First of all, I mention all the nonconformities I have found, then based on them, I propose new methods for a better and more robust ETL.

For each of the nonconformities, I develop a flagging field to indicate the suspicious row in the ETL:

## Abrechnung_rechnungen:

- I have assumed a prepaid system, then payment date must be greater than or equal to invoice date:

```
SELECT count(*) count --50
FROM public.abrechnung_rechnungen i
where i.redatum > i.zahlungsdatum
and i.zahlungsdatum is not null

SELECT count(*) count --1457
FROM public.abrechnung_rechnungen i
where i.redatum < i.zahlungsdatum
and i.zahlungsdatum is not null
```

- Detecting Inferred data

```
select count(*) count --0
from public.abrechnung_rechnungen i
where (i.kdnr is null or i.redatum is null)
```

- For all successful payments, there must be at least one position and sum of the amount of the positions related to one invoice must match exactly to net amount of the linked invoice. Otherwise, the invoice must get flagged as anomaly.

```
WITH position_data AS (
  SELECT
     p.reid,
     p.kdnr,
     SUM(p.nettobetrag)::text AS sum_amount
  FROM public.abrechnung_positionen p
  GROUP BY p.reid, p.kdnr
)
SELECT COUNT(DISTINCT i.renummer) count --47
FROM public.abrechnung_rechnungen i
inner JOIN position_data p
  ON (i.renummer = p.reid AND i.kdnr = p.kdnr)
WHERE i.zahlungsdatum IS NOT null
and i.zahlungsbetragbrutto <> 0
```

```
AND i.summenetto <> (REPLACE(REPLACE(p.sum_amount, '$', ''), ',', '')::numeric);
```

- Unique invoices that reach more than one customer are a serious financial issue: This could be the reason that some revenues never gets shown in the dashboards.

```
select i.renummer, count(distinct p.kdnr) count --1237
FROM public.abrechnung_rechnungen i
Inner join public.abrechnung_positionen p
        on (i.renummer = p.reid )
group by i.renummer
having count(distinct p.kdnr) > 1
```

- Orphan invoices : Already covered in data-analysis section.

```
SELECT count(*) count  --1011
FROM public.abrechnung_rechnungen i
left join public.abrechnung_positionen p
        on (i.renummer = p.reid and i.kdnr = p.kdnr)
where p.reid is null
```

- Rounding gross amounts that end up in financial conflicts:

```
SELECT count(*) count --88
FROM public.abrechnung_rechnungen i
where i.zahlungsbetragbrutto <> (i.summenetto  + ((i.summenetto * i.mwstsatz)/100))
-i.summenebenkosten
and i.zahlungsdatum is not null
and i.zahlungsbetragbrutto  <> 0
```

- One suspicious invoice with 0 gross amount but the zahlungsdatum, if the customer has had a voucher or gift card, it must have been flagged or added to table:

```
SELECT count(*) count --1
FROM public.abrechnung_rechnungen i
where i.zahlungsdatum is not null
and i.zahlungsbetragbrutto = 0
```

- Out of range dates:

```
SELECT COUNT(*) AS count --0
FROM public.abrechnung_rechnungen i
WHERE NOT (
```

```
  (i.redatum BETWEEN '1990-01-01' AND CURRENT_DATE)
  OR
  (i.zahlungsdatum BETWEEN '1990-01-01' AND CURRENT_DATE)
);
```

## Abrechnung_positionen:

- Detecting inferred data

```
SELECT count(*) count --4
FROM public.abrechnung_positionen p
where (p.reid is null or p.kdnr is null or p.verdatum is null)
```

- Out of range dates:

```
SELECT count(*) count --13
FROM public.abrechnung_positionen p
where not (p.verdatum BETWEEN '1990-01-01' AND CURRENT_DATE)
```

- Orphan positions:

```
select count(distinct p.id) count --70886
FROM public.abrechnung_positionen p
left join public.abrechnung_rechnungen i
        on (i.renummer = p.reid and i.kdnr = p.kdnr)
where i.renummer is null
```

- Conflicting dates with invoices:

  Both scenarios exist and each should have a valid scenario which needs meeting with backend developer and business owner colleagues:

```
select count(distinct p.id) count --42891
FROM public.abrechnung_positionen p
inner join public.abrechnung_rechnungen i
        on (i.renummer = p.reid and i.kdnr = p.kdnr)
where (p.verdatum < i.redatum or p.verdatum < i.zahlungsdatum)
select count(distinct p.id) count --18922
FROM public.abrechnung_positionen p
inner join public.abrechnung_rechnungen i
        on (i.renummer = p.reid and i.kdnr = p.kdnr)
where (p.verdatum > i.redatum or  p.verdatum > i.zahlungsdatum)
```

# ETL Fix:

All these nonconformities can be flagged as anomalies or suspicious data that need future alerts, anomaly dashboards and consideration in the more constraint checks at the application side.

- Using switch case condition in the source query of the extract or transformation part of ETL to detect these anomalies:

- Sending the suspicious data in SSIS or other ETL tools to another data flow to store them in different tables for later analysis, consideration or informing the backend developers for more constraint checks.

- Regarding nonconformities that have meaning on aggregated measures, daily jobs can run to find the conflicts and flag them for later consideration and conflict dashboards.

- Tableau subscriptions on serious anomalies like financial measures and invoice conflicts within different customers can be set to be sent every day on email.

- Out of range dates can be imputed:

  They can be substituted by other equivalent dates of the linked position or invoices. Also, the very old outdated range (e.g. 1600) can be replaced by min range of valid date (1990). Furthermore, Null dates can also be imputed through the average of the mean and max of the valid date range. Meetings with product owners can lead to better imputation.

- Defining status on the invoice table could remove alot of unclear rows and nonconformities. At the moment, given the limited knowledge I have on the business, I can define a Status dimension with three values:
    - Payment Pend:
      Invoices that have linked positions, but don't have zahlungsdatum or zahlungsbetragbrutto. They can be marked as Payment Failed after a while when the zahlungsdatum does not get filled. This should also be discussed by backend and business owner guys.

    - Payment Succeed: Invoices that have linked positions and have zahlungsdatum along with a valid zahlungsbetragbrutto amount.

    - Payment Failed: Invoices that never ever get paid and have a null zahlungsdatum and 0 zahlungsbetragbrutto amount.

    - With Invoices that have been assigned to more than one customer, accept the positions of the customer that have a closer date range to the invoice and a closer financial difference on the sum of nettobetrag and summenetto. This is just for abit of data cleaning and this nonconformity must get into serious consideration with the OLTP database owners.

# Downstream Reporting Impact:

By implementing the validations and flagging mechanisms:

- Reports will exclude or clearly mark suspicious or nonconforming data, increasing trust in metrics shown on Tableau dashboards.
- Revenue attributed to placeholder media or shared across multiple customers will no longer pollute content-level or client-level performance KPIs.
- Dashboards can now include anomaly overviews, like:
    - Invoices without matching positions.
    - Payments that don’t reconcile with invoice amounts.
    - Invoices that are shared across multiple customers.
- Subscriptions in Tableau can be configured to alert finance teams daily of these red-flag scenarios.

- ETL paths can separate suspicious records into staging or audit tables, enabling focused resolution workflows without blocking core pipeline performance.

# Business conversations to initiate

- **With Backoffice**:
    - Clarify whether placeholder media (e.g., Bildnummer = 100000000) are business-acceptable, or should be prevented at the source.
    - Agree on imputation rules for missing or invalid dates.
    - Validate whether multi-customer invoices are legal/valid or a violation.
- **With Finance**:
    - Confirm logic for calculating expected ZahlungsbetragBrutto, including whether Summenebenkosten or tax exemptions apply.
    - Define how long an invoice can remain in a Payment Pend state before being flagged as failed.
    - Decide on acceptable rounding thresholds and tolerances.
- **With Backend Developers / Data Owners**:
    - Embed status tracking directly in the application layer (PaymentStatus column).
    - Improve constraint handling and real-time validation during invoice or position creation.
    - Define canonical business rules in a shared data contract (e.g., cannot insert position with null KdNr or ReId).

After that, the data engineer team effort should focus on gaining trust from people in the company to use the data and ensure non conformities have been gets closed to being zero.