**База данный букмекерской конторы**

**1 Инфологическое проектирование**

**1.1 Анализ предметной области**

Букмекерская контора – это компания, которая принимает ставки на исход различных событий, чаще всего спортивных. Компания принимает ставки на различные события, такие как спортивные матчи, политические выборы и другие значимые события. Ставки могут быть сделаны как до начала события, так и в режиме реального времени. Также в компании рассчитывают коэффициенты, которые отражают вероятность того или иного исхода. Эти коэффициенты включают маржу, которая обеспечивает прибыль компании. Если ставка выигрывает, букмекерская контора выплачивает выигрыш игроку в соответствии с установленными коэффициентами.

Были проанализированы документы, которые отражают предметную область, далее они позволят выявить сущности, необходимые для создания базы данных.

Чеки ставок:

Эти документы фиксируют информацию о сделанных ставках. Они включают такие данные, как сумма ставки, коэффициенты, по которым была сделана ставка, и результаты ставок. С помощью них можно отслеживать все транзакции, связанные со ставками, также они могут быть полезны для анализа предпочтений клиентов, и в зависимости от этого изменять набор событий.

Отчеты о выплатах:

Документ содержит данные о выплатах выигрышей клиентам, в них указываются денежные суммы, выплаченные пользователям, даты выплат и другие детали, связанные с выплатами. Отчеты помогают контролировать финансовые потоки и обеспечивать прозрачность в расчетах с клиентами.

Регистрационные данные клиентов:

Данные включают в себя личную информацию о клиентах, такую как имя, контактные данные, а также историю их ставок и финансовых транзакций. Данный документ нужен для идентификации пользователей, анализа их активности и обеспечении безопасности транзакций.

Правила и условия:

Этот документ описывает правила работы букмекерской конторы, включая ограничения и условия ставок. Они помогают клиентам понять, как работает система ставок, какие ограничения существуют и какие права и обязанности у них есть. Документ также важен для обеспечения соблюдения законодательства и защиты интересов как клиентов, так и компании.

**1.2 Постановка задачи**

Далее были выделены основные объекты предметной области.

1. Пользователь – сущность, представляющая клиента букмекерской конторы, содержит личные данные и информацию о состоянии счета. Здесь должны содержаться данные о фамилии, имени и отчестве пользователя, его дате рождения, некоторые контактные данные, логин, пароль и баланс счета.
2. Ставка – сущность, представляющая сделанную пользователем ставку. Сущность содержит в себе данные о сумме ставки, типе ставки и ее статусе (выиграна, проиграна, в ожидании).
3. Событие – сущность, представляющая спортивное или другое событие, на которое можно сделать ставку. Предоставляет данные, касающиеся названия события, даты и времени начала, вида спорта, статуса события (завершено, в процессе, запланировано).
4. Коэффициент – сущность, представляющая коэффициенты для различных исходов события. Хранит информацию о событии, для которого коэффициент актуален, типе ставки (победа, ничья, поражение), значение коэффициента для ставки.
5. Транзакция – сущность, представляющая финансовые операции пользователя. Сущность хранит информацию о пользователе, осуществляющем перевод, дате и времени, сумме, типе транзакции (пополнение, вывод средств).
6. Команда – сущность, представляющая команду, которая участвует в каком-либо событии. Сущность содержит название команды, страну, которую она представляет, город, вид спорта.

Далее рассмотрены связи между сущностями базы данных.

1. Пользователь – Ставка: один пользователь может сделать много ставок, связь 1:М.
2. Пользователь ­– Транзакция. Один пользователь может сделать много транзакций, связь 1:М.
3. Ставка – Коэффициент. Один коэффициент может содержаться во многих ставках, связь 1:М.
4. Событие – Коэффициент. Одно событие может иметь несколько коэффициентов, связь 1:М.
5. Событие ­– Команда. Несколько команд могут принимать участие в нескольких событиях, связь М:М.

Из групп пользователей разрабатываемой базы данных было выделено несколько с указаниями требований к функциональности системы с точки зрения каждой из групп:

1. Пользователь, набор необходимых операций для него: регистрация (создание нового аккаунта с указанием личных данных), авторизация (вход в систему с использованием логина и пароля), просмотр баланса (проверка текущего состояния счета), создание ставки (выбор события и размещение ставки), просмотр истории ставок (отслеживание всех сделанных ставок), пополнение счета (внесение средств на счет через различные платежные системы), вывод средств (перевод средств с игрового счета на личный банковский счет). Из выводимых данных можно выделить информацию о текущем балансе, историю ставок и доступные события.
2. Работник, набор необходимых операций для него: управление событиями (добавление, редактирование и удаление событий), обновление коэффициентов (изменение коэффициентов для различных событий), проверка ставок (верификация и подтверждение ставок пользователей), обработка транзакций (управление финансовыми операциями, такими как пополнение счета и вывод средств). Выводимыми данными будут являться: список событий, текущие ставки, транзакции пользователей.
3. Администратор, операции для этой роли: управление пользователями (создание, редактирование и удаление учетных записей пользователей), резервное копирование данных (создание резервных копий данных для предотвращения потерь). Выводимые данные для этой роли: логи активности, информация о пользователях, системные уведомления.

**2 Разработка логической модели базы данных**

**2.1 Построение диаграммы «сущность-связь» в нотации П.Чена**

Для построения ERD-диаграммы необходимо обозначить атрибуты сущностей, а также выделить первичные ключи.

1. Пользователь:

* Логин (первичный ключ)
* Пароль
* ФИО
* Дата рождения
* Почта
* Телефон
* Баланс счета

1. Ставка:

* ID ставки (первичный ключ)
* Логин пользователя
* Сумма
* ID коэффициента
* Статус (выиграна, проиграна, в ожидании)

1. Событие:

* ID события (первичный ключ)
* ID соперников
* Название события
* Дата и время
* Вид спорта
* Статус (завершено, в процессе, запланировано)
* Результат

1. Коэффициент:

* ID коэффициента (первичный ключ)
* ID события
* Тип (победа, ничья, поражение)
* Значение коэффициента

1. Транзакция:

* ID транзакции (первичный ключ)
* Логин пользователя
* Дата и время
* Сумма
* Тип (пополнение, вывод средств)

1. Команда

* ID команды (первичный ключ)
* Название
* Страна
* Вид спорта

На основании выделенных свойств и обозначенных ранее связей между сущностями была построена ERD-диаграмма, которая показана на рисунке 2.1.

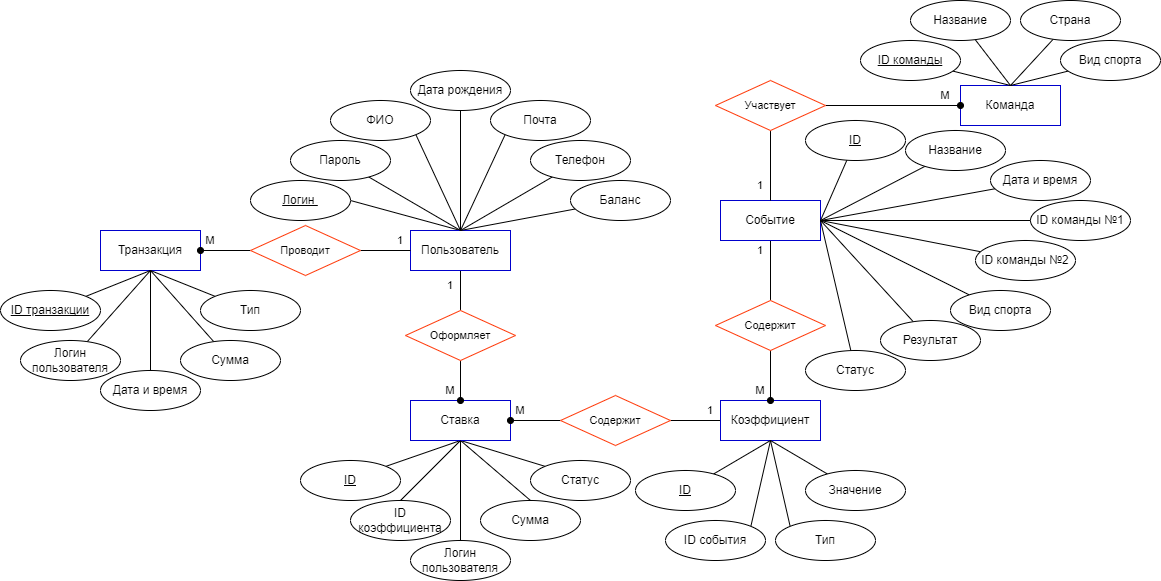


Рисунок 2.1 – ERD-диаграмма базы данных

**2.2 Построение модели, основанной на ключах и полной атрибутивной модели в нотации IDEF1X**

После получения ERD-диаграммы была построена модель, основанная на ключах, чтобы более детально рассмотреть базу данных, она показана на рисунке 2.2.



Рисунок 2.2 – Модель, основанная на ключах

Для проведения нормализации отношений в подходящей форме были выписаны все отношения:

Транзакция (id, логин\_пользователя, дата\_и\_время, сумма, тип);

Пользователь (логин, пароль, фио, дата\_рождения, почта, телефон, баланс);

Ставка (id, id\_коэффициента, логин\_пользователя, сумма, статус);

Коэффициент (id, id\_события, тип, значение);

Событие (id, название, дата\_и\_время, id\_команды1, id\_команды2, вид спорта, результат, статус, маржа);

Команда (id, название, страна, вид спорта);

**1НФ.** Все отношения в базе данных находятся в первой нормальной форме, так как все атрибуты атомарны.

**2НФ.** Все неключевые атрибуты зависят полностью от ключей, следовательно, схема находится в 2НФ.

**3НФ.** Отношение находится в 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа, соответственно схема находится в 3НФ.

**НФБК.** Так как все ключи простые, то схема, которая находится в 3НФ находится и в НФБК.

Итоговая полная атрибутивная модель после нормализации показана на рисунке 2.3.

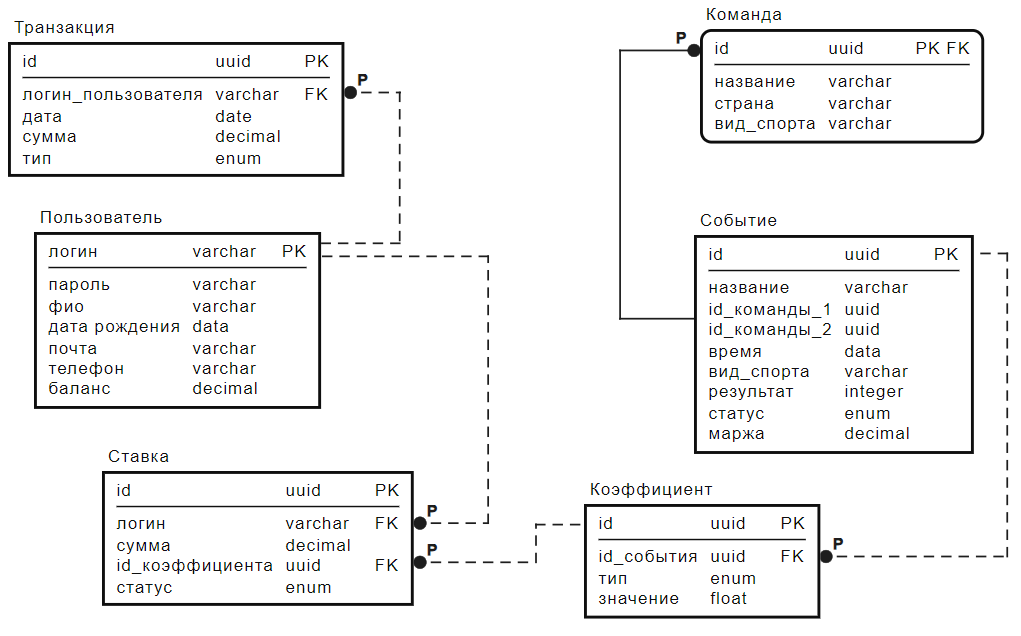


Рисунок 2.3 – Полная атрибутивная модель базы данных

**3 Разработка физической модели базы данных**

* 1. **Выбор аппаратной и программной платформы и реализация БД**

При выборе системы управления базами данных, был проведен сравнительный анализ нескольких вариантов, таких как PostgreSQL, MySQL и SQLite. На основе этого анализа, PostgreSQL была выбрана как оптимальное решение.

PostgreSQL предлагает высокую производительность при обработке сложных запросов и транзакций, что делает её подходящей для больших объемов данных. Её масштабируемость позволяет легко адаптироваться к растущим потребностям проекта.

С точки зрения надежности, PostgreSQL обеспечивает ACID-транзакции и механизмы восстановления после сбоев, что гарантирует целостность и сохранность данных.

Активное сообщество и обширная документация делают PostgreSQL доступной и поддерживаемой платформой. Кроме того, её бесплатная и открытая лицензия делает её экономически эффективным решением для проектов любого масштаба.

Для создания базы данных в начале были добавлены enum типы для облегчения работы, это показано в листинге 3.1.

Листинг 3.1 ­– Добавление enum полей

create type transaction\_type as enum ('deposit', 'withdrawal');

create type coefficient\_type as enum ('win\_team1', 'win\_team2', 'draw');

create type bet\_ status as enum ('win', 'lost', 'in\_progress');

create type event\_status as enum ('scheduled', 'in\_progress', 'completed')

create type result\_status as enum ('win\_team1', 'win\_team2', 'draw', 'in\_progress')

После этого с помощью запросов были созданы таблицы с необходимыми ограничениями, что продемонстрировано в листингах 3.2-3.7:

Листинг 3.2 ­– Запрос на создание таблицы «Пользователь»

create table customer (

username varchar(50) primary key,

password varchar(255) not null,

full\_name varchar(100) ,

birth\_date date,

email varchar(100),

phone varchar(20),

balance decimal(10, 2)

);

Листинг 3.3 ­– Запрос на создание таблицы «Транзакции»

create table transaction (

id uuid primary key,

username varchar(50) not null,

date\_time timestamp,

amount decimal(10, 2) not null,

type transaction\_type not null,

foreign key (username) references customer(username) on delete cascade on update cascade

);

Листинг 3.4 ­– Запрос на создание таблицы «Команда»

create table team (

id uuid primary key,

name varchar(100),

country varchar(50),

sport varchar(50) not null

);

Листинг 3.5 ­– Запрос на создание таблицы «Событие»

create table event (

id uuid primary key,

name varchar(100) not null,

date\_time timestamp not null,

team1\_id uuid,

team2\_id uuid,

sport varchar(50) not null,

result varchar(50),

status event\_status,

margin decimal(5, 2),

foreign key (team1\_id) references team(id) on delete set null on update cascade,

foreign key (team2\_id) references team(id) on delete set null on update cascade

);

Листинг 3.6 ­– Запрос на создание таблицы «Коэффициент»

create table coefficient (

id uuid primary key,

event\_id uuid not null,

type coefficient\_type not null,

value decimal(5, 2) not null,

foreign key (event\_id) references event(id) on delete restrict on update cascade

);

Листинг 3.7 ­– Запрос на создание таблицы «Ставка»

create table bet (

id uuid primary key,

coefficient\_id uuid not null,

username varchar(50) not null,

amount decimal(10, 2) not null,

status bet\_status,

foreign key (coefficient\_id) references coefficient(id) on delete cascade on update cascade,

foreign key (username) references customer(username) on delete cascade on update cascade

);

Для корректной работы базы данных необходимо создать триггерные функции и триггеры, чтобы правильно обновлять содержимое базы данных в зависимости от сложившейся ситуации. Триггерная функция для обновления баланса пользователя в зависимости от проведенной транзакции показана в листинге 3.8, а триггер для нее в листинге 3.9.

Листинг 3.8 – Триггерная функция для обновления баланса

create or replace function update\_user\_balance()

returns trigger

language plpgsql

as $$

begin

if (new.type = 'deposit') then

update customer set balance = balance + new.amount where username = new.username;

elsif (new.type = 'withdrawal') then

update customer set balance = balance - new.amount where username = new.username;

end if;

return new;

end;

$$

Листинг 3.9 – Триггер для обновления баланса

create trigger trg\_update\_user\_balance

after insert on transaction

for each row

execute function update\_user\_balance();

Далее была добавлена функция, которая инициирует создание транзакции для пользователей, когда ставка переходит из статуса ожидания в статус выигрыша или проигрыша, запрос для нее показан в листинге 3.10, а триггер в листинге 3.11.

Листинг 3.10 – Триггерная функция для создания транзакции

create or replace function handle\_bet\_status\_change()

returns trigger

language plpgsql

as $$

declare

coefficient\_value decimal(5, 2);

transaction\_amount decimal(10, 2);

begin

select value into coefficient\_value from coefficient where id = new.coefficient\_id;

transaction\_amount := new.amount \* coefficient\_value;

if (new.status = 'won') then

insert into transaction (id, username, date\_time, amount, type)

values (uuid\_generate\_v4(), new.username, now(), transaction\_amount, 'deposit');

elsif (new.status = 'lost') then

insert into transaction (id, username, date\_time, amount, type)

values (uuid\_generate\_v4(), new.username, now(), new.amount, 'withdrawal');

end if;

return new;

end;

$$

Листинг 3.11 – Триггер для создания транзакции

create trigger trg\_handle\_bet\_status\_change

after update of status on bet

for each row

execute function handle\_bet\_status\_change();

Затем была добавлена функция, которая проверяет одинаковый ли вид спорта у команд, для которых создается событие, запрос для нее показан в листинге 3.12, а триггер в листинге 3.13.

Листинг 3.12 – Триггерная функция для сравнения команд

create or replace function validate\_teams\_sport\_type()

returns trigger

language plpgsql

as $$

declare

sport1 varchar(50);

sport2 varchar(50);

begin

select sport into sport1 from team where id = new.team1\_id;

select sport into sport2 from team where id = new.team2\_id;

if sport1 != sport2 then

raise exception 'Teams must be of the same sport type';

end if;

return new;

end;

$$

Листинг 3.13 – Триггер для сравнения команд

create trigger trg\_validate\_teams\_sport\_type

before insert or update on event

for each row

execute function validate\_teams\_sport\_type();

Также была добавлена функция, которая пересчитывает коэффициенты при добавлении новой ставки на событие, запрос для нее показан в листинге 3.14, а триггер в листинге 3.15.

Листинг 3.14 – Триггерная функция пересчета коэффициентов

create or replace function recalculate\_coefficients()

returns trigger

language plpgsql

as $$

declare

total\_bets\_team1 integer;

total\_bets\_team2 integer;

total\_bets\_draw integer;

coefficient\_id uuid;

margin decimal(5, 2);

probability\_team1 decimal(5, 2);

probability\_team2 decimal(5, 2);

probability\_draw decimal(5, 2);

new\_value\_team1 decimal(5, 2);

new\_value\_team2 decimal(5, 2);

new\_value\_draw decimal(5, 2);

begin

coefficient\_id := new.coefficient\_id;

select margin into margin from event where id = (select event\_id from coefficient where id = coefficient\_id);

select count(\*) into total\_bets\_team1

from bet

where coefficient\_id in (

select id

from coefficient

where event\_id = (select event\_id from coefficient where id = coefficient\_id)

and type = 'win\_team1'

);

select count(\*) into total\_bets\_team2

from bet

where coefficient\_id in (

select id

from coefficient

where event\_id = (select event\_id from coefficient where id = coefficient\_id)

and type = 'win\_team2'

);

select count(\*) into total\_bets\_draw

from bet

where coefficient\_id in (

select id

from coefficient

where event\_id = (select event\_id from coefficient where id = coefficient\_id)

and type = 'draw'

);

probability\_team1 := total\_bets\_team1::decimal / (total\_bets\_team1 + total\_bets\_team2 + total\_bets\_draw);

probability\_team2 := total\_bets\_team2::decimal / (total\_bets\_team1 + total\_bets\_team2 + total\_bets\_draw);

probability\_draw := total\_bets\_draw::decimal / (total\_bets\_team1 + total\_bets\_team2 + total\_bets\_draw);

new\_value\_team1 := 1/(probability\_team1\*(1+margin));

new\_value\_team2 := 1/(probability\_team2\*(1+margin));

new\_value\_draw := 1/(probability\_draw\*(1+margin));

update coefficient set value = new\_value\_team1 where id = coefficient\_id and type = 'win\_team1';

update coefficient set value = new\_value\_team2 where id = coefficient\_id and type = 'win\_team2';

update coefficient set value = new\_value\_draw where id = coefficient\_id and type = 'draw';

return new;

end;

$$

Листинг 3.15 – Триггер для сравнения команд

create trigger trg\_recalculate\_coefficients

after insert on bet

for each row

execute function recalculate\_coefficients();

Помимо этого, была добавлена функция, которая проверяет состояние баланса пользователя перед совершением ставки, запрос для нее показан в листинге 3.16, а триггер в листинге 3.17.

Листинг 3.16 – Триггерная функция для проверки баланса для ставки

create or replace function check\_balance\_before\_bet()

returns trigger

language plpgsql

as $$

declare

user\_balance decimal(10, 2);

potential\_loss decimal(10, 2);

coefficient\_value decimal(5, 2);

begin

select balance into user\_balance from customer where username = new.username;

select value into coefficient\_value from coefficient where id = new.coefficient\_id;

potential\_loss := new.amount \* coefficient\_value;

if user\_balance < potential\_loss then

raise exception 'Insufficient funds for the bet';

end if;

return new;

end;

$$

Листинг 3.17 – Триггер для проверки баланса для ставки

create trigger trg\_check\_balance\_before\_bet

before insert on bet

for each row

execute function check\_balance\_before\_bet();

Последней была добавлена функция, которая не дает совершить транзакцию при недостаточном количестве денег на счету, запрос для нее показан в листинге 3.18, а триггер в листинге 3.19.

Листинг 3.18 – Триггерная функция для проверки баланса при транзакции

create or replace function check\_balance\_before\_transaction()

returns trigger

language plpgsql

as $$

declare

user\_balance decimal(10, 2);

begin

select balance into user\_balance from customer where username = new.username;

if new.type = 'withdrawal' and user\_balance < new.amount then

raise exception 'Insufficient funds for the withdrawal';

end if;

return new;

end;

$$

Листинг 3.19 – Триггер для проверки баланса при транзакции

create trigger trg\_check\_balance\_before\_transaction

before insert on transaction

for each row

execute function check\_balance\_before\_transaction();

Для того, чтобы проверить правильность выполнения написанных триггерных функций все таблицы были заполнены, что показано на запросах 3.16-3.21.

Листинг 3.16 ­– Запрос на заполнение таблицы «Пользователь»

create table customer (

username varchar(50) primary key,

password varchar(255) not null,

full\_name varchar(100) ,

birth\_date date,

email varchar(100),

phone varchar(20),

balance decimal(10, 2)

);