SQL Data Cleaning

Table of contents

1. Database Setup in PostgreSQL

-- Table: purchase_data

```sql
CREATE TABLE purchase_data (
    user_id TEXT,
    order_id TEXT,
    purchase_ts DATE,
    ship_ts DATE,
    product_name TEXT,
    product_id TEXT,
    usd_price NUMERIC,
    purchase_platform TEXT,
    marketing_channel TEXT,
    account_creation_method TEXT,
    country_code CHAR(2)
);
```

-- Table: country_region

```sql
CREATE TABLE country_region (
    country_code CHAR(2) PRIMARY KEY,
    region TEXT
);
```

2. Investigating and Cleaning the Data

Step 1: Conceptualize the Data

Understand the structure and meaning of the data.

1.1 What does each row represent?

Each row in purchase_data represents one product purchase made by a user.

1.2 What are the key metrics?

- usd_price
- (later derived) time_to_ship → depens on the analytical goal

1.3 What are the key dimensions?

- user_id
- order_id

- product_name
- product_id
- purchase_platform
- marketing_channel
- account_creation_method
- country_code
- purchase_ts, ship_ts

Step 2: Data Profiling

Get a first overview of the data

2.1 Column Data Types

```sql
SELECT
    column_name,
    data_type
FROM information_schema.columns
WHERE table_name = 'purchase_data';
```

column_name	data_type
ship_ts	date
usd_price	numeric
purchase_ts	date
product_id	text
purchase_platform	text
marketing_channel	text
account_creation_method	text
user_id	text
country_code	character
order_id	text
product_name	text

→ all columns have the right data type

2.2 Table view of the first 5 rows

```sql
SELECT *
FROM purchase_data
LIMIT 5;
```

user_id	order_id	purchase_ts	ship_ts	product_name	product_id	usd_price	purchase_platform	marketing_channel	account_creation_method	country_code
2c06175e	0001328c3c220830	2020-12-24	2020-12-13	Nintendo Switch	e682	168	website	affiliate	unknown	US
ee8e5bc2	0002af7a5c6100772	2020-10-01	2020-09-21	Nintendo Switch	e682	160.61	website	direct	desktop	DE
9eb4efe0	0002b8350e167074	2020-04-21	2020-02-16	Nintendo Switch	8d0d	151.2	website	direct	desktop	US
cac7cbaf	0006d06b9838572 9	2020-04-07	2020-04-04	Sony Play Station 5 Bundle	54ed	1132.82	website	direct	desktop	AU
6b0230bc	00097279a2f46150	2020-11-24	2020-08-02	Nintendo Switch	8d0d	33.89	website	direct	desktop	TR

➔ at first glance the data looks good, dates are formatted right as well as usd_price which uses a “.” as separator

2.3 Key Metrics Analysis

```sql
SELECT
    COUNT(*) AS total_rows,
    COUNT(DISTINCT user_id) AS unique_customers,
    COUNT(DISTINCT order_id) AS unique_orders,
    COUNT(DISTINCT product_id) AS unique_products,
    MIN(purchase_ts) AS first_purchase_date,
    MAX(purchase_ts) AS last_purchase_date,
    ROUND(AVG(usd_price),2) AS avg_order_value,
    ROUND(SUM(usd_price),2) AS total_revenue
FROM purchase_data;
```

total_rows	unique_customers	unique_orders	unique_products	first_purchase_date	last_purchase_date	avg_order_value	total_revenue
21864	19851	21717	46	2019-01-01	2021-02-28	281.41	6151266.49

Key Findings from Purchase Data (Jan 2019 - Feb 2021):

- **Total Transactions**: The dataset contains **21,864 purchases** across the 26-month period.
- **Customer Base**: **19,851 unique customers** made purchases, indicating most customers (91%) placed only one order (21,717 unique orders vs 19,851 customers).
- **Product Catalog**: Relatively small offering with just **46 unique products**.
- **Revenue Performance**:
    - Generated **6,151,266.49** in total revenue
    - Average order value of **281.41**
- **Time Coverage**: Data spans from **January 1, 2019** to **February 28, 2021**

2.4 Key Dimensions Analysis

```sql
SELECT
    purchase_platform,
    COUNT(*) AS order_count,
    ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM
purchase_data), 2) AS percentage
FROM purchase_data
GROUP BY purchase_platform
ORDER BY order_count DESC;
```

purchase_platform	order_count	percentage
website	19783	90.48
mobile app	2081	9.52

```sql
SELECT
    marketing_channel,
    COUNT(*) AS order_count
FROM purchase_data
GROUP BY marketing_channel
ORDER BY order_count DESC;
```

marketing_channel	order_count
direct	17434
email	3256
affiliate	721
social media	323
	83
unknown	47

```sql
SELECT
    cr.region,
    COUNT(*) AS order_count
FROM purchase_data pd
LEFT JOIN country_region cr ON pd.country_code =
cr.country_code
GROUP BY cr.region
ORDER BY order_count DESC;
```

Region	Order Count
NA	10342
EMEA	6509
APAC	2577
LATAM	1243
North America	963
	184
	42
X.x	4

Step 3: Locate Data Issues

Found issues are documented and solved (if possible) in Step 5.

3.1 Count Nulls/ empty cells per Column

```sql
SELECT
    COUNT(*) AS total_rows,
    COUNT(*) FILTER (WHERE NULLIF(user_id, '') IS NULL) AS
user_id_nulls_or_empty,
    COUNT(*) FILTER (WHERE NULLIF(order_id, '') IS NULL) AS
order_id_nulls_or_empty,
    COUNT(*) FILTER (WHERE purchase_ts IS NULL) AS
purchase_ts_nulls,
    COUNT(*) FILTER (WHERE ship_ts IS NULL) AS ship_ts_nulls,
    COUNT(*) FILTER (WHERE NULLIF(product_name, '') IS NULL)
AS product_name_nulls_or_empty,
    COUNT(*) FILTER (WHERE NULLIF(product_id, '') IS NULL) AS
product_id_nulls_or_empty,
    COUNT(*) FILTER (WHERE usd_price IS NULL) AS
usd_price_nulls,
    COUNT(*) FILTER (WHERE NULLIF(purchase_platform, '') IS
NULL) AS purchase_platform_nulls_or_empty,
    COUNT(*) FILTER (WHERE NULLIF(marketing_channel, '') IS
NULL) AS marketing_channel_nulls_or_empty,
    COUNT(*) FILTER (WHERE NULLIF(account_creation_method, '')
IS NULL) AS account_creation_method_nulls_or_empty,
    COUNT(*) FILTER (WHERE NULLIF(country_code, '') IS NULL)
AS country_code_nulls_or_empty
FROM purchase_data;
```

total_rows	user_id_nulls_or_empty	order_id_nulls_or_empty	purchase_ts_nulls	ship_ts_nulls	product_name_nulls_or_empty	product_id_nulls_or_empty	usd_price_nulls	purchase_platform_nulls_or_empty	marketing_channel_nulls_or_empty	account_creation_method_nulls_or_empty	country_code_nulls_or_empty
21864	0	0	1	0	0	0	5	0	83	83	37

3.2 Check for Duplicates

-- Check for duplicate rows (all columns identical)

```sql
SELECT user_id, order_id, purchase_ts, product_id, COUNT(*)
FROM purchase_data
GROUP BY user_id, order_id, purchase_ts, product_id
HAVING COUNT(*) > 1;
```

user_id	order_id	purchase_ts	product_id	count
515bfd3a	8f17d3f0be035984	2020-01-23	891b	**2**
c466caab	a4423107e2e83156	2020-01-27	891b	**2**
7,31E+08	1.37896E+13	2020-11-24	5142	**2**
e2d73069	dab4e97af2095826	2020-01-26	891b	**2**
a08e87f5	c6ae92bfd8a67910	2020-01-26	e7e6	**2**

[...]

Total rows: 39

```sql
SELECT order_id, COUNT(*)
FROM purchase_data
GROUP BY order_id
HAVING COUNT(*) > 1;
```

-- Check for duplicate orders

order_id	count
e6ef83e624f71059	2
ed2c17e3ea888232	2
e9e88f519b773427	2

a6a25e392b124142	2
f419f9ba0df20651	2

[...]

Total rows: 147

3.3 Check Data Consistency

-- Check for inconsistent purchase_platform values

```sql
SELECT purchase_platform, COUNT(*)
FROM purchase_data
GROUP BY purchase_platform
ORDER BY purchase_platform;
```

purchase_platform	count
mobile app	2081
website	19783

-- Check for inconsistent marketing_channel values

```sql
SELECT marketing_channel, COUNT(*)
FROM purchase_data
GROUP BY marketing_channel
ORDER BY marketing_channel;
```

Marketing Channel	Count
marketing_channel	83
affiliate	721
direct	17434
email	3256
social media	323
unknown	47

-- Check for inconsistent product_name values

```sql
SELECT product_name, COUNT(*)
FROM purchase_data
GROUP BY product_name
ORDER BY product_name;
```

Product Name	Count
27in 4K gaming monitor	4662
27inches 4k gaming monitor	61
Acer Nitro V Gaming Laptop	87
Dell Gaming Mouse	719
JBL Quantum 100 Gaming Headset	4296
Lenovo IdeaPad Gaming 3	669
Nintendo Switch	10386
Razer Pro Gaming Headset	7
Sony PlayStation 5 Bundle	977

3.4 Business logic violations

-- Check ship dates before purchase dates

```sql
SELECT COUNT(*)
FROM purchase_data
WHERE ship_ts < purchase_ts;
```

Attribute	Value
count	2000

3.5 Outliers

```sql
WITH stats AS (
    SELECT
        AVG(usd_price) AS mean,
        STDDEV(usd_price) AS stddev
    FROM purchase_data
)
SELECT
    COUNT(*) FILTER (WHERE usd_price < mean - 3 * stddev) AS
low_outliers,
    COUNT(*) FILTER (WHERE usd_price <= 0) AS price_0,
    COUNT(*) FILTER (WHERE usd_price > mean + 3 * stddev) AS
high_outliers,
    COUNT(*) AS total_count
FROM purchase_data, stats;
```

low_outliers	price_0	high_outliers	total_count
0	29	818	21864

Investigating high outliers further:

```sql
WITH stats AS (
    SELECT
        AVG(usd_price) AS mean,
        STDDEV(usd_price) AS stddev
    FROM purchase_data
),
product_medians AS (
    SELECT
        product_name,
        PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY usd_price)
AS median_price
    FROM purchase_data
    GROUP BY product_name
),
high_outliers AS (
    SELECT
        pd.*,
        pm.median_price,
        (pd.usd_price - stats.mean) / stats.stddev AS z_score
    FROM purchase_data pd
    CROSS JOIN stats
    JOIN product_medians pm ON pd.product_name =
pm.product_name
    WHERE pd.usd_price > stats.mean + 3 * stats.stddev
)
SELECT
    order_id,
    user_id,
    product_name,
    usd_price,
    median_price,
    ROUND(z_score, 2) AS z_score,
    CASE
        WHEN usd_price > 10 * median_price THEN
'EXTREME_OUTLIER'
        WHEN usd_price > 5 * median_price THEN
'SEVERE_OUTLIER'
        ELSE 'MODERATE_OUTLIER'
    END AS outlier_severity
FROM high_outliers
ORDER BY z_score DESC;
```

order_id	user_id	product_name	usd_price	median_price	z_score	outlier_severity
e7023c6eb9032084	47c3a16b	Sony PlayStation 5 Bundle	3146.88	1699.91	7.82	MODERATE_OUTLIER
a0306bced5a40489	5cc9065d	Sony PlayStation 5 Bundle	2509.67	1699.91	6.08	MODERATE_OUTLIER
9867234962049857	741a663e	Sony PlayStation 5 Bundle	2308.32	1699.91	5.53	MODERATE_OUTLIER
5920e517ea640884	5dbb56ac	Sony PlayStation 5 Bundle	2250.93	1699.91	5.38	MODERATE_OUTLIER
b20616b63f167635	9ffc43cc	Sony PlayStation 5 Bundle	2250.93	1699.91	5.38	MODERATE_OUTLIER

[...]

➔ Investigating the high outliers further, we see these are all the product Playstation 5 with a high price (median price of this product name is much higher then the median of all products) → therefore no data issue with too high prices

Step 4: Data Augmentation

4.1 Calculate Time to Ship

```sql
ALTER TABLE purchase_data_clean ADD COLUMN time_to_ship INT;
```

```sql
UPDATE purchase_data_clean
SET time_to_ship = ship_ts - purchase_ts;
```

4.2 Add Year, Month, Day, Weekday/Weekend

```sql
ALTER TABLE purchase_data_clean
ADD COLUMN purchase_year INT,
ADD COLUMN purchase_month INT,
ADD COLUMN purchase_day INT,
ADD COLUMN purchase_weekday TEXT;
```

Create weekday/weekend:

```sql
ALTER TABLE purchase_data_clean ADD COLUMN is_weekend BOOLEAN;
```

```sql
UPDATE purchase_data_clean
SET is_weekend =
    CASE
        WHEN EXTRACT(DOW FROM purchase_ts) IN (0, 6) THEN TRUE
        ELSE FALSE
    END;
```

Check new features by displaying the first 5 rows:

user_id	order_id	purchase_ts	ship_ts	product_name	product_id	usd_price	purchase_platform	marketing_channel	account_creation_method	country_code	data_quality_flag	time_to_ship	purchase_year	purchase_month	purchase_day	purchase_weekday	is_weekend
0003cba8	f5a46cd24b3316 1	2020-03-13	2020-03-16	JBL Quantum100	ab0f	24	website	direct	desktop	US		3	2020	3	13	Friday	false

				Gaming Headset													
00005fdf2	7766c44e7c4316 4	2020-02-29	2020-03-02	Nintendo Switch	e682	168	website	direct	desktop	US		2	2020	2	29	Saturday	true
00007b8400	8cb976ccb3d3165	2019-11-24	2019-11-25	Nintendo Switch	8d0d	1088.68	website	direct	mobile	AU		1	2019	11	24	Sunday	true
296c4bba	c42338f2fbc197765	2019-12-31	2020-01-03	AcerNitroVGamingLaptop	22ea	7455.04	website	direct	desktop	KR		3	2019	12	31	Tuesday	false

296dd8ac	de09186ca2f19769	2019-02-15	2019-02-18	Nintendo Switch	8d0d	149.02	website	email	mobile	JP		3	2019	2	15	Friday	false

Step 5: Note and Document (Issue Log)

The following issue log documents the found issues and their resolution. All changes are performed on a duplicated table. After this table there are in deep details of the issues ordered by issue number:

Issue Number	Table	Column	Issue	Row Count	Magnitude	Solvable	Resolution
1	purchase_data	purchase_ts	NULL or Empty cell	1	0%	No	Leave as it is – very small impact
2	purchase_data	usd_price	NULL or Empty cell	5	0%	No	Set empty to NULL, Low impact, check with stakeholders
3	purchase_data	marketing_channel	NULL or Empty cell	83	0.4%	Yes	Fill empty cells with the value “unknown”
4	purchase_data	account_creation_method	NULL or Empty cell	83	0.4%	Yes	Fill empty cells

							with the value “unknown”
5	purchase_data	country_code	NULL or Empty cell	37	0.2%	Yes	Fill empty cells with the value “--”
6	purchase_data	user_id,order_id, purchase_ts, product_id	duplicates	39	0.2%	Yes	Remove duplicates
7	purchase_data	order_id	dublicate order_id	108	0.5%	No (true user_id cannot be determined)	Data quality flag
8	purchase_data	product_name	Two identical names with different spelling	61	0.3%	Yes	Rename "27inches 4K gaming monitor” to "27in 4k gaming monitor”
9	country_region	region	Empty, X.x values	4	2%	Yes	Set values to unknown
10	country_region	region	NA values	21	11%	No (NA stands for North America or Caribb	Need to check with stakeholders

						ean and outlying territories?)	
11	Purchase_data, country_region	region	Country_code in purchase_data as no matching country_code in country_region	42	42 rows without match	No	Low impact, Check with business stakeholder
12	purchase_data	Ship_ts, purchase_ts	Shipping before ordering	2000	9.15%	No	Data quality flag
13	purchase_data	usd_price	0 values	29	0.13%	No	Data quality flag, check with stakeholders

Solving (if possible) the data issues:

Issue 3: empty marketing channels

-- First update empty strings to NULL

```sql
UPDATE purchase_data_clean
SET marketing_channel = NULL
WHERE marketing_channel = '';
```

-- Then update NULLs to 'unknown'

```sql
UPDATE purchase_data_clean
SET marketing_channel = 'unknown'
WHERE marketing_channel IS NULL;
```

-- Check the solution

```sql
SELECT marketing_channel, COUNT(*)
FROM purchase_data_clean
GROUP BY marketing_channel
ORDER BY marketing_channel;
```

Issue 4: empty account_creation_method

-- Check account creation

```sql
SELECT
    account_creation_method,
    COUNT(*) AS order_count
FROM purchase_data_clean
GROUP BY account_creation_method
ORDER BY order_count DESC;
```

-- Set empty to Null

```sql
UPDATE purchase_data_clean
SET account_creation_method = NULL
WHERE account_creation_method = '';
```

-- Set Null to unknown

```sql
UPDATE purchase_data_clean
SET account_creation_method = 'unknown'
WHERE account_creation_method IS NULL;
```

Issue 5: empty country code

-- Check country code

```sql
SELECT
    country_code,
    COUNT(*) AS order_count
FROM purchase_data_clean
GROUP BY country_code
ORDER BY order_count DESC;
```

-- Set empty to Null

```sql
UPDATE purchase_data_clean
SET country_code = NULL
WHERE country_code = '';
```

-- Set Null to unknown

```sql
UPDATE purchase_data_clean
SET country_code = '--'
WHERE country_code IS NULL;
```

Issue 6: Duplicates

--Create a clean table first

```sql
CREATE TABLE purchase_data_clean AS
SELECT DISTINCT ON (user_id, order_id, purchase_ts,
product_id) *
FROM purchase_data;
```

-- Verify counts match expected deduplication

```sql
    (SELECT COUNT(*) FROM purchase_data) AS original_count,
    (SELECT COUNT(*) FROM purchase_data_clean) AS
deduplicated_count,
    (SELECT COUNT(*) FROM purchase_data) - (SELECT COUNT(*)
FROM purchase_data_clean) AS duplicates_removed;
```

original_count	deduplicated_count	duplicates_removed
21864	21825	39

➔ Duplicates are removed from clean table

Issue 7: duplicate order_id

After investigating the issue, I noticed that the doble order_id is not due to different products sold in one order, but different user-ids. This violates fundamental database integrity principles.

-- 1. Count how many duplicates have different users

```sql
SELECT
    COUNT(DISTINCT order_id) AS problematic_orders_count
FROM (
    SELECT order_id
    FROM purchase_data
    GROUP BY order_id
    HAVING COUNT(DISTINCT user_id) > 1
) t;
```

Metric	Value
problematic_orders_count	108

➔ All duplicate order_ids are problematic

Because I can not determine which is the true user_id and the cause for this issue, I will flag the detected rows.

-- Add flag column to your existing table

```sql
ALTER TABLE purchase_data_clean ADD COLUMN data_quality_flag
VARCHAR(50);
```

-- Mark problematic records

```sql
UPDATE purchase_data_clean
SET data_quality_flag = 'order_user_conflict'
WHERE order_id IN (
    SELECT order_id
    FROM purchase_data
    GROUP BY order_id
    HAVING COUNT(DISTINCT user_id) > 1
);
```

UPDATE 216 → 2*108 check

Next steps:

Perform technical investigation

Initiate business process review

Develop exclusion rules for critical reports

Monitoring

Issue 8: Rename product name to standardize

```sql
UPDATE purchase_data_clean
SET product_name = '27in 4K gaming monitor'
WHERE product_name = '27inches 4k gaming monitor';
```

Issue 10: empty country region

-- Check country region

```sql
SELECT
    cr.region,
    COUNT(*) AS order_count
FROM purchase_data pd
LEFT JOIN country_region cr ON pd.country_code =
cr.country_code
GROUP BY cr.region
ORDER BY order_count DESC;
```

-- Set empty to Null

```sql
UPDATE country_region
SET region = NULL
WHERE region = '' OR region LIKE '%null%';
```

-- Set Null to unknown

```sql
UPDATE country_region
SET region = 'unknown'
WHERE region IS NULL OR region = 'X.x';
```

Issue 11: country_code in purchase_data with no matching partner in country_region

-- Get unmatched country codes and their frequency

```sql
SELECT
    pd.country_code,
    COUNT(*) AS unmatched_count
FROM purchase_data_clean pd
LEFT JOIN country_region cr ON pd.country_code =
cr.country_code
WHERE cr.country_code IS NULL
GROUP BY pd.country_code
ORDER BY unmatched_count DESC;
```

Result	Value
--	37
EU	3
AP	2

➔ Former empty values and unknown country codes → needs to be addressed to the stakeholders

Issue 12: Shipping date before Purchase date

```sql
UPDATE purchase_data_clean
SET data_quality_flag =
    CASE
        WHEN ship_ts < purchase_ts THEN 'ship_before_purchase'
        WHEN data_quality_flag IS NOT NULL THEN
data_quality_flag || ';ship_before_purchase'
        ELSE data_quality_flag
    END
WHERE ship_ts < purchase_ts;
```

Next steps:

Perform technical investigation

Initiate business process review

Develop exclusion rules for critical reports (time to shipment)

Monitoring

Issue 13: price 0

```sql
UPDATE purchase_data_clean
SET data_quality_flag =
    CASE
        WHEN usd_price = 0 THEN 'price is cero'
        WHEN data_quality_flag IS NOT NULL THEN
data_quality_flag || ';price is cero'
        ELSE data_quality_flag
    END
WHERE usd_price = 0;
```