Stanley Yang

Executive Summary

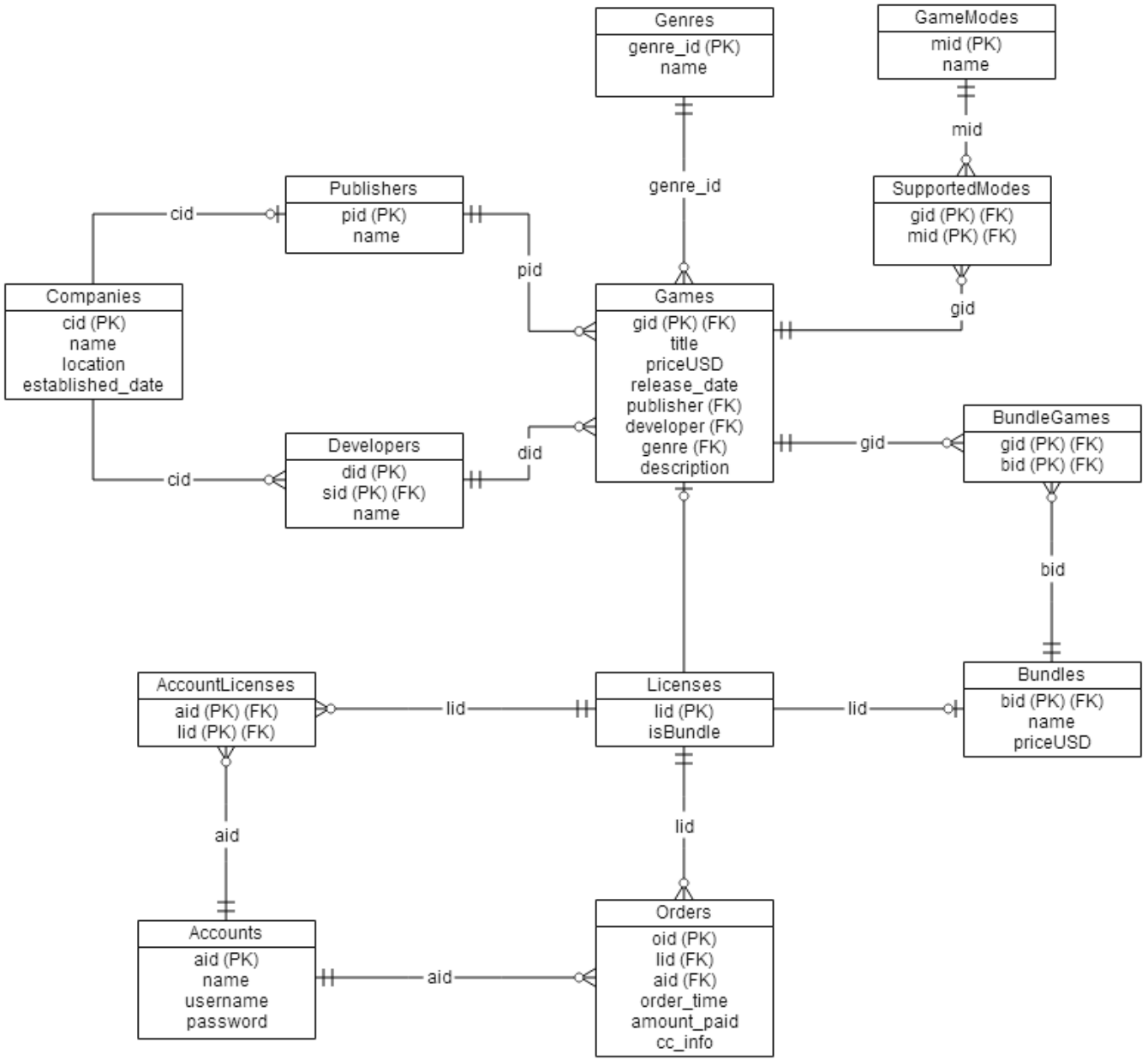
Digital Distribution is the future of PC gaming. Tap Gaming has created a new service to allow its customers to access the games they love. Mist is digital distribution platform that will carry a large selection of games from various publishers and developers. It will need a database that is capable of keeping track of all this and its users' personal libraries of games. There will be three users of this database. The employees of Tap Gaming, Tap Gaming customer support, and the Mist account owners.

The design of the database focus around ensuring game information is accurate and customers only have access to the games they have paid for. It will keep a log of all the orders that are placed and the relevant information. It also will have support for any bundles that Tap Gaming may wish to offer. In order to support this, every product, such as a game or bundle, offered by Mist will be given a license to uniquely identify it. Developer and Publisher information will also be supported by this database.

Following this summary will be a low level overview of the database. Then there will be further details on tables that will be implemented along with sample data. Sample views, stored procedures, and triggers will also be provided to demonstrate how the database will be useful and have integrity.

This database was written and tested on Postgresql 9.2.

E/R Diagram


Genres
genre_id (PK)
name
GameModes
mid (PK)
name
mid
genre_id
Publishers
pid (PK)
name
SupportedModes
gid (PK) (FK)
mid (PK) (FK)
cid
pid
gid
Companies
cid (PK)
name
location
established_date
Games
gid (PK) (FK)
title
priceUSD
release_date
publisher (FK)
developer (FK)
genre (FK)
description
BundleGames
gid (PK) (FK)
bid (PK) (FK)
Developers
did (PK)
sid (PK) (FK)
name
did
gid
cid
bid
AccountLicenses
aid (PK) (FK)
lid (PK) (FK)
Licenses
lid (PK)
isBundle
Bundles
bid (PK) (FK)
name
priceUSD
lid
lid
aid
lid
Accounts
aid (PK)
name
username
password
Orders
oid (PK)
lid (FK)
aid (FK)
order_time
amount_paid
cc_info
aid

Companies table

SQL Create Statements:

CREATE TABLE IF NOT EXISTS Companies (

cid SERIAL,

name VARCHAR (40),

location VARCHAR(40),

established_date DATETIME,

PRIMARY KEY (sid)

);

Functional Dependencies:

cid → name, location, established_date

Notes:

The Companies table is required for the following two tables, Developers and Publishers. Sometimes companies publish their own games, sometimes they use another company. The name field is only a reference name to a company. It identifies what most people call the entire company. For example, Ubisoft has 37 branches of developers and publishes their own games.

Sample Data:

cid	name	location	established_date
1	Tap	Private Island	2013-12-01 00:00:00
2	Valve	Bellevue, Washington	1996-08-24 00:00:00
3	Ubisoft	Montreuil, France	1986-03-13 00:00:00
4	Microsoft	Redmond, Washington	1975-04-04 00:00:00

Developers table

SQL Create Statements:

```
CREATE TABLE IF NOT EXISTS Developers (
did SERIAL,
cid INT REFERENCES companies(cid),
name VARCHAR(40),
PRIMARY KEY (did)
);
```

Functional Dependencies:

did → name

Notes:

Currently the only information required is the name of the studio and the parent company of the studio.

Sample Data:

did	cid	name
1	1	Tap Gaming
2	2	Valve
3	3	Ubisoft Montreal
4	3	RedLynx
5	3	THQ Montreal
6	4	Bungie
7	4	Ensemble

Publishers table

SQL Create Statements:

CREATE TABLE IF NOT EXISTS Publishers (

pid INT REFERENCES companies(cid),

name VARCHAR (40),

PRIMARY KEY (pid)

);

Functional Dependencies:

pid →name

Notes:

This table is basically a subtype of the studios. The name field is the official name the company publishes under.

Sample Data:

pid	name
1	Tap Gaming
2	Valve Corporation
3	Ubisoft Entertainment
4	Microsoft Game Studios

Genres table

SQL Create Statements:

CREATE TABLE IF NOT EXISTS Genres (

genre_id SERIAL,

name VARCHAR(20),

PRIMARY KEY (genre_id)

);

Functional Dependencies:

genre_id → name

Notes:

Genres are in their own table because sometimes genres change their names. For example, League of Legends and DotA 2 currently are undecided what their genre should be called. They currently consider themselves MOBA (massive online battle arena) and ARTS (Action Real Time Strategy) respectively. Suppose MOBA becomes the standard then it would be simple to change ARTS to MOBA as well and all games that were once considered ARTS now also be considered MOBA.

Sample Data:

genre_id	name
1	Puzzle
2	First Person Shooter
3	Action
4	Massive Multiplayer Battle Arena
5	Action Real Time Strategy

GameModes table

SQL Create Statements:

CREATE TABLE IF NOT EXISTS GameModes (

mid SERIAL,

name VARCHAR (20),

PRIMARY KEY (mid)

);

Functional Dependencies:

mid →name

Notes:

A table for all possible game modes.

Sample Data:

mid	name
1	Single Player
2	Online Multiplayer
3	Massive Multiplayer Online
4	Local Multiplayer
5	Cross-Platform Multiplayer
6	Local Co-op
7	Online Co-op
8	Cross-Platform Co-op

Licenses table

SQL Create Statements:

CREATE TABLE IF NOT EXISTS Licenses (

lid SERIAL,

isBundle BOOLEAN,

PRIMARY KEY (lid)

);

Functional Dependencies:

lid →isBundle

Notes:

A table containing all the licenses in Mist. This table is extremely critical to the operation of Mist. Every single product in Mist will have a license to identify it. The Licenses table is a supertype of the games and bundles. However this is a mutually exclusive relationship. A License can only ever be either a game or a bundle, never both. The isBundle field, triggers, and procedures will be put in place to ensure this is the case.

Sample Data:

lid	isBundle
1	False
2	False
3	False
4	False
5	True

Games table

SQL Create Statements:

```
CREATE TABLE IF NOT EXISTS Games (
gid INT REFERENCES licenses(lid),
title VARCHAR(40),
priceUSD INT,
release_date DATETIME,
publisher REFERENCES publisher(pid),
developer REFERENCES developers(did),
genre REFERENCES genres(genre_id),
description VARCHAR(10000),
PRIMARY KEY (gid)
);
```

Functional Dependencies:

gid →title, priceUSD, release_date, publisher, developer, genre, description

Notes:

A table containing all the games available in Mist. This table is also critical to the database. Without this, the rest of the tables are useless. This table should never be inserted into directly. A stored procedure should be used to insert into the table due to the nature of the Licenses table. Triggers will be in place to ensure the integrity and correctness of this table.

Sample Data:

gid	Title	priceUSD	release_date	publisher	developer	genre	description
1	Portal	20	2007-10-09 00:00:00	2	2	1	Best Game Ever
2	Age of Empires 2: HD	20	2013-04-09 00:00:00	4	7	8	Remake of Great Game
3	Halo: CE	40	2001-11-15 00:00:00	4	6	2	Pistol OP
4	Assassin's Creed	40	2007-11-13 00:00:00	3	3	3	Downhill from here
6	DotA 2	0	2013-07-09 00:00:00	2	2	5	Better than LoL
7	League of Legends	0	2009-10-27 00:00:00	11	11	4	Better than DotA
8	Path of Exile	0	2013-10-23 00:00:00	15	15	7	Actual Diablo 3

Bundles table

SQL Create Statements:

```
CREATE TABLE IF NOT EXISTS Bundles (
bid INT REFERENCES licenses(lid),
name VARCHAR(40),
priceUSD INT,
PRIMARY KEY (bid)
);
```

Functional Dependencies:

bid → name, priceUSD

Notes:

A table containing the bundles being offered by Mist.

Sample Data:

bid	name	priceUSD
6	The First 5 Games in Mist!	20
30	Puzzle Pack	50
48	Adventure Time	40
70	Valve Complete Pack	100
85	Assassin's Creed Pack	80
97	Ensemble Pack	40
107	The First 100 Games in Mist!	200

BundleGames table

SQL Create Statements:

CREATE TABLE IF NOT EXISTS BundleGames (

gid INT REFERENCES games (gid),

bid INT REFERENCES bundles(bid),

PRIMARY KEY (gid, bid)

);

Functional Dependencies:

(gid, bid) →

Notes:

This table will associate which games are in what bundle.

Sample Data:

gid	bid
1	6
2	6
3	6
4	6
5	6
18	30
27	30

SupportedModes table

SQL Create Statements:

CREATE TABLE IF NOT EXISTS SupportedModes (

gid INT REFERENCES games(gid),

mid INT REFERENCES gamemodes(mid),

PRIMARY KEY (gid, mid)

);

Functional Dependencies:

gid, mid →

Notes:

A table to link the games to their supported game modes

Sample Data:

gid	mid
1	1
2	1
2	2
2	7
3	1
3	2
4	1

Accounts table

SQL Create Statements:

CREATE TABLE IF NOT EXISTS Accounts (

aid SERIAL,

name VARCHAR (40),

username VARCHAR(40),

password VARCHAR (100000),

PRIMARY KEY (aid)

);

Functional Dependencies:

aid → name, username, password

Notes:

A table containing all the accounts on Mist.

Sample Data:

aid	name	username	password (pretend these are hashes)
1	Alan	coddrulez	alpaca
2	Bob	mongosuxz	alpaca
3	Stan	webdevsuxs	alpaca
4	Antony	mongodev	alpaca
5	Miles	chalkmagnet	alpaca
6	James	007	alpaca

AccountLicenses table

SQL Create Statements:

CREATE TABLE IF NOT EXISTS AccountLicenses (

aid INT REFERENCES accounts(aid),

lid INT REFERENCES licenses(lid),

PRIMARY KEY (aid, lid)

);

Functional Dependencies:

(aid, lid) →

Notes:

A table of what licenses are owned by which accounts. This table is extremely important to keep consistent. It will only be inserted into when the Orders table is inserted into. A trigger will be in place to handle giving out the licenses included in the bundles. Users should only have access to the games they have paid for. Another important feature to note is that users are able to buy a bundle even if they own some of the games in the bundle already but they will be unable to accidentally buy games they have already bought through a bundle.

Sample Data:

aid	lid
1	1
1	2
3	3
3	7

Orders table

SQL Create Statements:

CREATE TABLE IF NOT EXISTS Orders (

oid SERIAL,

lid REFERENCES licenses(lid),

aid REFERENCES accounts(aid),

order_time DATETIME,

amount_paid INT,

cc_info VARCHAR(4),

PRIMARY KEY (oid)

);

Functional Dependencies:

oid → aid, lid, order_time, amount_paid, cc_info

Notes:

A table of all the orders placed in Mist. cc_info will only be the last four digits of the credit card used. amount_paid exists to keep track of how much the user paid, in the case of a sale or price reduction.

Sample Data:

oid	aid	lid	order_time	amount_paid	cc_info
1	1	1	2007-11-13 22:53:28	20	1337
2	1	2	2013-02-13 15:03:17	18	1337
3	3	3	2013-03-07 10:23:13	30	2112

Views

Here are some sample views that can be generated from the database and how they can be useful.

GamesInfo - Generates all the information on the games in Mist

```
CREATE VIEW GamesInfo
    (name, release_date, publisher, developer, genre) AS
    SELECT g.title, g.release_date, p.name, d.name, ge.name
    FROM games g, publishers p, developers d, genres ge
    WHERE g.publisher = p.pid AND g.developer = d.did AND g.genre = ge.genre_id
```

Example Use

```
SELECT *
FROM GamesInfo
WHERE name = 'Portal'
```

Output:

name	release_date	publisher	developer	genre
Portal	2007-10-09 00:00:00	Valve Corporation	Valve	Puzzle

AccountLibraries - Generates what games every account owns

```
CREATE VIEW AccountLibraries
        (account, account_name, game) AS
        SELECT a.aid, a.name, g.title
        FROM accounts a, accountLicenses al, games g
        WHERE a.aid = al.aid AND g.gid = al.lid
```

Example Use

```
SELECT account_name, game
FROM AccountLibraries
WHERE a.aid = 1
```

Output:

account_name	game
Alan	Portal
Alan	Age of Empires 2: HD

Reports

Top Ten Games

This report will allow Mist to display what the most popular games at the time are.

```
SELECT COUNT(game)
FROM AccountLibraries
GROUP BY game
ORDER BY COUNT(game) DESC
LIMIT 10
```

Last Month's Revenue

This report will generate the money made last month.

```
SELECT SUM(amount_paid) AS “Revenue”
FROM orders
WHERE EXTRACT(MONTH FROM order_time) = (EXTRACT(MONTH FROM CURRENT_TIME) - 1)
```

Stored Procedures

An example of an essential stored procedure would be the addGame function. The addGame procedure will be used whenever a new game is added instead of the usual insert statement. This will ensure License is properly updated and the database stays consistent. A similar function called addBundle would be used as well.

The function will first generate a new License ID. Then a variable is set to that newly generated LicenseID. That variable along with the parameters inputted will be used to generate the new row in the Games table. addBundle would be the almost exactly the same procedure except we would insert a TRUE instead of a FALSE into Licenses.isBundle.

```
CREATE FUNCTION addGame (gameName TEXT, priceUSD INT, release DATE, pub INT, dev INT, gen INT, descp TEXT)
RETURN void AS $$
DECLARE newid INTEGER;
BEGIN
        INSERT INTO Licenses(isBundle) VALUES (False);

        SELECT MAX(lid) INTO newid
        FROM Licenses;

        INSERT INTO games(gid, title, priceUSD, release_date, publisher, developer, genre, description)
        VALUES
        (newid, gameName, priceUSD, release, pub, dev, gen, descp);
END;
$$ LANGUAGE plpgsql
```

Triggers

This is an example of an essential trigger. It will fire whenever someone places an order. It checks if the product purchased was a bundle or not. If it is a bundle then it will add the licenses of all the games that are associated with that bundle.

```
CREATE FUNCTION addedBundleTrigger()
RETURNS trigger AS $$
DECLARE checkBool BOOLEAN;
        gameID INTEGER;
BEGIN
        SELECT l.isBundle INTO checkBool
        FROM licenses l
        WHERE l.lid = NEW.lid

        IF checkBool THEN
                FOR gameID IN (SELECT bg.gid FROM BundleGames bg WHERE bg.bid = NEW.lid) LOOP
                        INSERT INTO accountLicenses(aid, lid)
                        VALUES
                                (NEW.aid, gameID)
                END LOOP;
        END IF;

RETURN NULL;
```

```
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER addedBundleTrigger
        AFTER INSERT ON orders
        FOR EACH ROW
        EXECUTE PROCEDURE addedBundleTrigger();
```

Security

Unless explicitly stated otherwise, assume that a user group does not have any access to a table.

Employees

- Complete access to the Companies, Publishers, Developers tables
- Complete access to the bundleGames in order to add or remove games from bundles.
- Complete access to supportedModes table in order to add the supported game modes.
- Complete access to Accounts table
- Complete access to orders table
- Access to stored procedures

Customer Support

- Able to insert or delete into the orders table. Doing so will activate the triggers in place that will handle what games are available to the accounts.

Mist Account Owners

- Able to update their information for their accounts
- Able to submit new orders

Known Problems

- Adding game modes is clunky. Currently an Employee has to do it by hand.
- Same problem with BundleGames.
- A game can only have one developer.
- Developers and Publishers tables are rather bare.

Future Enchantments

- Implementation for Sales. Perhaps another field called salePrice in the games and bundle?
- Friends List for accounts. General improvements to community capability of Mist.
- Chat Log system
- Gaming Related News table