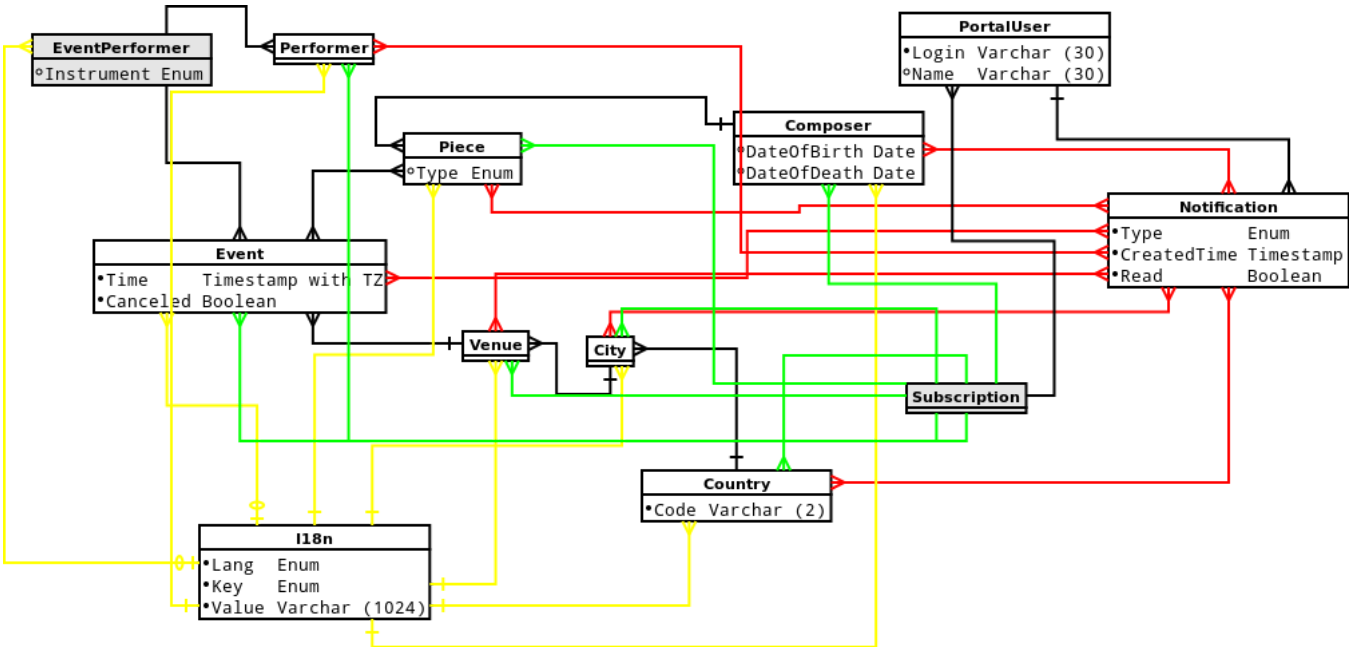
База концертов классической музыки

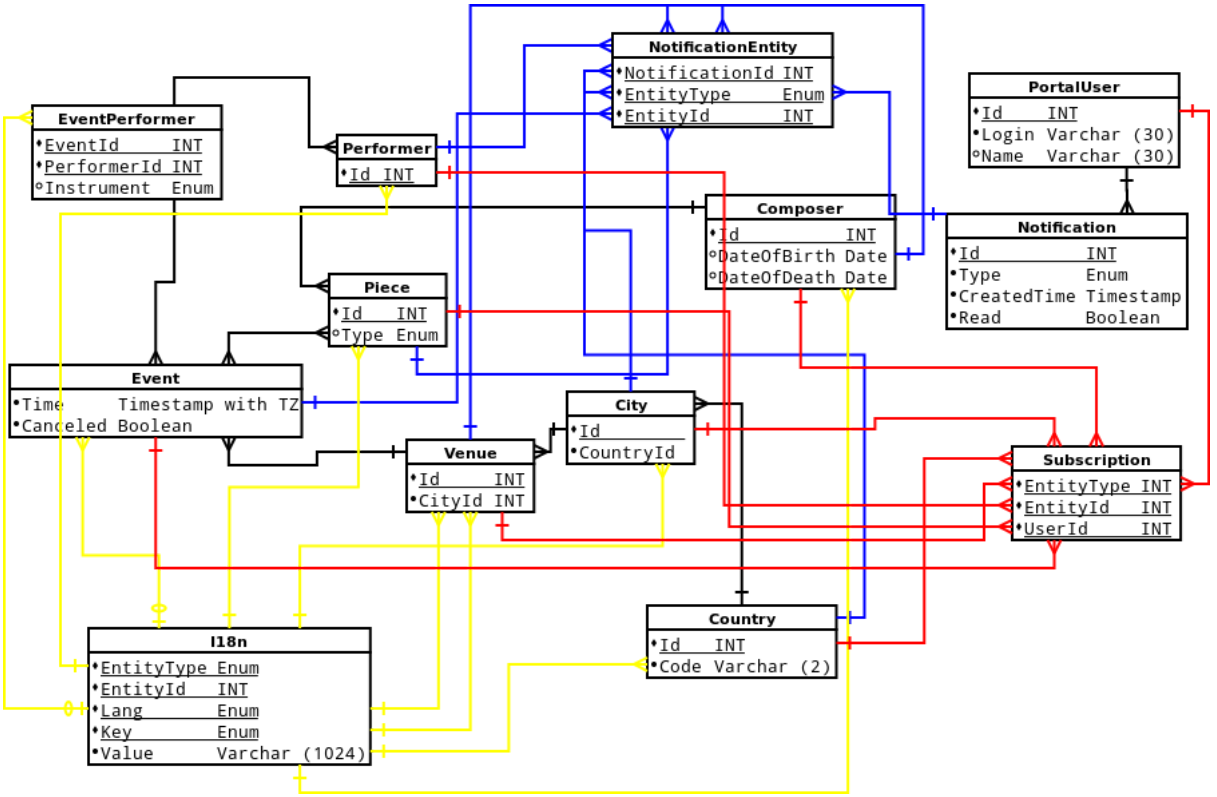
Курсовой проект по предмету “Базы данных”. Университет ИТМО, Январь 2016.

Выполнил Георгий Агапов, группа M3437

ERM



PDM



Описание сущностей

Сущность	Описание
Composer	Композитор
Piece	Музыкальное произведение
Venue	Концертная площадка (театр, концертный зал)
Event	Событие (концерт)
City	Город
Country	Страна
PortalUser	Пользователь сервиса (портала)

Notification Уведомления для пользователя (при появлении нового события, отмене старого)
Subscription Подписки пользователя (на сущности, для которых пользователь хочет получать уведомления)

На ERM и PDM диаграмах для всех сущностей указаны поля, которые они содержат.

Кроме того большинство сущностей содержат переводимые поля, которые хранятся в “общей” сущности I18n.

Рассмотрим поля таблицы I18n:

- EntityType :: Enum (Composer, Piece, Venue, City, Country, Event, EventPerformer) - тип сущности, для которой хранится перевод
- EntityId :: Int - Id сущности (предполагается, что у сущности должен быть суррогатный целочисленный ключ)
- Key :: Enum (Name, Role, Address) - ключ, к сущности может быть привязано до нескольких пар ключ-значение
- Lang :: Enum (ru, en, ge) - язык
- Value :: String - перевод

Сущности неявно ссылаются на таблицу I18n. В качестве EntityType используется название таблицы сущности, в качестве EntityId - ключ. В следующей таблице перечисленны все поля, хранимые в I18n:

Сущность	**Key**	**Описание**
Composer	Name	Имя, фамилия композитора
Piece	Name	Название произведения
Venue	Name	Название площадки (театр, концертный зал)
Venue	Address	Адрес площадки (театр, концертный зал)
Event	Name	Название события (опционально)
EventPerformer	Role	Роль исполнителя в конкретном событии (опционально)
City	Name	Название города
Country	Name	Название страны

SQL

Tables

```
CREATE TABLE Composer (
  Id SERIAL NOT NULL PRIMARY KEY,
  DateOfBirth DATE,
  DateOfDeath DATE
);

CREATE TYPE I18nEntity AS ENUM ('Composer', 'Piece', 'Venue', 'City', 'Performer', 'Event', 'EventPerformer', 'Country');
CREATE TYPE I18nKey AS ENUM ('Name', 'Address', 'Role');
CREATE TYPE I18nLang AS ENUM ('en', 'ru', 'ge');
CREATE TABLE I18n (
  EntityType I18nEntity NOT NULL,
  EntityId INT NOT NULL,
  Key I18nKey NOT NULL,
  Lang I18nLang NOT NULL,
  Value VARCHAR(1024) NOT NULL,
  PRIMARY KEY (EntityType, EntityId, Key, Lang)
);

CREATE TYPE PieceType AS ENUM ('Opera', 'Ballet', 'Symphony');
CREATE TABLE Piece (
  Id SERIAL NOT NULL PRIMARY KEY,
  Type PieceType,
  ComposerId INT NOT NULL,
  GeneralPieceId INT
);

CREATE TABLE Venue (
  Id SERIAL NOT NULL PRIMARY KEY,
  CityId INT NOT NULL
);

CREATE TABLE City (
  Id SERIAL NOT NULL PRIMARY KEY,
  CountryId INT NOT NULL
);

CREATE TABLE Country (
  Id SERIAL NOT NULL PRIMARY KEY,
  Code VARCHAR(3) NOT NULL
);

CREATE TABLE Performer (
  Id SERIAL NOT NULL PRIMARY KEY
);

CREATE TABLE Event (
  Id SERIAL NOT NULL PRIMARY KEY,
  Time TIMESTAMP WITH TIME ZONE,
  VenueId INT NOT NULL,
  Canceled BOOLEAN NOT NULL DEFAULT False
);

CREATE TABLE EventPiece (
  EventId INT NOT NULL,
  PieceId INT NOT NULL,
  PRIMARY KEY (EventId, PieceId)
);

CREATE TYPE EPInstrument AS ENUM ('Conductor', 'Piano', 'Flute', 'Violin', 'Cello', 'Voice');
CREATE TABLE EventPerformer (
  Id SERIAL NOT NULL PRIMARY KEY,
  EventId INT NOT NULL,
```

```
  PerformerId INT NOT NULL,
  Instrument EPInstrument,
  PRIMARY KEY (EventId, PerformerId)
);

CREATE TABLE PortalUser (
  Id SERIAL NOT NULL PRIMARY KEY,
  Login VARCHAR(50) NOT NULL,
  Name VARCHAR(50)
);

CREATE TYPE NotificationType AS ENUM ('NEW_EVENT', 'CANCELED_EVENT');
CREATE TABLE Notification (
  Id SERIAL NOT NULL PRIMARY KEY,
  Type NotificationType NOT NULL,
  UserId INT NOT NULL,
  CreatedTime TIMESTAMP NOT NULL DEFAULT current_timestamp,
  Read BOOLEAN NOT NULL DEFAULT False
);

CREATE TYPE NotificationEntityType AS ENUM ('Event', 'Composer', 'Piece', 'Venue', 'Performer', 'City', 'Country');
CREATE TABLE NotificationEntity (
  NotificationId INT NOT NULL,
  EntityType NotificationEntityType NOT NULL,
  EntityId INT NOT NULL,
  PRIMARY KEY (NotificationId, EntityType, EntityId)
);

CREATE TABLE Subscription (
  EntityType NotificationEntityType NOT NULL,
  EntityId INT NOT NULL,
  UserId INT NOT NULL,
  PRIMARY KEY (EntityType, EntityId, UserId)
);
```

Constraints, indexes

```
ALTER TABLE Piece ADD CONSTRAINT FK_Piece_ComposerId FOREIGN KEY (ComposerId) REFERENCES Composer (Id);
ALTER TABLE Piece ADD CONSTRAINT FK_Piece_GeneralPieceId FOREIGN KEY (GeneralPieceId) REFERENCES Piece (Id);
ALTER TABLE Venue ADD CONSTRAINT FK_Venue_CityId FOREIGN KEY (CityId) REFERENCES City (Id);
ALTER TABLE City ADD CONSTRAINT FK_City_CountryId FOREIGN KEY (CountryId) REFERENCES Country (Id);
ALTER TABLE Event ADD CONSTRAINT FK_Event_VenueId FOREIGN KEY (VenueId) REFERENCES Venue (Id);
ALTER TABLE EventPiece ADD CONSTRAINT FK_EventPiece_PieceId FOREIGN KEY (PieceId) REFERENCES Piece (Id);
ALTER TABLE EventPiece ADD CONSTRAINT FK_EventPiece_EventId FOREIGN KEY (EventId) REFERENCES Event (Id);
ALTER TABLE EventPerformer ADD CONSTRAINT FK_EventPerformer_EventId FOREIGN KEY (EventId) REFERENCES Event (Id);
ALTER TABLE EventPerformer ADD CONSTRAINT FK_EventPerformer_PerformerId FOREIGN KEY (PerformerId) REFERENCES Performer (Id);
ALTER TABLE Subscription ADD CONSTRAINT FK_Subscription_UserId FOREIGN KEY (UserId) REFERENCES PortalUser (Id);
ALTER TABLE Notification ADD CONSTRAINT FK_Notification_UserId FOREIGN KEY (UserId) REFERENCES PortalUser (Id);
ALTER TABLE NotificationEntity ADD CONSTRAINT FK_NotificationEntity_NotificationId FOREIGN KEY (NotificationId) REFERENCES Notification (

ALTER TABLE Country ADD CONSTRAINT UN_Country_Code UNIQUE (Code);
ALTER TABLE PortalUser ADD CONSTRAINT UN_PortalUser_Login UNIQUE (Login);

CREATE INDEX Ix_Event_Time ON Event (Time);
CREATE INDEX Ix_I18n_Lang ON I18n (Lang);

ALTER TABLE Composer ADD CONSTRAINT Composer_Dates CHECK (DateOfBirth < DateOfDeath);
ALTER TABLE I18n ADD CONSTRAINT I18n_Key_EntityType CHECK ((Key = 'Name' AND EntityType != 'EventPerformer') OR (Key = 'Address' AND Enti
```

Functions and triggers

Procedure `cancel_event` takes one param `EventId` and cancels event (with sending notifications).

Triggers `event_insert_trigger`, `event_performer_insert_trigger`, `event_piece_insert_trigger` handle notification sending for new events.

```
CREATE OR REPLACE FUNCTION cancel_event(int) RETURNS void AS $cancelEvent$
DECLARE
  u INT;
  _canceled BOOLEAN;
BEGIN
  SELECT Canceled FROM Event INTO _canceled WHERE Id = $1;
  IF _canceled THEN
    RAISE EXCEPTION 'already canceled';
  ELSE
    FOR u IN SELECT UserId FROM Subscription
             WHERE EntityType = 'Event' AND EntityId = $1
    LOOP
      INSERT INTO Notification (Type, UserId) VALUES ('CANCELED_EVENT', u);
      INSERT INTO NotificationEntity (NotificationId, EntityType, EntityId)
        VALUES (currval(pg_get_serial_sequence('notification','id')), 'Event', $1);
    END LOOP;
    UPDATE Event SET Canceled = True WHERE Id = $1;
  END IF;
END;
$cancelEvent$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION clear_matched() RETURNS void AS $clear_matched$
BEGIN
  CREATE TEMP TABLE IF NOT EXISTS Matched (
    EntityType NotificationEntityType,
    EntityId INT NOT NULL,
    UserId INT NOT NULL
  ) ON COMMIT DROP;
  TRUNCATE Matched;
END;

$clear_matched$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION event_process_matched(int) RETURNS void AS $event_process_matched$
DECLARE
  u INT;
  n INT;
```

```
BEGIN
  FOR u IN SELECT DISTINCT UserId FROM Matched
  LOOP
    SELECT nn.Id FROM Notification nn INTO n
      JOIN NotificationEntity nt ON nt.NotificationId = nn.Id
        AND nn.Type = 'NEW_EVENT' AND nt.EntityType = 'Event' AND nt.EntityId = $1
        AND nn.UserId = u;
    IF n IS NULL THEN
      INSERT INTO Notification (Type, UserId) VALUES ('NEW_EVENT', u);
      SELECT currval(pg_get_serial_sequence('notification','id')) INTO n;
      INSERT INTO NotificationEntity (NotificationId, EntityType, EntityId)
        SELECT n, 'Event', $1;
    END IF;
    INSERT INTO NotificationEntity (NotificationId, EntityType, EntityId)
      SELECT n, EntityType, EntityId FROM Matched;
  END LOOP;
END;
$event_process_matched$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION event_insert_trigger() RETURNS trigger AS $event_insert_trigger$
DECLARE
  u INT;
  n INT;
BEGIN
  PERFORM clear_matched();
  INSERT INTO Matched (EntityType, EntityId, UserId)
  SELECT s.EntityType, s.EntityId, s.UserId FROM Subscription s JOIN (
    SELECT 'Venue'::NotificationEntityType as t, NEW.VenueId as Id
    UNION
    SELECT 'City'::NotificationEntityType as t, v.CityId as Id FROM Venue v WHERE v.Id = NEW.VenueId
    UNION
    SELECT 'Country'::NotificationEntityType as t, c.CountryId as Id
        FROM Venue v
        JOIN City c ON c.Id = v.CityId
        WHERE v.Id = NEW.VenueId
  ) cp ON cp.t = s.EntityType AND s.EntityId = cp.id;
  PERFORM event_process_matched(NEW.Id);
  RETURN NEW;
END;
$event_insert_trigger$ LANGUAGE plpgsql;

CREATE TRIGGER event_insert_trigger AFTER INSERT ON Event
FOR EACH ROW EXECUTE PROCEDURE event_insert_trigger();

CREATE OR REPLACE FUNCTION event_performer_insert_trigger() RETURNS trigger AS $event_performer_insert_trigger$
DECLARE
  u INT;
  n INT;
BEGIN
  PERFORM clear_matched();
  INSERT INTO Matched (EntityType, EntityId, UserId)
  SELECT s.EntityType, s.EntityId, s.UserId FROM Subscription s JOIN (
    SELECT 'Performer'::NotificationEntityType as t, NEW.PerformerId as Id
  ) cp ON cp.t = s.EntityType AND s.EntityId = cp.id;
  PERFORM event_process_matched(NEW.EventId);
  RETURN NEW;
END;
$event_performer_insert_trigger$ LANGUAGE plpgsql;

CREATE TRIGGER event_performer_insert_trigger AFTER INSERT ON EventPerformer
FOR EACH ROW EXECUTE PROCEDURE event_performer_insert_trigger();


CREATE OR REPLACE FUNCTION event_piece_insert_trigger() RETURNS trigger AS $event_piece_insert_trigger$
DECLARE
  u INT;
  n INT;
BEGIN
  PERFORM clear_matched();
  INSERT INTO Matched (EntityType, EntityId, UserId)
  SELECT s.EntityType, s.EntityId, s.UserId FROM Subscription s JOIN (
    SELECT 'Composer'::NotificationEntityType as t, c.Id FROM Composer c JOIN Piece p ON c.Id = p.ComposerId AND p.Id = NEW.PieceId
    UNION
    SELECT 'Piece', p.GeneralPieceId FROM Piece p WHERE p.Id = NEW.PieceId AND p.GeneralPieceId IS NOT NULL
    UNION
    SELECT 'Piece', NEW.PieceId
  ) cp ON cp.t = s.EntityType AND s.EntityId = cp.id;
  PERFORM event_process_matched(NEW.EventId);
  RETURN NEW;
END;
$event_piece_insert_trigger$ LANGUAGE plpgsql;

CREATE TRIGGER event_piece_insert_trigger AFTER INSERT ON EventPiece
FOR EACH ROW EXECUTE PROCEDURE event_piece_insert_trigger();
```

Views

```
CREATE VIEW EventInfo AS
(
  SELECT e.*, l.Lang,
         iE.Value as EventName,
         iV.Value as VenueName,
         iVA.Value as VenueAddress,
         v.CityId,
         iCi.Value as CityName,
         ci.CountryId,
         iCu.Value as CountryName
  FROM Event e
  CROSS JOIN (SELECT DISTINCT Lang FROM I18n) l
  JOIN Venue v ON v.Id = e.VenueId
  JOIN City ci ON ci.Id = v.CityId
  JOIN Country cu ON cu.Id = ci.CountryId
  LEFT JOIN I18n iE ON iE.EntityType = 'Event' AND iE.EntityId = e.VenueId AND iE.Lang = l.Lang AND iE.Key = 'Name'
```

```
  LEFT JOIN I18n iV ON iV.EntityType = 'Venue' AND iV.EntityId = e.VenueId AND iV.Lang = l.Lang AND iV.Key = 'Name'
  LEFT JOIN I18n iVA ON iVA.EntityType = 'Venue' AND iVA.EntityId = e.VenueId AND iVA.Lang = l.Lang AND iVA.Key = 'Address'
  LEFT JOIN I18n iCi ON iCi.EntityType = 'City' AND iCi.EntityId = ci.Id AND iCi.Lang = l.Lang AND iCi.Key = 'Name'
  LEFT JOIN I18n iCu ON iCu.EntityType = 'Country' AND iCu.EntityId = ci.Id AND iCu.Lang = l.Lang AND iCu.Key = 'Name'
);
CREATE VIEW EventPieceInfo AS
(
  SELECT ep.*, l.Lang,
         iP.Value as PieceName,
         iC.Value as ComposerName,
         p.Type as PieceType,
         p.GeneralPieceId,
         pg.Type as GeneralPieceType,
         iG.Value as GeneralPieceName,
         co.DateOfBirth as composerDateOfBirth,
         co.DateOfDeath as composerDateOfDeath
  FROM EventPiece ep
  CROSS JOIN (SELECT DISTINCT Lang FROM I18n) l
  JOIN Piece p ON p.Id = ep.PieceId
  JOIN Composer co ON co.Id = p.ComposerId
  LEFT JOIN Piece pg ON pg.Id = p.GeneralPieceId
  LEFT JOIN I18n iP ON iP.EntityType = 'Piece' AND iP.EntityId = ep.PieceId AND iP.Lang = l.Lang AND iP.Key = 'Name'
  LEFT JOIN I18n iG ON iG.EntityType = 'Piece' AND iG.EntityId = p.GeneralPieceId AND iG.Lang = l.Lang AND iG.Key = 'Name'
  LEFT JOIN I18n iC ON iC.EntityType = 'Composer' AND iC.EntityId = p.ComposerId AND iC.Lang = l.Lang AND iC.Key = 'Name'
);
CREATE VIEW EventPerformerInfo AS
(
  SELECT ep.*, l.Lang,
         iP.Value as PerformerName,
         iR.Value as Role
  FROM EventPerformer ep
  CROSS JOIN (SELECT DISTINCT Lang FROM I18n) l
  LEFT JOIN I18n iP ON iP.EntityType = 'Performer' AND iP.EntityId = ep.PerformerId AND iP.Lang = l.Lang AND iP.Key = 'Name'
  LEFT JOIN I18n iR ON iR.EntityType = 'EventPerformer' AND iR.EntityId = ep.Id AND iR.Lang = l.Lang AND iR.Key = 'Role'
);
```

Example queries

```
-- Users that have subscription for piece and no for composer

SELECT s.UserId, p.ComposerId FROM Subscription s
JOIN Piece p ON p.Id = s.EntityId AND s.EntityType = 'Piece'
LEFT JOIN Subscription s2 ON s2.EntityId = p.ComposerId AND s.EntityType = 'Composer' AND s2.UserId = s.UserId
WHERE s2.EntityId IS NULL;

-- Missing non-optional translations
(SELECT a.*, l.Lang FROM
  (
    SELECT 'Composer'::I18nEntity as EntityType, 'Name'::I18nKey as Key, Id as EntityId FROM Composer
    UNION
    SELECT 'Piece', 'Name', Id FROM Piece
    UNION
    SELECT 'City', 'Name', Id FROM City
    UNION
    SELECT 'Country', 'Name', Id FROM Country
    UNION
    SELECT 'Venue', 'Name', Id FROM Venue
    UNION
    SELECT 'Venue', 'Address', Id FROM Venue
    UNION
    SELECT 'Performer', 'Name', Id FROM Performer
  ) a
  CROSS JOIN (SELECT DISTINCT Lang FROM I18n) l
)
EXCEPT
SELECT EntityType, Key, EntityId, Lang FROM I18n;

-- Conductors, performing in St.Petersburg in both February 2016 and February 2015
(
  SELECT DISTINCT ep.PerformerId FROM EventPerformer ep
  JOIN Event e ON ep.EventId = e.Id
  JOIN Venue v ON v.Id = e.VenueId
  WHERE v.CityId = 1 AND e.Time >= '2016-02-01' AND e.Time < '2016-03-01'
) INTERSECT (
  SELECT DISTINCT ep.PerformerId FROM EventPerformer ep
  JOIN Event e ON ep.EventId = e.Id
  JOIN Venue v ON v.Id = e.VenueId
  WHERE v.CityId = 1 AND e.Time >= '2015-02-01' AND e.Time < '2015-03-01'
);

-- Conductors, that had performed both in St. Petersburg Philharmonia and Covent-Garden in 2015
(
  SELECT DISTINCT ep.PerformerId FROM EventPerformer ep
  JOIN Event e ON ep.EventId = e.Id
  JOIN Venue v ON v.Id = e.VenueId
  WHERE v.Id = 2 AND e.Time >= '2015-01-01' AND e.Time < '2016-01-01'
) INTERSECT (
  SELECT DISTINCT ep.PerformerId FROM EventPerformer ep
  JOIN Event e ON ep.EventId = e.Id
  JOIN Venue v ON v.Id = e.VenueId
  WHERE v.Id = 4 AND e.Time >= '2015-01-01' AND e.Time < '2016-01-01'
);

-- Repertoire of all venues in Russian
SELECT p.*, iV.Value as VenueName, iP.Value as PieceName, l.Lang FROM (
  SELECT p.Type, e.VenueId, p.Id as PieceId, MIN(e.Time) as FirstPerformed, MAX(e.Time) as LastPerformed
  FROM EventPiece ep
  JOIN Event e ON ep.EventId = e.Id
  JOIN Piece p ON ep.PieceId = p.Id
  GROUP BY e.VenueId, p.Id
) p
CROSS JOIN (SELECT DISTINCT Lang FROM I18n) l
LEFT JOIN I18n iP ON iP.EntityType = 'Piece' AND iP.EntityId = p.PieceId AND iP.Lang = l.Lang AND iP.Key = 'Name'
```

```
LEFT JOIN I18n iV ON iV.EntityType = 'Venue' AND iV.EntityId = p.VenueId AND iV.Lang = l.Lang AND iV.Key = 'Name'
WHERE l.Lang = 'ru'
;
```

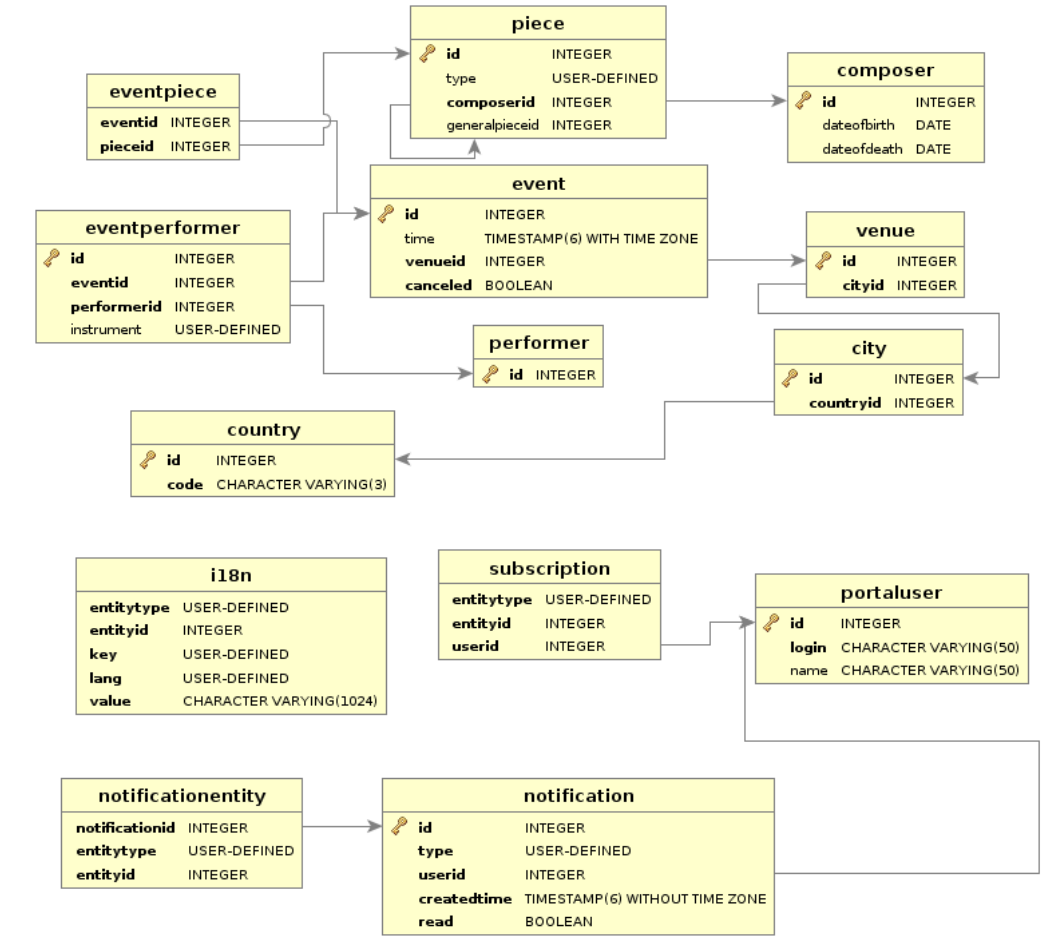
Example data

```
INSERT INTO Country (Id, Code) VALUES
  (1, 'ru'), (2, 'uk'), (3, 'de');
INSERT INTO I18n (EntityType, EntityId, Key, Lang, Value) VALUES
  ('Country', 1, 'Name', 'ru', 'Россия'),
  ('Country', 1, 'Name', 'en', 'Russia'),
  ('Country', 1, 'Name', 'ge', 'Russland'),
  ('Country', 2, 'Name', 'ru', 'Великобритания'),
  ('Country', 2, 'Name', 'en', 'Great Britain'),
  ('Country', 2, 'Name', 'ge', 'Großbritannien'),
  ('Country', 3, 'Name', 'ru', 'Германия'),
  ('Country', 3, 'Name', 'en', 'Germany'),
  ('Country', 3, 'Name', 'ge', 'Deutschland');
INSERT INTO City (Id, CountryId) VALUES
  (1, 1), (2, 1), (3, 2), (4, 3);

INSERT INTO I18n (EntityType, EntityId, Key, Lang, Value) VALUES
  ('City', 1, 'Name', 'ru', 'Санкт-Петербург'),
  ('City', 1, 'Name', 'en', 'St. Petersburg'),
  ('City', 1, 'Name', 'ge', 'St. Petersburg'),
  ('City', 2, 'Name', 'ru', 'Москва'),
  ('City', 2, 'Name', 'en', 'Moscow'),
  ('City', 2, 'Name', 'ge', 'Moskau'),
  ('City', 3, 'Name', 'ru', 'Лондон'),
  ('City', 3, 'Name', 'en', 'London'),
  ('City', 3, 'Name', 'ge', 'London'),
  ('City', 4, 'Name', 'ru', 'Берлин'),
  ('City', 4, 'Name', 'en', 'Berlin'),
  ('City', 4, 'Name', 'ge', 'Berlin');

INSERT INTO Venue (Id, CityId) VALUES
        (1, 1), (2, 1), (3, 2), (4, 3), (5, 4);

INSERT INTO I18n (EntityType, EntityId, Key, Lang, Value) VALUES
  ('Venue', 1, 'Name', 'ru', 'Мариинский театр'),
  ('Venue', 1, 'Name', 'en', 'Mariinsky theatre'),
  ('Venue', 1, 'Name', 'ge', 'Mariinski-Theater'),
  ('Venue', 2, 'Name', 'ru', 'Санкт-Петербургская Филармония'),
  ('Venue', 2, 'Name', 'en', 'Saint Petersburg Philharmonia'),
  ('Venue', 2, 'Name', 'ge', 'Sankt Petersburger Philharmonie'),
  ('Venue', 3, 'Name', 'ru', 'Большой театр'),
  ('Venue', 3, 'Name', 'en', 'Bolshoi theatre'),
  ('Venue', 3, 'Name', 'ge', 'Bolschoi-Theater'),
  ('Venue', 4, 'Name', 'ru', 'Ковент-Гарден'),
  ('Venue', 4, 'Name', 'en', 'Royal Opera House'),
  ('Venue', 4, 'Name', 'ge', 'Royal Opera House'),
  ('Venue', 5, 'Name', 'ru', 'Немецкая опера в Берлине'),
  ('Venue', 5, 'Name', 'en', 'German Opera House in Berlin'),
  ('Venue', 5, 'Name', 'ge', 'Deutsche Oper Berlin'),
  ('Venue', 1, 'Address', 'ru', 'Театральная площадь, 1'),
  ('Venue', 1, 'Address', 'en', '1 Theatre Square'),
  ('Venue', 1, 'Address', 'ge', 'Theaterplatz, 1'),
  ('Venue', 2, 'Address', 'ru', 'Михайловская ул., 2'),
  ('Venue', 2, 'Address', 'en', 'Mikhaylovskaya ul., 2'),
  ('Venue', 2, 'Address', 'ge', 'Michajlowskaja-Straße 2'),
  ('Venue', 3, 'Address', 'ru', 'Театральная площадь, 1'),
  ('Venue', 3, 'Address', 'en', '1 Theatre Square'),
  ('Venue', 3, 'Address', 'ge', 'Theaterplatz, 1'),
  ('Venue', 4, 'Address', 'ru', 'Bow St, London WC2E 9DD'),
  ('Venue', 4, 'Address', 'en', 'Bow St, London WC2E 9DD'),
  ('Venue', 4, 'Address', 'ge', 'Bow St, London WC2E 9DD'),
  ('Venue', 5, 'Address', 'ru', 'Биссмаркштрассе, 35'),
  ('Venue', 5, 'Address', 'en', 'Bismark street 35, 10627 Berlin, Germany'),
  ('Venue', 5, 'Address', 'ge', 'Bismarckstraße 35, 10627 Berlin, Germany');

INSERT INTO Composer (Id, DateOfBirth, DateOfDeath) VALUES
        (1, '1839-03-21', '1881-03-28'),
        (2, '1813-05-22', '1883-02-13'),
        (3, '1840-05-07', '1893-11-06'),
        (4, '1891-04-23', '1953-03-05');

INSERT INTO I18n (EntityType, EntityId, Key, Lang, Value) VALUES
  ('Composer', 1, 'Name', 'ru', 'Мусоргский, Модест Петрович'),
  ('Composer', 1, 'Name', 'en', 'Modest Mussorgsky'),
  ('Composer', 1, 'Name', 'ge', 'Modest Petrowitsch Mussorgski'),
  ('Composer', 2, 'Name', 'ru', 'Рихард Вагнер'),
  ('Composer', 2, 'Name', 'en', 'Richard Wagner'),
  ('Composer', 2, 'Name', 'ge', 'Richard Wagner'),
  ('Composer', 3, 'Name', 'ru', 'Чайковский, Пётр Ильич'),
  ('Composer', 3, 'Name', 'en', 'Pyotr Ilyich Tchaikovsky'),
  ('Composer', 3, 'Name', 'ge', 'Pjotr Iljitsch Tschaikowski'),
  ('Composer', 4, 'Name', 'ru', 'Прокофьев, Сергей Сергеевич'),
  ('Composer', 4, 'Name', 'en', 'Sergei Prokofiev'),
  ('Composer', 4, 'Name', 'ge', 'Sergei Prokofjew');

INSERT INTO Piece (Id, Type, ComposerId, GeneralPieceId) VALUES
        (1, 'Ballet', 3, NULL), (2, 'Opera', 2, NULL), (3, 'Opera', 1, NULL), (4, NULL, 3, NULL),
  (5, 'Symphony', 4, NULL), (6, 'Ballet', 4, NULL), (7, NULL, 4, 6);

INSERT INTO I18n (EntityType, EntityId, Key, Lang, Value) VALUES
  ('Piece', 1, 'Name', 'ru', 'Спящая красавица'),
  ('Piece', 1, 'Name', 'en', 'The Sleeping Beauty'),
  ('Piece', 1, 'Name', 'ge', 'Dornröschen'),
  ('Piece', 2, 'Name', 'ru', 'Валькирия'),
  ('Piece', 2, 'Name', 'en', 'The Valkyrie'),
  ('Piece', 2, 'Name', 'ge', 'Die Walküre'),
  ('Piece', 3, 'Name', 'ru', 'Хованщина'),
  ('Piece', 3, 'Name', 'en', 'Khovanshchina'),
```

```
  ('Piece', 3, 'Name', 'ge', 'Chowanschtschina'),
  ('Piece', 4, 'Name', 'ru', 'Ромео и Джульетта, увертюра-фантазия'),
  ('Piece', 4, 'Name', 'en', 'Romeo and Juliet, an overture-fantasy'),
  ('Piece', 4, 'Name', 'ge', 'Romeo und Julia, eine Fantasie-Ouvertüre'),
  ('Piece', 5, 'Name', 'ru', 'Симфония № 2'),
  ('Piece', 5, 'Name', 'en', 'Symphony No. 2 in D minor'),
  ('Piece', 5, 'Name', 'ge', 'Sinfonie No. 2'),
  ('Piece', 6, 'Name', 'ru', 'Ромео и Джульетта'),
  ('Piece', 6, 'Name', 'en', 'Romeo and Juliet'),
  ('Piece', 6, 'Name', 'ge', 'Romeo und Julia'),
  ('Piece', 7, 'Name', 'ru', 'Отрывки из балета "Ромео и Джульетта"'),
  ('Piece', 7, 'Name', 'en', 'Excerpts from ballet "Romeo and Juliet"'),
  ('Piece', 7, 'Name', 'ge', 'Auszüge aus Ballett "Romeo und Julia"');

INSERT INTO Performer (Id) VALUES
       (1), (2), (3), (4), (5);

INSERT INTO I18n (EntityType, EntityId, Key, Lang, Value) VALUES
  ('Performer', 1, 'Name', 'ru', 'Гергиев, Валерий Абисалович'),
  ('Performer', 1, 'Name', 'en', 'Valery Gergiev'),
  ('Performer', 1, 'Name', 'ge', 'Waleri Gergijev'),
  ('Performer', 2, 'Name', 'ru', 'Оркестр Мариинского театра'),
  ('Performer', 2, 'Name', 'en', 'Orchestra of Mariinsky theatre'),
  ('Performer', 2, 'Name', 'ge', 'Das Orchester des Mariinski-Theater'),
  ('Performer', 3, 'Name', 'ru', 'Ольга Бородина'),
  ('Performer', 3, 'Name', 'en', 'Olga Borodina'),
  ('Performer', 3, 'Name', 'ge', 'Olga Borodina'),
  ('Performer', 4, 'Name', 'ru', 'Синайский, Василий Серафимович'),
  ('Performer', 4, 'Name', 'en', 'Vassily Sinaysky'),
  ('Performer', 4, 'Name', 'ge', 'Wassili Sinaiski'),
  ('Performer', 5, 'Name', 'ru', 'Заслуженный оркестр Санкт-Петербургской Филармонии'),
  ('Performer', 5, 'Name', 'en', 'Honored Orchestra of Saint-Petersburg Philharmonia'),
  ('Performer', 5, 'Name', 'ge', 'Geehrt Orchester des Sankt Petersburger Philharmonie');

INSERT INTO Event (Id, Time, VenueId) VALUES
  (1, '2016-01-27 19:00 MSK', 1);

INSERT INTO EventPiece (EventId, PieceId) VALUES
  (1, 3); -- Khovanshschina

INSERT INTO EventPerformer (Id, EventId, PerformerId, Instrument) VALUES
  (1, 1, 1, 'Conductor'), (2, 1, 3, 'Voice');

INSERT INTO I18n (EntityType, EntityId, Key, Lang, Value) VALUES
  ('EventPerformer', 2, 'Role', 'ru', 'Марфа'),
  ('EventPerformer', 2, 'Role', 'en', 'Marfa'),
  ('EventPerformer', 2, 'Role', 'ge', 'Marfa');

INSERT INTO PortalUser (Id, Login, Name) VALUES
  (1, 'georgeee', 'George Agapov'),
  (2, 'jagger', 'Mick Jagger'),
  (3, 'johny', 'John Lennon');

INSERT INTO Subscription (EntityType, EntityId, UserId) VALUES
  ('Composer', 2, 1), ('Event', 2, 1), ('Performer', 4, 1), ('Piece', 6, 1);

INSERT INTO Event (Id, Time, VenueId) VALUES
  (2, '2015-02-03 20:00 MSK', 2),
  (3, '2016-02-27 19:00 MSK', 1),
  (4, '2015-02-17 19:00 MSK', 1),
  (5, '2015-07-15 19:00 UTC', 4);
INSERT INTO EventPiece (EventId, PieceId) VALUES
  (2, 4), -- Romeo & Juliet, fantasy-overture
  (2, 5), -- Symphony no. 2 by Prokofiev
  (5, 5), -- Symphony no. 2 by Prokofiev
  (2, 7), -- Excerpts from ballet Romeo and Juliet
  (3, 2), -- Walküre
  (4, 2); -- Walküre

INSERT INTO EventPerformer (Id, EventId, PerformerId, Instrument) VALUES
 (3, 2, 4, 'Conductor'),
 (4, 3, 1, 'Conductor'), (5, 3, 3, 'Voice'),
 (6, 4, 1, 'Conductor'),
 (7, 5, 4, 'Conductor');
INSERT INTO I18n (EntityType, EntityId, Key, Lang, Value) VALUES
  ('EventPerformer', 5, 'Role', 'ru', 'Брунгильда'),
  ('EventPerformer', 5, 'Role', 'en', 'Brünnhilde'),
  ('EventPerformer', 5, 'Role', 'ge', 'Brünnhilde');

SELECT cancel_event (2) ;


INSERT INTO Performer (Id) VALUES
  (6), (7);

INSERT INTO I18n (EntityType, EntityId, Key, Lang, Value) VALUES
  ('Performer', 6, 'Name', 'en', 'Yury Temirkanov');
```

Appendix

Generated from set-up db by DBVis tool:


piece
id INTEGER
type USER-DEFINED
composerid INTEGER
generalpieceid INTEGER
composer
id INTEGER
dateofbirth DATE
dateofdeath DATE
eventpiece
eventid INTEGER
pieceid INTEGER
event
id INTEGER
time TIMESTAMP(6) WITH TIME ZONE
venueid INTEGER
canceled BOOLEAN
eventperformer
id INTEGER
eventid INTEGER
performerid INTEGER
instrument USER-DEFINED
venue
id INTEGER
cityid INTEGER
performer
id INTEGER
city
id INTEGER
countryid INTEGER
country
id INTEGER
code CHARACTER VARYING(3)
i18n
entitytype USER-DEFINED
entityid INTEGER
key USER-DEFINED
lang USER-DEFINED
value CHARACTER VARYING(1024)
subscription
entitytype USER-DEFINED
entityid INTEGER
userid INTEGER
portaluser
id INTEGER
login CHARACTER VARYING(50)
name CHARACTER VARYING(50)
notificationentity
notificationid INTEGER
entitytype USER-DEFINED
entityid INTEGER
notification
id INTEGER
type USER-DEFINED
userid INTEGER
createdtime TIMESTAMP(6) WITHOUT TIME ZONE
read BOOLEAN