**Music Label Database – project documentation**

Author: Mariusz Galjan

Date: 14.12.2019

Table of Contents:

[1 Introduction 4](#_Toc40914967)

[2 Problem description 4](#_Toc40914968)

[3 Project assumptions 4](#_Toc40914969)

[3.1 Business description. 4](#_Toc40914970)

[3.2 Description by method of nouns and verbs. 5](#_Toc40914971)

[3.3 Relationships model. 6](#_Toc40914972)

[6](#_Toc40914973)

[4 Conceptual Database Model (CDM) 7](#_Toc40914974)

[5 Physical Database model (PDM) 8](#_Toc40914975)

[6 Create the database 9](#_Toc40914976)

[5.1 Create a database structure. 9](#_Toc40914977)

[6.2 Data entry. 9](#_Toc40914978)

[6.3 Listings of entered data (alphabetically) 9](#_Toc40914979)

[7 Queries (sorted by difficulty) 13](#_Toc40914980)

[7.1 Show studios cheaper than 100 $. 13](#_Toc40914981)

[7.2 Show Led Zeppelin tours 13](#_Toc40914982)

[7.3 Show tracks longer than 5 minutes 13](#_Toc40914983)

[7.4 Show musician names and the bands they play in 14](#_Toc40914984)

[7.5 Show average track duration from each album with album name and its ID 14](#_Toc40914985)

[7.6 Show the numer of concerts each tour consists of 15](#_Toc40914986)

[7.7 Show bands with more than 5 tracks composed 15](#_Toc40914987)

[7.8 Show producer IDs, their names and tracks they produced 16](#_Toc40914988)

[7.9 Show bands and the number of concerts they are playing, sorted by the number of concerts descending 17](#_Toc40914989)

[8 Views 18](#_Toc40914990)

[8.1 View of studios with a price per hour smaller than 100 $ 18](#_Toc40914991)

[8.2 View of bands and the number of concerts they are playing, sorted by the number of concerts descending 18](#_Toc40914992)

[8.3 View of number of concerts per tour 19](#_Toc40914993)

[9 Procedures 20](#_Toc40914994)

[9.1 Procedure adding to the database a band that is not present in the database 20](#_Toc40914995)

[9.2 Procedure deleting a record of ID\_CONCERT field equal to concertId from Concert table 21](#_Toc40914996)

[9.3 Procedure deleting a record of ID\_TOUR field equal to tourId from Tour table and concerts from that tour from the Concert table 22](#_Toc40914997)

[10 Functions 24](#_Toc40914998)

[10.1 Get salary of musician with ID\_MUSICIAN = musicianId 24](#_Toc40914999)

[10.2 Get id of the most expensive studio thats within the budget 24](#_Toc40915000)

[11 Cursors 25](#_Toc40915001)

[11.1 Procedure printing band IDs with their names 25](#_Toc40915002)

[11.2 Function returning number of planned concerts (using cursors) 26](#_Toc40915003)

[11.3 Procedure printing producers with the tracks they worked on 27](#_Toc40915004)

[12 Triggers 29](#_Toc40915005)

[12.1 Insert trigger checking, if the start date of a contract is later than the end date 29](#_Toc40915006)

[12.2 Insert trigger updating Album.Number\_of\_tracks field on adding a new track 29](#_Toc40915007)

[12.3 Delete trigger deleting concerts from tour when deleting a tour 30](#_Toc40915008)

# Introduction

This is a database containing information that could be used by a music label to store information about the “assets” and collaborators they possess such as bands, albums, studios and producers. The database was implemented using SQL Anywhere Developer Edition. It contains various procedures, functions and triggers.

# Problem description

Music labels often work with hundreds of bands at once. Over the years of its existence, such a label can accumulate in its portfolio thousands of bands, albums, tours, etc. Therefore, there is a need to create a database to manage these resources efficiently.

# Project assumptions

## Business description.

The record label works with various bands. Each band has a contract with the label. In addition, each band has a certain number of musicians to pay the appropriate remuneration. The label makes money mainly from the sale of albums created by its bands. Each album consists of a number of tracks and a title. The bands work with the producer to create the album. To record an album, a band must have a rented studio.

In addition, the label can make money from organizing concerts in different cities. Each such concert must have a fixed date and the number of tickets available for sale. Concerts can be combined into tours.

In the event of a breakdown, the team may report a need for new equipment.

## Description by method of nouns and verbs.

The team has a signed contract.

The band releases an album.

The producer is working on the album.

The album consists of tracks.

The album is recorded in the studio.

The team has a rented studio.

The band plays a tour.

The tour consists of concerts.

The musician belongs to the band.

The team is requesting equipment.

## Relationships model.

Band

Contract

Album

Musicianian

Tour

Producer

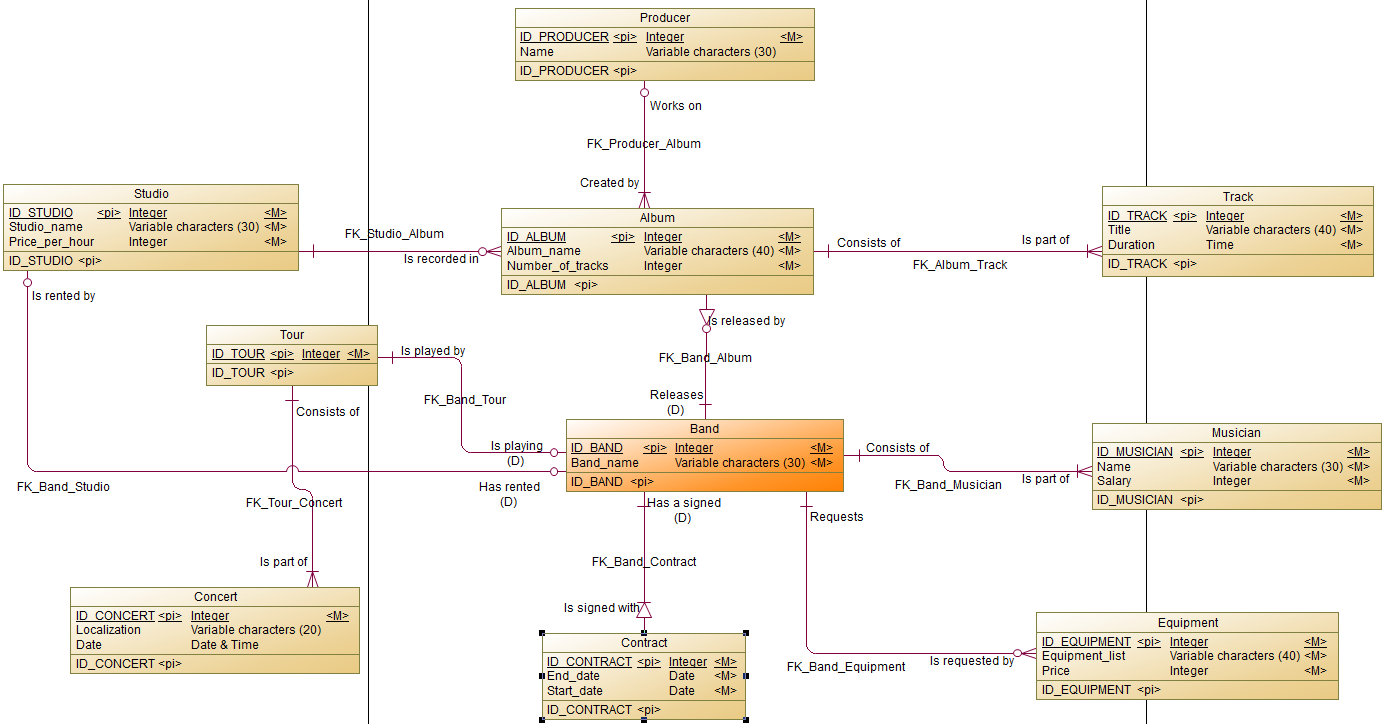
Studio

Track

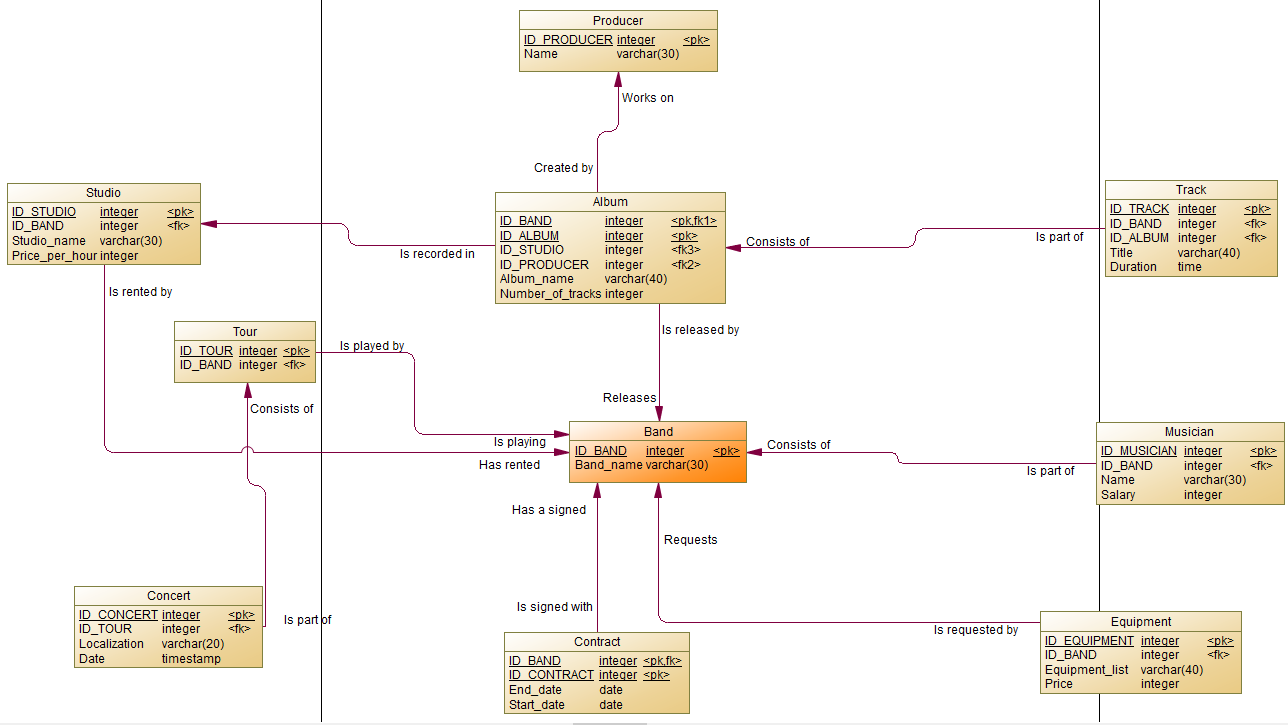
Concert

Equipment

# Conceptual Database Model (CDM)



# Physical Database model (PDM)



# Create the database

## 5.1 Create a database structure.

Database structure creation scripts can be found in music\_label\_db\_create.sql file.

## Data entry.

Example data entry scripts can be found in the data\_insert.sql

## Listings of entered data (alphabetically)

Table Album:

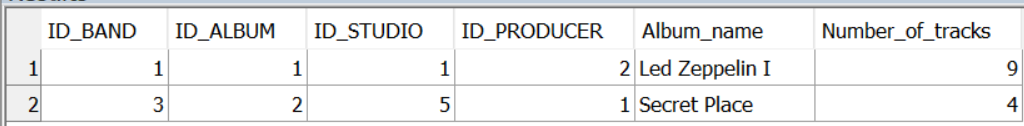


Table Band:

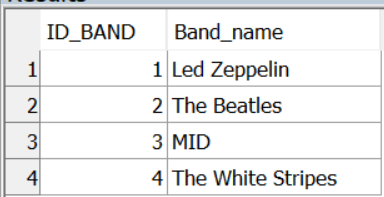


Table Concert:



Table Contract:

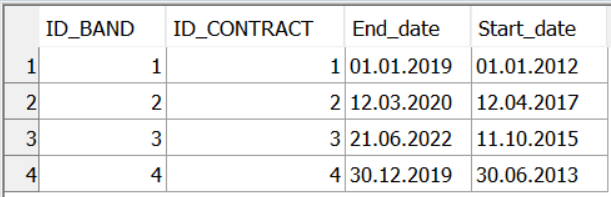


Table Equipment:

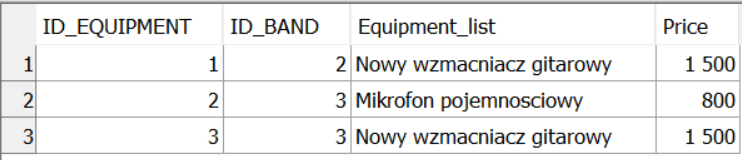


Table Musician:

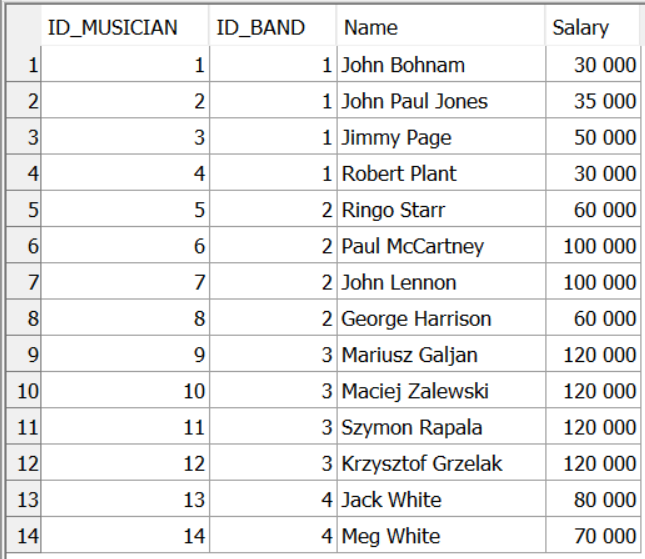


Table Producer:

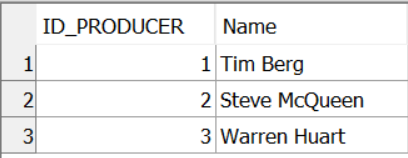


Table Studio:



Table Tour:

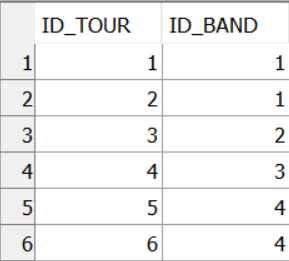


Table Track:



# Queries (sorted by difficulty)

## Show studios cheaper than 100 $.

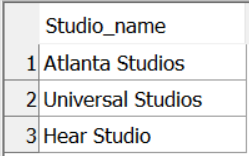
/\* Shows studios where the price per hour is smaller than 100$ \*/

SELECT Studio\_name

    FROM Studio

    WHERE Price\_per\_hour < 100

Result:



## Show Led Zeppelin tours

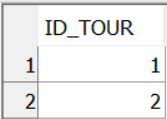
/\* shows Led Zeppelin tours \*/

SELECT ID\_TOUR

    FROM TOUR

    WHERE ID\_BAND = 1

Result:



## Show tracks longer than 5 minutes

/\* show tracks longer than 5 minutes \*/

SELECT Title

    FROM Track

    WHERE Duration >= '00:05'

Result:



## Show musician names and the bands they play in

/\* show musician names and the bands the play in. \*/

SELECT ID\_MUSICIAN, Band.Band\_name, Name

    FROM Musician

    JOIN Band ON  Band.ID\_BAND = Musician.ID\_BAND

Results:



## Show average track duration from each album with album name and its ID

/\* show average track duration from each album with album name and its ID \*/

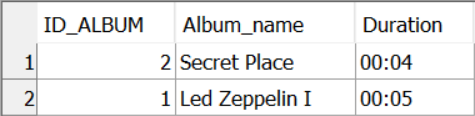
SELECT Album.ID\_ALBUM, Album.Album\_name, convert(time, DATEADD(ms, AVG(DATEDIFF(ms, '00:00:00.000', Duration)), '00:00:00.000')) as Duration

    FROM Track

    JOIN Album ON Album.ID\_ALBUM = Track.ID\_ALBUM

    GROUP BY Album.ID\_ALBUM, Album\_name

Result:



## Show the numer of concerts each tour consists of

/\* 6.6 show the number of concerts a tour consists of \*/

SELECT Tour.ID\_TOUR, count(Concert.ID\_CONCERT) as Number\_of\_concerts

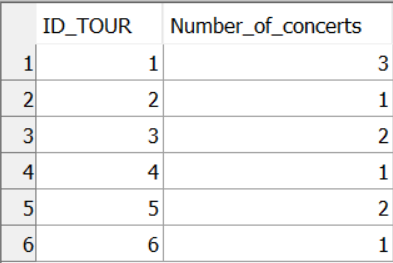
    FROM Tour

    JOIN Concert ON Tour.ID\_TOUR = Concert.ID\_TOUR

    GROUP BY Tour.ID\_TOUR

    ORDER BY Tour.ID\_TOUR ASC

Results:



## Show bands with more than 5 tracks composed

/\* show names of bands with more than 5 tracks together with the numbers of those tracks \*/

SELECT Band.ID\_BAND, Band\_name, count(Track.ID\_BAND) AS number\_of\_tracks

    FROM Band

    JOIN Track ON Track.ID\_BAND = Band.ID\_BAND

    WHERE Band.ID\_BAND IN (

        SELECT ID\_BAND

            FROM Track

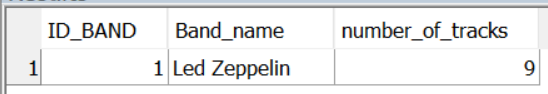
            GROUP BY ID\_BAND

            HAVING  count(\*) > 5

    )

    GROUP BY Band.ID\_BAND, Band\_name;

Results:



## Show producer IDs, their names and tracks they produced

/\* 6.8 show producer IDs, their names and tracks they worked on  \*/

SELECT Producer.ID\_PRODUCER, Name, Track.Title

    FROM Producer, (

        SELECT ID\_PRODUCER, Track.Title

            FROM Album

            JOIN Track ON Album.ID\_ALBUM = Track.ID\_ALBUM

    ) AS Track

    WHERE Producer.ID\_PRODUCER = Track.ID\_PRODUCER

    ORDER BY Producer.ID\_PRODUCER ASC

Results:



## Show bands and the number of concerts they are playing, sorted by the number of concerts descending

/\* 6.9 show bands and the number of their concerts, sorted by the concerts\_number descending \*/

SELECT Band.Band\_name, sum(Concerts.Number\_of\_concerts) AS Concerts

    FROM Band, (

        SELECT Tour.ID\_BAND, Tour.ID\_TOUR, count(Concert.ID\_CONCERT) as Number\_of\_concerts

            FROM Tour

            JOIN Concert ON Tour.ID\_TOUR = Concert.ID\_TOUR

            GROUP BY Tour.ID\_TOUR, Tour.ID\_BAND

            ORDER BY Tour.ID\_TOUR ASC

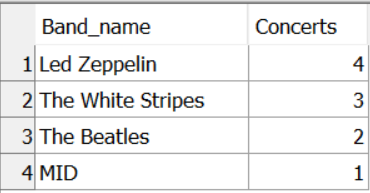
    ) AS Concerts

    WHERE Band.ID\_BAND = Concerts.ID\_BAND

    GROUP BY Band.Band\_name

    ORDER BY Concerts DESC

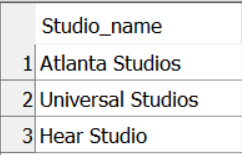
Results:



# Views

## View of studios with a price per hour smaller than 100 $

View:



## View of bands and the number of concerts they are playing, sorted by the number of concerts descending

/\* show bands and the number of their concerts, sorted by the concerts\_number descending \*/

CREATE VIEW Band\_concerts AS

    SELECT Band.Band\_name, sum(Concerts.Number\_of\_concerts) AS Concerts

    FROM Band, (

        SELECT Tour.ID\_BAND, Tour.ID\_TOUR, count(Concert.ID\_CONCERT) as Number\_of\_concerts

            FROM Tour

            JOIN Concert ON Tour.ID\_TOUR = Concert.ID\_TOUR

            GROUP BY Tour.ID\_TOUR, Tour.ID\_BAND

            ORDER BY Tour.ID\_TOUR ASC

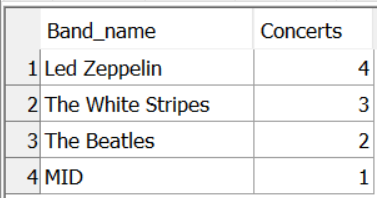
    ) AS Concerts

    WHERE Band.ID\_BAND = Concerts.ID\_BAND

    GROUP BY Band.Band\_name

    ORDER BY Concerts DESC

View:



## View of number of concerts per tour

/\* 7.3  View of number of concerts per tour \*/

CREATE VIEW Concert\_of\_tour AS

    SELECT Tour.ID\_TOUR, count(Concert.ID\_CONCERT) as Number\_of\_concerts

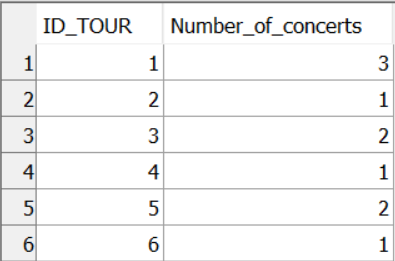
    FROM Tour

    JOIN Concert ON Tour.ID\_TOUR = Concert.ID\_TOUR

    GROUP BY Tour.ID\_TOUR

    ORDER BY Tour.ID\_TOUR ASC

View:



# Procedures

## Procedure adding to the database a band that is not present in the database

/\* Procedure adding to the database a band that is not present in the database \*/

ALTER PROCEDURE "DBA"."addBand"(IN bandId INTEGER , IN bandName VARCHAR(30) )

BEGIN

    IF NOT EXISTS (SELECT Band\_name FROM Band WHERE Band\_name = bandName) THEN

    BEGIN

        INSERT INTO Band VALUES

            (bandId, bandName);

    END

    ELSE BEGIN

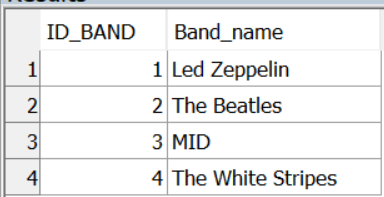
        RAISERROR 99999 'Band already exists'

    END

    ENDIF;

END

Before:



After command ‘CALL "dba".addBand("bandId" = 6, "bandName" = 'The Jets')’:



## Procedure deleting a record of ID\_CONCERT field equal to concertId from Concert table

ALTER PROCEDURE "dba"."deleteConcert"( IN concertId INTEGER )

BEGIN

    IF NOT EXISTS (SELECT ID\_CONCERT FROM Concert WHERE ID\_CONCERT = concertId) THEN

        BEGIN

            RAISERROR 99999 'No record found'

        END

    ELSE

        BEGIN

            DELETE FROM Concert WHERE DBA.Concert.ID\_CONCERT = concertId

        END

    ENDIF

END

Before:



After ‘CALL "dba"."deleteConcert"("concertId" = 4)’:



## Procedure deleting a record of ID\_TOUR field equal to tourId from Tour table and concerts from that tour from the Concert table

ALTER PROCEDURE "dba"."deleteTour"( IN tourId INTEGER )

BEGIN

    IF NOT EXISTS (SELECT ID\_TOUR FROM Tour WHERE ID\_TOUR = tourId) THEN

        BEGIN

            RAISERROR 99999 'No record found'

        END

    ELSE

        BEGIN

            DELETE FROM Concert WHERE DBA.Concert.ID\_TOUR = tourId;

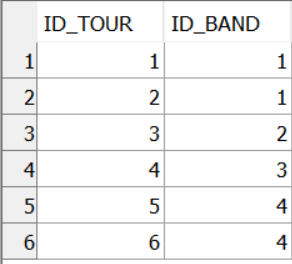
            DELETE FROM Tour WHERE DBA.Tour.ID\_TOUR = tourId;

        END

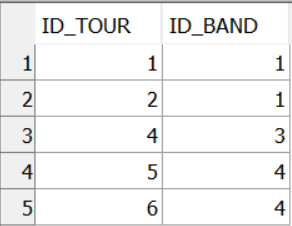
    ENDIF

END

Before:



After:



# Functions

## Get salary of musician with ID\_MUSICIAN = musicianId

/\* 9.1 Get salary of musician with ID\_MUSICIAN = musicianId \*/

ALTER FUNCTION "DBA"."getSalary"( IN musicianId INTEGER )

RETURNS INTEGER

DETERMINISTIC

BEGIN

    DECLARE salary INTEGER;

    DECLARE musician INTEGER;

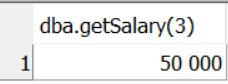
    SET musician = musicianId;

    SET salary = (SELECT Salary FROM Musician WHERE musician = Musician.ID\_MUSICIAN);

    RETURN salary;

END

Results of calling ‘SELECT "dba"."getSalary"(3)’:



## Get id of the most expensive studio thats within the budget

/\* 9.2 Get id of the most expensive studio thats within the budget \*/

ALTER FUNCTION "dba"."studioInBudget"( IN budget INTEGER )

RETURNS INTEGER

DETERMINISTIC

BEGIN

    DECLARE "studioId" INTEGER;

    SET studioId = (SELECT ID\_STUDIO

                        FROM (SELECT TOP 1 ID\_STUDIO, Price\_per\_hour AS Price

                                FROM Studio

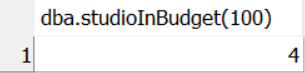
                                WHERE Price\_per\_hour < 100

                                ORDER BY Price DESC) AS StudioPrice );

    RETURN "studioId";

END

Result of ‘SELECT "dba"."studioInBudget"(100)’:



# Cursors

## Procedure printing band IDs with their names

/\* 10.1 Procedure printing band IDs with their names \*/

ALTER PROCEDURE "dba"."showBands"()

BEGIN

    DECLARE i INTEGER;

    DECLARE bandId INTEGER;

    DECLARE bandName VARCHAR(30);

    DECLARE bandCursor DYNAMIC SCROLL CURSOR FOR

        SELECT ID\_BAND, Band\_name FROM Band;

    SET i = 0;

    OPEN bandCursor;

    WHILE i < (SELECT count(ID\_BAND) FROM Band)

    LOOP

        FETCH NEXT bandCursor INTO bandId, bandName;

        MESSAGE 'ID: ', + bandId, + ' Band name: ', + bandName TO CLIENT;

        SET i = i + 1;

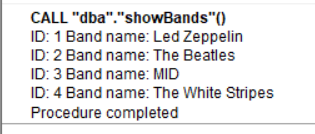
    END LOOP;

    CLOSE bandCursor;

    DEALLOCATE bandCursor;

END

Result:



## Function returning number of planned concerts (using cursors)

/\* 10.2 Function returning number of planned concerts \*/

ALTER FUNCTION "DBA"."numberOfPlannedConcerts"( )

RETURNS INTEGER

DETERMINISTIC

BEGIN

    DECLARE numberOfConcerts INTEGER;

    DECLARE i INTEGER;

    DECLARE tourId INTEGER;

    DECLARE concertsInTour INTEGER;

    DECLARE cursorVar DYNAMIC SCROLL CURSOR FOR

        /\* selects number of concerts for each tour\*/

        SELECT Tour.ID\_TOUR, count(Concert.ID\_CONCERT) as number\_of\_concerts

            FROM Tour

            JOIN Concert ON Tour.ID\_TOUR = Concert.ID\_TOUR

            GROUP BY Tour.ID\_TOUR

            ORDER BY Tour.ID\_TOUR ASC;

    SET numberOfConcerts = 0;

    SET i = 0;

    OPEN cursorVar;

    WHILE i < (SELECT count(Concerts.ID\_TOUR) FROM (

                    /\* counts number of tours in the Concerts table \*/

                    SELECT Tour.ID\_TOUR, count(Concert.ID\_CONCERT) as number\_of\_concerts

                        FROM Tour

                        JOIN Concert ON Tour.ID\_TOUR = Concert.ID\_TOUR

                        GROUP BY Tour.ID\_TOUR ) AS Concerts )

    LOOP

        FETCH NEXT cursorVar INTO tourId, concertsInTour;

        SET numberOfConcerts = numberOfConcerts + concertsInTour;

        SET i = i + 1;

    END LOOP;

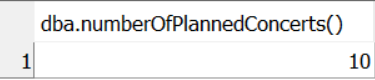
    CLOSE cursorVar;

    DEALLOCATE cursorVar;

    RETURN "numberOfConcerts";

END

Results:



## Procedure printing producers with the tracks they worked on

/\* Procedure printing producers with the tracks they worked on \*/

ALTER PROCEDURE "dba"."showProducersTracks"( )

BEGIN

    DECLARE i INTEGER;

    DECLARE producerId INTEGER;

    DECLARE actProducerId INTEGER;

    DECLARE producerName VARCHAR(30);

    DECLARE trackId INTEGER;

    DECLARE trackTitle VARCHAR(30);

    DECLARE cursorVar DYNAMIC SCROLL CURSOR FOR

        /\* selects producer's ID, Name and the tracks he/she's been working on \*/

        SELECT Producer.ID\_PRODUCER, Name, Track.Title

            FROM Producer, (

                SELECT ID\_PRODUCER, Track.Title

                FROM Album

                JOIN Track ON Album.ID\_ALBUM = Track.ID\_ALBUM

            ) AS Track

            WHERE Producer.ID\_PRODUCER = Track.ID\_PRODUCER

            ORDER BY Producer.ID\_PRODUCER ASC;

    SET actProducerId = 0;

    SET i = 0;

    OPEN cursorVar;

    WHILE i < (SELECT count(ProducentTrack.Title) FROM (

        SELECT Producer.ID\_PRODUCER, Name, Track.Title

            FROM Producer, (

                SELECT ID\_PRODUCER, Track.Title

                FROM Album

                JOIN Track ON Album.ID\_ALBUM = Track.ID\_ALBUM

            ) AS Track

            WHERE Producer.ID\_PRODUCER = Track.ID\_PRODUCER

            ORDER BY Producer.ID\_PRODUCER ASC) AS ProducentTrack)

    LOOP

        FETCH NEXT cursorVar INTO producerId, producerName, trackTitle;

        /\*If we fetched the next producer, we print his name before his tracks \*/

        IF actProducerId != producerId THEN

            BEGIN

                MESSAGE 'Producent: ', + producerName TO CLIENT;

                SET actProducerId = producerId;

            END

        ENDIF;

        MESSAGE trackTitle TO CLIENT;

        SET i = i + 1;

    END LOOP;

    CLOSE cursorVar;

    DEALLOCATE cursorVar;

END

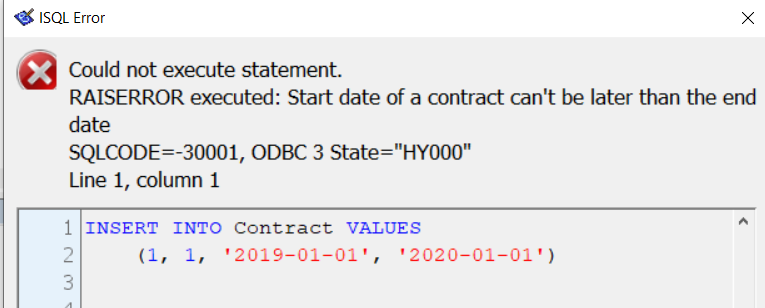
Results of ‘CALL "dba"."showProducersTracks"()’:



# Triggers

## Insert trigger checking, if the start date of a contract is later than the end date

Result:



## Insert trigger updating Album.Number\_of\_tracks field on adding a new track

/\* 11.2 Insert trigger updating Album.Number\_of\_tracks field on adding a new track \*/

CREATE TRIGGER "addTrack" AFTER INSERT

ORDER 1 ON "dba"."Track"

REFERENCING NEW AS newTrack

FOR EACH ROW

BEGIN

    DECLARE numberOfTracks INTEGER;

    SET numberOfTracks = (SELECT Number\_of\_tracks FROM Album WHERE ID\_ALBUM = newTrack.ID\_ALBUM);

    MESSAGE 'Current number of tracks: ', + numberOfTracks TO CLIENT;

    SET numberOfTracks = numberOfTracks + 1;

    UPDATE Album

        SET Number\_of\_tracks = numberOfTracks

        WHERE ID\_ALBUM = newTrack.ID\_ALBUM;

    MESSAGE 'Updated number of tracks: ', + numberOfTracks TO CLIENT;

END

## Delete trigger deleting concerts from tour when deleting a tour

/\* 11.3 Delete trigger deleting concerts from tour when deleting a tour \*/

CREATE TRIGGER "deleteConcertsFromTour" BEFORE DELETE

ORDER 1 ON "dba"."Tour"

REFERENCING OLD AS tour

FOR EACH ROW

BEGIN

    DELETE FROM Concert WHERE Concert.ID\_TOUR = tour.ID\_TOUR;

    MESSAGE 'Deleted concerts from tour with ID ', + tour.ID\_TOUR TO CLIENT;

END