Wondertix

A Ticketing and CRM Database

Contents

[**Users**](#_e8un2p5l9wum) **3**

[users](#_pyr5jhv0b1dw) 4

[task](#_1a68qj6q9ucw) 5

[tasknotes](#_6p1wipeq1620) 6

[**Ticketing**](#_llhs75enq1tu) **7**

[seasons](#_dz6opw7037kp) 8

[events](#_wk6lera6jcj9) 9

[eventinstances](#_hlg9nvvg5ta2) 10

[eventtickets](#_qu8c463li89c) 11

[tickettype](#_71xvsugvc630) 12

[ticketrestrictions](#_ad4u3g8k707u) 13

[**CRM**](#_1gndes9eh5xn) **14**

[contacts](#_x5r7m6eeprkn) 15

[orders](#_rjl5nwppce87) 16

[discounts](#_5klmr62dto4y) 17

[orderitems](#_ii1rykk0waje) 18

[singletickets](#_v1xvgvkkz2zz) 19

[seasontickets](#_vcpfijnk7ljv) 20

[seasontickettype](#_rbw1zjz4nu21) 21

[donations](#_2x0g5f9mabnn) 22

[**Misc**](#_fatgouqjycd7) **23**

[date](#_8d81pxknel9h) 23

[savedreports](#_g41tx83cwqtj) 25

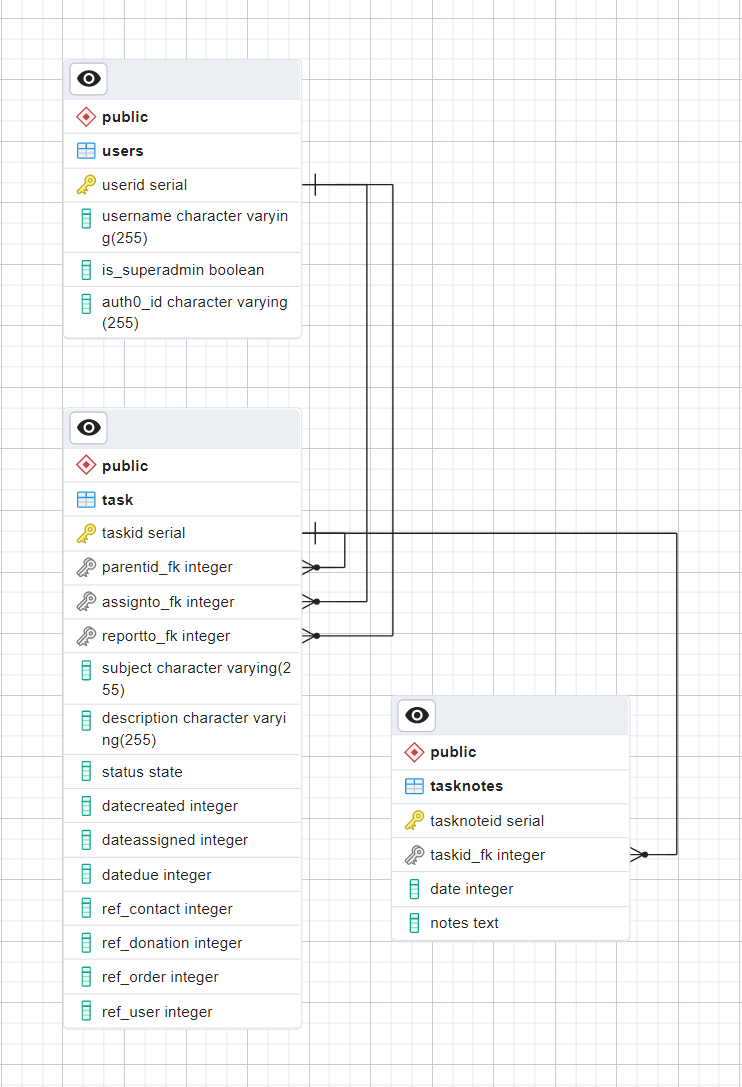
The Wondertix database models the business operations of Portland Playhouse. The database can be viewed as having three main components, namely:

* Users (or employees)
* Ticketing, which models events, their showings, and admission tickets
* Customer Relations Management, which contains customer information and purchase history

There are other tables that facilitate database functionality, which are grouped under Misc.

Each section starts with a schema diagram showcasing tables and their primary relations. Other relations will be detailed in individual table breakdowns.

# Users



## users

| users | | |
| --- | --- | --- |
| userid | serial | Primary key. Auto-incrementing integer |
| username | varchar (255) |  |
| is\_superadmin | boolean | Marks user as having superuser privileges |
| auth0\_id | varchar (255) | ID string for Auth0 authentication |

The users table models individual employees of Portland Playhouse. Users are assigned a userid and may create events, showings, ticket types, and have access to all other day to day operations. Superadmins have access to reporting functions.

## task

| task | | |
| --- | --- | --- |
| taskid | serial | Primary key. Auto-incrementing integer |
| parentid\_fk | integer | Foreign key. Refers to some other *taskid* |
| assignto\_fk | integer | Foreign key. Refers to a *userid* from the ***users*** table |
| reportto\_fk | integer | Foreign key. Refers to a *userid* from the ***users*** table |
| subject | varchar (255) | Task subject |
| description | varchar (255) | Task description |
| status | state | Custom enum. Can be any of:  “not\_started”, “in\_progress”, “completed” |
| datecreated | integer | Foreign key. Refers to a *dateid* from the ***date*** table |
| dateassigned | integer | Foreign key. Refers to a *dateid* from the ***date*** table |
| datedue | integer | Foreign key. Refers to a *dateid* from the ***date*** table |
| ref\_contact | integer | Foreign key.  Refers to a *contactid* from the ***contacts*** table |
| ref\_donation | integer | Foreign key.  Refers to a *donationid* from the ***donations*** table |
| ref\_order | integer | Foreign key. Refers to a *orderid* from the ***orders*** table |
| ref\_user | integer | Foreign key. Refers to a *userid* from the ***users*** table |

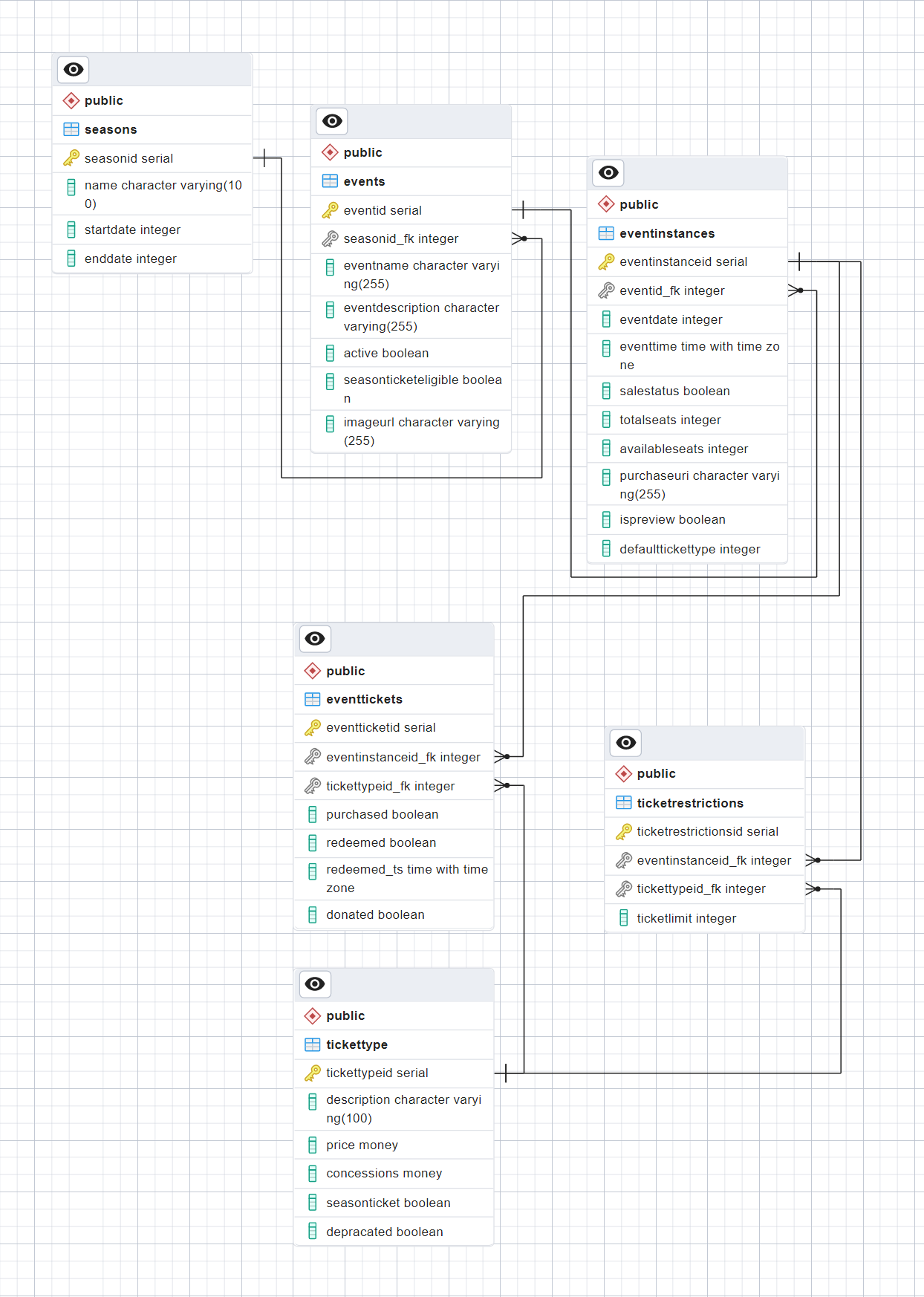
A task allows a user to create a to-do list or a memo, identify an issue with an order or account that requires resolution, etc. Tasks may be linked to other tasks for follow up, they can be assigned to users, and may contain references to a task note, an order, donation, or contact.

## tasknotes

| tasknotes | | |
| --- | --- | --- |
| tasknoteid | serial | Primary key. Auto-incrementing integer |
| taskid\_fk | integer | Foreign key. Refers to a *taskid* from the ***task*** table |
| date | integer | Foreign key. Refers to a *dateid* from the ***date*** table |
| notes | text | Text/note contents |

Task notes are for storing longer/larger bodies of text that do not fit into the standard fields of a task.

# Ticketing



## seasons

| seasons | | |
| --- | --- | --- |
| seasonid | serial | Primary key. Auto-incrementing integer |
| name | varchar (100) | Name of the season |
| startdate | integer | Foreign key. Refers to a *dateid* from the ***date*** table |
| enddate | integer | Foreign key. Refers to a *dateid* from the ***date*** table |

A season is a run of events over some fixed period of time, i.e. all events fall under a season. Seasons are given a start and end date, and automatically assigned a season number.

## events

| events | | |
| --- | --- | --- |
| eventid | serial | Primary key. Auto-incrementing integer |
| seasonid\_fk | integer | Foreign key.  Refers to a *seasonid* from the ***seasons*** table |
| eventname | varchar (255) | Event name |
| eventdescription | varchar (255) | Event description |
| active | boolean | Indicates if event is active or not |
| seasonticketeligible | boolean | Indicates if the event eligible for season tickets |
| imageurl | varchar (255) | Image URL |

An event encompasses the production of some play, or the screenings of some movie. An event will subsequently have a number of event instances, which are the actual performances of the event on a given date.

Events have a name, description and are linked to/fall under some given season. After an event closes and is no longer being performed/shown, events should be marked as inactive. Events can be marked as eligible/not-eligible to be seen with a season ticket purchase, and contain a link to an image used for display on the website.

## eventinstances

| eventinstances | | |
| --- | --- | --- |
| eventinstanceid | serial | Primary key. Auto-incrementing integer |
| eventid\_fk | integer | Foreign key. Refers to a *eventid* from the ***date*** table |
| eventdate | integer | Foreign key. Refers to a *dateid* from the ***date*** table |
| eventtime | time with timezone | Event description |
| salestatus | boolean | Indicates is tickets can/are being sold |
| totalseats | integer | Total number of seats for this instance |
| availableseats | integer | Number of seats left for purchase |
| purchaseuri | varchar (255) |  |
| ispreview | boolean | Instance is a preview showing |
| defaulttickettype | integer | Foreign key.  Refers to a *tickettypeid* from the ***tickettype*** table |

An event instance is the specific performance/showing of an event (e.g. a Saturday 5pm showing of “Guys and Dolls”). Event instances link to their parent event and have a date and time. When the time and date for an instance has passed, their sales status should be marked as false.

Instances track both the total number of seats available, and the number of seats left to purchase. Instances can be marked as a preview (a performance before opening night), and are designated a default ticket type/price (e.g. a $10 weekday matinee vs a $25 weekend evening)

## eventtickets

| eventtickets | | |
| --- | --- | --- |
| eventticketid | serial | Primary key. Auto-incrementing integer |
| eventinstanceid\_fk | integer | Foreign key. Refers to a *eventinstanceid* from the ***eventinstance*** table |
| tickettypeid\_fk | integer | Foreign key.  Refers to a *tickettypeid* from the ***tickettype*** table |
| purchased | boolean | Indicates ticket has been purchased |
| redeemed | boolean | Indicates the purchasing customer has checked in for the instance |
| redeemed\_ts | time with timezone | Timestamp for ticket redemption |
| donated | boolean | Indicates the purchaser has donated the ticket instead of requesting a refund. Tickets may be resold. |

Event tickets represent the seats available for purchase to any given event instance (an event instance with 100 available seats will have 100 event tickets). Tickets have a reference to their event instance and are assigned a ticket type upon purchase.

Purchased tickets are marked as such, and can be filtered on this field to create a doorlist for an instance. Tickets are marked as redeemed and are time stamped when the customer arrives/checks in for the showing. Tickets may also be donated back to Portland Playhouse instead of being refunded.

## tickettype

| tickettype | | |
| --- | --- | --- |
| tickettypeid | serial | Primary key. Auto-incrementing integer |
| description | varchar (100) | Description of the ticket type |
| price | money | Ticket price |
| concessions | money | Included concessions credit |
| seasonticket | boolean | (left over from previous design) |
| deprecated | boolean | Marks type as deprecated/no longer in use |

Ticket types identify the various kinds of tickets sold by Portland Playhouse (matinee vs primetime tickets, weekday vs weekend, child vs adult, etc).

They contain a description of the type, a base price, and may include some amount of concessions credit. Types may be marked as deprecated to prevent their use in the future.

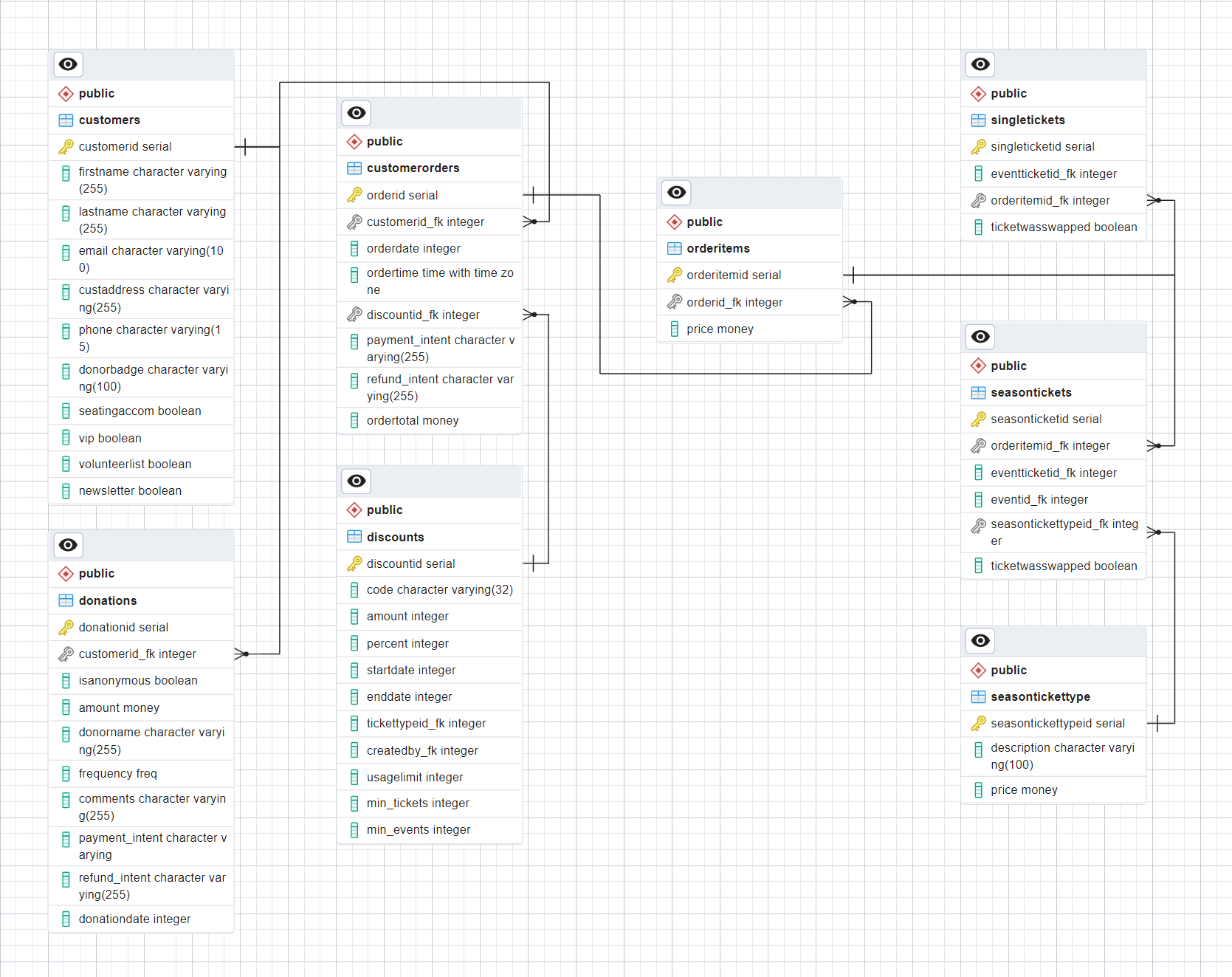
## ticketrestrictions

| ticketrestrictions | | |
| --- | --- | --- |
| ticketrestrictionid | serial | Primary key. Auto-incrementing integer |
| eventinstanceid\_fk | integer | Foreign key. Refers to a *eventinstanceid* from the ***eventinstance*** table |
| tickettypeid\_fk | integer | Foreign key.  Refers to a *tickettypeid* from the ***tickettype*** table |
| ticketlimit | integer | Number of tickets that may be sold |
| ticketssold | integer | Number of tickets that have been sold |

For some event instances, Portland Playhouse may have more than one kind of ticket available for purchase other than the default ticket type. An instance may have 100 seats in total, but there might be up to 10 handicap accessible seats available, or maybe they want to sell no more than 20 children's tickets. The ***ticketrestrictions*** table enables this by linking a given event with a ticket type and storing a ticket limit. As tickets of this type are sold, the *ticketssold* field is incremented up to the *ticketlimit*.

Ex: Assume an event instance has 100 seats available, the default ticket type is a general admission ticket, and there are no more than 10 handicap tickets available. It is possible to sell all 100 tickets as general admission, or up to 90 as general admission plus all 10 handicap tickets.

# CRM



## contacts

| contacts | | |
| --- | --- | --- |
| contactid | serial | Primary key. Auto-incrementing integer |
| firstname | varchar (255) | First name |
| lastname | varchar (255) | Last name |
| email | varchar (100) | Email address |
| address | varchar (255) | Mailing address |
| phone | varchar (15) | Phone number |
| donorbadge | boolean | Special donor badge |
| seatingaccom | boolean | Requests seating accommodations |
| vip | boolean | Is VIP |
| volunteerlist | boolean | Is on volunteer list |
| newsletter | boolean | Is signed up for the newsletter |

The ***contacts*** table stores basic information about Portland Playhouse customers, like their name, phone number, email address, etc. There are fields that identify a customer as requiring seating accommodations, being a VIP or volunteer, and being subscribed to the newsletter.

## orders

| orders | | |
| --- | --- | --- |
| orderid | serial | Primary key. Auto-incrementing integer |
| contactid\_fk | integer | Foreign key.  Refers to a *contactid* from the contacts table |
| orderdate | integer | Foreign key. Refers to a *dateid* from the date table |
| ordertime | time with timezone | Time order was placed |
| discountid\_fk | integer | Foreign key.  Refers to a *discountid* from the discounts table |
| payment\_intent | varchar (255) | Stripe payment auth/key |
| refund\_intent | varchar (255) | Stripe payment auth/key |
| ordertotal | money | Sum total of all order items in order |

The orders table is a representation of ticket purchases made by customers. Each order will have one or more order items linked to it.

Orders contain a link to the customers entry in the ***contacts*** table, an order date and time, and the final purchase total. Orders may contain a link to a discount applied to the order. The *payment\_intent* and *refund\_intent* fields are for storing unique ids related to the Stripe payment system. The presence of a value in the *refund\_intent* field indicates an order has been refunded.

## discounts

| discounts | | |
| --- | --- | --- |
| discountid | serial | Primary key. Auto-incrementing integer |
| code | varchar (32) | Discount code to be shared/used publicly |
| amount | integer | Dollar amount off |
| percent | integer | Percentage off |
| startdate | integer | Foreign key. Refers to a *dateid* from the ***date*** table |
| enddate | integer | Foreign key. Refers to a *dateid* from the ***date*** table |
| tickettypeid\_fk | integer | Foreign key.  References *tickettypeid* in ***tickettype*** table |
| createdby | integer | Foreign key. Refers to a *userid* from the ***users*** table |
| usagelimit | integer | Max number of times the discount code may be used |
| min\_tickets | integer | Minimum number of tickets that must be in the customers order in order for the discount to be valid |
| min\_events | integer | Minimum number of events that must be in the customers order in order for the discount to be valid |

Discounts are identified internally by the database by their *discountid*, and can be given a more human readable code that may be shared publicly (eg “SUMMERSAVE10”). Discounts may be either a dollar amount off an order, or a percentage discount. Both fields may be used in conjunction with one another, in order to provide a percentage off discount with a hard cap to the dollar amount (eg save 20% or $50, whichever is less).

Discounts can be given a date range to constrict their usage, and an overall usage limit. Discounts may also be tied to a specific ticket type, or a specific event.

## orderitems

| orderitems | | |
| --- | --- | --- |
| orderitemid | serial | Primary key. Auto-incrementing integer |
| orderid\_fk | integer | Foreign key. Refers to a *orderid* from the ***orders*** table |
| price | money | Ticket price |

The order items table serves as a junction table between a customer's order and their ticket purchase. Every order consists of one or more order items. The price field contains the nominal or list price of the purchased ticket.

Any given *orderitemid* will be referred to by an entry in the **singletickets** table, or in the **seasontickets** table.

## singletickets

| singletickets | | |
| --- | --- | --- |
| singleticketid | serial | Primary key. Auto-incrementing integer |
| eventticketid\_fk | integer | Foreign key.  Refers to a *eventticketid* from the ***eventtickets*** table |
| orderitemid\_fk | integer | Foreign key.  Refers to a *orderitemid* from the ***orderitem*** table |
| ticketwasswapped | boolean | Flags that a previously selected ticket was changed to a new ticket. Default is False |

The single ticket table models the purchase of a singular ticket/seat to a single event instance. It contains a reference to the *eventticketid* and the *orderitemid*.

There are instances where a customer may not be able to attend the showing they purchased a ticket for, and Portland Playhouse lets customers exchange their tickets for another showing. When this happens the relevant entry in the ***singletickets*** table should be found, and the *ticketwasswapped* field should be set to True, and the original tickets *purchased* field should be changed back to False.

Another row should then be added to the table, containing the original *orderitemid* and the new *eventticketid*. A search of the table for this *orderitemid* will now yield two results, the original entry and the new. This process can be repeated multiple times if needed. The correct/most up to date ticket selection will always be the one with the *ticketwasswapped* field as false.

## seasontickets

| seasontickets | | |
| --- | --- | --- |
| seasonticketid | serial | Primary key. Auto-incrementing integer |
| orderitemid\_fk | integer | Foreign key.  Refers to a *orderitemid* from the ***orderitem*** table |
| eventticketid\_fk | integer | Foreign key.  Refers to a *eventticketid* from the ***eventtickets*** table |
| eventid\_fk | integer | Foreign key. Refers to a *eventid* from the ***events*** table |
| seasontickettypeid\_fk | integer | Foreign key.  Refers to a *seasonticketid* from the ***singletickets*** table |
| ticketwasswapped | boolean | Flags that a previously selected ticket was changed to a new ticket. Default is False |

Customers may purchase a package of tickets for an entire season of events to save money, and this table enables that purchase.

Example: Let us assume that a customer buys a package for the current season, which has 5 events total, 4 of which are marked as eligible for season ticket purchases. The result of this will be 4 entries to this table, one for each of the 4 unique *eventid*. All entries will have the same *orderitem* and *seasontickettype*.

The *eventticketid* field may be filled at the time of row insertion, but it is not required. Portland Playhouse frequently sells season ticket packages before performance dates are finalized so the *eventticketid* defaults to null. This table can then be searched to see which customers have not yet selected tickets for all of their events, and Portland Playhouse can contact them to fulfill their ticket package, at which point the *eventticketid* can then be inserted.

Just like single ticket purchases, ticket selections may be changed using the same procedure of marking the original ticket as unpurchased, setting the *ticketwasswapped* field to True, and inserting a new row with all of the original information and the new *eventticketid*.

## seasontickettype

| seasontickettype | | |
| --- | --- | --- |
| seasontickettypeid | serial | Primary key. Auto-incrementing integer |
| description | integer | Description |
| price | money | Price |

This table stores the description and price of various season ticket packages.

## donations

| donations | | |
| --- | --- | --- |
| donationid | serial | Primary key. Auto-incrementing integer |
| contactid\_fk | integer | Foreign key.  Refers to a *contactid* from the ***contacts*** table |
| isanonymous | boolean | Mark donation as anonymous |
| amount | money | Donation amount |
| donorname | varchar (255) | Name of donor, if not a contact |
| frequency | freq | Custom enum. Can be any one of: “” |
| comments | varchar (255) | Donation comments |
| payment\_intent | varchar (255) | Stripe payment auth/key |
| refund\_intent | varchar (255) | Stripe payment auth/key |
| donationdate | integer | Foreign key. Refers to a *dateid* from the ***date*** table |

The donations table tracks donations made to Portland Playhouse. Donations may be linked to a specific contact/customer, or be made anonymously. Donations can also be one time instances or flagged as recurring. The *payment\_intent* and *refund\_intent* fields are for storing unique ids related to the Stripe payment system. The presence of a value in the *refund\_intent* field indicates a donation has been refunded.

# Misc

## date

| date | | |
| --- | --- | --- |
| dateid | serial | Primary key. Integer representation of a date |
| date\_actual | date | Date in SQL date format |
| day\_name | varchar (9) | Name of the day |
| day\_of\_week | integer | Day of the week (1-7) |
| day\_of\_month | integer | Day of the month (1-31) |
| day\_of\_quarter | integer | Days from the start of the quarter |
| day\_of\_year | integer | Days from the start of the year (1-366) |
| week\_of\_month | integer | Week of the month (1-5) |
| week\_of\_year | integer | Week of the year (1-52) |
| month\_actual | integer | Month of the year (1-12) |
| month\_name | varchar (9) | Name of the month |
| quarter | integer | Quarter of the year (1-4) |
| year\_actual | integer | Year |
| first\_day\_of\_week | date | First day of the week |
| last\_day\_of\_week | date | Last day of the week |
| first\_day\_of\_month | date | First day of the month |
| last\_day\_of\_month | date | Last day of the month |
| first\_day\_of\_quarter | date | First day of the quarter |
| last\_day\_of\_quarter | date | Last day of the quarter |
| first\_day\_of\_year | date | First day of the year |
| last\_day\_of\_year | date | Last day of the year |
| weekend | boolean | Is a weekend day |

The date table provides easy date filtering for reporting on business performance. The primary key for a date is an integer representation of that date (eg May 25, 2022 == 20220525). The rest of the columns allow for precise filtering by days of the week, weekends only, a range of weeks within a year, specific quarters of the year, etc.

Example: The query below returns all event instances that are currently selling tickets and that are scheduled for a weekend.

SELECT \*

FROM eventinstances

JOIN date ON eventinstances.eventdate = date.dateid

WHERE eventinstances.salestatus = true

AND date.weekend = true

ORDER BY eventinstanceid

The code for converting a JS Date object to its integer equivalent used in the database and vice versa are given below:

// converts the databases integer representation of a date into a JS date object

// 20220512 -> 2022-05-12 == May 12 2022

export const parseIntToDate = (d : number) => {

const year = d / 10000 | 0;

d -= year \*10000;

const month = d / 100 | 0;

const day = d - month\*100;

// month-1 because the month field is 0-indexed in JS

return new Date(year, month-1, day);

};

// converts a JS date object to the databases integer representation of the date

// May 12 2022 -> 20220512

export const parseDateToInt = (d : Date) => {

return d.getFullYear()\*10000 + (d.getMonth()+1)\*100 + d.getDate();

};

## savedreports

|  | | |
| --- | --- | --- |
| savedreportid | serial | Primary key. Auto-incrementing integer |
| tablename | varchar (255) |  |
| queryattr | text |  |

Provides a way to store custom reports.

*If you’ve made it this far, I hope this document has served you well and adequately explained the what, why, and how of this database. To all the future capstone developers, I wish you luck on this project and in your burgeoning careers.*

*-E.R. and the 2022 Summer/Fall Capstone team*