**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.

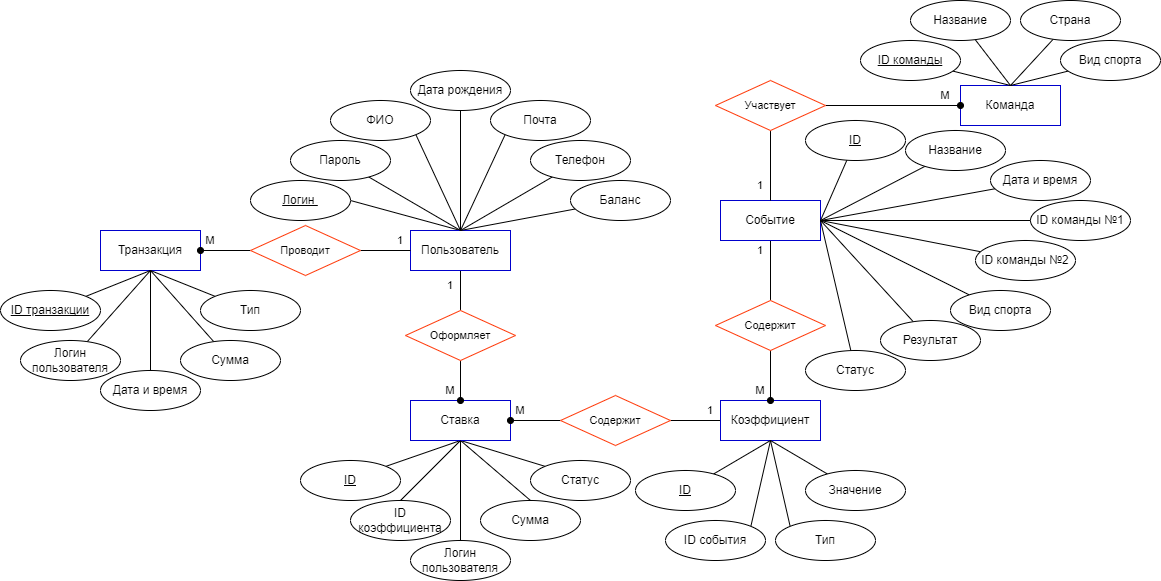


Рисунок 2.1 – ERD-диаграмма базы данных

**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.

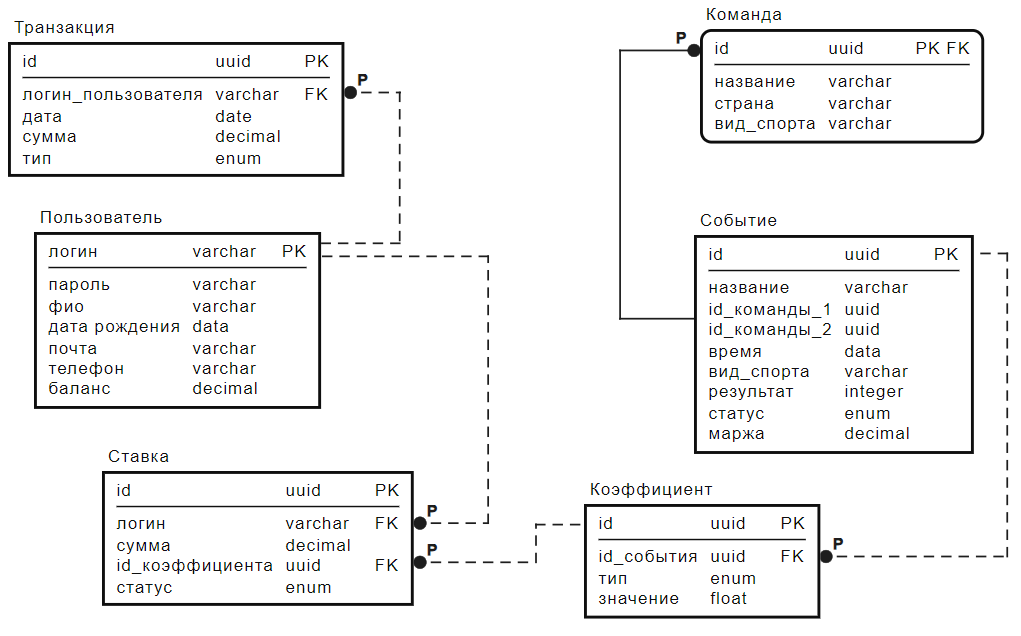


Рисунок 2.3 – Полная атрибутивная модель базы данных

**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 bet\_status as enum ('win', 'lost', 'in\_progress');

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

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

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) not null

primary key,

password varchar(255) not null,

full\_name varchar(100),

birth\_date date,

email varchar(100),

phone varchar(20),

balance numeric(10, 2)

);

Листинг 3.3 ­– Запрос на создание таблицы «Транзакции»

create table transaction

(

id uuid default uuid\_generate\_v4() not null

primary key,

username varchar(50) not null

references customer

on update cascade on delete cascade,

date\_time timestamp default CURRENT\_TIMESTAMP,

amount numeric(10, 2) not null,

type transaction\_type not null

);

Листинг 3.4 ­– Запрос на создание таблицы «Команда»

create table team

(

id uuid default uuid\_generate\_v4() not null

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 default uuid\_generate\_v4() not null

primary key,

name varchar(100) not null,

date\_time timestamp not null,

team1\_id uuid references team

on update cascade on delete set null,

team2\_id uuid references team

on update cascade on delete set null,

sport varchar(50) not null,

result result\_status default 'in\_progress'::result\_status,

margin numeric(5, 2)

);

Листинг 3.6 ­– Запрос на создание таблицы «Коэффициент»

create table coefficient

(

id uuid default uuid\_generate\_v4() not null

primary key,

event\_id uuid not null

references event

on update cascade on delete restrict,

type coefficient\_type not null,

value numeric(5, 2) default 1 not null

);

Листинг 3.7 ­– Запрос на создание таблицы «Ставка»

create table bet

(

id uuid default uuid\_generate\_v4() not null

primary key,

coefficient\_id uuid not null

references coefficient

on update cascade on delete cascade,

username varchar(50) not null

references customer

on update cascade on delete cascade,

amount numeric(10, 2) not null,

status bet\_status default 'in\_progress'::bet\_status

);

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 function update\_user\_balance() returns trigger

language plpgsql

as

$$

declare

balance\_tmp decimal;

begin

if (new.type = 'deposit') then

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

elsif (new.type = 'withdrawal') then

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

if (balance\_tmp < new.amount) then

raise exception 'Недостаточно средств на счету для снятия.';

else

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

end if;

end if;

return new;

end;

$$;

Листинг 3.9 – Триггер для обновления баланса

create trigger change\_user\_balance

before insert or update

on transaction

for each row

execute procedure update\_user\_balance();

Далее была добавлена функция, которая инициирует создание транзакции для снятия средств с баланса при создании ставки, запрос для нее показан в листинге 3.10, а триггер в листинге 3.11.

Листинг 3.10 – Триггерная функция для создания ставки

create function make\_bet() returns trigger

language plpgsql

as

$$

begin

begin

insert into transaction

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

exception

when others then

raise exception 'Ошибка при вставке значений в таблицу transaction: %', sqlerrm;

end;

return new;

end;

$$;

Листинг 3.11 – Триггер для создания транзакции

create trigger on\_create\_bet

before insert

on bet

for each row

execute procedure make\_bet();

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 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 check\_sport\_types

before insert or update

on event

for each row

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

Также была добавлена функция, которая инициирует создание трех коэффициентов при создании события, запрос для нее показан в листинге 3.14, а триггер в листинге 3.15.

Листинг 3.14 – Триггерная функция для создания коэффициентов

create function create\_event\_coefficients() returns trigger

language plpgsql

as

$$

begin

insert into coefficient(id, event\_id, type) values

(uuid\_generate\_v4(), new.id, 'win\_team1'),

(uuid\_generate\_v4(), new.id, 'win\_team2'),

(uuid\_generate\_v4(), new.id, 'draw');

return new;

end;

$$;

Листинг 3.15 – Триггер для создания коэффициентов

create trigger add\_coefficients

after insert

on event

for each row

execute procedure create\_event\_coefficients();

Помимо этого, была добавлена функция, которая пересчитывает коэффициенты события при добавлении новой ставки, запрос для нее показан в листинге 3.16, а триггер в листинге 3.17.

Листинг 3.16 – Триггерная функция для пересчета коэффициентов

create function recalculate\_coefficients() returns trigger

language plpgsql

as

$$

declare

total\_amount\_team1 decimal(10, 2) := 0;

total\_amount\_team2 decimal(10, 2) := 0;

total\_amount\_draw decimal(10, 2) := 0;

coeff\_id uuid;

v\_event\_id uuid;

event\_margin decimal(5, 2);

probability\_team1 decimal(5, 2) := 0;

probability\_team2 decimal(5, 2) := 0;

probability\_draw decimal(5, 2) := 0;

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

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

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

total\_amount decimal(10, 2);

begin

coeff\_id := new.coefficient\_id;

select coefficient.event\_id into v\_event\_id from coefficient where coefficient.id = coeff\_id;

select event.margin into event\_margin

from event

where event.id = v\_event\_id;

select coalesce(sum(amount), 0) into total\_amount\_team1

from bet

where bet.coefficient\_id in (

select coefficient.id

from coefficient

where coefficient.event\_id = v\_event\_id

and coefficient.type = 'win\_team1'

);

select coalesce(sum(amount), 0) into total\_amount\_team2

from bet

where bet.coefficient\_id in (

select coefficient.id

from coefficient

where coefficient.event\_id = v\_event\_id

and coefficient.type = 'win\_team2'

);

select coalesce(sum(amount), 0) into total\_amount\_draw

from bet

where bet.coefficient\_id in (

select coefficient.id

from coefficient

where coefficient.event\_id = v\_event\_id

and coefficient.type = 'draw'

);

total\_amount := total\_amount\_team1 + total\_amount\_team2 + total\_amount\_draw;

if total\_amount > 0 then

probability\_team1 := total\_amount\_team1::decimal / total\_amount;

probability\_team2 := total\_amount\_team2::decimal / total\_amount;

probability\_draw := total\_amount\_draw::decimal / total\_amount;

end if;

if total\_amount\_team1 = 0 then

new\_value\_team1 := 10.00;

else

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

end if;

if total\_amount\_team2 = 0 then

new\_value\_team2 := 10.00;

else

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

end if;

if total\_amount\_draw = 0 then

new\_value\_draw := 10.00;

else

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

end if;

update coefficient set value = new\_value\_team1 where coefficient.event\_id = v\_event\_id and coefficient.type = 'win\_team1';

update coefficient set value = new\_value\_team2 where coefficient.event\_id = v\_event\_id and coefficient.type = 'win\_team2';

update coefficient set value = new\_value\_draw where coefficient.event\_id = v\_event\_id and coefficient.type = 'draw';

return new;

end;

$$;

Листинг 3.17 – Триггер для пересчета коэффициентов

create trigger update\_event\_status

after update

of result

on event

for each row

execute procedure on\_update\_event\_status();

Далее была добавлена функция, которая отмечает выигрышные ставки при закрытии события и проставляет соответствующие статусы, запрос для нее показан в листинге 3.18, а триггер в листинге 3.19.

Листинг 3.18 – Триггерная функция для изменения статуса ставки

create function on\_update\_event\_status() returns trigger

language plpgsql

as

$$

declare

changed\_event\_id uuid;

win\_status result\_status;

win\_coefficient\_id uuid;

begin

changed\_event\_id := new.id;

win\_status := new.result;

select coefficient.id into win\_coefficient\_id from coefficient

where cast(win\_status as text) = cast(coefficient.type as text) and changed\_event\_id = coefficient.event\_id;

update bet set status = 'win'

where bet.coefficient\_id = win\_coefficient\_id;

RAISE NOTICE 'This is a log message: %', win\_coefficient\_id;

update bet set status = 'lost'

where bet.coefficient\_id in (

select coefficient.id from coefficient

where changed\_event\_id = coefficient.event\_id

) and status != 'win';

return new;

end;

$$;

Листинг 3.19 – Триггер для изменения статуса ставки

create trigger update\_event\_status

after update

of result

on event

for each row

execute procedure on\_update\_event\_status();

Последней была добавлена функция, которая проводит транзакции по пополнению счета при закрытии ставки в случае его выигрыша, запрос для нее показан в листинге 3.20, а триггер в листинге 3.21.

Листинг 3.20 – Триггерная функция для проведения транзакций при выигрыше

create function on\_update\_event\_status() returns trigger

language plpgsql

as

$$

declare

changed\_event\_id uuid;

win\_status result\_status;

win\_coefficient\_id uuid;

begin

changed\_event\_id := new.id;

win\_status := new.result;

select coefficient.id into win\_coefficient\_id from coefficient

where cast(win\_status as text) = cast(coefficient.type as text) and changed\_event\_id = coefficient.event\_id;

update bet set status = 'win'

where bet.coefficient\_id = win\_coefficient\_id;

RAISE NOTICE 'This is a log message: %', win\_coefficient\_id;

update bet set status = 'lost'

where bet.coefficient\_id in (

select coefficient.id from coefficient

where changed\_event\_id = coefficient.event\_id

) and status != 'win';

return new;

end;

$$;

Листинг 3.21 – Триггер для проведения транзакций при выигрыше

create trigger closed\_bet\_transaction\_trigger

after update

of status

on bet

for each row

execute procedure make\_closed\_bet\_transactions();

create function delete\_bet() returns trigger

language plpgsql

as

$$

begin

begin

if old.status = 'in\_progress' then

insert into transaction

values (uuid\_generate\_v4(), old.username, CURRENT\_TIMESTAMP, old.amount, 'deposit');

else

raise exception 'Событие уже закончилось';

end if;

exception

when others then

raise exception 'Ошибка при вставке значений в таблицу transaction: %', sqlerrm;

end;

return old;

end;

$$;

alter function delete\_bet() owner to postgres;