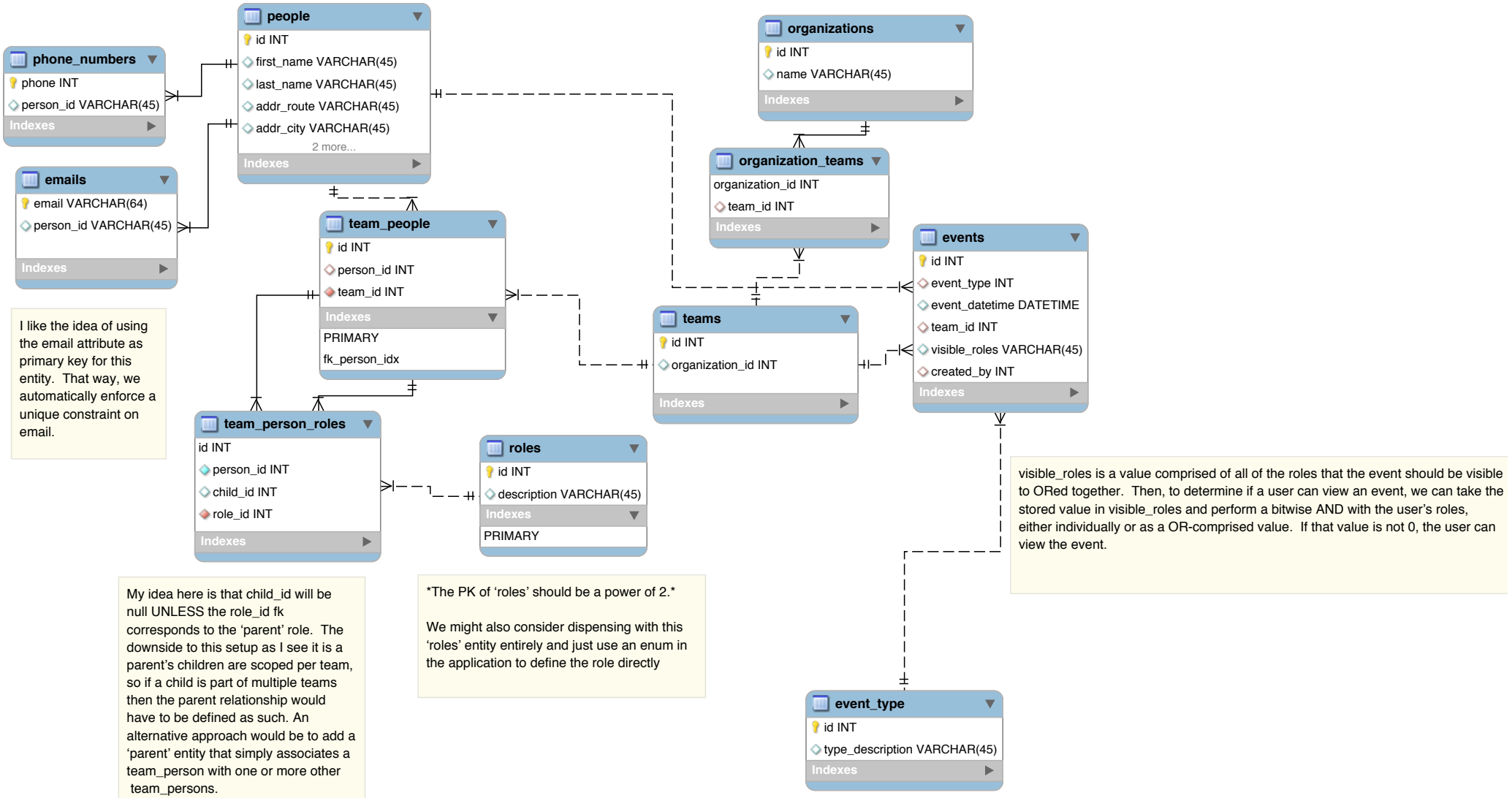
people
id INT
first_name VARCHAR(45)
last_name VARCHAR(45)
addr_route VARCHAR(45)
addr_city VARCHAR(45)
2 more...
Indexes
phone_numbers
phone INT
person_id VARCHAR(45)
Indexes
emails
email VARCHAR(64)
person_id VARCHAR(45)
Indexes
I like the idea of using the email attribute as primary key for this entity. That way, we automatically enforce a unique constraint on email.
team_people
id INT
person_id INT
team_id INT
Indexes
PRIMARY
fk_person_idx
team_person_roles
id INT
person_id INT
child_id INT
role_id INT
Indexes
My idea here is that child_id will be null UNLESS the role_id fk corresponds to the 'parent' role. The downside to this setup as I see it is a parent's children are scoped per team, so if a child is part of multiple teams then the parent relationship would have to be defined as such. An alternative approach would be to add a 'parent' entity that simply associates a team_person with one or more other team_persons.
roles
id INT
description VARCHAR(45)
Indexes
PRIMARY
The PK of 'roles' should be a power of 2.
We might also consider dispensing with this 'roles' entity entirely and just use an enum in the application to define the role directly
organizations
id INT
name VARCHAR(45)
Indexes
organization_teams
organization_id INT
team_id INT
Indexes
teams
id INT
organization_id INT
Indexes
events
id INT
event_type INT
event_datetime DATETIME
team_id INT
visible_roles VARCHAR(45)
created_by INT
Indexes
visible_roles is a value comprised of all of the roles that the event should be visible to ORed together. Then, to determine if a user can view an event, we can take the stored value in visible_roles and perform a bitwise AND with the user's roles, either individually or as a OR-comprised value. If that value is not 0, the user can view the event.
event_type
id INT
type_description VARCHAR(45)
Indexes