//notes

my database is located in the ***schema*** attendants (`SET search_path TO attendants;`)
currently there are 6 tables;
events, guests, locations, contributions, guest_list, and data_schedual

and two views;
schedule;
guest _total_ contributions (will be expanded)

all foreign keys have set references, (contribution has two, and checks if “is_event” is true to make sure an event id is listed.) the event checks to see if they begin and end time make sense. primary keys are set to ids for all tables.

guest list will be populated produced from a added contribution of type is_event =true, when this occurs, it should create a ticket number and add the gust and the event in question to the guest_list database

schedule is a view giving you the date of event, the name of the event and the location name. (will add “attaining count”/attend count)

why “data_schedual” the issue is, some events will use more than one space, some locations may be hosting more than one oven concurrently. thus a datatable must be constructed from their views can be easily made. however, each event that is made will have to also have an insert in their if it's to make it on the schedule.

currently no functions work, and triggers have yet to be added.

the information is organized in a way which would reflect how it will be used, and also the area of discourse, as stated in the assignment part the organization needs to keep things separated and many situations occur which may be outside of normal “routine” thus, the tables are split in ways which allow for information to more accurately be deployed. currently cascade constraints need to be add.

// rules

cascade rules.

insert or update or delete

table_name

create rule “”

()

insert into guest_list (frkey_id_guest, frkey_id_event)

select frkey_id_guest, frkey_id_event from contributions
where id_contri=$in_id_contri

end;
$ticketnum$ LANGUAGE plpgsql;

create table contribution (id_c char(999) constraint id_ckey Primary key, c_date date not null, c_ammount decimal (18,4), c_type char(10), c_guest_id char(999)

create type g_contrabution as
(g_toalcontbution decimal(18,4),
g_toalcontbution_event decimal (18,4),
g_totalcontrbution_donation decimal (18,4),
g_eventattended char(50)
);

update guest_list set frkey_id_guest=,

frkey_id_guest gl_frkey_id_event= ,

frkey_id_event from contributions where id_contri=$in_id_contri

((cint+eint+gint),
)
create view schedule select events.name as Event, events.begin_event,
events.end_event, location.name as Location,

declare
frkey_id_guest
frkey_id_event

$frkey_id_guest = select frkey_id_guest from contributions where
id_contri=$in_id_contri
$frkey_id_event = select frkey_id_event from contributions where
id_contri=$in_id_contri

select row_number() OVER (order by table(collom)) des/asnd) as

row_number()

CREATE FUNCTION foo(a int, b int DEFAULT 2, c int DEFAULT 3) RETURNS int
Extending SQL 966 LANGUAGE SQL AS $$ SELECT $1 + $2 + $3; $$; SELECT foo(10,
20, 30); foo ----- 60 (1 row) SELECT foo(10, 20); foo ----- 33 (1 row) SELECT foo(10);
foo ----- 15 (1 row) SELECT foo(); -- fails since there is no default for the first
argument ERROR: function foo() does not exist

//log function for users
 need to be end in a select, or preform baisc proess. insert etc.

CREATE FUNCTION clean_emp() RETURNS void AS ' DELETE FROM emp WHERE
salary < 0; ' LANGUAGE SQL; Extending SQL 959 SELECT clean_emp();
CREATE SCHEMA ***schema_name*** [AUTHORIZATION ***user_name***] [
schema_element [...]]
CREATE SCHEMA AUTHORIZATION ***user_name*** [***schema_element*** [...]]

    CREATE VIEW guest_list AS
        SELECT guest_id, event_id, g_name, attend

ALTER TABLE ADD

ALTER TABLE Price ADD CONSTRAINT CK_Price_Current_vs_Original CHECK (CurrentPrice <= OriginalPrice);

create table events,create table guests,create table contributions, create locations table;

alter table events

current_user	na me

37.4. Query Language (SQL) Functions
37.4.1. Arguments for SQL Functions
37.4.2. SQL Functions on Base Types
37.4.3. SQL Functions on Composite Types
37.4.4. SQL Functions with Output Parameters
37.4.5. SQL Functions with Variable Numbers of Arguments
37.4.6. SQL Functions with Default Values for Arguments
37.4.7. SQL Functions as Table Sources
37.4.8. SQL Functions Returning Sets
37.4.9. SQL Functions Returning TABLE
37.4.10. Polymorphic SQL Functions
37.4.11. SQL Functions with Collations
37.5. Function Overloading
37.6. Function Volatility Categories
37.7. Procedural Language Functions
37.8. Internal Functions
37.9. C-Language Functions
37.9.1. Dynamic Loading
37.9.2. Base Types in C-Language Functions
37.9.3. Version 1 Calling Conventions
37.9.4. Writing Code
37.9.5. Compiling and Linking Dynamically-loaded Functions
37.9.6. Composite-type Arguments
37.9.7. Returning Rows (Composite Types)
37.9.8. Returning Sets
37.9.9. Polymorphic Arguments and Return Types
37.9.10. Transform Functions
37.9.11. Shared Memory and LWLocks

37.9.12. Using C++ for Extensibility

select count(contributions.frkey_id_guest), guests.name,
sum(contributions.c_amount) from guests join contributions on guests.id_guest =
contributions.frkey_id_guest group by guests.name having count(frkey_id_guest) >1;

create view count(*) as names, guests.name, sum(contributions.c_amount) from guests
where guests.name in (select guests.name from guests where id_guest in (select
id_guest from)join guests on contributions.fkey_id_guest = guests.id_guest)

create view count(*) as
select names, guests.name, sum(contributions.c_amount) from guests natural join
contributions on contributions.fkey_id_guest= guest.id_guest group by guests.name
having count(fkey_id_guest not null) >1;