

NOTOWN
MUSICIANS
DATABASE

Table of Contents

Introduction	2
Problem Statement	3
Overview	4
Goals	4
Specifications	4
Conceptual database design – ER Model	5
Logical database design	7
Schema Design	8
The Starting point...	9
The Constraints on the Relations	11
musicians relation	11
addresses relation	11
instruments relation	11
musician_instruments relation	11
albums relation	12
songs relation	12
Code for creation of the database	13
Creating the relations – DDL	13
Foreign Key Constraints	13
Inserting the data – DML	16
Retrieval – DQL	18
Views	22
PL/SQL	25
Cursors	25
Functions	27
Triggers	29
References/Tools used	31

Introduction

This project attempts to address the requirements of Notown Records who have decided to store information about musicians who perform on its albums (as well as other company data) in a database. Database was developed entirely using Oracle services.

We have chosen NoTown Records out of all the other problem statements because of its creative flexibility and opportunities to learn and strengthen our knowledge on fundamental DBMS concepts like ER models, integrity constraints, decomposition and process of building a working database capable of handling real-life applications and demands. We team members are also passionate and curious about the workings of a database in real-time applications and also the music industry.

Record label companies and the music industry in general is a huge industry. Hence organizing and managing its data is a huge and daunting task. The music industry in India is estimated to reach 28 Billion rupees by the end of 2024. So it is a growing and thriving industry. So a database is absolutely necessary to keep track of all the musicians under a record label company.

Sales are a major part of any company so a database becomes a necessity rather than an option. Databases ensure data is consistent across all branches of the company. Music industry is Global so it is an assumption that the data is going to be spread out and distributed all over the world. Data integrity and consistency becomes utmost priority in this case. So keeping track of this data in this context is impossible in the traditional File Management systems which have several major drawbacks like storage inefficiency, data inconsistency, slow access speeds and many more.

Problem Statement

Notown Records has decided to store information about musicians who perform on its albums (as well as other company data) in a database. The company has wisely chosen to hire you as a database designer (at your usual consulting fee of \$2500/day).

- Each musician that records at Notown has an SSN, a name, an address, and a phone number. Poorly paid musicians often share the same address, and no address has more than one phone.
- Each instrument used in songs recorded at Notown has a name (e.g., guitar, synthesizer, flute) and a musical key (e.g., C, B-flat, E-flat).
- Each album recorded on the Notown label has a title, a copyright date, a format (e.g., CD or MC), and an album identifier.
- Each song recorded at Notown has a title and an author.
- Each musician may play several instruments, and a given instrument may be played by several musicians.
- Each album has a number of songs on it, but no song may appear on more than one album.
- Each song is performed by one or more musicians, and a musician may perform a number of songs.
- Each album has exactly one musician who acts as its producer. A musician may produce several albums, of course.

Design a conceptual schema for Notown and draw an ER diagram for your schema. The preceding information describes the situation that the Notown database model.

Overview

The ER Model is designed to meet the requirements of Notown Records. Based on the ER model created, The schema is created to represent the structure of the database. The Database is created consisting of relations (Musicians, Albums, Instruments, Songs) with the constraints and relations as requested by the client.

Goals

1. Create a DataBase Management system for Notown Records Company to efficiently manage their data about its music albums.
2. Creating a conceptual database design for the given problem statement
3. Create the schema with the help of conceptual design.
4. Create entities and link the relations using constraints.
5. Insert the data into the relations.
6. Retrieve the data from the relations.

Specifications

- The database is capable of storing musician details like their SSN (Social Security Number), name, address and other basic details.
- The record company primarily deals with albums and songs distribution so the database must be adept at storing details like album names, identifiers, contents, artists, instruments used and its copyright time period.
- Every instrument used in songs recorded at Notown has a name and a Musical Key.
- Musicians may play several instruments, and a given instrument may be played by several musicians.
- Each song recorded at Notown has a title and an author.
- Each album has a number of songs on it, but no song may appear on more than one album.
- Each song is performed by one or more musicians, and a musician may perform a number of songs.
- Each album has exactly one musician who acts as its producer. A musician may produce several albums.

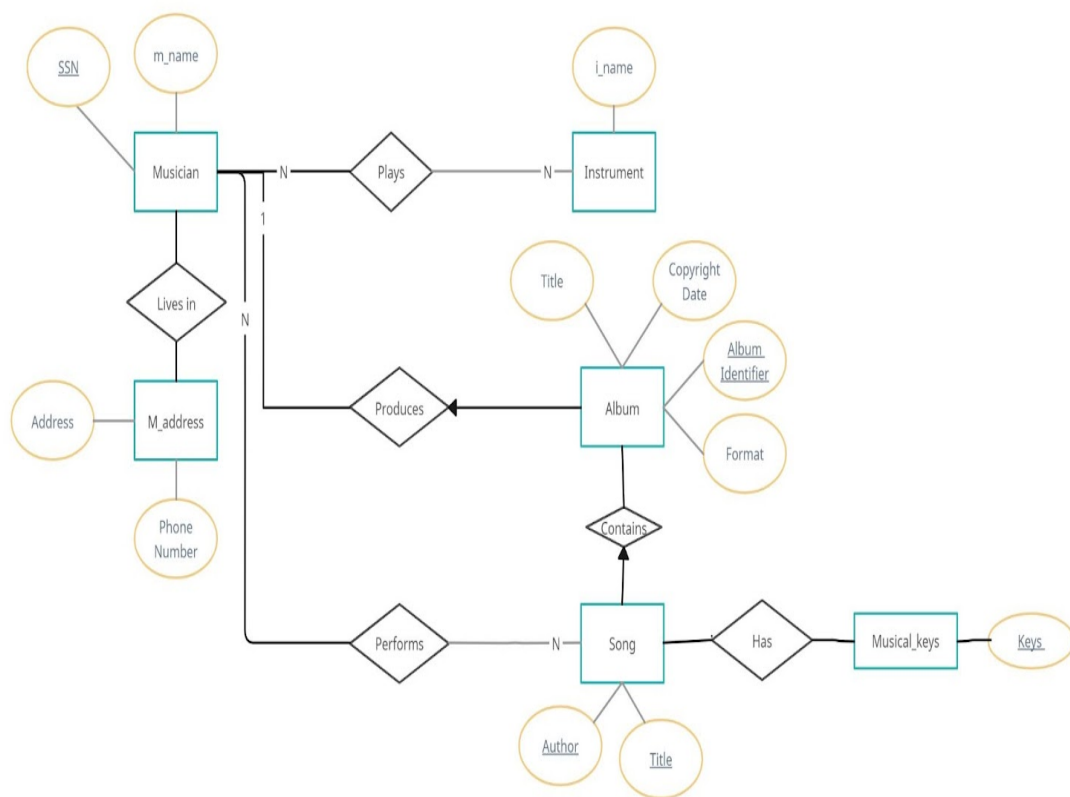
Description

Conceptual database design – ER Model

The ER Model is created using Creately Online Tool

<https://app.creately.com/d/create/?templateId=4hTQ5g1y3Yt>

ER Diagram



Entities

- Musician
- Instrument
- Album
- Song
- M_address
- Musical keys

Attributes

- Musician
 - SSN
 - m_name
- Instrument
 - i_name
- Album
 - Album Identifier
 - Title
 - Copyright Date
 - Format
- Song
 - Author
 - Title
- M_address
 - Address
 - Phone number
- Musical_Keys
 - Keys

Relationships

- Musician **Plays** Instrument (Many to Many)
- Musician **Lives in** M_address
- Musician **Produces** Album (One to Many)
- Album **Contains** Song (One to Many)
- Song **Has** Keys (One to Many)

Logical database design

Converting the ER diagram to schema

1. Identifying the strong entities

From the ER diagram musicians, addresses, songs, albums, instruments and musical_keys are identified strong entities

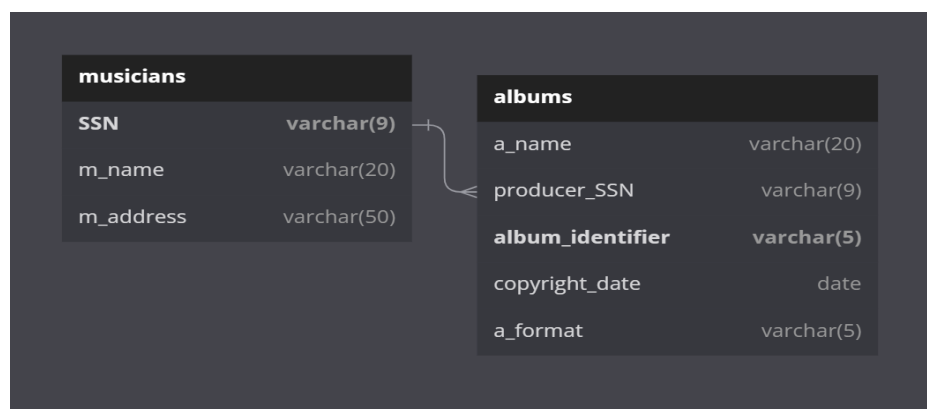
2. Identifying the One to Many relationships between the entities

Although the ER diagram doesn't have any one to one relationships in it, there are several one to many relationships present. To convert those one to many relationships, modify the relation in the "Many" side of the relationship and add an attribute containing the primary keys of the other relation as foreign key.



Consider this example. There is a One to Many relationship between musicians and albums and the relationship depicts musicians produce albums.

An attribute to the Albums relation named "producer_ssn" is added which holds the primary key of the musicians relation as foreign key.



Similarly all One to Many relationships are identified and integrated.

3. Identifying the Many to Many relations between the entities

In the ER diagram there are two Many to Many relationships. These relations are converted to a third relation called Junction relations. They contain primary keys from both the relations and they are made as foreign keys referencing their respective relations.

Then for the Primary key of the resulting relation we use the combination of both the foreign keys as the Composite Primary key.

For example consider the many to many relationship “Performs”.

A third relation “song_artists” is used to store the Primary keys of both musicians and songs Relation.

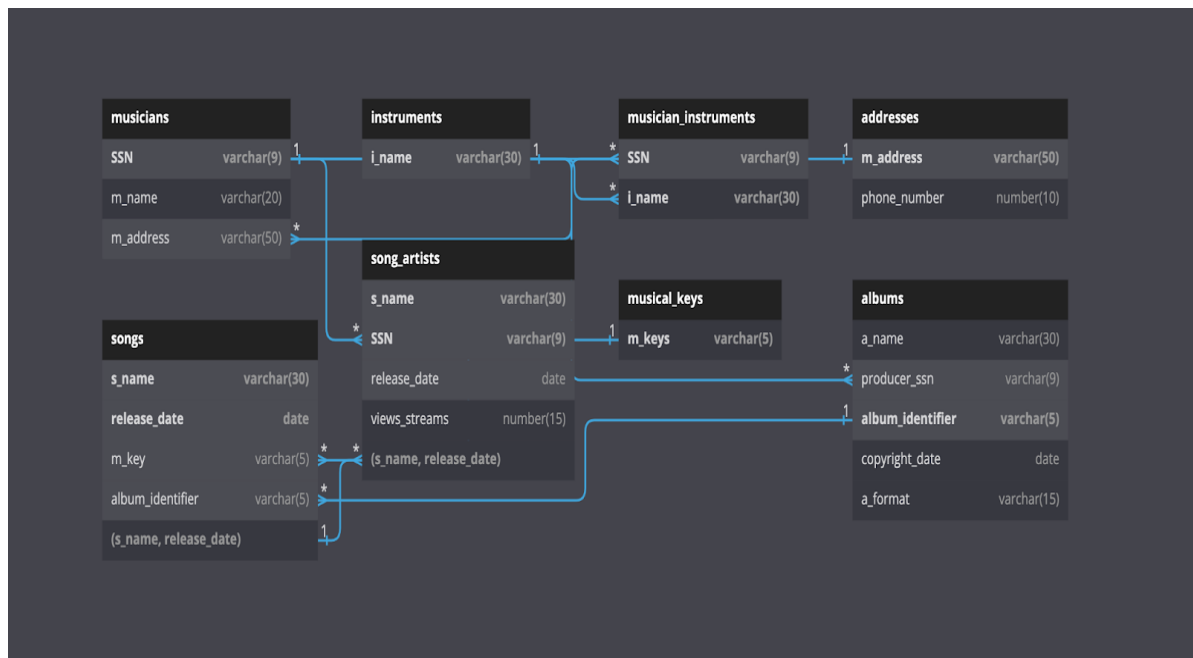
The Composite Primary key of songs_artists is a combination of Primary key of musicians relation and Composite Primary key of songs relation.

Schema Design

The schema is created using dbdiagram.io

<https://dbdiagram.io/d>

[Schema](#)



According to the requirements of the client (Notown Records company). The DataBase needs to be capable of storing data like

1. Data of musicians like their Name(m_name), SSN (Social Security Number), Address(m_address) and contact number(phone_number).
2. Instruments used in songs((i_name) and the Musical key(m_keys) they are played in.
3. Albums licensed by the record company and the songs(s_name) it contain along with the artists(m_name) and producers(producer_ssn)
4. Information about the instrument playing capabilities of the musicians under the record company

Some additional requirements:

1. No address may have more than one phone number
2. No song may appear more than once in its album
3. A song may contain more than one lead performer
4. Each album has only one Producer who is also a musician

The above additional requirements have been achieved by the use of basic Integrity and Referential constraints and allows easy access of data. The relations have been created with as little redundancy of data and as practical as possible.

The Starting point...

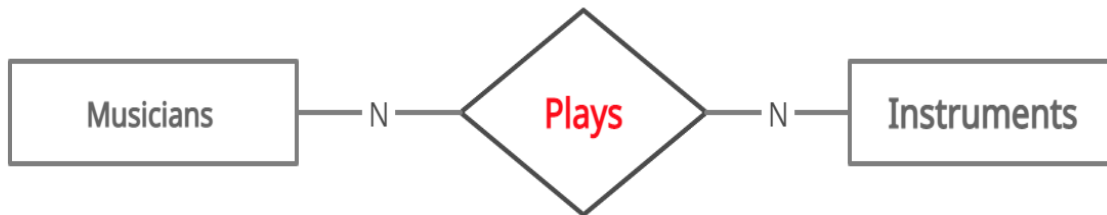
The number of relations needed by the DataBase to meet the requirements of the client is a minimum of 8. The names of the relations are as follows

1. musicians
2. addresses
3. songs
4. albums
5. song_artists
6. instruments
7. musician_Instruments
8. musical_keys

Of these relations, the song_artists and musician_instruments relations are Junction relations. Junction relations are often used when there is a Many to Many relationship between two relations. Using this junction Relationship the ease in storing and retrieving data is improved.



The performs relationship between the Musicians Relations and Songs Relations



The plays relationship between the Musicians Relations and Songs Relations

The above relationships depict many to many relationship between musicians and the songs they perform, as well as relationship between musicians and the instruments they can play.

The Musical Keys

Musical keys are stored as a separate relation instead of only as an attribute in songs because key signatures of songs are often repeating and there are only 12 major key signatures used in music and another 12 minor forms. Almost every song is written following only one key signature and rarely song writers use more than one key so we can assume one song contains one key.

The List of **24** keys used in songs is:

12 Major Keys	A	A#	B	C	C#	D	D#	E	F	F#	G	G#
12 Minor Keys	Am	A#m	Bm	Cm	C#m	Dm	D#m	Em	Fm	F#m	Gm	G#m

Here the # is read as sharp and m is read as minor

So A# is read as ‘ A sharp Major’ and A#m is read as “A sharp minor”

The Constraints on the Relations

musicians relation

The data in this relation is SSN, m_name and m_address. In these attributes the good candidate for a primary key is SSN. Social security numbers are unique by nature and

assigned by the government. m_name attribute won't be reliable because names might repeat. The m_address attribute here refers to the same attribute in the addresses Relation mentioned below to meet one of the client's request.

addresses relation

The addresses relation stores the addresses and the phone_number of that residence.

As per the requirement specified by the client “ No address may have more than one phone number and some musicians may share the same address ”. To meet this requirement a separate relation for addresses is used. To solve the issue of multiple contacts for one address we make the address attribute in this relation as the primary key.

This is practical because of the following reasons:

1. Multiple occurrences of an address is avoided and hence only one phone will be paired with one address.
2. Since the address attribute in this relation is referred to by the m_address attribute in the musicians relation, the possibility of musicians having the same address is also addressed.

instruments relation

The instruments relation stores the name of the instrument (i_name). Here the only attribute is the Instrument name (i_name) hence it is the primary key.

musician_instruments relation

This Junction relation stores the SSN of the musicians relation and the name of the instruments played by them. Here SSN is a foreign key which refers to the same attribute in the musicians relation. Similarly Instrument name (i_name) is also a foreign key which refers to the same attribute in the Instruments relation mentioned above. Here an ordinary primary key won't be sufficient because both musicians name (m_name) and instruments name (i_name) can repeat because this is a junction relation and it represents a Many to Many relationship.

So we make the combination of the musicians name (m_name) and the instruments name (i_name) the composite primary key.

albums relation

This relation contains the information about the albums name (a_name), its producer (producer_ssn), a copyright date (copyright_date), the album format (a_format) and its album_identification number (album_identifier).

Here the producer is a foreign key which refers to the Musicians relation's SSN attribute. Producers are always musicians themselves so it's convenient for us to use that data to store the albums producer.

Here choosing the primary key is easy because we also store the album identification number in our Relations. It will be a perfect candidate for the primary key because the album identification number is a 7 character unique code for every album assigned by their respective record companies.

songs relation

Here the song name, release date, ID of the album it's from and it's Musical Key

Choosing the primary is not straightforward since a song name is not always unique and there are multiple album IDs occurring in the songs Relations. Hence the combination of song name and its release date is made as the composite primary key. Though there is a possibility of two songs with the same name being released on the same date, the chances of that occurring is very less.

Here the musical key and album_identifier attributes are both foreign keys which refer to the Musical Keys Relations and the albums Relations respectively. With the album identifier as the foreign key we can easily retrieve the album name and its data of any song.

song_artists Relation

This is a Junction Relations and as mentioned previously Many to Many relations are stored as separate relations or Third relations. This Relations contains song name, SSN of its performer and its release date.

Song's name cannot be chosen as the primary key because it is very common for songs to have the same name. So the combination of the song's name and its artist's SSN is taken as a composite primary key.

Song name is a foreign key which refers to the attribute in the Songs Relations and SSN refers to the same attribute in the Musicians Relations

musical-keys Relations

This relationship only contains the keys used in songs and it is also the primary key.

Code for creation of the database

Creating the relations – DDL

The 8 relations are created using the basic DDL CREATE command. The primary keys are created at column level whereas the foreign key is at the Relations level. Foreign keys are given at Relations level to provide easy modification and better readability.

musicians Relation

```
CREATE TABLE musicians (  
  SSN varchar(9) PRIMARY KEY,  
  m_name varchar(20),
```

TABLE MUSICIANS		
Column	Null?	Type
SSN	NOT NULL	VARCHAR2(9)
M_NAME	-	VARCHAR2(20)
M_ADDRESS	-	VARCHAR2(50)

```
m_address varchar(50)
);
```

addresses Relation

```
CREATE TABLE addresses (
  m_address varchar(50) PRIMARY KEY,
  phone_number number(10)
);
```

TABLE ADDRESSES		
Column	Null?	Type
M_ADDRESS	NOT NULL	VARCHAR2(50)
PHONE_NUMBER	-	NUMBER(10,0)

instruments Relation

```
CREATE TABLE instruments (
  i_name varchar(30) PRIMARY KEY
);
```

TABLE INSTRUMENTS		
Column	Null?	Type
I_NAME	NOT NULL	VARCHAR2(30)

musician_instruments Relation

```
CREATE TABLE
musician_instruments (
  SSN varchar(9),
  i_name varchar(30),
  PRIMARY KEY (SSN, i_name)
);
```

TABLE MUSICIAN_INSTRUMENTS		
Column	Null?	Type
SSN	NOT NULL	VARCHAR2(9)
I_NAME	NOT NULL	VARCHAR2(30)

albums Relation

```
CREATE TABLE albums (
  a_name varchar(30),
  producer_ssn varchar(9),
  album_identifier varchar(5) PRIMARY KEY,
  copyright_date date,
  a_format varchar(15)
);
```

TABLE ALBUMS		
Column	Null?	Type
A_NAME	-	VARCHAR2(30)
PRODUCER_SSN	-	VARCHAR2(9)
ALBUM_IDENTIFIER	NOT NULL	VARCHAR2(5)
COPYRIGHT_DATE	-	DATE
A_FORMAT	-	VARCHAR2(15)

);

songs Relation

```
CREATE TABLE songs (  
  s_name varchar(30),  
  release_date date,  
  m_key varchar(5),  
  album_identifier varchar(5),  
  PRIMARY KEY (s_name, release_date));
```

TABLE SONGS		
Column	Null?	Type
S_NAME	NOT NULL	VARCHAR2(30)
RELEASE_DATE	NOT NULL	DATE
M_KEY	-	VARCHAR2(5)
ALBUM_IDENTIFIER	-	VARCHAR2(5)

song_artists Relation

```
CREATE TABLE song_artists (  
  s_name varchar(30),  
  SSN varchar(9),  
  release_date date,  
  views_streams number(15),  
  PRIMARY KEY (s_name, SSN));
```

TABLE SONG_ARTISTS		
Column	Null?	Type
S_NAME	NOT NULL	VARCHAR2(30)
SSN	NOT NULL	VARCHAR2(9)
RELEASE_DATE	-	DATE

musical_keys Relation

```
CREATE TABLE musical_keys (  
  m_keys varchar(5) PRIMARY KEY);
```

TABLE MUSICAL_KEYS		
Column	Null?	Type
M_KEYS	NOT NULL	VARCHAR2(5)

Foreign Key Constraints

Foreign key between musicians and addresses

```
ALTER TABLE musicians ADD FOREIGN KEY (m_address) REFERENCES addresses  
(m_address);
```

Foreign key between musician_instruments and musicians

```
ALTER TABLE musician_instruments ADD CONSTRAINT ssn_fk FOREIGN KEY (SSN)  
REFERENCES musicians (SSN);
```

Foreign key between musician_instruments and instruments

```
ALTER TABLE musician_instruments ADD CONSTRAINT i_name_fk FOREIGN KEY  
(i_name) REFERENCES instruments (i_name);
```

Foreign key between song_artists and songs

```
ALTER TABLE song_artists ADD CONSTRAINT s_artists_fk FOREIGN KEY (s_name, release_date) REFERENCES songs (s_name, release_date);
```

Foreign key between song_artists and musicians

```
ALTER TABLE song_artists ADD CONSTRAINT s_artists_SSN_fk FOREIGN KEY (SSN) REFERENCES musicians (SSN);
```

Foreign key between songs and albums

```
ALTER TABLE songs ADD CONSTRAINT a_name_fk FOREIGN KEY (album_identifier) REFERENCES albums (album_identifier);
```

Foreign key between albums and musicians

```
ALTER TABLE albums ADD FOREIGN KEY (producer_ssn) REFERENCES musicians (SSN);
```

Foreign key between songs and musical_keys

```
ALTER TABLE songs ADD CONSTRAINT m_key_fk FOREIGN KEY (m_key) REFERENCES musical_keys (m_keys);
```

Inserting the data – DML

Insertion is done by deactivating the foreign key constraints or done before they are declared because insertion and deletion with foreign key constraints activated is a tedious task. After insertion is done we can activate them.

Inserting into musicians Relations

```
insert into musicians values('123456789','Armaan malik','Street-2');
insert into musicians values('835732871','Sid sriram','Street-1');
```

Inserting into songs Relations

```
insert into songs values('Padi padi leche manasu','21-DEC-2018','Am','c-101');
insert into songs values('Emai poyave','21-DEC-2018','A#','c-101');
```

Inserting into albums Relations

```
insert into albums values('Padi padi leche manasu','835732871','c-101','21-DEC-2018','LP');
insert into albums values('Hit_2','835732871','c-102','02-DEC-2022','Double LP');
```

Inserting into instruments Relations

```
insert into instruments values('Bass guitar');
insert into instruments values('Guitar');
```

Inserting into addresses Relations


```
insert into addresses values('Street-2',9848451542);  
insert into addresses values('Street-1',8465478134);
```

Inserting into musician_instruments Relations

```
insert into musician_instruments values('835732871','Piano');  
insert into musician_instruments values('835732871','Harmonium');
```

Inserting into musical_keys Relations

```
insert into musical_keys values('Am');  
insert into musical_keys values('A#m');
```

Inserting into song_artists Relations

```
insert into song_artists values('Padi padi leche manasu','123456789','21-DEC-2018');  
insert into song_artists values('Emai poyave','835732871','21-DEC-2018');
```

The rows are inserted before the Foreign key constraints are activated to allow easy insertion. To insert rows this way we need to make sure our data is consistent across the relations. Data like names, ID numbers all must match exactly and the data is case sensitive. If this is not taken care of, we will get an error at the time of activating the constraints.

Retrieval – DQL

Below are few queries that the client might use frequently

Display poor musicians that share the same address

```
select m_name "poor musicians",m_address  
from musicians  
where m_address in (select m_address  
from musicians m  
group by m_address  
having count(m_name)>1)  
order by m_address;
```

poor musicians	M_ADDRESS
Ram miryala	Street-12
Harish raghavendra	Street-12
Armaan malik	Street-2
Ananya bhat	Street-2
Sri krishna	Street-2
KK	Street-20
Devan ekambaram	Street-20

Display Producers and albums produced by them

```
select m_name "Producer",a_name "Album"
from musicians m
join albums al on m.ssn = al.producer_ssn;
```

Producer	Album
Sid sriram	Padi padi leche manasu
Sid sriram	Hit_2
Anirudh	Ori devuda
Sid sriram	Bhramstra
Haricharan	Awara
Karthik	Orange

Display the album and number of songs in it

```
select album_identifier as "Album ID", a_name as Album, count(s_name) as Songs
from songs
natural join albums
group by album_identifier,a_name;
```

Album_ID	Album	Songs
c-101	Padi padi leche manasu	2
c-106	Orange	2
c-112	Kirak party	2
c-127	Pournami	1
c-110	Raahu	1

Display all the songs and their artist

```

select s.s_name "Song",m.m_name "Singer"
from song_artists s join musicians m
on s.ssn=m.ssn;

```

Song	Singer
Appudo ippudo	Sidharth
Asha pasham	Anurag kulkarni
Chilipiga	Karthik
Chiru chiru	Haricharan
Chitti	Ram miryala

Display all the songs and the musical key they are played in

```

select s.s_name "Song",m.m_keys"musical_keys"
from songs s join musical_keys m
on s.m_key=m.m_keys;

```

Song	musical_keys
Urike urike	A#m
Gundelona Gundelona	B
Tholiprema	C#m
Why this kolaveri d	D
Uppenantha e premaki	D#m
Chitti	Dm

Display all the musicians and their home phone number

```

select m_name, phone_number
from musicians
natural join addresses;

```

M_NAME	PHONE_NUMBER
Armaan malik	9848451542
Sid sriram	8465478134
Anirudh	9965432155
Haricharan	9965432155
Karthik	9634554321
Vijay prakash	9564783154

Views

```
SQL>create view musician_name_view as select m_name from musicians;
```

```
SQL>create view musician_album_view as select mu.m_name,al.a_name from musicians  
mu join albums al on
```

```
al.producer_ssn=mu.ssn;
```

```
SQL>create view musician_name_address_view as select m_name,m_address from  
musicians mu with read only constraint munaaddviewrc;
```

```
create view albums_view as select * from albums with read only constraint albumrc;
```

```
SQL>Create view musician_song_streams as select m.m_name,sa.s_name,sa.views_streams
from musicians m join song_artists sa on m.SSN = sa.SSN;
```

Displaying all the musicians using views

```
select * from musician_name_view;
```

M_NAME
Sid sriram
Anirudh
SP Charan
Ram miryala
Anurag kulkarni
DSP
VV Prassanna
Sri krishna

Displaying albums produced by the musicians

```
select * from musician_album_view;
```

M_NAME	A_NAME
Sid sriram	Hit_2
Karthik	Orange
SP Charan	Sita ramam
KK	Aarya-2

Anurag kulkarni	c/o kancherapalam
DSP	Pournami
VV Prassanna	Surya s/o krishnan
Ananya bhat	KGF Chapter-1

Updating the read only view

update musician_name_address_view set m_address = 'Street-3' where m_name = 'Armaan malik';

ORA-42399: cannot perform a DML operation on a read-only view

Displaying the number of views/streams for a song

select * from musician_song_streams;

M_NAME	S_NAME	VIEWS_STREAMS
Sid sriram	Urike urike	9185367
Anirudh	Gundelona Gundelona	48384325
Karthik	Chilipiga	6544543
Vijay prakash	Hello rammante	4370924
Kala bhairava	Tholiprema	6416534

Dhanush	Why this kolaveri d	1346704
Sid sriram	Emo emo	1952977
Ram miryala	Chitti	1418317
Haricharan	Dum dhara	1032370
DSP	Flute music bit	1523546

PL/SQL

Cursors

1. PL/SQL code to display top 3 artists with most number of songs:

```

DECLARE
    cursor c_top3_artists IS
        select m_name, count(m_name) as c from musicians
        natural join song_artists
        group by m_name
        order by count(m_name) desc;
    c_var c_top3_artists %ROWTYPE;
    v_i number(5) := 0;
BEGIN
    DBMS_OUTPUT.PUT_LINE('The top 3 artists with most songs are:');
    OPEN c_top3_artists;
    LOOP
        FETCH c_top3_artists INTO c_var;

```



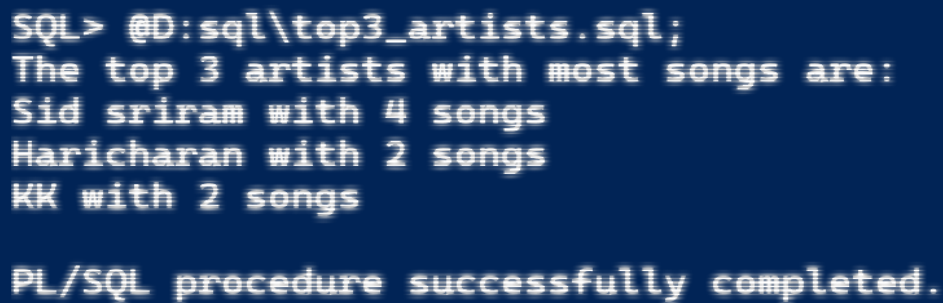
```

        DBMS_OUTPUT.PUT_LINE(c_var.m_name||' with ' ||
c_var.c || ' songs ');
        v_i := v_i + 1;
        EXIT WHEN c_top3_artists %NOTFOUND OR v_i>=3;

    END LOOP;
    CLOSE c_top3_artists;
END;
/

```

OUTPUT:



```

SQL> @D:\sql\top3_artists.sql;
The top 3 artists with most songs are:
Sid sriram with 4 songs
Haricharan with 2 songs
KK with 2 songs

PL/SQL procedure successfully completed.

```

2. PL/SQL code to display top 5 most recent songs released:

set serveroutput on;

```

DECLARE
    cursor c_top5 IS
    select e.m_name, s.s_name, s.release_date
    from musicians e inner join song_artists s
    on e.ssn = s.ssn order by release_date desc;
    c_var c_top5 %ROWTYPE;
    v_i number(5) := 0;
BEGIN
    DBMS_OUTPUT.PUT_LINE('The top 5 latest songs are:');
    OPEN c_top5;
    LOOP
        FETCH c_top5 INTO c_var;
        DBMS_OUTPUT.PUT_LINE(c_var.s_name||' By '
||c_var.m_name ||' released on ' || c_var.release_date);
        v_i := v_i + 1;
        EXIT WHEN c_top5 %NOTFOUND OR v_i>=5;

    END LOOP;
    CLOSE c_top5;
END;

```

/

OUTPUT:

```
SQL> @D:sql\top5.sql;
The top 5 latest songs are:
Urike urike By Sid sriram released on 02-DEC-22
Gundelona Gundelona By Anirudh released on 21-OCT-22
Kumkumala nuve By Sid sriram released on 09-SEP-22
Inthandham By SP Charan released on 05-AUG-22
Chitti By Ram miryala released on 11-MAR-21

PL/SQL procedure successfully completed.
```

Functions

1. A function returning the earnings of the artists from the song

(Function declaration)

```
CREATE OR REPLACE FUNCTION earnings (n1 IN NUMBER)
RETURN NUMBER
IS n2 NUMBER(15) := 0;
   temp NUMBER(15);

BEGIN
IF n1<1000000 THEN
   n2 := 0;
ELSIF n1>1000000 AND n1<2000000 THEN
   n2 := n1*0.2;
ELSE
   temp := n1-2000000;
   n2 := (temp * 0.4)+(2000000*0.2);
END IF;
RETURN n2;
END;
```

/

Note: The revenue is calculated as follows.

If a song has less than a million (1,000,000) views/streams, It gets no earnings but if a song gets more than a million it earns 0.2 rupees per view till 2 million views. After 2 million views the songs earn 0.4 rupees per view.

OUTPUT:

A screenshot of a SQL command prompt with a dark blue background and white text. The prompt shows the command 'SQL> @D:\SQL\func.sql;' followed by the output 'Function created.'

```
SQL> @D:\SQL\func.sql;  
  
Function created.
```

(function call):

DECLARE

cursor c_earn IS

select m.m_name, s.s_name, s.views_streams from musicians m inner join song_artists s on
s.ssn=m.ssn where m.ssn=&SSN;

temp c_earn %ROWTYPE;

tot_sal number(20):=0;

BEGIN

DBMS_OUTPUT.PUT_LINE('earnings of the artist are: ');

OPEN c_earn;

LOOP

 FETCH c_earn INTO temp;

 EXIT WHEN c_earn %NOTFOUND;

 DBMS_OUTPUT.PUT_LINE(temp.m_name || ' earned
\$'||earnings(temp.views_streams)/81.49 || ' from '||temp.s_name);

 tot_sal := tot_sal + earnings(temp.views_streams);

END LOOP;

DBMS_OUTPUT.PUT_LINE('Total earnings are \$'|| tot_sal/81.49);

CLOSE c_earn;

END;

/

OUTPUT:

```

SQL> @D:\sql\func_call.sql;
Enter value for ssn: 835732871
old 3:      select m.m_name, s.s_name, s.views_streams from musicians m inner join song_artists s on s.ssn=m.ssn where m.ssn=&SSN;
new 3:      select m.m_name, s.s_name, s.views_streams from musicians m inner join song_artists s on s.ssn=m.ssn where m.ssn=835732871;
earnings of the artist are:
Sid sriram earned $379333.8875 from Emai poyave
Sid sriram earned $40926.8375 from Urike uriike
Sid sriram earned $29127.75 from Kumkumala nuve
Sid sriram earned $4882.4375 from Emo emo
Total earnings are $454270.9125

PL/SQL procedure successfully completed.

```

Triggers

1. Trigger for updating the address of musicians

```

CREATE OR REPLACE TRIGGER tgr_update
  BEFORE UPDATE OF m_address on musicians
  FOR EACH ROW
DECLARE
  v_add number(5) := 0;
BEGIN
  select count(m_address) into v_add from addresses
  where m_address = :new.m_address;
  DBMS_OUTPUT.PUT_LINE(v_add);
  IF v_add=0 then
    insert into addresses values(:new.m_address, 0);

  END IF;

END;
/

```

OUTPUT:

```

SQL> @D:\SQL\trigger.sql;

Trigger created.

```

This triggers when an update operation is performed on the m_address attribute of musicians relation.

If the updated m_address record is already present in the master relation 'addresses'. There is no conflict with referential integrity constraints so we face no problem.

But If the updated m_address attribute is not present in the master relation then this trigger adds the newly updated address to addresses relation as well.

Updating the address of a musician:

```
SQL> update musicians set m_address = 'Street-1' where ssn=123456789;
```

// Note: Street-1 already exists in the addresses relation

OUTPUT:

```
SQL> update musicians set m_address = 'Street-1' where ssn=123456789;
1

1 row updated.
```

Successfully updated

```
SQL> select m_name, m_address from musicians where ssn=123456789;

M_NAME          M_ADDRESS
-----
Armaan malik    Street-1
```

Updating the address of a musician with an unseen address:

```
SQL> update musicians set m_address = 'Street-99' where ssn=123456789;
```

OUTPUT:

```
SQL> update musicians set m_address = 'Street-99' where ssn=123456789;  
0  
  
1 row updated.
```

Changes made in musicians relation:

```
SQL> select m_name, m_address from musicians where ssn=123456789;
```

M_NAME	M_ADDRESS
Armaan malik	Street-99

Changes in the addresses relation:

```
SQL> select m_address from addresses where m_address='Street-99';
```

M_ADDRESS
Street-99

A new record of the updated address is added to the addresses relation by the trigger.

References/Tools used

1. "Database Management Systems", Raghurama Krishnan, Johannes Gehrke, TATA McGraw Hill 3rd Edition
2. "Database System Concepts", Silberschatz, Korth, McGraw hill, V Edition.
3. "Introduction to Database Systems", C.J. Date Pearson Education.
4. "Database Systems design, Implementation, and Management", Rob & Coronel 5th Edition.
5. "Database Management Systems", P. Radha Krishna HI-TECH Publications 2005.
6. "Database Management System ", Elmasri Navathe, Pearson Education.
7. "Database Management System", Mathew Leon, Leo.
8. The Complete Reference, 3rd edition by James R. Groff, Paul N. Weinberg, Andrew J. Oppel
9. SQL & PL/SQL for Oracle 19c, Black Book, Dr. P. S. Deshpande.
10. [ER Diagram tool](#)
11. [Logical database design](#)
12. [Live SQL](#)