Universidad de Valparaíso
CHILE

PROYECTO INFORMATICO

APLICACIÓN WEB: VENTAS EN LÍNEA DE TICKETS PARA EVENTOS EN CHILE.

Integrantes:

Leonardo Arancibia

Manuel Carreño

Nicolás Veas

Académicos:

Manuel García

Eliana Providel

Asignatura:

ICI 324

Base de datos y programación web

Diciembre, 2020

INTRODUCCIÓN

Presentación

En el presente informe describe el proyecto de la asignatura Base de datos y programación web de la carrera de Ingeniería civil informática. El proyecto consiste en el desarrollo de un sistema con similares características a un carro de compras, donde el tema asignado es “Venta de tickets para eventos” con el objetivo de comprar tickets de manera online para los futuros eventos, además de desarrollar un back-end donde permita manejar los datos de los eventos y un front-end que les permita a los usuarios que interactúen con el sistema.

Descripción del proyecto

La problemática del proyecto es que las personas para adquirir tickets de un evento tienen que realizar de forma presencial, lo que ocupa tiempo, por lo tanto para priorizar otras actividades del mundo cotidiano una manera más eficiente y fácil es adquirir los tickets de manera online, donde optimizará el tiempo y no solo tendrán la facilidad para comprar tickets sino que podrá quedar registrada la compra y se podrá ahorrar comprobar el ticket de un evento mediante un papel y también podrán ver otros eventos de interés.

El equipo informático realizó la siguiente propuesta de solución la cual consiste en desarrollar una aplicación web con diversas tecnologías, donde en esta se puedan visualizar eventos de diversa índole tales como conciertos, deporte, culturales, familiares y especiales, que forman las categorías de eventos, estos contienen características como número identificación, nombre, fecha, lugar, hora y cantidad de tickets disponibles, asimismo los tickets derivan de cada evento por lo cual tienen cualidades como número de identificación y precio.

Se abordará sólo las actividades realizadas en Chile, donde se ofrece un servicio para todo tipo de personas en este caso compradores que es un tipo de usuario que desea adquirir tickets de los eventos disponibles en el sistema mediante un carro de compras y un administrador que operará con los eventos, usuarios y ventas.

REQUERIMIENTOS

REQUERIMIENTOS FUNCIONALES

- Los usuarios deben ingresar al sistema mediante un login.
- El sistema debe permitir cerrar sesión a los usuarios.
- Debe haber un registro de cada transacción del sistema.
- El sistema debe estar dividido en módulos para cada tipo de usuario.
- El sistema debe permitir al comprador visualizar eventos y ver información específica de cada uno.
- El sistema debe permitir al comprador adquirir tickets disponibles de los eventos.
- El sistema debe contener el carro de compras para cada usuario para que pueda visualizar la compra en detalle antes de finalizar.
- El sistema debe permitir solamente al usuario de tipo administrador operar (crear, actualizar, eliminar, leer) con eventos, usuarios y ventas.
- El sistema debe permitir al usuario de tipo comprador pueda adquirir ticket en el carro de compras de distintos eventos.
- Cada usuario de tipo comprador podrá ver los tickets obtenidos luego de finalizar cada compra.
- El comprador puede agregar, eliminar, consultar los tickets dentro del carro de compras.
- El sistema debe permitir que los usuarios de tipo comprador y administrador puedan visualizar a los eventos separados en categoría y mes.
- El sistema debe permitir a los usuarios visualizar la cantidad total de tickets disponible de cada evento y su respectivo precio.
- El administrador podrá visualizar un gráfico en el cual le indicará la categoría de eventos donde se venden más tickets.

REQUERIMIENTOS NO FUNCIONALES

- Los datos deben estar seguros para evitar un error de filtración o mal uso de estos.
- Bajos tiempos de respuesta (entre 1 a 5 segundos) entre el servidor y aplicación web.
- La plataforma debe soportar múltiples compras simultáneas de los usuarios.
- Aplicación web compatible con distintos navegadores de internet.
- El sistema debe considerar a los usuarios con poca experiencia un fácil uso.
- El sistema debe ser escalable para ser mantenible en el tiempo, independiente de los lenguajes de programación.

DIAGRAMA DE CASOS DE USO DE SOLUCIÓN GENERAL

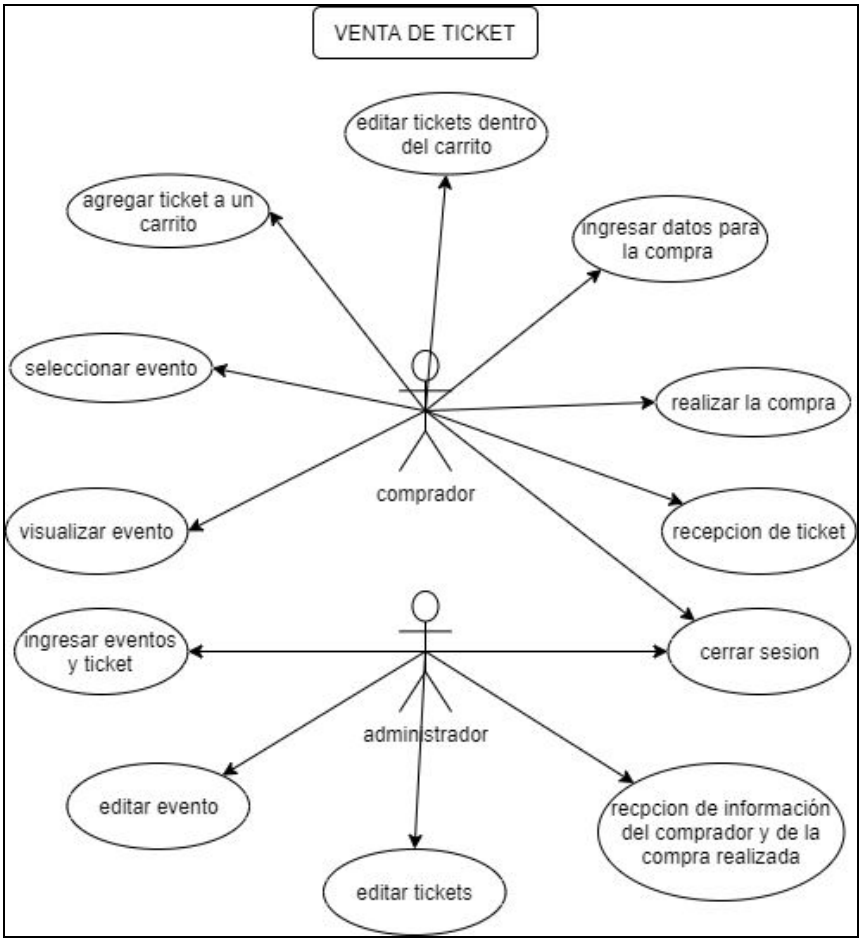

Figura 1: Diagrama de casos de usos según la perspectiva de cada tipo de usuario como administrador y comprador.

El diagrama de casos de usos (ver Figura 1) representa los servicios que tendrá disponible los dos tipos de usuarios que tendrá el sistema, en este caso se tiene al usuario de tipo comprador y administrador. Los usuarios tendrán distintas opciones según el tipo de usuario que identifica el sistema.

DIAGRAMA DE SECUENCIA DE LA SOLUCIÓN

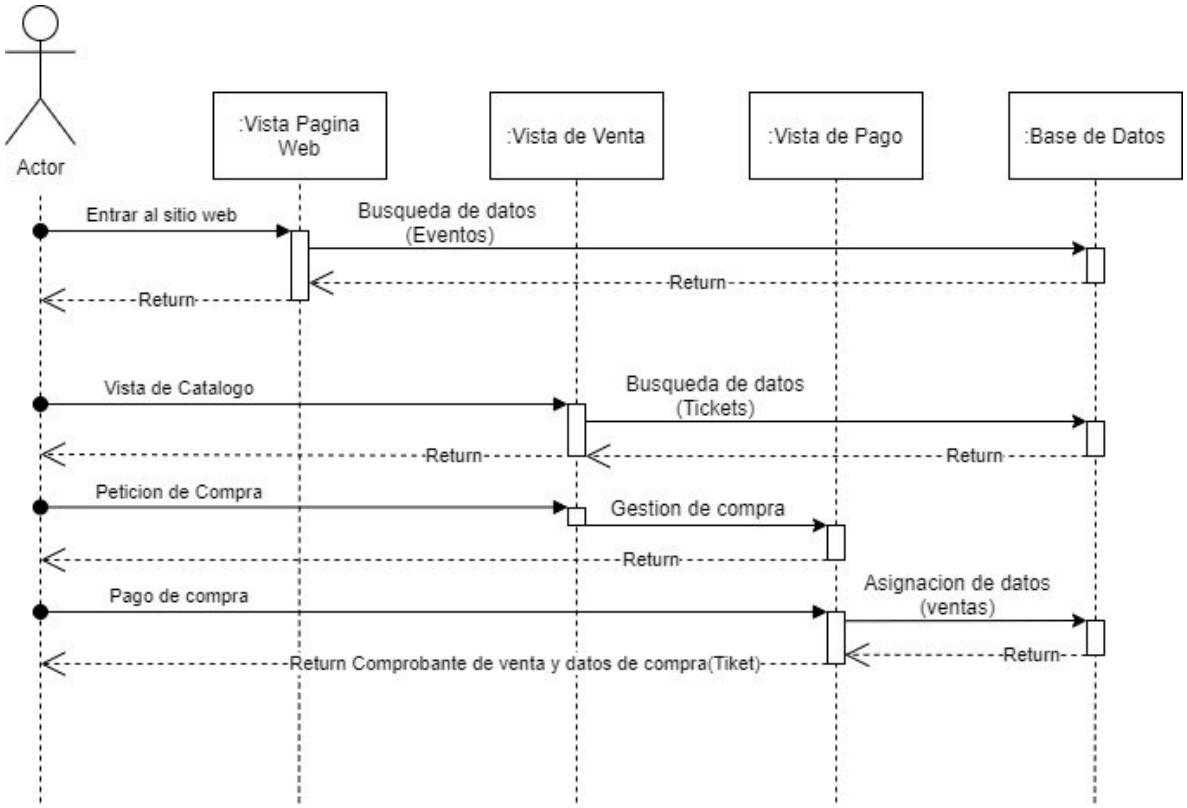

Figura 2: Diagrama de secuencia con la perspectiva del usuario principal que es el comprador.

El diagrama de secuencia (ver Figura 2) se puede visualizar el proceso que ocurre dentro del sistema al momento de que un comprador comienza el proceso de compra para un ticket.donde se comienza en el momento en que el comprador busca dentro del inventario disponible en la vista de eventos en la página,la cual obtiene la información de los eventos desde la base de datos,para luego mostrar toda la información de estos al cliente,este al seleccionar un ticket para comprar se le enviará la información de este ticket a la unidad de ventas para mostrar los datos de precompra para el cliente en una vista de pagos.Una vez el pago sea realizado toda la información del ticket queda asociada a la cuenta de usuario del cliente,datos los cuales serán guardados y enlazados en la base de datos.

MOCKUPS DE PANTALLAS DE SOLUCIÓN

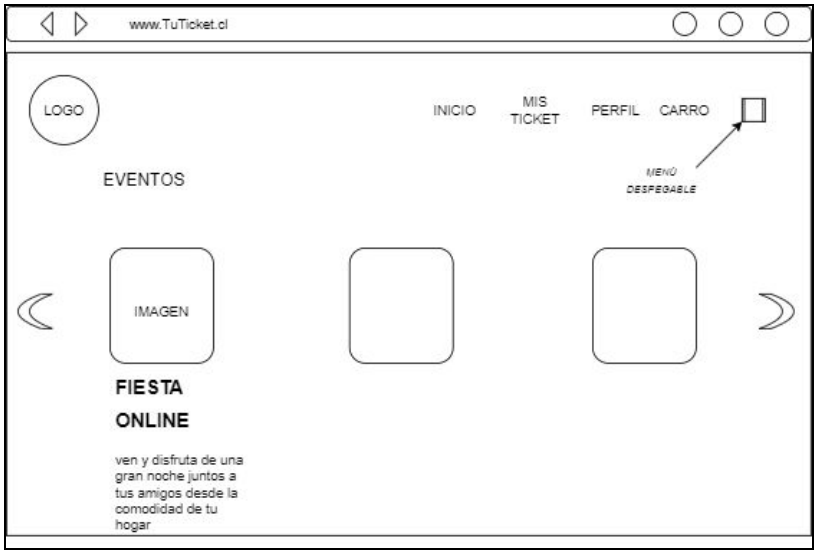

Figura 3: Mockup de la pantalla principal de la aplicación web.

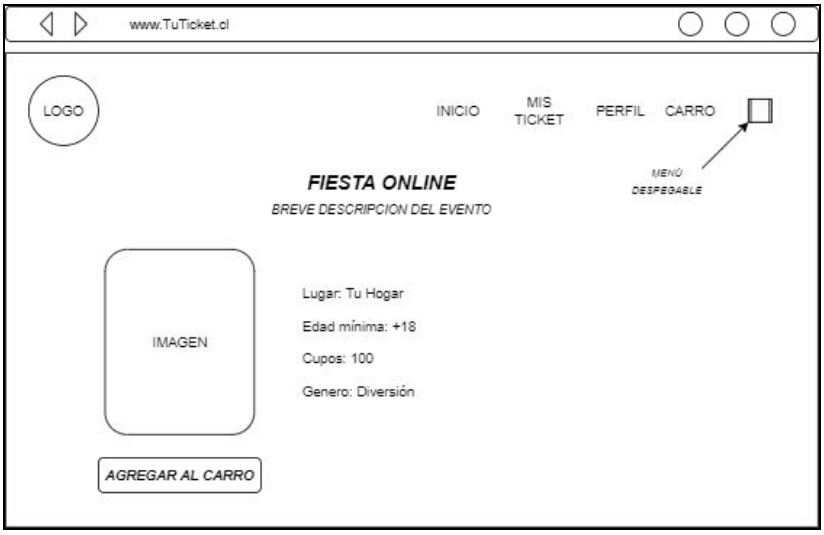

Figura 4: Mockup de la pantalla cuando seleccionas un evento.

Los Mockups (ver Figura 3 y 4) son una supuesta representación de como se verá tanto la pantalla principal de la aplicación web y las distintas opciones que observarán los usuarios en la vista de la aplicación web, en este caso al seleccionar un evento.

MODELO DE DATOS

MODELO E-R

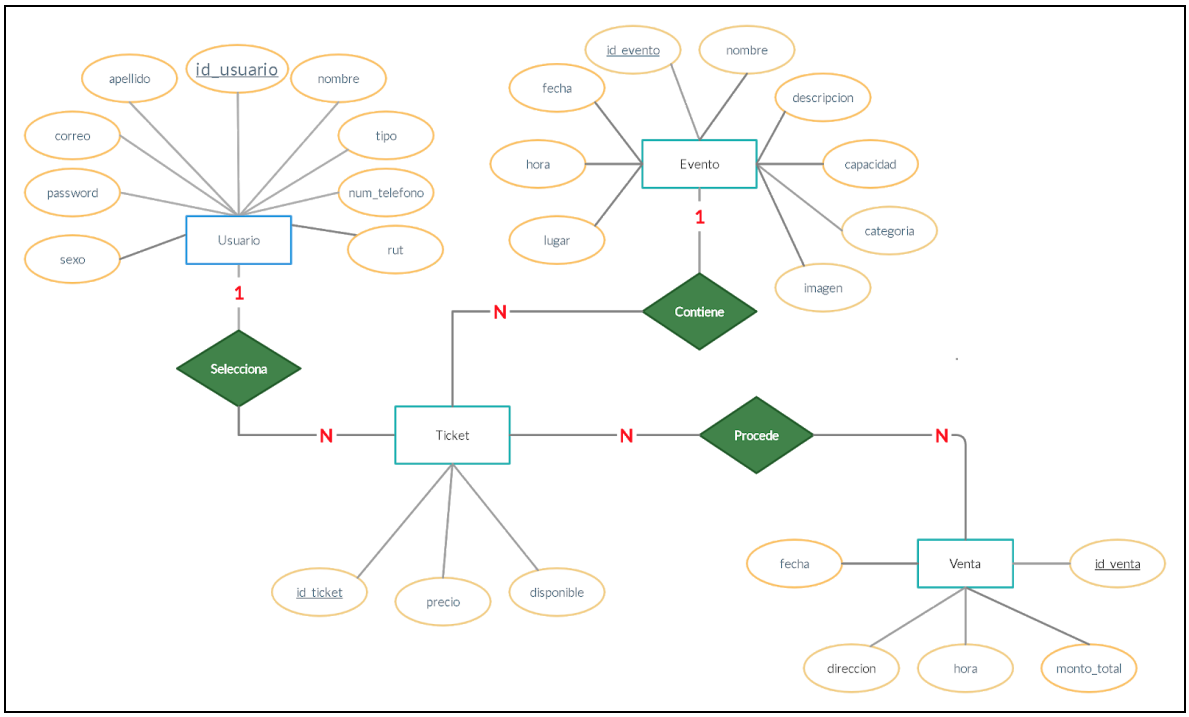

Figura 5: Modelo entidad relación de la base de datos.

En el modelo entidad relación (ver Figura 5), se ilustra las entidades que participaran en el proceso de negocio y cómo se relacionan entre sí, en este caso se tienen cuatro entidades que son: Usuario, Evento, Ticket y Venta, con sus respectivos atributos. Los rectángulos representan a los atributos, luego los círculos representan a los atributos, en donde estos mediante las líneas se relacionan con cada entidad y por último los rombos indican que relación tienen las entidades, además con la cardinalidad en cada lado del rombo.

MODELO RELACIONAL

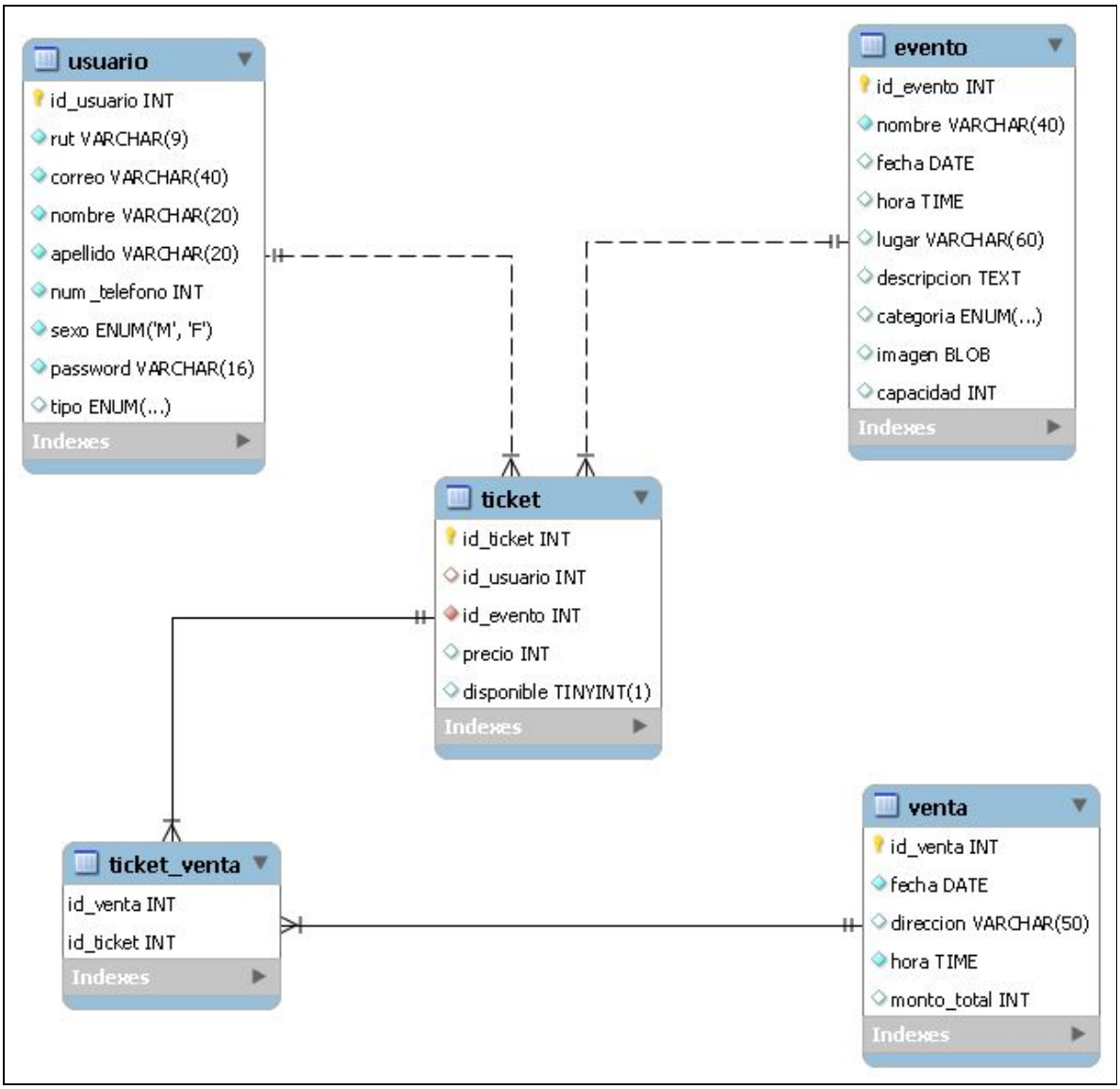

Figura 6: Modelo relacional derivado del Modelo entidad relación.

El modelo relacional (ver Figura 6) fue derivado o traspasado desde el Modelo entidad relación, en el cual se generaron las tablas y estas se relacionan mediante

claves primarias o foráneas, también se puede ver que los atributos tienen su tipo de dato. Por último este modelo de datos permite gestión de base de datos.

CONSULTAS CRUD

CONSULTAS EN LENGUAJE NATURAL

1. ADMINISTRADOR – COMPRADOR se les muestra los eventos disponibles en la página inicial (**SELECT**).
2. ADMINISTRADOR - COMPRADOR consultan eventos según categoría en un cuadro de opciones CATEGORÍAS (conciertos, culturales, especiales, familiares, deportes) **(SELECT)**.
3. ADMINISTRADOR consulta las ventas realizadas por los compradores (**SELECT**).
4. COMPRADOR consulta sus ventas y tickets realizados por el mismo usuario (**SELECT JOIN**).
5. ADMINISTRADOR añade eventos (**INSERT**).
6. ADMINISTRADOR consulta ticket de cada evento (**SELECT**).
7. ADMINISTRADOR elimina eventos (**DELETE**).
8. ADMINISTRADOR modifica registro deseado de la tabla venta en un cuadro de texto (**UPDATE**).
9. ADMINISTRADOR modifica registro deseado de la tabla evento en un cuadro de texto (**UPDATE**).
10. ADMINISTRADOR modifica registro deseado de la tabla usuario en un cuadro de texto (**UPDATE**).
11. COMPRADOR ejecuta compra y se realizan los siguientes procesos: insertar venta (**INSERT**), insertar venta-tickets (**INSERT**) y actualizar los datos del ticket id_usuario-disponible (**UPDATE**).
12. ADMINISTRADOR añade tickets a eventos (**INSERT**).
13. ADMINISTRADOR agrega campo a tabla ventas en un cuadro de texto (**ALTER**).
14. ADMINISTRADOR agrega campo a tabla ticket en un cuadro de texto (**ALTER**).
15. ADMINISTRADOR agrega campo a tabla usuario en un cuadro de texto (**ALTER**).

16. ADMINISTRADOR consulta los tickets comprados en una fecha específica (**SELECT JOIN)**.
17. COMPRADOR consulta tickets disponible de un evento específico (**SELECT**).
18. ADMINISTRADOR elimina ventas (**DELETE**).
19. ADMINISTRADOR consulta los usuarios de tipo comprador (**SELECT**).
20. Comprobar inicio de sesión (**SELECT**).

CONSULTAS SELECT EN ÁLGEBRA RELACIONAL

1. **ADMINISTRADOR – COMPRADOR se les muestra los eventos disponibles en la página inicial (SELECT).**

π id_evento, nombre, fecha, hora, lugar, descripcion, categoria, imagen, capacidad (Evento)

2. **ADMINISTRADOR - COMPRADOR consultan eventos según categoría en un cuadro de opciones CATEGORÍAS (conciertos, culturales, especiales, familiares, deportes) (SELECT).**

π id_evento, nombre, fecha, hora, lugar, descripcion, categoria, imagen, capacidad (σ categoria='conciertos' (Evento))

π id_evento, nombre, fecha, hora, lugar, descripcion, categoria, imagen, capacidad (σ categoria='familiares' (Evento))

π id_evento, nombre, fecha, hora, lugar, descripcion, categoria, imagen, capacidad (σ categoria='deportes' (Evento))

π id_evento, nombre, fecha, hora, lugar, descripcion, categoria, imagen, capacidad (σ categoria='especiales' (Evento))

π id_evento, nombre, fecha, hora, lugar, descripcion, categoria, imagen, capacidad (σ categoria='culturales' (Evento))

3. **ADMINISTRADOR consulta las ventas realizadas por los compradores (SELECT).**

π id_venta, fecha, direccion, hora, monto_total (Evento)

4. **COMPRADOR consulta sus ventas y tickets realizados por el mismo usuario (SELECT JOIN)**.

π ticket.id_usuario, ticket.id_ticket, ticket_venta.id_venta (σ ticket.id_usuario=1 ((ticket) ⋈ ticket.id_ticket = ticket_venta.id_ticket (ticket_venta)))

5. **ADMINISTRADOR consulta ticket de cada evento (SELECT).**

π id_ticker, id_usuario, id_evento, precio, disponible (σ id_evento=1 (Ticket))

6. **ADMINISTRADOR consulta los tickets comprados en una fecha específica (SELECT JOIN).**

π ticket_venta.id_ticket, venta.fecha (σ venta.fecha="2020-08-16" ((ticket_venta) ⋈ ticket_venta.id_venta = vent.id_venta (venta)))

7. **COMPRADOR consulta tickets disponible de un evento específico (SELECT).**

π id_ticker, id_usuario, id_evento, precio, disponible (σ id_evento=1 ∧ disponible=1 (Ticket))

8. **ADMINISTRADOR consulta los usuarios de tipo comprador (SELECT).**

π id_usuario, rut, correo, nombre, apellido, num_telefono, sexo, password, tipo (σ tipo="comprador"(usuario))

ETAPA II

CONSULTAS CRUD

CONSULTAS EN LENGUAJE NATURAL Y SQL

1) **ADMINISTRADOR consulta los usuarios que han comprado ticket (Subconsulta).**

```
SELECT DISTINCT u.id_usuario, u.nombre, u.apellido, u.sexo
FROM usuario u JOIN
  (SELECT id_usuario, id_ticket
  FROM ticket )AS t
ON u.id_usuario = t.id_usuario;
```

2) **ADMINISTRADOR consulta los usuarios hombres que han comprado ticket (Subconsulta).**

```
SELECT DISTINCT u.id_usuario
FROM usuario u JOIN
  (SELECT id_usuario, id_ticket
  FROM ticket )AS t
ON u.id_usuario = t.id_usuario
WHERE u.sexo = "M";
```

3) **ADMINISTRADOR puede visualizar las ventas y tickets con su información general (Subconsulta).**

```
SELECT vt.*, p.id_usuario, p.id_ticket, p.precio, id_evento
```

```
FROM venta as vt INNER JOIN
(SELECT t.id_usuario, t.id_ticket, t.precio, t.id_evento, tv.id_venta
FROM ticket as t INNER JOIN ticket_venta tv
ON t.id_ticket = tv.id_ticket) p
ON vt.id_venta = p.id_venta;
```

4) **ADMINISTRADOR consulta la cantidad de Tickets sin vender de cada eventos (Operadores).**

```
SELECT count(*) as "Numero de Tickets sin vender", e.nombre
FROM ticket as t INNER JOIN evento e
ON t.id_evento = e.id_evento
WHERE t.id_usuario is  null
group by nombre;
```

5) **ADMINISTRADOR consulta la cantidad de Tickets vendidos de cada evento (Operadores).**

```
SELECT count(*) as "Numero de Tickets vendidos", e.nombre
FROM ticket as t INNER JOIN evento e
ON t.id_evento = e.id_evento
WHERE t.id_usuario is not null
group by nombre;
```

6) **ADMINISTRADOR consulta la cantidad de dinero gastado en tickets de un usuario en específico (Funciones agregadas).**

```
SELECT SUM(precio), t.id_usuario
FROM ticket as t INNER JOIN ticket_venta as tv
```

```
ON t.id_ticket = tv.id_ticket
WHERE t.id_usuario = 1;
```

7) **ADMINISTRADOR consulta la cantidad de tickets disponibles según la fecha de realización de los eventos (Group by).**

```
SELECT fecha, count(*) as tickets
FROM ticket as t INNER JOIN evento e
ON t.id_evento=e.id_evento
group by fecha;
```

8) **ADMINISTRADOR consulta la cantidad de tickets disponibles según el nombre de cada evento (Group by).**

```
SELECT nombre, count(*) as tickets
FROM ticket as t INNER JOIN evento e
ON t.id_evento=e.id_evento
group by nombre;
```

9) **ADMINISTRADOR consulta la cantidad de tickets disponibles según la categoría de cada evento (Group by).**

```
SELECT categoria, count(*) as tickets
FROM ticket as t INNER JOIN evento e
ON t.id_evento=e.id_evento
group by categoria;
```

10) **ADMINISTRADOR consulta la cantidad de tablas dentro de la base de datos (INFORMATION_ESCHEMA)**

```
SELECT count(*) cantidad_tablas
FROM information_schema.tables
WHERE table_schema = 'proyecto_ticket'
ORDER BY table_name DESC;
```

11) **ADMINISTRADOR pide información de una columna en especifico de una tabla (INFORMATION_ESCHEMA)**

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'evento'
AND table_schema = 'proyecto_ticket'
AND column_name LIKE 'id_evento';
```

CONSULTAS EN ÁLGEBRA RELACIONAL

1. **ADMINISTRADOR consulta los usuarios que han comprado ticket (Subconsulta).**

π distinct usuario.id_usuario, usuario.nombre, usuario.apellido, usuario.sexo ((usuario) ⋈ usuario.id_usuario = c.id_usuario **ρ**c(**π** id_usuario, id_ticket (ticket)))

2. **ADMINISTRADOR consulta los usuarios hombres que han comprado ticket (Subconsulta).**

π distinct usuario.id_usuario (**σ** usuario.sexo = 'M' ((usuario) ⋈ usuario.id_usuario = c.id_usuario **ρ**c(**π** id_usuario, id_ticket (ticket))))

3. **ADMINISTRADOR consulta la cantidad de dinero gastado en tickets de un usuario en específico (Funciones agregadas).**

π SUM(precio), ticket.id_usuario (**σ** t.id_usuario = '1' ((ticket) ⋈ ticket.id_ticket = ticket_venta.id_ticket (ticket_venta)))

4. **ADMINISTRADOR puede visualizar las ventas y tickets con su información general (Subconsulta).**

π venta.id_venta, venta.fecha, venta.direccion, venta.hora, venta.monto_total ((venta) ⋈ venta.id_venta=c.id_venta **ρ**c(**π** ticket.id_usuario, ticket.id_ticket, ticket.precio, ticket.id_evento, ticket_venta.id_venta ((ticket) ⋈ ticket.id_ticket = ticket_venta.id_ticket (ticket_venta))))

5. **ADMINISTRADOR consulta la cantidad de tickets disponibles según la categoría de cada evento (Group by).**

π evento.categoria, count(*) (**γ** evento.categoria (((ticket) ⋈ ticket.id_evento = evento.id_evento (evento))))

- **γ** = group by

6. **ADMINISTRADOR consulta la cantidad de tickets disponibles según el nombre de cada evento (Group by).**

π evento.nombre, count(*) (evento.nombre (((ticket) ⋈ ticket.id_evento = evento.id_evento (evento))))

- **γ** = group by

ETAPA III

CONSULTAS CRUD

1) **ADMINISTRADOR consulta la cantidad de dinero que se ha generado hasta la fecha actual con los tickets vendidos, según un evento específico (función agregada).**

```
SELECT SUM(precio), t.id_evento
FROM ticket as t INNER JOIN ticket_venta as tv
ON t.id_ticket = tv.id_ticket
WHERE t.id_evento = 4;
```

2) **ADMINISTRADOR consulta la cantidad de dinero que se ha generado hasta la fecha actual con los tickets vendidos, según una categoría específica (función agregada).**

```
SELECT SUM(precio), e.categoria
FROM ticket as t INNER JOIN evento as e
ON t.id_evento = e.id_evento
WHERE e.categoria ='conciertos';
```

3) **ADMINISTRADOR consulta la cantidad de dinero gastado en tickets de todos los usuarios (función agregada)(Group by).**

```
SELECT SUM(precio), t.id_usuario
FROM ticket as t INNER JOIN ticket_venta as tv
ON t.id_ticket = tv.id_ticket
WHERE t.id_usuario is not null
group by id_usuario;
```

4) **El USUARIO consulta la cantidad de tickets que aún no se han vendido, según el nombre(s) específico(s) de cada evento (Operadores) (Group by).**

```
SELECT nombre, count(*) as tickets
FROM ticket as t INNER JOIN evento e
ON t.id_evento=e.id_evento
WHERE (nombre in ('bad bunny','Lollapalooza'))
group by nombre;
```

5) **El USUARIO consulta la cantidad de tickets que aún no se han vendido, según la(s) hora(s) en que se realicen los eventos. (Operadores) (Group by).**

```
SELECT nombre, count(*) as tickets
FROM ticket as t INNER JOIN evento e
ON t.id_evento=e.id_evento
WHERE (hora in ('14:00:00','10:00:00'))
group by nombre;
```

6) **El USUARIO consulta la cantidad de tickets que aún no se han vendido, según una fecha específica de realización de los eventos (Group by)(Operadores).**

```
SELECT fecha, count(*) as tickets
FROM ticket as t INNER JOIN evento e
ON t.id_evento=e.id_evento
WHERE (fecha in ('2020-09-18'))
group by fecha;
```

7) **ADMINISTRADOR consulta los usuarios que han comprado ticket (Subconsulta)(Group by).**

```
SELECT DISTINCT u.id_usuario
FROM usuario u JOIN
(SELECT id_usuario, id_ticket
FROM ticket )AS t
ON u.id_usuario = t.id_usuario
WHERE u.id_usuario is not null
 group by id_usuario;
```

8) **ADMINISTRADOR consulta el id de los tickets que fueron comprados, correspondientes a una venta específica (Subconsulta).**

```
SELECT DISTINCT  t.id_ticket
FROM ticket t JOIN
(SELECT id_venta, id_ticket
FROM ticket_venta )AS ti
ON ti.id_ticket = t.id_ticket
WHERE ti.id_venta =6;
```

9) **ADMINISTRADOR consulta la cantidad de ticket que serán entregados por evento, según el lugar donde este se realice.(Subconsulta).**

```
SELECT lugar, count(*) as tickets
FROM ticket t JOIN
(SELECT lugar, id_evento
FROM evento )AS e
ON t.id_evento = e.id_evento
WHERE e.lugar ='Sport club';
```

10) **ADMINISTRADOR consulta los tickets que se vendieron, según una fecha de venta.(Subconsulta).**

```
SELECT DISTINCT ti.id_ticket
FROM ticket_venta ti JOIN
(SELECT fecha, id_venta
FROM venta )AS v
ON ti.id_venta = v.id_venta
WHERE v.fecha ='2020-08-16';
```

11) **ADMINISTRADOR consulta el número de conexiones de cada tipo de usuario(INFORMATION_ESCHEMA).**

```
SELECT USER, COUNT(*)
FROM information_schema.processlist
GROUP BY USER
```

12) **ADMINISTRADOR consulta todas las columnas de una tabla específica con sus correspondientes tipos de datos (INFORMATION_ESCHEMA).**

```
SELECT * FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME LIKE 'evento';
```

13) **ADMINISTRADOR consulta la cantidad de entidades dentro de la base de datos. (INFORMATION_ESCHEMA).**

```
SELECT count(*) cantidad_tablas
FROM information_schema.tables
WHERE table_schema = 'proyecto_ticket';
```

CONSULTAS EN ÁLGEBRA RELACIONAL

1. **ADMINISTRADOR consulta la cantidad de dinero que se ha generado hasta la fecha actual con los tickets vendidos, según un evento específico (función agregada).**

   π SUM(precio), ticket.id_evento(σ ticket.id_evento = ? ((ticket) ⋈ ticket.id_ticket = ticket_venta.id_ticket (ticket_venta)))

2. **ADMINISTRADOR consulta la cantidad de dinero que se ha generado hasta la fecha actual con los tickets vendidos, según una categoría específica (función agregada).**

   π SUM(precio), evento.categoria (σ evento.categoria = ? ((ticket) ⋈ ticket.id_evento = evento.id_evento (evento)))

3. **ADMINISTRADOR consulta la cantidad de dinero gastado en tickets de todos los usuarios (función agregada)(Group by)**

   SUM(precio), ticket.id_usuario (γ id_usaurio (σ ticket.id_usuario is not null ((ticket) ⋈ ticket.id_ticket = ticket_venta.id_ticket (ticket_venta))))

   - γ = group by

4. **El USUARIO consulta la cantidad de tickets que aún no se han vendido, según el nombre(s) específico(s) de cada evento (Operadores) (Group by).**

   π SUM(*), evento.nombre (γ evento.nombre (σ nombre in ('?') ((ticket) ⋈ ticket.id_evento = evento.id_evento (evento))))

- **γ** = group by

5. **El USUARIO consulta la cantidad de tickets que aún no se han vendido, según la(s) hora(s) en que se realicen los eventos. (Operadores) (Group by).**

   π SUM(*), evento.nombre (**γ** evento.nombre (**σ** hora in ('?') ((ticket) ⋈ ticket.id_evento = evento.id_evento (evento))))

- **γ** = group by

6. **El USUARIO consulta la cantidad de tickets que aún no se han vendido, según una fecha específica de realización de los eventos (Group by)(Operadores).**

   π fecha, count(*) (**γ** evento.fecha (**σ** fecha in ('?') ((ticket) ⋈ ticket.id_evento = evento.id_evento (evento))))

- **γ** = group by

7. **ADMINISTRADOR consulta los usuarios que han comprado ticket (Subconsulta)(Group by).**
   π DISTINCT usuario.id_usuario (**γ** id_usuario (**σ** usuario.id_usuario is not null ((**π** id_usuario, id_ticket (ticket)) ⋈ ticket.id_usuario = usuario.id_usuario (usuario))))

- **γ** = group by

8. **ADMINISTRADOR consulta el id de los tickets que fueron comprados, correspondientes a una venta específica (Subconsulta).**
   π DISTINCT ticket.id_ticket (**σ** ticket_venta.id_venta = ? ((**π** id_venta, id_ticket (ticket_venta)) ⋈ ticket_venta.id_ticket = ticket.id_ticket (ticket)))

9. **ADMINISTRADOR consulta la cantidad de ticket que serán entregados por evento, según el lugar donde este se realice.(Subconsulta)**
   π evento.lugar, count(*) (**σ** evento.lugar = ? ((**π** lugar, id_evento (evento)) ⋈ evento.id_evento = ticket.id_evento (ticket)))

10. **ADMINISTRADOR consulta los tickets que se vendieron, según una fecha de venta.(Subconsulta).**

    π DISTINC ticket_venta.id_ticket (**σ** venta.fecha = ? ((**π** fecha, id_venta (venta)) ⋈ venta.id_venta = ticket_venta.id_venta (ticket_venta)))