University of British Columbia, Department of Computer Science

CPSC 304

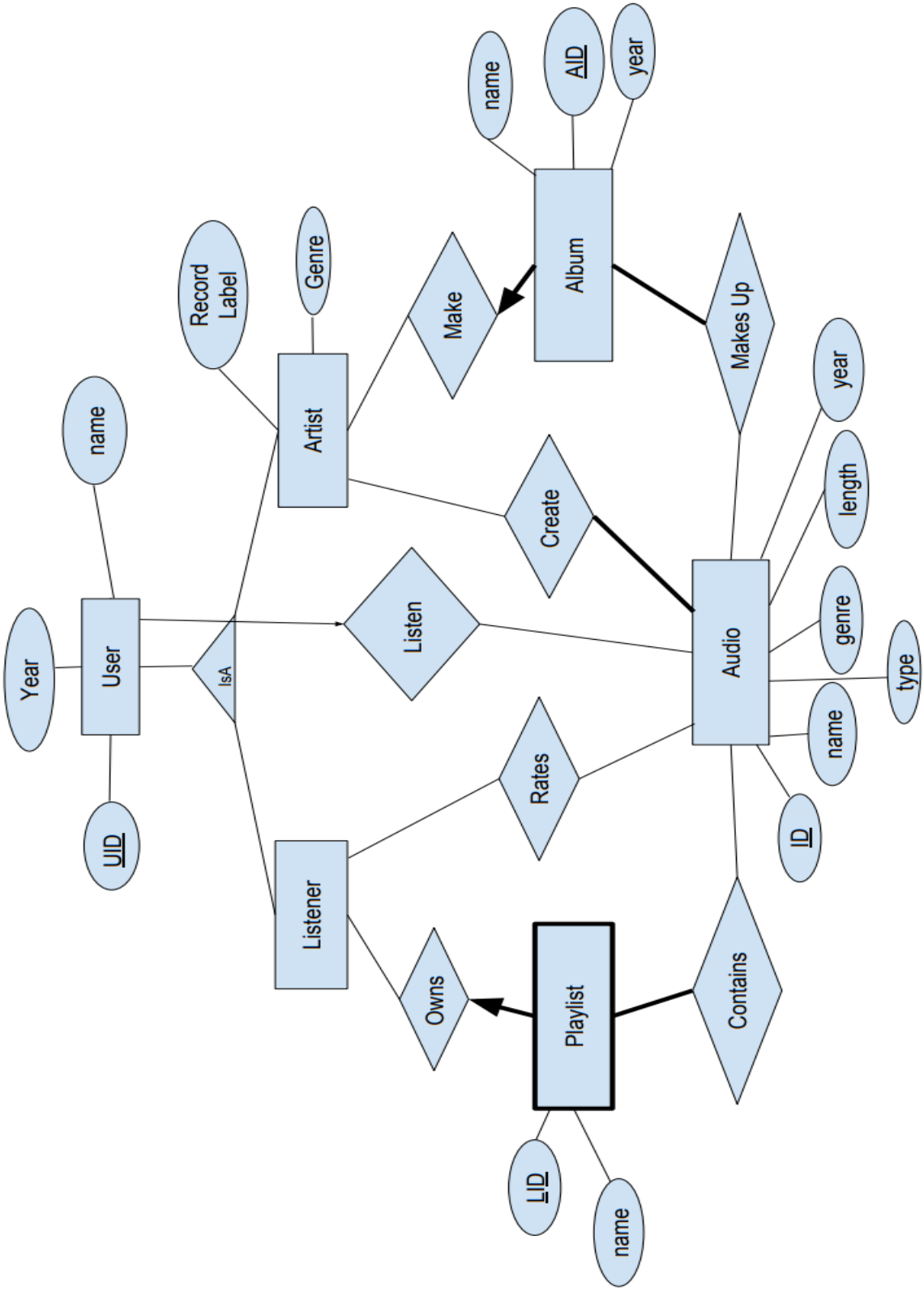
Summer 2018

Project Part 2

Group Members:

Name	Student Number	Unix ID	Tutorial Section	Email Address
Bijan Karrobi	**44307156**	**z8j0b**	**T1C**	**bijankarrobi@gmail.com**
Shaun Yuson	**20953162**	**t3a1b**	**T1E**	**shaunyuson@gmail.com**
Terence Chen	**42602136**	**w2b9**	**T1F**	**tcbc@alumni.ubc.ca**
Jonathan Fleming	**35659135**	**o7d9**	**T1F**	**jfleming@alumni.ubc.ca**

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above.
In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia


name
AID
year
Album
Make
Makes Up
Record Label
Genre
Artist
Create
name
Year
User
UID
IsA
Listen
Audio
year
length
genre
type
name
ID
Rates
Listener
Owns
Playlist
Contains
LID
name

Generated IDs:
Users / UID - 1xxx
Audio / ID - 2xxx
Playlists / LID - 3xxx
Albums / AID - 4xxx

```
CREATE TABLE User                    // has a total disjoint ISA bt. Artist and Listener
    ( UID INTEGER,
    name CHAR(20),                   // user's name
    year INT(4),                     // birth year
    ID INTEGER,                      // represents user listening to song
                                     // if null, user is not currently listening to
                                     // a song (Listens relationship)
    PRIMARY KEY (UID),
    FOREIGN KEY (ID) REFERENCES Audio(ID) )
```

UID	name	year	ID
1000	Shaun	1987	2000
1234	Terence	1990	2018
1672	Bijan	1991	2221
1023	Rihanna	1988	2213
1111	John	1970	2345
1999	Jonathan	1993	null
1456	Jessica	1990	null
1413	Stacy	1980	2000
1612	Sam	1996	2213
1025	Tom	1996	null

```
CREATE TABLE Listener                // Users who listen and are not artists
    ( UID INTEGER,
    PRIMARY KEY (UID),
    FOREIGN KEY (UID) REFERENCES User(UID) )
```

UID
1000
1234
1672
1999
1456

```
CREATE TABLE Artist                    // Users who are artists
        ( UID INTEGER,
        RecordLabel CHAR(20),
        genre CHAR(20),                // Usual genre
        PRIMARY KEY (UID),
        FOREIGN KEY (UID) REFERENCES User (UID))
```

UID	RecordLabel	genre
1023	Universal Music Group	Pop
1111	J Label	Rock
1413	S label	Pop
1612	null	Educational
1025	null	Educational

```
CREATE TABLE Audio                     // Audio Entity
        ( ID INTEGER,
        name CHAR(20),                 // title
        year INTEGER(4),               // year audio is released
        genre CHAR(20),                // genre
        length INTEGER,                // in seconds
        type CHAR(1),                  // p if podcast, s if song
        AID INTEGER,                   // Album song is a part of
        PRIMARY KEY (ID),
        FOREIGN KEY (AID) REFERENCES Album(AID) )
```

ID	name	year	genre	length	type	AID
2000	Umbrella	2007	Pop	90	s	4000
2018	John's Rock Song	2001	Rock	75	s	4100
2221	Stacy's song	2010	Pop	120	s	4132
2213	Sam and Tom's Science Podcast	2010	Educational	800	p	null
2345	Sam's Podcast	2010	null	1000	p	null

```
CREATE TABLE Album                                  // Album entity
        ( AID INTEGER,
        name CHAR(20),
        year INTEGER,
        UID INTEGER not NULL,                       // Album needs an Artist that created it
        PRIMARY KEY (AID)
        FOREIGN KEY (UID) REFERENCES Artist(UID) )
```

AID	name	year	UID
4000	Good Girl Gone Bad	2007	1023
4100	John's album	2001	1111
4132	Stacy's album	2010	1413
4444	Anti	2016	1023
4321	John's 2nd album	2008	1111

Note: Album 4444 and 4321 do not have songs, and therefore can not exist. But for now they are used as an example.

```
CREATE TABLE Playlist                                   // Playlist entity
        ( LID INTEGER,
        name CHAR(20),                                  // name of playlist
        UID INTEGER not NULL,                   // Playlist needs a Listener who created it
        PRIMARY KEY (UID, LID) // weak entity
        FOREIGN KEY (UID) REFERENCES Listener(UID)
                ON DELETE CASCADE
                ON UPDATE CASCADE )
```

LID	name	UID
3000	Workout	1000
3010	Running	1234
3100	Party	1672
3333	Sad	1999
3456	Studying	1456

```
CREATE TABLE Create                     // Relationship where Artist creates Audio
                                        // allows for multiple artists to create one song
        ( UID INTEGER,
        ID INTEGER,
        Primary Key (UID, ID),
        Foreign Key (UID) references Artist(UID),
        Foreign Key (ID) references Audio(ID) )
```

UID	ID
1023	2000
1111	2018
1413	2221
1612	2213
1025	2213
1612	2345

```
CREATE TABLE Contains                    // Relationship where Playlist has Audio
        ( LID INTEGER,
        ID INTEGER),
        Primary Key (LID, ID)
        Foreign Key (LID) references Playlist(LID),
        Foreign Key (ID) references Audio(ID) )
```

LID	ID
3000	2000
3010	2018
3010	2221
3333	2213
3456	2345

```
CREATE TABLE Rates                       // Relationship where Listeners rate Audio
        ( UID INTEGER,
        ID INTEGER,
        rating INTEGER(1), // 1 - 5 stars
        Primary Key (UID, ID),
        Foreign Key (UID) references Listener(UID),
        Foreign Key (ID) references Audio(ID) )
```

UID	ID	rating
1000	2000	5
1234	2018	2
1672	2221	3
1999	2213	4
1456	2345	5

Functional Dependencies:

User (UID, name, year, **ID**)

- UID -> name, year, ID

Listener (UID)

- None

Artist (UID, RecordLabel, genre)

- UID → RecordLabel, genre

Audio (ID, name, year, genre, length)

- ID → name, year, genre, length

Album (AID, name, year, **UID**)

- AID → name, year, UID

Playlist (LID, name, **UID**)

- LID → name, UID

Create (**UID**, **ID**)

- None

Contains (**LID**, **ID)**

- None

Rates(**UID**, **ID**, rating)

- UID, ID → rating

Our schema is already in BCNF. Looking at each individual relationship, all attributes except for the primary keys and foreign keys, are not unique and have trivial functional dependencies (key dependencies). For example (WLOG), Audio's relationship shows 5 trivial attributes: name, year, genre, type, and length. These attributes can have the same values as another audio entity instance. None of these can be considered superkeys as two songs can have the same values for their attributes except for their primary and foreign keys. Additionally, all primary keys are super keys in their respective relationships. In conclusion, all functional dependencies that we could have are key dependencies.