IS497 Final Report
Hotel Room Booking Database Management

Group 6 Member: Alisha Rawat, Hsin-Tzu Lin, Sutthana Koo-Anupong

---

1. Project Overview

This project presents the simulation of a hotel room reservation database system using MySQL, deployed on Microsoft Azure. To meet the given project requirements, we implemented table partitioning to optimize query performance. Additionally, we created a custom script to simulate workload spikes for stress testing and to explore the concept of auto-scaling database resources.

The project also demonstrates fundamental database administration skills, including system installation and configuration, data loading and querying, role-based access control, backup and restoration processes, and basic cost estimation for cloud-based deployments.

2. Dataset and Database Setup

We used a hotel booking dataset with an original size of approximately 3.24 MB with 36,275 unique records[1]. This dataset was later expanded to support partitioning and scaling experiments. The dataset consists of the attributes as outlined below.

- Booking_ID: unique identifier of each booking
- no_of_adults: Number of adults
- no_of_children: Number of Children
- no_of_weekend_nights: Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel
- no_of_week_nights: Number of week nights (Monday to Friday) the guest stayed or booked to stay at the hotel
- type_of_meal_plan: Type of meal plan booked by the customer
- required_car_parking_space: Does the customer require a car parking space? (0 - No, 1- Yes)
- room_type_reserved: Type of room reserved by the customer. The values are ciphered (encoded) by INN Hotels.
- lead_time: Number of days between the date of booking and the arrival date
- arrival_year: Year of arrival date
- arrival_month: Month of arrival date
- arrival_date: Date of the month
- market_segment_type: Market segment designation.
- repeated_guest: Is the customer a repeated guest? (0 - No, 1- Yes)
- no_of_previous_cancellations: Number of previous bookings that were canceled by the customer prior to the current booking

- no_of_previous_bookings_not_canceled: Number of previous bookings not canceled by the customer prior to the current booking
- avg_price_per_room: Average price per day of the reservation; prices of the rooms are dynamic. (in euros)
- no_of_special_requests: Total number of special requests made by the customer (e.g. high floor, view from the room, etc)
- booking_status: Flag indicating if the booking was canceled or not.

We deployed the database using the Bitnami MySQL stack (Gen1) on an Azure virtual machine with the following specifications:

- Architecture: x64

- VM Size: Standard B1s (1 vCPU, 1 GiB RAM)

The MySQL server was configured and managed through both Git Bash (for SSH and backup work) and MySQL Workbench (for query development and testing).

After setting up the MySQL server on Azure, we configured the database schema by creating a new database named hotel_db. The raw dataset was initially provided in CSV format, which required preprocessing to ensure compatibility with MySQL's data import process. We defined appropriate column data types based on the attributes in the dataset below.

Column	Type	Default Value	Nullable
Booking_ID	varchar(50)		YES
no_of_adults	int		YES
no_of_children	int		YES
no_of_weekend_nights	int		YES
no_of_week_nights	int		YES
type_of_meal_plan	varchar(100)		YES
required_car_parking...	int		YES
room_type_reserved	varchar(100)		YES
lead_time	int		YES
arrival_year	int		YES
arrival_month	int		YES
arrival_date	int		YES
market_segment_type	varchar(100)		YES
repeated_guest	int		YES
no_of_previous_canc...	int		YES
no_of_previous_book...	int		YES
avg_price_per_room	float		YES
no_of_special_requests	int		YES
booking_status	varchar(50)		YES

3. Query Analysis

To improve query performance and scalability in our hotel room reservation system, we applied MySQL table partitioning[2]. This allowed us to reduce the volume of scanned data and enhance response times for common analytical queries. We compared query performance before and after partitioning using both the original (small) dataset and an expanded version simulating real-world load.

— **Partitioning Factor Used**
We used 'arrival_year' as the primary partitioning column in our MySQL schema. This allowed us to :

- Efficiently filter data by year in queries.
- Take advantage of range partitioning, where each partition stores data for a specific year.
- Minimize the number of rows scanned in year-specific analytical queries.

Part 1 : Evaluation on Original Dataset (36,275 Rows)

We first tested partitioning on a smaller dataset to understand its impact in a controlled environment. Below are three representative queries executed before and after applying partitioning.

Sample Queries:
1. Average Room Price for 2017 Bookings
2. Count of Bookings per Room Type for 2017
3. Most Common Booking Month by Market Segment

Query Description	Table	Rows Scanned	Avg Duration (secs)
Average Room Price for 2017 Bookings	hotel_data	36k	0.047secs
	hotel_data_partitioned	6k	0.031secs
Count of Bookings per Room Type for 2017	hotel_data	36k	0.067secs
	hotel_data_partitioned	6k	0.031secs
Most Common Booking Month by Market Segment	hotel_data	36k	0.067secs
	hotel_data_partitioned	6k	0.031secs

Insight : Partitioning reduced the number of rows scanned by over 80%, and consistently decreased query execution time by 30–50% on average.

Part 2 : Evaluation on Expanded Dataset (1 Million Rows)

To simulate production-scale workloads, we expanded the dataset to over 1 million rows. We executed more complex queries before and after partitioning to assess performance under higher data volume.

Sample Queries:
1. Average Room Price for Guests with Special Requests
2. Bookings Distribution by Meal Plan in 2017
3. Maximum Lead Time for 2018 Bookings
4. Total Revenue Estimate by Market Segment

Query Description	Table	Rows Scanned	Avg Duration (secs)
Average Room Price for Guests with Special Requests	hotel_data	1 mil	8.4 secs
	hotel_data_partitioned	952k	6.8 secs
Bookings Distribution by Meal Plan in 2017	hotel_data	1 mil	8.1 secs
	hotel_data_partitioned	208k	1.34 secs
Find Maximum Lead Time for 2018 Bookings	hotel_data	1 mil	8.4secs
	hotel_data_partitioned	952k	6.9secs
Total Revenue Estimate by Market Segment (Price × No of Adults) (2017)	hotel_data	1 mil	8.1secs
	hotel_data_partitioned	208k	1.3 secs

Insight : Partitioning dramatically reduced the workload in filtered queries (e.g., year-based or meal plan filters). Execution times dropped by up to 80%, especially when partitions eliminated the need to scan large portions of the table.

Takeaways :

1. Partitioning Strategy: We used range partitioning based on arrival_year, enabling MySQL to target only relevant partitions during queries.
2. Performance Boost: Partitioning enhanced query speed, especially with filtering conditions like date, room type, or market segment.
3. Scalability: The partitioned table handled larger datasets efficiently, making it suitable for production-like environments.

4. Vertical Auto-Scaling

Vertical scaling was implemented using Azure-native tools and automation.

- Automation Setup:
    - Automation Account: Created in Azure to host scaling logic.
    - Runbook: A PowerShell script named ScaleUpVM was created and published, which:
        1. Stops the VM
        2. Changes the VM size from Standard_B1s to Standard_B2s
        3. Starts the VM again

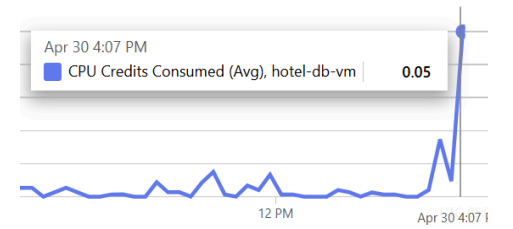
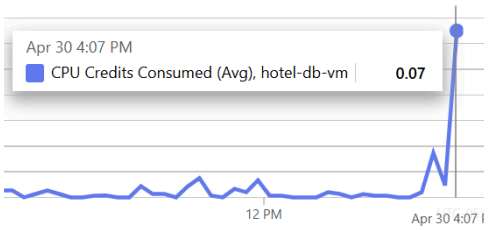
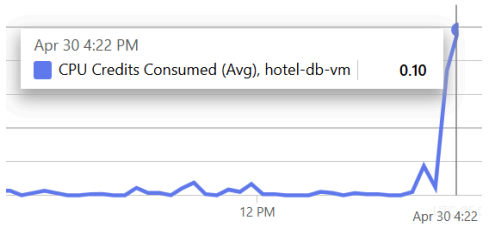
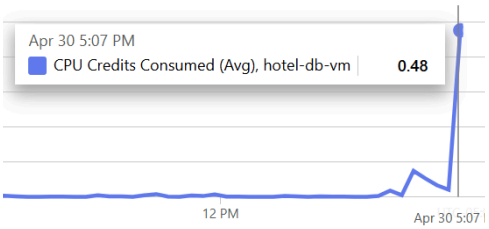
- Alert Configuration:
    - Metric Monitored in the experiments: Percentage CPU of the VM
    - Metrics that could be considered together while are not in the experiments: Disk I/O, Memory Usage and Disk Queue Length
    - Threshold: >80% average over 1 minutes
    - Check Frequency: Every 1 minute
    - Trigger Action: Invoke the automation runbook via an Action Group

- Action Group:
    - Linked the VM alert to the ScaleUpVM runbook
    - Passed necessary parameters (vmName, resourceGroup) to the script

5. Stress Test

- Stress Testing Setup
    - Environment: A MySQL database running on a Bitnami image inside an Azure Virtual Machine.
    - Data Model: The database contained hotel reservation data with fields like booking_id, no_of_adults, avg_price_per_room, etc.
    - Testing Script:
        - Implemented in Python using mysql-connector-python and faker.
        - Simulated 15000 concurrent queries (mix of realistic SELECT and INSERT operations).
        - Utilized ThreadPoolExecutor to simulate parallel clients under load.
        - Designed to saturate CPU resources and trigger a scaling event.
- Stress Test Experiment

Number of total queries	Number of insert and select queries	Number of threads	Time	CPU usage percentage
200	53 insert	20	16.84 seconds	5%

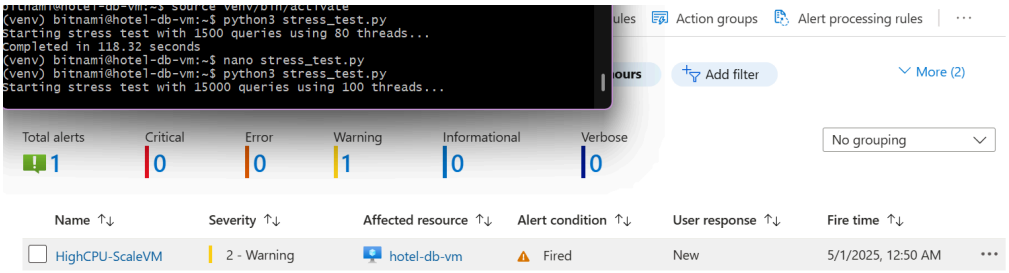
	queries and 147 select queries			 (venv) bitnami@hotel-db-vm:~$ python3 stress_test.py Starting stress test with 200 queries using 20 threads... Completed in 16.84 seconds
500	151 insert queries and 349 select queries	20	47.86 seconds	7%  (venv) bitnami@hotel-db-vm:~$ python3 stress_test.py Starting stress test with 500 queries using 20 threads... Completed in 47.86 seconds
500	159 insert queries and 341 select queries	40	41.42 seconds	10%  (venv) bitnami@hotel-db-vm:~$ python3 stress_test.py Starting stress test with 500 queries using 40 threads... Completed in 41.42 seconds
5000	1475 insert queries and 3525 select queries	100	411.47 seconds	48%  (venv) bitnami@hotel-db-vm:~$ python3 stress_test.py Starting stress test with 5000 queries using 100 threads... Completed in 411.47 seconds

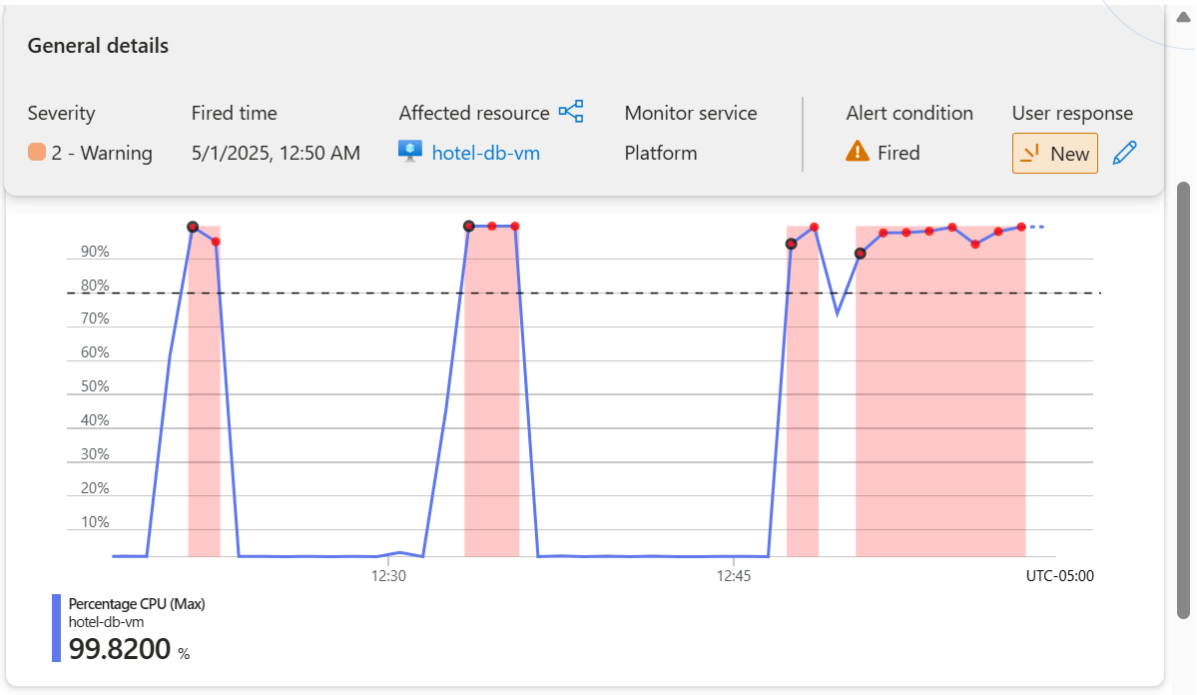
10000	N/A(Failed)	200	N/A(Failed)	Resulting in too many connections that crash the db. (venv) bitnami@hotel-db-vm:~$ python3 stress_test.py Starting stress test with 10000 queries using 200 threads... [INSERT ERROR]: 1040 (HY000): Too many connections [SELECT ERROR]: 1040 (HY000): Too many connections [SELECT ERROR]: 1040 (HY000): Too many connections [SELECT ERROR]: 1040 (HY000): Too many connections [INSERT ERROR]: 1040 (HY000): Too many connections [SELECT ERROR]: 1040 (HY000): Too many connections [SELECT ERROR]: 1040 (HY000): Too many connections [SELECT ERROR]: 1040 (HY000): Too many connections [INSERT ERROR]: 1040 (HY000): Too many connections [SELECT ERROR]: 1040 (HY000): Too many connections

- Stress Test triggering the Alert

When the auto-scaling is not properly set up and when the CPU usage percentage exceeded 80%, an alert was sent.

For example, 15000 queries executed using 100 threads. The operation utilized more than 80% of the GPU.

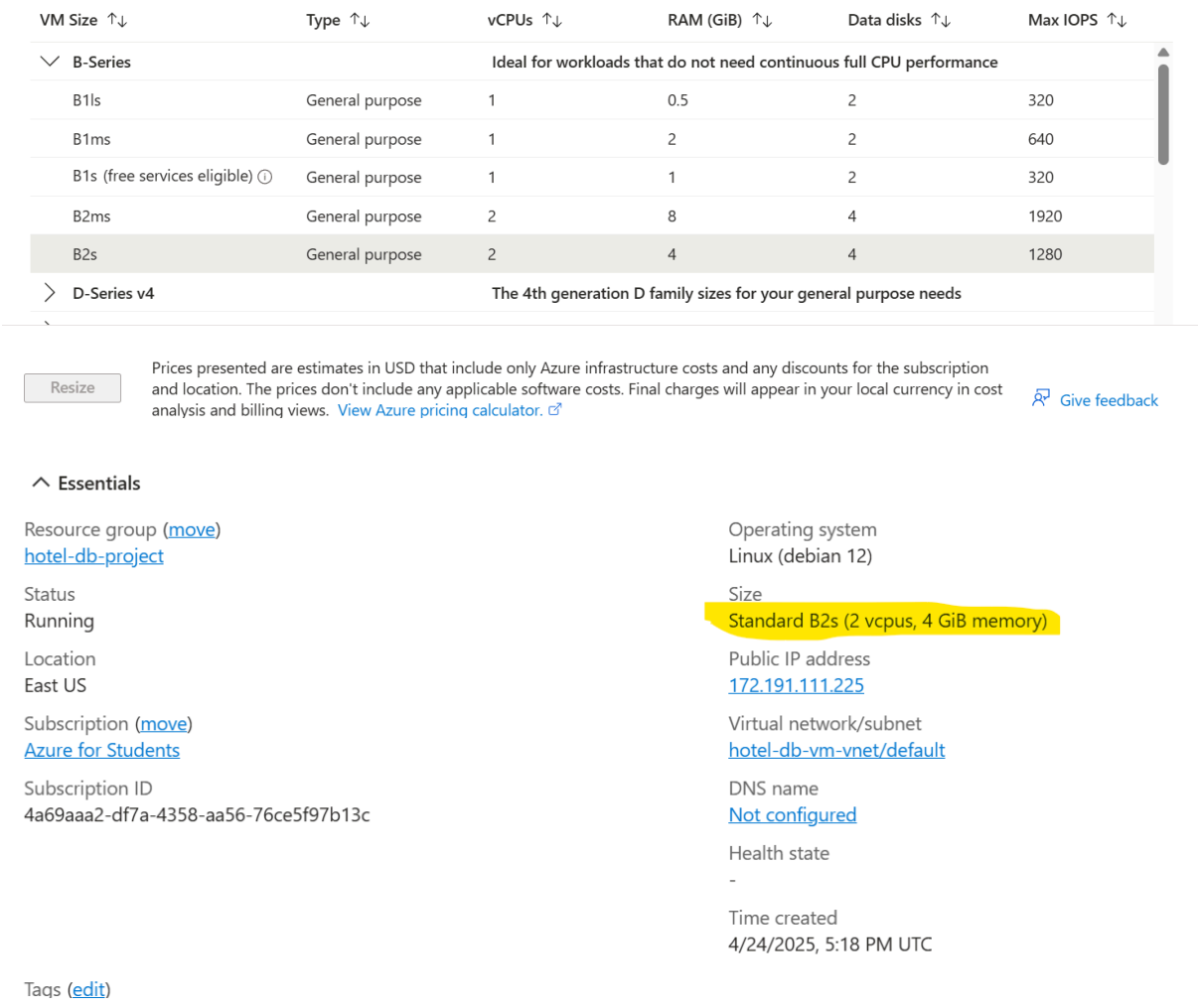




- Stress Test triggering the Auto-scaling

When the auto-scaling is properly set up and when the CPU usage percentage exceeds 80%, the automation workflow automatically resizes the VM to a more powerful instance, enabling better performance and availability.

For example, the VM was resized from B1s to B2s when the CPU usage percentage exceeded 80%.



VM Size	Type	vCPUs	RAM (GiB)	Data disks	Max IOPS
B-Series	Ideal for workloads that do not need continuous full CPU performance				
B1ls	General purpose	1	0.5	2	320
B1ms	General purpose	1	2	2	640
B1s (free services eligible)	General purpose	1	1	2	320
B2ms	General purpose	2	8	4	1920
B2s	General purpose	2	4	4	1280
D-Series v4	The 4th generation D family sizes for your general purpose needs				

Resize

Prices presented are estimates in USD that include only Azure infrastructure costs and any discounts for the subscription and location. The prices don't include any applicable software costs. Final charges will appear in your local currency in cost analysis and billing views. View Azure pricing calculator.

Give feedback

Essentials

Resource group (move)
hotel-db-project

Status
Running

Location
East US

Subscription (move)
Azure for Students

Subscription ID
4a69aaa2-df7a-4358-aa56-76ce5f97b13c

Operating system
Linux (debian 12)

Size
Standard B2s (2 vcpus, 4 GiB memory)

Public IP address
172.191.111.225

Virtual network/subnet
hotel-db-vm-vnet/default

DNS name
Not configured

Health state
-

Time created
4/24/2025, 5:18 PM UTC

Tags (edit)

6. Role-based Access Control

To ensure data security and enforce access policies, we implemented Role-Based Access Control (RBAC) in accordance with the Principle of Least Privilege to ensure that users are granted the minimum level of access necessary to perform their specific functions.

We defined four user roles as outlined in the table below. However, since our team

configured the database and managed all administrative tasks using the root account, we effectively operated as the Admin role throughout the project. Therefore, the Admin role is excluded from the setup process in this section.

Role	Access Level	Accessible Columns	Description
Admin	ALL PRIVILEGES	All columns	Full database control (can modify structure, manage users, etc.)
Manager	SELECT, INSERT, UPDATE	All columns	Oversees full operations but cannot drop table
Receptionist	SELECT, INSERT, UPDATE	All columns except financial and marketing data (avg_price_per_room, market_segment_type)	Can handle check-ins and changes, but cannot view financial or marketing data
Guest	SELECT via a filtered view (guest_view)	Only their own record via a guest_view	See their own bookings

The steps to configure secure, role-specific access control are summarized as follows.

- User account creation: we created separate user accounts for each non-admin role including manager, receptionist, and guest_user to simulate real-world access scenarios.

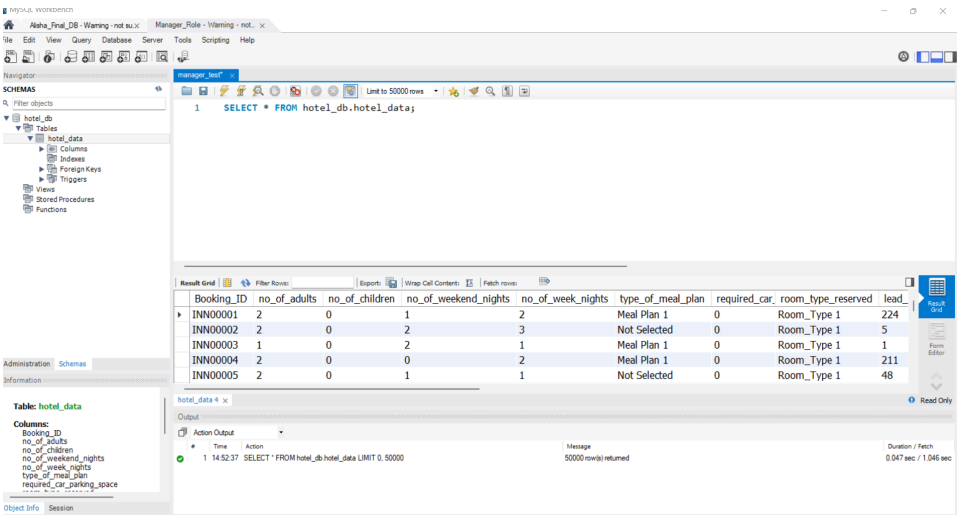
```
1    -- Manager
2 •  CREATE USER 'manager'@'%' IDENTIFIED BY 'manager123';
3
4    -- Receptionist
5 •  CREATE USER 'receptionist'@'%' IDENTIFIED BY 'rec123';
6
7    -- Guest
8 •  CREATE USER 'guest_user'@'%' IDENTIFIED BY 'guest123';
```

- Privilege Assignment: each user was granted permissions tailored to their role:
  - Manager: Granted SELECT, INSERT, and UPDATE privileges on all columns.

- Receptionist: Granted SELECT, INSERT, and UPDATE privileges, excluding access to specific columns including avg_price_per_room and market_segment_type.
- Guest: Granted read-only access to a filtered view (guest_view) that returns only booking information associated with a specific booking ID. In this case, we used Booking_ID = 'INN00001' as a proxy for guest identity.

```
10 •  GRANT SELECT, INSERT, UPDATE ON hotel_db.hotel_data TO 'manager'@'%';
11
12 •  GRANT SELECT (
13      Booking_ID, no_of_adults, no_of_children, no_of_weekend_nights,
14      no_of_week_nights, type_of_meal_plan, required_car_parking_space,
15      room_type_reserved, lead_time, arrival_year, arrival_month,
16      arrival_date, repeated_guest, no_of_previous_cancellations,
17      no_of_previous_bookings_not_canceled, no_of_special_requests,
18      booking_status
19    )
20    ON hotel_db.hotel_data TO 'receptionist'@'%';
21
22 •  GRANT INSERT, UPDATE ON hotel_db.hotel_data TO 'receptionist'@'%';
23
24 •  GRANT SELECT ON hotel_db.guest_view TO 'guest_user'@'%';
25 •   CREATE VIEW guest_view AS
26     SELECT Booking_ID, arrival_year, arrival_month, arrival_date,
27            room_type_reserved, booking_status
28     FROM hotel_db.hotel_data
29     WHERE Booking_ID = "INN00001";
```

- Permission Verification: Separate connections in MySQL Workbench were established for each role. Each user could only perform actions permitted by their assigned privileges.
    - Manger: can select, update, insert but cannot drop or delete



```
3 •  UPDATE hotel_db.hotel_data
4    SET type_of_meal_plan = 'Meal Plan 2'
5    WHERE Booking_ID = 'INN00040';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	r
INN00040	2	0	2	1	Meal Plan 2	0

```
7 • INSERT INTO hotel_db.hotel_data (
8       Booking_ID, no_of_adults, no_of_children, no_of_weekend_nights,
9       no_of_week_nights, type_of_meal_plan, required_car_parking_space,
10      room_type_reserved, lead_time, arrival_year, arrival_month, arrival_date,
11      market_segment_type, repeated_guest, no_of_previous_cancellations,
12      no_of_previous_bookings_not_canceled, avg_price_per_room, no_of_special_requests,
13      booking_status
14  ) VALUES (
15      'INNTEST00', 2, 0, 1, 2, 'Meal Plan 1', 0, 'Room_Type 10', 30,
16      2025, 5, 1, 'Online', 0, 0, 0, 100.00, 1, 'Not Canceled'
17  );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car	room_type_reserved	lead_
INNTEST00	2	0	1	2	Meal Plan 1	0	Room_Type 10	30

```
19 • DELETE FROM hotel_db.hotel_data
20   WHERE Booking_ID = 'INN00001';
21
22 • DROP TABLE hotel_db.hotel_data;
```

Output | Action Output

#	Time	Action	Message
1	15:38:22	DELETE FROM hotel_db.hotel_data WHERE Booking_ID = 'INN00001'	Error Code: 1142. DELETE command denied to user 'manager'@'127.0.0.1' for table 'hotel_data'
2	15:38:40	DROP TABLE hotel_db.hotel_data	Error Code: 1142. DROP command denied to user 'manager'@'127.0.0.1' for table 'hotel_data'

- Receptionist: can select some columns, update, insert all columns but cannot drop or delete

hotel_data | Limit to 50000 rows

```
1 • SELECT Booking_ID, no_of_adults, room_type_reserved, booking_status
2   FROM hotel_db.hotel_data;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Booking_ID	no_of_adults	room_type_reserved	booking_status
INN00001	2	Room_Type 1	Not_Canceled
INN00002	2	Room_Type 1	Not_Canceled
INN00003	1	Room_Type 1	Canceled
INN00004	2	Room_Type 1	Canceled
INN00005	2	Room_Type 1	Canceled
INN00006	2	Room_Type 1	Canceled
INN00007	2	Room_Type 1	Not_Canceled
INN00008	2	Room_Type 4	Not_Canceled
INN00009	3	Room_Type 1	Not_Canceled
INN00010	2	Room_Type 4	Not_Canceled
INN00011	1	Room_Type 1	Not_Canceled
INN00012	1	Room_Type 4	Not_Canceled
INN00013	2	Room_Type 1	Canceled

```sql
SELECT avg_price_per_room
FROM hotel_db.hotel_data;

SELECT *
FROM hotel_db.hotel_data;
```

Output

Action Output

#	Time	Action	Message	Durat
1	15:51:25	SELECT avg_price_per_room FROM hotel_db.hotel_data LIMIT 0, 50000	Error Code: 1143. SELECT command denied to user 'receptionist'@'127.0.0.1' for column 'avg_price_per_...	0.031
2	15:51:29	SELECT * FROM hotel_db.hotel_data LIMIT 0, 50000	Error Code: 1142. SELECT command denied to user 'receptionist'@'127.0.0.1' for table 'hotel_data'	0.046

```sql
INSERT INTO hotel_db.hotel_data (
    Booking_ID, no_of_adults, no_of_children, no_of_weekend_nights,
    no_of_week_nights, type_of_meal_plan, required_car_parking_space,
    room_type_reserved, lead_time, arrival_year, arrival_month, arrival_date,
    market_segment_type, repeated_guest, no_of_previous_cancellations,
    no_of_previous_bookings_not_canceled, avg_price_per_room, no_of_special_requests,
    booking_status
) VALUES (
    'INNTEST01', 2, 0, 1, 2, 'Meal Plan 1', 0, 'Room_Type 10', 30,
    2025, 5, 1, 'Online', 0, 0, 0, 100.00, 1, 'Not Canceled'
);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Booking_ID	type_of_meal_plan	booking_status
INNTEST01	Meal Plan 1	Not Canceled

```sql
UPDATE hotel_db.hotel_data
SET type_of_meal_plan = 'Meal Plan 3'
WHERE Booking_ID = 'INN00040';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Booking_ID	type_of_meal_plan
INN00040	Meal Plan 3

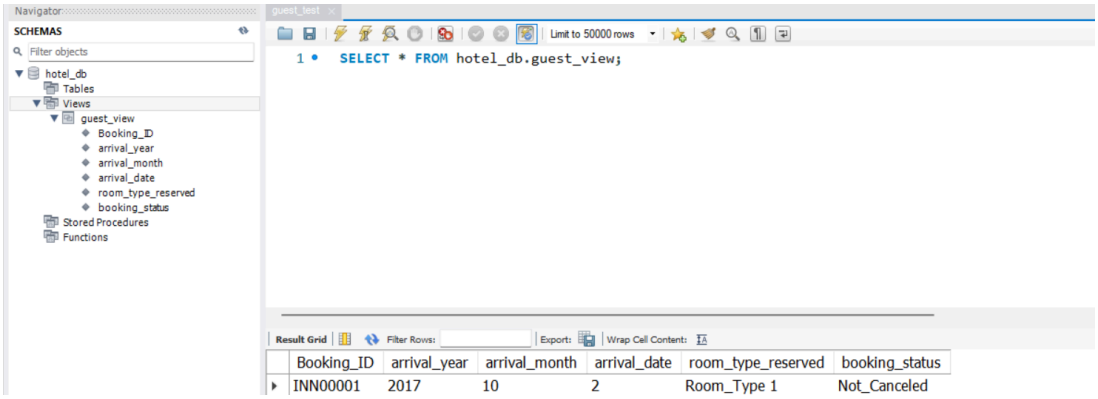
```sql
DELETE FROM hotel_db.hotel_data
WHERE Booking_ID = 'INN00001';

DROP TABLE hotel_db.hotel_data;
```

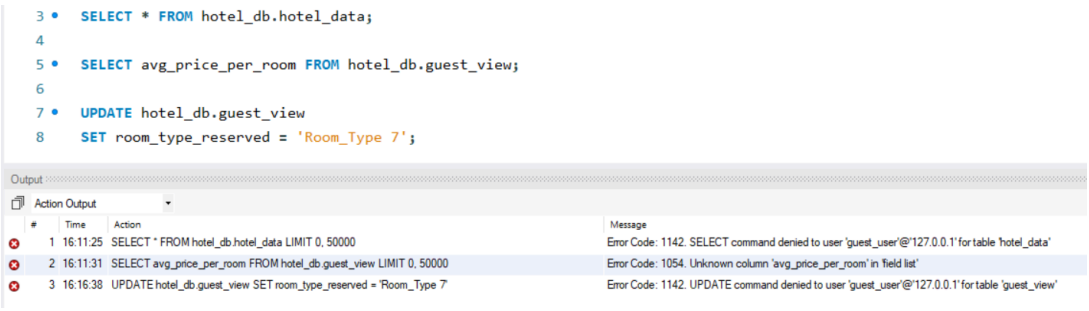
Output

Action Output

#	Time	Action	Message
1	15:59:41	DELETE FROM hotel_db.hotel_data WHERE Booking_ID = 'INN00001'	Error Code: 1142. DELETE command denied to user 'receptionist'@'127.0.0.1' for table 'hotel_data'
2	15:59:43	DROP TABLE hotel_db.hotel_data	Error Code: 1142. DROP command denied to user 'receptionist'@'127.0.0.1' for table 'hotel_data'

- Guest: can select only in guest_view only

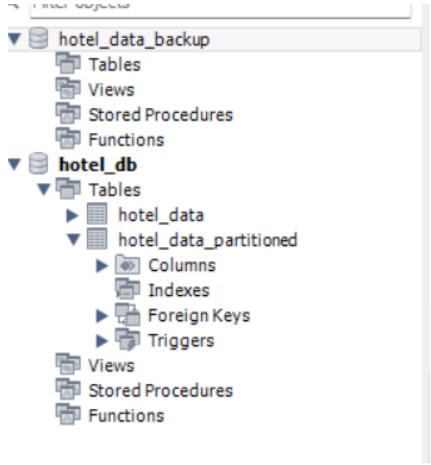




7. Backup

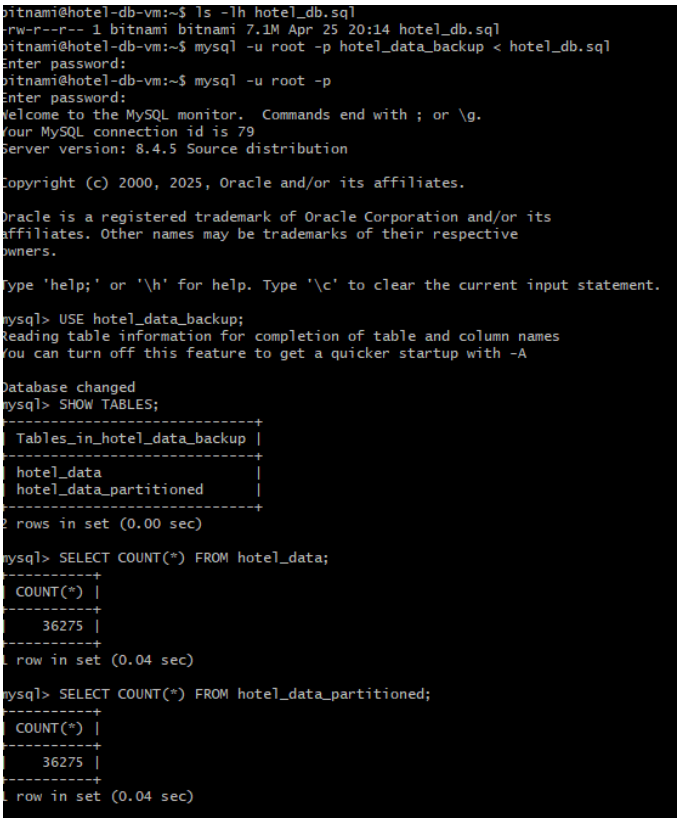
We implemented and tested both **manual** and **fully automated** database backup procedures.

- **Manual Backup**
  1. Created a new empty database named hotel_data_backup to act as the restore target.

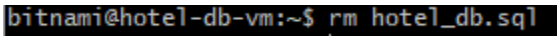


  2. Performed a database dump using mysqldump on the primary database hotel_db. We then restored the dump file into the newly created hotel_data_backup database. Then, we verified by comparing row counts

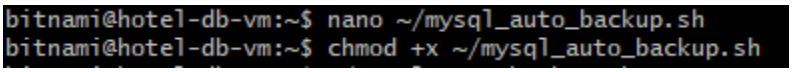
between the source and target tables.



3. Deleted the dump file for implementing an automated backup process.



- Fully Automated
    1. Created a custom backup shell script (mysql_auto_backup.sh) using the nano text editor. The script used mysqldump to export the hotel_db database with timestamped filenames.



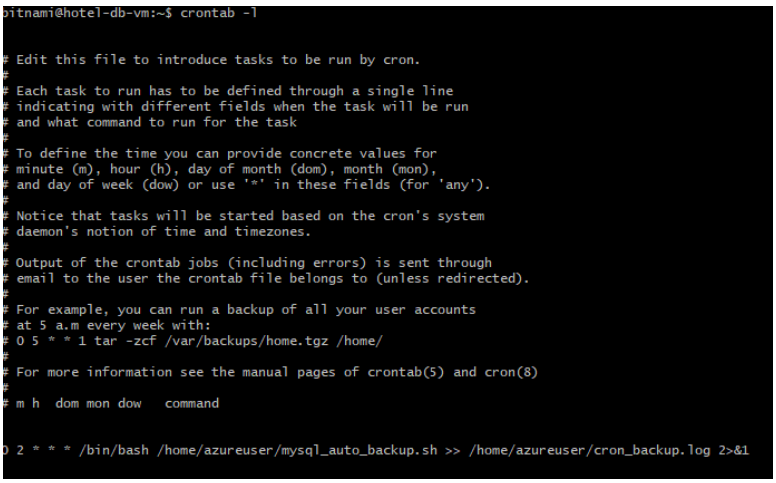
```
bitnami@hotel-db-vm: ~
  GNU nano 7.2                                        /home/azureuser/mysql_auto_backup.sh
#!/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/opt/bitnami/mysql/bin


DATE=$(date +%F_%H-%M)
BACKUP_DIR="/home/azureuser/mysql_backups"
BACKUP_FILE="$BACKUP_DIR/hotel_db_$DATE.sql"

mkdir -p $BACKUP_DIR

/opt/bitnami/mysql/bin/mysqldump --defaults-extra-file=/home/azureuser/.my.cnf hotel_db > $BACKUP_FILE

if [ $? -ne 0 ]; then
  echo "Backup failed at $DATE" >> /home/azureuser/cron_backup.log
else
  echo "Backup succeeded at $DATE" >> /home/azureuser/cron_backup.log

fi
```

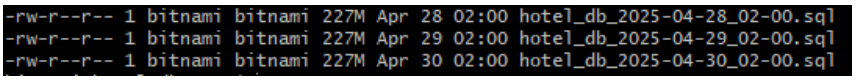
2. Set up a cron job using the crontab -e command (crontab -1 command to view) to schedule the script to run every 5 minutes. The backup files were successfully generated in the backup directory.

```
bitnami@hotel-db-vm:~$ crontab -l


# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command


*/5 * * * * /bin/bash /home/azureuser/mysql_auto_backup.sh >> /home/azureuser/cron_backup.log 2>&1
```

```
bitnami@hotel-db-vm:~$ ls -lh ~/mysql_backups/
total 5.8G
-rw-r--r-- 1 bitnami bitnami 7.1M Apr 25 20:38 hotel_db_2025-04-25_20-38.sql
-rw-r--r-- 1 bitnami bitnami 7.1M Apr 25 21:54 hotel_db_2025-04-25_21-54.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 26 21:38 hotel_db_2025-04-26_21-38.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 26 21:40 hotel_db_2025-04-26_21-40.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 26 21:45 hotel_db_2025-04-26_21-45.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 26 21:50 hotel_db_2025-04-26_21-50.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 26 21:55 hotel_db_2025-04-26_21-55.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 26 22:00 hotel_db_2025-04-26_22-00.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 26 22:05 hotel_db_2025-04-26_22-05.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 26 22:10 hotel_db_2025-04-26_22-10.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 26 22:15 hotel_db_2025-04-26_22-15.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 26 22:20 hotel_db_2025-04-26_22-20.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 27 02:00 hotel_db_2025-04-27_02-00.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 27 02:10 hotel_db_2025-04-27_02-10.sql
-rw-r--r-- 1 bitnami bitnami    0 Apr 27 02:15 hotel_db_2025-04-27_02-15.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 16:43 hotel_db_2025-04-27_16-43.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 16:55 hotel_db_2025-04-27_16-55.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 17:13 hotel_db_2025-04-27_17-13.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 17:14 hotel_db_2025-04-27_17-14.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 17:30 hotel_db_2025-04-27_17-30.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 17:35 hotel_db_2025-04-27_17-35.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 17:40 hotel_db_2025-04-27_17-40.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 17:45 hotel_db_2025-04-27_17-45.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 17:50 hotel_db_2025-04-27_17-50.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 17:55 hotel_db_2025-04-27_17-55.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:00 hotel_db_2025-04-27_18-00.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:05 hotel_db_2025-04-27_18-05.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:10 hotel_db_2025-04-27_18-10.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:15 hotel_db_2025-04-27_18-15.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:20 hotel_db_2025-04-27_18-20.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:25 hotel_db_2025-04-27_18-25.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:30 hotel_db_2025-04-27_18-30.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:35 hotel_db_2025-04-27_18-35.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:40 hotel_db_2025-04-27_18-40.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:45 hotel_db_2025-04-27_18-45.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:50 hotel_db_2025-04-27_18-50.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 18:55 hotel_db_2025-04-27_18-55.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 19:00 hotel_db_2025-04-27_19-00.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 19:05 hotel_db_2025-04-27_19-05.sql
-rw-r--r-- 1 bitnami bitnami 227M Apr 27 19:10 hotel_db_2025-04-27_19-10.sql
```

3. The cron job was later modified to execute daily at 2:00 AM, and the backup files were successfully generated in the backup directory with timestamps at 2.00 AM.





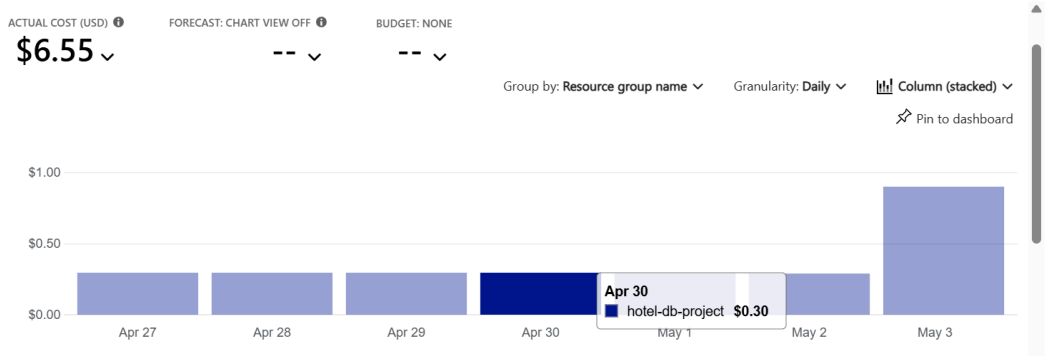
8. Cost Estimation

Our deployment of the hotel room booking system was hosted on Microsoft Azure using a virtual machine (VM) configured with the Bitnami MySQL Gen1 stack. The cost estimation[3] below reflects our realistic usage pattern, primarily operating on a Standard B1s VM, with temporary vertical auto-scaling to Standard B2s during high workload conditions.

A. Actual Observed Costs

Based on Azure's Cost Management insights, our system incurred a total cost of $6.55 USD as of May 3, 2025, with daily operating costs ranging from $0.20 to $1.00 depending on VM usage and configuration.

- Resource Group: hotel-db-project
- Cost includes: VM runtime, public IP, storage, and monitoring.



B. Virtual Machine Pricing

VM Size	Specs	Purpose	Monthly Cost
Standard B1s	1 vCPU, 1 GiB RAM	Main configuration	$10.22
Standard B2s	2 vCPU, 4 GiB RAM	Temporary scale-up	$36.21

We used B1s as our default VM for most of the project's life cycle. The B2s size was used temporarily during stress tests to evaluate performance under simulated high-concurrency workloads. Once the load subsided, the system could be reverted to the more cost-effective B1s tier, maintaining a low operational budget.

C. Itemized Monthly Cost (Baseline – B1s)

Component	Monthly Cost (USD)	Description
VM (Standard B1s)	$10.22	Primary VM tier
OS Disk (30 GB SSD)	~$3.00	OS disk with Bitnami MySQL
Public IP (Static)	~$3.65	Persistent external IP
Outbound Bandwidth (100 GB)	~$8.70	Data transfer costs beyond free tier
Backup Storage (Daily .sql)	~$5.00	Automated via cron jobs
Monitoring & Automation	~$2.00	Alerts and scaling runbooks

Total Monthly Estimate (B1s setup): ~$32.57

D. Additional (Optional) Scaling Cost

If auto-scaling triggers and the system moves to B2s for 5 days per month, the cost adjustment would be:

- B2s Daily Rate (approx.): ~$1.20/day
- 5-day usage: 5 × 1.20 = $6.00 extra/month

Once performance stabilizes, the admin can manually or automatically revert to B1s to control costs.

E. Annual Estimate (with light scaling)

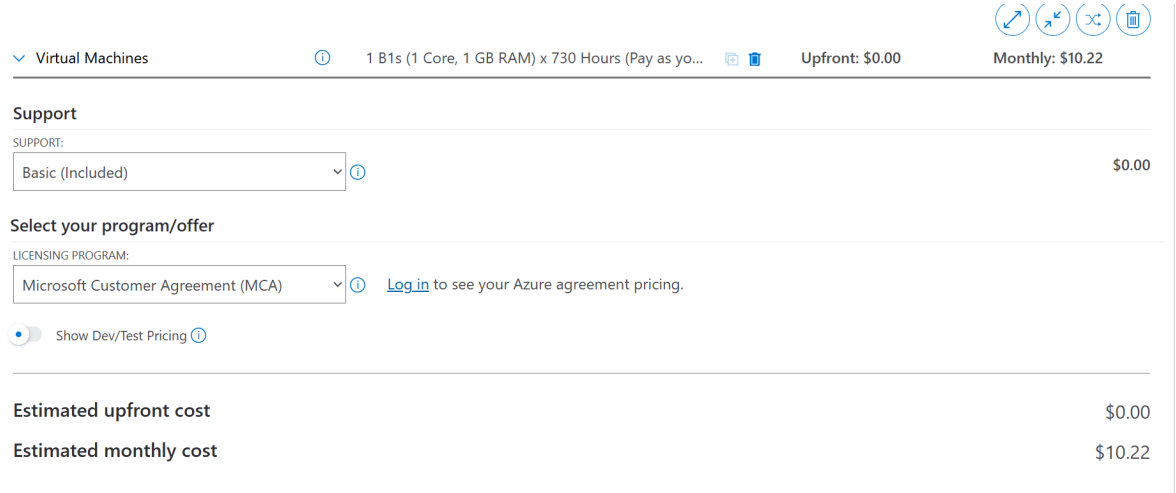
Usage Scenario	Annual Cost
B1s only	~$32.57 × 12 = $390.84
B1s with occasional B2s scaling	$390.84 + ($6 × 12) = $462.84

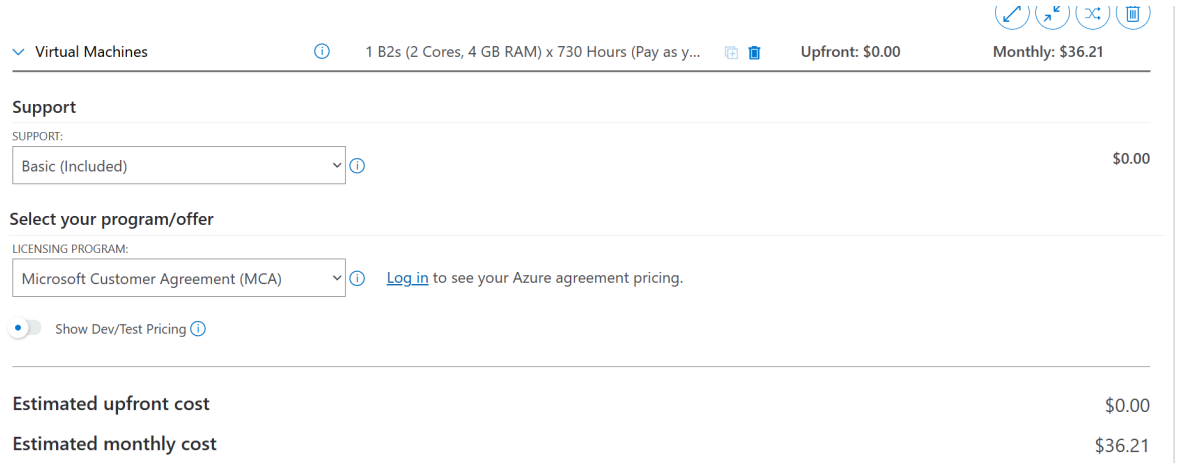
9. Reference

[1]. Data source:
https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset/data

[2]. Partitioning:
https://dev.mysql.com/doc/refman/8.0/en/partitioning-overview.html
https://stackoverflow.com/questions/69323216/whats-the-best-way-performance-wise-to-select-from-a-partitioned-table-in-mysql

[3]. Azure pricing:
https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/





https://azure.microsoft.com/en-us/pricing/details/managed-disks/
https://azure.microsoft.com/en-us/pricing/details/ip-addresses/
https://azure.microsoft.com/en-us/pricing/details/bandwidth/