Information Management II Project

Stephen Rowe – ID 14319662

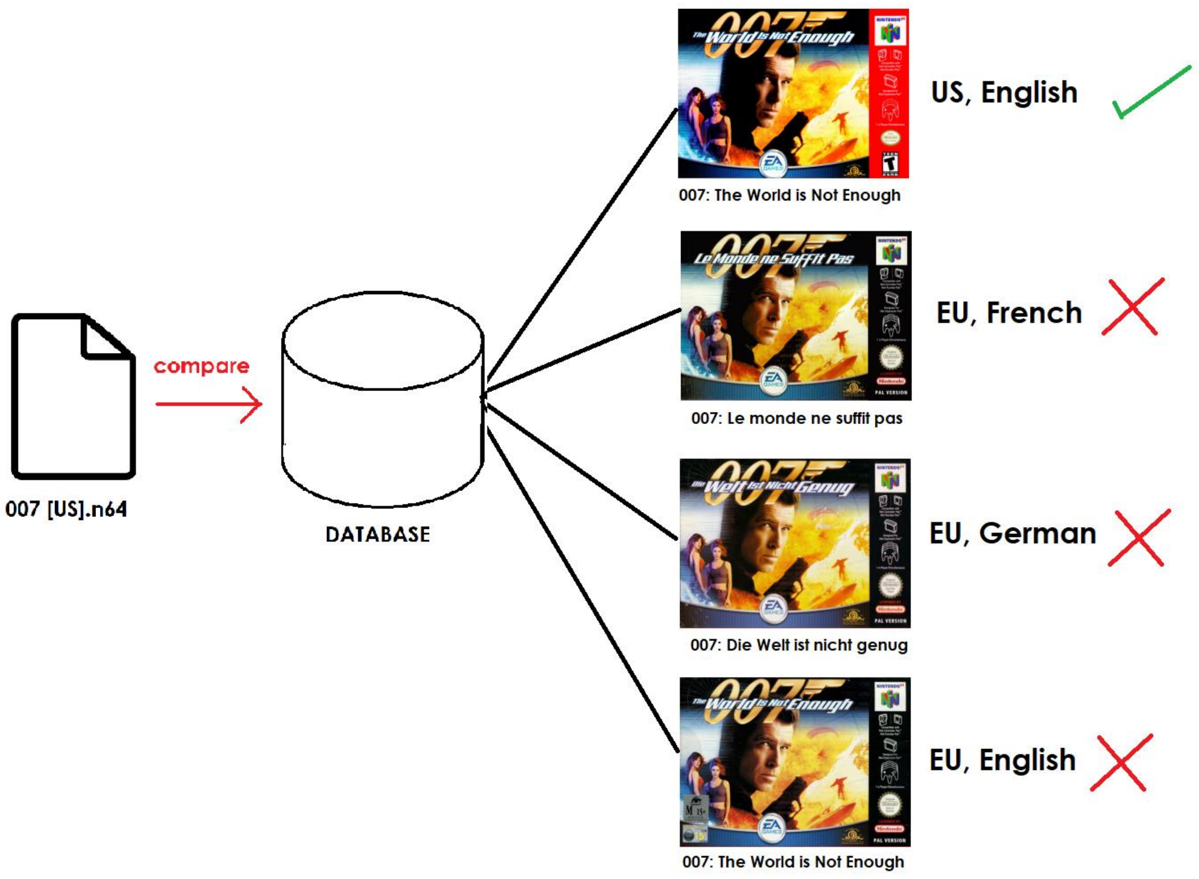
Introduction

For this MySQL project, I decided to make a database for Video Game ROMs, where a user has their own personal collection of ROM files on their own computer, and this database associates all the real-world information about this Game and ROM (i.e. Game Title, Region, Language, Box Art, Developer, Release Date, etc.).

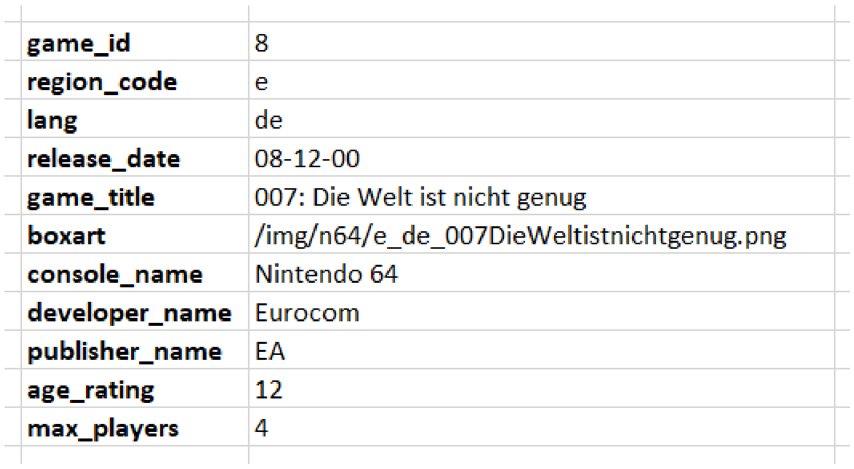
Let's say we have a game, '007: The World is Not Enough' for Nintendo 64. This game is released in United States and Europe. In Europe, there are multiple versions of this game in different languages. All of these versions would have significant differences between them (e.g. the US box-art would be different from UK box art, the French and UK versions would have different titles and box-arts with language differences). We now have an issue with balancing languages and regions of the game releases for providing useful data about the game. We also have a situation where there could be a massive number of different ROM files circulating online for the same regional version of a game, e.g.

- *007 [US].**n64***
- *The_World_Is_Not_Enough.**z64***
- *007: The World Is Not Enough (BETA) (U).**n64***

These are all the US Version of the game but are different files (different file-types and produce different SHA1 results). So now we need to extract the region and language from the ROM and associate it with the exact regional version of the game in question.


compare
007 [US].n64
DATABASE
007: The World is Not Enough
US, English
007: Le monde ne suffit pas
EU, French
007: Die Welt ist nicht genug
EU, German
007: The World is Not Enough
EU, English

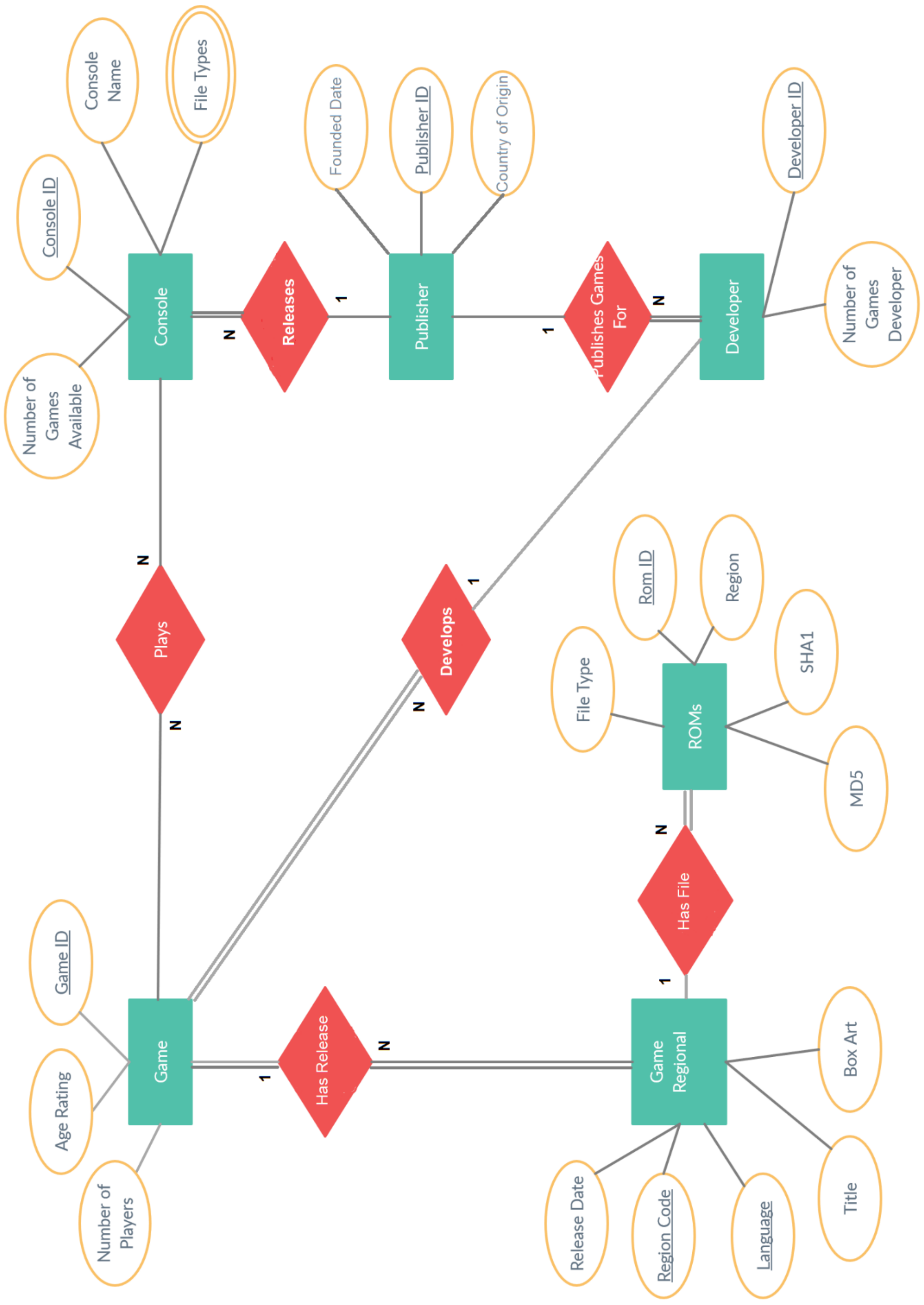
The user will match the MD5 and SHA1 strings derived from their own file and compared with the list of ROMs in our database, and with this we will be able to provide an ample amount of useful information about the ROM and game to the user, which they could use in a ROM browser application:



game_id	8
region_code	e
lang	de
release_date	08-12-00
game_title	007: Die Welt ist nicht genug
boxart	/img/n64/e_de_007DieWeltistnichtgenug.png
console_name	Nintendo 64
developer_name	Eurocom
publisher_name	EA
age_rating	12
max_players	4

The Project started with 6 fundamental Entities: **Publisher, Developer, Console, Game, Game Regional Version, ROM**. The relationships between these entities can be seen in the ER diagram on the next page:

ER Diagram


Console ID
Console Name
File Types
Number of Games Available
Console
Releases
Publisher
Founded Date
Publisher ID
Country of Origin
Publishes Games For
Developer
Developer ID
Number of Games Developer
Plays
Develops
Game
Game ID
Age Rating
Number of Players
Has Release
Game Regional
Release Date
Region Code
Language
Title
Box Art
Has File
ROMs
Rom ID
Region
File Type
SHA1
MD5
N
1

Publisher is here is serves two purposes in our database: Parent company/distributor for a Developer, and publisher of Consoles. For example:

- EA is the parent company of Eurocom but does not create any consoles. Publisher will have an entry for EA (with *founded date, Country of Origin,* etc.), Developer will have an entry for Eurocom (with a foreign key pointing to EA), but there are no Console in the Console table associated with EA.
- Sony is the parent company of Insomniac Games. Sony also creates consoles but is not a game developer in itself. There will be an entry for Sony in Publisher, and an entry for Insomniac Games in Developer, linking it to Sony. Consoles such as the PlayStation will also have this same Sony *Publisher* entry as its publisher through a foreign key.
- Nintendo is all of the above; they create consoles, develop games, and distribute the games. Therefore, there will be an Entry for Nintendo in Publisher and there will be an entry in Developer for Nintendo. So, there is a foreign key from *Developer* Nintendo to *Publisher* Nintendo.

Developer keeps track of the number of games released by a Developer (games themselves, not regional versions of the games), and will be referenced to by a foreign key in the Games relation.

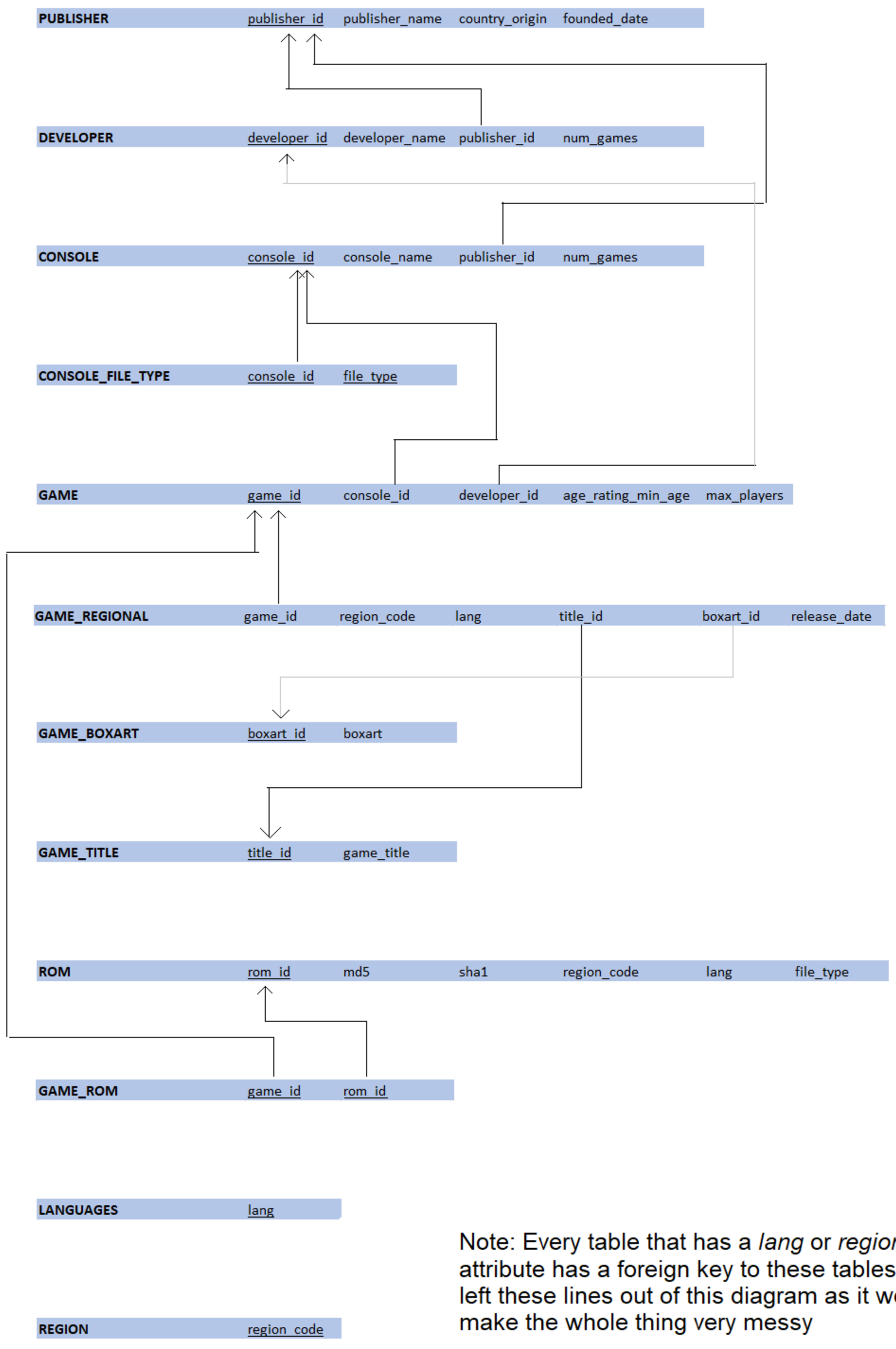
Console similarly keeps track of the number of games released for it, but also has the multi-valued attribute *File Types*. For a given console, there are a very limited number of ROM file-types associated with it, e.g. *'.n64'* and *'.z64'* are the only filetypes in the wild for Nintendo 64 games. It wouldn't make sense to allow a file-type of *'.jpg´* as an entry for a ROM for a game for this console, so keeping track of this will allow us to restrict invalid entries.

Game refers to the very base attributes of a game, that are consistent regardless of whatever regions or languages the game is released in. This includes its ID in the database, number of players, and age rating.

Game Regional will keep track of all the regional releases of a given game. Its composite primary key is Game ID, Region Code, and Language. This also stores the Title and Box Art for a given regional version.

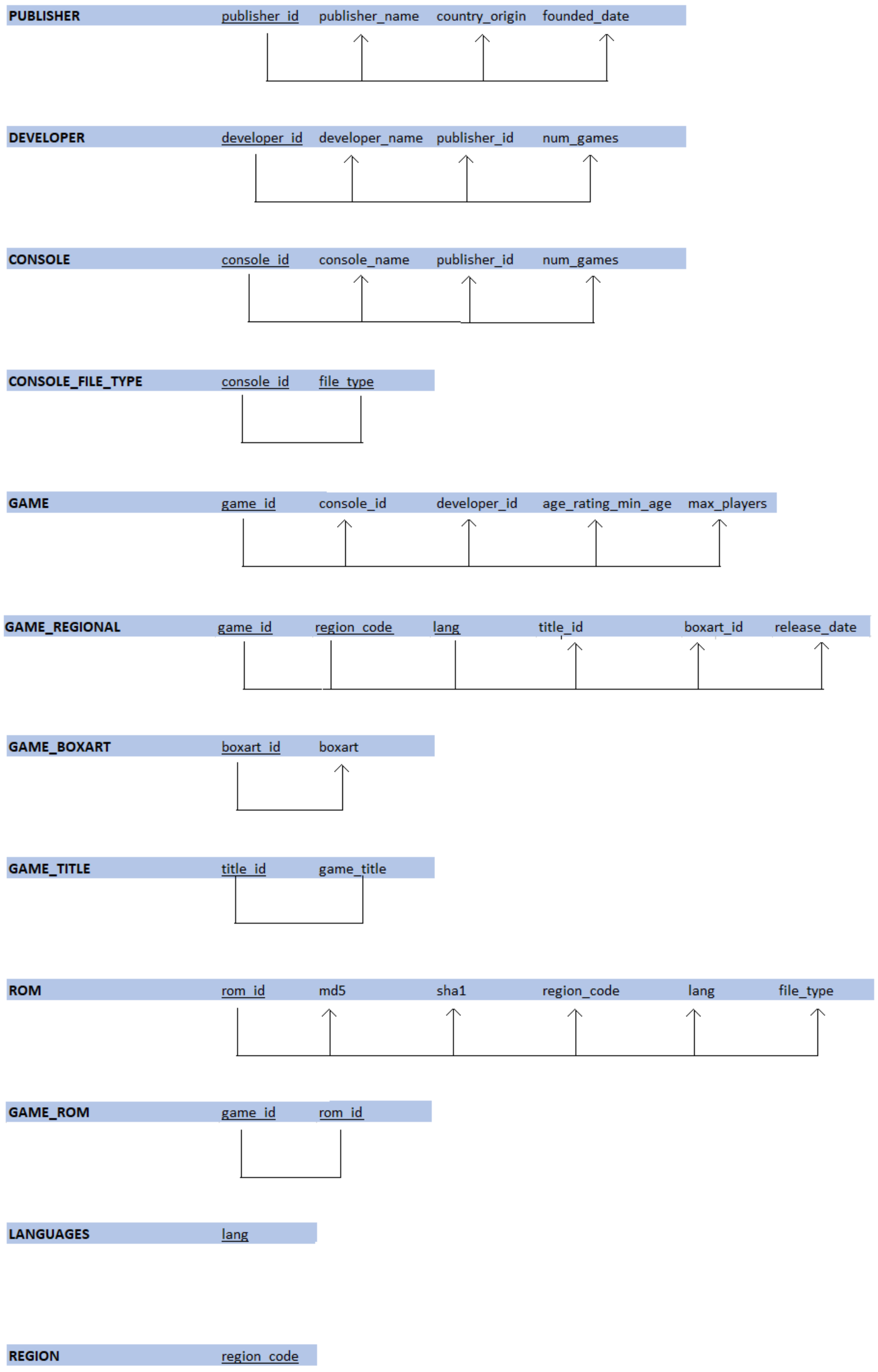
ROM contains the MD5, SHA1, file type, region, and language of a rom. Its primary key is the ROM ID.

Mapping to Relational Schema


PUBLISHER publisher_id publisher_name country_origin founded_date
DEVELOPER developer_id developer_name publisher_id num_games
CONSOLE console_id console_name publisher_id num_games
CONSOLE_FILE_TYPE console_id file_type
GAME game_id console_id developer_id age_rating_min_age max_players
GAME_REGIONAL game_id region_code lang title_id boxart_id release_date
GAME_BOXART boxart_id boxart
GAME_TITLE title_id game_title
ROM rom_id md5 sha1 region_code lang file_type
GAME_ROM game_id rom_id
LANGUAGES lang
REGION region_code

Note: Every table that has a *lang* or *region_code* attribute has a foreign key to these tables, but I left these lines out of this diagram as it would make the whole thing very messy

Functional Dependencies


PUBLISHER publisher_id publisher_name country_origin founded_date
DEVELOPER developer_id developer_name publisher_id num_games
CONSOLE console_id console_name publisher_id num_games
CONSOLE_FILE_TYPE console_id file_type
GAME game_id console_id developer_id age_rating_min_age max_players
GAME_REGIONAL game_id region_code lang title_id boxart_id release_date
GAME_BOXART boxart_id boxart
GAME_TITLE title_id game_title
ROM rom_id md5 sha1 region_code lang file_type
GAME_ROM game_id rom_id
LANGUAGES lang
REGION region_code

1st Normal Form

The multi-valued attribute File-Type in Console became its own table.

2nd Normal Form

For the tables with one primary key, proving that the table obeys 2nd is straight-forward, as each of the non-prime keys in these tables are functionally dependent on the primary keys in all the above cases.

Game Regional is the only table with a multi-valued primary key. All the non-prime attributes (title_id, boxart_id, release_date) are functionally dependent on all three primary keys (as a game can be released at different dates in the same region for different languages), and not a proper subset of them; if we omit any of these candidate keys, then the table won't make sense. To reduce redundancy, I created two new tables, Game_Title and Game_Boxart. This is so that if multiple regions or multiple languages use the exact same title (e.g. 'Super Mario Bros.' has no alternate names regardless of region/language), or uses the exact same box-art, then this is only stored once in these tables and referenced with a foreign key. For the values of Box Art, I decided to just use a file directory string, but in practice this would likely be a URL.

3rd Normal Form

For each of the tables above, if we take the primary keys, and pick any one non-prime (treating the rest of the non-prime attributes as if they are not there at all), then no matter what attribute we pick, they are fully functionally dependent on the whole key.

Schema

Note: If you want to run my SQL files in a database, the order the files should be fully run is

1. Schema
2. Triggers
3. Views
4. Insert

```
drop database game_project_14319662;
create database game_project_14319662;
use game_project_14319662;
create table Publisher (
    publisher_id int auto_increment primary key,
    publisher_name varchar(32) not null,
    country_origin varchar(32) not null,
    founded_date date not null
);

create table Developer (
    developer_id int auto_increment primary key,
    developer_name varchar(32),
    publisher_id int not null,
    num_games int not null default 0
);
```

```
-- Showing an Alter Statement
ALTER TABLE Developer ADD
FOREIGN KEY (publisher_id)
REFERENCES Publisher (publisher_id)
ON DELETE RESTRICT;

With ROMs, there is a convention wrt. Regions. J, E, U, are Japan, Europe, North America respectively.
For this project, I will stick with using only J, E, U, A, S (A is Australia and S is South America),
as this is enough for demonstration
create table Region (
    region_code char not null primary key
);

-- Will stick to 5 languages for this demonstration - en, jp, fr, de, es
create table Languages (
    lang char(2) not null primary key
);

create table Console (
    console_id int auto_increment primary key,
    console_name varchar(30) not null,
    publisher_id int not null,
    foreign key (publisher_id) references Publisher (publisher_id) ON DELETE RESTRICT,
    num_games int not null default 0
);

create table Console_File_Type (
    console_id int,
    foreign key (console_id) references Console (console_id) ON DELETE RESTRICT,
    file_type varchar(10),
    primary key (console_id, file_type)
);

create table Game (
    game_id int auto_increment primary key,
    console_id int not null,
    foreign key (console_id) references Console (console_id) ON DELETE RESTRICT,
    developer_id int not null,
    foreign key (developer_id) references Developer (developer_id) ON DELETE RESTRICT,
    age_rating_min_age int not null default 3,
    max_players int not null default 1
);

The following two tables were created to eliminate data redundancy in the table
after, Game_Regional, as the UK and US versions of the game may have the same title (although
for retro games, this was much less common than today). Also, associating Game_Title with Game_BoxArt
would create a massive amount of redundancy
```

```
as in general, Game_BoxArt would be vary for every region and language, but Game_Title would not vary
to the same extent. We can reuse the same tuple
from this table for the EU English and US English Regional Versions, if they are the same

- Unique constraints on these two tables, as the purpose of these is to eliminate redundancy
create table Game_Title (
    title_id int auto_increment primary key,
    game_title varchar(50) not null unique          -
);

create table Game_BoxArt (
    boxart_id int auto_increment primary key,
    boxart varchar(255) not null unique
);

Each game has a different version for each region and language. Each tuple on this table signifies the
region it is released in (EU, US, etc.) and the language of this version. So for Europe, we have a Eng
lish, a French and a Spanish tuple. We also have EU English and US English tuples, due to the
difference in box-arts/titles. The tables following this one break the table down into NF and
eliminate redundancy.
create table Game_Regional (
    game_id int,
    foreign Key (game_id) references Game (game_id) ON DELETE CASCADE,
    region_code char,
    foreign Key (region_code) references Region (region_code) ON DELETE RESTRICT,
    lang char(2),
    foreign key (lang) references Languages (lang) ON DELETE RESTRICT,
    primary key(game_id, region_code, lang),
    title_id int,
    foreign Key (title_id) references Game_Title (title_id) ON UPDATE CASCADE ON DELETE RESTRICT,
    boxart_id int,
    foreign key (boxart_id) references Game_BoxArt (boxart_id) ON UPDATE CASCADE ON DELETE RESTRICT,
    release_date date not null
);

create table Rom (
    rom_id int not null auto_increment primary key,
    md5 char(32) not null unique,
    sha1 char(40) not null unique,
    region_code char not null,
    foreign key (region_code) references Region (region_code) ON DELETE RESTRICT,   -
- ROMs are specified with a region, e.g. [U] for US, [E] for Europe, etc.
    lang char(2) not null,
    foreign key (lang) references Languages (lang) ON DELETE RESTRICT,
    file_type varchar(10) not null
);
```

```
matches rom to a game (adding game_id to Rom would require it to be a primary key, but this would brea
k 2NF since md5 and sha1 is not dependendant on game_id)
CREATE TABLE Game_Rom (
    game_id int,
    foreign key (game_id) references Game (game_id) ON DELETE CASCADE,
    rom_id int,
    foreign Key (rom_id) references Rom (rom_id) ON DELETE CASCADE,
    primary key(game_id, rom_id)
);
```

Views

```
This View makes the Game_ROM table more useful, showing the full contents of the rows associated with
its two foreign keys
CREATE OR REPLACE VIEW Game_Rom_View AS
SELECT Game.*, Rom.* FROM Game_Rom
INNER JOIN Rom ON Game_Rom.rom_id = Rom.rom_id
INNER JOIN Game ON Game_Rom.game_id = Game.game_id
WITH CHECK OPTION;

select * from Game_Rom_View;

CREATE OR REPLACE VIEW Insert_Rom_View AS
SELECT GR.game_id, R.rom_id FROM Game_Rom AS GR
INNER JOIN Rom AS R ON GR.rom_id = R.rom_id;

select * from Insert_Rom_View;

Finds all Games, and all the Regional Versions of those games, accompanied with the metadata (title,
boxart) associated with that Regional Version
CREATE OR REPLACE VIEW Game_All_Regions_View AS
SELECT G.game_id, GR.region_code, GR.lang, GR.release_date, T.game_title, B.boxart, C.console_name, D.
developer_name, P.publisher_name, G.age_rating_min_age AS age_rating, G.max_players
FROM Game_Regional AS GR
INNER JOIN Game AS G ON GR.game_id = G.game_id
INNER JOIN Game_Title AS T ON GR.title_id = T.title_id
INNER JOIN Game_Boxart AS B ON GR.boxart_id = B.boxart_id
INNER JOIN Console AS C ON G.console_id = C.console_id
INNER JOIN Developer AS D ON G.developer_id = D.developer_id
INNER JOIN Publisher As P On D.publisher_id = P.publisher_id;

SELECT * FROM Game_All_Regions_View;
```

This next table isn't really part of my schema, its just a test table to demonstrate that the MD5 searching is working as intended. This is five ROM files picked at random, showing we can uniquely identify and provide ample information about the ROM/Game using just the MD5 value.

```
create table My_ROMs ( md5 varchar(32) not null );
INSERT INTO My_ROMs VALUES ('3279ACEED4663F9584689C0036776B8B');
INSERT INTO My_ROMs VALUES ('18FB2CF8B58C144CD12BC035755CBC12');
INSERT INTO My_ROMs VALUES ('F462C6B1098A890A225106AA5F2A1B20');
INSERT INTO My_ROMs VALUES ('0D4F4DE68E2C9DD76076F12098BC1874');
INSERT INTO My_ROMs VALUES ('0DB0E82831555607EE9830612106DBAB');
```

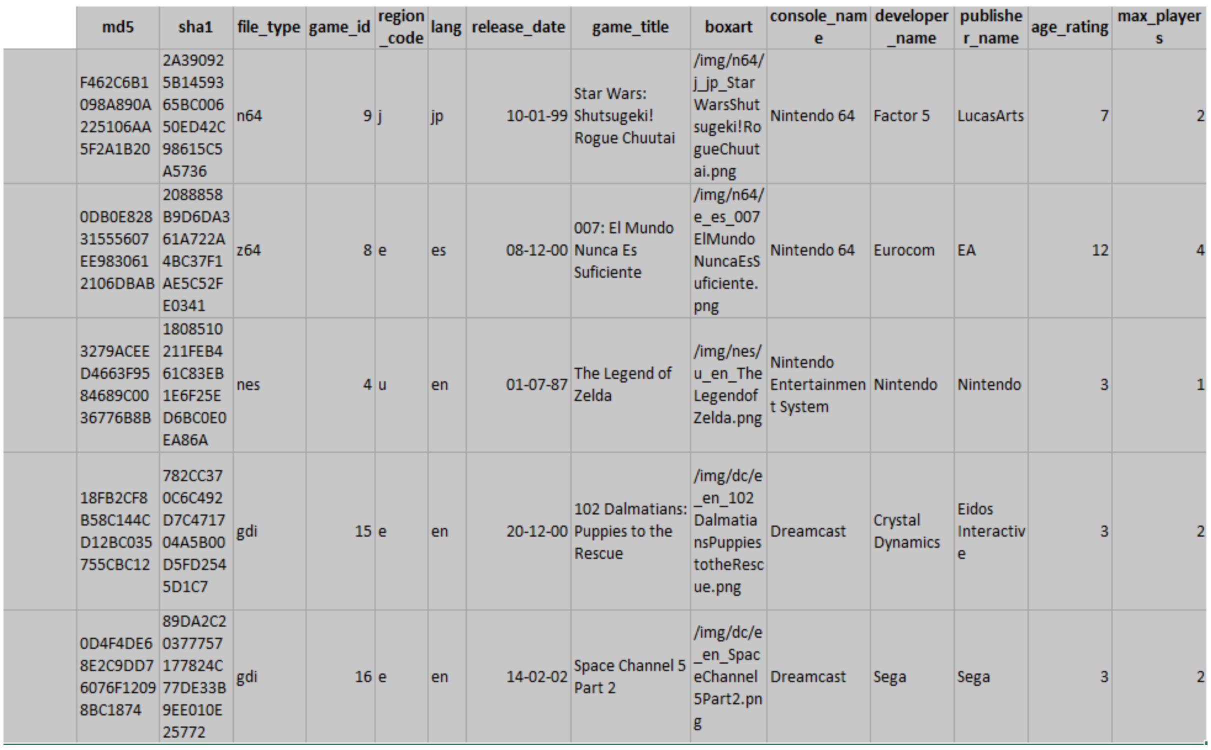
Search for roms by their MD5 and/or SHA1, delivering all relevant metadata for this ROM

```
CREATE OR REPLACE VIEW MD5_SHA1_Search_View AS
SELECT GRV.md5, GRV.sha1, GRV.file_type, GAR.* FROM Game_All_Regions_View AS GAR
INNER JOIN Game_Rom_View AS GRV ON
    GRV.game_id = GAR.game_id AND GRV.region_code = GAR.region_code AND GRV.lang = GAR.lang;
```

Demonstrating using MD5_SHA1_Search_View with the My_ROMs table

```
SELECT Results.* FROM MD5_SHA1_Search_View AS Results
INNER JOIN My_ROMs ON Results.md5 = My_ROMs.md5;
```

Result from this table, we can see it accurately associates the all the meta-data, especially region and language, using just the MD5 value:



	md5	sha1	file_type	game_id	region_code	lang	release_date	game_title	boxart	console_name	developer_name	publisher_name	age_rating	max_players
	F462C6B1098A890A225106AA5F2A1B20	2A390925B1459365BC00650ED42C98615C5A5736	n64	9	j	jp	10-01-99	Star Wars: Shutsugeki! Rogue Chuutai	/img/n64/j_jp_StarWarsShutsugeki!RogueChuutai.png	Nintendo 64	Factor 5	LucasArts	7	2
	0DB0E82831555607EE9830612106DBAB	2088858B9D6DA361A722A4BC37F1AE5C52FE0341	z64	8	e	es	08-12-00	007: El Mundo Nunca Es Suficiente	/img/n64/e_es_007ElMundoNuncaEsSuficiente.png	Nintendo 64	Eurocom	EA	12	4
	3279ACEED4663F9584689C0036776B8B	1808510211FEB461C83EB1E6F25ED6BC0E0EA86A	nes	4	u	en	01-07-87	The Legend of Zelda	/img/nes/u_en_TheLegendofZelda.png	Nintendo Entertainment System	Nintendo	Nintendo	3	1
	18FB2CF8B58C144CD12BC035755CBC12	782CC370C6C492D7C471704A5B00D5FD2545D1C7	gdi	15	e	en	20-12-00	102 Dalmatians: Puppies to the Rescue	/img/dc/e_en_102DalmatiansPuppiestotheRescue.png	Dreamcast	Crystal Dynamics	Eidos Interactive	3	2
	0D4F4DE68E2C9DD76076F12098BC1874	89DA2C20377757177824C77DE33B9EE010E25772	gdi	16	e	en	14-02-02	Space Channel 5 Part 2	/img/dc/e_en_SpaceChannel5Part2.png	Dreamcast	Sega	Sega	3	2

```sql
CREATE OR REPLACE VIEW Publisher_Developer_View AS
SELECT d.developer_id, d.developer_name, p.publisher_id, p.publisher_name, p.country_origin, p.founded
_date
FROM Developer AS D
INNER JOIN Publisher AS P ON D.publisher_id = P.publisher_id;

SELECT * FROM Publisher_Developer_View;
```

Triggers

```sql
-- This trigger is necessary, as there is no way to DELETE CASCADE on ROM as it has no foreign keys
 DELIMITER //
CREATE TRIGGER delete_rom_entirely
    AFTER DELETE ON Game_ROM
    FOR EACH ROW
    BEGIN
        DELETE FROM ROM WHERE ROM.rom_id = OLD.rom_id;
    END //
DELIMITER ;

The following is a procedure I wrote that allows anyone that has been granted access to this procedure
to add ROMs to any game in the database. With ‘SQL SECURITY DEFINER’, this means that only grant
access to this procedure is required to be able to use it, users don’t require general INSERT
permissions.
-- drop procedure insert_new_rom;
DELIMITER //
CREATE PROCEDURE insert_new_rom (IN game_id int, IN md5 char(32), IN sha1 char(40),
                                 IN region_code char, IN lang char(2), file_type varchar(10))
COMMENT 'Allows users to submit ROMs, inserting them in the ROM and GAME_ROM tables'
LANGUAGE SQL
MODIFIES SQL DATA
SQL SECURITY DEFINER
BEGIN
    -- If file_type of ROM is not valid for this console, abort without modifying anything
    IF NOT EXISTS (SELECT * FROM Console_File_Type AS CFT
                  INNER JOIN Game
                  WHERE CFT.console_id = Game.console_id AND CFT.file_type = file_type)
    THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'INVALID FILETYPE';
    END IF;

    IF EXISTS (SELECT * FROM Game WHERE Game.game_id = game_id)
    THEN
        INSERT INTO ROM VALUES (null, md5, sha1, region_code, lang, file_type);
        SET @new_ROM_id = (SELECT rom_id FROM ROM WHERE ROM.md5 = md5 AND ROM.sha1 = sha1);
        INSERT INTO Game_ROM VALUES (game_id, @new_ROM_id);
    ELSE SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'GAME_ID NOT PRESENT IN DATABASE';
    END IF;
END //
```

```
 DELIMITER ;

 DELIMITER //
CREATE TRIGGER increment_num_games
    AFTER INSERT ON Game
    FOR EACH ROW
    BEGIN
     UPDATE Console SET num_games = num_games+1 WHERE (NEW.Console_id = Console.console_id);
     UPDATE Developer SET num_games = num_games+1 WHERE (NEW.developer_id = Developer.developer_id);
    END //
 DELIMITER ;

 DELIMITER //
CREATE TRIGGER decrement_num_games
    BEFORE DELETE ON Game
    FOR EACH ROW
    BEGIN
     UPDATE Console SET num_games = num_games-1 WHERE (OLD.Console_id = Console.console_id);
     UPDATE Developer SET num_games = num_games-1 WHERE (OLD.developer_id = Developer.developer_id);
    END //
 DELIMITER ;
```

Inserting in Database

I won't include the data for inserting values into the tables here, as the file is too long. I will upload it with my project in the file **INSERT.SQL**.

Security

The Security of this database has to be built around the idea that this database would be a collaborative project to compile a large list of ROMs and provide useful metadata for them all. So, I create three roles, 'contributor_user', 'read_only_user', and 'moderator'. Contributors are allowed to add ROMs and associate them with an (already created) Game with the INSERT_NEW_ROM procedure. Read-Only Users don't have this privilege, but they are still able to SELECT anything from the tables. Moderators are highly trusted and can modify any of the tables on the database. However, with the selective use of ON UPDATE/DELETE CASCADE/RESTRICT in the schema, I have severely limited the amount of data they could remove accidentally. I did not give Moderators the ability to modify the schema itself, add tables, etc.

```
CREATE ROLE IF NOT EXISTS 'contributor_user', 'read_only_user', 'moderator';

-- Any user can SELECT any information from the database
GRANT SELECT ON game_project_14319662.* TO 'contributor_user', 'read_only_user', 'moderator';

-- Only moderators and contributor_users can add ROMs to database
GRANT EXECUTE ON PROCEDURE insert_new_rom TO 'moderator', 'contributor_user';

-- Moderators can delete, insert and update any values on any tables on the database,
-- but cannot delete tables
```

```sql
GRANT INSERT ON game_project_14319662.* TO 'moderator';
GRANT UPDATE ON game_project_14319662.* TO 'moderator';
GRANT DELETE ON game_project_14319662.* TO 'moderator';

CREATE USER 'Tom'@'localhost';
CREATE USER 'Dick'@'localhost';
CREATE USER 'Harry'@'localhost';

CREATE USER 'MOD1'@'localhost';
CREATE USER 'MOD2'@'localhost';
CREATE USER 'MOD3'@'localhost';

GRANT 'contributor_user' TO 'Tom'@'localhost', 'Dick'@'localhost';
GRANT 'read_only_user' TO 'Harry'@'localhost';
GRANT 'moderator' TO 'MOD1'@'localhost', 'MOD2'@'localhost', 'MOD3'@'localhost';

REVOKE 'moderator' FROM 'MOD3'@'localhost';

SET DEFAULT ROLE 'contributor_user' TO
  'Tom'@'localhost',
  'Dick'@'localhost';

SET DEFAULT ROLE 'read_only_user' TO
  'Harry'@'localhost';

SET DEFAULT ROLE 'moderator' TO
  'MOD1'@'localhost',
  'MOD2'@'localhost';
```

Constraints

Outside of Primary Key/Foreign Key, NOT NULL and UNIQUE constraints, the primary example of a Semantic Constraint in my table is the implementation of the Console_File_Type table. This constraint is enforced within the INSERT_NEW_ROM trigger, which will throw an error if the user tries to enter a ROM file-type that isn't on the list of pre-approved list of file-types for the Console that this game is for.

Also, by creating the 'Languages' and 'Region' tables, with entries defined by the moderators/admin and using these two-letter values as foreign keys, then we can ensure that regions/languages inputted are in the correct format.

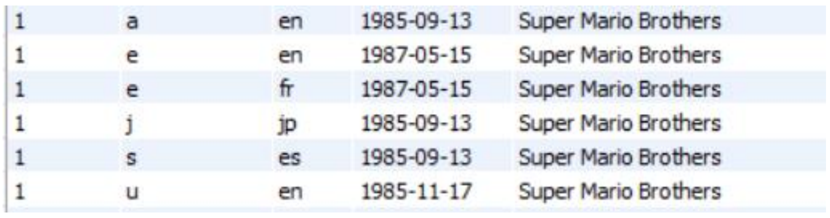
With respect to UPDATE/DELETE CASCADE/RESTRICT, I generally used ON DELETE RESTRICT when a column of a table was a foreign key to an auto-incremented int primary ID key of another table, as it wouldn't make sense that these int ID values themselves would be updated. If, for example, a console was deleted, this would be restricted if that console has any games in the database. If a moderator intended to delete a console, they would be expected to delete the games and related filetypes first.

Update

An example of an update I used in the database is:

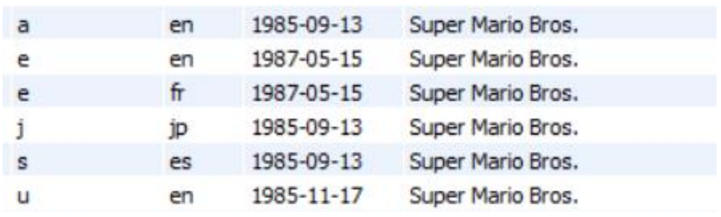
```
INSERT INTO Game_Title SET game_title = "Super Mario Brothers";


UPDATE Game_Title SET game_title = "Super Mario Bros." WHERE title_id = 1;
```

If we omit the above update line and select the Game_All_Regions_View, we get this result.



1	a	en	1985-09-13	Super Mario Brothers
1	e	en	1987-05-15	Super Mario Brothers
1	e	fr	1987-05-15	Super Mario Brothers
1	j	jp	1985-09-13	Super Mario Brothers
1	s	es	1985-09-13	Super Mario Brothers
1	u	en	1985-11-17	Super Mario Brothers

However, after we use the UPDATE line, we will see this:



a	en	1985-09-13	Super Mario Bros.
e	en	1987-05-15	Super Mario Bros.
e	fr	1987-05-15	Super Mario Bros.
j	jp	1985-09-13	Super Mario Bros.
s	es	1985-09-13	Super Mario Bros.
u	en	1985-11-17	Super Mario Bros.

Showing that update of this one row in Game_Title has propagated successfully.