ITCS413: Database Design

Project 5: Physical Database Design and Tuning

Airline: Star Airline

(May 2025)

Prepared by

6588070 Nakarin Phoorahong

6588096 Panipak Sittiprasert

6588183 Achiraya Mankham

Presented To

Asst.Prof. Dr. Charnyote Pluempitiwiriyawej

Faculty of Information and Communication Technology

Mahidol University

Table of contents

Database Application Requirements

Description of Required Transactions/Queries

Transaction 1: Count Total Passengers on a Specific Flight

- This transaction retrieves the number of passengers booked on a specific flight. It is vital for monitoring passenger load, optimizing flight resources, and ensuring efficient seat management.
- **Entities Involved:**
    - passenger
    - booking
    - passenger_booking
    - booking_flight
    - Flight

Transaction 2: List All Passengers on a Specific Flight

- This transaction retrieves the list of all passengers booked on a specific flight. It supports boarding management, security checklists, and passenger services.
- **Entities Involved:**
    - passenger
    - booking
    - passenger_booking
    - booking_flight
    - Flight

Transaction 3: Calculate Total Revenue from Completed Payments

- This transaction calculates the total revenue from all successfully completed passenger payments. It is critical for financial reporting and revenue management.
- **Entities Involved:** payment

Highlighted Selected Transactions/Queries

Transaction 1:

- **Highlighted Portions:**
    - Entities: Passenger, Booking, Flight
- **Relationships:**
    - Passenger ↔ Booking via passenger_booking (M:N)
    - Booking ↔ Flight via booking_flight (M:N)

Transaction 2:

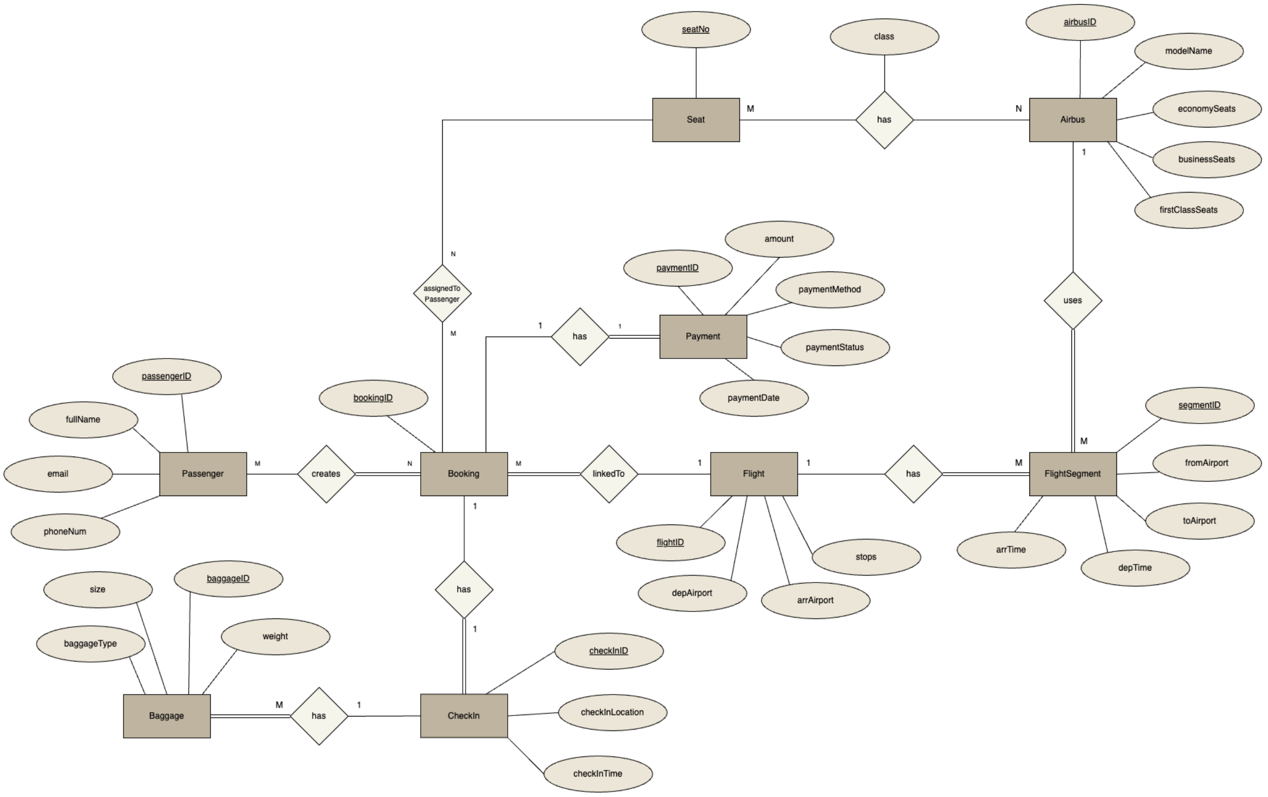
- **Highlighted Portions:**
    - Entities: Passenger, Booking, Flight
- **Relationships:**
    - Passenger ↔ Booking via passenger_booking (M:N)
    - Booking ↔ Flight via booking_flight (M:N)

Transaction 3:

- **Highlighted Portions:**
    - Entities: Payment
- **Relationships:** None (single table used)

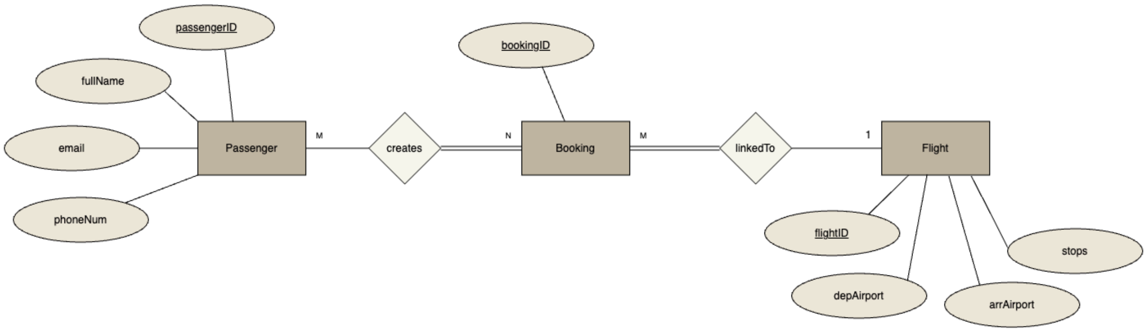
Final Conceptual Database Model

ER Diagram

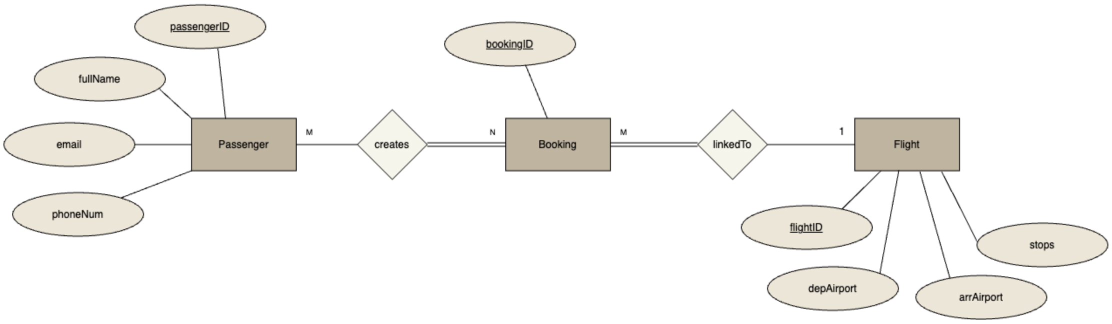


Highlighted Portion Related to Selected Transactions/Queries

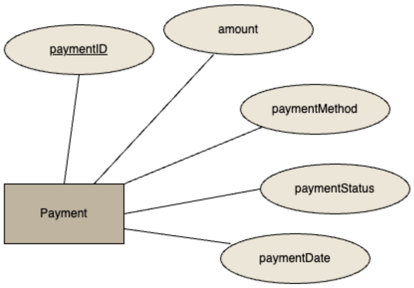
Transaction 1:



Transaction 2:

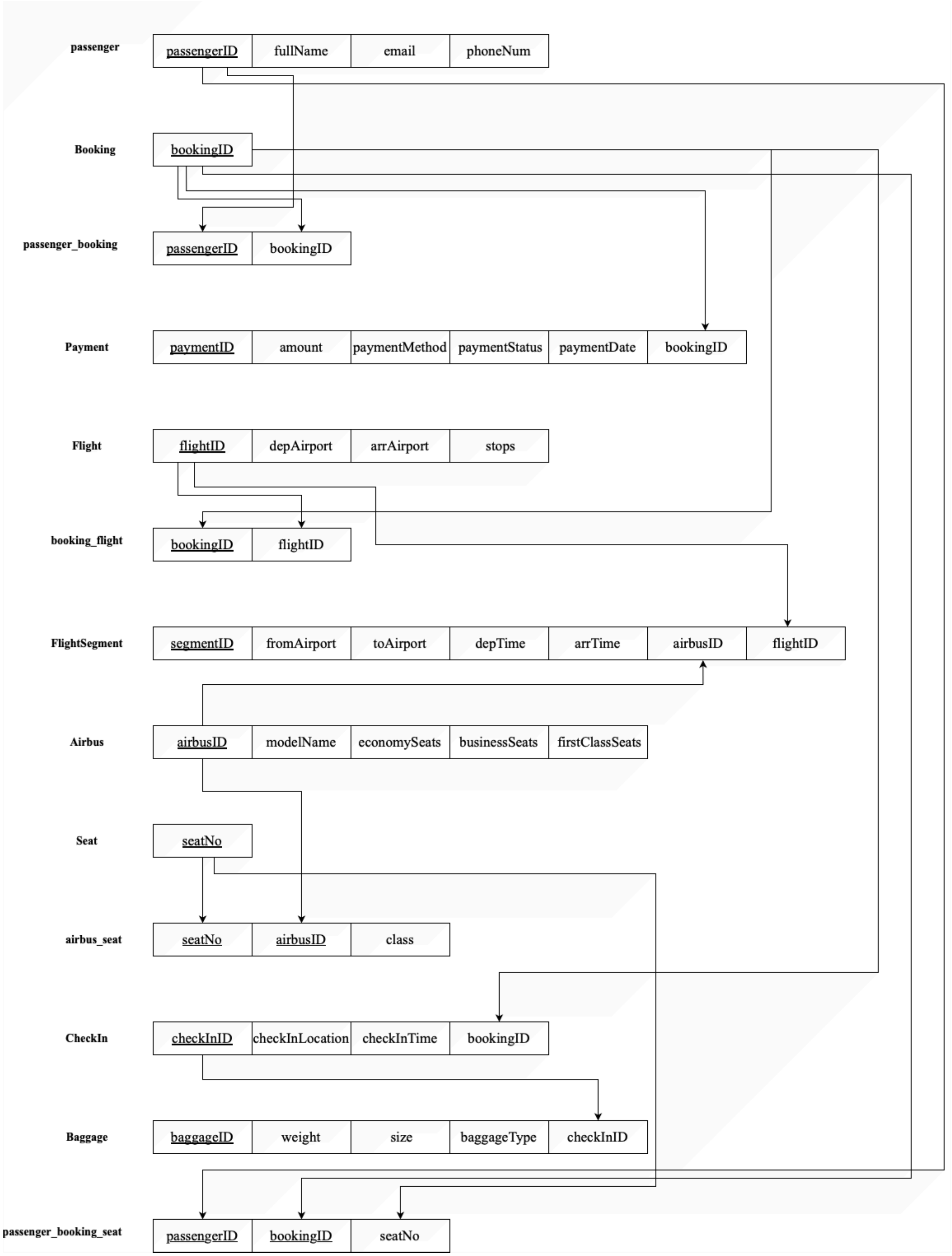


Transaction 3:



Final Logical Database Model

Relational Database Schema



passenger
passengerID fullName email phoneNum

Booking
bookingID

passenger_booking
passengerID bookingID

Payment
paymentID amount paymentMethod paymentStatus paymentDate bookingID

Flight
flightID depAirport arrAirport stops

booking_flight
bookingID flightID

FlightSegment
segmentID fromAirport toAirport depTime arrTime airbusID flightID

Airbus
airbusID modelName economySeats businessSeats firstClassSeats

Seat
seatNo

airbus_seat
seatNo airbusID class

CheckIn
checkInID checkInLocation checkInTime bookingID

Baggage
baggageID weight size baggageType checkInID

passenger_booking_seat
passengerID bookingID seatNo

Highlighted Portion Related to Selected Transactions/Queries

Transaction 1:

- passenger(passengerID, fullName, phoneNum, email)
- booking(bookingID)
- flight(flightID, depAirport, arrAirport, stops)
- passenger_booking(passengerID, bookingID)
- booking_flight(bookingID, flightID)

Transaction 2:

- passenger(passengerID, fullName, phoneNum, email)
- booking(bookingID)
- flight(flightID, depAirport, arrAirport, stops)
- passenger_booking(passengerID, bookingID)
- booking_flight(bookingID, flightID)

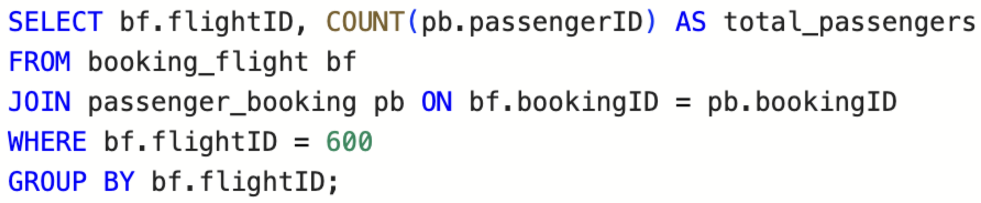
Transaction 3:

- payment(paymentID, amount, paymentStatus)

SQL Commands and Table Specifications

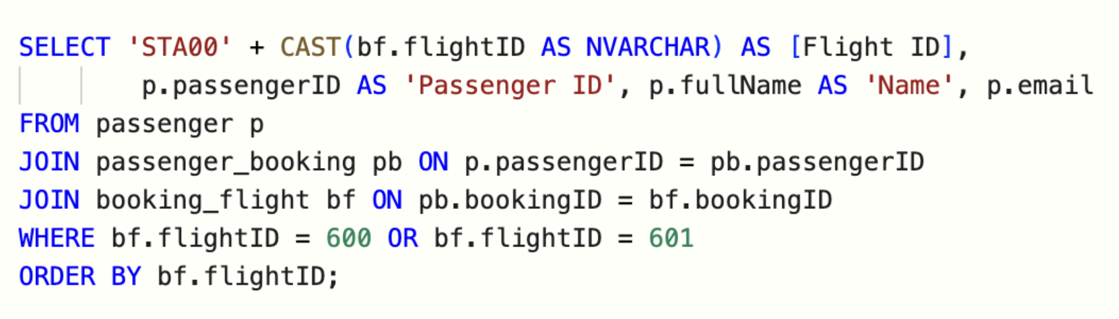
SQL Commands for Selected Transactions/Queries

Transaction 1:



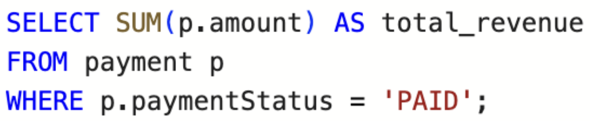
```sql
SELECT bf.flightID, COUNT(pb.passengerID) AS total_passengers
FROM booking_flight bf
JOIN passenger_booking pb ON bf.bookingID = pb.bookingID
WHERE bf.flightID = 600
GROUP BY bf.flightID;
```

Transaction 2:



```sql
SELECT 'STA00' + CAST(bf.flightID AS NVARCHAR) AS [Flight ID],
        p.passengerID AS 'Passenger ID', p.fullName AS 'Name', p.email
FROM passenger p
JOIN passenger_booking pb ON p.passengerID = pb.passengerID
JOIN booking_flight bf ON pb.bookingID = bf.bookingID
WHERE bf.flightID = 600 OR bf.flightID = 601
ORDER BY bf.flightID;
```

Transaction 3:



```sql
SELECT SUM(p.amount) AS total_revenue
FROM payment p
WHERE p.paymentStatus = 'PAID';
```

Table Specifications (Number of Records, Record Sizes)

Transaction 1:

Table	No. of Records	Est. Record Size	Primary Key(s)
passenger	800	~120 bytes	passengerID
booking	60	~4 bytes	bookingrID
flight	251	~220 bytes	flightID
passenger_booking	155	~8 bytes	(passengerID, bookingID)
booking_flight	60	~8 bytes	(bookingID, flightID)

Transaction 2:

Table	No. of Records	Est. Record Size	Primary Key(s)
passenger	800	~120 bytes	passengerID
booking	60	~4 bytes	bookingrID
flight	251	~220 bytes	flightID
passenger_booking	155	~8 bytes	(passengerID, bookingID)
booking_flight	60	~8 bytes	(bookingID, flightID)

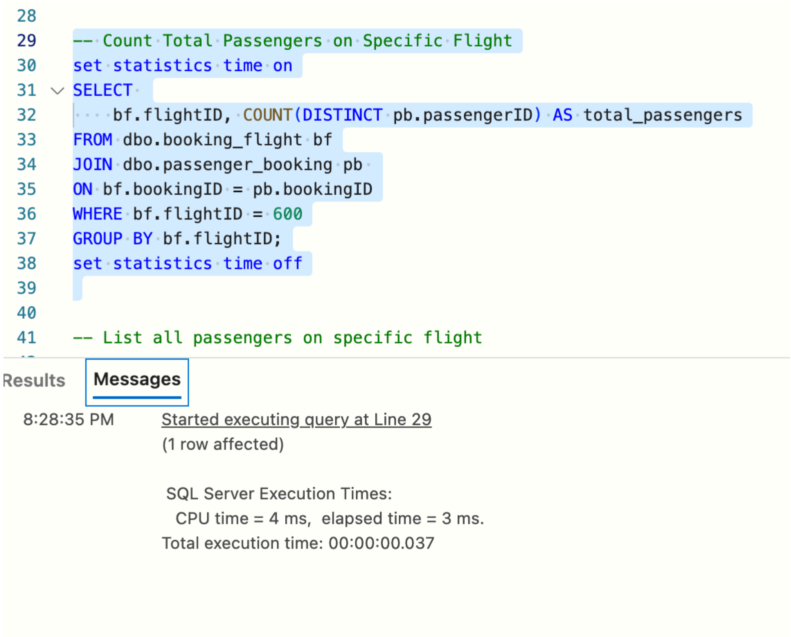
Transaction 3:

Table	No. of Records	Est. Record Size	Primary Key(s)
payment	60	~8 bytes	paymentID

Pre-Improvement Transaction Analysis

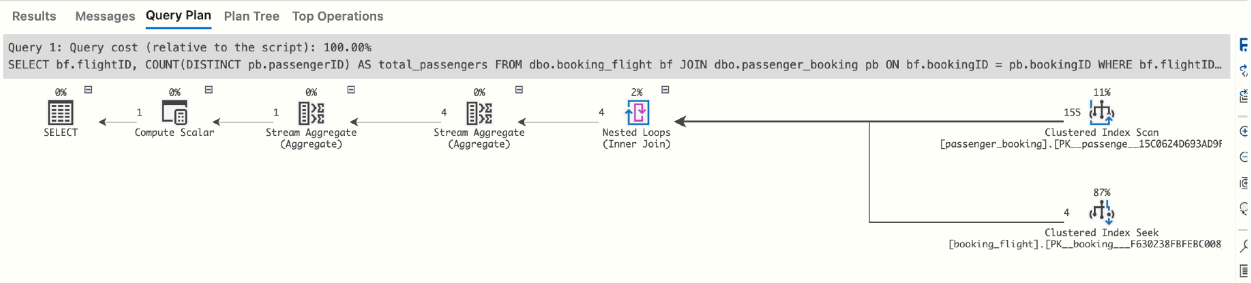
Performance Results from RDBMS Query Analyzer

Transaction 1: Count Total Passengers on a specific flight

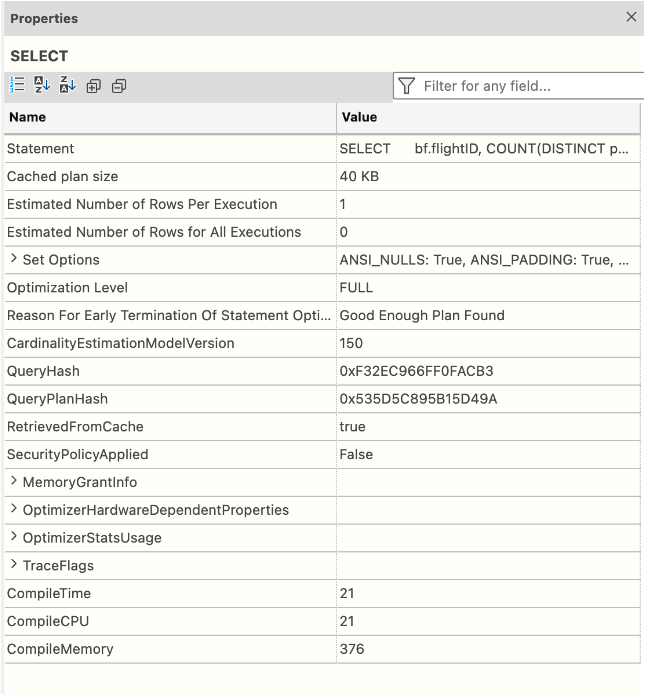


```sql

-- Count Total Passengers on Specific Flight
set statistics time on
SELECT
    bf.flightID, COUNT(DISTINCT pb.passengerID) AS total_passengers
FROM dbo.booking_flight bf
JOIN dbo.passenger_booking pb
ON bf.bookingID = pb.bookingID
WHERE bf.flightID = 600
GROUP BY bf.flightID;
set statistics time off


-- List all passengers on specific flight
```

```
8:28:35 PM          Started executing query at Line 29
                    (1 row affected)

                     SQL Server Execution Times:
                       CPU time = 4 ms,  elapsed time = 3 ms.
                    Total execution time: 00:00:00.037
```



```
Query 1: Query cost (relative to the script): 100.00%
SELECT bf.flightID, COUNT(DISTINCT pb.passengerID) AS total_passengers FROM dbo.booking_flight bf JOIN dbo.passenger_booking pb ON bf.bookingID = pb.bookingID WHERE bf.flightID…
```



Name	Value
Statement	SELECT bf.flightID, COUNT(DISTINCT p...
Cached plan size	40 KB
Estimated Number of Rows Per Execution	1
Estimated Number of Rows for All Executions	0
› Set Options	ANSI_NULLS: True, ANSI_PADDING: True, ...
Optimization Level	FULL
Reason For Early Termination Of Statement Opti...	Good Enough Plan Found
CardinalityEstimationModelVersion	150
QueryHash	0xF32EC966FF0FACB3
QueryPlanHash	0x535D5C895B15D49A
RetrievedFromCache	true
SecurityPolicyApplied	False
› MemoryGrantInfo	
› OptimizerHardwareDependentProperties	
› OptimizerStatsUsage	
› TraceFlags	
CompileTime	21
CompileCPU	21
CompileMemory	376

These pictures represent the result from the current version in which index structure has not been implemented yet. As can be seen in the Query Plan, It still uses the clustered index scan. Execution time: CPU Time = 4ms and elapsed time = 3 msp; Result in Total execution time: 00:00:00:037

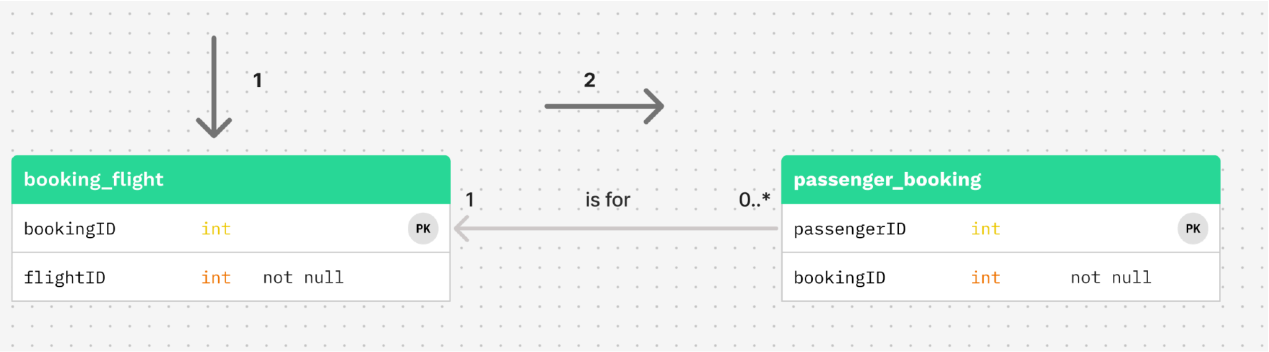
Transaction Analysis Form for Transaction 1		
		April 28, 2025
Transaction	Count Total Passengers on Specific Flight	
Volume	Average	Depends on how many flight at that specific moment in time But best guess is around 50 per hour
	Peak	May be hundreds or more

```
SELECT
   bf.flightID,
COUNT(DISTINCT
pb.passengerID) AS
total_passengers
FROM dbo.booking_flight bf
JOIN dbo.passenger_booking pb
ON bf.bookingID =
pb.bookingID
WHERE bf.flightID = 600
GROUP BY bf.flightID;
```

Predicate	`bf.flightID = 600`

Join Attribute	`FROM dbo.booking_flight bf`
	`JOIN dbo.passenger_booking pb`
	`ON bf.bookingID = pb.bookingID`

Ordering Attribute	-
Grouping Attribute	`GROUP BY bf.flightID;`
Built-in functions	-
Attributes Updated	-

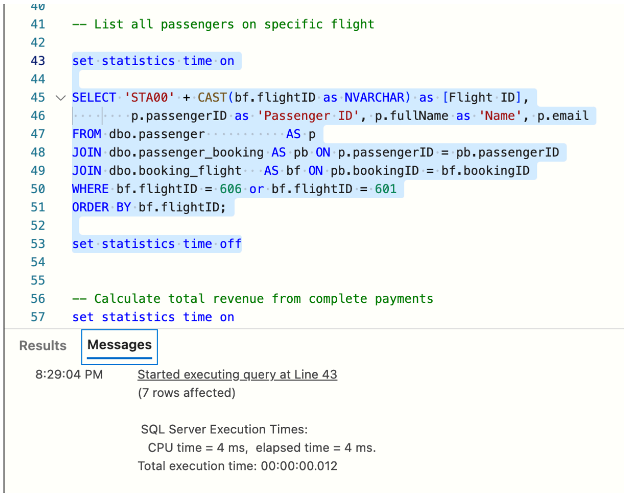


Access	Entity	Type of	No. of References		
			Per Transactions	Per Transaction	Peak Per Hour*
1	Booking_flight	Read	4	4	400 or more
2	passenger_booking	Read	155	155	155000 or more
Total References			159	159	155400 or more

The Number of references taken from the Query Plan number.

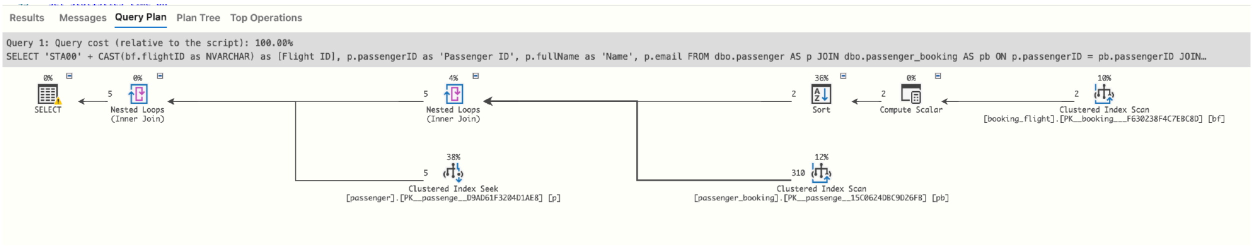
- For the Peak per hour the number per transaction multiplied with the volume.
  One big note is that it really depends on how big the tables are.

Transaction 2: List all passengers on specific flight

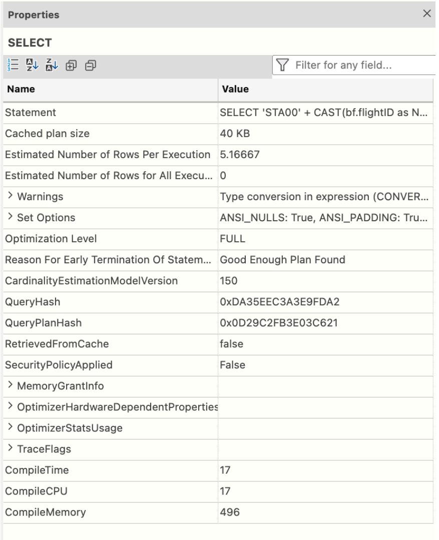


```sql
-- List all passengers on specific flight

set statistics time on

SELECT 'STA00' + CAST(bf.flightID as NVARCHAR) as [Flight ID],
       p.passengerID as 'Passenger ID', p.fullName as 'Name', p.email
FROM dbo.passenger          AS p
JOIN dbo.passenger_booking AS pb ON p.passengerID = pb.passengerID
JOIN dbo.booking_flight    AS bf ON pb.bookingID = bf.bookingID
WHERE bf.flightID = 606 or bf.flightID = 601
ORDER BY bf.flightID;

set statistics time off


-- Calculate total revenue from complete payments
set statistics time on
```

```
8:29:04 PM    Started executing query at Line 43
              (7 rows affected)

               SQL Server Execution Times:
                 CPU time = 4 ms,  elapsed time = 4 ms.
              Total execution time: 00:00:00.012
```



```
Query 1: Query cost (relative to the script): 100.00%
SELECT 'STA00' + CAST(bf.flightID as NVARCHAR) as [Flight ID], p.passengerID as 'Passenger ID', p.fullName as 'Name', p.email FROM dbo.passenger AS p JOIN dbo.passenger_booking AS pb ON p.passengerID = pb.passengerID JOIN…
```



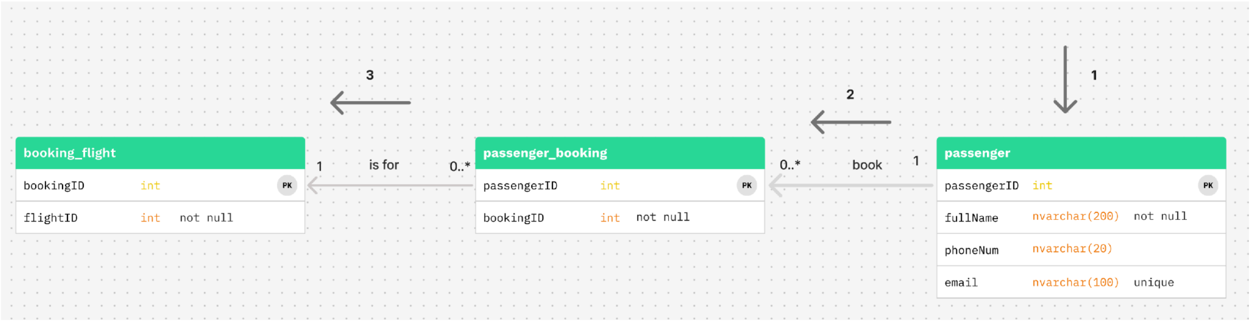
Name	Value
Statement	SELECT 'STA00' + CAST(bf.flightID as N...
Cached plan size	40 KB
Estimated Number of Rows Per Execution	5.16667
Estimated Number of Rows for All Execu...	0
› Warnings	Type conversion in expression (CONVER...
› Set Options	ANSI_NULLS: True, ANSI_PADDING: Tru...
Optimization Level	FULL
Reason For Early Termination Of Statem...	Good Enough Plan Found
CardinalityEstimationModelVersion	150
QueryHash	0xDA35EEC3A3E9FDA2
QueryPlanHash	0x0D29C2FB3E03C621
RetrievedFromCache	false
SecurityPolicyApplied	False
› MemoryGrantInfo	
› OptimizerHardwareDependentProperties	
› OptimizerStatsUsage	
› TraceFlags	
CompileTime	17
CompileCPU	17
CompileMemory	496

These pictures represent the result from the current version in which index structure has not been implemented yet. As can be seen in the Query Plan, It still uses the clustered index scan. As can be seen in the Query Plan, It still uses the clustered index scan. Execution time: CPU Time = 4ms and elapsed time = 4 ms; Result in Total execution time: 00:00:00:012

Transaction Analysis Form for Transaction 2		
		April 28, 2025
Transaction	List all passengers on specific flight	
Volume	Average	~ 100 queries at specific point in time
	Peak	~ 200 or more at specific point in time

SELECT 'STA00' + CAST(bf.flightID as NVARCHAR) as [Flight ID],
p.passengerID as 'Passenger ID', p.fullName as 'Name', p.email
FROM dbo.passenger AS p
JOIN dbo.passenger_booking AS pb ON p.passengerID = pb.passengerID
JOIN dbo.booking_flight AS bf ON pb.bookingID = bf.bookingID
WHERE bf.flightID = 606 or bf.flightID = 601
ORDER BY bf.flightID;
	Predicate	bf.flightID = 606 or bf.flightID = 601

	Join Attribute	FROM dbo.passenger AS p JOIN dbo.passenger_booking AS pb ON p.passengerID = pb.passengerID JOIN dbo.booking_flight AS bf ON pb.bookingID = bf.bookingID
	Ordering Attribute	bf.flightID
	Grouping Attribute	-
	Built-in functions	CAST(bf.flightID as NVARCHAR) as [Flight ID]
	Attributes Updated	-

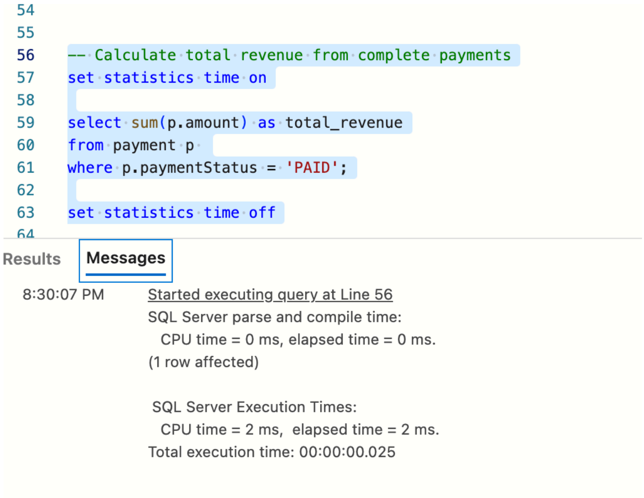


	Entity	Type of	No. of References

Access			Per Transactions	Per Transaction	Peak Per Hour*
1	passenger	Read	2	2	400 or more
2	passenger _booking	Read	310	310	62000 or more
3	Booking _flight	Read	7	7	1400 or more
Total References			319	319	63800 or more

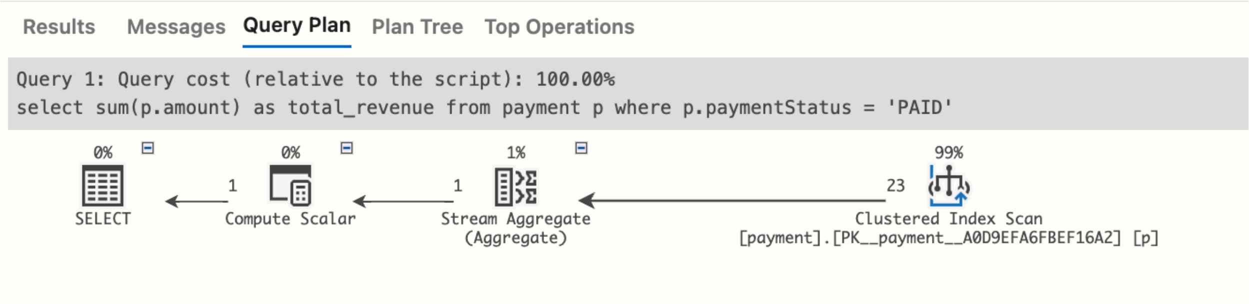
The Number of references taken from the Query Plan number.
For the Peak per hour, the number per transaction multiply with the volume.
One big note is that it really depends on how big tables are.

Transaction 3: Calculate the total revenue from complete payments

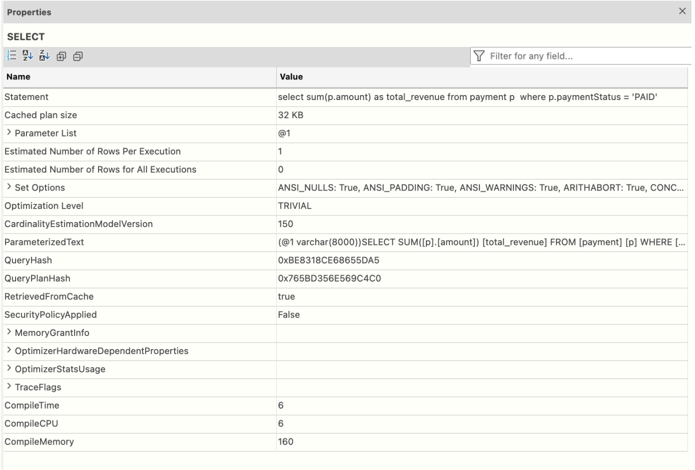


```
-- Calculate total revenue from complete payments
set statistics time on

select sum(p.amount) as total_revenue
from payment p
where p.paymentStatus = 'PAID';

set statistics time off
```

```
8:30:07 PM        Started executing query at Line 56
                  SQL Server parse and compile time:
                     CPU time = 0 ms, elapsed time = 0 ms.
                  (1 row affected)

                   SQL Server Execution Times:
                     CPU time = 2 ms,  elapsed time = 2 ms.
                  Total execution time: 00:00:00.025
```

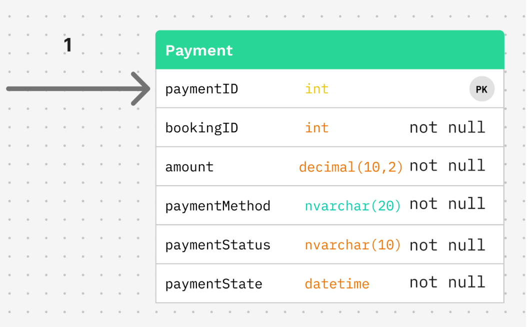


```
Query 1: Query cost (relative to the script): 100.00%
select sum(p.amount) as total_revenue from payment p where p.paymentStatus = 'PAID'
```

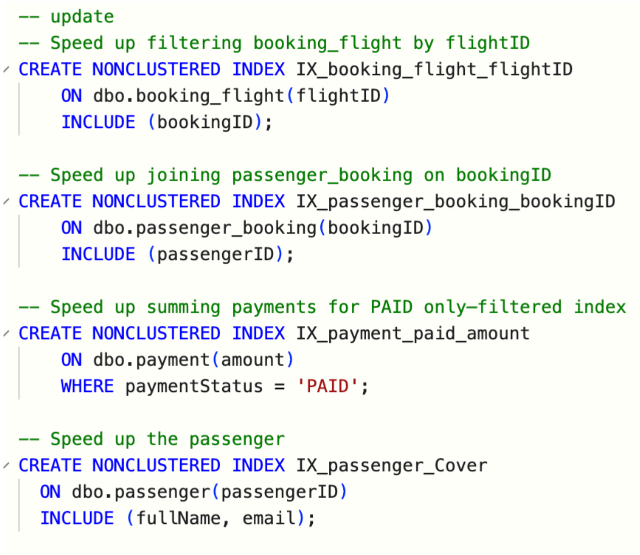


Name	Value
Statement	select sum(p.amount) as total_revenue from payment p where p.paymentStatus = 'PAID'
Cached plan size	32 KB
› Parameter List	@1
Estimated Number of Rows Per Execution	1
Estimated Number of Rows for All Executions	0
› Set Options	ANSI_NULLS: True, ANSI_PADDING: True, ANSI_WARNINGS: True, ARITHABORT: True, CONC...
Optimization Level	TRIVIAL
CardinalityEstimationModelVersion	150
ParameterizedText	(@1 varchar(8000))SELECT SUM([p].[amount]) [total_revenue] FROM [payment] [p] WHERE [...
QueryHash	0xBE8318CE68655DA5
QueryPlanHash	0x765BD356E569C4C0
RetrievedFromCache	true
SecurityPolicyApplied	False
› MemoryGrantInfo	
› OptimizerHardwareDependentProperties	
› OptimizerStatsUsage	
› TraceFlags	
CompileTime	6
CompileCPU	6
CompileMemory	160

These pictures represent the result from the current version, in which the index structure has not yet been implemented. As seen in the Query Plan, the clustered index scan is still used. As seen in the Query Plan, It still uses the clustered index scan. SQL Server Execution time:
CPU Time = 2ms and elapsed time = 2 ms; Result in Total execution time: 00:00:00:025

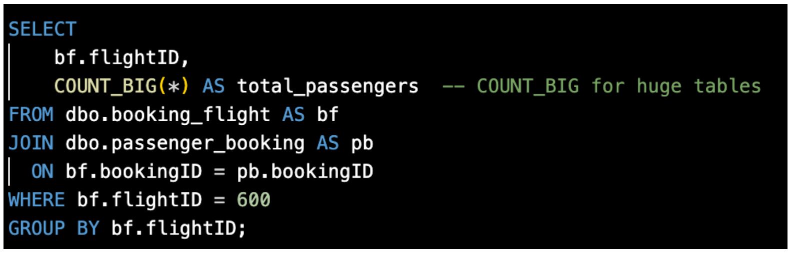
<table>
<tr><td colspan="6">Transaction Analysis Form for Transaction 3</td></tr>
<tr><td colspan="6">April 28, 2025</td></tr>
<tr><td colspan="2">Transaction</td><td colspan="4">Calculate total revenue from completed payment</td></tr>
<tr><td colspan="2" rowspan="2">Volume</td><td>Average</td><td colspan="3">5 from time to time because this SQL script is not commonly used a lot.</td></tr>
<tr><td>Peak</td><td colspan="3">At most 10</td></tr>
<tr><td colspan="3" rowspan="6">select sum(p.amount) as total_revenue
from payment p
where p.paymentStatus = 'PAID';</td><td>Predicate</td><td colspan="2">p.paymentStatus = 'PAID'</td></tr>
<tr><td>Join Attribute</td><td colspan="2">-</td></tr>
<tr><td>Ordering Attribute</td><td colspan="2">-</td></tr>
<tr><td>Grouping Attribute</td><td colspan="2">-</td></tr>
<tr><td>Built-in functions</td><td colspan="2">sum(p.amount)</td></tr>
<tr><td>Attributes Updated</td><td colspan="2">-</td></tr>
<tr><td colspan="6">
1
Payment
paymentID int PK
bookingID int not null
amount decimal(10,2) not null
paymentMethod nvarchar(20) not null
paymentStatus nvarchar(10) not null
paymentState datetime not null
It’s just 1 table.</td></tr>
<tr><td rowspan="2">Access</td><td rowspan="2">Entity</td><td rowspan="2">Type of</td><td colspan="3">No. of References</td></tr>
<tr><td>Per Transactions</td><td>Per Transaction</td><td>Peak Per Hour</td></tr>
<tr><td>1</td><td>payment</td><td>Read</td><td>23</td><td>23</td><td>23*10= 230</td></tr>
<tr><td colspan="3">Total References</td><td>23</td><td>23</td><td>230</td></tr>
</table>

Post-Improvement Transaction Analysis



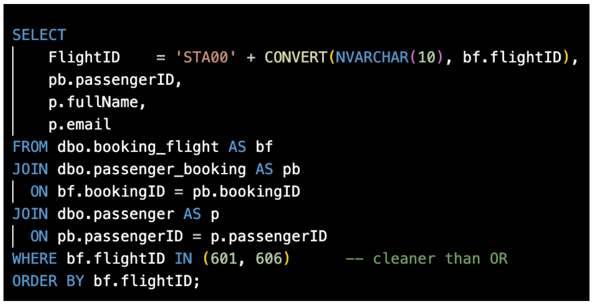
```sql
-- update
-- Speed up filtering booking_flight by flightID
CREATE NONCLUSTERED INDEX IX_booking_flight_flightID
    ON dbo.booking_flight(flightID)
    INCLUDE (bookingID);

-- Speed up joining passenger_booking on bookingID
CREATE NONCLUSTERED INDEX IX_passenger_booking_bookingID
    ON dbo.passenger_booking(bookingID)
    INCLUDE (passengerID);

-- Speed up summing payments for PAID only—filtered index
CREATE NONCLUSTERED INDEX IX_payment_paid_amount
    ON dbo.payment(amount)
    WHERE paymentStatus = 'PAID';

-- Speed up the passenger
CREATE NONCLUSTERED INDEX IX_passenger_Cover
  ON dbo.passenger(passengerID)
  INCLUDE (fullName, email);
```

We Improved by transforming the current clustered index (ordered) of each transaction as have been seen in the query plan in the pre improvement section into nonclustered index which would allow the transaction to be speed up with the help of pointer from the non clustered index, we also add the nonclustered index into the passenger to faster passenger table up. Additionally, we updated the sql to allow even faster response time.



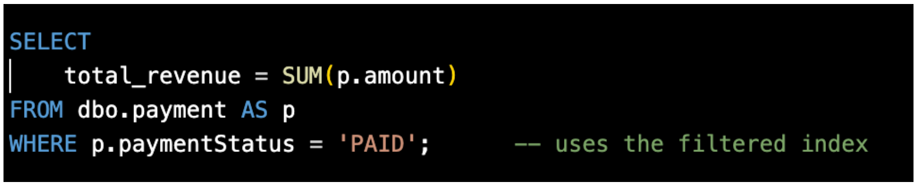
```sql
SELECT
    bf.flightID,
    COUNT_BIG(*) AS total_passengers  -- COUNT_BIG for huge tables
FROM dbo.booking_flight AS bf
JOIN dbo.passenger_booking AS pb
  ON bf.bookingID = pb.bookingID
WHERE bf.flightID = 600
GROUP BY bf.flightID;
```

To handle if transaction getting larger by using COUNT_BIG(*)



```sql
SELECT
    FlightID    = 'STA00' + CONVERT(NVARCHAR(10), bf.flightID),
    pb.passengerID,
    p.fullName,
    p.email
FROM dbo.booking_flight AS bf
JOIN dbo.passenger_booking AS pb
  ON bf.bookingID = pb.bookingID
JOIN dbo.passenger AS p
  ON pb.passengerID = p.passengerID
WHERE bf.flightID IN (601, 606)      -- cleaner than OR
ORDER BY bf.flightID;
```

Change the OR to set in (...) because it is cleaner.

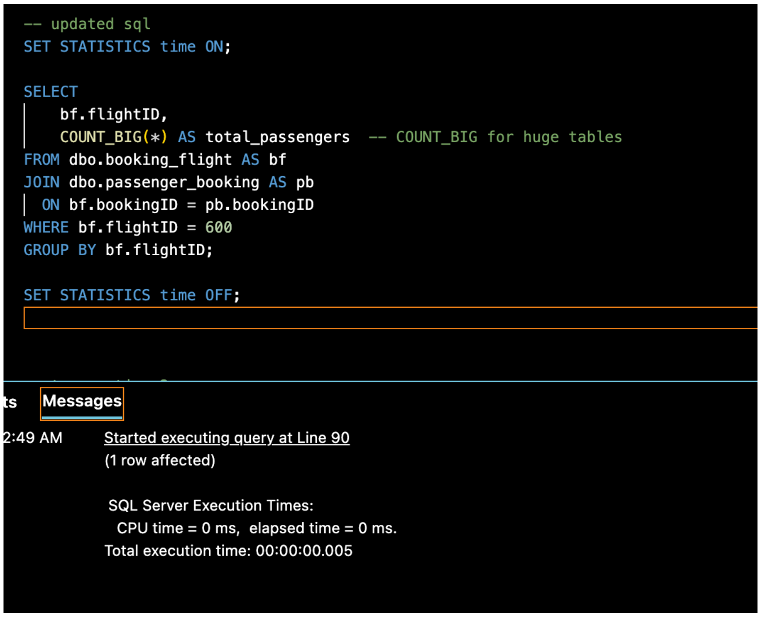


```
SELECT
    total_revenue = SUM(p.amount)
FROM dbo.payment AS p
WHERE p.paymentStatus = 'PAID';      -- uses the filtered index
```

Use the newly implemented index.

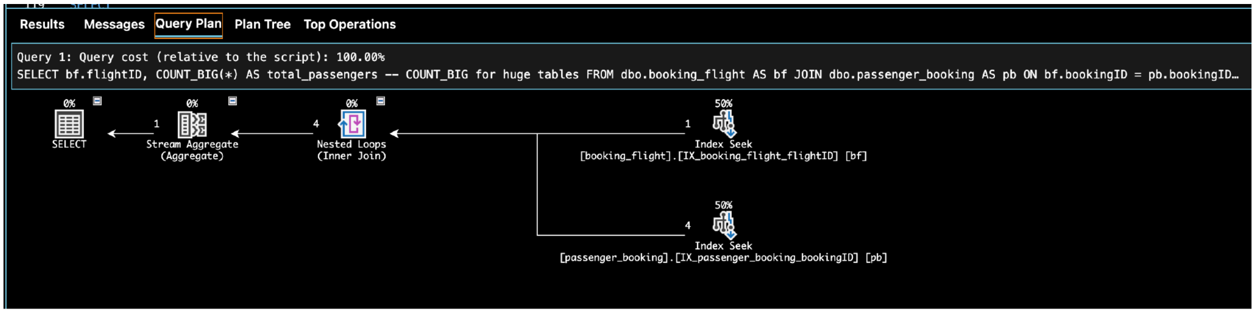
Performance Results after Optimization

Transaction 1: Count Total Passengers on a specific flight



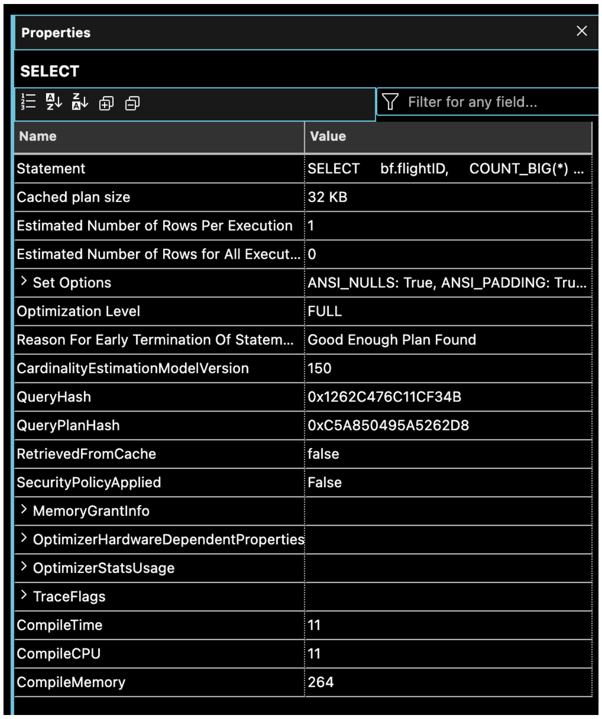
```
-- updated sql
SET STATISTICS time ON;

SELECT
    bf.flightID,
    COUNT_BIG(*) AS total_passengers  -- COUNT_BIG for huge tables
FROM dbo.booking_flight AS bf
JOIN dbo.passenger_booking AS pb
  ON bf.bookingID = pb.bookingID
WHERE bf.flightID = 600
GROUP BY bf.flightID;

SET STATISTICS time OFF;
```

```
2:49 AM     Started executing query at Line 90
            (1 row affected)

             SQL Server Execution Times:
               CPU time = 0 ms,  elapsed time = 0 ms.
            Total execution time: 00:00:00.005
```



```
Query 1: Query cost (relative to the script): 100.00%
SELECT bf.flightID, COUNT_BIG(*) AS total_passengers -- COUNT_BIG for huge tables FROM dbo.booking_flight AS bf JOIN dbo.passenger_booking AS pb ON bf.bookingID = pb.bookingID…
```

0%
SELECT
1
0%
Stream Aggregate
(Aggregate)
4
0%
Nested Loops
(Inner Join)
1
50%
Index Seek
[booking_flight].[IX_booking_flight_flightID] [bf]
4
50%
Index Seek
[passenger_booking].[IX_passenger_booking_bookingID] [pb]



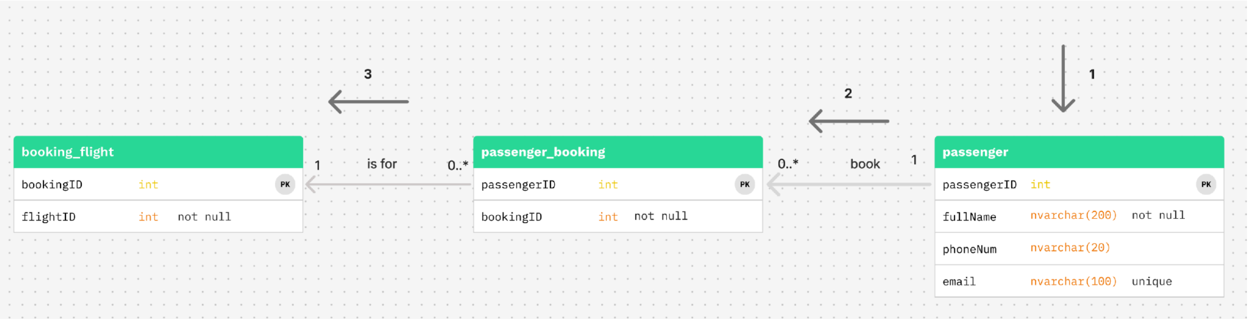
Name	Value
Statement	SELECT bf.flightID, COUNT_BIG(*) ...
Cached plan size	32 KB
Estimated Number of Rows Per Execution	1
Estimated Number of Rows for All Execut...	0
› Set Options	ANSI_NULLS: True, ANSI_PADDING: Tru...
Optimization Level	FULL
Reason For Early Termination Of Statem...	Good Enough Plan Found
CardinalityEstimationModelVersion	150
QueryHash	0x1262C476C11CF34B
QueryPlanHash	0xC5A850495A5262D8
RetrievedFromCache	false
SecurityPolicyApplied	False
› MemoryGrantInfo	
› OptimizerHardwareDependentProperties	
› OptimizerStatsUsage	
› TraceFlags	
CompileTime	11
CompileCPU	11
CompileMemory	264

From the picture above, the query plan is noticeably shorter than the original one.

Transaction Analysis Form for Transaction 1		
		April 28, 2025
Transaction	Count Total Passengers on Specific Flight	
Volume	Average	Depends on how many flight at that specific moment in time But best guess is around 50 per hour
	Peak	May be hundreds or more

SELECT
   bf.flightID,
   COUNT_BIG(*) AS total_passengers -- COUNT_BIG for huge tables
FROM dbo.booking_flight AS bf
JOIN dbo.passenger_booking AS pb
 ON bf.bookingID = pb.bookingID
WHERE bf.flightID = 600
GROUP BY bf.flightID;
	Predicate	bf.flightID = 600

	Join Attribute	FROM dbo.booking_flight AS bf JOIN dbo.passenger_booking AS pb ON bf.bookingID = pb.bookingID
	Ordering Attribute	-
	Grouping Attribute	GROUP BY bf.flightID;
	Built-in functions	-
	Attributes Updated	-

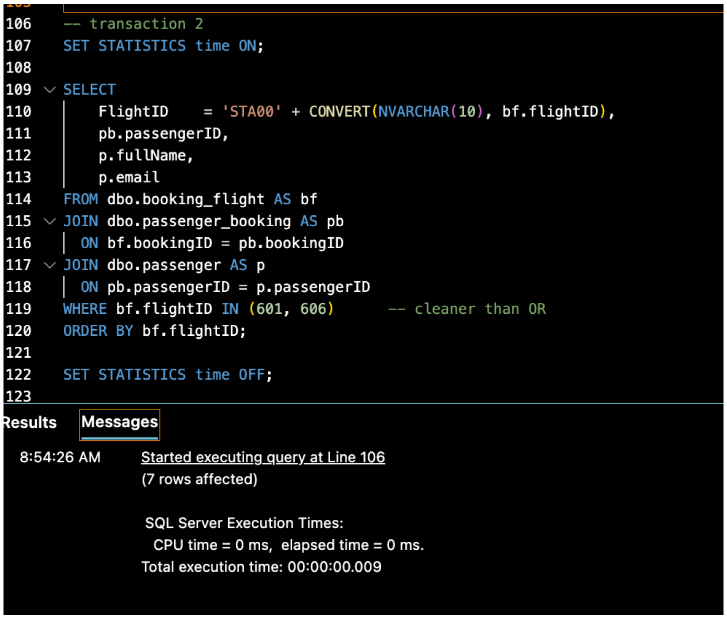


Access	Entity	Type of	No. of References		
			Per Transactions	Per Transaction	Peak Per Hour
1	Booking _flight	Read	1	1	1*100 = 100 or more
2	passenger _booking	Read	4	4	4*100 = 400 or more

Total References	5	5	500 or more

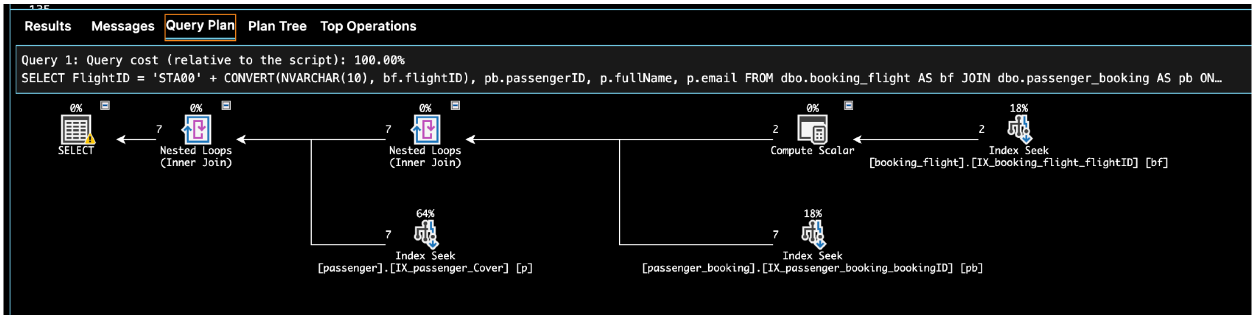
The Number of references taken from the Query Plan number.
For the Peak per hour the number per transaction multiply with the volume.
One big note is that it really depends on how big tables are.

Transaction 2: List all passengers on specific flight



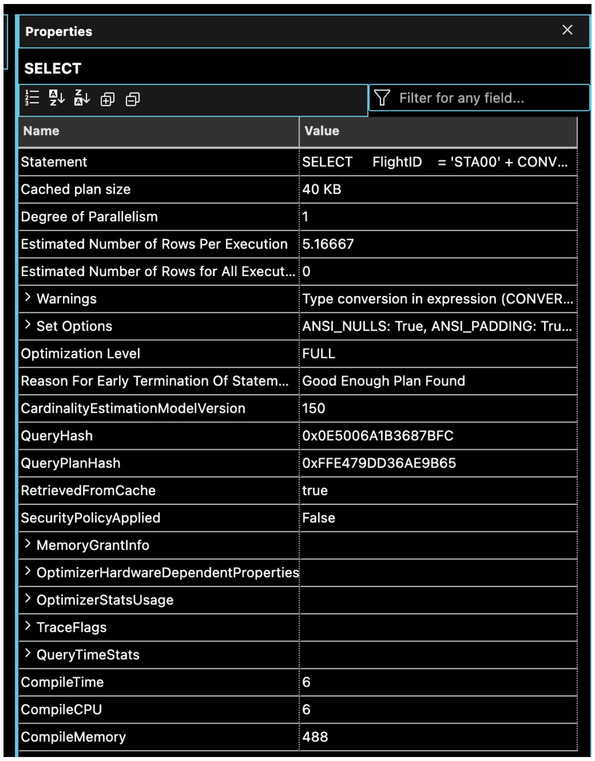
```
106    -- transaction 2
107    SET STATISTICS time ON;
108
109  ∨ SELECT
110        FlightID    = 'STA00' + CONVERT(NVARCHAR(10), bf.flightID),
111        pb.passengerID,
112        p.fullName,
113        p.email
114    FROM dbo.booking_flight AS bf
115  ∨ JOIN dbo.passenger_booking AS pb
116      ON bf.bookingID = pb.bookingID
117  ∨ JOIN dbo.passenger AS p
118      ON pb.passengerID = p.passengerID
119    WHERE bf.flightID IN (601, 606)      -- cleaner than OR
120    ORDER BY bf.flightID;
121
122    SET STATISTICS time OFF;
123
```

```
8:54:26 AM        Started executing query at Line 106
                  (7 rows affected)

                   SQL Server Execution Times:
                     CPU time = 0 ms,  elapsed time = 0 ms.
                  Total execution time: 00:00:00.009
```



```
Query 1: Query cost (relative to the script): 100.00%
SELECT FlightID = 'STA00' + CONVERT(NVARCHAR(10), bf.flightID), pb.passengerID, p.fullName, p.email FROM dbo.booking_flight AS bf JOIN dbo.passenger_booking AS pb ON…
```

0%
SELECT
7
0%
Nested Loops
(Inner Join)
7
0%
Nested Loops
(Inner Join)
2
0%
Compute Scalar
2
18%
Index Seek
[booking_flight].[IX_booking_flight_flightID] [bf]
7
64%
Index Seek
[passenger].[IX_passenger_Cover] [p]
7
18%
Index Seek
[passenger_booking].[IX_passenger_booking_bookingID] [pb]



Name	Value
Statement	SELECT FlightID = 'STA00' + CONV...
Cached plan size	40 KB
Degree of Parallelism	1
Estimated Number of Rows Per Execution	5.16667
Estimated Number of Rows for All Execut...	0
› Warnings	Type conversion in expression (CONVER...
› Set Options	ANSI_NULLS: True, ANSI_PADDING: Tru...
Optimization Level	FULL
Reason For Early Termination Of Statem...	Good Enough Plan Found
CardinalityEstimationModelVersion	150
QueryHash	0x0E5006A1B3687BFC
QueryPlanHash	0xFFE479DD36AE9B65
RetrievedFromCache	true
SecurityPolicyApplied	False
› MemoryGrantInfo	
› OptimizerHardwareDependentProperties	
› OptimizerStatsUsage	
› TraceFlags	
› QueryTimeStats	
CompileTime	6
CompileCPU	6
CompileMemory	488

From the picture above, the query plan is slightly shorter than the pre improved one. This version cuts the sort out and transforms the clustered index into the non-clustered index which allows the execution time to be faster than the pre-improve one.

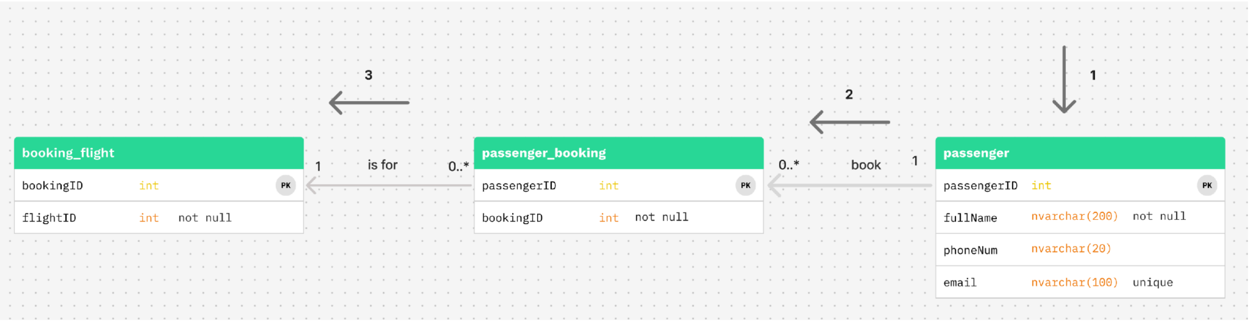
Transaction Analysis Form for Transaction 2		
		April 28, 2025
Transaction	List all passengers on specific flight	
Volume	Average	~ 100 queries at specific point in time
	Peak	~ 200 or more at specific point in time

```
SELECT
   FlightID    = 'STA00' +
CONVERT(NVARCHAR(10),
bf.flightID),
   pb.passengerID,
   p.fullName,
   p.email
FROM dbo.booking_flight AS bf
JOIN dbo.passenger_booking AS
pb
 ON bf.bookingID =
pb.bookingID
JOIN dbo.passenger AS p
 ON pb.passengerID =
p.passengerID
WHERE bf.flightID IN (601,
606)      -- cleaner than OR
ORDER BY bf.flightID;
```

Predicate	`bf.flightID IN (601, 606)`

Join Attribute	`FROM dbo.booking_flight AS bf`
	`JOIN dbo.passenger_booking AS pb`
	` ON bf.bookingID = pb.bookingID`
	`JOIN dbo.passenger AS p`
	` ON pb.passengerID = p.passengerID`

Ordering Attribute	`bf.flightID`
Grouping Attribute	-
Built-in functions	`FlightID    = 'STA00' + CONVERT(NVARCHAR(10),`
Attributes Updated	-

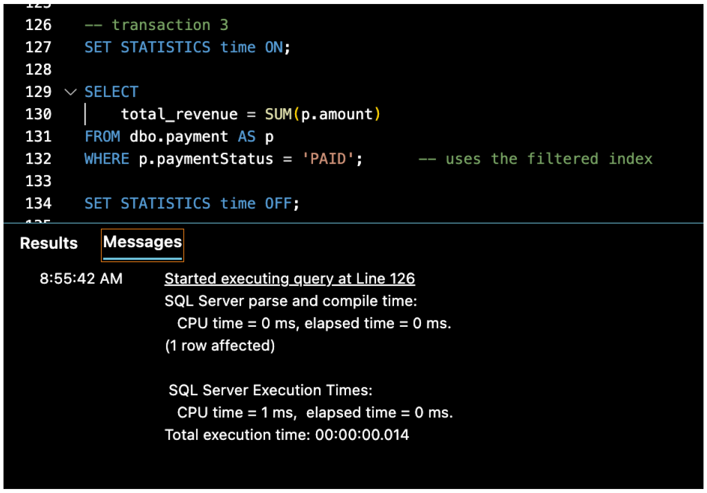


Access	Entity	Type of	No. of References		
			Per Transactions	Per Transaction	Peak Per Hour
1	passenger	Read	2	2	400 or more

2	passenger_booking	Read	7	7	1400 or more
3	Booking_flight	Read	7	7	1400 or more
Total References			16	16	3200 or more

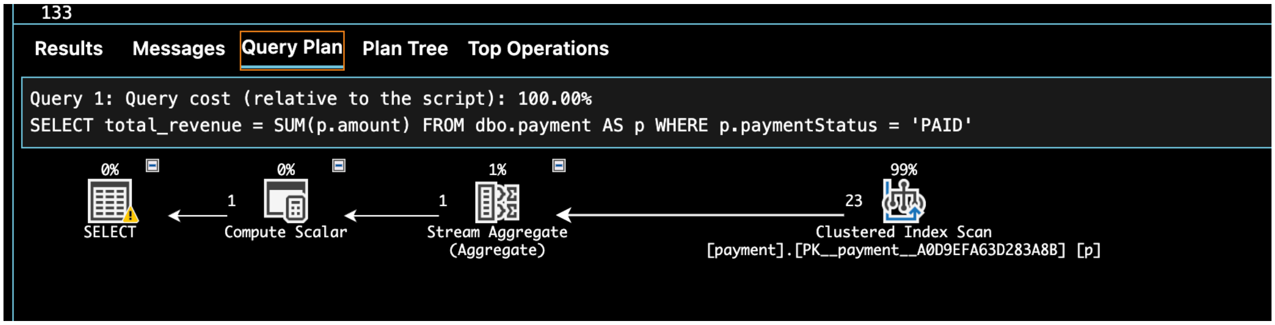
The Number of references taken from the Query Plan number.
For the Peak per hour the number per transaction multiply with the volume.
One big note is that it really depends on how big tables are.

Transaction 3: Calculate the total revenue from complete payments

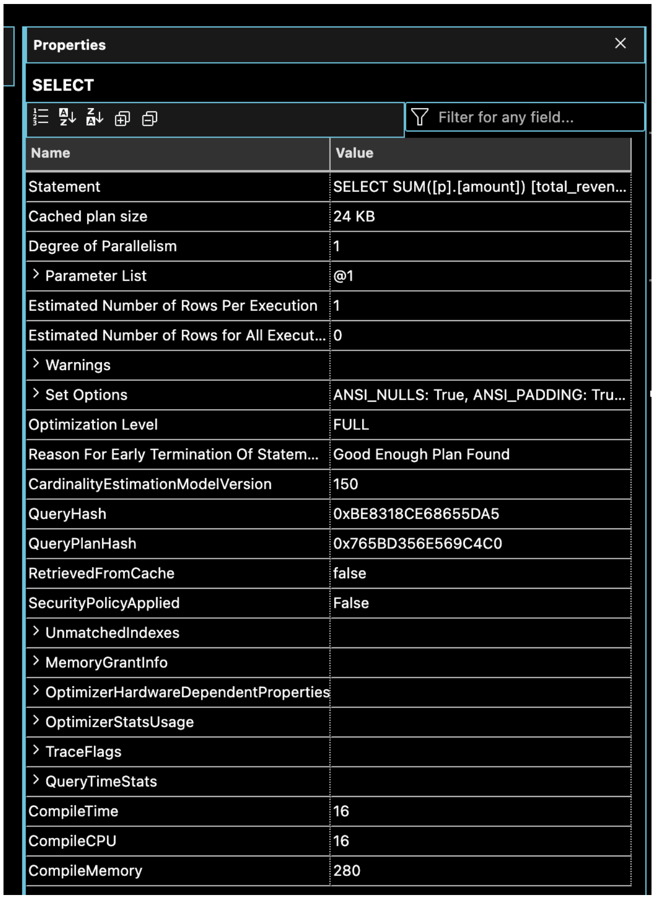


```
-- transaction 3
SET STATISTICS time ON;

SELECT
    total_revenue = SUM(p.amount)
FROM dbo.payment AS p
WHERE p.paymentStatus = 'PAID';      -- uses the filtered index

SET STATISTICS time OFF;
```

```
8:55:42 AM      Started executing query at Line 126
                SQL Server parse and compile time:
                   CPU time = 0 ms, elapsed time = 0 ms.
                (1 row affected)

                 SQL Server Execution Times:
                   CPU time = 1 ms,  elapsed time = 0 ms.
                Total execution time: 00:00:00.014
```

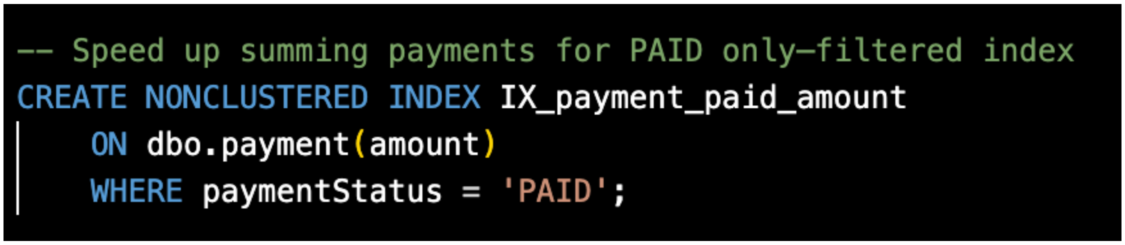


```
Query 1: Query cost (relative to the script): 100.00%
SELECT total_revenue = SUM(p.amount) FROM dbo.payment AS p WHERE p.paymentStatus = 'PAID'
```

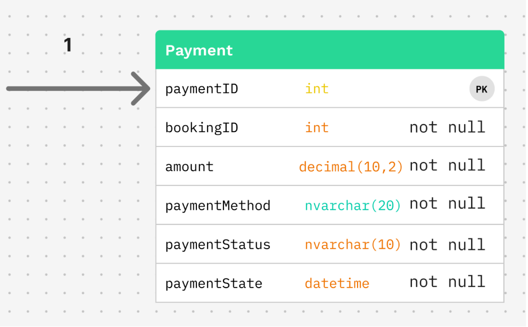


Name	Value
Statement	SELECT SUM([p].[amount]) [total_reven...
Cached plan size	24 KB
Degree of Parallelism	1
› Parameter List	@1
Estimated Number of Rows Per Execution	1
Estimated Number of Rows for All Execut...	0
› Warnings	
› Set Options	ANSI_NULLS: True, ANSI_PADDING: Tru...
Optimization Level	FULL
Reason For Early Termination Of Statem...	Good Enough Plan Found
CardinalityEstimationModelVersion	150
QueryHash	0xBE8318CE68655DA5
QueryPlanHash	0x765BD356E569C4C0
RetrievedFromCache	false
SecurityPolicyApplied	False
› UnmatchedIndexes	
› MemoryGrantInfo	
› OptimizerHardwareDependentProperties	
› OptimizerStatsUsage	
› TraceFlags	
› QueryTimeStats	
CompileTime	16
CompileCPU	16
CompileMemory	280

From the picture above, the query plan is similar to the pre improvement but the difference is the index has been implemented to the filter level, which allows the search to be faster, and reduces the execution time.



```sql
-- Speed up summing payments for PAID only—filtered index
CREATE NONCLUSTERED INDEX IX_payment_paid_amount
    ON dbo.payment(amount)
    WHERE paymentStatus = 'PAID';
```

<table>
<tr><td colspan="7">Transaction Analysis Form for Transaction 3 <b>After improvement</b></td></tr>
<tr><td colspan="7">April 28, 2025</td></tr>
<tr><td colspan="2">Transaction</td><td colspan="5">Calculate total revenue from completed payment</td></tr>
<tr><td colspan="2" rowspan="2">Volume</td><td>Average</td><td colspan="4">5 from time to time because this SQL script is not commonly used a lot.</td></tr>
<tr><td>Peak</td><td colspan="4">10 or more</td></tr>
<tr><td colspan="3" rowspan="6">SELECT
total_revenue = SUM(p.amount)
FROM dbo.payment AS p
WHERE p.paymentStatus = 'PAID';   -- uses the filtered index</td><td>Predicate</td><td colspan="3">total_revenue = SUM(p.amount),
p.paymentStatus = 'PAID';</td></tr>
<tr><td>Join Attribute</td><td colspan="3">-</td></tr>
<tr><td>Ordering Attribute</td><td colspan="3">-</td></tr>
<tr><td>Grouping Attribute</td><td colspan="3">-</td></tr>
<tr><td>Built-in functions</td><td colspan="3">sum(p.amount)</td></tr>
<tr><td>Attributes Updated</td><td colspan="3">-</td></tr>
<tr><td colspan="7">
1
Payment
paymentID int PK
bookingID int not null
amount decimal(10,2) not null
paymentMethod nvarchar(20) not null
paymentStatus nvarchar(10) not null
paymentState datetime not null
It’s just 1 table.</td></tr>
<tr><td rowspan="2">Access</td><td rowspan="2">Entity</td><td rowspan="2">Type of</td><td colspan="4">No. of References</td></tr>
<tr><td colspan="2">Per Transactions</td><td>Per Transaction</td><td>Peak Per Hour</td></tr>
<tr><td>1</td><td>payment</td><td>Read</td><td colspan="2">23</td><td>23</td><td>230 or more</td></tr>
<tr><td colspan="3">Total References</td><td colspan="2">23</td><td>23</td><td>230 or more</td></tr>
</table>

Comparison with Pre-Improvement Results

Type / Time	Pre-Improvement			Post-Improvement		
	Server Execution CPU / elapsed	**Compile Time/CPU**	**Total Execution**	**Server Execution CPU / elapsed**	**Compile Time/CPU**	**Total Execution**
Transaction 1	4ms/ 3ms	21ms/21ms	00:00:00.037	0ms/0ms	11ms/11ms	00:00:00.005
Transaction 2	4ms/4ms	17ms/17ms	00:00:00.012	0ms/0ms	5ms/5ms	00:00:00.009
Transaction 3	2ms/2ms	6ms/6ms	00:00:00.025	1ms/0ms	1ms/1ms	00:00:00.014

Conclusion

After implementing indexes, it does seem to be faster across all transactions and make look up and search from related tables faster. One big limitation is Data size. We have limited data size so the improvement that we made might not seem to be a big difference if we compare both pre-improvement and post-improvement together.

Discussion

The analysis of all three selected transactions counting passengers on a specific flight, listing passengers, and calculating total revenue from completed payments shows a marked improvement in performance after applying database tuning techniques. Initially, all transactions relied on clustered index scans, leading to suboptimal execution times. After optimization, we implemented non-clustered indexes on frequently filtered and joined columns (e.g., flightID, passengerID, paymentStatus). This significantly reduced the CPU and elapsed execution times.

Transaction 1 (Passenger Count) benefited from using COUNT_BIG(*) and optimized join paths, reducing execution time from 37ms to 5ms.

Transaction 2 (Passenger List) saw improvements by replacing OR with IN, avoiding unnecessary sorts, and leveraging the new indexes, reducing execution time from 12ms to 9ms.

Transaction 3 (Revenue Calculation) gained from a filtered index on paymentStatus, bringing the execution time down from 25ms to 14ms.

The improvement is consistent across all performance metrics: CPU time, elapsed time, and total execution time. This validates the effectiveness of the applied tuning strategies.

However, it is important to note that real-world performance can vary with larger data volumes. Future improvements may include partitioning, query refactoring, and materialized views for extremely high-traffic systems.

References

MikeRayMSFT, “Clustered and nonclustered indexes - SQL Server,” *Microsoft Learn*. https://learn.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver16