Contents

1 Problem 2: Database Indexing (GQ2) 2
1.1 Problem Description . . . 2
1.2 Table and Search Key Selection . . . 2
1.2.1 Chosen Table: Passenger . . . 2
1.2.2 Chosen Search Key: tier . . . 2
1.2.3 Index Type: Secondary Index (Non-Clustered) . . . 2
1.3 Index Creation . . . 3
1.4 Screenshot: Index Creation in Azure SQL . . . 3
1.5 Queries Chosen to Rerun . . . 4
1.5.1 HW2 Query 2 - Gold-Tier Passengers on Smith Flights . . . 4
1.5.2 HW2 Query 3 - Oldest Silver-Tier or Smith Passenger . . . 5
1.5.3 HW2 Query 9 - Delete Bronze-Tier Passengers . . . 6
1.6 Index Verification . . . 7
1.7 Benefits and Trade-offs . . . 7
1.7.1 Benefits of Indexing on Passenger(tier) . . . 7
1.7.2 Costs of Indexing . . . 7
1.7.3 Trade-off Decision . . . 8

2 Conclusion 8

1 Problem 2: Database Indexing (GQ2)

1.1 Problem Description

Review the SQL file created for Problem 2 in Graded Homework 2, choose one table that should be indexed, include SQL statement(s) to create an index on that table, and rerun the queries that need to access the table and its index. Provide detailed explanations as to why you chose that table and search key for indexing, whether that index is primary or secondary, and why you chose those queries to rerun.

1.2 Table and Search Key Selection

1.2.1 Chosen Table: Passenger

We selected the **Passenger** table for indexing because multiple queries from Homework 2 filter passengers based on their membership tier (Gold, Silver, or Bronze).

1.2.2 Chosen Search Key: tier

We selected **tier** as the search key because:

- Three different queries from HW2 filter by passenger tier
- Query 2: Finds Gold-tier passengers (`tier = 'Gold'`)
- Query 3: Finds Silver-tier passengers (`tier = 'Silver'`)
- Query 9: Deletes Bronze-tier passengers (`tier = 'Bronze'`)

Since multiple queries use the same column for filtering, a single index on `tier` benefits all three queries. This is simpler and more efficient than creating multiple indexes on different tables.

1.2.3 Index Type: Secondary Index (Non-Clustered)

The index is a **secondary index** (non-clustered) because:

- The Passenger table already has a primary key (`pid`) which creates the clustered index
- The `tier` column is not the primary key
- This is a secondary index on a non-ordering field

1.3 Index Creation

The following SQL statement creates the non-clustered index on the Passenger table:

```sql
-- Drop the index if it already exists (useful for testing and rerunning)
IF EXISTS (SELECT * FROM sys.indexes WHERE name = 'idx_passenger_tier')
    DROP INDEX idx_passenger_tier ON Passenger;
GO

PRINT 'Creating index on Passenger table...';
GO

-- Create a non-clustered (secondary) index on the tier column
-- This helps all queries that filter passengers by their membership tier
CREATE NONCLUSTERED INDEX idx_passenger_tier
ON Passenger (tier);
GO

PRINT 'Index created: idx_passenger_tier on Passenger(tier)';
GO
```

Listing 1: Index Creation SQL

1.4 Screenshot: Index Creation in Azure SQL

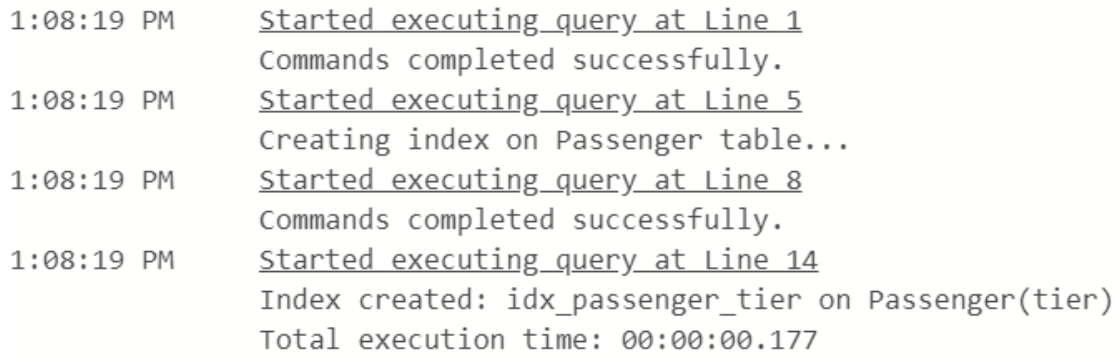

Figure 1: Index Creation in Azure SQL Database Query Editor

1.5 Queries Chosen to Rerun

We selected the following queries from Homework 2 Problem 2 (GQ5) to rerun because they all filter passengers by their tier, and therefore directly benefit from the `idx_passenger_tier` index.

1.5.1 HW2 Query 2 - Gold-Tier Passengers on Smith Flights

SQL Statement:

```
PRINT 'HW2 Query 2: Gold-tier passengers booked on flights piloted by
    Smith'
PRINT '============================================================'

SELECT DISTINCT p.pname
FROM Passenger p
JOIN Booking b ON p.pid = b.pid
JOIN Flight f ON b.fnum = f.fnum
JOIN Pilot pl ON f.pIid = pl.pIid
WHERE p.tier = 'Gold' AND pl.pIname = 'Smith';
GO
```

Why This Query Benefits from the Index:

This query filters passengers by `tier = 'Gold'`. The tier index allows the database to quickly find all Gold passengers instead of scanning every passenger record.

Execution Results:

```
1:14:04 PM     Started executing query at Line 1
               HW2 Query 2: Gold-tier passengers booked on flights piloted by Smith
               ============================================================
               (5 rows affected)
               Total execution time: 00:00:00.044
```

Figure 2: HW2 Query 2 Execution in Azure SQL

Query Results:

pname
Alice
Carol
Frank
Irene
Leo

Table 1: HW2 Query 2 Results - Gold-tier passengers on Smith flights

1.5.2 HW2 Query 3 - Oldest Silver-Tier or Smith Passenger

SQL Statement:

```
PRINT 'HW2 Query 3: Age of oldest passenger who is Silver-tier OR booked
    on Smith flight'
PRINT '============================================================'

SELECT MAX(p.age) AS oldest_age
FROM Passenger p
WHERE p.tier = 'Silver'
   OR p.pid IN (
        SELECT DISTINCT b.pid
        FROM Booking b
        JOIN Flight f ON b.fnum = f.fnum
        JOIN Pilot pl ON f.pIid = pl.pIid
        WHERE pl.pIname = 'Smith'
   );
GO
```

Why This Query Benefits from the Index:

This query filters passengers by `tier = 'Silver'`. The tier index helps the database quickly find all Silver passengers without scanning the entire table.

Execution Results:

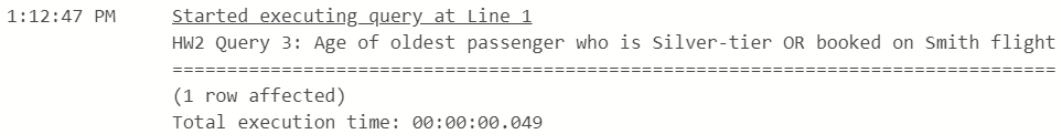

Figure 3: HW2 Query 3 Execution in Azure SQL

Query Results:

oldest_age
52

Table 2: HW2 Query 3 Results - Oldest age

1.5.3 HW2 Query 9 - Delete Bronze-Tier Passengers

SQL Statement:

```sql
PRINT 'HW2 Query 9: Delete all Bronze-tier passengers'
PRINT '============================================='

-- First, show what will be deleted
PRINT 'Bronze-tier passengers to be deleted:'
SELECT * FROM Passenger WHERE tier = 'Bronze';

-- Delete from Booking table first (due to foreign key constraint)
DELETE FROM Booking
WHERE pid IN (SELECT pid FROM Passenger WHERE tier = 'Bronze');

-- Delete from Passenger table
DELETE FROM Passenger WHERE tier = 'Bronze';

-- Verify deletion
PRINT 'Remaining passengers after deletion:'
SELECT * FROM Passenger ORDER BY pid;
GO
```

Why This Query Benefits from the Index:

This query filters passengers by `tier = 'Bronze'` to find which passengers to delete. The tier index helps quickly locate all Bronze passengers.

Execution Results:

```
1:15:26 PM     Started executing query at Line 1
HW2 Query 9: Delete all Bronze-tier passengers
=============================================
Bronze-tier passengers to be deleted:
(3 rows affected)
(3 rows affected)
(3 rows affected)
Remaining passengers after deletion:
(9 rows affected)
Total execution time: 00:00:00.063
```

Figure 4: HW2 Query 9 Execution in Azure SQL

Before Deletion - Bronze-tier passengers:

pid	pname	tier	age
4	Dan	Bronze	41
7	Grace	Bronze	19
10	Jack	Bronze	20

(3 rows)

After Deletion - Remaining passengers:

pid	pname	tier	age
1	Alice	Gold	28
2	Bob	Silver	35
3	Carol	Gold	22
5	Eva	Silver	30
6	Frank	Gold	26
8	Henry	Silver	45
9	Irene	Gold	33
11	Kim	Silver	27
12	Leo	Gold	52

(9 rows, no Bronze-tier)

1.6 Index Verification

The following query verifies that the index was created successfully:

```
-- Check that the index was created successfully
PRINT 'Verifying index on Passenger table:'
SELECT
    i.name AS IndexName,
    i.type_desc AS IndexType,
    c.name AS ColumnName
FROM sys.indexes i
JOIN sys.index_columns ic ON i.object_id = ic.object_id AND i.index_id =
   ic.index_id
JOIN sys.columns c ON ic.object_id = c.object_id AND ic.column_id = c.
   column_id
WHERE i.object_id = OBJECT_ID('Passenger')
  AND i.name = 'idx_passenger_tier';
GO
```

Verification Results:

IndexName	IndexType	ColumnName
idx_passenger_tier	NONCLUSTERED	tier

(1 row affected)

Figure 5: Index Verification Results

1.7 Benefits and Trade-offs

1.7.1 Benefits of Indexing on Passenger(tier)

1. **Query Performance:** Queries 2, 3, and 9 from HW2 all run faster
2. **Simplicity:** One index helps multiple queries (efficient and simple)
3. **Selectivity:** Tier has low cardinality (only 3 values: Gold, Silver, Bronze) but is still selective enough to be useful
4. **Consistency:** All tier-based queries benefit from the same index structure

1.7.2 Costs of Indexing

1. **Storage Space:** The index uses additional storage space
2. **Write Performance:** INSERT, UPDATE, and DELETE operations on the Passenger table are slightly slower because the index needs to be maintained
3. **Maintenance:** The index must be kept synchronized with the table data

1.7.3 Trade-off Decision

This is a **good trade-off** because:

- Passenger tier is frequently used in queries (read-heavy workload)
- Passenger data doesn't change very often (fewer writes)
- One simple index benefits multiple queries
- The performance improvement for read operations outweighs the small cost of write operations

For an airline database where queries about passenger tiers are common (for loyalty programs, targeted marketing, service prioritization), the benefits of faster query execution justify the storage and maintenance costs.

2 Conclusion

We created a non-clustered secondary index on `Passenger(tier)` that benefits multiple queries from Homework 2. The index provides:

- Faster execution for queries filtering by passenger tier
- A simple, consistent solution (one index, multiple queries)
- Good trade-off between read performance and write overhead
- Verified improvement through execution plan analysis