибуты R \ X \ Y в ответа, что означало бы неравенство самому отношению R. При этом X ⊄ Y и Y ⊄ X, иначе одному из них пришло бы быть всем множеством атрибутов из R, а мы договорились, что ЗС нетривиальная. Тогда введем множества Z = X ∩ Y, Ÿ = Y \ Z, Ẍ = X \ Z. Пусть опять же выбранные нами два кортежа — единственные в теле отношения, причем значения на атрибутах Z у них равны, а на атрибутах из множеств Ÿ и Ẍ — различные (исходя из вышесказанного, это возможно). Тогда X ⋈ Y будет содержать 4 различных кортежа, вместо двух исходных. Притиворечие. Следовательно, все всех ЗС, любое множество, входящее в его — надключ.
* В итоге мы показали, что не найдется такой ЗС, в которой какой-то Xi был бы не надключем.
* Следовательно отношение находится в 5 НФ.

## Отношение TournamentParticipants

Атрибуты:

* TournamentId — уникальный идентификатор турнира, на который записываются пользователи.
* UserId — уникальный идентификатор пользователя, который записывается на участие в турнире.
* BoardScore — значение рейтинга пользователя в турнирной таблице. Поле является обязательным и по-умолчанию инициализируется нулем.

Функциональные зависимости:

* TournamentId, UserId → BoardScore

Ключи:

* {TournamentId, UserId}

## Отношение TournamentGames

Атрибуты:

* TournamentId — уникальный идентификатор турнира, в котором проходит игра.
* GameId — уникальный идентификатор игры, которая проходит в турнире.

Функциональные зависимости:

* GameId → TournamentId

Ключи:

* {GameId}

## Отношение Forums

Атрибуты:

* ForumId — уникальный идентификатор форума.
* Topic — название форума.

Функциональные зависимости:

* ForumId → Topic

Ключи:

* {ForumId}

## Отношение ForumMessages

Атрибуты:

* MessageId — уникальный идентификатор сообщения. Данный атрибут является уникальный среди всех сообщений разного типа, описанных в разных отношениях.
* ForumMessageId — уникальный идентификатор сообщения на форуме.
* ForumId — уникальный идентификатор форума, где было отправлено сообщение.
* SenderId — уникальный идентификатор пользователя-отправителя.
* MessageText — текст сообщения.
* SendTime — время отправки сообщения.

Функциональные зависимости:

* MessageId → ForumMessageId
* ForumMessageId → MessageId
* ForumMessageId → ForumId
* ForumMessageId → SenderId
* ForumMessageId → MessageText
* ForumMessageId → SendTime

Ключи:

* {MessageId}
* {ForumMessageId}

## Отношение ReplyForumMessages

Атрибуты:

* FromMessageId — уникальный идентификатор сообщения, которое отправило ответ.
* ToMessageId — уникальный идентификатор сообщения, которому был отправлен ответ.

Функциональные зависимости:

* FromMessageId → ToMessageId

Ключи:

* {FromMessageId}

## Отношение DirectMessages

Атрибуты:

* MessageId — уникальный идентификатор сообщения. Данный атрибут является уникальный среди всех сообщений разного типа, описанных в разных отношениях.
* DirectMessageId — уникальный идентификатор личного сообщения.
* ReceiverId — уникальный идентификатор пользователя-получателя.
* SenderId — уникальный идентификатор пользователя-отправителя.
* MessageText — текст сообщения.
* SendTime — время отправки сообщения.

Функциональные зависимости:

* MessageId → DirectMessageId
* DirectMessageId → MessageId
* DirectMessageId → ReceiverId
* DirectMessageId → SenderId
* DirectMessageId → MessageText
* DirectMessageId → SendTime

Ключи:

* {MessageId}
* {DirectMessageId}

## Отношение ReplyDirectMessages

Атрибуты:

* FromMessageId — уникальный идентификатор сообщения, которое отправило ответ.
* ToMessageId — уникальный идентификатор сообщения, которому был отправлен ответ.

Функциональные зависимости:

* FromMessageId → ToMessageId

Ключи:

* {FromMessageId}

## Отношение GameComments

Атрибуты:

* MessageId — уникальный идентификатор сообщения. Данный атрибут является уникальный среди всех сообщений разного типа, описанных в разных отношениях.
* GameCommentId — уникальный идентификатор комментария под игрой.
* GameId — уникальный идентификатор игры, под которой был оставлен комментарий.
* SenderId — уникальный идентификатор пользователя-отправителя.
* MessageText — текст сообщения.
* SendTime — время отправки сообщения.

Функциональные зависимости:

* MessageId → GameCommentId
* GameCommentId → GameId
* GameCommentId → ReceiverId
* GameCommentId → SenderId
* GameCommentId → MessageText
* GameCommentId → SendTime

Ключи:

* {MessageId}
* {GameCommentId}

## Отношение ReplyGameComments

Атрибуты:

* FromMessageId — уникальный идентификатор сообщения, которое отправило ответ.
* ToMessageId — уникальный идентификатор сообщения, которому был отправлен ответ.

Функциональные зависимости:

* FromMessageId → ToMessageId

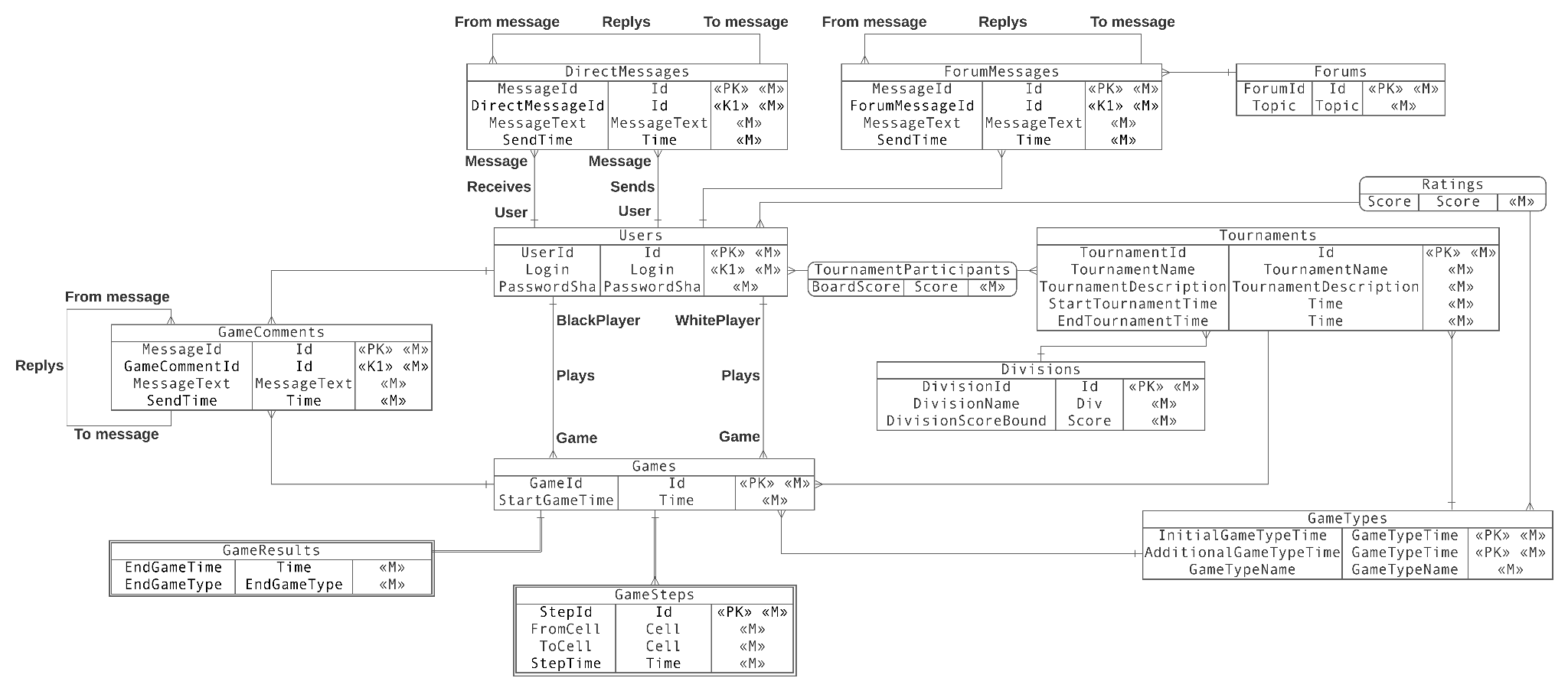
Ключи:

* {FromMessageId}

# Модель сущность-связь

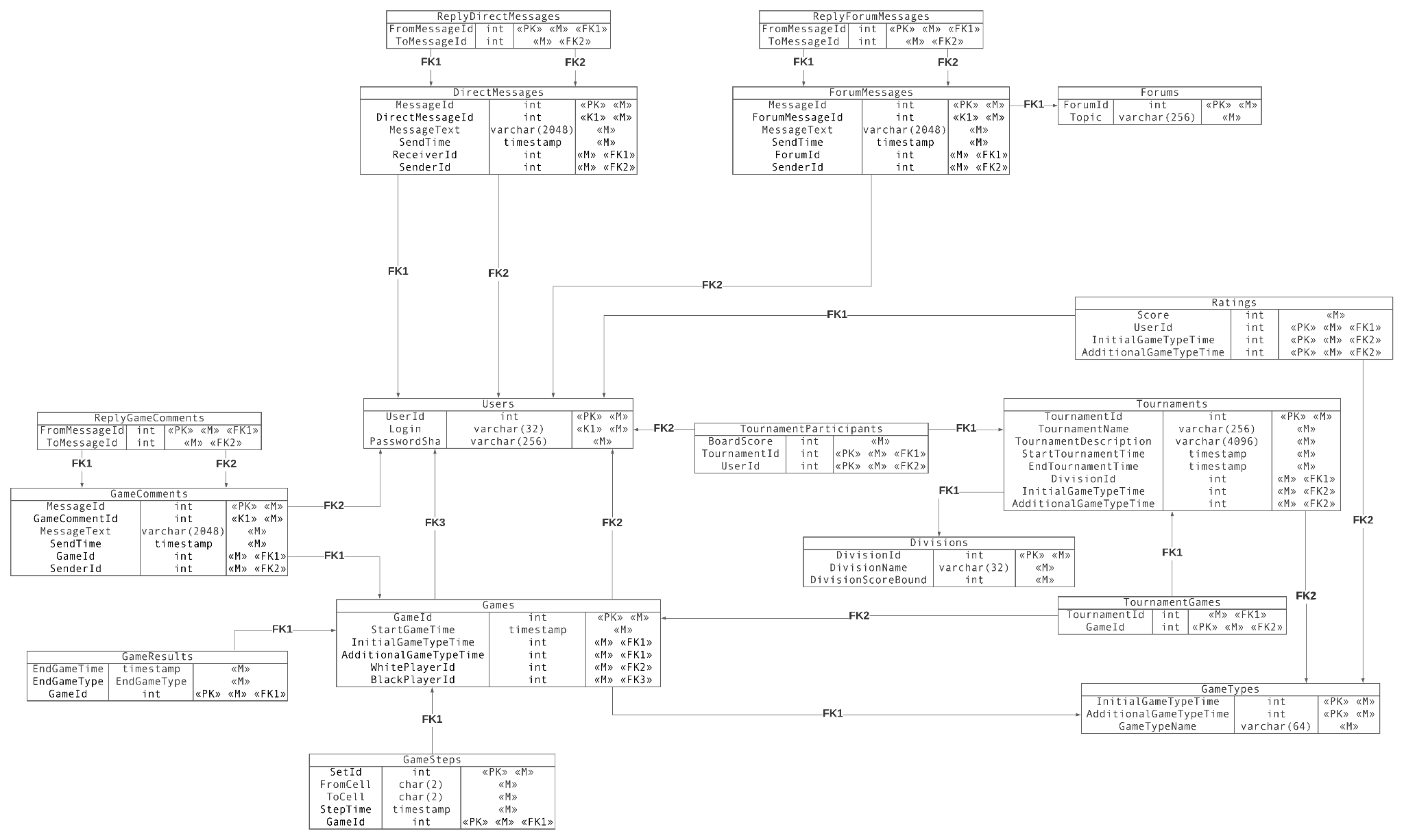
Для общности картины, скажем, что ограничения по Чену.

[Ссылка для просмотра картинки на стороннем ресурсе.](https://ibb.co/vdm6tm8)



# Физическая модель

[Ссылка для просмотра картинки на стороннем ресурсе.](https://ibb.co/Vpb79Kg)



При построении физической модели использовалось следующее отображение доменов в типы:

|  |  |
| --- | --- |
| Домен | Тип |
| Id | int |
| Topic | varchar(256) |
| MessageText | varchar(2048) |
| Login | varchar(32) |
| PasswordSha | varchar(256) |
| Score | int |
| TournamentName | varchar(256) |
| TournamentDescription | varchar(4096) |
| Time | timestamp |
| GameTypeTime | int |
| GameTypeName | varchar(64) |
| EndGameType | enum('WhiteWins', 'BlackWins', 'Draw', 'WhiteSurrender', 'BlackSurrender')  as EndGameType |
| Cell | char(2) |
| MessageType | enum('DirectMessage', 'GameComment', 'ForumMessage')  as MessageType |
| Div | varchar(32) |

# Определения таблиц

Для реализации проекта использовалась СУБД PostgreSQL 12.9. Определения таблиц и их индексов приведено в файле ddl.sql.

# Тестовые данные

Скрипт для добавления тестовых данных приведен в файле data.sql.

# Запросы на получение данных

В рамках проекта были реализованы следующие запросы:

* GetUserByLoginAndPasswordSha — запрос для авторизации пользователя. Человек при авторизации вводит :Login и :Password. Запрос проверяет, есть ли в базе данных такой пользователь, и возвращает его уникальный идентификатор в случае успеха.
* GetUserByLogin — так как логин пользователя обычно будет использоваться на ресурсе для отображения различных игроков, а запросы в базу данных базируются на уникальном идентификаторе, то данная функция позволяет получить его по :Login. Заметим, что другие запросы базируются именно на идентификаторе пользователя, а не на его уникальном логине, так как суррогатные ключи (:UserId) не меняются со временем, а естественные (:Login) имеют свойство меняться, что влечет за собой изменение большого количества данных.
* GetUserRatings — возвращает рейтинг пользователя :UserId, базируясь на человеко-читаемом названии типа игры. В личном кабинете пользователя с уникальным идентификатором :UserId предусмотрено место место для просмотра его личного рейтинга. Дабы не нагружать интерфейс пользователя типами игр вида X:Y, пользователю показывается его рейтинг в виде GameTypeName *—* Score. Для рейтингов различного типа и одного наименования пользователя берется среднее и возвращается в результате.
* GetUserGameTypeNameRatings — возвращает по :UserId и :GameTypeName более подробную статистику пользователя по типу игр. Так как предыдущий запрос сделан для удобства визуализации в личном кабинете пользователя, но несмотря на это у пользователя по одному названию типа игры может быть несколько типов игр с разным рейтингом.
* GetLeaderboard — возвращает таблицу лидеров по всем пользователям с определенным названием типа игры :GameTypeName.
* GetUserTookPartGames — возвращает по :UserId пользователя все его игры в которых он принимал/принимает участие. В личном кабинете пользователь может захотеть посмотреть все игры, которые им были сыграны или еще играются. Поэтому по данному запросу возвращается вся информация, которая известна о игре, чтобы лишний раз не ходить в базу данных, и сразу, например, отобразить на странице всю статистику.
* GetUserCurrentlyPlayingGames — возвращает игры в процессе на текущий момент времени у пользователя :UserId. Если пользователь случайно вышел из игры или играет параллельно несколько игр, тогда для быстрого доступа к играм в процессе на главной странице должны быть ссылки на данные игры. В данной ситуации нам не важна вся информация об игре, так как мы пока не хотим как-то ее анализировать, а просто хотим быстро переподключиться к ней. Также, стоит отметить, что в обычном мире количество активных игр у пользователя на конкретный момент времени исчисляется единицами, а значит, если нам понадобится вдруг информация об играх, сходить в базу не составит труда.
* GetUserExceptedAdditionalRatingScore — функция, возвращающая по игре GameId, игроку UserId и его оппоненту OpponentId, значение ожидаемого количества очков (не рейтинга), которое получит игрок UserId в партии с OpponentId. Данное значение является частью формулы, для вычисления “рейтинга Эло”. Данный запрос был обернут в функцию, так как данная формула также будет использоваться в запросах на изменение при подсчете рейтинга. И дабы избежать дублирования большого куска когда, была создана функция.
* GetUserExceptedAdditionalRating — функция, которая по результату игры ResultScore (1 для выигрышной игры, 0.5 для ничьей и 0 для проигрышной), пользователю UserId, его оппоненту OpponentId и игре GameId, возвращает текущий рейтинг пользователя и ожидаемое значение рейтинга, которое будет добавлен к текущему, если игра завершится с исходом ResultScore для данного игрока. Само значение считается через формула “рейтинга Эло”. Данный запрос обернут в функцию, так как данная логика также понадобится в запросах на изменение данных, и дабы избежать дублирование большого куска кадо, была создана функция.
* GetUserExpectedRating — возвращает по игре :GameId, ее игроку :UserId и его оппоненту :OpponentId, текущий рейтинг пользователя по играм того же типа, что и данная игра :GameId, и ожидаемые значения изменения рейтинга при выигрыше, ничьей и проигрыше.
* GetUserWinningGames — возвращается выигранные игры пользователя :UserId. Все в той же статистике пользователь может захотеть посмотреть выигранные игры, проанализировать их. По аналогии с GetUserTookPartGames, мы возвращаем всю информацию об игре, чтобы не ходить лишний раз в базу.
* GetUserSpecialTypeGames — возвращает игры пользователя :UserId с определенным типом. По аналогии с предыдущем, пользователь в личном кабинете может отфильтровать игры по их типу.
* GetUserSpecialTypeNameGames — возвращает игры пользователя :UserId с определенным названием типа игры. По аналогии с предыдущем, пользователь в личном кабинете может отфильтровать игры по их названию.
* GetGameSteps — возвращает сделанные игроками ходы в игре :GameId. Может понадобится, например, во время игры для мониторинга сделанных ходов или после партии для ее анализа.
* GetPlayerSteps — возвращает ходы сделанные пользователем :UserId в игре :GameId.
* GetTournamentsInfo — возвращает подробную информацию о всех турнирах.
* GetPossibleToJoinTournaments — возвращает турниры, в которых может принять участие :UserId. Так как для участия в турнире требуется определенный рейтинг в определенном типе игр, то для простоты можно будет отфильтровать все турниры на те, в которые можно участвовать на данный момент. Так это просто является фильтром для предыдущего запроса, то, очевидно мы также хотим получить всю имеющуюся информацию о турнире.
* GetPlayingTournaments — возвращает турниры, в которых на данный момент участвует пользователь :UserId. Так как турниры могут идти достаточно долго (например, несколько дней), чтобы пользователь не забыл о этих турнирах на следующий день, на главной странице будут отображаться список активных для пользователя турниров. Очевидно, что в данном случае мы уже знаем, что это за турнир и хотим видеть по-минимуму информации (Например просто ссылку с названием турнира).
* GetTournamentLeaderBoard — возвращает турнирную таблицу турнира :TournamentId.
* GetTournamentWinners — возвращает победителей турнира :TournamentId.
* GetForumMessages — возвращает переписку, принадлежащую некому форуму. Запрос возвращает сразу много атрибутов, чтобы один раз сходить в базу и отобразить по получившимся данным историю чата со всеми ответами и ссылками на пользователей.
* GetUserDirects — возвращает переписки с пользователями в личном кабинете пользователя. Более новые сообщения считаются более приоритетными и появляются в начале списка.
* GetUserDirectMessagesWithCompanion — возвращает переписку пользователя :UserId с :CompanionId. По аналогии с GetForumMessages, возвращаем сразу всю информацию о переписке, чтобы корректно без дополнительных походов в базу отобразить, кто, кому, когда писал, на какие сообщения отвечал, и т. д.
* GetGameComment — возвращает переписку, оставленную под игрой. По аналогии с get\_forum\_messages, возвращаем сразу много атрибутов, чтобы корректно отобразить всю переписку и не ходить лишний раз в базу данных.

Для реализации запросов были созданы вспомогательные представления:

* Messages — представление сообщений. Так как в нашей базе данных есть наследование, а для его реализации был выбран способ “таблицы конкретных классов”. Таким образом у нас есть неявное представление сообщений, разбитое на несколько таблиц. В целом, запрос ко всем сообщениям — это редкость, но может в целом понадобится запрашивать данные из всех сообщений. Для этого и было создано данное отношение.

Запросы на получение данных и вспомогательные представления приведены в файле selects.sql.

# Запросы на изменение данных

В рамках проекта были реализованы следующие запросы:

* RegisterUser — добавляет нового зарегистрировавшегося пользователя по его логину :Login и паролю :Password. Пользователю генерируется своя соль и при помощи нее хешируется пароль, после чего добавляется в базу.
* UpdatePlayersRating — функция, возвращающая триггер, который при вставки записи в таблицу GameResult пересчитывает рейтинг для пользователей, игравших в игре, о которой была добавлена информация и применяет эти изменения в таблицу Ratings.
* CreateGameStepsSequence — функция, возвращающая триггер, который при создании новой игры создает для нее уникальную последовательность для записи ходов.
* StartGame — добавляет запись о новой начавшейся игре со следующими параметрами игры: начальное время типа игры :InitialGameTypeTime, дополнительное время типа игры :AdditionalGameTypeTime, игрок, играющий белыми фигурами :WhitePlayerId, игрок, играющий черными фигурами :BlackPlayerId.
* DoStep — добавляет запись о ходе, сделанном в игре :GameId, с клетки :FromCell, на клетку :ToCell.
* DropGameStepsSequence — функция, возвращающая триггер, который при окончании игры удаляет последовательность, созданную для данной игры в функции CreateGameStepsSequence для ходов, сделанных в данной игре.
* EndGame — добавляет запись об окончании игры :GameId и ее типе окончания :EndGameType.
* CreateTournament — добавляет запись о новом турнире с названием :TournamentName, описанием :TournamentDescription, временем начала турнира :StartTournamentTime, временем окончания турнира :EndTournamentTime, форматом записи вышеперечисленных времен :TimeFormat (например DD-MM-YYYY), дивизионом, который допущен к участию :DivisionId, начальным временем типа игры :InitialGameTypeTime и добавочным временем типа игры :AdditionalGameTypeTime.
* JoinTournament — добавляет запись о том, что пользователь :UserId решил принять участие в турнире :TournamentId. Также определяя ему изначальный рейтинг в турнире равный нулю.
* LeaveTournament — удаляет запись о том, что пользователь :UserId принимает участие в турнире :TournamentId.
* StartTournamentGame — функция, которая по турниру TournamentId, пользователю, который будет играть за белые фигуры WhitePlayerId и пользователю, который будет играть за черные фигуры BlackPlayerId добавляет информацию о новой игре и о том, что это игра играется в данном турнире. Данный запрос обернут в функцию, так как требует изменение нескольких таблиц одновременно, а значит и использование транзакции.
* SendForumMessage — добавляет информацию о сообщении от пользователя :SenderId, на форуме :ForumId и с текстом сообщения :MessageText.
* SendReplyForumMessage — функция, которая по пользователю SenderId, форуму ForumId, сообщению MessageText и сообщению, на которое отвечают ReplyMessageId, добавляет запись о новом сообщении на форуме от данного пользователя с данным текстом, которое является ответом на какое-то прошлое сообщение. Данный запрос обернут в функцию, так как требует изменение нескольких таблиц, а значит и использование транзакций.
* SendDirectMessage — добавляет информацию о сообщении от пользователя :SenderId, к пользователю :ReceiverId и с текстом сообщения :MessageText.
* SendReplyDirectMessage — функция, которая по пользователю SenderId, получателю ReceiverId, сообщению MessageText и сообщению, на которое отвечают ReplyMessageId, добавляет запись о новом личном сообщении от данного пользователя другому с данным текстом, которое является ответом на какое-то прошлое сообщение. Данный запрос обернут в функцию, так как требует изменение нескольких таблиц, а значит и использование транзакций.
* SendGameComment — добавляет информацию о сообщении от пользователя :SenderId, под игрой:GameId и с текстом сообщения :MessageText.
* SendReplyGameComment — функция, которая по пользователю SenderId, под игрой GameId, сообщению MessageText и сообщению, на которое отвечают ReplyMessageId, добавляет запись о новом сообщении на форуме от данного пользователя с данным текстом, которое является ответом на какое-то прошлое сообщение. Данный запрос обернут в функцию, так как требует изменение нескольких таблиц, а значит и использование транзакций.

В рамках проекта были реализованы следующие триггеры:

* UpdatePlayersRatingTrigger — триггер, который при добавлении новой записи в таблицу GameResults вызывает функцию UpdatePlayersRating для пересчета рейтинг пользователей.
* CreateGameStepsSequenceTrigger — триггер, который при добавлении новой записи в таблицу Games, вызывает функцию CreateGameStepsSequence, тем самым создает новую последовательность для записи ходов по данной игре в таблицу GameSteps.
* DropGameStepsSequenceTrigger — триггер, который при добавлении новой записи об окончании игры в таблицу GameResults, вызывает функцию DropGameStepsSequence, тем самым удаляет ранее созданную последовательность для записи ходов по данной игре.

Запросы на изменение данных, хранимые процедуры и триггеры приведены в файле updates.sql.