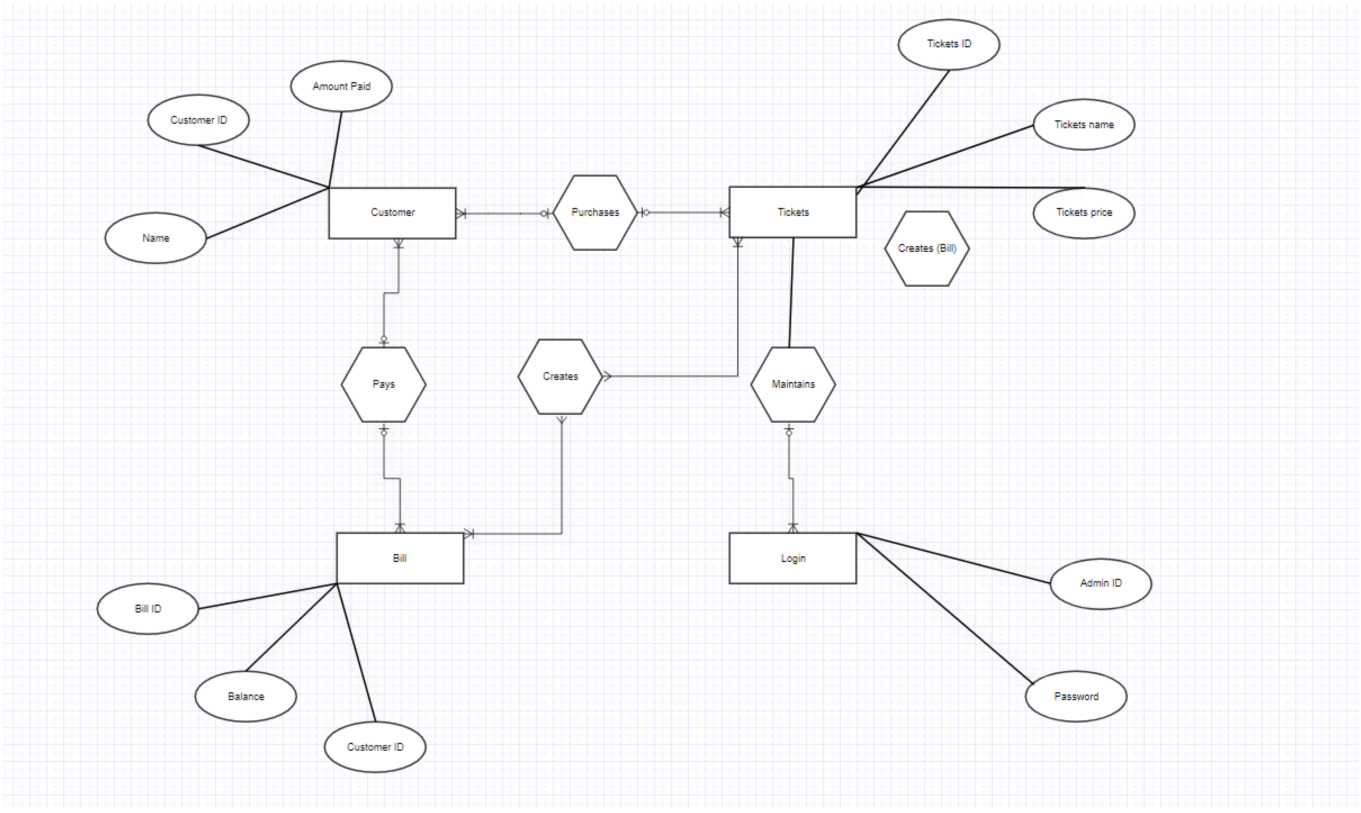
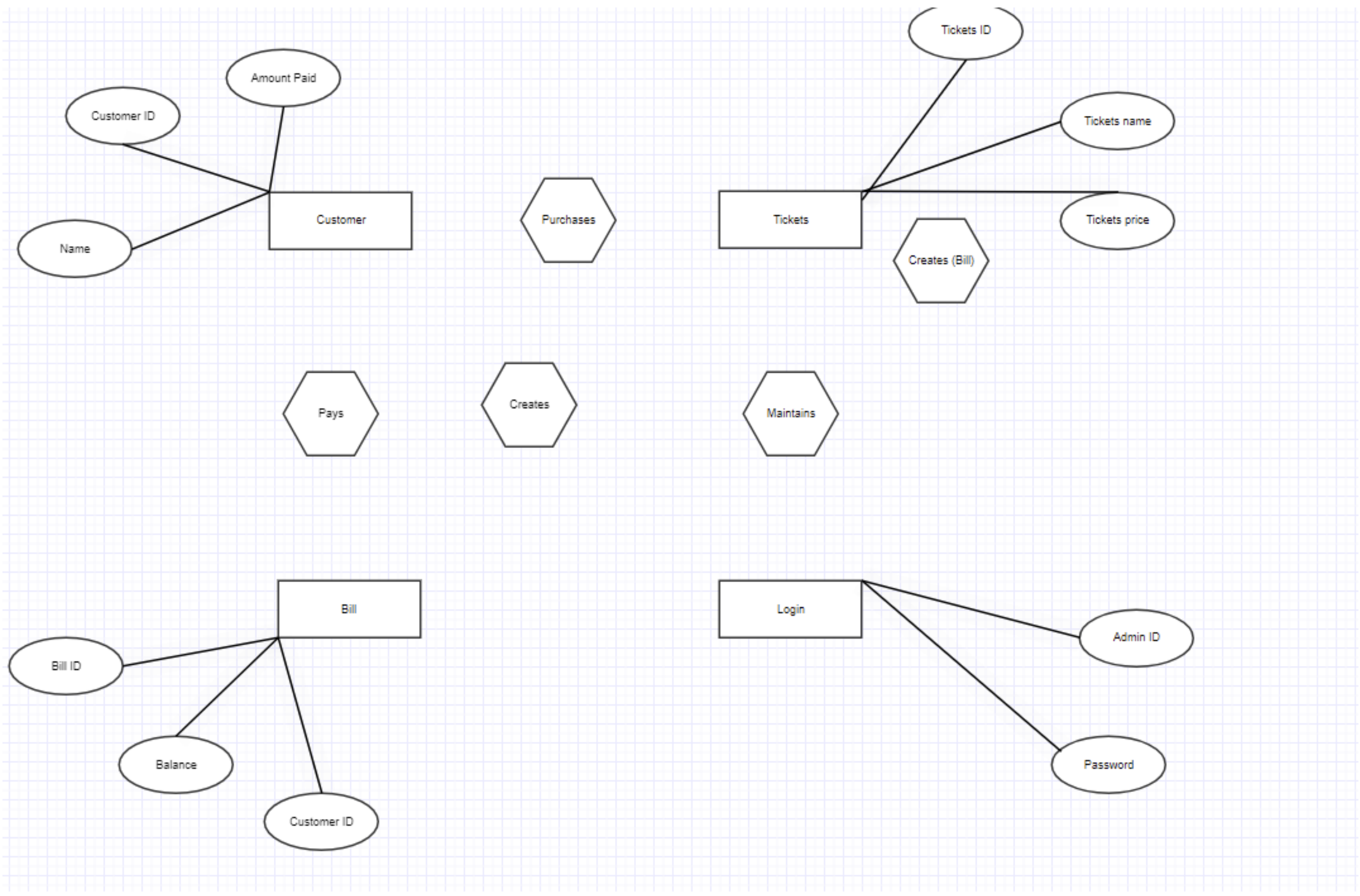
Web

Report

By Ramy Naiem

Logical Design





Physical Design and DDL Statments

SQL Statements

DDL
Create
Alter

DDL

- **Create:**

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `searchmovie` (IN `movie` VARCHAR(1000))  NO SQL
SELECT * from movies where Movie_Name like CONCAT('%', movie, '%')$$

–
CREATE TABLE `admin` (
  `username` varchar(50) NOT NULL,
  `password` varchar(15) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

–

CREATE TABLE `bookings` (
  `username` varchar(200) NOT NULL,
  `movie` varchar(200) NOT NULL,
  `theatre` varchar(200) NOT NULL,
  `seats` varchar(2000) NOT NULL,
  `date` varchar(100) NOT NULL,
  `movie_time` varchar(100) NOT NULL,
  `location` varchar(200) NOT NULL,
  `amount` int(200) NOT NULL,
  `id` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

–

CREATE TABLE `movies` (
  `Movie_Name` varchar(50) NOT NULL,
  `Actor` varchar(25) NOT NULL,
  `Actress` varchar(25) NOT NULL,
  `Release_date` varchar(50) NOT NULL,
  `Director` varchar(50) NOT NULL,
  `Movie_id` int(100) NOT NULL,
  `poster` varchar(300) NOT NULL,
  `RunTime` varchar(100) NOT NULL,
  `type` varchar(100) NOT NULL,
  `ActorImg` varchar(300) NOT NULL,
  `ActressImg` varchar(400) NOT NULL,
  `DirectorImg` varchar(300) NOT NULL,
  `Description` varchar(4000) DEFAULT NULL,
  `trailer` varchar(400) NOT NULL,
  `wiki` varchar(400) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

–
```

```
CREATE TABLE `theatres` (
  `Theatre_id` int(200) NOT NULL,
  `Theatre_Name` varchar(200) NOT NULL,
  `Location` varchar(300) NOT NULL,
  `Movie_Name` varchar(200) NOT NULL,
  `time1` varchar(200) NOT NULL,
  `time2` varchar(200) NOT NULL,
  `time3` varchar(200) NOT NULL,
  `time4` varchar(200) NOT NULL,
  `time5` varchar(200) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
–

CREATE TABLE `timings` (
  `id` int(200) NOT NULL,
  `showtime` varchar(200) NOT NULL,
  `Theatre_Name` varchar(200) NOT NULL,
  `ticket_rate_Gold` int(50) NOT NULL,
  `ticket_rate_Silver` int(50) NOT NULL,
  `seats` int(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
–

CREATE TABLE `users` (
  `username` varchar(100) NOT NULL,
  `email` varchar(100) NOT NULL,
  `password` varchar(100) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


–

CREATE DEFINER=`root`@`localhost` EVENT `update_seat` ON SCHEDULE EVERY 24 HOUR STARTS '2018-10-13 00:00:00' ENDS '2018-11-30 00:00:00' ON COMPLETION NOT PRESERVE ENABLE DO Update timings set seats=120 where seats<120$$
```

- Alter:

```
ALTER TABLE `admin`
  ADD PRIMARY KEY (`username`);
–

ALTER TABLE `bookings`
  ADD PRIMARY KEY (`id`);
–

ALTER TABLE `movies`
  ADD PRIMARY KEY (`Movie_id`);
–
```

```
ALTER TABLE `theatres`
  ADD PRIMARY KEY (`Theatre_id`);
–

ALTER TABLE `timings`
  ADD PRIMARY KEY (`id`);
–

ALTER TABLE `users`
  ADD PRIMARY KEY (`username`);
–

ALTER TABLE `bookings`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=18;
–
ALTER TABLE `movies`
  MODIFY `Movie_id` int(100) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=54;
–
ALTER TABLE `theatres`
  MODIFY `Theatre_id` int(200) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=15;
–
ALTER TABLE `timings`
  MODIFY `id` int(200) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=22;
```

DML
Insert
Update

- Insert:

```
INSERT INTO `admin` (`username`, `password`) VALUES
–
INSERT INTO `bookings` (`username`, `movie`, `theatre`, `seats`, `date`, `movie_time`, `location`, `amount`,
`id`) VALUES
('ramy', 'Gold', 'P90', 'A3 A4', '5-12-2022', '13:00', 'Galaxy Cinema', 400, 10),
('ramy', 'Gold', 'P90', 'B3 B4', '5-12-2022', '13:00', 'Galaxy Cinema', 400, 13),
('ramy', 'Gold', 'P90', 'A5 B5', '5-12-2022', '13:00', 'Galaxy Cinema', 400, 15),
('ramy', 'Gold', 'P90', 'E2 J2', '5-12-2022', '13:00', 'Galaxy Cinema', 380, 16),
('ramy', 'Gold', 'P90', 'A1 B1 C1 C2', '5-12-2022', '13:00', 'Galaxy Cinema', 800, 17);
–
```

```
INSERT INTO `timings` (`id`, `showtime`, `Theatre_Name`, `ticket_rate_Gold`, `ticket_rate_Silver`, `seats`)
VALUES
(5, '13:00', 'P90', 200, 180, 106),
(6, '10:00', 'P90', 150, 120, 120),
(7, '09:00', 'Cairo-Festival', 150, 100, 120),
(8, '13:00', 'Cairo-Festival', 200, 180, 120),
(9, '17:00', 'Cairo-Festival', 200, 150, 120),
(10, '20:00', 'Cairo-Festival', 250, 200, 120),
(11, '10:00', 'MALL OF EGYPT', 150, 100, 120),
(12, '22:00', 'MALL OF EGYPT', 250, 220, 120),
(13, '22:00', 'Cairo-Festival', 200, 180, 120),
(14, '13:00', 'MALL OF EGYPT', 200, 180, 120),
(15, '17:00', 'MALL OF EGYPT', 250, 220, 120),
(16, '20:00', 'MALL OF EGYPT', 300, 280, 120),
(17, '20:00', 'P90', 250, 220, 120),
(18, '17:00', 'P90', 250, 200, 120),
(19, '23:00', 'P90', 250, 210, 120),
(20, '23:00', 'MALL OF EGYPT', 250, 220, 120),
(21, '16:00', 'Cairo-Festival', 250, 180, 120);
_


INSERT INTO `users` (`username`, `email`, `password`) VALUES
('Ramy', 'test@gmail.com', '$2y$10$PO7NHZKsZE1f69lMLGtLxuhgCy0.929yS8Mmc6MpKUuD0zbEXIzDi'),
```

"Code is like humor. When you have to explain it, it's bad"

SQL Injection

Types of SQL Attacks

Union-based SQL Injection — This is the most common sort of SQL injection, and it makes use of the UNION command. The UNION statement combines two select statements to get information from a database.

SQL Injection with Errors – this approach can only be used with MS-SQL servers. A malicious user causes an application to display an error in this attack. Typically, you ask a database a query, and it responds with an error message that includes the data they requested.

No error warnings are received from the database in this attack, and we extract the data by making queries to the database. The two types of blind SQL injections are boolean-based SQL injections and time-based SQL injections.

Impact of SQL injections

- Steal credentials—Through SQLi, attackers may gain credentials and then impersonate users in order to take advantage of their privileges.
- Attackers may acquire access to sensitive data on database servers via accessing databases.
- Attackers might change or add new data to the database that has been accessed.
- Data may be deleted or whole tables can be dropped by attackers.
- Lateral movement—attackers having operating system rights may get access to database servers and then utilise those privileges to gain access to other critical systems.

SQL injections

SQL injection based on user input - online applications take user input through forms, which then send the information to the database for processing. An attacker may insert malicious SQL statements if the web application accepts these inputs without sanitising them.

SQL injection through cookies - altering cookies to "poison" database requests is another method of SQL injection. Cookies are often loaded and used by web applications as part of database operations. Cookies might be modified by a malicious user or malware installed on a user's device to inject SQL in an unexpected fashion.

SQL injection using HTTP headers — SQL injection may also be performed using server variables such as HTTP headers. Fake HTTP headers containing arbitrary SQL may inject code into a database if a web application allows input from HTTP headers.

Second-order SQL injections are the most complicated SQL injection attacks because they may go undetected for a long time. A second-order SQL injection attack sends out poisoned data that may seem harmless in one context but is harmful in another. Even if all application inputs are sanitised, developers may still be exposed to this sort of attack.

Real Life examples for injecting SQL

GhostShell attack—hackers from APT group Team GhostShell targeted 53 universities using SQL injection, stole and published 36,000 personal records belonging to students, faculty, and staff.

Turkish government—another APT group, RedHack collective, used SQL injection to breach the Turkish government website and erase debt to government agencies.

7-Eleven breach—a team of attackers used SQL injection to penetrate corporate systems at several companies, primarily the 7-Eleven retail chain, stealing 130 million credit card numbers.

HBGary breach—hackers related to the Anonymous activist group used SQL Injection to take down the IT security company's website. The attack was a response to HBGary CEO publicizing that he had names of Anonymous organization members.

How to prevent SQL injection

1. Validation of input:

   The validation procedure checks to see whether the kind of input given by the user is permitted. Input validation ensures that the type, length, and format are correct. Only those values that pass validation are handled. It assists in preventing any instructions from being injected into the input string.

2. Queries with parameters

   Parameterized queries are a way of pre-compiling a SQL statement such that the parameters may be supplied after the statement is run. This approach allows the database to detect and separate the code from the input data.

This coding approach helps minimise a SQL injection attack since the user input is automatically quoted and the given input will not cause the intent to change.

The MySQLi extension allows parameterized queries, however PHP 5.1 introduced a superior technique to interacting with databases: PHP Data Objects (PDO). PDO uses methods to make parameterized queries easier to use.

3. Stored Procedures:

To design an execution plan for stored procedures, the developer must arrange one or more SQL statements into a logical unit. Statements may be automatically parameterized in subsequent executions. Simply explained, it's a sort of code that can be saved and reused again.

Instead of writing the question again and over, you may just call the stored procedure anytime you need it.

4. Escaping

Always utilize the character-escaping mechanisms offered by each database management system for user-supplied input (DBMS). This is done to ensure that the DBMS does not mix it up with the developer's SQL query.

Use PHP's MySQL real escape string() to eliminate characters that can result in an unexpected SQL statement.

5. Avoiding administrative Privilegs

Don't use a root account to connect your application to the database. Because the attackers might get access to the whole system, this should only be done if necessary. Even non-administrative accounts servers may put an application at danger, especially if the database server is shared by many apps and databases.

As a result, to protect the application against SQL injection, it's best to apply least privilege on the database. Ascertain that each program has its own database credentials, with the minimal permissions required for the application.

Sources:

- https://brightsec.com/blog/sql-injection-attack/
- https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/
- https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- https://www.lucidchart.com/pages/er-diagrams