Club Penguin - Relational Database

Sophia Masone
CMPT 308
May 2025

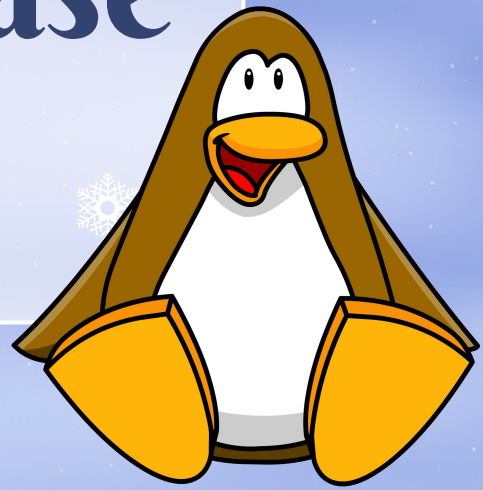
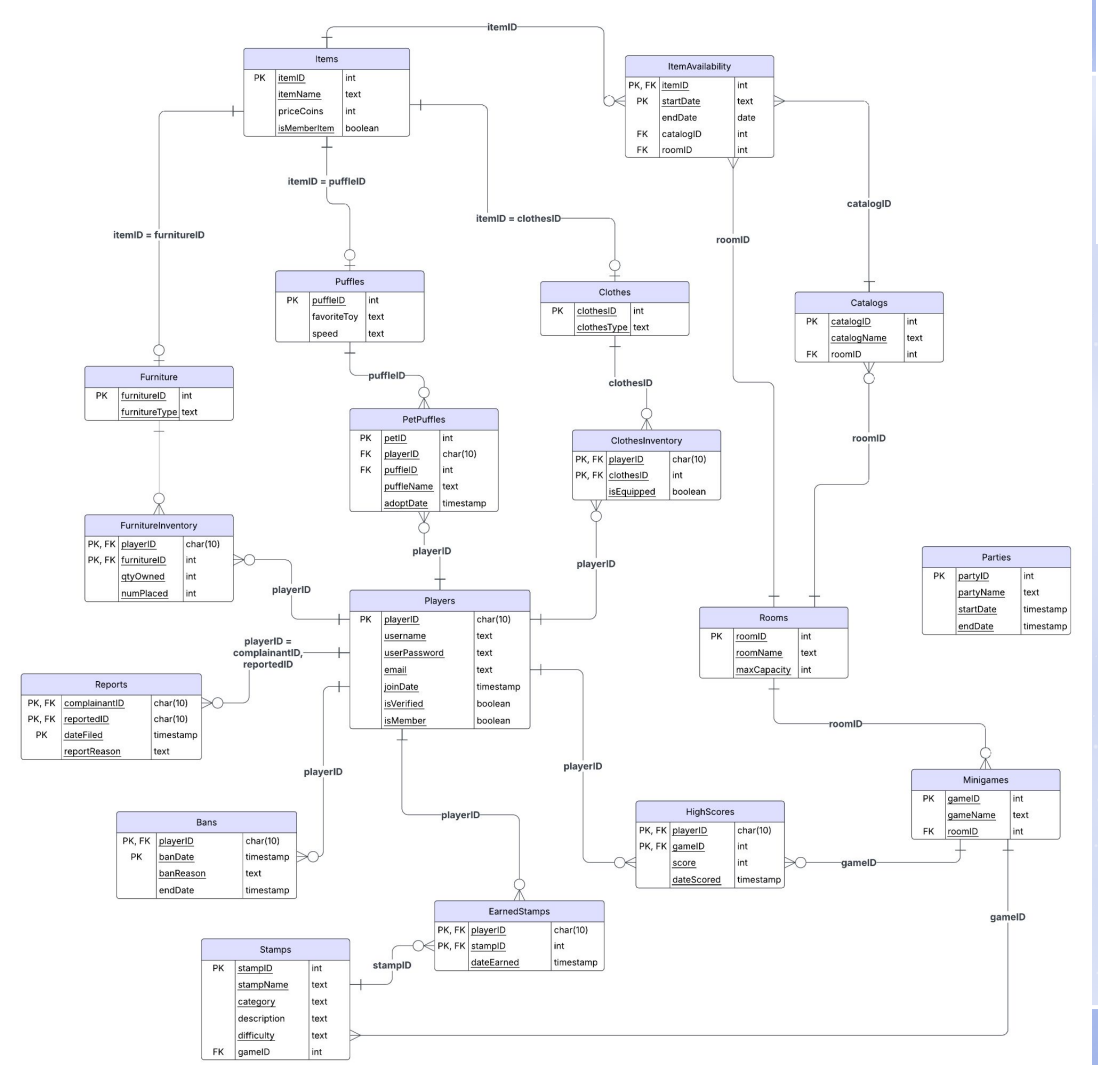


Table of Contents

Executive Summary

Club Penguin is children's online game that I grew up playing. Though it shut down in 2017, I still think about it now and then, and how I would design things were I in charge. Throughout this document, I will go over my interpretation of a database for this game. My objective was to design a database that would include all data necessary to keep track of in managing an online game. As such, the database keeps track of things like players, locations, in-game items and their acquisition, and more. It also allows moderators to monitor user reports and prior bans. Besides player data and certain IDs, the sample data I use is all real data from the original game.

Entity-Relationship Diagram



Note: Underlined items are non-nullable

Tables

Players

This table keeps track of all accounts made in the game.

Functional dependencies: playerID → username, userPassword, email, joinDate, isVerified, isMember

```
CREATE TABLE Players (
    playerID char(10)      not null unique,
    username text          not null unique,
    userPassword text      not null,
    email text             not null unique,
    joinDate timestamp     not null,
    isVerified boolean     not null,
    isMember boolean       not null,
  primary key(playerID)
);
```

	playerid [PK] character (10)	username text	userpassword text	email text	joindate timestamp without time zone	isverified boolean	ismember boolean
1	p123456789	databasegod	$1$8ulMJcz0$tqt4Ae8vApezNLclgXhd6.	labouseur@email.com	2025-01-01 12:30:43	true	true
2	p000000000	penguin	1sHyZX7hq$4ON/OBxH.IBYi1T6u1ToB1	penguin@email.com	2005-10-04 00:00:00	false	false
3	p777777777	birdy14193	1yUd98w/E$qLcJ0pMUgzN.dHwCO.YZb/	bird@birdy.com	2010-08-08 14:12:48	true	true
4	p111222333	EVILPENGUIN	1gRlatD0A$ZhD5N7tZGhfC7sgBrcezf0	someone@gmail.com	2008-05-27 22:09:52	true	false
5	p002000001	happyguy	1FmMXAFJU$dZa.ZYNhdYPnJOg1XFrcX/	sunshine@yahoo.com	2010-02-12 06:31:09	true	true

Parties

This table keeps track of all events that occur in-game.

Functional dependencies: partyID → partyName, startDate, endDate

```
CREATE TABLE Parties (
    partyID int           not null unique,
    partyName text        not null,
    startDate timestamp   not null,
    endDate timestamp     not null,
  primary key(partyID)
);
```

	partyid [PK] integer	partyname text	startdate timestamp without time zone	enddate timestamp without time zone
1	0	Beta Test Party	2005-09-21 15:00:00	2005-09-21 17:00:00
2	12345	April Fools Party 2009	2009-03-28 00:00:00	2009-04-02 00:00:00
3	55555	Operation: Blackout	2012-11-15 00:00:00	2012-12-04 00:00:00

Rooms

This table keeps track of all locations in-game.

Functional dependencies: roomID → roomName, maxCapacity


```
CREATE TABLE Rooms (
    roomID int        not null unique,
    roomName text     not null unique,
    maxCapacity int   not null,
  primary key(roomID)
);
```

	roomid [PK] integer	roomname text	maxcapacity integer
1	110	Coffee Shop	80
2	130	Gift Shop	80
3	100	Town	120
4	310	Pet Shop	80
5	808	Mine	80
6	340	Stage	80

Catalogs

This table keeps track of all catalogs and their locations. Since some catalogs are found in a specific room, it references roomID to point to where.

Functional dependencies: catalogID → catalogName, roomID

```
CREATE TABLE Catalogs (
    catalogID int       not null unique,
    catalogName text    not null unique,
    roomID int          references Rooms(roomID),
  primary key(catalogID)
);
```

	catalogid [PK] integer	catalogname text	roomid integer
1	0	Furniture & Igloo Catalog	[null]
2	1	Penguin Style	130
3	2	Puffle Catalog	310
4	3	Costume Trunk	340

Items

This table keeps track of all in-game items. priceCoins is nullable, since some items are free.

Functional dependencies: itemID → itemName, priceCoins, isMemberItem

```
CREATE TABLE Items (
    itemID int              not null unique,
    itemName text           not null unique,
    priceCoins int,
    isMemberItem boolean not null,
  primary key(itemID)
);
```

	itemid [PK] integer	itemname text	pricecoins integer	ismemberitem boolean
1	477	Court Jester Hat	250	true
2	204	Astro Barrier T-Shirt	200	true
3	5588	Fruitcake	[null]	false
4	112	Light Blue	20	false
5	7259	Herbertech Pin	[null]	false
6	5189	Cool Mittens	200	true
7	106	Mona Lisa	3000	true
8	893	Banana Couch	[null]	false
9	617	Salon Chair	400	true
10	750	Blue Puffle	400	false
11	759	Brown Puffle	400	true
12	5230	Rainbow Puffle	[null]	true

ItemAvailability

This table keeps track of when and where items are available in-game. Items can be available in multiple place / over multiple windows of time, can be permanently available, and can be available in either a catalog or a room.

Functional dependencies: itemID, startDate → endDate, catalogID, roomID

```
CREATE TABLE ItemAvailability (
    itemID int          not null references Items(itemID),
    startDate date      not null,
    endDate date,
    catalogID int       references Catalogs(catalogID),
    roomID int          references Rooms(roomID)
                        CHECK (catalogID is not null or roomID is not null),
  primary key(itemID, startDate)
);
```

	itemid [PK] integer	startdate [PK] date	enddate date	catalogid integer	roomid integer
1	477	2007-12-14	2008-01-11	3	[null]
2	477	2009-05-01	2009-09-04	1	[null]
3	112	2005-08-22	[null]	1	[null]
4	7259	2016-02-17	2016-03-02	[null]	100
5	106	2005-08-22	2006-10-20	0	[null]
6	750	2006-03-17	[null]	2	[null]

Clothes

This table is a subtype of Items. It contains data regarding things the player can have their avatar wear.

Functional dependencies: clothesID → clothesType

	clothesid [PK] integer	clothestype text
1	477	Head
2	204	body
3	5588	hand
4	112	color
5	5189	hand

```
CREATE TABLE Clothes (
    clothesID int       not null unique references Items(itemID),
    clothesType text    not null
                        CHECK(lower(clothesType) in ('head', 'face', 'neck', 'body', 'feet', 'hand', 'color', 'background', 'pin')),
  primary key(clothesID)
);
```

ClothesInventory

This table maps which players own which clothing items, and if they are wearing them.

Functional dependencies: playerID, clothesID → isEquipped

	playerid [PK] character (10)	clothesid [PK] integer	isequipped boolean
1	p777777777	477	true
2	p123456789	204	true
3	p123456789	477	false
4	p000000000	112	true
5	p777777777	5588	false

```
CREATE TABLE ClothesInventory (
    playerID char(10)     not null references Players(playerID),
    clothesID int         not null references Clothes(clothesID),
    isEquipped boolean    default FALSE,
  primary key(playerID, clothesID)
);
```

Furniture

This table is a subtype of Items. It contains data regarding things the player can place in their igloo (a player's house).

Functional dependencies: furnitureID → furnitureType

	furnitureid [PK] integer	furnituretype text
1	106	wall
2	893	floor
3	617	pet

```
CREATE TABLE Furniture (
    furnitureID int       not null unique references Items(itemID),
    furnitureType text    not null
                          CHECK(lower(furnitureType) in ('wall', 'room', 'floor', 'pet')),
  primary key(furnitureID)
);
```

FurnitureInventory

This table maps which players own which furniture items, how many they own, and how many they have placed.

Functional dependencies: playerID, furnitureID → qtyOwned, numPlaced

```
CREATE TABLE FurnitureInventory (
    playerID char(10)    not null references Players(playerID),
    furnitureID int      not null references Furniture(furnitureID),
    qtyOwned int         not null,
    numPlaced int        default 0,
    CHECK(numPlaced <= qtyOwned),
  primary key(playerID, furnitureID)
);
```

	playerid [PK] character (10)	furnitureid [PK] integer	qtyowned integer	numplaced integer
1	p123456789	106	1	0
2	p111222333	106	5	3
3	p777777777	893	2	2

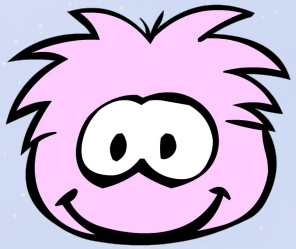
Puffles

This table is a subtype of Items. Puffles are in-game pets. This table keeps track of the different types of puffles.

Functional dependencies: puffleID → favoriteToy, speed

	puffleid [PK] integer	favoritetoy text	speed text
1	750	Beach ball	Slow
2	759	Rocket	[null]
3	5230	Cloud	Fast

```
CREATE TABLE Puffles (
    puffleID int        not null unique references Items(itemID),
    favoriteToy text,
    speed text,
  primary key(puffleID)
);
```



PetPuffles

This table holds data about puffles that have been adopted and belong to players. A player can have many puffles, including multiple of the same type.

Functional dependencies: petID → playerID, puffleID, puffleName, adoptDate

```
CREATE TABLE PetPuffles (
    petID int              not null unique,
    playerID char(10)      not null references Players(playerID),
    puffleID int           not null references Puffles(puffleID),
    puffleName text        not null,
    adoptDate timestamp    not null,
  primary key(petID)
);
```

	petid [PK] integer	playerid character (10)	puffleid integer	pufflename text	adoptdate timestamp without time zone
1	0	p777777777	759	Cookie	2010-08-10 11:19:35
2	1	p777777777	5230	Lucky	2014-09-05 19:05:44
3	2	p002000001	750	Sky	2015-10-31 00:00:00

Minigames

This table keeps track of all minigames within the game, and where they can be found.

Functional dependencies: gameID → gameName, roomID

	gameid [PK] integer	gamename text	roomid integer
1	0	Bean Counters	110
2	1	Smoothie Smash	110
3	2	Puffle Launch	310
4	3	Pufflescapes	310
5	4	Cart Surfer	808

```
CREATE TABLE Minigames (
    gameID int      not null unique,
    gameName text   not null unique,
    roomID int      not null references Rooms(roomID),
  primary key(gameID)
);
```

Highscores

This table keeps track of players' personal highscores in minigames.

Functional dependencies: playerID, gameID → score, dateScored

```
CREATE TABLE Highscores (
    playerID char(10)       not null references Players(playerID),
    gameID int              not null references Minigames(gameID),
    score int               not null,
    dateScored timestamp    not null,
  primary key(playerID, gameID)
);
```

	playerid [PK] character (10)	gameid [PK] integer	score integer	datescored timestamp without time zone
1	p123456789	4	2100	2025-01-02 20:05:28
2	p777777777	1	110	2015-12-23 14:54:19
3	p777777777	4	2300	2013-04-17 18:04:52
4	p000000000	1	160	2009-05-05 13:02:55
5	p002000001	1	220	2012-09-12 15:09:28

Stamps

This table keeps track of data regarding stamps, which are Club Penguin's form of achievements. Some stamps are associated with certain minigames.

Functional dependencies: stampID → stampName, category, description, difficulty, gameID

```
CREATE TABLE Stamps (
    stampID int         not null unique,
    stampName text      not null unique,
    category text       not null
                        CHECK(lower(category) in ('characters', 'party', 'activities', 'games')),
    description text,
    difficulty text     not null
                        CHECK(lower(difficulty) in ('easy', 'medium', 'hard', 'extreme')),
    gameID int          references Minigames(gameID),
  primary key(stampID)
);
```

	stampid [PK] integer	stampname text	category text	description text	difficulty text	gameid integer
1	212	Great Balance stamp	Games	Recover from a wobble	Easy	4
2	439	Mountaineer stamp	Party	Reach a mountain peak	Hard	[null]
3	466	Herbert stamp	characters	Be in the same room as Herbert	extreme	[null]
4	15	Going Places stamp	Activities	Waddle around 30 rooms without using the map	medium	[null]

EarnedStamps

This table keeps track of which stamps players have earned.

Functional dependencies: stampID, playerID → dateEarned

```
CREATE TABLE EarnedStamps (
    playerID char(10)       not null references Players(playerID),
    stampID int             not null references Stamps(stampID),
    dateEarned timestamp    not null,
  primary key(playerID, stampID)
);
```

	playerid [PK] character (10)	stampid [PK] integer	dateearned timestamp without time zone
1	p002000001	15	2013-03-12 14:16:25
2	p123456789	439	2025-01-05 00:04:54
3	p777777777	466	2015-08-08 15:29:38
4	p111222333	15	2008-05-28 02:05:33

Reports

This table keeps track of the in-game report feature, in which players can report other players for breaking the rules, so that a moderator can review the reported player.

Functional dependencies: complainantID, reportedID, dateFiled → reportReason

```
CREATE TABLE Reports (
    complainantID char(10)    not null references Players(playerID),
    reportedID char(10)       not null references Players(playerID),
    dateFiled timestamp       not null,
    reportReason text         not null
                              CHECK(lower(reportReason) in ('bad words', 'personal information', 'rude or mean', 'bad penguin name')),
  primary key(complainantID, reportedID, dateFiled)
);
```

	complainantid [PK] character (10)	reportedid [PK] character (10)	datefiled [PK] timestamp without time zone	reportreason text
1	p002000001	p111222333	2011-05-07 10:26:32	rude or mean
2	p777777777	p111222333	2012-10-12 20:54:09	bad words
3	p111222333	p000000000	2012-11-01 08:05:33	personal information

Bans

This table keeps track of players who have been banned, and the date their ban will end. Players can also be banned permanently.

Functional dependencies: playerID, banDate → banReason, endDate

```
CREATE TABLE Bans (
    playerID char(10)   not null references Players(playerID),
    banDate timestamp   not null,
    banReason text      not null,
    endDate timestamp,
  primary key(playerID, banDate)
);
```

	playerid [PK] character (10)	bandate [PK] timestamp without time zone	banreason text	enddate timestamp without time zone
1	p111222333	2011-05-08 08:17:11	rude or mean	2011-05-09 08:17:11
2	p111222333	2012-10-12 22:58:12	bad words	2012-10-15 22:58:12
3	p111222333	2012-10-16 14:45:43	bad words	[null]

Views

ClothesDetails

This view contains all data regarding clothes items.

```
CREATE OR REPLACE VIEW ClothesDetails as (
    select c.clothesID, i.itemName,  c.clothesType, i.priceCoins, i.isMemberItem
    from Clothes c inner join Items i on c.clothesID = i.itemID
);
```

	clothesid integer	itemname text	clothestype text	pricecoins integer	ismemberitem boolean
1	477	Court Jester Hat	Head	250	true
2	204	Astro Barrier T-Shirt	body	200	true
3	5588	Fruitcake	hand	[null]	false
4	112	Light Blue	color	20	false
5	5189	Cool Mittens	hand	200	true

ClothesDetails - Example query

This query returns a table of non-member clothing items.

```
select *
from ClothesDetails
where not isMemberItem;
```

	clothesid integer	itemname text	clothestype text	pricecoins integer	ismemberitem boolean
1	5588	Fruitcake	hand	[null]	false
2	112	Light Blue	color	20	false

FurnitureDetails

This view contains all data regarding furniture items.

```
CREATE OR REPLACE VIEW FurnitureDetails as (
    select f.furnitureID, i.itemName,  f.furnitureType, i.priceCoins, i.isMemberItem
    from Furniture f inner join Items i on f.furnitureID = i.itemID
);
```

	furnitureid integer	itemname text	furnituretype text	pricecoins integer	ismemberitem boolean
1	106	Mona Lisa	wall	3000	true
2	893	Banana Couch	floor	[null]	false
3	617	Salon Chair	pet	400	true

FurnitureDetails - Example query

This query returns a table of all floor type furniture items.

```
select *
from furnitureDetails
where furnitureType ilike 'floor';
```

	furnitureid integer	itemname text	furnituretype text	pricecoins integer	ismemberitem boolean
1	893	Banana Couch	floor	[null]	false

ValidPlayers

This view contains all players with full access to the game, meaning their account is verified and they are not currently banned.

```
CREATE OR REPLACE VIEW ValidPlayers as (
    select p.playerID, p.username, p.userPassword, p.email, p.joinDate, p.isMember
    from Players as p
    where p.isVerified and p.playerID not in (select playerID
                                              from Bans
                                              where endDate is null or endDate > now())
);
```

	playerid character (10)	username text	userpassword text	email text	joindate timestamp without time zone	ismember boolean
1	p123456789	databasegod	1dJYqHW8V$67wkAGcNwq1XhTZ00…	labouseur@email.com	2025-01-01 12:30:43	true
2	p777777777	birdy14193	1q6fFk39w$vNcYy3v0/ihY/mGN.jVf21	bird@birdy.com	2010-08-08 14:12:48	true
3	p002000001	happyguy	1lQuH6ylX$jL8Cumooy0akud9PAkhaA1	sunshine@yahoo.com	2010-02-12 06:31:09	true

ValidPlayers - Example query

This query returns the EarnedStamps of currently valid accounts.

```
select *
from EarnedStamps
where playerID in (select playerID
                   from ValidPlayers);
```

	playerid [PK] character (10)	stampid [PK] integer	dateearned timestamp without time zone
1	p002000001	15	2013-03-12 14:16:25
2	p123456789	439	2025-01-05 00:04:54
3	p777777777	466	2015-08-08 15:29:38

Reports

Report 1

This query returns the number of players who joined in 2010.

```
select count(playerID) as "Number of players"
from Players
where date_part('year', joinDate) = 2010;
```

	Number of players bigint
1	2

Report 2

This query returns data on items that are currently available to players in the game, and the room in which they can be found. If they are found in a catalog, it returns the room the catalog is in.

```
select i.itemName, coalesce(r1.roomName, r2.roomName) as roomName, i.priceCoins, i.isMemberItem
from Items i inner join ItemAvailability ia on i.itemID = ia.itemID
             left outer join Rooms r1 on ia.roomID = r1.roomID
             left outer join Catalogs c on ia.catalogID = c.catalogID
             left outer join Rooms r2 on c.roomID = r2.roomID
where ia.endDate is null or ia.endDate > now()
order by itemName ASC;
```

	itemname text	roomname text	pricecoins integer	ismemberitem boolean
1	Blue Puffle	Pet Shop	400	false
2	Light Blue	Gift Shop	20	false

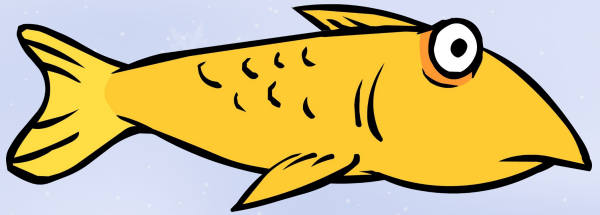
Report 3

This query returns the total number of clothing and furniture items a player owns.

```
select playerID, sum(numClothes) as "Clothes owned", sum(numFurn) as "Furniture owned"
from (
      select playerID, 1 as numClothes, 0 as numFurn
      from ClothesInventory
      union
      select playerID, 0 as numClothes, qtyOwned as numFurn
      from FurnitureInventory
)
group by playerID;
```

	playerid character (10)	Clothes owned bigint	Furniture owned bigint
1	p777777777	1	2
2	p000000000	1	0
3	p123456789	1	1
4	p111222333	0	5

Stored Procedures



minigameLeaderboard

This procedure returns a ranking of highscores among all players in a given minigame, along with the username of the player.

```
CREATE OR REPLACE FUNCTION minigameLeaderboard (minigameID int)
returns table (username text, score int) as $$
begin
    return query
        select p.username, h.score
        from Highscores h inner join Players p on h.playerID = p.playerID
        where h.gameID = 04
        order by score DESC;
end;
$$
language plpgsql;
```

minigameLeaderboard - Example outputs

```
select *
from minigameLeaderboard(04);
```

	username text	score integer
1	birdy14193	2300
2	databasegod	2100

```
select *
from minigameLeaderboard(01);
```

	username text	score integer
1	happyguy	220
2	penguin	160
3	birdy14193	110

checkPlayerOutfit

This procedure shows all currently equipped clothing items and what type they are for a given player.

```
CREATE OR REPLACE FUNCTION checkPlayerOutfit (pid char(10))
returns table (clothesType text, clothesName text) as $$
begin
    return query
        select c.clothesType, c.itemName
        from clothesInventory inv inner join ClothesDetails c on inv.clothesID = c.clothesID
        where pid = inv.playerID and inv.isEquipped;
end;
$$
language plpgsql;
```

checkPlayerOutfit - Example outputs

```
select *
from checkPlayerOutfit('p777777777');
```

	clothestype text	clothesname text
1	Head	Court Jester Hat
2	hand	Fruitcake

```
select *
from checkPlayerOutfit('p123456789');
```

	clothestype text	clothesname text
1	body	Astro Barrier T-Shirt

uniqueTypeEquipped

This procedure checks new items/updates in ClothesInventory that isEquipped, to see if the player is already wearing another item of that type. If so, the old one is set to no longer be isEquipped. This ensures players are only ever wearing one head item, body item, etc.

```
CREATE OR REPLACE FUNCTION uniqueTypeEquipped ()
returns TRIGGER as $$
begin
    if new.isEquipped then
        UPDATE ClothesInventory inv
        set isEquipped = FALSE
        from Clothes c
        where inv.clothesID = c.clothesID
            and inv.clothesID != new.clothesID
            and inv.playerID = new.playerID
            and c.clothesType ilike (select clothesType from Clothes where clothesID = new.clothesID)
            and inv.isEquipped;
    end if;
    return new;
end;
$$
language plpgsql;
```

*Usage as a trigger shown on p.43-44

hashPassword

This procedure encrypts userPasswords added to the Players table.

```
CREATE OR REPLACE FUNCTION hashPassword ()
returns TRIGGER as $$
begin
    new.userPassword = crypt(new.userPassword, gen_salt('md5'));
    return new;
end;
$$
language plpgsql;
```

*Usage as a trigger shown on p.44-45

Triggers

uniqueTypeEquipped

After an item is added or updated in ClothesInventory, this trigger runs to make sure clothing items are not overlapping.

```
CREATE OR REPLACE TRIGGER uniqueTypeEquipped
after INSERT OR UPDATE of isEquipped
                          on ClothesInventory
for each row
execute procedure uniqueTypeEquipped();
```

uniqueTypeEquipped - Example

(Note that 5588 and 5189 are both hand items)

```
INSERT INTO ClothesInventory (playerID,     clothesID, isEquipped)
VALUES                       ('p777777777', 5189,      TRUE);
```

Before insert:

	playerid [PK] character (10)	clothesid [PK] integer	isequipped boolean
1	p777777777	477	true
2	p123456789	204	true
3	p123456789	477	false
4	p000000000	112	true
5	p777777777	5588	true

After insert:

	playerid [PK] character (10)	clothesid [PK] integer	isequipped boolean
1	p777777777	477	true
2	p123456789	204	true
3	p123456789	477	false
4	p000000000	112	true
5	p777777777	5588	false
6	p777777777	5189	true

hashPassword

Whenever a userPassword is added/updated in Players, this trigger runs to automatically encrypt and store only the encryption.

```
CREATE OR REPLACE TRIGGER hashPassword
before INSERT OR UPDATE on Players
for each row
execute procedure hashPassword(userPassword);
```

hashPassword - Example

```
INSERT INTO Players (playerID,      username,     userPassword, email,                 joinDate,              isVerified,  isMember)
VALUES              ('p123456789', 'databasegod', 'alpaca',     'labouseur@email.com', '01.01.2025 12:30:43', TRUE,        TRUE),
                    ('p000000000', 'penguin',     'abcdefg',    'penguin@email.com',   '10.04.2005 00:00:00', FALSE,       FALSE),
                    ('p777777777', 'birdy14193',  'yay!yay',    'bird@birdy.com',      '08.08.2010 14:12:48', TRUE,        TRUE),
                    ('p111222333', 'EVILPENGUIN', '12345',      'someone@gmail.com',   '05.27.2008 22:09:52', TRUE,        FALSE),
                    ('p002000001', 'happyguy',    'password',   'sunshine@yahoo.com',  '02.12.2010 06:31:09', TRUE,        TRUE);
```

	playerid [PK] character (10)	username text	userpassword text	email text	joindate timestamp without time zone	isverified boolean	ismember boolean
1	p123456789	databasegod	$1$8ulMJcz0$tqt4Ae8vApezNLclgXhd6.	labouseur@email.com	2025-01-01 12:30:43	true	true
2	p000000000	penguin	1sHyZX7hq$4ON/OBxH.IBYi1T6u1ToB1	penguin@email.com	2005-10-04 00:00:00	false	false
3	p777777777	birdy14193	1yUd98w/E$qLcJ0pMUgzN.dHwCO.YZb/	bird@birdy.com	2010-08-08 14:12:48	true	true
4	p111222333	EVILPENGUIN	1gRIatD0A$ZhD5N7tZGhfC7sgBrcezf0	someone@gmail.com	2008-05-27 22:09:52	true	false
5	p002000001	happyguy	1FmMXAFJU$dZa.ZYNhdYPnJOg1XFrcX/	sunshine@yahoo.com	2010-02-12 06:31:09	true	true

Security

User Roles - Admin

This role is for the user given total administrative control over the database.

```
CREATE ROLE admin;
grant all on all tables in schema public to admin;
```

User Roles - Game management

These are roles for individuals involved in management of the game and the features within it.

gameDeveloper - As someone who will be coding in new items, parties, etc., they are able to select, insert, and update in tables related to game features, but cannot view or edit any player data.

```
CREATE ROLE gameDeveloper;
grant SELECT, INSERT, UPDATE on Parties to gameDeveloper;
grant SELECT, INSERT, UPDATE on Rooms to gameDeveloper;
grant SELECT, INSERT, UPDATE on Catalogs to gameDeveloper;
grant SELECT, INSERT, UPDATE on Items to gameDeveloper;
grant SELECT, INSERT, UPDATE on ItemAvailability to gameDeveloper;
grant SELECT, INSERT, UPDATE on Clothes to gameDeveloper;
grant SELECT, INSERT, UPDATE on Furniture to gameDeveloper;
grant SELECT, INSERT, UPDATE on Puffles to gameDeveloper;
grant SELECT, INSERT, UPDATE on Minigames to gameDeveloper;
grant SELECT, INSERT, UPDATE on Stamps to gameDeveloper;
```

gameWriter - Similar to gameDeveloper, except gameWriters can only select.

```
CREATE ROLE gameWriter;
grant SELECT on Parties to gameWriter;
grant SELECT on Rooms to gameWriter;
grant SELECT on Catalogs to gameWriter;
grant SELECT on Items to gameWriter;
grant SELECT on ItemAvailability to gameWriter;
grant SELECT on Clothes to gameWriter;
grant SELECT on Furniture to gameWriter;
grant SELECT on Puffles to gameWriter;
grant SELECT on Minigames to gameWriter;
grant SELECT on Stamps to gameWriter;
```

User Roles - Player management

These are roles for individuals involved in management of players and their associated data.

playerManager - Able to select, insert, and update player into tables that keep track of individual player information. In a full implementation, this would be taken care of by the system itself, but it's still helpful to have a role for someone who ensures accuracy of this data.

```
CREATE ROLE playerManager;
grant SELECT, INSERT, UPDATE on Players to playerManager;
grant SELECT, INSERT, UPDATE on ClothesInventory to playerManager;
grant SELECT, INSERT, UPDATE on FurnitureInventory to playerManager;
grant SELECT, INSERT, UPDATE on PetPuffles to playerManager;
grant SELECT, INSERT, UPDATE on Highscores to playerManager;
grant SELECT, INSERT, UPDATE on EarnedStamps to playerManager;
grant SELECT, INSERT, UPDATE on Reports to playerManager;
grant SELECT, INSERT, UPDATE on Bans to playerManager;
```

moderator - As someone tasked with managing player behavior / enforcement of rules, moderators can select player data and reports, and can select, insert, and update bans.

```
CREATE ROLE moderator;
grant SELECT on Players to moderator;
grant SELECT on Reports to moderator;
grant SELECT, INSERT, UPDATE on Bans to moderator;
```

User Roles - Revoke

What if moderators abuse the information they find in the Players table?

Their access can easily be taken away:

```
revoke all on Players from moderator;
```

```
REVOKE

Query returned successfully in 46 msec.
```

Known Problems/Future Enhancements

- ★ Currently, there is nothing to prevent timestamps from being set in the future, or acquisition timestamps from being set as after the player's join date. In the future, this could be solved with check constraints and triggers. (Some timestamps, particularly end dates, should be able to be set in the future)
- ★ The current model does not allow for minigames or catalogs to change rooms, which is something that happened a few times in the later years of Club Penguin. This could be solved using an associative entity to map features to their locations.
- ★ There is nothing preventing non-members from acquiring or equipping/placing member-only items. It's important to note that players can still *own* member-only items from past memberships. The acquisition/equipment issue could be solved with a trigger.

Known Problems/Future Enhancements, cont.

- ★ It could be useful to have a “Moderators” table as a subtype of Players, which would allow Bans to keep track of which moderator placed the ban. There should also be a way to keep track of if the ban was executed by an automatic system.
- ★ It could also be useful to have a “ChatLogs” table for moderation purposes.
- ★ ItemAvailability could be updated to include parties as part of an item’s availability alongside catalogs and rooms, since some items are party-exclusive.
- ★ Player’s adopted puffles can be walked, which will be shown on their avatar, but the current checkPlayerOutfit procedure doesn’t show this. A future version might have a boolean “isWalking” as a field of PetPuffles, which is then referenced in the checkPlayerOutfit procedure to be incorporated into the returned table. isWalking would also need a trigger similar to uniqueTypeEquipped to ensure only one puffle is being walked.