1. 调研某一应用领域，给出该应用领域的详细需求描述;

在音乐唱片领域，数据库管理需求包括：

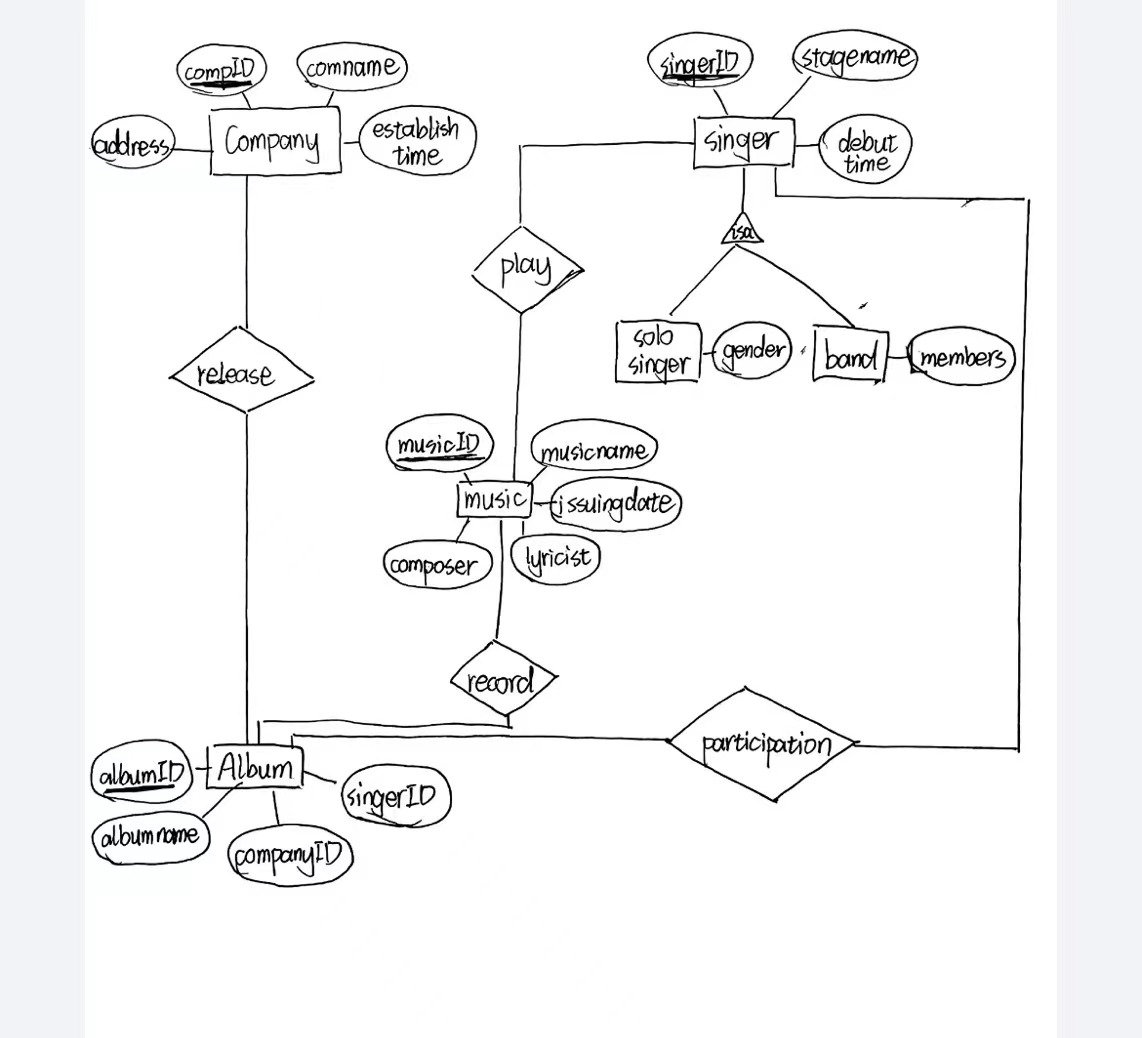
1. 数据收集与整理：收集专辑歌曲的基本信息，如专辑名、艺术家、发行日期、发行公司、流派、封面图片等，并进行整理分类，以便后续的数据分析和查询；
2. 市场需求数据调研：收集不同销量唱片的流派、歌手、专辑风格、发行公司等信息并进行综合分析，以得出当前市场需求和行业风向并帮助规划后续的唱片发行和推广策略，更有助于在竞争激烈的市场中脱颖而出；

（3）数据查询与报表生成：设计高效的数据查询功能，支持按照不同条件（如歌手、流派、发行日期等）进行检索；同时，生成各类销售报表、市场分析报告等，以支持决策制定。

数据安全性与备份：确保数据库的安全性，防止数据泄露或被篡改；同时，定期备份数据，以防数据丢失或损坏。

2.采用教材中介绍的方法实现如下设计:

a)画出该领域的概念模型ER图(至少有五个以上的实体，含有子类的形式，注意一定标明每个实体的主键);



b)请按课堂上讲授的ER图转换成关系模式的方法，将上述ER图转换成关系模式，并标明每个关系的主键属性和外键属性

company { compID，compname，address，establishtime }；

album {albumID, albumname, companyID,singerID};

solosinger {singerID, stagename, debuttime, gender};

band {singerID, stagename, debuttime, member};

music {musicID, singerID,albumID, musicname,issuingdate,composer,lyricist}; （ 为外键）

c)用SQL语句创建上述关系模式。

CREATE TABLE company (

compID INT PRIMARY KEY,

compname VARCHAR(255) NOT NULL,

address VARCHAR(255),

establishtime DATE

);

CREATE TABLE solosinger (

singerID INT PRIMARY KEY,

stagename VARCHAR(255) NOT NULL,

debuttime DATE,

gender ENUM('Male', 'Female', 'Other')

);

CREATE TABLE band (

singerID INT PRIMARY KEY,

stagename VARCHAR(255) NOT NULL,

debuttime DATE,

member TEXT

);

CREATE TABLE album (

albumID INT PRIMARY KEY,

albumname VARCHAR(255) NOT NULL,

solosingerID INT, -- 外键指向solosinger表

bandID INT, -- 外键指向band表

FOREIGN KEY (solosingerID) REFERENCES solosinger(singerID),

FOREIGN KEY (bandID) REFERENCES band(singerID)

);

CREATE TABLE music (

musicID INT PRIMARY KEY,

musicname VARCHAR(255) NOT NULL,

issuingdate DATE,

composer VARCHAR(255),

lyricist VARCHAR(255),

albumID INT, -- 外键指向album表

solosingerID INT, -- 外键指向solosinger表

bandID INT, -- 外键指向band表

FOREIGN KEY (albumID) REFERENCES album(albumID),

FOREIGN KEY (solosingerID) REFERENCES solosinger(singerID),

FOREIGN KEY (bandID) REFERENCES band(singerID)

);

d)给出该数据库模式上5个查询语句样例，分别为:单表查询、多表连接查询、多表嵌套查询和、EXISTS查询和聚合操作查询

1. 单表查询

Eg:查询所有独唱歌手的名字和性别：

sql

SELECT stagename, gender FROM solosinger;

2. 多表连接查询

Eg:查询所有专辑及其对应的独唱歌手的名字（假设专辑只能有一个独唱歌手）：

sql

SELECT album.albumname, solosinger.stagename

FROM album

JOIN solosinger ON album.solosingerID = solosinger.singerID;

3. 多表嵌套查询

Eg:查询发行过至少3张专辑的独唱歌手的名字：

sql

SELECT stagename

FROM solosinger

WHERE singerID IN (

SELECT solosingerID

FROM album

GROUP BY solosingerID

HAVING COUNT(\*) >= 3

);

4. EXISTS查询

Eg:查询有发行过专辑的独唱歌手的名字：

sql

SELECT stagename

FROM solosinger s

WHERE EXISTS (

SELECT 1

FROM album a

WHERE a.solosingerID = s.singerID

);

5. 聚合操作查询

Eg:查询每个solo歌手发行的专辑数量：

sql

SELECT solosinger.stagename, COUNT(album.albumID) AS album\_count

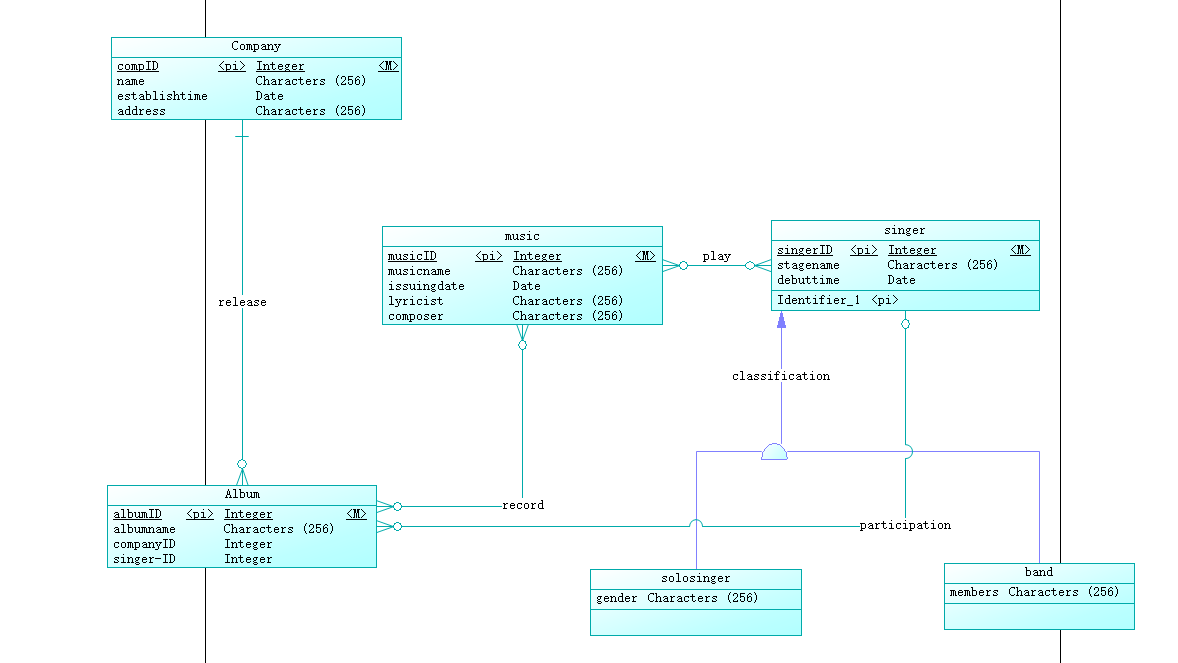
FROM solosinger

LEFT JOIN album ON solosinger.singerID = album.solosingerID

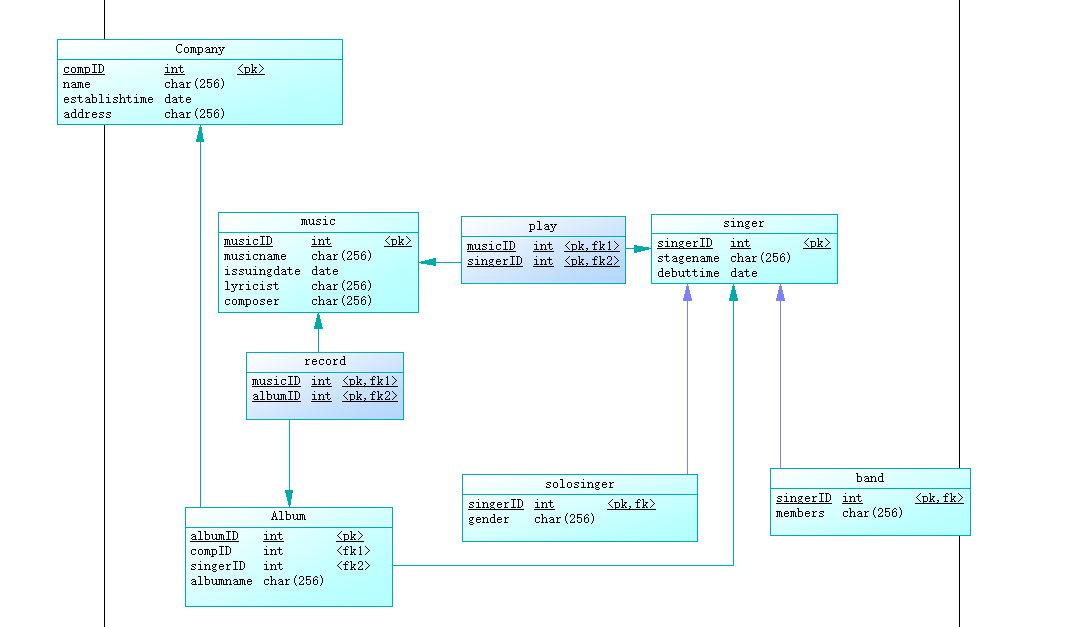
GROUP BY solosinger.singerID, solosinger.stagename;

1. 使用PowerDesianer工具实现如下设计:

a)画出该领域的概念模型ER图，给出ER图截图:



b)使用PowerDesigner工具，将上述ER图转为关系模型图，给出关系模型图截图



c)使用PowerDesigner工具，生成创建数据库的SQL语句。

/\*==============================================================\*/

/\* DBMS name: MySQL 5.0 \*/

/\* Created on: 2024/4/21 22:50:00 \*/

/\*==============================================================\*/

drop table if exists Album;

drop table if exists Company;

drop table if exists band;

drop table if exists music;

drop table if exists play;

drop table if exists record;

drop table if exists singer;

drop table if exists solosinger;

/\*==============================================================\*/

/\* Table: Album \*/

/\*==============================================================\*/

create table Album

(

albumID int not null,

compID int not null,

singerID int,

albumname char(256),

primary key (albumID)

);

/\*==============================================================\*/

/\* Table: Company \*/

/\*==============================================================\*/

create table Company

(

compID int not null,

name char(256),

establishtime date,

address char(256),

primary key (compID)

);

/\*==============================================================\*/

/\* Table: band \*/

/\*==============================================================\*/

create table band

(

singerID int not null,

members char(256),

primary key (singerID)

);

/\*==============================================================\*/

/\* Table: music \*/

/\*==============================================================\*/

create table music

(

musicID int not null,

musicname char(256),

issuingdate date,

lyricist char(256),

composer char(256),

primary key (musicID)

);

/\*==============================================================\*/

/\* Table: play \*/

/\*==============================================================\*/

create table play

(

musicID int not null,

singerID int not null,

primary key (musicID, singerID)

);

/\*==============================================================\*/

/\* Table: record \*/

/\*==============================================================\*/

create table record

(

musicID int not null,

albumID int not null,

primary key (musicID, albumID)

);

/\*==============================================================\*/

/\* Table: singer \*/

/\*==============================================================\*/

create table singer

(

singerID int not null,

stagename char(256),

debuttime date,

primary key (singerID)

);

/\*==============================================================\*/

/\* Table: solosinger \*/

/\*==============================================================\*/

create table solosinger

(

singerID int not null,

gender char(256),

primary key (singerID)

);

alter table Album add constraint FK\_participation foreign key (singerID)

references singer (singerID) on delete restrict on update restrict;

alter table Album add constraint FK\_release foreign key (compID)

references Company (compID) on delete restrict on update restrict;

alter table band add constraint FK\_classification2 foreign key (singerID)

references singer (singerID) on delete restrict on update restrict;

alter table play add constraint FK\_play foreign key (musicID)

references music (musicID) on delete restrict on update restrict;

alter table play add constraint FK\_play2 foreign key (singerID)

references singer (singerID) on delete restrict on update restrict;

alter table record add constraint FK\_record foreign key (musicID)

references music (musicID) on delete restrict on update restrict;

alter table record add constraint FK\_record2 foreign key (albumID)

references Album (albumID) on delete restrict on update restrict;

alter table solosinger add constraint FK\_classification foreign key (singerID)

references singer (singerID) on delete restrict on update restrict;

1. 分析比较采用上述两种方法
2. 两种关系模式的设计是否存在差异?如有差异，这种差异是否对后期的实现带来不同的影响?

答：

在第一种关系模式中，将关系release/play/record表转换成了实体集合的属性从而进行了省略，但在PowerDesigner生成的SQL中仍保留了play/record表的生成；

如果保留了play和record表的生成，实际数据库中就会存在额外的表结构和索引，可能会增加存储空间的占用和数据库的查询复杂度，在大型数据库中，可能会对性能产生负面影响。

b) PowerDesigner工具生成的SQL语句有什么样的特点?为什么会出现一些附加语句?它的作用是什么?

答：

（1）SQL语句通常是规范且结构化的，遵循数据库设计的规范；

1. 生成的SQL语句包含了表结构的定义，还可能包括字段类型、长度、约束、索引等详细信息；

附加语句使得数据库的完整性和性能大大提升，PowerDesigner可能会根据表之间的关系自动生成外键约束，这样就确保了数据的引用完整性；或者会根据查询需求自动生成索引，以此提高查询速度。