FactoryTwin FAQ List

**Do not disseminate this document to anybody outside of this project**

The FAQs below are a small subset of the overall FAQ list. However, to keep the project within a reasonable scope and focus more on the architecture than every possible question, we are currently limiting the question list to those focused on a key use case: understanding demand, on-time delivery, and how to improve on-time delivery by assessing the root cause(s) of late deliveries. There are three primary classes of questions which you should consider, **in descending order of priority**:

1. Descriptive Questions: These questions are queries for information which can be derived directly from the data or existing software APIs without further analysis. Many of these questions will have a simple, objective answer, e.g. questions 1-3. Other questions may require some minor reasonable assumptions to be made on the part of the model; for example, question 5 may require the model to decide what constitutes a “demand spike.”
2. Judgement Questions: Here, the user is asking for the agent’s opinion. It will still have to query data, as for a descriptive question, but it will also have to provide some further analysis and present its conclusion and reasoning to the user. The user is not necessarily asking for a solution (yet) but instead asking for direction and confirmation.
3. Suggestion/Advice Questions: These are likely to be the most challenging questions for the agent to answer in a satisfactory manner. Here, the user is asking for broader guidance on a more fundamental problem. In these cases, it is likely going to be useful for the agent to direct the user to parts of the software which will assist in answering the question.

Late Delivery Root Cause Analysis FAQs

1. DESCRIPTIVE QUESTIONS:
2. What is my projected revenue over the next <time period>?
3. Which parts/part families have the highest contribution margin?
4. Which parts/part families have had the worst on-time delivery in the last <time period>?
5. Which purchased materials have had the worst on-time arrival in the last <time period>?
6. For which customer do I have the worst on-time delivery?
7. When are my biggest demand spikes?
8. How much (value/quantity/no. of orders) overdue do I currently have?
9. Which parts/part families have above average demand in the next <time period>?
10. What is the most common root cause of my late deliveries overall?
11. What is the most common root cause for <part/part family> to be delivered late?
12. JUDGEMENT QUESTIONS:
13. Can I make my due date for <line item>?
14. Why was <line item> delivered late?
15. Which parts/part families are most likely to be delivered late in the future?
16. Is my system lead time for <supplier/part> accurate as compared to demonstrated lead times?
17. Should I modify my system lead times?
18. SUGGESTION/ADVICE QUESTIONS:
19. Which parts/part families should I focus on improving on-time delivery for?
20. How can I improve on-time delivery for <part/part family/customer>?
21. How can I address the root causes of a late delivery?
22. How can I fix/mitigate <specific root cause> of a late delivery?
23. What should my new system lead time be for <part>?

Some things to think about:

1. How will the LLM handle a situation in which neither the database nor the software has adequate information to answer the prompt?
2. What sort of external APIs/tools (i.e., not the model itself) might the LLM use to answer certain questions? You may assume that each chart in the software has an available API which accepts CSV input and produces CSV output.
3. In what situations is it better or worse to use an external API vs. only using output from the LLM itself? How will the agent decide which is more appropriate for a given prompt?
4. If the agent executes a query which fails, how will the agent handle it?
5. Does the agent require any “helper agents?”
6. Is the agent “stateful,” i.e., does it maintain memory of past interactions? If so, how long should its memory be? How does it store its memories?
7. How will the agent’s API handle multiple concurrent users? What constitutes one “instance” of the agent? If the agent is allowed to have multiple instances, to what extent can information be shared and retrieved across instances?
8. What can be done to mitigate risk of hallucination on the part of the agent?
9. Will the model require intermittent fine-tuning? What additions to the architecture might be needed to support it?
10. -- Get all customers

SELECT \* FROM customer

1. -- Get all vendors in a specific state

SELECT \* FROM vendor WHERE "addr\_ste" = 'TX';

1. -- Join sales orders with customer info

SELECT s.order\_no, s.order\_date, c.name, c.addr\_city

FROM sales\_mst s

JOIN customer c ON s.cust = c.id;

1. -- Parts with their vendor and unit cost

SELECT P.p\_code, V.name AS vendor\_name, VP.unit\_cost

FROM vend\_part VP

JOIN vendor V ON VP.vendor = V.id

JOIN part\_mst P ON VP.part = P.id;

1. -- Total number of sales per customer

SELECT C.name, COUNT(S.order\_no) AS total\_orders

FROM customer C

JOIN sales\_mst S ON S.cust = C.id

GROUP BY C.name

order by COUNT(S.order\_no) desc;

1. -- Average unit price of each part

SELECT p\_code, AVG(unit\_price) AS avg\_price

FROM part\_mst

GROUP BY p\_code

order by AVG(unit\_price);

1. -- Jobs released in last 30 days

SELECT \* FROM job\_mst where date(job\_rls)>= date\_sub(curdate() , interval 30 day);

1. -- Purchase orders due next week

SELECT \* FROM purchase\_line where week(due) = week(date\_add(curdate() , interval 7 day));

1. -- All open jobs

SELECT \* FROM JOB\_MST WHERE job\_stat = 'Q';

1. -- All sales lines with ‘COMPLETE’ status

SELECT \* FROM sales\_line WHERE order\_stat = 'C';

1. -- Get purchase line for specific order and line

SELECT \* FROM purchase\_line

WHERE order\_no = 'PO-5436934';

1. -- Customers with more than 3 orders

SELECT name FROM CUSTOMER

WHERE id IN (

SELECT cust FROM SALES\_MST

GROUP BY cust HAVING COUNT(order\_no) > 3

);

1. **List Sales Orders with Customer Names**SELECT s.order\_no,

s.order\_date,

c.name AS customer\_name,

c.addr\_city AS customer\_city

FROM

SALES\_MST s

JOIN

CUSTOMER c ON s.cust = c.id;

1. **Show Details of Parts on a Specific Sales Order  
     
   SELECT**

**sl.order\_no,**

**sl.line\_no,**

**p.p\_code AS part\_code,**

**p.unit\_meas,**

**sl.qty AS quantity\_ordered,**

**sl.unit\_price AS price\_on\_order,**

**p.unit\_price AS current\_part\_price,**

**sl.order\_stat AS line\_status**

**FROM**

**SALES\_LINE sl**

**JOIN**

**PART\_MST p ON sl.part = p.id**

**where order\_no = "SO-1490003";**

1. **Calculate Total Quantity Sold for Each Part  
     
   SELECT**

**p.p\_code,**

**p.id AS part\_id,**

**SUM(sl.qty) AS total\_quantity\_sold**

**FROM**

**SALES\_LINE sl**

**JOIN**

**PART\_MST p ON sl.part = p.id**

**WHERE sl.order\_stat = 'C'**

**GROUP BY**

**p.p\_code, p.id**

**ORDER BY**

**total\_quantity\_sold DESC;**

1. **Calculate Total Value of Purchases per Vendor  
     
   SELECT**

**v.name AS vendor\_name,**

**COUNT(DISTINCT pl.order\_no) AS number\_of\_pos,**

**SUM(pl.qty \* pl.unit\_cost) AS total\_purchase\_value**

**FROM**

**PURCHASE\_LINE pl**

**JOIN**

**PURCHASE\_MST pm ON pl.order\_no = pm.order\_no**

**JOIN**

**VENDOR v ON pm.vendor = v.id**

**GROUP BY**

**v.name**

**ORDER BY**

**total\_purchase\_value DESC;  
  
  
17. Find Customers and Vendors Located in the Same State  
  
SELECT**

**c.name AS customer\_name,**

**v.name AS vendor\_name,**

**c.addr\_ste AS state**

**FROM**

**CUSTOMER c**

**JOIN**

**VENDOR v ON c.addr\_ste = v.addr\_ste**

**ORDER BY**

**c.addr\_ste, c.name, v.name;**

1. **Compare Current Stock vs. Total Purchased vs. Total Sold (Simplified)  
     
   SELECT**

**p.id,**

**p.p\_code,**

**p.curr\_stock AS current\_stock\_on\_hand,**

**COALESCE(pp.total\_purchased, 0) AS total\_units\_purchased,**

**COALESCE(ps.total\_sold, 0) AS total\_units\_sold,**

**(p.curr\_stock + COALESCE(pp.total\_purchased, 0) - COALESCE(ps.total\_sold, 0)) AS calculated\_stock\_balance**

**FROM**

**PART\_MST p**

**LEFT JOIN**

**PartPurchases pp ON p.id = pp.part**

**LEFT JOIN**

**PartSales ps ON p.id = ps.part**

**ORDER BY**

**p.p\_code;  
  
  
18. List Parts Supplied by a Specific Vendor:** SELECT

v.id,

v.name AS vendor\_name,

p.p\_code AS part\_code,

vp.unit\_cost AS vendor\_unit\_cost,

vp.part\_lt AS vendor\_lead\_time

FROM

VEND\_PART vp

left JOIN

VENDOR v ON vp.vendor = v.id

left JOIN

PART\_MST p ON vp.part = p.id

where v.id = 7

ORDER BY p.p\_code;

1. **Top 5 Customers by Sales Value in the Last 6 Months**SELECT

c.id AS customer\_id,

c.name AS customer\_name,

SUM(sl.qty \* sl.unit\_price) AS total\_spent

FROM

CUSTOMER c

JOIN

SALES\_MST s ON c.id = s.cust

JOIN

SALES\_LINE sl ON s.order\_no = sl.order\_no

WHERE

s.order\_date >= DATE\_SUB(CURDATE(), INTERVAL 6 MONTH)

AND sl.order\_stat <> 'X' -- Exclude cancelled lines

GROUP BY

c.id, c.name

ORDER BY

total\_spent DESC

LIMIT 5;  
  
19. **Parts Below Preferred Order Quantity**SELECT

id,

p\_code,

curr\_stock,

pref\_order\_qty

FROM

PART\_MST

WHERE

curr\_stock < pref\_order\_qty

AND procurement <> 'S' -- Exclude Service parts

ORDER BY

(pref\_order\_qty - curr\_stock) DESC; -- Order by largest deficit

1. **Monthly Sales Revenue Trend:**SELECT

DATE\_FORMAT(s.order\_date, '%Y-%m') AS sales\_month,

round(SUM(sl.qty \* sl.unit\_price),2) AS monthly\_revenue

FROM

SALES\_MST s

JOIN

SALES\_LINE sl ON s.order\_no = sl.order\_no

WHERE

sl.order\_stat <> 'X' -- Exclude cancelled lines

GROUP BY

sales\_month

ORDER BY

sales\_month;

1. **Direct Components for a Manufactured Part (Bill of Materials)**SELECT

mp.p\_code AS manufactured\_part\_code,

cp.p\_code AS component\_part\_code,

bom.qty\_req AS quantity\_required,

cp.unit\_meas AS component\_unit

FROM

BILL\_OF\_M bom

JOIN

PART\_MST mp ON bom.m\_part = mp.id

JOIN

PART\_MST cp ON bom.c\_part = cp.id

WHERE

bom.m\_part = 5638;

1. **Total Component Quantity Required for Open Production Jobs**SELECT

bom.c\_part AS component\_part\_id,

p.p\_code AS component\_part\_code,

SUM(j.qty \* bom.qty\_req) AS total\_required\_for\_open\_jobs

FROM

JOB\_MST j

JOIN

BILL\_OF\_M bom ON j.part = bom.m\_part -- Join Job's manufactured part to BOM

JOIN

PART\_MST p ON bom.c\_part = p.id -- Get component part details

WHERE

j.job\_stat IN ('O', 'Q') -- Only consider Open or Queued jobs

GROUP BY

bom.c\_part, p.p\_code

ORDER BY

total\_required\_for\_open\_jobs DESC;

1. **Nonconformance Rate per Part (Simplified: NC Qty / Qty Purchased)**

WITH PartPurchasesReceived AS (

SELECT

pl.part,

SUM(pl.qty) AS total\_received

FROM PURCHASE\_LINE pl

WHERE pl.order\_stat = 'C'

GROUP BY pl.part

),

PartNonconformance AS (

SELECT

pl.part,

SUM(nc.qty) AS total\_nonconforming

FROM NONCONFORM\_MST nc

JOIN PURCHASE\_MST pm ON nc.po\_no = pm.order\_no

JOIN PURCHASE\_LINE pl ON pm.order\_no = pl.order\_no AND nc.job IS NULL GROUP BY pl.part

)

SELECT

p.id AS part\_id,

p.p\_code,

COALESCE(ppr.total\_received, 0) AS total\_received,

COALESCE(pnc.total\_nonconforming, 0) AS total\_nonconforming,

CASE

WHEN COALESCE(ppr.total\_received, 0) > 0 THEN

(COALESCE(pnc.total\_nonconforming, 0) / ppr.total\_received) \* 100

ELSE 0

END AS nonconformance\_rate\_percent

FROM

PART\_MST p

LEFT JOIN

PartPurchasesReceived ppr ON p.id = ppr.part

LEFT JOIN

PartNonconformance pnc ON p.id = pnc.part

WHERE

p.procurement = 'B'

ORDER BY

nonconformance\_rate\_percent DESC;

1. **Customers Who Haven't Ordered in Over a Year**SELECT

c.id,

c.name,

MAX(s.order\_date) AS last\_order\_date

FROM

CUSTOMER c

LEFT JOIN

SALES\_MST s ON c.id = s.cust AND s.order\_date >= DATE\_SUB(CURDATE(), INTERVAL 1 YEAR)

WHERE

s.order\_no IS NULL

GROUP BY

c.id, c.name

ORDER BY

c.name;

1. **Parts Purchased from Multiple Vendors with Cost Comparison**SELECT

p.id AS part\_id,

p.p\_code,

COUNT(DISTINCT vp.vendor) AS number\_of\_vendors,

MIN(vp.unit\_cost) AS min\_vendor\_cost,

MAX(vp.unit\_cost) AS max\_vendor\_cost,

(MAX(vp.unit\_cost) - MIN(vp.unit\_cost)) AS cost\_difference

FROM

PART\_MST p

JOIN

VEND\_PART vp ON p.id = vp.part

GROUP BY

p.id, p.p\_code

HAVING

COUNT(DISTINCT vp.vendor) > 1

ORDER BY

cost\_difference DESC, p.p\_code;