M-STAY PLATFORM

Name Subbulakshmi Natarajan

Student ID 34069178

Table of Contents

Introduction

M-Stay is a residential service that provides homestay and rental options in the Melbourne area to Monash employees and students. The firm now keeps and saves all the business transaction data (properties, hosts, listings, bookings, etc.) needed for the management's day-to-day operations in an operational database. M-Stay has decided to construct a data warehouse as their company expands to enhance their analysis and productivity. Following in the report we will be discussing various techniques to design, develop and quickly generate BI reports from a Data Warehouse.

Exploring the operational database

The first step before doing the data cleaning stage, we must first explore the operational database. The below screenshot shows that I have explored the operational database by using a simple command that is **Select * from MStay. <table name>.**

```sql
-- Retrieve all records from the REVIEW table
SELECT *
FROM MStay.REVIEW;

-- Retrieve all records from the BOOKING table
SELECT *
FROM MStay.BOOKING;

-- Retrieve all records from the GUEST table
SELECT *
FROM MStay.GUEST;

-- Retrieve all records from the LISTING table
SELECT *
FROM MStay.LISTING;
```

```sql
-- Retrieve all records from the HOST table
SELECT *
FROM MStay.HOST;

-- Retrieve all records from the HOST_VERIFICATION table
SELECT *
FROM MStay.HOST_VERIFICATION;

-- Retrieve all records from the CHANNEL table
SELECT *
FROM MStay.CHANNEL;

-- Retrieve all records from the LISTING_TYPE table
SELECT *
FROM MStay.LISTING_TYPE;

-- Retrieve all records from the PROPERTY table
SELECT *
FROM MStay.PROPERTY;

-- Retrieve all records from the PROPERTY_AMMENITY table
SELECT *
FROM MStay.PROPERTY_AMMENITY;

-- Retrieve all records from the AMENITY table
SELECT *
FROM MStay.AMENITY;
```

Data Cleaning

1. Null Values

In the review table there are null values that can be found in review_comments as seen below in the screenshot. This is a screenshot before updating the null values in the review table.

SELECT * FROM Cleaned_REVIEW W | Enter a SQL expression to filter results (use Ctrl+Space)

Grid

	123 REVIEW_ID	REVIEW_DATE	A-Z REVIEW_COMMENT	123 BOOKING_ID
1	128,829,335	2017-01-27 00:00:00.000	[NULL]	749
2	276,158,344	2018-06-13 00:00:00.000	[NULL]	4,417

After updating the null values in the review column. I have updated the column which has null values as “No comment”.

```sql
-- View records with NULL Review_Comment before updating
SELECT *
FROM Cleaned_REVIEW
WHERE Review_Comment IS NULL;

-- Set reviews without comments to 'No Comment'
UPDATE Cleaned_REVIEW
SET Review_Comment = 'No Comment'
WHERE Review_Comment IS NULL;
```

	123 REVIEW_ID	REVIEW_DATE	A-Z REVIEW_COMMENT	123 BOOKING_ID
1	128,829,335	2017-01-27 00:00:00.000	No Comment	749
2	276,158,344	2018-06-13 00:00:00.000	No Comment	4,417

2. Inconsistent Values

In the guest table, we have inconsistent values like guest names have weird characters. That is data redundancy when we create a schema, so it is better to remove those characters. This screenshot is before updating the weird characters in the guest table.

	123 GUEST_ID	A-Z GUEST_NAME
1	19,926,966	StÃ©phane
2	792,502	Maureen + Callum
3	4,625,694	ChloÃ©
4	294,860,714	ë?„ì—°
5	4,731,740	EugÃ©nie
6	12,496,698	RÃ©gine
7	38,713,647	åŸä¿Š
8	81,982,737	æ€?æ‡¿
9	15,311,673	AmÃ©lie And Anthony

After updating the weird characters, you can see the below screenshot how it is updated in the below screenshot.

```sql
-- View the problematic Guest_Names before updating
SELECT * FROM Cleaned_GUEST
WHERE Guest_Name LIKE '%ë%'
   OR Guest_Name LIKE '%ì%'
   OR Guest_Name LIKE '%–%'
   OR Guest_Name LIKE '%+%'
   OR Guest_Name LIKE '%Ã©%'
   OR Guest_Name LIKE '%åŸä¿Š%'
   OR Guest_Name LIKE '%æ€ðæ‡¿%';

-- Update problematic Guest_Names, removing unwanted characters
UPDATE Cleaned_GUEST
SET Guest_Name = REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(Guest_Name, 'ë', ''), 'ì', ''), '–', ''), '+', ''), 'Ã©', ''), 'åŸä¿Š', ''), 'æ€ðæ‡¿', '');

-- Verify that the characters have been removed
SELECT *
FROM Cleaned_GUEST
WHERE Guest_Name LIKE '%ë%'
   OR Guest_Name LIKE '%ì%'
   OR Guest_Name LIKE '%–%'
   OR Guest_Name LIKE '%+%'
   OR Guest_Name LIKE '%Ã©%'
   OR Guest_Name LIKE '%åŸä¿Š%'
   OR Guest_Name LIKE '%æ€ðæ‡¿%';

SELECT * FROM Cleaned_GUEST;
```

3. Duplicate Values

We have found duplicates in two tables, one from the booking table and the other from the host table. Before screenshot for showing the duplicates, values are as follows:

Booking table

```sql
-- View duplicate records in Cleaned_BOOKING
SELECT Booking_ID, COUNT(*)
FROM Cleaned_BOOKING
GROUP BY Booking_ID
HAVING COUNT(*) > 1;
```

	123 BOOKING_ID	123 COUNT(*)
1	537	2

```sql
-- View the duplicates to decide how to handle them
SELECT *
FROM Cleaned_BOOKING
WHERE Booking_ID IN (
    SELECT Booking_ID
    FROM Cleaned_BOOKING
    GROUP BY Booking_ID
    HAVING COUNT(*) > 1
);

-- Delete duplicate records from Cleaned_BOOKING, keeping only one row per Booking_ID
DELETE FROM Cleaned_BOOKING
WHERE ROWID NOT IN (
    SELECT MIN(ROWID)
    FROM Cleaned_BOOKING
    GROUP BY Booking_ID
);
```

Enter a SQL expression to filter results (use Ctrl+Space)

	123 BOOKING_ID	BOOKING_DATE	BOOKING_STAY_START_DATE	123 BOOKING_DURATION	123 BOOKING_COST	123 BOOKING_NUM_GUESTS	123 LIS
1	537	2015-05-11 00:00:00.000	2015-05-12 00:00:00.000	76	11,400	1	
2	537	2015-05-11 00:00:00.000	2015-05-12 00:00:00.000	76	11,400	1	

After viewing the duplicates, I have deleted the values that are not repeating here is the screenshot.

```
-- Delete duplicate records from Cleaned_BOOKING, keeping only one row per Booking_ID
DELETE FROM Cleaned_BOOKING
WHERE ROWID NOT IN (
    SELECT MIN(ROWID)
    FROM Cleaned_BOOKING
    GROUP BY Booking_ID
);
```

```
Updated Rows 1
Query          -- Delete duplicate records from Cleaned_BOOKING, keeping only one row per Booking_ID
               DELETE FROM Cleaned_BOOKING
               WHERE ROWID NOT IN (
                  SELECT MIN(ROWID)
                  FROM Cleaned_BOOKING
                  GROUP BY Booking_ID
               )
Start time     Sat Sep 21 16:20:10 AEST 2024
Finish time    Sat Sep 21 16:20:10 AEST 2024
```

Host Table

This is before the screenshot of duplicate values you can see in the following screenshots.

```
-- View duplicate records in Cleaned_HOST
SELECT Host_ID, COUNT(*)
FROM Cleaned_HOST
GROUP BY Host_ID
HAVING COUNT(*) > 1;

-- View the duplicates to decide how to handle them
SELECT *
FROM Cleaned_HOST
WHERE Host_ID IN (
    SELECT Host_ID
    FROM Cleaned_HOST
    GROUP BY Host_ID
    HAVING COUNT(*) > 1
);
```

	HOST_ID	COUNT(*)
1	7,046,664	4

	HOST_ID	HOST_NAME	HOST_SINCE	HOST_LOCATION	HOST_ABOUT	HOST_LISTING_COUNT
1	7,046,664	Dave	2013-06-22 00:00:00.000	Rosebud, Victoria, Australia	Living the dream on the Mornington Peninsula!	77
2	7,046,664	Dave	2013-06-22 00:00:00.000	Rosebud, Victoria, Australia	Living the dream on the Mornington Peninsula!	77
3	7,046,664	Dave	2013-06-22 00:00:00.000	Rosebud, Victoria, Australia	Living the dream on the Mornington Peninsula!	77
4	7,046,664	Dave	2013-06-22 00:00:00.000	Rosebud, Victoria, Australia	Living the dream on the Mornington Peninsula!	77

After the screenshot I have deleted the duplicates values in the host table here is the screenshot below.

```
-- Delete duplicate records from Cleaned_HOST, keeping only one row per Host_ID
DELETE FROM Cleaned_HOST
WHERE ROWID NOT IN (
    SELECT MIN(ROWID)
    FROM Cleaned_HOST
    GROUP BY Host_ID
);
```

Name	Value
Updated Rows	3
Query	-- Delete duplicate records from Cleaned_HOST, keeping only one row per Host_ID DELETE FROM Cleaned_HOST WHERE ROWID NOT IN (SELECT MIN(ROWID) FROM Cleaned_HOST GROUP BY Host_ID)
Start time	Sat Sep 21 16:28:06 AEST 2024
Finish time	Sat Sep 21 16:28:06 AEST 2024

4. Outlier

In the listing table, there is an outlier as you can see in the screenshot in the listing_price there is -150. Listing price should never be -150 so I updated the pricing listing to 150.

Before screenshot:

Enter a SQL expression to filter results (use Ctrl+Space)

	LISTING_ID	LISTING_DATE	LISTING_TITLE	LISTING_PRICE	LISTING_MIN_NIGHTS	LISTING_MAX_NIGHTS	PROP_ID	
1	99,999	2018-12-18 03:16:06.000	Melbourne accomodation	-150	1	7	9,999	

After screenshot:

```
-- Update Listing_Price to 150 for values greater than 10000 or less than 0
UPDATE Cleaned_LISTING
SET Listing_Price = 150
WHERE Listing_Price > 10000 OR Listing_Price < 0;
```

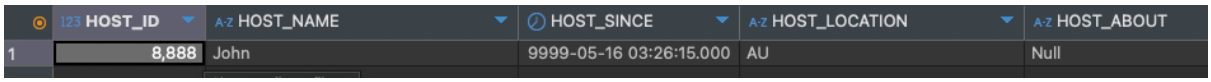
	LISTING_ID	LISTING_DATE	LISTING_TITLE	LISTING_PRICE	LISTING_MIN_NIGHTS	LISTING_MAX_NIGHTS	PROP_ID	TYPE_ID	HOST_ID
1	99,999	018-12-18 03:16:06.000	Melbourne accomodatior	150	1	7	9,999	2	9,999

5. Incorrect Values

Host table

In the host table, as you can see below screenshot, I have corrected the values of the date that has written 9999-05-16 for that I have made it null since no year has 9999 in this dataset, and it is irrelevant to the dataset we are doing. The below screenshots are both before and after data cleaning.

Before Screenshot

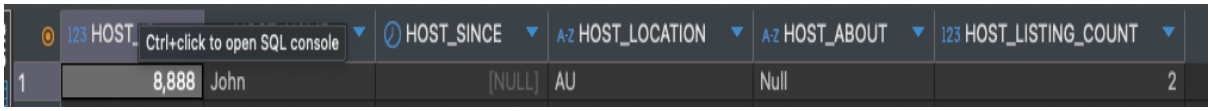


	HOST_ID	HOST_NAME	HOST_SINCE	HOST_LOCATION	HOST_ABOUT
1	8,888	John	9999-05-16 03:26:15.000	AU	Null

After Screenshot

```
-- Update Host_Since to NULL for records with '9999' date
UPDATE Cleaned_HOST
SET Host_Since = NULL
WHERE EXTRACT(YEAR FROM Host_Since) = 9999;

-- Verify the update by selecting rows where Host_Since is NULL
SELECT *
FROM Cleaned_HOST
WHERE Host_Since IS NULL;
```



	HOST_	HOST_SINCE	HOST_LOCATION	HOST_ABOUT	HOST_LISTING_COUNT
1	8,888	John	[NULL]	AU	Null	2

6. Relationship problems

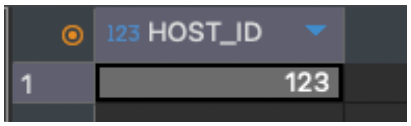
Host_Verification Table

In the host verification table when we select host_id it is seen in the screenshot that

Before screenshot:

```
-- Find Host_IDs in Cleaned_HOST_VERIFICATION that do not exist in Cleaned_HOST - foreign relational problems
SELECT Host_ID
FROM Cleaned_HOST_VERIFICATION
WHERE Host_ID NOT IN (
    SELECT Host_ID
    FROM Cleaned_HOST
);
```

Output:

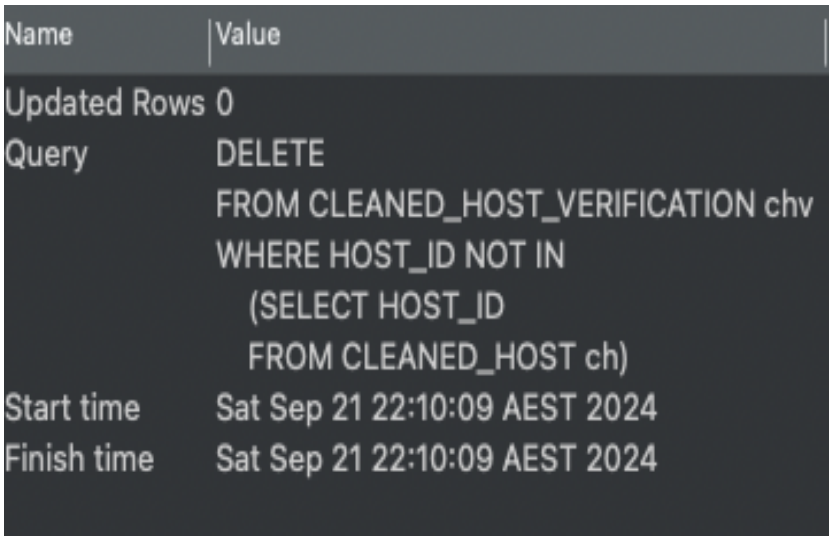


	HOST_ID
1	123

After screenshot:

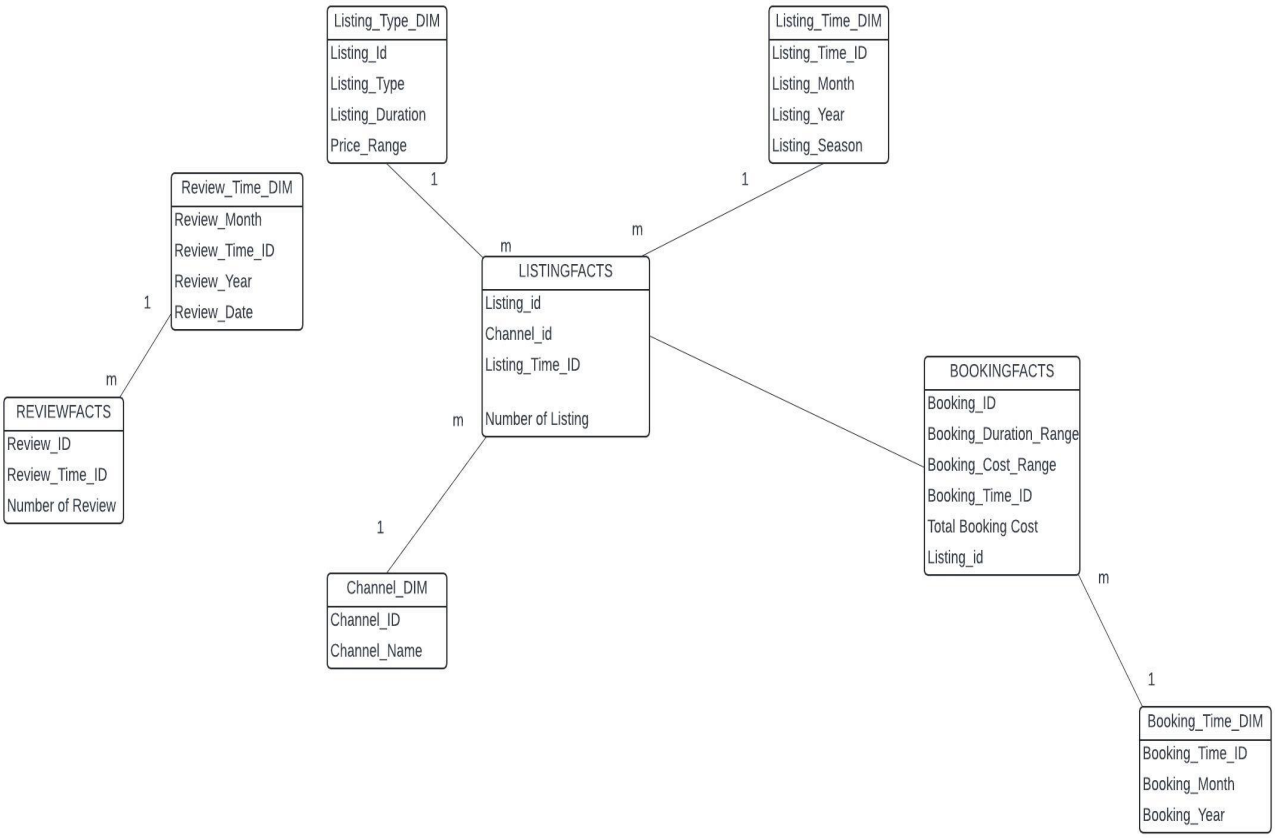
```
DELETE
FROM CLEANED_HOST_VERIFICATION chv
WHERE HOST_ID NOT IN
      (SELECT HOST_ID
      FROM CLEANED_HOST ch);
```

Output:



Name	Value
Updated Rows	0
Query	DELETE FROM CLEANED_HOST_VERIFICATION chv WHERE HOST_ID NOT IN (SELECT HOST_ID FROM CLEANED_HOST ch)
Start time	Sat Sep 21 22:10:09 AEST 2024
Finish time	Sat Sep 21 22:10:09 AEST 2024

Star Schema Diagram



List suggestions for increasing the granularity of your fact table.

Let's look at each fact table and dimension in the star schema to give you a thorough explanation of how to make the fact tables from Design Task A more granular. Additionally, I will suggest other dimensions and facts qualities that might be added to lower the aggregate level and enable the manager of the M-Stay company to do more in-depth drilldowns into operational areas.

1. Improving Time Specificity

Present circumstances: The only information included in Dim_Review_time and Dim_Booking_time is the Year and Month.

Suggestion

Daily Granularity: Expand both dimensions (Dim_Review_time, Dim_Booking_time) by adding a Day or Date column. This enables the management to monitor reservations and evaluations daily as opposed to merely monthly.

Hour/Minute Granularity: You could provide the time of day (e.g., the hour and minute of the booking or review) if a precise analysis is needed. This could be useful for analysing consumer behaviour depending on when they book or review.

The manager could drill down from monthly booking trends to daily or even hourly trends by adding this level of data. This is particularly helpful for promotions or events where the timing could affect how people book.

2. Guest and Host demographics

Present Situation: There are currently no fact tables or dimensions pertaining to guest or host demographic data.

Suggestion

Information about the guest: Include a Dim_Guest dimension that records important demographic details like age, gender, country, and if the guest is new or returning. This enables the management to monitor visitor loyalty and examine how various guest segments book properties.

Host Information: Establish a Dim_Host table with the host's years of hosting experience, verification status (confirmed or unverified), and ratings. Drill-downs into how host attributes impact listing performance, reservations, and ratings are made possible by this.

If we do this we can drill down by host attributes or guest demographics using these parameters. For instance, you could monitor which listing types are preferred by visitors of different ages or nationalities, or you could find out which hosts, based on their reputation and expertise, draw in the most bookings.

3. Information about Payment and Reservations

Present Situation: Payment details and the length of the booking are not included in the BOOKINGFACTS table.

Suggestions

Payment Method: Include the payment method (credit card, PayPal, bank transfer, etc.) in the Payment_Method dimension. This gives the manager insight into transaction behaviours by enabling them to monitor the preferred payment methods used by clients.

Booking Start and End Dates: To track the precise duration of each booking, enter Booking Start Date and Booking End Date in the BOOKINGFACTS table in place of Booking_duration_range.

Details of the Discount: Create a Dim_Discount dimension to describe the specific kind of discount (e.g., seasonal, promotional code), and add a field called Discount_Applied to BOOKINGFACTS to indicate if a discount was applied at the time of booking.

4. Geographic Information

Present Situation: Other than what the listing type might suggest, there is no geographic division.

Suggestions

Geographical Data: Establish a Dim_Location dimension with the postal code, region, city, and suburb. The manager can now examine specific locations to determine which ones are generating more reservations and reviews.

5. Cancellation Details

Present Situation: The BOOKINGFACTS table does not currently keep track of cancellations.

Suggestions

Cancellation Tracking: To keep track of cancelled reservations, add a Cancellation_Flag and Cancellation_Reason to BOOKINGFACTS. A Dim_Cancellation_Reason dimension can also

be made to hold predetermined explanations for cancellations (such as host or guest decisions or outside circumstances).

Drill-down analysis of booking cancellations will be possible as a result, assisting management in understanding the reasons behind and contributing factors to booking cancellations.

6. Marketing insights and source channels

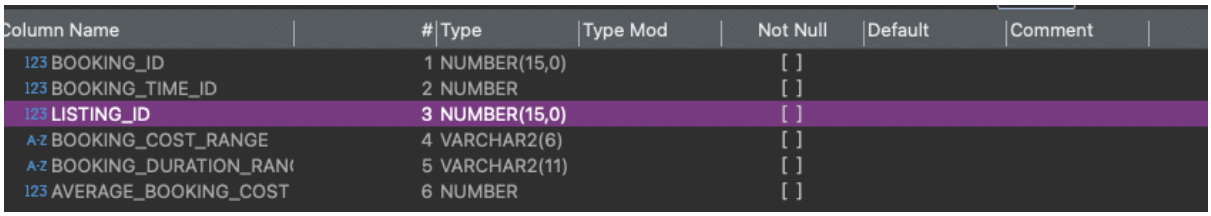
Present circumstances: Right now, the Dim_Channel keeps track of the channel name.

Suggestions

Channel Performance data: To measure marketing data like cost-per-click, click-through rates, and conversion rates for each channel, add more fields to Dim_Channel. This would make it possible to analyse how various marketing channels work in terms of drawing in reservations and reviews.

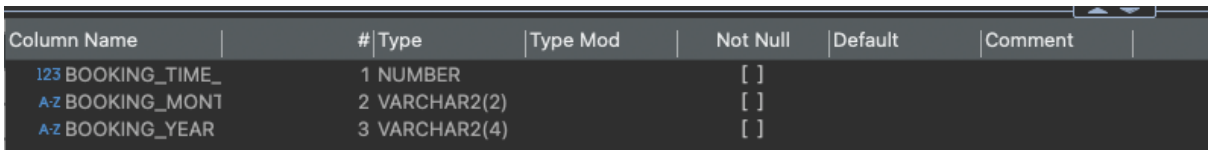
Screenshot of table structure - Design Task A

Booking FACTS Table Structure



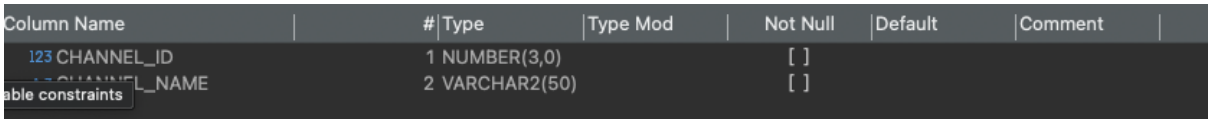
Column Name	#	Type	Type Mod	Not Null	Default	Comment
123 BOOKING_ID	1	NUMBER(15,0)		[ ]		
123 BOOKING_TIME_ID	2	NUMBER		[ ]		
123 LISTING_ID	3	NUMBER(15,0)		[ ]		
A-Z BOOKING_COST_RANGE	4	VARCHAR2(6)		[ ]		
A-Z BOOKING_DURATION_RAN(	5	VARCHAR2(11)		[ ]		
123 AVERAGE_BOOKING_COST	6	NUMBER		[ ]		

Dim_Booking_Time Table Structure



Column Name	#	Type	Type Mod	Not Null	Default	Comment
123 BOOKING_TIME_	1	NUMBER		[ ]		
A-Z BOOKING_MONT	2	VARCHAR2(2)		[ ]		
A-Z BOOKING_YEAR	3	VARCHAR2(4)		[ ]		

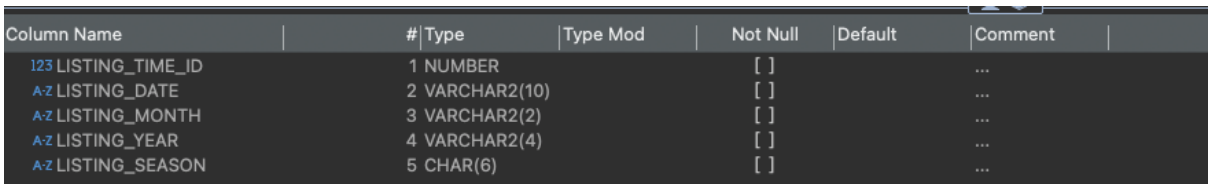
Dim_Channel Table Structure



Column Name	#	Type	Type Mod	Not Null	Default	Comment
123 CHANNEL_ID	1	NUMBER(3,0)		[ ]		
L_NAME	2	VARCHAR2(50)		[ ]		

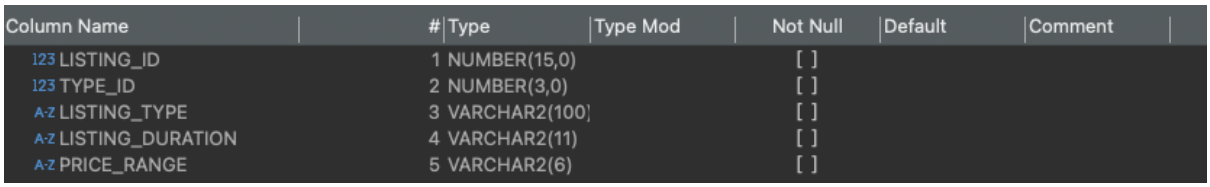
able constraints

Dim_Listing_Time Table Structure



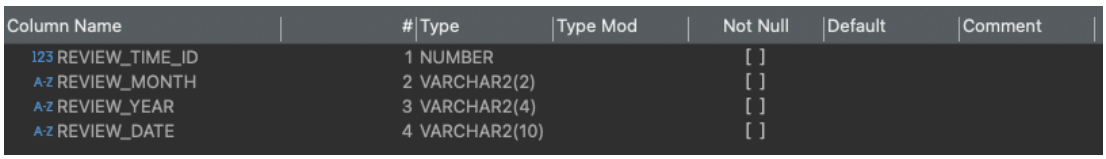
Column Name	#	Type	Type Mod	Not Null	Default	Comment
123 LISTING_TIME_ID	1	NUMBER		[ ]		...
A-Z LISTING_DATE	2	VARCHAR2(10)		[ ]		...
A-Z LISTING_MONTH	3	VARCHAR2(2)		[ ]		...
A-Z LISTING_YEAR	4	VARCHAR2(4)		[ ]		...
A-Z LISTING_SEASON	5	CHAR(6)		[ ]		...

Dim_Listing_Type Table Structure



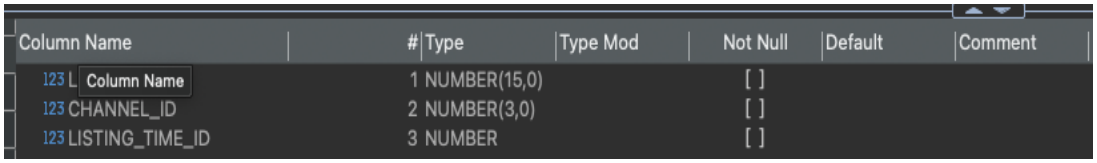
Column Name	#	Type	Type Mod	Not Null	Default	Comment
123 LISTING_ID	1	NUMBER(15,0)		[ ]		
123 TYPE_ID	2	NUMBER(3,0)		[ ]		
A-Z LISTING_TYPE	3	VARCHAR2(100)		[ ]		
A-Z LISTING_DURATION	4	VARCHAR2(11)		[ ]		
A-Z PRICE_RANGE	5	VARCHAR2(6)		[ ]		

Dim_Review_Time Table Structure



Column Name	#	Type	Type Mod	Not Null	Default	Comment
123 REVIEW_TIME_ID	1	NUMBER		[ ]		
A-Z REVIEW_MONTH	2	VARCHAR2(2)		[ ]		
A-Z REVIEW_YEAR	3	VARCHAR2(4)		[ ]		
A-Z REVIEW_DATE	4	VARCHAR2(10)		[ ]		

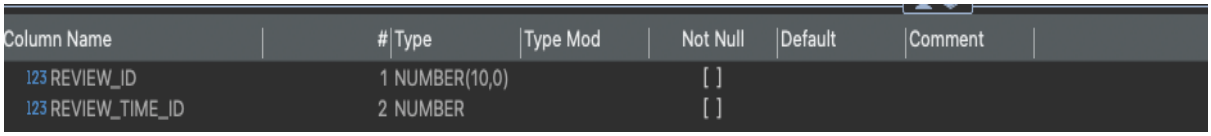
Dim_Listing_Fact Table Structure



Column Name	#	Type	Type Mod	Not Null	Default	Comment
123 L	1	NUMBER(15,0)		[ ]		
123 CHANNEL_ID	2	NUMBER(3,0)		[ ]		
123 LISTING_TIME_ID	3	NUMBER		[ ]		

Column Name

Review_Fact Table Structure



Column Name	#	Type	Type Mod	Not Null	Default	Comment
123 REVIEW_ID	1	NUMBER(10,0)		[ ]		
123 REVIEW_TIME_ID	2	NUMBER		[ ]		

Data Analytics Report

1. Bookings by Cost and Duration Range

```
SELECT
    Booking_Cost_range,
    Booking_duration_range,
    COUNT(Booking_id) AS total_bookings
FROM BOOKINGFACTS
GROUP BY Booking_Cost_range, Booking_duration_range
ORDER BY total_bookings DESC;
```

Explanation of the code:

This search examines the distribution of reservations over various Booking_Duration_ranges (Short-term, Medium-term, Long-term) and Booking_Cost_ranges (Low, Medium, High).

The number of bookings that fit into each pricing and time range is counted using COUNT(Booking_id).

Output

BOOKING_COST_RANGE	BOOKING_DURATION_RANGE	TOTAL_BOOKINGS
Medium	Medium-term	87,788
Low	Short-term	65,007
Low	Medium-term	37,852
High	Medium-term	26,473
High	Long-term	14,018
Medium	Long-term	11,492
Medium	Short-term	1,767
High	Short-term	155
Low	Long-term	13

Findings of the results

- The majority of reservations (87,788) are Medium-Cost, Medium-Term bookings, indicating that guests favour reasonably priced mid-range stays of moderate length.

- Low-cost, short-term reservations (65,007) are also in demand, indicating budget-conscious clients who choose shorter visits.

- High-cost, Medium-Term reservations (26,473 bookings) show that customers are drawn to premium listings for stays of moderate duration.

- Long-term, Low-Cost bookings are incredibly uncommon (only 13 bookings), indicating a lack of interest in low-cost, protracted stays.

- In general, mid-range listings yield the best results, but high-end reservations are less common yet generate substantial income from extended or moderate stays.

2. Average Booking Cost by Year and Month

```
SELECT
    bt.Booking_Year,
    bt.Booking_Month,
    AVG(bf.AVERAGE_BOOKING_COST) AS avg_cost
FROM BOOKINGFACTS bf
JOIN Dim_Booking_Time bt ON bf.Booking_time_id = bt.Booking_time_id
GROUP BY bt.Booking_Year, bt.Booking_Month;
```

Explanation of the code:

The average booking fee for the month and year is determined by this query. The average cost of bookings is kept in the BOOKINGFACTS table's AVERAGE_BOOKING_COST column. The query computes the monthly and annual averages using AVG (). The Dim Booking Timetable's Booking Year and Booking Month groups the results.

Output

	A-Z BOOKING_YEAR	A-Z BOOKING_MONTH	123 AVG_COST
	2013	03	6,458.0954960849
2	2013	07	6,458.0954960849
3	2014	07	6,458.0954960849
4	2014	10	6,458.0954960849
5	2014	12	6,458.0954960849
6	2015	02	6,458.0954960849
7	2015	06	6,458.0954960849
8	2015	10	6,458.0954960849
9	2016	02	6,458.0954960849
10	2017	05	6,458.0954960849
11	2017	09	6,458.0954960849
12	2018	01	6,458.0954960849
13	2019	02	6,458.0954960849
14	2019	07	6,458.0954960849
15	2012	01	6,458.0954960849
16	2021	06	6,458.0954960849

Justification

The table shows a constant average booking cost of 6,458.09 across all years and months from 2010 to 2021. This likely indicates a data or query issue, as booking costs should normally vary over time. The result suggests either the same value is being used repeatedly or there's an error in how the average is being calculated.

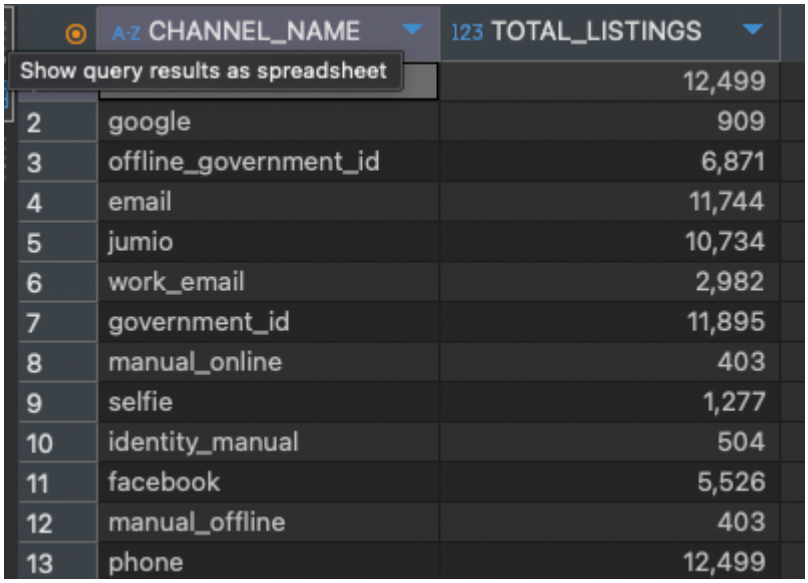
3. Total Listings by Channel

```
SELECT
    dc.Channel_name,
    COUNT(lf.Listing_id) AS total_listings
FROM LISTINGFACTS lf
JOIN Dim_Channel dc ON lf.Channel_id = dc.Channel_id
GROUP BY dc.Channel_name;
```

Explanation:

This query determines how many listings are connected to every marketing channel (website, social media, etc.). The COUNT function determines the total number of listings linked to each channel (lf.Listing_id).

Output



	CHANNEL_NAME	TOTAL_LISTINGS
		12,499
2	google	909
3	offline_government_id	6,871
4	email	11,744
5	jumio	10,734
6	work_email	2,982
7	government_id	11,895
8	manual_online	403
9	selfie	1,277
10	identity_manual	504
11	facebook	5,526
12	manual_offline	403
13	phone	12,499

Explanation:

With 12,499 listings each, the table reveals phones are the most popular channels for listing creation. Google, email, and government IDs are the next most popular ways of verification. The decreased prevalence of manual and offline procedures is probably caused by slower or less effective processes when compared to digital verification methods. This indicates a clear preference for technology-driven, quicker methods for listing creation and verification.

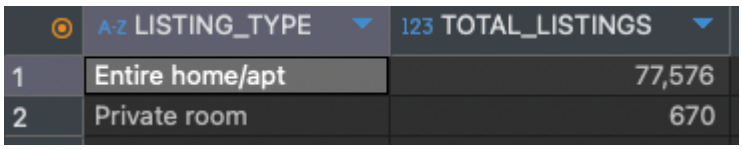
4. Total Listings by Listing Type

```
SELECT
    dlt.Listing_type,
    COUNT(lf.Listing_id) AS total_listings
FROM LISTINGFACTS lf
JOIN Dim_Listing_Type dlt ON lf.Listing_id = dlt.Listing_id
GROUP BY dlt.Listing_type;
```

Explanation:

The total number of listings for each Listing Type—such as an apartment or a house—is determined by this query. Listing_type is used to group results after COUNT (lf.Listing_id) counts the total number of listings.

Output



	LISTING_TYPE	TOTAL_LISTINGS
1	Entire home/apt	77,576
2	Private room	670

Explanation

With 77,576 listings, "entire home/apt" has the highest percentage, suggesting that clients prefer to book entire properties since they provide privacy and convenience during their stay. There are a lot less listings for private rooms—just 670. This may indicate that there is a greater demand for complete homes or apartments, or that hosts are giving fewer private rooms than entire properties.

5. Total Reviews by year

```
SELECT
    rt.Review_Year,
    COUNT(rf.Review_id) AS total_reviews
FROM Review_FACT rf
JOIN Dim_Review_Time rt ON rf.Review_time_id = rt.Review_time_id
GROUP BY rt.Review_Year
ORDER BY rt.Review_Year;
```

Explanation

The total number of reviews submitted annually is determined by this query. Review Year is used to group the results after COUNT (rf.Review_id) counts the total number of reviews.

Output

	REVIEW_YEAR	TOTAL_REVIEWS
1	2010	3
2	2011	52
3	2012	391
4	2013	1,069
5	2014	1,429
6	2015	1,924
7	2016	1,795
8	2017	2,123
9	2018	1,834
10	2019	1,766
11	2020	607
12	2021	399

Explanation

The reviews totalled annually from 2010 to 2021 are included in the table. From 2013, reviews have increased gradually, reaching a peak of 2,123 in 2017. After 2017, there is a discernible reduction, which drops off quickly in 2020 (perhaps because of the pandemic). Reviews somewhat increased to 399 in 2021, but they were still below pre-pandemic levels.

According to the data, customer interaction through reviews increased significantly up until 2017 before declining, particularly during the pandemic. The resurgence of the platform in 2021 is suggestive of a potential comeback.

Appendix

Creation of dimensions and facts

```sql
DROP TABLE Dim_Listing_Type CASCADE CONSTRAINTS;

CREATE TABLE Dim_Listing_Type AS
SELECT DISTINCT
    cl.Listing_id,
    lt.type_id,
    lt.type_description AS Listing_type,
    CASE
        WHEN cl.Listing_max_nights < 14 THEN 'Short-term'
        WHEN cl.Listing_max_nights BETWEEN 14 AND 30 THEN 'Medium-term'
        ELSE 'Long-term'
    END AS Listing_duration,
    CASE
        WHEN cl.listing_price < 100 THEN 'Low'
        WHEN cl.listing_price BETWEEN 100 AND 200 THEN 'Medium'
        ELSE 'High'
    END AS price_range
FROM Cleaned_LISTING cl
JOIN Cleaned_LISTING_TYPE lt ON cl.type_id = lt.type_id;

SELECT * FROM DIM_LISTING_TYPE dlt;

DROP TABLE Dim_Listing_Time CASCADE CONSTRAINTS;
CREATE TABLE Dim_Listing_Time AS
SELECT DISTINCT
    ROW_NUMBER() OVER (ORDER BY TO_CHAR(listing_date, 'YYYY-MM-DD')) AS Listing_time_id,
    TO_CHAR(listing_date, 'DD/MM/YYYY') AS Listing_Date,
    TO_CHAR(listing_date, 'MM') AS Listing_Month,
    TO_CHAR(listing_date, 'YYYY') AS Listing_Year,
    CASE
        WHEN TO_CHAR(listing_date, 'MM') BETWEEN '09' AND '11' THEN 'Spring'
        WHEN TO_CHAR(listing_date, 'MM') IN ('12', '01', '02') THEN 'Summer'
        WHEN TO_CHAR(listing_date, 'MM') BETWEEN '03' AND '05' THEN 'Autumn'
        ELSE 'Winter'
    END AS Listing_Season
FROM Cleaned_LISTING;

SELECT * FROM DIM_LISTING_TIME dlt;

DROP TABLE Dim_Channel CASCADE CONSTRAINTS;
CREATE TABLE Dim_Channel AS
SELECT DISTINCT
    ch.Channel_id,
    ch.Channel_name
FROM Cleaned_CHANNEL ch;

SELECT * FROM DIM_CHANNEL dc;
DROP TABLE Dim_review_time CASCADE CONSTRAINTS;
CREATE TABLE Dim_Review_Time AS
SELECT DISTINCT
    ROW_NUMBER() OVER (ORDER BY TO_CHAR(review_date, 'YYYY-MM-DD')) AS Review_time_id, -- Unique ID for each date
    TO_CHAR(review_date, 'MM') AS Review_Month,   -- Month from review_date
    TO_CHAR(review_date, 'YYYY') AS Review_Year,  -- Year from review_date
    TO_CHAR(review_date, 'DD/MM/YYYY') AS Review_Date -- Full date in DD/MM/YYYY format
FROM Cleaned_REVIEW;

SELECT * FROM DIM_REVIEW_TIME drt;

DROP TABLE Dim_Booking_Time CASCADE CONSTRAINTS;
CREATE TABLE Dim_Booking_Time AS
SELECT DISTINCT
    ROW_NUMBER() OVER (ORDER BY TO_CHAR(booking_date, 'YYYY-MM')) AS Booking_time_id,
    TO_CHAR(booking_date, 'MM') AS Booking_Month,
    TO_CHAR(booking_date, 'YYYY') AS Booking_Year
FROM Cleaned_BOOKING;

SELECT * FROM DIM_BOOKING_TIME dbt;
```

```sql
DROP TABLE LISTINGFACTS CASCADE CONSTRAINTS;

CREATE TABLE LISTINGFACTS AS
SELECT
    inner_query.Listing_id,
    inner_query.Channel_id,
    inner_query.Listing_time_id
FROM (
    SELECT
        cl.Listing_id,
        hv.Channel_id,
        lt.Listing_time_id,
        COUNT(cl.Listing_id) OVER () AS total_listings -- This counts the listings but doesn't show it in the final result
    FROM Cleaned_LISTING cl
    JOIN Cleaned_HOST_VERIFICATION hv ON cl.Host_id = hv.Host_id
    JOIN Dim_Channel ch ON hv.Channel_id = ch.Channel_id
    JOIN Dim_Listing_Time lt ON TO_CHAR(cl.Listing_date, 'DD/MM/YYYY') = lt.Listing_Date
    GROUP BY cl.Listing_id, hv.Channel_id, lt.Listing_time_id
) inner_query;


SELECT * FROM LISTINGFACTS l;

DROP TABLE Review_FACT CASCADE CONSTRAINTS;

CREATE TABLE Review_FACT AS
SELECT
    cr.Review_id,           -- Review ID from Cleaned_REVIEW
    rt.Review_time_id       -- Time ID from Dim_Review_Time
FROM Cleaned_REVIEW cr
JOIN Dim_Review_Time rt
  ON TO_CHAR(cr.Review_Date, 'DD/MM/YYYY') = rt.Review_Date -- Linking by exact review date
GROUP BY cr.Review_id, rt.Review_time_id;

DROP TABLE BOOKINGFACTS CASCADE CONSTRAINTS;

CREATE TABLE BOOKINGFACTS AS
SELECT
    cb.Booking_id,
    bt.Booking_time_id,
    cb.Listing_id,  -- Added Listing_id
    -- Categorizing Booking Cost Range
    CASE
        WHEN cb.Booking_Cost < 5000 THEN 'Low'
        WHEN cb.Booking_Cost BETWEEN 5000 AND 10000 THEN 'Medium'
        WHEN cb.Booking_Cost > 10000 THEN 'High'
    END AS Booking_Cost_range,
    -- Categorizing Booking Duration Range
    CASE
        WHEN cb.Booking_Duration < 30 THEN 'Short-term'
        WHEN cb.Booking_Duration BETWEEN 30 AND 90 THEN 'Medium-term'
        WHEN cb.Booking_Duration > 90 THEN 'Long-term'
    END AS Booking_duration_range,
    AVG(cb.Booking_Cost) OVER () AS Average_Booking_Cost  -- Displaying average booking cost as a column
FROM Cleaned_BOOKING cb
LEFT JOIN Dim_Booking_Time bt
  ON EXTRACT(MONTH FROM cb.Booking_Date) = bt.Booking_Month
 AND EXTRACT(YEAR FROM cb.Booking_Date) = bt.Booking_Year
LEFT JOIN Cleaned_LISTING cl
  ON cb.Listing_id = cl.Listing_id;


SELECT * FROM BOOKINGFACTS b;
```

Queries

1. How many long-term stay duration listings are listed on Facebook?

```
-- Question 1
SELECT COUNT(lf.Listing_id) AS Long_Term_Listings_On_Facebook
FROM LISTINGFACTS lf
JOIN Dim_Channel dc ON lf.Channel_id = dc.Channel_id
JOIN Dim_Listing_Type dlt ON lf.Listing_id = dlt.Listing_id
WHERE dlt.Listing_duration = 'Long-term'
AND dc.Channel_name = 'facebook';
```

Output

	123 LONG_TERM_LISTINGS_ON_FACEBOOK
1	5,299

2. How many listings are listed in June 2015?

```
-- Question 2
SELECT COUNT(lf.Listing_id) AS Listings_in_June_2015
FROM LISTINGFACTS lf
JOIN Dim_Listing_Time dlt ON lf.Listing_time_id = dlt.Listing_time_id
WHERE dlt.Listing_Month = '06'
AND dlt.Listing_Year = '2015';
```

Output:

	123 LISTINGS_IN_JUNE_2015
1	963

3. How many listings are there in summer for an “Entire home/apt” in a medium price range?

```
-- Question 3
SELECT COUNT(lf.Listing_id) AS Summer_Entire_Home_Apt_Medium_Price
FROM LISTINGFACTS lf
JOIN Dim_Listing_Type dlt ON lf.Listing_id = dlt.Listing_id
JOIN Dim_Listing_Time dltime ON lf.Listing_time_id = dltime.Listing_time_id
WHERE dlt.Listing_type = 'Entire home/apt'
AND dlt.price_range = 'Medium'
AND dltime.Listing_Season = 'Summer';
```

Output:

	123 SUMMER_ENTIRE_HOME_APT_MEDIUM_PRICE
1	11,752

4. How much is the average booking cost in March 2013?

```
-- Question 4

SELECT AVG(bf.Average_Booking_Cost) AS Avg_Booking_Cost_March_2013
FROM BOOKINGFACTS bf
JOIN Dim_Booking_Time dbt ON bf.Booking_time_id = dbt.Booking_time_id
WHERE dbt.Booking_Month = '03'  -- March
AND dbt.Booking_Year = '2013';
```

Output:

	123 AVG_BOOKING_COST_MARCH_2013
1	6,458.095496084

5. How many bookings were there for “Private rooms” with a short-term stay duration in 2015?

```
-- Question 5
SELECT COUNT(bf.Booking_id) AS Short_Term_Private_Room_Bookings_2015
FROM BOOKINGFACTS bf
JOIN Dim_Listing_Type dlt ON bf.Listing_id = dlt.Listing_id
JOIN Dim_Booking_Time dbt ON bf.Booking_time_id = dbt.Booking_time_id
WHERE dlt.Listing_type = 'Private room'  -- Trimmed "Private room"
AND bf.Booking_duration_range = 'Short-term'  -- Trimmed short-term stay duration
AND dbt.Booking_Year = '2015';
```

Output:

	123 SHORT_TERM_PRIVATE_ROOM_BOOKINGS_2015
1	95

6. How many high-cost bookings were made in April 2014?

```
-- Question 6

SELECT COUNT(bf.Booking_id) AS High_Cost_Bookings_April_2014
FROM BOOKINGFACTS bf
JOIN Dim_Booking_Time dbt ON bf.Booking_time_id = dbt.Booking_time_id
WHERE TRIM(bf.Booking_Cost_range) = 'High'  -- Filter for high-cost bookings
AND TRIM(dbt.Booking_Month) = '04'  -- Filter for April
AND dbt.Booking_Year = '2014';  -- Filter for the year 2014
```

Output:

	123 HIGH_COST_BOOKINGS_APRIL_2014
1	273

7. How many reviews were given in February 2016?

```
-- Question 7
SELECT COUNT(rf.Review_id) AS Reviews_February_2016
FROM Review_FACT rf
JOIN Dim_Review_Time drt ON rf.Review_time_id = drt.Review_time_id
WHERE TRIM(drt.Review_Month) = '02'  -- Filter for February
AND drt.Review_Year = '2016';  -- Filter for the year 2016
```

Output:

	123 REVIEWS_FEBRUARY_2016
1	188

Reference

FIT5137. (n.d.). *SQL Exercise*. Retrieved from https://docs.google.com/document/d/1xceRZskCardPc2V2IYDPTTew3tUgDJ1mfugf_aL9GqY/edit

FIT5137. (n.d.). *M-Stay project breakdown*. Retrieved from https://docs.google.com/document/d/1aUcHPV1WqnDBI9yHcVRnzSiW32hM4TKYhT9r-mRYjC4/edit

FIT5137. (n.d.). *Data Cleaning*. Retrieved from https://docs.google.com/document/d/1cfFhZBUbpYvJzj_w3vKJDFDFMr3R1F6EkL_SIE_VIAc/edit

FIT5137. (n.d.). *Pre-Data Warehouse: Exploring Dirty Data*. Retrieved from https://docs.google.com/document/d/1JrdqoPV9Apy1xo7NVxZQqWg_qbGIQocmntBfIg9uz9g/edit

FIT5137. (n.d.). *Bridge Table.* Retrieved from https://docs.google.com/document/d/1S8YCF9qjL8rEFIgdG_G33laSd9XOUBtNxcUCt1FcqiE/edit

FIT5137. (n.d.). *Granularity*. Retrieved from https://docs.google.com/document/d/1SpYefy5kBu05iYqwbRQPTDZtaUSf4p_H1cji1nwuYG4/edit

Databricks. (n.d.). *Star schema*. Retrieved from https://www.databricks.com/glossary/star-schema#:~:text=A%20star%20schema%20is%20used,like%20transaction%20amounts%20and%20quantities

Stack Overflow. (n.d.). *Modelling and querying multiple fact tables*. Retrieved from https://stackoverflow.com/questions/49251334/modelling-and-querying-multiple-fact-tables

Stack Overflow. (n.d.). *SQL query to select records with altering granularity*. Retrieved from https://stackoverflow.com/questions/66302739/sql-query-to-select-records-with-altering-granularity

Johnson, C. (2023, April 13). *Granularity in queries*. Retrieved from https://chrisjohnson120.com/2023/04/13/granularity-in-queries/