

CPSC 304 Project Cover Page

Milestone #: 4

Date: 2023/04/03

Group Number: 46

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Justin Li			contactJustinLi@gmail.com
Zach Chernenko			zach@chernenko.com
Anthony Hayek			anthony382@hotmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

Milestone 4

Project Description: Our project Totally Not Ticketmaster is a tool that can be used to store and view data related to tickets, users, and purchases, as well as perform specific queries to gather information.

In order to develop our project, we used the following tools:

- Node.js - Hosts the server
- React.js - Front-end user interface
- Express - Manages the node server routes, request/response/error handling
- Postgres - Our database management system, this is where our data is stored

Queries are executed using postgres' connection pool query function, which is done after the proper API endpoint is called by the user interface

Schema Differences: Due to time limitations, we ended up going with a much simplified version of our original schema. Our plan of Users, Artists, Tickets, Venues, Coupons, etc. was reduced to just Tickets and Users, with a Purchases relation between them. Further tables would not be necessary to satisfy the project requirements.

Schema (Underline = PK, Bold = FK):

- tickets(ticketid: SERIAL, price: INTEGER, type: VARCHAR(30), artist: VARCHAR(50), date: DATE)
- users(userid: SERIAL, firstname: VARCHAR(30), lastname: VARCHAR(30), email: VARCHAR(50), birthday: DATE) (email must be unique)
- purchases(purchaseid: SERIAL, **userid**: INTEGER, **ticketid**: INTEGER)

```
ticketmanager=# SELECT * FROM tickets;
```

ticketid	price	type	artist	date
1	5	Seated	Drake	2000-01-01
2	5	Seated	Drake	2000-01-01
3	10	Standing	Drake	2000-01-01
4	20	Standing	Adele	2020-03-03
5	20	Standing	Adele	2020-03-03
6	20	Standing	Adele	2020-03-03
7	20	Standing	Adele	2020-03-03
8	20	Standing	Adele	2020-03-03
9	20	Standing	Adele	2020-03-03
10	20	Standing	Adele	2020-03-03
11	15	Seated	Rihanna	2023-05-01
12	20	Seated	Rihanna	2023-05-01
13	25	Seated	Rihanna	2023-05-01
14	30	Seated	Rihanna	2023-05-01
15	33	Seated	Rihanna	2023-05-01
16	3	Seated	Rihanna	2023-05-01
17	50	Seated	Adele	2023-05-01
18	30	Premium	Adele	2023-05-01
19	35	Premium	Adele	2023-05-01
20	20	VIP	Drake	2002-02-01
21	25	VIP	Drake	2002-02-01
22	40	VIP	Drake	2002-02-01
23	50	VIP	Beyonce	2000-02-22
24	50	VIP	Beyonce	2000-02-22
25	60	VIP	Beyonce	2000-02-22

(25 rows)

```
ticketmanager=# SELECT * FROM purchases;
```

purchaseid	userid	ticketid
1	1	1
2	1	4
3	1	11
4	2	5
5	3	6
6	4	7
7	5	8
8	6	9
9	7	10
10	2	12
11	1	23
12	2	20
13	2	24

(13 rows)

```
ticketmanager=# SELECT * FROM users;
```

userid	firstname	lastname	email	birthday
1	Joe	Biden	j@biden.com	0100-01-01
2	Joe	Max	j@max.com	2000-01-01
3	Joe	Marr	j@marr.com	2000-01-01
4	Joe	Bryant	j@br.com	2000-01-01
5	Joe	Erickson	j@e.com	2002-01-01
6	Joe	Patrick	j@p.com	1990-01-01
7	Tiffany	Patrick	t@p.com	1991-01-01
8	Emily	Summer	emily@summer.com	1980-02-01
9	Fabian	Bentley	fabian@b.com	1950-02-01
10	John	Cena	john@cena.com	1950-07-05
11	Alex	Fray	a@fray.com	1967-01-02

(11 rows)

SQL Queries:

Query	Description	Query	Location (server/index.js)
-------	-------------	-------	-------------------------------

INSERT	Add new purchases	"INSERT INTO purchases (purchaseid, userid, ticketid) VALUES(\$1, \$2, \$3) RETURNING *"	Line 16
DELETE	Delete tickets by specific values	`DELETE FROM tickets WHERE \${column} = \$1`, [val]	Line 32
UPDATE	Edit multiple values of a ticket	"UPDATE tickets SET price = \$1, type = \$2, artist = \$3, date = \$4 WHERE ticketid = \$5", [price, type, artist, date, id]	Line 48
SELECTION	Return users who share a given first name	"SELECT * FROM users WHERE firstname = \$1", [firstname]	Line 60
PROJECTION	Dynamically view columns in a table	"SELECT " + input + " FROM " + table	Line 73
JOIN	Return all users who own a ticket for a given artist	"SELECT users.userid, tickets.ticketid, users.firstname, users.lastname FROM tickets INNER JOIN purchases ON tickets.ticketid = purchases.ticketid INNER JOIN users ON purchases.userid =	Line 85
		users.userid WHERE artist = \$1", [artist]	

GROUP BY	Return number of tickets sold for each artist	"SELECT tickets.artist, COUNT(*) FROM tickets INNER JOIN purchases ON tickets.ticketid = purchases.ticketid GROUP BY tickets.artist"	Line 95
HAVING	Return average price of tickets of each type with average of at least 20	"SELECT type, AVG(price) FROM tickets GROUP BY type HAVING 20 <= AVG(price)"	Line 106
NESTED	Return average number of tickets owned per user	"With userTickets AS (SELECT purchases.userid, COUNT(*) AS ticketsPurchased FROM purchases GROUP BY purchases.userid) SELECT AVG(ticketsPurchased) FROM userTickets"	Line 117
DIVISION	Return email of users who own a ticket for every artist	"SELECT U.email FROM users U WHERE NOT EXISTS ((SELECT DISTINCT T.artist FROM tickets T) EXCEPT (SELECT DISTINCT T1.artist FROM purchases P, Tickets T1 WHERE P.userid = U.userid AND P.ticketid = T1.ticketid))"	Line 128

Query Results:

INSERT

Add Purchase

PurchaseID:

UserID:

TicketID:

Add

PurchaseID	UserID	TicketID
1	1	1
2	1	4
3	1	11
4	2	5
5	3	6
6	4	7
7	5	8
8	6	9
9	7	10
10	2	12
11	1	23
12	2	20
13	2	24

Add Purchase

PurchaseID:

UserID:

TicketID:

Add

Purchase Added! (Refresh)

Add Purchase

PurchaseID:

UserID:

TicketID:

Add

UserID Not Found

Add Purchase

PurchaseID:

UserID:

TicketID:

Add

PurchaseID	UserID	TicketID
1	1	1
2	1	4
3	1	11
4	2	5
5	3	6
6	4	7
7	5	8
8	6	9
9	7	10
10	2	12
11	1	23
12	2	20
13	2	24
14	3	21

Add Purchase

PurchaseID:

UserID:

TicketID:

Add

TicketID Not Found

DELETE

Delete By Value

TicketID	Price	Type	Artist	Date	Edit	Delete
0	5	Demo	Adele	2023-02-04T08:00:00.000Z	Edit	Delete
1	5	Demo	Drake	2000-01-01T08:00:00.000Z	Edit	Delete
2	5	Seated	Drake	2000-01-01T08:00:00.000Z	Edit	Delete

Add

PurchaseID	UserID	TicketID
0	7	0
1	7	1

Delete Tickets

Enter Column

Enter Value

Delete

Close

Delete Tickets

Type

Demo

Row(s) Deleted (Refresh)

Delete

Close

Delete By Value

TicketID	Price	Type	Artist	Date	Edit	Delete
2	5	Seated	Drake	2000-01-01T08:00:00.000Z	Edit	Delete
3	10	Standing	Drake	2000-01-01T08:00:00.000Z	Edit	Delete
4	20	Standing	Adele	2020-03-03T08:00:00.000Z	Edit	Delete

Add

PurchaseID	UserID	TicketID
2	1	4
3	1	11

UPDATE

TicketID	Price	Type	Artist	Date	Edit	Delete
1	5	Seated	Drake	2000-01-01T08:00:00.000Z	Edit	Delete
2	5	Seated	Drake	2000-01-01T08:00:00.000Z	Edit	Delete
3	10	Standing	Drake	2000-01-01T08:00:00.000Z	Edit	Delete

Edit Ticket

5

Seated

Drake

01/01/2000

Edit

Close

Edit Ticket

10

Demo

Usher

01/01/2000

Row Updated (Refresh)

Edit

Close

TicketID	Price	Type	Artist	Date	Edit	Delete
1	10	Demo	Usher	2000-01-01T08:00:00.000Z	Edit	Delete
2	5	Seated	Drake	2000-01-01T08:00:00.000Z	Edit	Delete
3	10	Standing	Drake	2000-01-01T08:00:00.000Z	Edit	Delete

SELECTION

SELECTION

Returns all users who's first name matches the given name

UserID	First Name	Last Name	Email	Birthday
--------	------------	-----------	-------	----------

SELECTION

Returns all users who's first name matches the given name

UserID	First Name	Last Name	Email	Birthday
--------	------------	-----------	-------	----------

SELECTION

Returns all users who's first name matches the given name

UserID	First Name	Last Name	Email	Birthday
1	Joe	Biden	j@biden.com	0100-01-01T08:12:28.000Z
2	Joe	Max	j@max.com	2000-01-01T08:00:00.000Z
3	Joe	Marr	j@marr.com	2000-01-01T08:00:00.000Z
4	Joe	Bryant	j@br.com	2000-01-01T08:00:00.000Z
5	Joe	Erickson	j@e.com	2002-01-01T08:00:00.000Z
6	Joe	Patrick	j@p.com	1990-01-01T08:00:00.000Z

PROJECTION

Table Viewer

Currently Viewing:

Table Viewer

Currently Viewing: users

userid,firstname,email

users

View Table

Table Viewer

Currently Viewing: users

userid,firstname,email

users

View Table

userid	firstname	email
1	Joe	j@biden.com
2	Joe	j@max.com
3	Joe	j@marr.com
4	Joe	j@br.com
5	Joe	j@e.com
6	Joe	j@p.com
7	Tiffany	t@p.com
8	Emily	emily@summer.com
9	Fabian	fabian@b.com
10	John	john@cena.com
11	Alex	a@fray.com

JOIN

JOIN

Returns all users who own a ticket for a given artist

UserID

TicketID

First Name

Last Name

Enter Artist

Query

JOIN

Returns all users who own a ticket for a given artist

UserID	TicketID	First Name	Last Name
--------	----------	------------	-----------

Query

JOIN

Returns all users who own a ticket for a given artist

UserID	TicketID	First Name	Last Name
2	20	Joe	Max
3	21	Joe	Marr

Query

GROUP

GROUP

Returns the number of tickets sold for each artist

Artist	# Tickets Sold
--------	----------------

Query

GROUP

Returns the number of tickets sold for each artist

Artist	# Tickets Sold
Drake	2
Rihanna	2
Adele	7
Beyonce	2

Query

HAVING

HAVING

Returns average price of sold tickets for each type with average greater than 20

Type	Average Price of Sold Tickets
------	-------------------------------

Query

HAVING

Returns average price of sold tickets for each type with average greater than 20

Type	Average Price of Sold Tickets
Premium	32.5000000000000000
Seated	20.6666666666666667
VIP	40.8333333333333333

Query

NESTED

NESTED

Returns the average number of tickets owned per user

Average # of Tickets Owned Per User	Rounded Down
-------------------------------------	--------------

Query

NESTED

Returns the average number of tickets owned per user

Average # of Tickets Owned Per User	Rounded Down
1.8571428571428571	1

Query

DIVISION

DIVISION

Returns the email of users who own a ticket for every artist

Email

Query

DIVISION

Returns the email of users who own a ticket for every artist

Email

j@biden.com

j@max.com

Query