THE MEADOWS

MUSIC AND ARTS FESTIVAL → QUEENS • NEW YORK

Database Design Documentation

Marcus Zimmermann

Table Of Contents

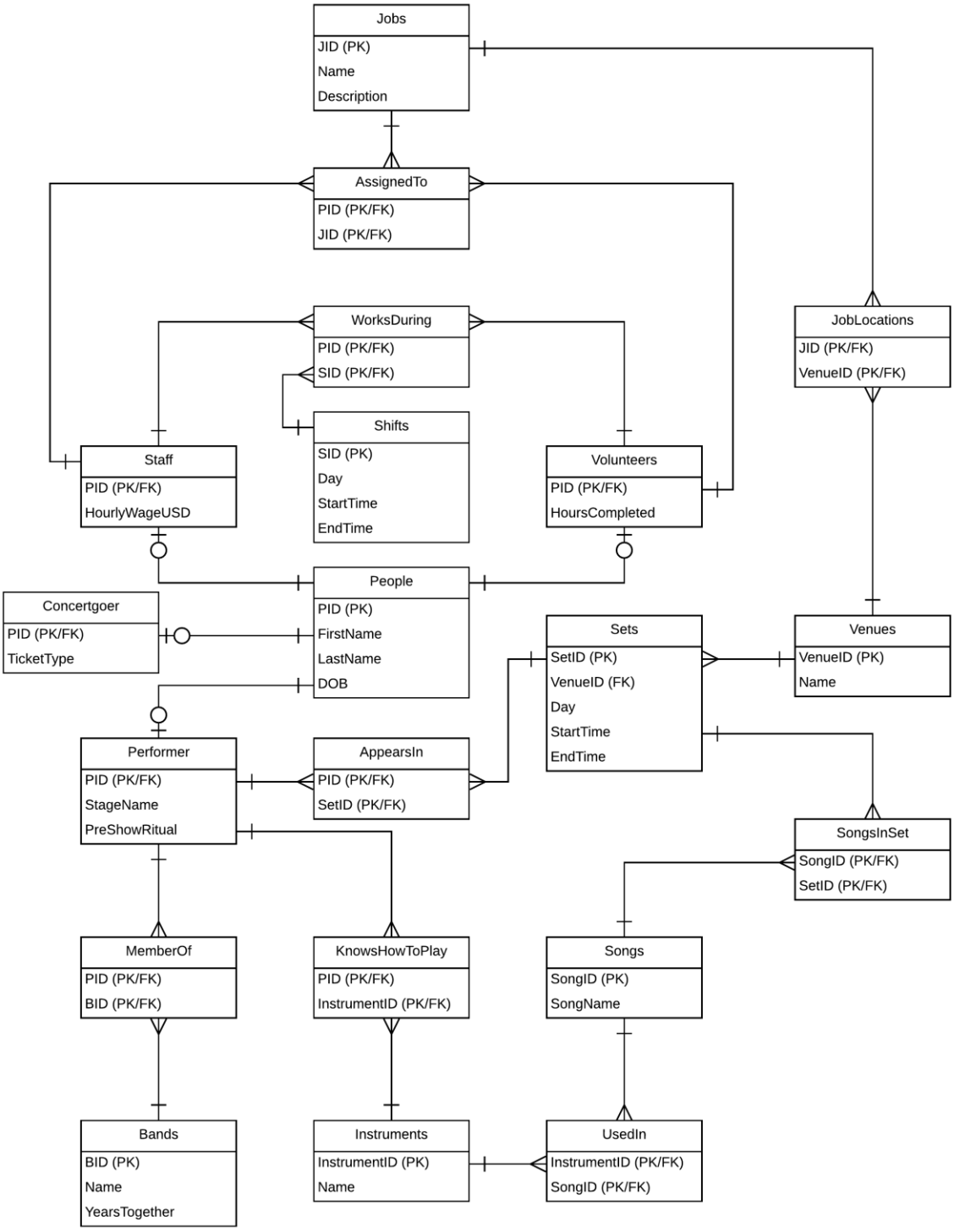
Executive Summary

The Meadows was a two-day, multistage music and arts festival that showcased the very best in hip hop, electronic, indie rock, and more. It featured two impressive lineups, forty artists in total, over the course of a single weekend (October 1st and October 2nd). Unlike most other music festivals, The Meadows had four separate concert venues, allowing multiple artists to perform at the same time. In addition to the many performers, there was also a huge number of staff, volunteers, and of course, concertgoers. Needless to say, there were a lot of people…

So, to keep track of everyone at the concert, a large database had to be designed and implemented, one that kept records of concertgoers, performers, staff, and volunteers. For performers, set times and venue assignments had to be defined and potentially manipulated. Similarly, for staff and volunteers, shift assignements and job assignments had to be defined and potentially manipulated. Lastly, the databse had to be capable of supporting certain views, queries, stored procedures, and triggers, each one providing staff and volunteers with valuable tools and/or important information that kept the The Meadows running smoothly.

This document outlines the design and implementation of The Meadows Database. To start, an ER diagram will be shown as a general overview, followed by create statements for special data types. Then, create table statements will be shown with sample data given for each one. Similarly, view definitions, reports and their queries, stored procedures, and triggers will be shown with sample output given for each one. Lastly, security priviliges, implementation notes, known problems, and future enhancements will be shared.

ER Diagram


Jobs
JID (PK)
Name
Description
AssignedTo
PID (PK/FK)
JID (PK/FK)
WorksDuring
PID (PK/FK)
SID (PK/FK)
Shifts
SID (PK)
Day
StartTime
EndTime
JobLocations
JID (PK/FK)
VenueID (PK/FK)
Staff
PID (PK/FK)
HourlyWageUSD
Volunteers
PID (PK/FK)
HoursCompleted
People
PID (PK)
FirstName
LastName
DOB
Concertgoer
PID (PK/FK)
TicketType
Sets
SetID (PK)
VenueID (FK)
Day
StartTime
EndTime
Venues
VenueID (PK)
Name
Performer
PID (PK/FK)
StageName
PreShowRitual
AppearsIn
PID (PK/FK)
SetID (PK/FK)
SongsInSet
SongID (PK/FK)
SetID (PK/FK)
MemberOf
PID (PK/FK)
BID (PK/FK)
KnowsHowToPlay
PID (PK/FK)
InstrumentID (PK/FK)
Songs
SongID (PK)
SongName
Bands
BID (PK)
Name
YearsTogether
Instruments
InstrumentID (PK)
Name
UsedIn
InstrumentID (PK/FK)
SongID (PK/FK)

Types

dayText: Only Saturday or Sunday can be entered for Day

```
CREATE TYPE dayText AS ENUM ('Saturday', 'Sunday') ;
```

TTText: Only certain ticket types can be entered for TicketType

```
CREATE TYPE TTText AS ENUM ('Saturday GA', 'Sunday GA', '2-Day
GA', 'Saturday VIP', 'Sunday VIP', '2-Day VIP', '2-Day Super
VIP') ;
```

HCText: Only yes or no can be entered for HoursCompleted

```
CREATE TYPE HCText AS ENUM ('yes', 'no') ;
```

Tables

People: The People table has the following attributes: PID, FirstName, LastName, and DOB. These attributes are shared with the subtypes Concertgoers, Performers, Staff, and Volunteers.

Functional Dependencies: PID → FirstName, LastName, DOB

```
CREATE TABLE People (
     PID       SERIAL NOT NULL PRIMARY KEY,
     FirstName TEXT   NOT NULL,
     LastName  TEXT   NOT NULL,
     DOB       DATE   NOT NULL
);
```

	pid integer	firstname text	lastname text	dob date
1	1	Marcus	Zimmermann	1997-01-26
2	2	G	Leaden	1997-02-24
3	3	Marcella	Rose	1996-05-29
4	4	Janine	Lim	1997-09-11
5	5	Lauren	Vecchio	1997-01-02
6	6	Katie	Kilkullen	1997-09-03
7	7	Chancellor	Bennett	1993-04-16
8	8	Nico	Segal	1993-06-25
9	9	Bryson	Tiller	1993-01-02
10	10	Austin	Post	1995-07-04

Concertgoers: Concertgoers are a subtype of People. The Concertgoers table has the following attributes: PID and TicketType.

Functional Dependencies: PID → TicketType

```
CREATE TABLE Concertgoers (
     PID         SERIAL NOT NULL REFERENCES People(PID),
     TicketType TTText NOT NULL
);
```

	pid integer	tickettype tttext
1	1	Sunday GA
2	2	Sunday GA
3	3	Sunday GA
4	4	Sunday GA
5	5	Sunday GA
6	6	2-Day Super VIP

Performers: Performers are a subtype of People. The Performers table has the following attributes: PID, StageName, and PreShowRitual.

Functional Dependencies: PID → StageName, PreShowRitual

```
CREATE TABLE Performers (
     PID            SERIAL NOT NULL REFERENCES People(PID),
     StageName      TEXT,
     PreShowRitual TEXT
);
```

	pid integer	stagename text	preshowritual text
1	7	Chance The Rapper	Eats some Sunday candy
2	8	Donnie Trumpet	Talks to his trumpet
3	9		Looks for Waldo amongst all the concertgoers
4	10	Post Malone	Balls like Iverson
5	11		Admires himself in the mirror for ten minutes
6	12		Puts on an entirely plaid outfit
7	13		Tunes his guitar more times than necessary
8	14		Smokes two packs of cigarette
9	15		Walks around aimlessly
10	16		Prays to Codd
11	17		Prays to Codd

Staff: Staff are a subtype of People. The Staff table has the following attributes: PID and HourlyWageUSD.

Functional Dependencies: PID → HourlyWageUSD

```
CREATE TABLE Staff (
     PID            SERIAL NOT NULL REFERENCES People(PID),
     HourlyWageUSD INT    NOT NULL
);
```

	pid integer	hourlywageusd integer
1	20	10
2	21	11
3	22	11
4	23	11

Volunteers: Volunteers are a subtype of People. The Volunteers table has the following attributes: PID and HoursCompleted.

Functional Dependencies: PID → HoursCompleted

```
CREATE TABLE Volunteers (
     PID             SERIAL NOT NULL REFERENCES People(PID),
     HoursCompleted HCText NOT NULL
);
```

	pid integer	hourscompleted hctext
1	24	no
2	25	no
3	26	no

Venues: The Meadows featured four separate concert venues, each one capable of entertaining huge crowds. Their names were The Meadows, Linden Blvd, Queens Blvd, and Shea. The Venues table has the following attributes: VenueID and Name.

Functional Dependencies: VenueID → Name

```
CREATE TABLE Venues (
     VenueID Serial NOT NULL PRIMARY KEY,
     Name    TEXT   NOT NULL
);
```

	venueid integer	name text
1	1	The Meadows
2	2	Linden Blvd
3	3	Queens Blvd
4	4	Shea

Sets: The full line-up at The Meadows included forty separate performances. So, venue assignments, day assignments, and set times had to be given to each one. The Sets table has the following attributes: SetID, VenueID, Day, StartTime, EndTime.

Functional Dependencies: SetID → VenueID, Day, StartTime, EndTime

```
CREATE TABLE Sets (
     SetID     SERIAL  NOT NULL PRIMARY KEY,
     VenueID   Serial  NOT NULL REFERENCES Venues(VenueID),
     Day       dayText NOT NULL,
     StartTime TIME    NOT NULL,
     EndTime   TIME    NOT NULL
);
```

	setid integer	venueid integer	day daytext	starttime time without time zone	endtime time without time zone
1	1	1	Saturday	15:00:00	15:45:00
2	2	1	Saturday	20:45:00	22:00:00
3	3	2	Sunday	13:15:00	14:00:00
4	4	1	Sunday	15:45:00	16:45:00
5	5	1	Sunday	17:45:00	19:00:00
6	6	4	Sunday	19:00:00	20:00:00
7	7	3	Sunday	20:00:00	21:30:00
8	8	1	Sunday	20:15:00	22:00:00

AppearsIn: To link performers to the sets they appear in, an AppearsIn table had to be created. The AppearsIn table has the following attributes: PID and SetID. These two attributes combine to form a composite key, allowing for a many to many relationship between People (Performers) and Sets. This is necessary because it is possible for a performer to be in muliple sets, and a set can have multiple performers.

Functional Dependencies: SetID → PID

```
CREATE TABLE AppearsIn (
     PID   Serial NOT NULL REFERENCES People(PID),
     SetID Serial NOT NULL REFERENCES Sets(SetID),
     PRIMARY KEY (PID, SetId)
);
```

	pid integer	setid integer
1	10	1
2	18	2
3	16	3
4	17	3
5	9	4
6	7	5
7	8	5
8	12	6
9	13	6
10	14	6
11		

Bands: Because many performers perform in a band together, a Bands table was created to keep track of all bands performing at The Meadows. The Bands table has the following attributes: BID, Name, and YearsTogether.

Functional Dependencies: BID → Name, YearsTogether

```
CREATE TABLE Bands (
     BID            SERIAL NOT NULL PRIMARY KEY,
     Name           TEXT   NOT NULL,
     YearsTogether INT
);
```

	bid integer	name text	yearstogether integer
1	1	The 1975	14
2	2	Chairlift	11

MemberOf: Just like the AppearsIn table, the MemberOf table allows for a many to many relationship between People (Performers) and Bands, as a performer can be a member of multiple bands, and a band can be comprised of multiple performers. The Bands table has the following attributes: PID and BID.

Functional Dependencies: BID → PID

```
CREATE TABLE MemberOf (
     PID Serial NOT NULL REFERENCES People(PID),
     BID Serial NOT NULL REFERENCES Bands(BID),
     PRIMARY KEY (PID, BID)
);
```

	pid integer	bid integer
1	12	1
2	13	1
3	14	1
4	15	1
5	16	2
6	17	2

Instruments: Many instruments are used in each song, let alone each performance. So, an Instruments table was created to keep track of each type of instrument onsite during the two-day music festival. The Instruments table has the following attributes: InstrumentID and Name.

Functional Dependencies: InstrumentID → Name

```
CREATE TABLE Instruments (
     InstrumentId Serial NOT NULL PRIMARY KEY,
     Name          TEXT   NOT NULL
);
```

	instrumentid integer	name text
1	1	Trumpet
2	2	Guitar
3	3	Bass Guitar
4	4	Drums
5	5	Piano

KnowsHowToPlay: The KnowsHowToPlay table allows for a many to many relationship between People (Performers) and Instruments, as a performer can know how to play multiple instruments, and an instrument can be played by multiple performers. The KnowsHowToPlay table has the following attributes: PID and InstrumentID.

Functional Dependencies: InstrumentID → PID

```
CREATE TABLE KnowsHowToPlay (
     PID          Serial NOT NULL REFERENCES People(PID),
     InstrumentID Serial NOT NULL REFERENCES
     Instruments(InstrumentID),
     PRIMARY KEY (PID, InstrumentID)
);
```

	pid integer	instrumentid integer
1	8	1
2	12	2
3	13	2
4	17	2
5	17	4
6	14	3
7	15	4
8	7	5
9	16	5

Songs: Many songs are performed in each set. So, a Songs table was created to keep track of all songs to be performed. The Songs table has the following attributes: SongID and SongName.

Functional Dependencies: SongID → SongName

```
CREATE TABLE Songs (
     SongID   Serial NOT NULL PRIMARY KEY,
     SongName TEXT   NOT NULL
);
```

	songid integer	songname text
1	1	White Iverson Intro
2	2	Too Young
3	3	Go Flex
4	4	Hollywood Dreams Come Down
5	5	Tear$
6	6	Window Shopper
7	7	Money Made Me Do It
8	8	White Iverson
9	9	A Tale of 2 Citiez
10	10	Fire Squad

UsedIn: The UsedIn table allows for a many to many relationship between Instruments and Songs, as an instrument can be used in multiple songs, and a song can have multiple instruments. The UsedIn table has the following attributes: InstrumentID and SongID.

Functional Dependencies: SongID → InstrumentID

```
CREATE TABLE UsedIn (
     InstrumentID Serial NOT NULL REFERENCES
     Instruments(InstrumentID),
     SongID       Serial NOT NULL REFERENCES Songs(SongID),
     PRIMARY KEY (InstrumentID, SongID)
);
```

	instrumentid integer	songid integer
1	1	50
2	1	51
3	1	68
4	1	69
5	1	70
6	1	71
7	2	74
8	2	75
9	2	76
10	2	78

SongsInSet: The SongsInSet table allows for a many to many relationship between Songs and Sets, as an song can be used in multiple sets, and a set can have multiple songs. The SongsInSet table has the following attributes: SongID and SetID.

Functional Dependencies: SetID → SongID

```
CREATE TABLE SongsInSet (
     SongID Serial NOT NULL REFERENCES Songs(SongID),
     SetID  Serial NOT NULL REFERENCES Sets(SetID),
     PRIMARY KEY (SongID, SetID)
);
```

	songid integer	setid integer
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1
9	9	2
10	10	2

Shifts: Staff and Volunteers are assigned different shifts. More specifcally, they are assigned an early Saturday shift, a late Saturday shift, an early Sunday shift, and/or a late Sunday shift. The Shifts table has the following attributes: SID, Day, StartTime, and EndTime.

Functional Dependencies: SID → Day, StartTime, EndTime

```
CREATE TABLE Shifts (
     SID       Serial  NOT NULL PRIMARY KEY,
     Day       DayText NOT NULL,
     StartTime TIME    NOT NULL,
     EndTime   TIME    NOT NULL
);
```

	sid integer	day daytext	starttime time without time zone	endtime time without time zone
1	1	Saturday	06:00:00	15:00:00
2	2	Saturday	15:00:00	00:00:00
3	3	Sunday	06:00:00	15:00:00
4	4	Sunday	15:00:00	00:00:00

WorksDuring: The WorksDuring table allows for a many to many relationship between People (Staff and Volunteers) and Shifts, as a person can be assigned multiple shifts, and a shift can be assigned to multiple people. The WorksDuring table has the following attributes: PID and SID.

Functional Dependencies: SID → PID

```
CREATE TABLE WorksDuring (
     PID Serial NOT NULL REFERENCES People(PID),
     SID Serial NOT NULL REFERENCES Shifts(SID),
     PRIMARY KEY (PID, SID)
);
```

	pid integer	sid integer
1	20	1
2	21	2
3	22	3
4	23	4
5	24	1
6	25	1
7	26	3

Jobs: During the early and late shifts of Saturday and Sunday, Staff and Volunteers are assigned different jobs. The Jobs table has the following attributes: JID, Name, Description.

Functional Dependencies: JID → Name, Description

```
CREATE TABLE Jobs (
     JID         Serial NOT NULL PRIMARY KEY,
     Name        TEXT   NOT NULL,
     Description TEXT   NOT NULL
);
```

	jid integer	name text	description text
1	1	Water Filling Station	Fill the water bottles of thirsty concertgoers
2	2	Catering	Serve food to hungry concertgoers
3	3	Trash	Keep our venues clean by picking up unwanted litter
4	4	VIP Set-Up and Maintenance	Set-up VIP lounges and keep them clean
5	5	Decorations	Set up galleries for our sponsored artists and decorate the venues

AssignedTo: The AssignedTo table allows for a many to many relationship between People (Staff and Volunteers) and Jobs, as a person can be assigned multiple jobs, and a job can be assigned to multiple people . The AssignedTo table has the following attributes: PID and JID.

Functional Dependencies: JID → PID

```
CREATE TABLE AssignedTo (
     PID Serial NOT NULL REFERENCES People(PID),
     JID Serial NOT NULL REFERENCES Jobs(JID),
     PRIMARY KEY (PID, JID)
);
```

	pid integer	jid integer
1	20	4
2	21	5
3	22	4
4	23	1
5	24	3
6	25	2
7	26	2

JobLocations: The JobLocations table allows for a many to many relationship between Jobs and Venues, as a job can be in multiple venues, and a venue can have multiple jobs . The JobLocations table has the following attributes: JID and VenueID.

Functional Dependencies: VenueID → JID

```
CREATE TABLE JobLocations (
     JID     Serial NOT NULL REFERENCES Jobs(JID),
     VenueID Serial NOT NULL REFERENCES Venues(VenueID),
     PRIMARY KEY (JID, VenueID)
);
```

	jid integer	venueid integer
1	1	1
2	1	2
3	1	3
4	1	4
5	2	1
6	2	2
7	2	3
8	2	4
9	3	1
10	3	2

Views

ConcertgoerInfo: There are many situations in which it would be benefical for Staff and Volunteers to have a view of all concertgoer information. Consider the following examples: a concertgoer who bought a general admission ticket attempts to enter one of the VIP lounges, a concertgoer tries to pass as 21 to buy alcohol, and a concertgoer tries to enter The Meadows for a second day despite purchasing only a single day ticket.

```
CREATE OR REPLACE VIEW ConcertgoerInfo AS
SELECT p.PID, FirstName, LastName, DOB, TicketType
FROM People p INNER JOIN Concertgoers c ON p.PID = c.PID;
```

	pid integer	firstname text	lastname text	dob date	tickettype tttext
1	1	Marcus	Zimmermann	1997-01-26	Sunday GA
2	2	G	Leaden	1997-02-24	Sunday GA
3	3	Marcella	Rose	1996-05-29	Sunday GA
4	4	Janine	Lim	1997-09-11	Sunday GA
5	5	Lauren	Vecchio	1997-01-02	Sunday GA
6	6	Katie	Kilkullen	1997-09-03	2-Day Super VIP

PerformerShowtimes: As performers finish their set, it's important to know who is on stage next. The PerformerShowtimes view allows Staff and Volunteers to view the full lineup.

```
CREATE OR REPLACE VIEW PerformerShowtimes AS
SELECT p.PID, FirstName, LastName, Day, Name AS Venue,
StartTime, EndTime
FROM People p INNER JOIN Performers pe ON p.PID = pe.PID
  INNER JOIN AppearsIn a ON p.PID     = a.PID
  INNER JOIN Sets s      ON a.SetID   = s.SetID
  INNER JOIN Venues v    ON s.VenueID = v.VenueID;
```

	pid integer	firstname text	lastname text	day daytext	venue text	starttime time without time zone	endtime time without time zone
1	10	Austin	Post	Saturday	The Meadows	15:00:00	15:45:00
2	18	Jermaine	Cole	Saturday	The Meadows	20:45:00	22:00:00
3	16	Caroline	Polacheck	Sunday	Linden Blvd	13:15:00	14:00:00
4	17	Patrick	Wimberly	Sunday	Linden Blvd	13:15:00	14:00:00
5	9	Bryson	Tiller	Sunday	The Meadows	15:45:00	16:45:00
6	7	Chancellor	Bennett	Sunday	The Meadows	17:45:00	19:00:00
7	8	Nico	Segal	Sunday	The Meadows	17:45:00	19:00:00
8	12	Matthew	Healy	Sunday	Shea	19:00:00	20:00:00
9	13	Adam	Hann	Sunday	Shea	19:00:00	20:00:00
10	14	Ross	Macdonald	Sunday	Shea	19:00:00	20:00:00
11	15	George	Daniel	Sunday	Shea	19:00:00	20:00:00

StaffSchedule: The StaffSchedule view allows staff and volunteers to see what day and time every staff member is on duty.

```
CREATE OR REPLACE VIEW StaffSchedule AS
SELECT p.PID, FirstName, LastName, Day, StartTime, EndTime, Name
FROM People p INNER JOIN Staff s ON p.PID = s.PID
  INNER JOIN WorksDuring w ON p.PID = w.PID
  INNER JOIN Shifts sh     ON w.SID = sh.SID
  INNER JOIN AssignedTo a  ON p.PID = a.PID
  INNER JOIN Jobs j        ON a.JID = j.JID;
```

	pid integer	firstname text	lastname text	day daytext	starttime time without time zone	endtime time without time zone	name text
1	20	Tom	Russel	Saturday	06:00:00	15:00:00	VIP Set-Up and Maintenance
2	21	Jordan	Wolowitz	Saturday	15:00:00	00:00:00	Decorations
3	22	Jennifer	Styles	Sunday	06:00:00	15:00:00	VIP Set-Up and Maintenance
4	23	Alex	Joffe	Sunday	15:00:00	00:00:00	Water Filling Station

VolunteerSchedule: The VolunteerSchedule view allows staff and volunteers to see what day and time every volunteer is on duty.

```
CREATE OR REPLACE VIEW VolunteerSchedule AS
SELECT p.PID, FirstName, LastName, Day, StartTime, EndTime, Name
FROM People p INNER JOIN Volunteers v ON p.PID = v.PID
  INNER JOIN WorksDuring w ON p.PID = w.PID
  INNER JOIN Shifts sh     ON w.SID = sh.SID
  INNER JOIN AssignedTo a  ON p.PID = a.PID
  INNER JOIN Jobs j        ON a.JID = j.JID;
```

	pid integer	firstname text	lastname text	day daytext	starttime time without time zone	endtime time without time zone	name text
1	24	Jack	Black	Saturday	06:00:00	15:00:00	Trash
2	25	Kyle	Gass	Saturday	06:00:00	15:00:00	Catering
3	26	Dave	Grohl	Sunday	06:00:00	15:00:00	Catering

Reports And Interesting Queries

Preshow rituals of all peformers in a band

```
SELECT p.PID, FirstName, LastName, b.BID, Name, PreShowRitual
FROM People p INNER JOIN Performers pe ON p.PID = pe.PID
     INNER JOIN MemberOF m ON pe.PID = m.PID
     INNER JOIN Bands b    ON m.BID  = b.BID;
```

Preshow Rituals of The 1975 and Chairlift

	pid integer	firstname text	lastname text	bid integer	name text	preshowritual text
1	12	Matthew	Healy	1	The 1975	Puts on an entirely plaid outfit
2	13	Adam	Hann	1	The 1975	Tunes his guitar more times than necessary
3	14	Ross	Macdonald	1	The 1975	Smokes two packs of cigarette
4	15	George	Daniel	1	The 1975	Walks around aimlessly
5	16	Caroline	Polacheck	2	Chairlift	Prays to Codd
6	17	Patrick	Wimberly	2	Chairlift	Prays to Codd

Stored Procedures

GetSongsAfter: Backstage, staff and volunteers are busy tuning guitars, adjusting lights, conducting sound checks, and doing much more. To properly prepare for the rest of a set, it helps to know what songs are coming up next. By inputing SongID and SetID, GetSongsAfter returns all songs following a song of SongID within a set of SetID.

```
CREATE OR REPLACE FUNCTION GetSongsAfter(INT, INT, REFCURSOR)
RETURNS refcursor AS $$
DECLARE
     iSongID INT := $1;
     iSetID INT := $2;
     result REFCURSOR := $3;
BEGIN
  OPEN result FOR
     WITH RECURSIVE SongsAfter(SongID, SongName) AS (
          SELECT s.SongID, s.SongName
          FROM Songs s
          INNER JOIN SongsInSet sis ON s.SongID = sis.SongID
          WHERE s.SongID=iSongID AND sis.SetID = iSetID
               UNION
          SELECT sa.SongID+1, s.SongName
          FROM SongsAfter sa, Songs s
          INNER JOIN SongsInSet sis ON s.SongID = sis.SongID
          WHERE s.songID=sa.SongID+1 AND sis.SetID = iSetID
     ) SELECT * FROM SongsAfter;
  RETURN result;
END; $$ LANGUAGE plpgsql;
SELECT GetSongsAfter(43, 4, 'ref');
FETCH ALL FROM ref;
```

	songid integer	songname text
1	43	Sorry Not Sorry
2	44	Rambo
3	45	502 Come Up
4	46	For However Long
5	47	The Sequence
6	48	Been That Way
7	49	Don't

LongestSet: This stored procedure returns the set or sets with the longest runtime.

```
CREATE OR REPLACE FUNCTION LongestSet(REFCURSOR) RETURNS
refcursor AS

$body$ DECLARE

     duration TIME := ( SELECT MAX(y.SetDuration)

       FROM (SELECT SetID, (EndTime - StartTime)AS SetDuration

            FROM Sets

            GROUP BY SetID

            ORDER BY SetDuration DESC LIMIT 1) y);

     result REFCURSOR := $1;

BEGIN

OPEN result FOR

     SELECT SetID, StartTime, EndTime, (EndTime - StartTime)AS
SetDuration

     FROM Sets

     WHERE (EndTime - StartTime) = duration;

  RETURN result;

END; $body$ LANGUAGE plpgsql;

SELECT LongestSet('ref');

FETCH ALL FROM ref;
```

	setid integer	starttime time without time zone	endtime time without time zone	setduration interval
1	8	20:15:00	22:00:00	01:45:00

MostSongs: This stored procedure returns the set or sets with the greatest number of songs being performed.

```
CREATE OR REPLACE FUNCTION MostSongs(REFCURSOR) RETURNS
refcursor AS

$body$ DECLARE

     num INT := ( SELECT MAX(y.SongTotal)

       FROM (SELECT s.SetID, COUNT(SongID)AS SongTotal

            FROM Sets s INNER JOIN SongsInSet sis ON s.SetID =
sis.SetID

            GROUP BY s.SetID) y);

     result REFCURSOR := $1;

BEGIN

OPEN result FOR

     SELECT s.SetID, s.StartTime, s.EndTime, COUNT(SongID)AS
SongTotal

     FROM Sets s INNER JOIN SongsInSet sis ON s.SetID =
sis.SetID

     GROUP BY s.SetID

     HAVING COUNT(s.SetID) = num;

  RETURN result;

END; $body$ LANGUAGE plpgsql;

SELECT MostSongs('ref');

FETCH ALL FROM ref;
```

	setid integer	starttime time without time zone	endtime time without time zone	songtotal bigint
1	5	17:45:00	19:00:00	24

MostPlayedInstrument: This stored procedure returns the instrument or instruments played by the greatest number of performers.

```
CREATE OR REPLACE FUNCTION MostPlayedInstrument(REFCURSOR)
RETURNS refcursor AS

$body$ DECLARE

     num INT := ( SELECT MAX(y.HowManyCanPlay)

       FROM (SELECT i.InstrumentID, COUNT(p.PID)AS
HowManyCanPlay

       FROM Instruments i INNER JOIN KnowsHowToPlay khtp ON
i.InstrumentID = khtp.InstrumentID

       INNER JOIN Performers p ON khtp.PID = p.PID

       GROUP BY i.InstrumentID) y);

     result REFCURSOR := $1;

BEGIN

OPEN result FOR

     SELECT i.InstrumentID, i.Name, COUNT(p.PID)AS
HowManyCanPlay

     FROM Instruments i INNER JOIN KnowsHowToPlay khtp ON
i.InstrumentID = khtp.InstrumentID

     INNER JOIN Performers p    ON khtp.PID =       p.PID

     GROUP BY i.InstrumentID

     HAVING COUNT(p.PID) = num;

  RETURN result;

END; $body$ LANGUAGE plpgsql;

SELECT MostPlayedInstrument('ref');

FETCH ALL FROM ref;
```

	instrumentid integer	name text	howmanycanplay bigint
1	2	Guitar	3
2	5	Piano	3

UpdateHoursCompleted: Volunteers who complete their work shift are given a full ticket refund and free merchandise. So, as soon as they are scheduled to work a certain shift, their HoursCompleted status is changed to yes.

```
CREATE OR REPLACE FUNCTION UpdateHoursCompleted() RETURNS
TRIGGER AS $$
BEGIN
     IF new.PID IS NOT NULL
     AND (SELECT HoursCompleted
          FROM Volunteers
          WHERE PID = new.PID) = 'no'
     THEN
          UPDATE Volunteers
          SET HoursCompleted = 'yes'
          WHERE PID = new.PID;
     END IF;
     RETURN new;
END; $$ LANGUAGE plpgsql;
```

Triggers

UpdateHoursCompleted: This trigger triggers the function UpdateHoursCompleted.

```
CREATE TRIGGER UpdateHoursCompleted
AFTER INSERT ON WorksDuring
FOR EACH ROW
EXECUTE PROCEDURE updateHoursCompleted();
```

Security

Administrators: Administrators are granted the full range of capabilities on all tables within the database.

```
CREATE ROLE Admin;

GRANT ALL ON ALL TABLES

IN SCHEMA PUBLIC

TO Admin;
```

Staff: Staff members can select, insert, and update all tables within the database.

```
CREATE ROLE Staff;

GRANT SELECT, INSERT, UPDATE ON ALL TABLES

IN SCHEMA PUBLIC

TO Staff;
```

Volunteers: Volunteers can select all tables within the database.

```
CREATE ROLE Volunteers;

GRANT SELECT ON ALL TABLES

IN SCHEMA PUBLIC

TO Volunteers;
```

Implementation Notes

Songs are meticulously entered into the Songs table in the order in which they are performed. This allows us to recursively select all songs that come after one song in a set.

Known Problems and Future Enhancements

Songs are not easily moved in the databse. This is because SongID is a serial, autoincrementing integer. A future enhancement might be the ability to insert, update, and delete SongIDs while automatically changing all SongIDs that come after by an increment of one. This would allow for the order in which songs were entered to remain as it should.

While it is possible to see what job a staff member or volunteer is assigned to, it is not clear as to which venue the job is being done at. This is because the same job can be done at muliple venues.

For future implementations of the database, it may be beneficial to add more information about each venue. For example, information about available food stands, merchandice booths, emergency care centers, and other amenities could all be incorporated.