Project 5: Physical Database Design and Tuning

Database Planning and Requirement Analysis
Event Management System: OccasionOrganizer

By

Group ID: 15

Miss Chommakorn	**Sontesadisai**	**6488189**
Miss Nattanicha	**Sinsawet**	**6488190**
Miss Ravikarn	**Jarungjitvittawas**	**6488210**

To

Asst. Prof. Dr. Charnyote Pluempitiwiriyawej

A Report Submitted in Partial Fulfillment of
the Requirements for

ITCS413 Database Design

Faculty of Information and Communication Technology
Mahidol University

2023

Last Updated: May 3, 2024

Table of Content

The database application requirements including description of required transactions (or queries) with highlighted the selected transactions (or queries)

Transaction requirements

- **Data Entry**
    - Enter the details for a new attendee and the account's attendee data (such as details of email contacts, username, and password).
    - Enter the details of the concert (such as location, concert name, date, start time, and end time)
    - Enter the details of sponsor (such as sponsor name, sponsor detail, amount, concert name, id)
    - Enter the details of the equipped equipment (such as the equipment name, equipment type, status, quantity, cost planned, cost actual, concert ID, and concert name)
    - Enter the details of the artist that performs the concert (such as full name, genre, date, start time, end time, and contact details)
    - Enter the details of administrative data (such as first_name, last_name, username, and password)
- **Data update/deletion**
    - Update/delete the details for a new attendee.
    - Update/delete the details of the concert.
    - Update/delete the details of the sponsor.
    - Update/delete the details of the equipped equipment.
    - Update/delete the details of the artist.
    - Update/delete the details of administrative data.
- **Data queries**
    - (a) List the details of the ticket by the attendee who owned the ticket.
    - (b) List the details of the concert ascending by ticket ID.
    - (c) List the details of the location where that concert was performed.

(d) List the details of the admin who created the concert in the system.

(e) List the details of the sponsor who supported the concert.

(f) List the details of the name, genre, and contact details that the artist performed.

(g) List the details of the equipment and quantity, cost planned, and actual cost that will be used in the concert.

(h) Identify the timestamp of the date, start time, and end time of the artist who participated in the concert.

Final selected transactions

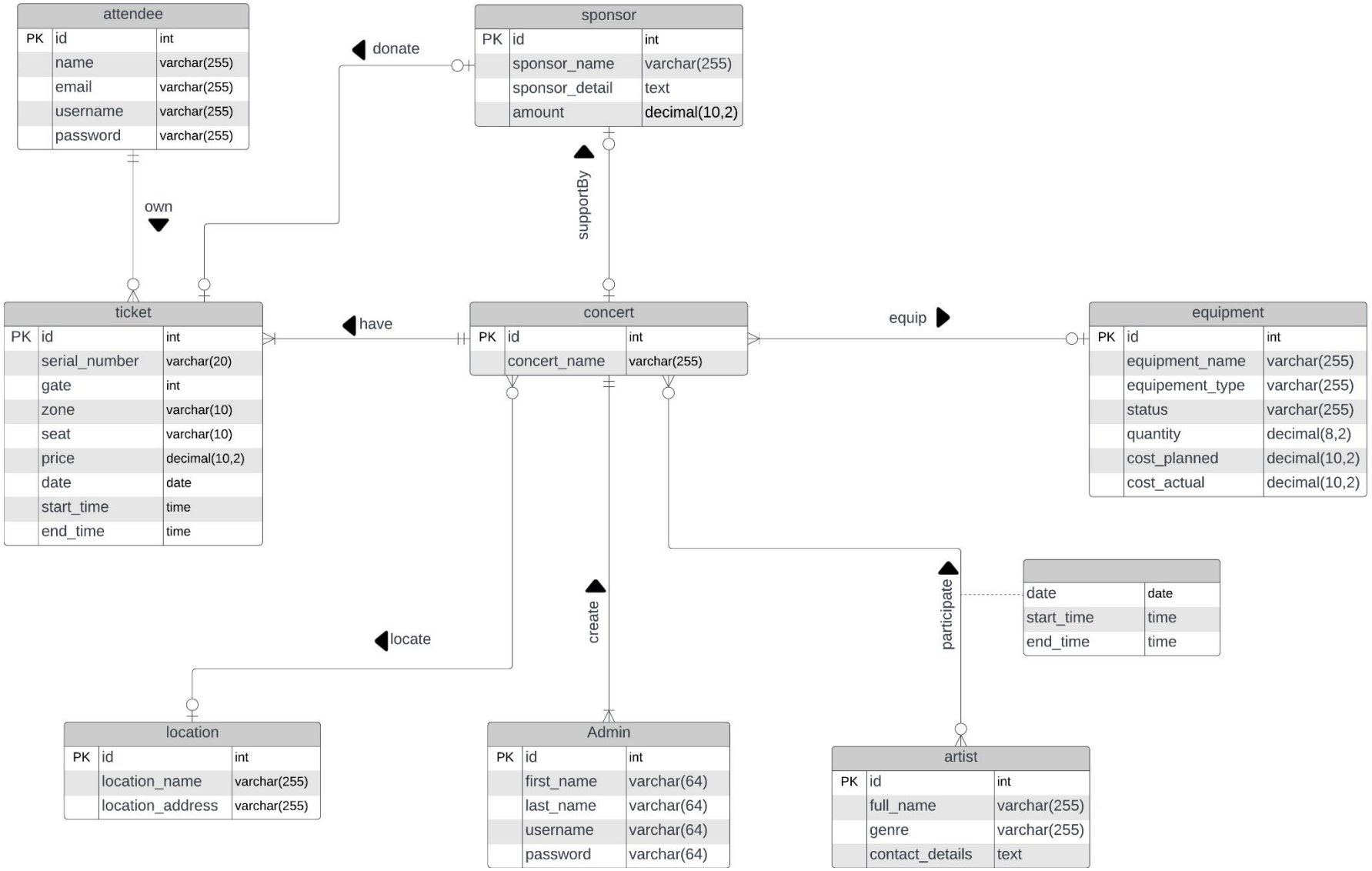
- List the details of the ticket by the attendee who owned the ticket
  - This list displays the names, emails, and account of concert ticket holders, along with tickets they purchased. This information serves as a record of ticket purchases and allows users to redeem their tickets and access their concert history. Additionally, the list includes details about the concert, such as serial number, seating, zone, gate entrance, performance date, and start and end times.
- List the details of the concert ascending by ticket ID
  - This provides comprehensive concert ticket information, including serial number, seating, gate, zone, price, date, time, and the concert name. This clearly identifies the concert and accurately reflects the number of tickets issued for each concert, and the system also completely stores concert data.
- List the details of the sponsor who supported the concert
  - This gathers information on sponsor-provided tickets, totaling 50,000 tickets across over 200,000 tickets in total from 8 concerts. It displays information such as attendee name, concert name, ticket price, zone, seating, sponsoring company name, and sponsor details.

The final conceptual database model (i.e., ER diagram) with highlighted the portion that related to the selected transactions (or queries)

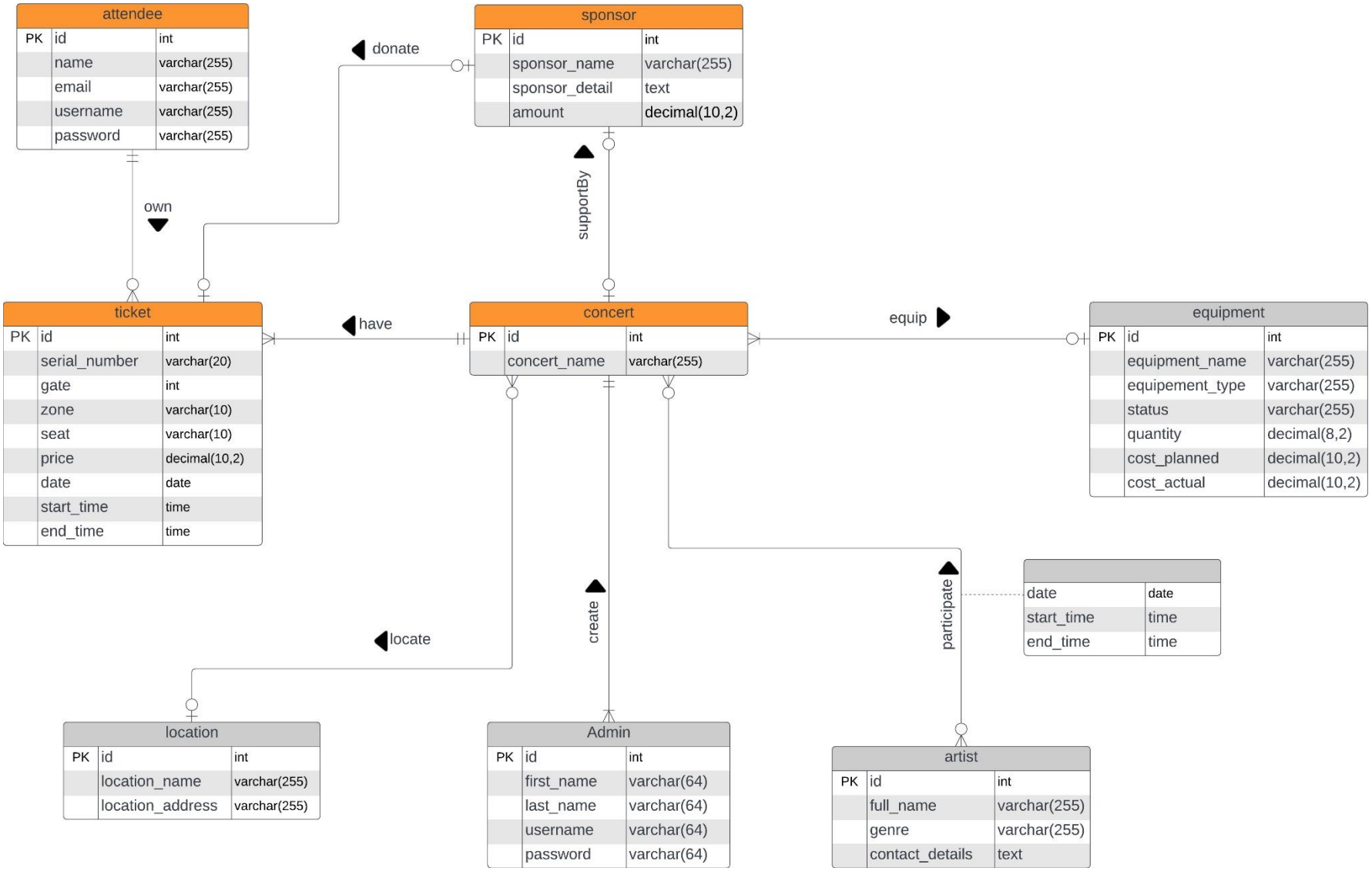
The final ER diagram represents the conceptual database model for a concert ticket booking system. It illustrates the key entities involved, such as attendee, ticket, concert, admin, sponsor, location, equipment, and artist, along with their attributes and relationships.

The diagram helps visualize how different pieces of data are connected. For example, an attendee can own tickets, each ticket is for a specific concert, concerts are located at certain locations, and have both sponsors and performing artists associated with them. Equipment is another entity that relates to concerts.

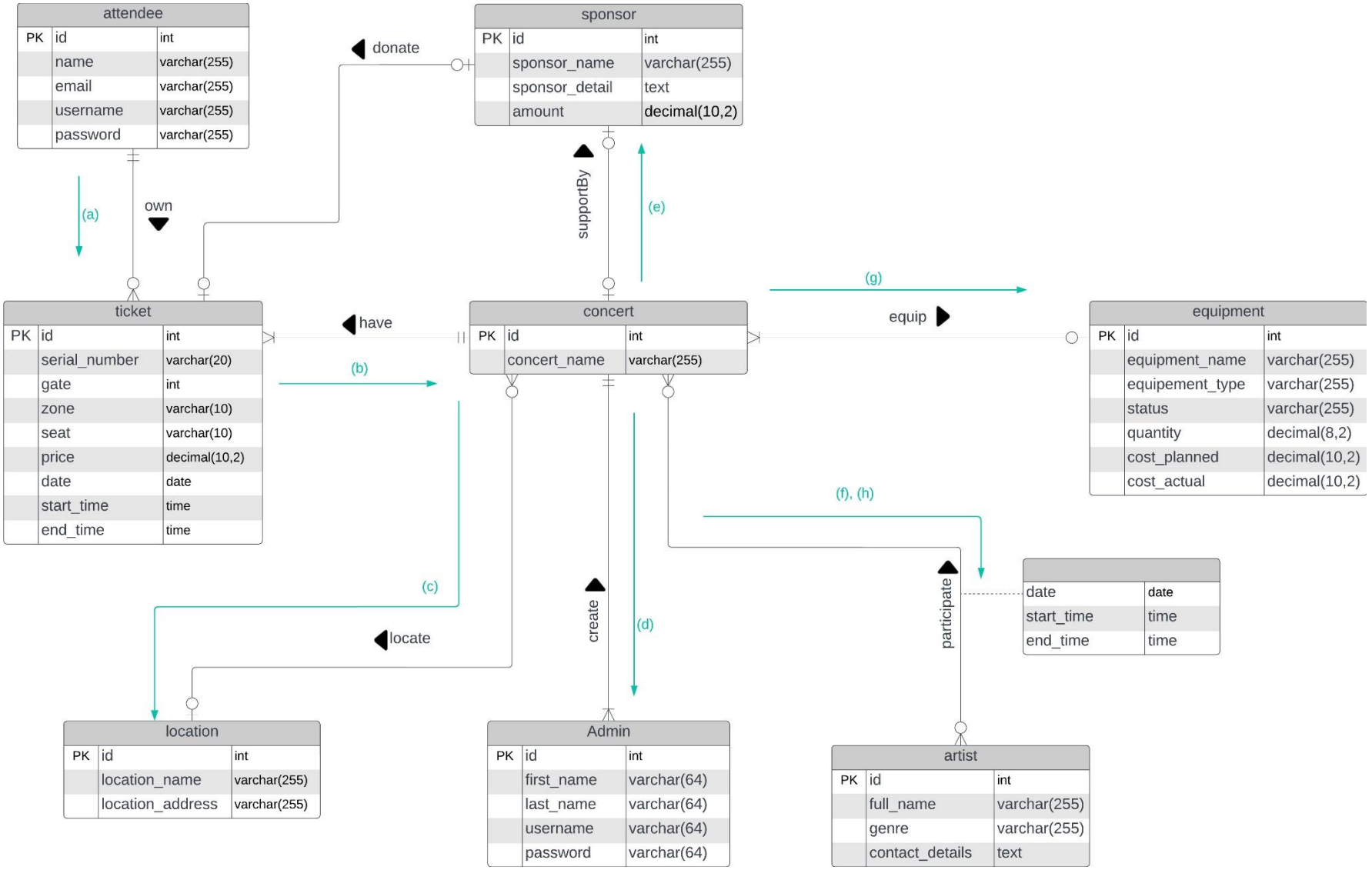
Final - ER diagram



Highlighted - ER diagram



Pathway


attendee
PK id int
name varchar(255)
email varchar(255)
username varchar(255)
password varchar(255)
sponsor
PK id int
sponsor_name varchar(255)
sponsor_detail text
amount decimal(10,2)
donate
own
(a)
supportBy
(e)
ticket
PK id int
serial_number varchar(20)
gate int
zone varchar(10)
seat varchar(10)
price decimal(10,2)
date date
start_time time
end_time time
have
(b)
concert
PK id int
concert_name varchar(255)
(g)
equip
equipment
PK id int
equipment_name varchar(255)
equipement_type varchar(255)
status varchar(255)
quantity decimal(8,2)
cost_planned decimal(10,2)
cost_actual decimal(10,2)
(f), (h)
participate
date date
start_time time
end_time time
(c)
locate
create
(d)
location
PK id int
location_name varchar(255)
location_address varchar(255)
Admin
PK id int
first_name varchar(64)
last_name varchar(64)
username varchar(64)
password varchar(64)
artist
PK id int
full_name varchar(255)
genre varchar(255)
contact_details text

The final logical database model (i.e., Relational database schema) with highlighted the portion that related to the selected transactions (or queries)

Relation Schema

- **<u>Attributes</u>** – which are bold and underlined are the Primary Keys
- *Attributes* – which are Italic are the Foreign Keys
- ***<u>Attributes</u>*** – which are bold, italic and underlined are both Primary Keys and Foreign Keys

Final

- Attendee

attendee_id	name	email	username	password

- Ticket

ticket_id	serial_number	gate	zone	seat	price	date	start_time	end_time	*attendee_id*	*concert_id*

- Concert

concert_id	concert_name	date	start_time	end_time	*sponsor_id*	sponsor_name	sponsor_detail	amount	*location_id*	*equipment_id*	*artist_id*

- Location

location_id	location_name	location_address

- Admin

admin_id	first_name	last_name	username	password	*concert_id*

- Artist

artist_id	full_name	genre	contact_details	*location_id*

- Equipment

equipment_id	equipment _name	equipment_type	status	quantity	cost_planned	cost_actual

Highlighted

- Attendee

attendee_id	name	email	username	password

- Ticket

ticket_id	serial_number	gate	zone	seat	price	date	start_time	end_time	*attendee_id*	*concert_id*

- Concert

concert_id	concert_name	date	start_time	end_time	*sponsor_id*	sponsor_name	sponsor_detail	amount	*location_id*	*equipment_id*	*artist_id*

- Location

location_id	location_name	location_address

- Admin

admin_id	first_name	last_name	username	password	*concert_id*

- Artist

artist_id	full_name	genre	contact_details	*location_id*

- Equipment

equipment_id	equipment _name	equipment_type	status	quantity	cost_planned	cost_actual

The SQL commands only related to the selected transactions (or queries) and the specification of the related table

(a) List the details of the ticket by the attendee who owned the ticket

- Select

```sql
-- (a) List the details of the ticket by the attendee who owned the ticket
SELECT
    Attendee.name,
    Attendee.email,
    Ticket.serial_number,
    Ticket.gate,
    Ticket.zone,
    Ticket.seat,
    Ticket.price,
    Ticket.date,
    Ticket.start_time,
    Ticket.end_time
FROM
    Ticket
JOIN
    Attendee ON Ticket.attendee_id = Attendee.id;
```

- Result

SELECT Attendee.name, Attendee.email, Ticket.serial_number, Ticket.gate, Ticket.zone, Ticket.seat, Ticket.price... 227133 row(s) returned

name	email	serial_number	gate	zone	seat	price	date	start_time	end_time
Subin Harnphadungkit	subin.harnphadungkit@example.com	LIU-0600001	1	A3	A44	6500.00	2024-06-29	17:00:00	21:00:00
Bawontat Chatchonbut	bawontat.chatchonbut@example.com	LIU-0600002	1	A3	A45	6500.00	2024-06-29	17:00:00	21:00:00
Chupong Aroonwatanaporn	chupong.aroonwatanaporn@example.com	LIU-0600003	1	A3	A46	6500.00	2024-06-29	17:00:00	21:00:00
Chupong Aroonwatanaporn	chupong.aroonwatanaporn@example.com	LIU-0600004	1	A3	A47	6500.00	2024-06-29	17:00:00	21:00:00

- Create Index

```
-- Create index
CREATE INDEX idx_ticket_attendee_id ON Ticket (attendee_id);
CREATE INDEX idx_attendee_id ON Attendee (id);
CREATE UNIQUE INDEX idx_ticket_serial_number ON Ticket (serial_number);
```

- The number records in the table

```
-- The number records in the table
SELECT COUNT(*)
FROM Ticket
JOIN Attendee ON Ticket.attendee_id = Attendee.id;
```

	COUNT(*)
▶	227133

- The record sizes

```
-- The record sizes
SELECT
    SUM(
        LENGTH(Attendee.name) +
        LENGTH(Attendee.email) +
        LENGTH(Ticket.serial_number) +
        LENGTH(Ticket.gate) +
        LENGTH(Ticket.zone) +
        LENGTH(Ticket.seat) +
        LENGTH(CAST(Ticket.price AS CHAR)) +
        LENGTH(CAST(Ticket.date AS CHAR)) +
        LENGTH(CAST(Ticket.start_time AS CHAR)) +
        LENGTH(CAST(Ticket.end_time AS CHAR))
    ) AS total_record_size
FROM Ticket
JOIN Attendee ON Ticket.attendee_id = Attendee.id;
```

	total_record_size
▶	23533094

(b) List the details of the concert ascending by ticket ID

- Select

```
SELECT
    Ticket.id AS TicketID,
    Ticket.serial_number AS SerialNumber,
    Ticket.gate AS Gate,
    Ticket.zone AS Zone,
    Ticket.seat AS Seat,
    Ticket.price AS Price,
    Ticket.date AS Date,
    Ticket.start_time AS StartTime,
    Ticket.end_time AS EndTime,
    Concert.concert_name AS ConcertName
FROM
    Ticket
INNER JOIN
    Concert
ON
    Ticket.concert_id = Concert.id
ORDER BY
    TicketID ASC;
```

- Result

```
SELECT    Ticket.id AS TicketID,    Ticket.serial_number AS SerialNumber,    Ticket.gate AS Gate,    Ticket.zone AS Zone,    Ti...    227133 row(s) returned
```

TicketID	SerialNumber	Gate	Zone	Seat	Price	Date	StartTime	EndTime	ConcertName
300330822	LIU-0630823	1	F5	FK99	3500.00	2024-06-30	17:00:00	21:00:00	2024 IU H.E.R. WORLD TOUR CONCERT IN BA...
300330823	LIU-0630824	1	F5	FK100	3500.00	2024-06-30	17:00:00	21:00:00	2024 IU H.E.R. WORLD TOUR CONCERT IN BA...
300330824	NCT-1270000	1	AL	STA...	4800.00	2019-06-21	20:00:00	23:00:00	NCT 127 WORLD TOUR 'NEO CITY : BANGKOK' ...
300330825	NCT-1270001	1	AL	STA...	4800.00	2019-06-21	20:00:00	23:00:00	NCT 127 WORLD TOUR 'NEO CITY : BANGKOK' ...

- Create Index

```
-- Create index
CREATE INDEX idx_ticket_concert_id ON Ticket(concert_id);
CREATE INDEX idx_ticket_id_concert ON Ticket(id, concert_id);
```

- The number records in the table

```
-- The number records in the table
SELECT COUNT(*)
FROM Ticket
INNER JOIN Concert ON Ticket.concert_id = Concert.id;
```

	COUNT(*)
▶	227133

- The record sizes

```
-- The record sizes
SELECT
    SUM(
        LENGTH(CAST(Ticket.id AS CHAR)) +
        LENGTH(Ticket.serial_number) +
        LENGTH(Ticket.gate) +
        LENGTH(Ticket.zone) +
        LENGTH(Ticket.seat) +
        LENGTH(CAST(Ticket.price AS CHAR)) +
        LENGTH(CAST(Ticket.date AS CHAR)) +
        LENGTH(CAST(Ticket.start_time AS CHAR)) +
        LENGTH(CAST(Ticket.end_time AS CHAR)) +
        LENGTH(Concert.concert_name)
    ) AS total_record_size
FROM Ticket
INNER JOIN Concert ON Ticket.concert_id = Concert.id;
```

	total_record_size
▶	22896703

(e) List the details of the sponsor who supported the concert

- Select

```
ALTER TABLE Ticket
ADD COLUMN sponsor_id INT;

UPDATE Ticket
SET sponsor_id = 5001 + MOD(CEIL((id - 300300000) / 200), 110)
WHERE id BETWEEN 300300000 AND 300527132;

SELECT
    A.name,
    T.price,
    T.zone,
    T.seat,
    C.concert_name,
    S.sponsor_name,
    S.sponsor_detail
FROM
    Attendee A
JOIN
    Ticket T ON A.id = T.attendee_id
JOIN
    Concert C ON T.concert_id = C.id
JOIN
    Sponsor S ON T.sponsor_id = S.id
ORDER BY RAND()
LIMIT 50000;
```

- Result

```
SELECT  A.name,  T.price,  T.zone,  T.seat,  C.concert_name,  S.sponsor_name,  S.sponsor_detail FROM  Atte...  50000 row(s) returned
```

	name	price	zone	seat	concert_name	sponsor_name	sponsor_detail
	Pimpan Nongnuch	5500.00	AL	STANDING	789 SPECIAL STAGE THE TIME CAPSULE	Predator Thailand	Technology sponsor
	Chintana Piyaseth	4500.00	C	C19	NCT 127 WORLD TOUR 'NEO CITY : BANGKOK' ...	Sizzler	Food and beverage partner
	Ganokphorn Wongkamlao	4500.00	I	H12	Wanna One World Tour < ONE : THE WORLD >...	7-Eleven Thailand	Convenience store partner
	Pruethichai Likitawong	6000.00	SJ	D20	NCT DREAM TOUR 'THE DREAM SHOW2 : In A ...	Maybelline New York T...	Makeup sponsor
	Putthinart Phattaravinij	5500.00	ST	STANDING	BRUNO MARS LIVE IN BANGKOK	JOOX	Official music streaming partner

- Create Index

```
-- Create index
CREATE INDEX idx_concert_id ON Ticket(concert_id);
CREATE INDEX idx_sponsor_id ON Ticket(sponsor_id);
```

- The number records in the table

```
-- The number records in the table
SELECT COUNT(*)
FROM Attendee A
JOIN Ticket T ON A.id = T.attendee_id
JOIN Concert C ON T.concert_id = C.id
JOIN Sponsor S ON T.sponsor_id = S.id;
```

	COUNT(*)
▶	225133

- The record sizes

```
-- The record sizes
SELECT
    SUM(
        LENGTH(A.name) +
        LENGTH(CAST(T.price AS CHAR)) +
        LENGTH(T.zone) +
        LENGTH(T.seat) +
        LENGTH(C.concert_name) +
        LENGTH(S.sponsor_name) +
        LENGTH(S.sponsor_detail)
    ) AS total_record_size
FROM Attendee A
JOIN Ticket T ON A.id = T.attendee_id
JOIN Concert C ON T.concert_id = C.id
JOIN Sponsor S ON T.sponsor_id = S.id;
```

	total_record_size
▶	23764222

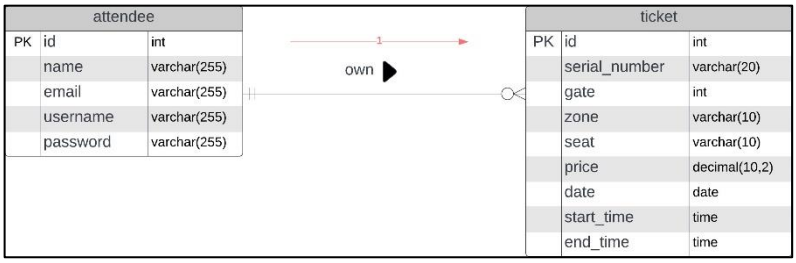
The results of the analysis of each selected transaction before improvement

(a) List the details of the ticket by the attendee who owned the ticket

Transaction Analysis				
April 30, 2024				
Transaction	(a) List the details of the ticket by the attendee who owned the ticket			
Volume	Average:	1000 per hour		
	Peak:	100,000 per hour (between 10.00-12.00 Friday-Saturday)		

```
SELECT
   Attendee.name,
   Attendee.email,
   Ticket.serial_number,
   Ticket.gate,
   Ticket.zone,
   Ticket.seat,
   Ticket.price,
   Ticket.date,
   Ticket.start_time,
   Ticket.end_time
FROM
   Ticket
JOIN
   Attendee ON Ticket.attendee_id = Attendee.id;
```

Predicate:	No predicate specified (no WHERE clause)
Join attributes:	Attendee ON Ticket.attendee_id = Attendee.id;
Ordering attribute:	Not specified
Grouping attribute:	Not used
Built-in functions:	-
Attribute Updated:	N/A (this is a SELECT query, no update performed)



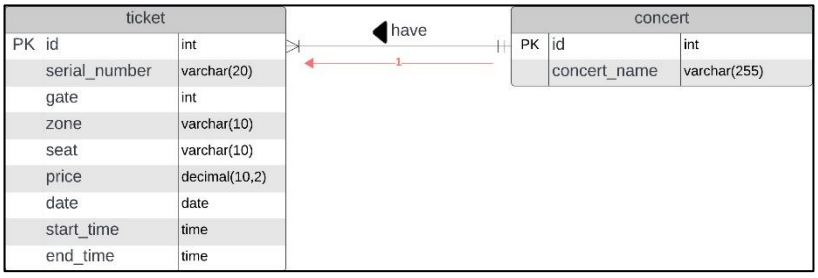
Access	Entry	Type of Access	No. of References		
			Per Transaction	Per Transaction	Peak Per Hour
1	Ticket	R	225,133	227,133	250,000
2	Attendee	R	293,941	293,941	300,000
Total References			519,074	519,074	550,000

(b) List the details of the concert ascending by ticket ID

Transaction Analysis				
April 30, 2024				
Transaction	(b) List the details of the concert ascending by ticket ID			
Volume	Average:	1000 per hour		
	Peak:	100,000 per hour (between 10.00-12.00 Friday-Saturday)		

```
SELECT
   Ticket.id AS TicketID,
   Ticket.serial_number AS SerialNumber,
   Ticket.gate AS Gate,
   Ticket.zone AS Zone,
   Ticket.seat AS Seat,
   Ticket.price AS Price,
   Ticket.date AS Date,
   Ticket.start_time AS StartTime,
   Ticket.end_time AS EndTime,
   Concert.concert_name AS ConcertName
FROM
   Ticket
INNER JOIN
   Concert
ON
   Ticket.concert_id = Concert.id
ORDER BY
   TicketID ASC;
```

Predicate:	No predicate specified (no WHERE clause)
Join attributes:	INNER JOIN Concert ON Ticket.concert_id = Concert.id
Ordering attribute:	TicketID ASC;
Grouping attribute:	Not used
Built-in functions:	-
Attribute Updated:	N/A (this is a SELECT query, no update performed)



Access	Entry	Type of Access	No. of References		
			Per Transaction	Per Transaction	Peak Per Hour
1	Ticket	R	227,133	227,133	250,000
2	Concert	R	70	70	200
Total References			227,203	227,203	250,200

(e) List the details of the sponsor who supported the concert

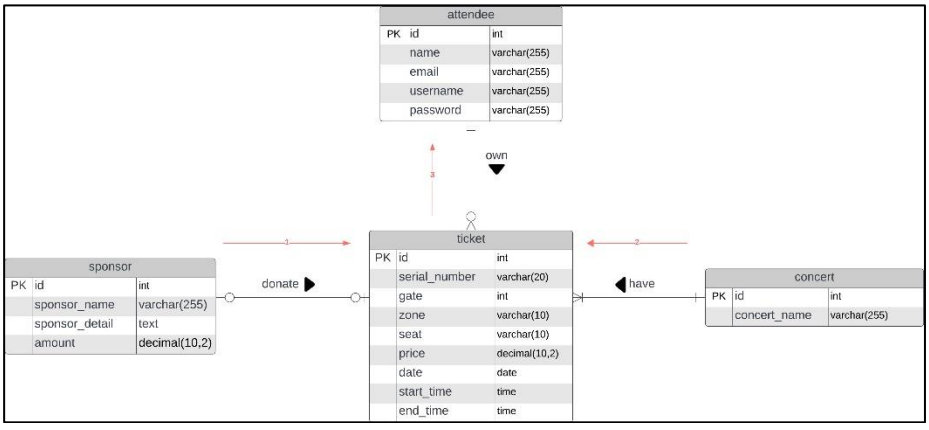
Transaction Analysis		
April 30, 2024		
Transaction	(e) List the details of the sponsor who supported the concert	
Volume	Average:	1000 per hour
	Peak:	100,000 per hour (between 10.00-12.00 Friday-Saturday)

```
ALTER TABLE Ticket
ADD COLUMN sponsor_id INT;

UPDATE Ticket
SET sponsor_id = 5001 + MOD(CEIL((id - 300300000) / 200), 110)
WHERE id BETWEEN 300300000 AND 300527132;

SELECT
   A.name,
   T.price,
   T.zone,
   T.seat,
   C.concert_name,
   S.sponsor_name,
   S.sponsor_detail
FROM
   Attendee A
JOIN
   Ticket T ON A.id = T.attendee_id
JOIN
   Concert C ON T.concert_id = C.id
JOIN
   Sponsor S ON T.sponsor_id = S.id
ORDER BY RAND()
LIMIT 50000;
```

Predicate:	- UPDATE: WHERE id BETWEEN 300300000 AND 300527132 - SELECT: No predicate (the ORDER BY RAND() and LIMIT are not filtering conditions)
Join attributes:	- Attendee.id = Ticket.attendee_id - Ticket.concert_id = Concert.id - Ticket.sponsor_id = Sponsor.id
Ordering attribute:	RAND()
Grouping attribute:	Not used
Built-in functions:	- MOD(), CEIL() in UPDATE - RAND() in SELECT
Attribute Updated:	sponsor_id in Ticket table



Access	Entry	Type of Access	No. of References		
			Per Transaction	Per Transaction	Peak Per Hour
1	Ticket	R	227,133	227,133	250,000
2	Ticket	W	227,133	227,133	250,000
3	Attendee	R	293,941	293,941	300,000
4	Concert	R	70	70	200
5	Sponsor	R	109	109	300
Total References			748,386	748,386	800,500

The results of the analysis of each selected transaction after improvement

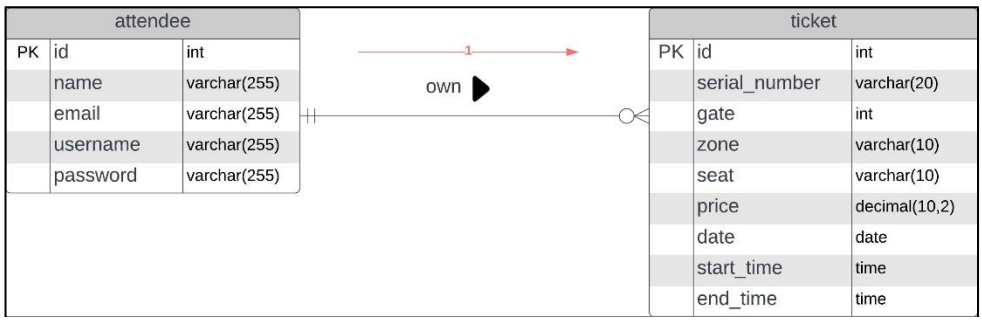
(a) List the details of the ticket by the attendee who owned the ticket

<table>
<tr><th colspan="4">Transaction Analysis</th></tr>
<tr><td colspan="4">May 1, 2024</td></tr>
<tr><td>Transaction</td><td colspan="3">(a) List the details of the ticket by the attendee who owned the ticket</td></tr>
<tr><td rowspan="2">Volume</td><td>Average:</td><td colspan="2">1000 per hour</td></tr>
<tr><td>Peak:</td><td colspan="2">100,000 per hour (between 10.00-12.00 Friday-Saturday)</td></tr>
<tr><td rowspan="7">SELECT
Attendee.name,
Attendee.email,
Ticket.serial_number,
Ticket.gate,
Ticket.zone,
Ticket.seat,
Ticket.price,
Ticket.date,
Ticket.start_time,
Ticket.end_time
FROM
Ticket
JOIN
Attendee ON Ticket.attendee_id = Attendee.id;

-- Create index

CREATE INDEX idx_ticket_attendee_id ON Ticket (attendee_id);

CREATE INDEX idx_attendee_id ON Attendee (id);

CREATE UNIQUE INDEX idx_ticket_serial_number ON Ticket (serial_number);</td><td>Predicate:</td><td colspan="2">No predicate specified (no WHERE clause)</td></tr>
<tr><td>Join attributes:</td><td colspan="2">Attendee ON Ticket.attendee_id = Attendee.id;</td></tr>
<tr><td>Ordering attribute:</td><td colspan="2">Not specified</td></tr>
<tr><td>Grouping attribute:</td><td colspan="2">Not used</td></tr>
<tr><td>Built-in functions:</td><td colspan="2">-</td></tr>
<tr><td>Attribute Updated:</td><td colspan="2">N/A (this is a SELECT query, no update performed)</td></tr>
<tr><td colspan="4"> 
</td></tr>
</table>

Access	Entry	Type of Access	No. of References		
			Per Transaction	Per Transaction	Peak Per Hour
1	Ticket	R	225,133	227,133	250,000
2	Attendee	R	293,941	293,941	300,000
Total References			519,074	519,074	550,000

(b) List the details of the concert ascending by ticket ID

Transaction Analysis				
May 1, 2024				
Transaction	(b) List the details of the concert ascending by ticket ID			
Volume	Average:	1000 per hour		
	Peak:	100,000 per hour (between 10.00-12.00 Friday-Saturday)		

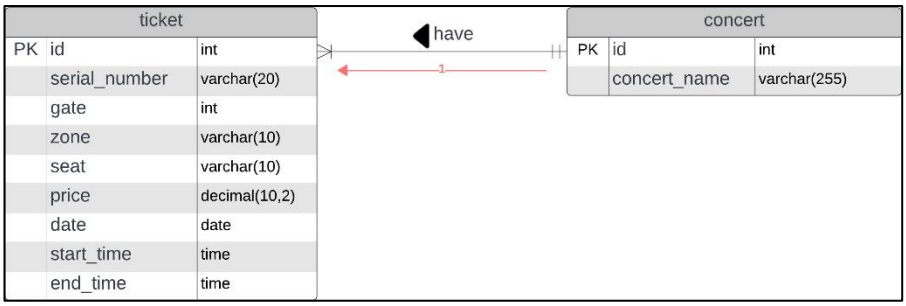
```
SELECT
   Ticket.id AS TicketID,
   Ticket.serial_number AS SerialNumber,
   Ticket.gate AS Gate,
   Ticket.zone AS Zone,
   Ticket.seat AS Seat,
   Ticket.price AS Price,
   Ticket.date AS Date,
   Ticket.start_time AS StartTime,
   Ticket.end_time AS EndTime,
   Concert.concert_name AS ConcertName
FROM
   Ticket
INNER JOIN
   Concert
ON
   Ticket.concert_id = Concert.id
ORDER BY
   TicketID ASC;

-- Create index

CREATE INDEX idx_ticket_concert_id ON Ticket(concert_id);

CREATE INDEX idx_ticket_id_concert ON Ticket(id,
concert_id);
```

Predicate:	No predicate specified (no WHERE clause)
Join attributes:	INNER JOIN Concert ON Ticket.concert_id = Concert.id
Ordering attribute:	TicketID ASC;
Grouping attribute:	Not used
Built-in functions:	-
Attribute Updated:	N/A (this is a SELECT query, no update performed)



Access	Entry	Type of Access	No. of References		
			Per Transaction	Per Transaction	Peak Per Hour
1	Ticket	R	227,133	227,133	250,000
2	Concert	R	70	70	200
Total References			227,203	227,203	250,200

(e) List the details of the sponsor who supported the concert

Transaction Analysis				
				May 2, 2024
Transaction	(e) List the details of the sponsor who supported the concert			
Volume	Average:	1000 per hour		
	Peak:	100,000 per hour (between 10.00-12.00 Friday-Saturday)		
ALTER TABLE Ticket ADD COLUMN sponsor_id INT; UPDATE Ticket SET sponsor_id = 5001 + MOD(CEIL((id - 300300000) / 200), 110) WHERE id BETWEEN 300300000 AND 300527132; SELECT A.name, T.price, T.zone, T.seat, C.concert_name, S.sponsor_name,			Predicate:	- UPDATE: WHERE id BETWEEN 300300000 AND 300527132 - SELECT: No predicate (the ORDER BY RAND() and LIMIT are not filtering conditions)
			Join attributes:	- Attendee.id = Ticket.attendee_id - Ticket.concert_id = Concert.id - Ticket.sponsor_id = Sponsor.id
			Ordering attribute:	RAND()
			Grouping attribute:	Not used

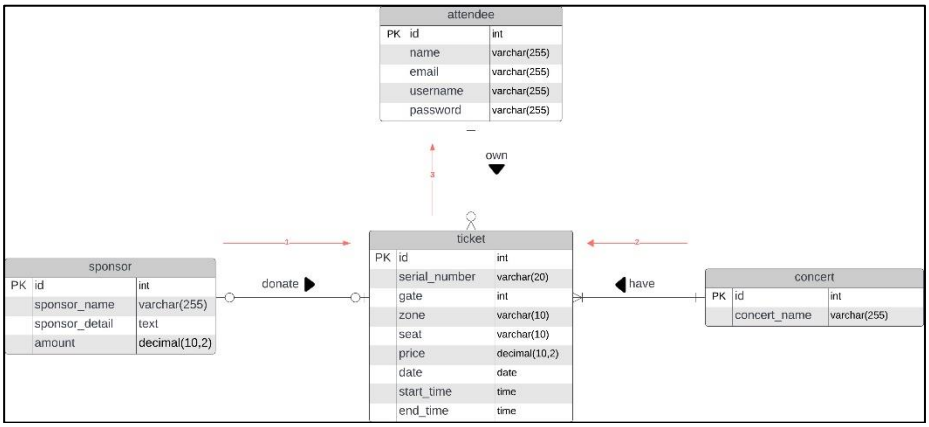
```
   S.sponsor_detail
FROM
   Attendee A
JOIN
   Ticket T ON A.id = T.attendee_id
JOIN
   Concert C ON T.concert_id = C.id
JOIN
   Sponsor S ON T.sponsor_id = S.id
ORDER BY RAND()
LIMIT 50000;

-- Create index

CREATE INDEX idx_concert_id ON Ticket(concert_id);

CREATE INDEX idx_sponsor_id ON Ticket(sponsor_id);
```

Built-in functions:	- MOD(), CEIL() in UPDATE - RAND() in SELECT
Attribute Updated:	sponsor_id in Ticket table



Access	Entry	Type of Access	No. of References		
			Per Transaction	Per Transaction	Peak Per Hour
1	Ticket	R	227,133	227,133	250,000
2	Ticket	W	227,133	227,133	250,000
3	Attendee	R	293,941	293,941	300,000
4	Concert	R	70	70	200
5	Sponsor	R	109	109	300
Total References			748,386	748,386	800,500

The discussion on the results

(a) List the details of the ticket by the attendee who owned the ticket

#	Time	Action	Message	Duration / Fetch
81	21:28:39	SELECT Attendee.name, Attendee.email, Ticket.serial_number, Ticket.gate, Ticket.zone, Ticket.seat, Ticket.price...	227133 row(s) returned	0.000 sec / 0.531 sec
82	21:28:47	CREATE INDEX idx_ticket_attendee_id ON Ticket (attendee_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.609 sec
83	21:28:51	CREATE INDEX idx_attendee_id ON Attendee (id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.500 sec
84	21:28:54	CREATE UNIQUE INDEX idx_ticket_serial_number ON Ticket (serial_number)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.734 sec
85	21:28:57	SELECT Attendee.name, Attendee.email, Ticket.serial_number, Ticket.gate, Ticket.zone, Ticket.seat, Ticket.price...	227133 row(s) returned	0.000 sec / 0.453 sec

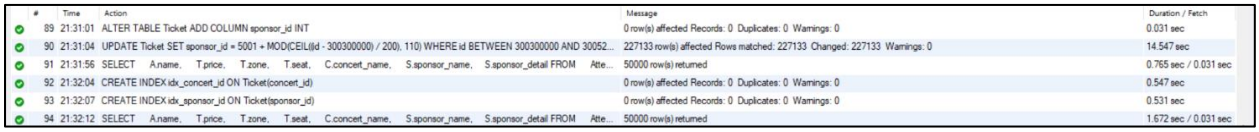
- Query retrieves information about tickets and the attendees who own them by joining the Ticket and Attendee tables on the attendee ID. This provides details about the ticket's serial number, gate, zone, seat, price, date, and times, along with the attendee's name and email.
- Create indexes on the Ticket and Attendee tables. Indexes can improve query performance by allowing the database to quickly locate and retrieve data based on the indexed columns.
    - CREATE INDEX idx_ticket_attendee_id ON Ticket (attendee_id); creates an index on the attendee_id column in the Ticket table, which should help speed up queries that filter or join on this column.
    - CREATE INDEX idx_attendee_id ON Attendee (id); creates an index on the id column in the Attendee table, which can improve performance when joining or filtering on this column.
    - CREATE UNIQUE INDEX idx_ticket_serial_number ON Ticket (serial_number); creates a unique index on the serial_number column in the Ticket table, ensuring that each serial number is unique and allowing for faster lookups based on this column.
- Timing information shows the query execution times before and after creating the indexes. The without index timing appears to be 0.000 seconds for the query and 0.531 seconds for possibly the join or data retrieval. After creating the indexes, the timing is 0.000 seconds for the query and 0.453 seconds for possibly the join or data retrieval, indicating a performance improvement.

(b) List the details of the concert ascending by ticket ID

#	Time	Action	Message	Duration / Fetch
1	21:21:19	SELECT Ticket.id AS TicketID, Ticket.serial_number AS SerialNumber, Ticket.gate AS Gate, Ticket.zone AS Zone, Tick...	227133 row(s) returned	0.000 sec / 0.328 sec
2	21:21:29	CREATE INDEX idx_ticket_concert_id ON Ticket(concert_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.531 sec
3	21:21:32	CREATE INDEX idx_ticket_id_concert ON Ticket(id, concert_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.531 sec
4	21:21:42	SELECT Ticket.id AS TicketID, Ticket.serial_number AS SerialNumber, Ticket.gate AS Gate, Ticket.zone AS Zone, Tick...	227133 row(s) returned	0.000 sec / 0.266 sec

- Query retrieves details of tickets and the associated concert information, ordered by the ticket ID in ascending order. The 'Ticket' and 'Concert' tables are joined using an inner join based on the 'concert_id' foreign key relationship. The selected columns include the ticket ID, serial number, gate, zone, seat, price, date, start time, end time, and the concert name. And ordering of the results by 'TicketID ASC' ensures that the output is sorted in ascending order based on the ticket ID column.
- Create indexes on the Ticket tables. Indexes can improve query performance by allowing the database to quickly locate and retrieve data based on the indexed columns.
    - CREATE INDEX idx_ticket_concert_id ON Ticket(concert_id); creates an index on the 'concert_id' column in the `Ticket` table. This index can help speed up the join operation between the 'Ticket' and 'Concert' tables by allowing faster lookups of ticket records based on the 'concert_id' value.
    - CREATE INDEX idx_ticket_id_concert ON Ticket(id, concert_id); creates a composite index on the 'id' and 'concert_id' columns in the 'Ticket' table. This index can optimize both the join operation and the ordering of results by the 'TicketID' column.
- Timing information shows the query execution times before and after creating the indexes. The without index timing is 0.000 seconds for the query and 0.328 seconds for some possibly the join, data retrieval, or sorting. After creating the indexes, the timing is 0.000 seconds for the query and 0.266 seconds for possibly the join or data retrieval, indicating a performance improvement. Indexing can significantly improve query performance, especially for queries involving joins, filters, or sorting on the indexed columns. In this case, the indexes likely helped optimize the join between the 'Ticket' and 'Concert' tables, as well as the ordering of results by the 'TicketID' column.

(e) List the details of the sponsor who supported the concert



#	Time	Action	Message	Duration / Fetch
89	21:31:01	ALTER TABLE Ticket ADD COLUMN sponsor_id INT	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.031 sec
90	21:31:04	UPDATE Ticket SET sponsor_id = 5001 + MOD(CEIL((id - 300300000) / 200), 110) WHERE id BETWEEN 300300000 AND 30052...	227133 row(s) affected Rows matched: 227133 Changed: 227133 Warnings: 0	14.547 sec
91	21:31:56	SELECT A.name, T.price, T.zone, T.seat, C.concert_name, S.sponsor_name, S.sponsor_detail FROM Atte...	50000 row(s) returned	0.765 sec / 0.031 sec
92	21:32:04	CREATE INDEX idx_concert_id ON Ticket(concert_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.547 sec
93	21:32:07	CREATE INDEX idx_sponsor_id ON Ticket(sponsor_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.531 sec
94	21:32:12	SELECT A.name, T.price, T.zone, T.seat, C.concert_name, S.sponsor_name, S.sponsor_detail FROM Atte...	50000 row(s) returned	1.672 sec / 0.031 sec

- Query retrieves details of sponsors who supported concerts, and displays information such as attendee name, concert name, ticket price, zone, seating, sponsoring company name, and sponsor details. It performs a multi-table join involving the 'Attendee', 'Ticket', 'Concert', and 'Sponsor' tables.
- ALTER TABLE Ticket ADD COLUMN sponsor_id INT; adds a new column named sponsor_id of integer data type to the Ticket table. This column will likely store a reference to a sponsor associated with each ticket.
- UPDATE Ticket SET sponsor_id = 5001 + MOD(CEIL((id - 300300000) / 200), 110) WHERE id BETWEEN 300300000 AND 300527132; updates the sponsor_id column for a specific range of ticket IDs (between 300300000 and 300527132) with a calculated value based on the ticket ID. The calculation involves subtracting 300300000 from the ticket ID, dividing by 200, taking the ceiling, and then finding the remainder when divided by 110, and finally adding 5001 to the result.
- Create indexes on the Ticket tables. Indexes can improve query performance by allowing the database to quickly locate and retrieve data based on the indexed columns.
    - CREATE INDEX idx_concert_id ON Ticket(concert_id); creates an index on the 'concert_id' column in the 'Ticket' table to optimize the join with the 'Concert' table.
    - CREATE INDEX idx_sponsor_id ON Ticket(sponsor_id); creates an index on the 'sponsor_id' column in the 'Ticket' table to optimize the join with the 'Sponsor' table.
- Timing information shows the query execution times before and after creating the indexes. The without index timing is 0.765 seconds for the query and 0.031 seconds for some possibly the join, data retrieval, or sorting. After creating the indexes, the timing is 1.672 seconds for the query and 0.031 seconds for possibly the join or data retrieval, in

this **case**, creating the indexes did not improve the query performance; instead, the query execution time increased from 0.765 seconds to 1.672 seconds. This could be due to **reasons** use random ordering (ORDER BY RAND()) may negate the potential benefits of indexing, as the database must sort the entire result set randomly.

An **example** can be provided to show this **fact**: when the ORDER BY RAND() function is removed, it becomes evident that utilizing an index is highly efficient and rapid. Nevertheless, I choose to employ a random sorting approach in order to enhance the diversity and efficiency of the data.

```
SELECT
    A.name,
    T.price,
    T.zone,
    T.seat,
    C.concert_name,
    S.sponsor_name,
    S.sponsor_detail
FROM
    Attendee A
JOIN
    Ticket T ON A.id = T.attendee_id
JOIN
    Concert C ON T.concert_id = C.id
JOIN
    Sponsor S ON T.sponsor_id = S.id
LIMIT 50000;
```