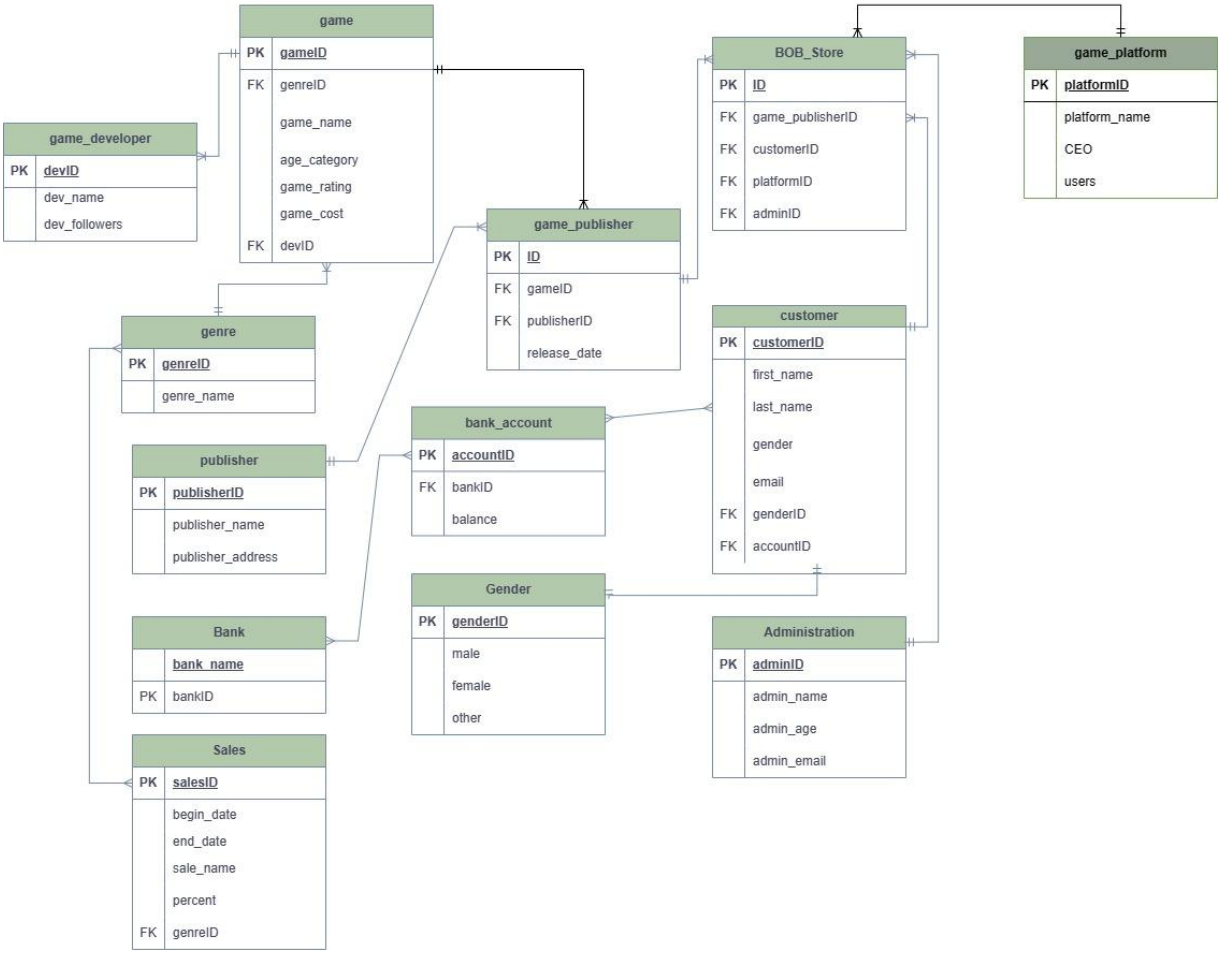
DBMS2 Project Report

Batyikhan Ismamutov 210103207
Abeuov Olzhas 210103109
Belgibayev Bagzhan 210103416

Introduction to the system

BOB Store is an online game store which provides users with games from different platforms, working with different game publishers. In our store users can buy games with multiple bank accounts, our administrators check orders and provide a wide range of help for our customers.We have weekly, monthly and seasonal sales.When you register to our store you can choose “other” gender, because BOB store is modern and tolerant to all people.
Our system is very stable and has different technologies to check our DataBase, we analyze data to provide service of more quality.

ER diagram

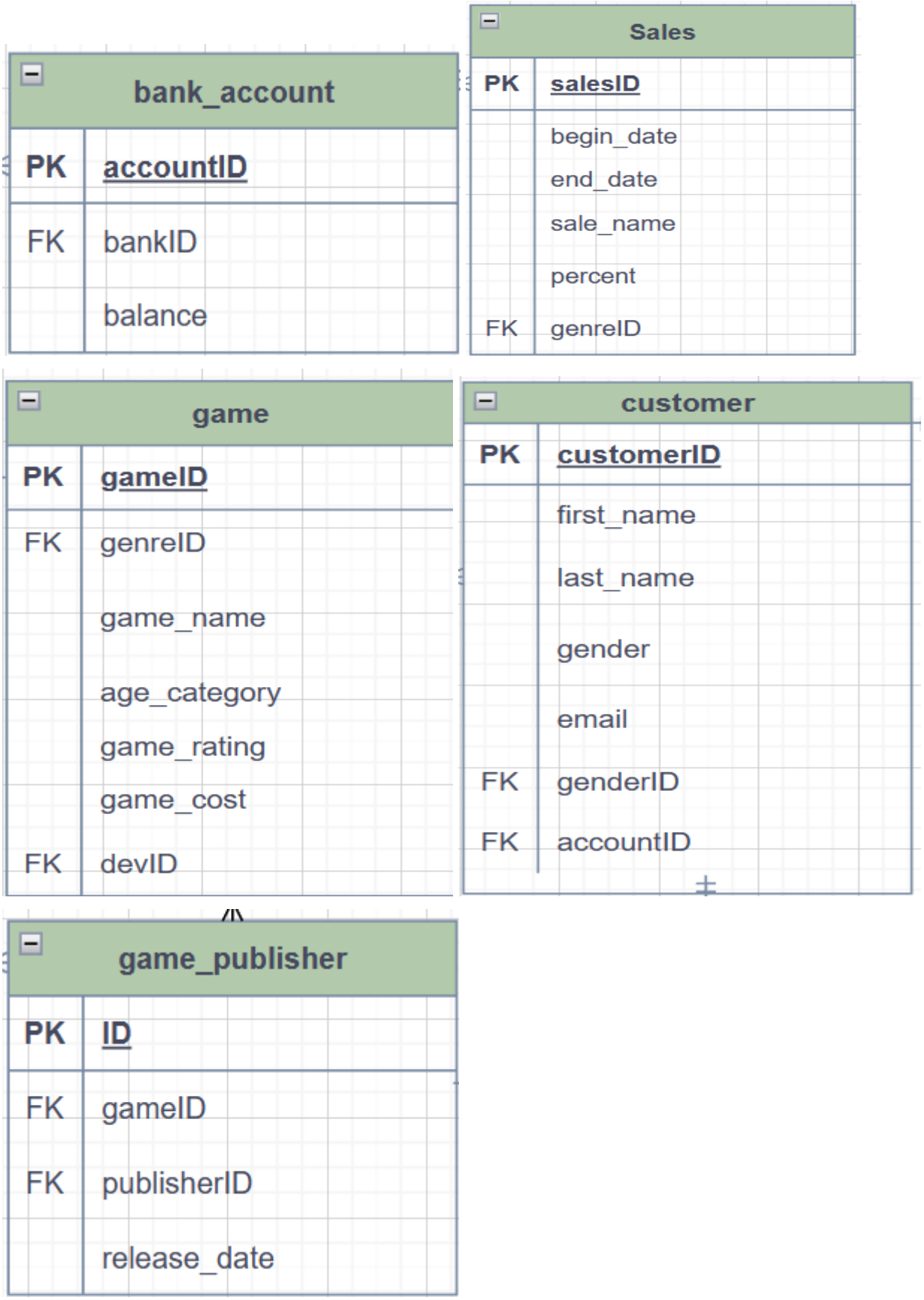


Explanation of why the structure follows normal forms

1NF

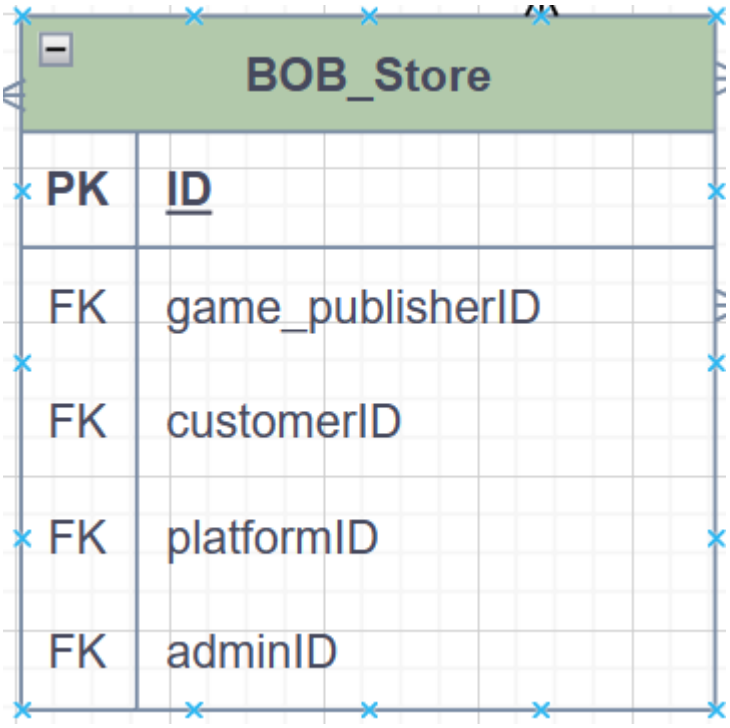
All tables above follow 1NF, they have primary keys, and all columns contain atomic values (meaning there are no repeating groups or arrays).1 Table has only one meaning.

2 NF


bank_account
PK accountID
FK bankID
balance
Sales
PK salesID
begin_date
end_date
sale_name
percent
FK genreID
game
PK gameID
FK genreID
game_name
age_category
game_rating
game_cost
FK devID
customer
PK customerID
first_name
last_name
gender
email
FK genderID
FK accountID
game_publisher
PK ID
FK gameID
FK publisherID
release_date

In 2NF, a table must first satisfy 1NF and then each non-key column in the table must be dependent on the entire primary key, not just a part of it. In other words, each non-key column must be functionally dependent on the primary key. For example in table bank_account, all non-key columns functionally dependent on accountID.
In table Sales we need salesID, which determines other columns, so that it is a unique identifier for them.

3 NF



In 3NF, this table follow second normal form; any column is dependent on a non-Primary key, so there is no partial dependency.
A transitive dependency occurs when a non-key column is dependent on another non-key column, which is itself dependent on the primary key.
For example game_publisherID is dependent on ID, but any other column is dependent on game_publisherID and this is true for every other column

Explanation and coding part

1. Procedure which does group by information

```
CREATE OR REPLACE PROCEDURE group_by_genreID IS
   CURSOR c_game IS
      SELECT genreID, COUNT(gameID) AS total_games, AVG(game_rating) AS avg_rating,
SUM(game_cost) AS total_cost
      FROM game
```

```
      GROUP BY genreID;
   rec_game c_game%ROWTYPE;
BEGIN
   OPEN c_game;
   LOOP
      FETCH c_game INTO rec_game;
      EXIT WHEN c_game%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('GenreID: ' || rec_game.genreID);
      DBMS_OUTPUT.PUT_LINE('Total Games: ' || rec_game.total_games);
      DBMS_OUTPUT.PUT_LINE('Average Rating: ' || TO_CHAR(rec_game.avg_rating,
'fm99990.00'));
      DBMS_OUTPUT.PUT_LINE('Total Cost: ' || rec_game.total_cost);
      DBMS_OUTPUT.PUT_LINE(' ');
   END LOOP;
   CLOSE c_game;
END;
```

This is procedure named "group_by_genreID" that retrieves data from the "game" table and groups it by the "genreID" column. The purpose of this procedure is to display summary information about games grouped by their genreID. The procedure takes no input parameters and has no return value. The output is displayed using the DBMS_OUTPUT.PUT_LINE function, so it is typically viewed in a console or terminal window rather than returned to an application or user interface.

2. Function which counts the number of records

```
DECLARE
   games SYS_REFCURSOR;
   v_genreID game.genreID%TYPE;
   v_game_name game.game_name%TYPE;
   v_game_rating game.game_rating%TYPE;
   i NUMBER := 1;
BEGIN
   games := top_games();
LOOP
FETCH games INTO v_genreID, v_game_name, v_game_rating;
EXIT WHEN games%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE(i || ') ' || 'Genre '|| v_genreID || ' - ' || v_game_name || ' - '|| 'Rating
'|| v_game_rating);
   i := i + 1;
END LOOP;
   i := i-1;
   DBMS_OUTPUT.PUT_LINE('Number of rows affected: ' || i);
CLOSE games;
END;

CREATE OR REPLACE FUNCTION top_games
RETURN SYS_REFCURSOR
IS
games SYS_REFCURSOR;
```

```
BEGIN
OPEN games FOR
SELECT genreID, game_name, game_rating
FROM game
WHERE game_rating >= 9;
RETURN games;
END;
```

This is function named "top_games" that returns the list of games that have a rating of 9 or higher. The purpose of this function is to return the number of games that have a high rating, which could be used for various purposes such as generating reports, calculating statistics, or making decisions based on the popularity of certain games.

3 Procedure which uses SQL%ROWCOUNT to determine the number of rows affected

```
CREATE OR REPLACE PROCEDURE count_customers_by_platform(p_platformID IN
NUMBER) AS
   v_num_customers NUMBER;
BEGIN
   SELECT COUNT(DISTINCT customerID) INTO v_num_customers
   FROM BOB_Store
   WHERE platformID = p_platformID;
   DBMS_OUTPUT.PUT_LINE('Number of customers using platform ' || p_platformID || ': ' ||
v_num_customers);
   DBMS_OUTPUT.PUT_LINE('Number of rows affected: ' || SQL%ROWCOUNT);
END;
BEGIN
   count_customers_by_platform(2);
END;
```

This is a procedure named "count_customers_by_platform" that takes an input parameter "p_platformID" of type NUMBER representing the ID of a platform. The procedure retrieves the count of distinct customers who have used the specified platform by executing a SELECT statement that filters the "BOB_Store" table based on the given platform ID. Purpose of this procedure is to retrieve the count of distinct customers who have used a specific platform and display the result in the console or terminal window using the DBMS_OUTPUT.PUT_LINE function. This procedure takes one input parameter and has no return value.

4 Add user-defined exception which disallows to enter title of item (e.g. book) to be less than 5 characters

```
CREATE OR REPLACE TRIGGER validate_game_name
BEFORE INSERT ON game
```

```
FOR EACH ROW
BEGIN
IF LENGTH(:NEW.game_name) < 5 THEN
   RAISE_APPLICATION_ERROR(-20001, 'GAME_NAME should be at least 5 characters
long');END IF;
END;
```

This is a trigger named "validate_game_name" that is executed before an insert operation is performed on the "game" table. This trigger is to ensure that the "game_name" column in the "game" table contains at least 5 characters before a new row is inserted into it. If the length is less than 5, the trigger raises an RAISE_APPLICATION_ERROR function to prevent the insert operation from being executed. This trigger takes no input parameters and has no return value.

5 Create a trigger before insert on any entity which will show the current number of rows in the table

```
CREATE OR REPLACE TRIGGER sales_before_insert
BEFORE INSERT ON sales
FOR EACH ROW
DECLARE
   rows_count NUMBER;
BEGIN
   SELECT COUNT(*) INTO rows_count FROM sales;
   DBMS_OUTPUT.PUT_LINE('Current number of rows in sales table: ' || rows_count);
END;
```

This is a trigger named "sales_before_insert" that is executed before an insert operation is performed on the "sales" table.
The purpose of this trigger is to display the current number of rows in the "sales" table before a new row is inserted into it. This trigger takes no input parameters and has no return value.