SQL WORKBOOK

GIVEN: album_collection_schema

```
CREATE TABLE IF NOT EXISTS artists (
  artist_id INTEGER(10) AUTO_INCREMENT PRIMARY KEY,
  artist_name varchar(128)
);

CREATE TABLE IF NOT EXISTS albums (
  album_id INTEGER(10) AUTO_INCREMENT PRIMARY KEY,
  album_name varchar(256), -- note that two different albums may have the
same name
  date_of_release DATE -- date of release of the album
);

CREATE TABLE IF NOT EXISTS songs (
  song_id INTEGER(10) AUTO_INCREMENT PRIMARY KEY,
  song_name varchar(128),
  genre varchar(32),
  song_length FLOAT, -- length of the song in minutes
  date_of_release DATE -- date of release of the song (may be different
form the date of release of any album the song is on)
);

CREATE TABLE IF NOT EXISTS album_artist (
  album_id INTEGER(10) REFERENCES albums(album_id),
  artist_id INTEGER(10) REFERENCES artists(artist_id), -- artist who is
listed as one of the artists on the album
  PRIMARY KEY(artist_id, album_id)
);

CREATE TABLE IF NOT EXISTS song_artist (
  -- note that a song may have multiple artists collaborating on it
  song_id INTEGER(10) REFERENCES songs(song_id),
  artist_id INTEGER(10) REFERENCES artists(artist_id),
  PRIMARY KEY(song_id, artist_id)
);

CREATE TABLE IF NOT EXISTS song_album (
  -- note that a song may appear on multiple albums
  song_id INTEGER(10) REFERENCES songs(song_id),
  album_id INTEGER(10) REFERENCES albums(album_id),
  track_no INTEGER NOT NULL, -- position of the song on the tracklist of
the album
  PRIMARY KEY(song_id, album_id),
  UNIQUE(album_id, track_no)
);
```

QUESTION 1.

Create view **Exceptions(artist_name, album_name)**. (A, B) is a data row in this view if and only if artist A contributes to at least one song on album B (according to table **song_artist**) but artist A is not listed as one of the artists on album B in table **album_artist**. There should be no duplicate data rows in the view.

SOLUTION:

```
CREATE VIEW Exceptions AS

SELECT DISTINCT artists.artist_name, albums.album_name
FROM song_artist
JOIN song_album ON song_artist.song_id = song_album.song_id
JOIN artists ON song_artist.artist_id = artists.artist_id
JOIN albums ON song_album.album_id = albums.album_id

EXCEPT

SELECT artists.artist_name, albums.album_name
FROM album_artist
JOIN artists ON album_artist.artist_id = artists.artist_id
JOIN albums ON album_artist.album_id = albums.album_id
```

QUESTION 2.

Create view **AlbumInfo(album_name, list_of_artist, date_of_release, total_length)**. Each album should be listed exactly once. For each album, the value in column **list_of_artists** is a comma-separated list of all artists on the album according to table **album_artist**. The value in column **total_length** is the total length of the album in minutes.

SOLUTION:

```
CREATE VIEW AlbumInfo AS
SELECT tab1.album_name, tab1.list_of_artist, tab1.date_of_release,
tab2.total_length

FROM

(SELECT albums.album_id, albums.album_name, albums.date_of_release,
string_agg(artists.artist_name,',') as list_of_artist
FROM albums
JOIN album_artist ON album_artist.album_id = albums.album_id
JOIN artists ON artists.artist_id = album_artist.artist_id
GROUP BY albums.album_id, albums.date_of_release) AS tab1

JOIN

(SELECT albums.album_id,albums.album_name, SUM(songs.song_length) as
total_length
FROM albums
JOIN song_album ON song_album.album_id = albums.album_id
JOIN songs ON songs.song_id = song_album.song_id
GROUP BY albums.album_id) AS tab2 ON tab1.album_id = tab2.album_id
```

QUESTION 3.

Write trigger **CheckReleaseDate** that does the following. Assume a new row (S, A, TN) is inserted into table **song_album** with **song_id** S, **album_id** A and **track_no** TN. Check if the release date of song S is later than the release date of album A. If this is the case, then change the release date of song S in table **songs** to be the same as the release date of album A.

SOLUTION:

```
CREATE OR REPLACE FUNCTION log_checkreleasedate()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS
$$
BEGIN

        IF ( SELECT songs.date_of_release FROM songs WHERE songs.song_id =
NEW.song_id)
               >
               (SELECT albums.date_of_release FROM albums WHERE
albums.album_id = NEW.album_id)
        THEN
                UPDATE songs
                SET date_of_release = (SELECT albums.date_of_release FROM
albums WHERE albums.album_id = NEW.album_id)
                WHERE songs.song_id = NEW.song_id;
        END IF;

        RETURN NEW;
END;
$$

CREATE OR REPLACE TRIGGER CheckReleaseDate
  AFTER INSERT
  ON song_album
  FOR EACH ROW
  EXECUTE PROCEDURE log_checkreleasedate();
```

QUESTION 4.

Write stored procedure **AddTrack(A, S)** where A is an **album_id** and S is a **songs_id**. The procedure should check if A is an **album_id** already existing in table **albums** and S is a **song_id** already existing in table **songs**. If both conditions are satisfied then the procedure should insert data row (S, A, TN+1) into table **song_album** where TN is the highest **track_no** for album A in table **song_album** before inserting the row.

SOLUTION:

```
CREATE OR REPLACE PROCEDURE AddTrack(a_id INTEGER, s_id INTEGER) AS $$
BEGIN
IF (EXISTS(SELECT * FROM albums WHERE albums.album_id = a_id))
        AND
   (EXISTS(SELECT * FROM songs WHERE songs.song_id = s_id))
```

```
THEN
        INSERT INTO song_album(song_id, album_id, track_no)
                VALUES(
                s_id, a_id,
                (SELECT Max(track_no) + 1
                 FROM song_album
                 WHERE song_album.album_id = a_id));
END IF;
END;
$$
LANGUAGE plpgsql
```

QUESTION 5.

Write stored function **GetTrackList(A)** which, for a given **album_id** A, returns a comma-separated list of the names of all songs on the album ordered according to their **track_no**.

SOLUTION:

```
CREATE OR REPLACE FUNCTION GetTrackList(a_id INTEGER)
RETURNS VARCHAR(255) AS $list_of_songs$

DECLARE list_of_songs VARCHAR(250);
BEGIN
   SELECT string_agg(tab4.song_name,',') into list_of_songs
        FROM
        (
                SELECT albums.album_id as al_id, songs.song_name,
song_album.track_no
                FROM albums
                JOIN song_album
                ON albums.album_id = song_album.album_id
                JOIN songs
                ON songs.song_id = song_album.song_id
                WHERE albums.album_id = a_id
                ORDER BY song_album.track_no ASC
        ) as tab4

        GROUP BY tab4.al_id;
        RETURN list_of_songs;
END;
$list_of_songs$
LANGUAGE plpgsql;
```