BAB III
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Posisi magang sebagai *Data Analyst* merupakan jabatan fungsional yang berkoordinasi dengan Kepala Departemen MIS & Project Support. Selama kegiatan magang, Kepala Departmen MIS & Project Support berperan sebagai *supervisor*. Kepala Departmen MIS & Project Support memberikan tugas, proyek, arahan serta pengawasan terhadap hasil kerja. *Data Analyst* memiliki tugas untuk mengolah data perusahaan dan menganalisis dengan tujuan menciptakan *insight* terkait proses bisnis. Proyek yang dilakukan oleh *intern* akan menjadi arsip bagi divisi IT yang dapat digunakan ketika diperlukan di masa depan. Perkembangan proyek dilaporkan secara langsung setiap beberapa hari di kantor.

Berikut merupakan tabel realisasi magang yang berisi tentang tugas, koordinasi, serta tanggal pelaksanaannya.

Tabel 3. 1 Tabel Realisasi Magang

No	Kegiatan	Koordinasi	Pelaksanaan
1	**Pengenalan Proses Bisnis (17 Januari – 30 Januari)**		
1.1	Membuat diagram proses bisnis	Kepala MIS & Project Support	Minggu 1-2
No	**Kegiatan**	**Koordinasi**	**Pelaksanaan**
2	**Masa belajar mandiri (1 – 29 Februari)**		
2.1	Belajar Mandiri (Platform Leetcode)	Mandiri	Minggu 3-6
2.2	Belajar Mandiri (Platform Hackerrank)	Mandiri	Minggu 3-6
No	**Kegiatan**	**Koordinasi**	**Pelaksanaan**
3	**Proyek Analisis Data Routing (1 Maret – 17 Mei)**		
3.1	Mempelajari Dataset	Kepala MIS & Project Support	Minggu 7
3.2	Importing dan Merging Dataset	Kepala MIS & Project Support	Minggu 8
3.3	Rekomendasi setup time & runtime master data	Kepala MIS & Project Support	Minggu 9
3.4	Penentuan wait time jurnal	Kepala MIS & Project Support	Minggu 10
3.5	Penentuan waktu produksi item dan BOM Tree	Kepala MIS & Project Support	Minggu 11 - 13
3.6	Rekomendasi alternatif work center	Kepala MIS & Project Support	Minggu 14

3.7	Rekomendasi alternatif material	Kepala MIS & Project Support	Minggu 15 - 17

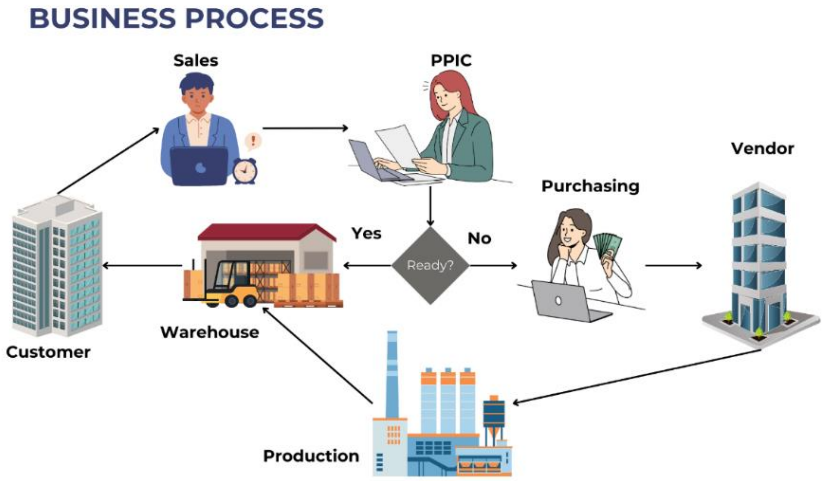
3.2 Tugas dan Uraian Kerja Magang

Proyek yang dilakukan oleh *Data Analyst intern* PT. Sagatrade Murni adalah menganalisis data *routing* produksi dari jurnal/ data aktual dan membandingkannya dengan *master data*. *Routing* adalah urutan proses yang perlu dilakukan untuk memproduksi sebuah produk. Analisis dilakukan dengan menggunakan bahasa pemrograman Python. Jupyter Notebook adalah lingkungan komputasi interaktif yang memungkinkan pengguna membuat dan berbagi dokumen yang berisi kode langsung, persamaan, visualisasi, dan teks naratif [11]. Jupyter Notebook telah mendapatkan popularitas yang signifikan di berbagai bidang ilmu data, pembelajaran mesin, dan pendidikan ilmu komputer [12]. Jupyter Notebook memungkinkan pengguna menggabungkan eksekusi kode dengan penjelasan naratif, menjadikannya alat serbaguna untuk analisis dan penelitian data [13].

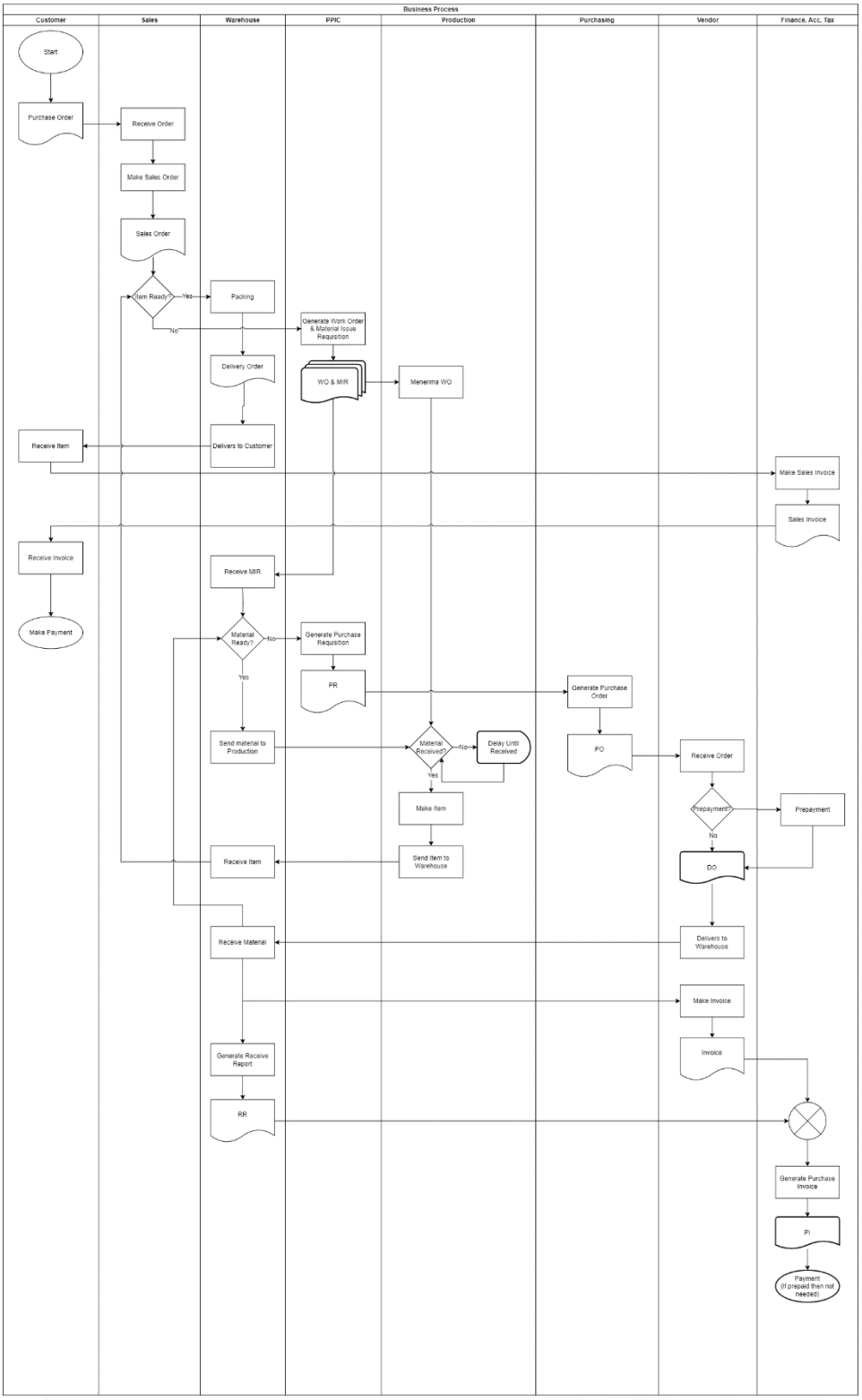
3.3.1 Minggu 1 - 2: Pengenalan Proses Bisnis

Proses bisnis PT. Sagatrade Murni berawal dari *Purchase Order* dari *customer* yang akan diterima oleh departemen Sales. Kemudian departemen *Sales* akan membuat *Sales Order* terkait barang yang dipesan. Jika barang tersedia, maka barang akan dikemas oleh departemen *Warehouse*, dan dikirimkan kepada *customer* beserta dengan *Delivery Order*. Setelah barang diterima, departemen *Finance, Accounting, and Tax* akan membuat *Sales Invoice* yang berisi tagihan kepada *customer* yang kemudian dibayar. Jika barang tidak tersedia, Maka departemen *Production Planning and Inventory Control* (PPIC) akan membuat *Work Order* yang dikirim pada departemen *Production* serta *Material Issue Requisition* (MIR), yang akan dikirimkan pada departemen *Warehouse*. Jika material tersedia, maka akan dikirimkan pada departemen *Production*. Jika material tidak tersedia, maka departemen PPIC akan membuat *Purchase Requisition* yang akan dikirimkan pada departemen *Purchasing*, kemudian departemen *Purchasing* akan membuat

Purchase Order, yang dikirimkan pada *vendor*. Pembayaran dapat dilakukan sebelum atau sesudah barang diterima. Setelah barang diterima, departemen *warehouse* akan membuat *Receive Report* dan *vendor* akan mengirimkan *Invoice* pada departemen *Finance, Accounting, and Tax*. Kemudian akan dibuat *Purchase Invoice* dan melakukan pembayaran (jika belum dibayar sebelumnya). Proses bisnis PT. Sagatrade Murni digambarkan secara sederhana dalam Gambar 3.1 dan flowchart proses bisnis digambarkan dalam Gambar 3.2. Departemen IT tidak termasuk dalam diagram tersebut oleh karena departemen IT merupakan salah satu departemen pendukung, yang memfasilitasi infrastruktur IT dan program yang digunakan selama proses bisnis.



Gambar 3. 1 Diagram Proses Bisnis Sales



Gambar 3. 2 Flowchart Proses Bisnis Sales

3.3.2 Minggu 3 - 6: Pembelajaran Mandiri

Pembelajaran mandiri dilakukan melalui *online platform* Hackerrank dan Leetcode. Pembelajaran dilakukan dengan mengerjakan soal-soal SQL dan Python yang disediakan oleh platform tersebut. Dalam contoh soal di Gambar 3.3, terdapat 2 tabel, yakni *Employee* dan *Department*. Tabel *Employee* terdiri dari kolom *id, name, salary,* dan *departmentId,* dengan *id* sebagai *primary key*. Sedangkan tabel *Department* terdiri dari kolom *id*, dan *name*, dengan *id* sebagai *primary key*. *DepartmentId* merupakan *foreign key* tabel *Employee* terhadap *id* tabel *Department*. Hasil yang diharapkan adalah 3 pegawai dengan pendapatan tertinggi yang unik di masing-masing departemen. Query yang dapat menghasilkan *output* tersebut adalah:

```
WITH MaxSalary AS(
SELECT d.name as Department, e.name as Employee, e.salary as salary,
DENSE_RANK() OVER (PARTITION BY departmentId ORDER BY salary DESC)
AS rn
FROM Employee e JOIN Department d ON e.departmentId = d.id
)
SELECT Department, Employee,salary
FROM MaxSalary
WHERE rn <= 3
```

Dibuat *Common Table Expression* bernama MaxSalary yang merupakan terdiri dari kolom gabungan kedua tabel melalui INNER JOIN, yakni kolom *Department* (*name* tabel Department**)**, *name* (*name* tabel Employee**)**, *salary* serta rn yang merupakan ranking dari *salary* setiap department diurutkan dari yang paling tinggi ke paling rendah. Fungsi DENSE_RANK() akan memberikan ranking yang sama jika ada beberapa orang yang memiliki *salary* yang sama. Misalnya jika ada 2 orang dengan *salary* 90000 dan *salary* merupakan yang paling tinggi, maka kedua orang tersebut akan diberikan ranking 1. Kalimat SELECT me-*return* kolom *Deparment, Employee*, dan *salary* dimana rn kurang atau sama dengan 3. Solusi tersebut tercantum dalam Gambar 3.4.

Table: `Employee`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| id           | int     |
| name         | varchar |
| salary       | int     |
| departmentId | int     |
+--------------+---------+
id is the primary key (column with unique values) for this table.
departmentId is a foreign key (reference column) of the ID from the Department table.
Each row of this table indicates the ID, name, and salary of an employee. It also
contains the ID of their department.
```

Table: `Department`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
+-------------+---------+
id is the primary key (column with unique values) for this table.
Each row of this table indicates the ID of a department and its name.
```

A company's executives are interested in seeing who earns the most money in each of the company's departments. A **high earner** in a department is an employee who has a salary in the **top three unique** salaries for that department.

Write a solution to find the employees who are **high earners** in each of the departments.

Return the result table **in any order**.

The result format is in the following example.

Example 1:

Input:
Employee table:

id	name	salary	departmentId
1	Joe	85000	1
2	Henry	80000	2
3	Sam	60000	2
4	Max	90000	1
5	Janet	69000	1
6	Randy	85000	1
7	Will	70000	1

Department table:

id	name
1	IT
2	Sales

Output:

Department	Employee	Salary
IT	Max	90000
IT	Joe	85000
IT	Randy	85000
IT	Will	70000
Sales	Henry	80000
Sales	Sam	60000

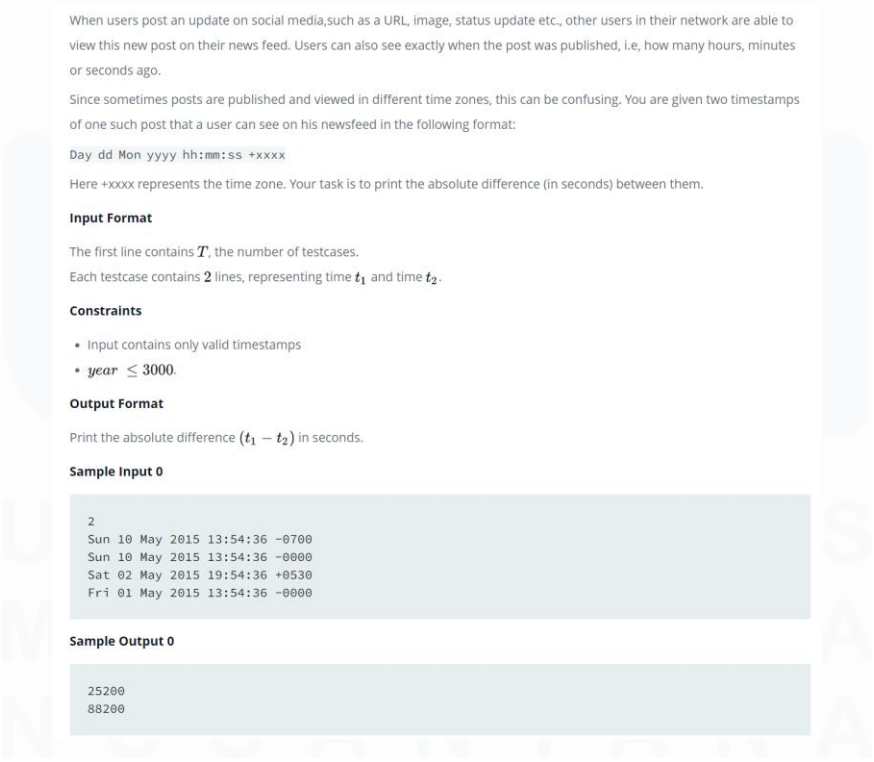
Explanation:
In the IT department:
- Max earns the highest unique salary
- Both Randy and Joe earn the second-highest unique salary
- Will earns the third-highest unique salary

In the Sales department:
- Henry earns the highest salary
- Sam earns the second-highest salary
- There is no third-highest salary as there are only two employees

Gambar 3. 3 Contoh 1 Soal SQL

Gambar 3. 4 Solusi Soal 1 SQL

Dalam contoh soal di Gambar 3.5, terdapat latar belakang masalah, yakni ketika seseorang mengunggah ke media social, orang lain dapat melihat kapan unggahan tersebut dibuat. Namun, oleh karena perbedaan zona waktu, hal tersebut dapat menjadi membingungkan. Maka dari itu diberikan dua buah *timestamp* dengan format Day dd Mon yyyy hh:mm:ss +xxx, dengan +xxx sebagai zona waktu dan tipe data *string*. Hasil yang diharapkan adalah perbedaan waktu antara dua waktu tersebut dalam satuan detik. Solusi dari masalah tersebut adalah dengan menggunakan fungsi **strptime()**, yang mengubah *string* menjadi format *datetime*. Format yang digunakan dalam fungsi tersebut adalah '%a %d %b %Y %H:%M:%S %z'. %a artinya nama hari yang disingkat (Mon,Tue), %d artinya tanggal (1-31), %b artinya bulan yang disingkat (Jan,Feb), %Y artinya tahun termasuk abad (1991,2000), %H:%M:%S artinya jam, menit, dan detik, dan %z adalah UTC *offset*. Setelah mengubah format, digunakan fungsi str(int(abs().total_seconds)) untuk menentukan perbedaan dalam satuan detik. Solusi tersebut tercantum dalam Gambar 3.6.



When users post an update on social media,such as a URL, image, status update etc., other users in their network are able to view this new post on their news feed. Users can also see exactly when the post was published, i.e, how many hours, minutes or seconds ago.

Since sometimes posts are published and viewed in different time zones, this can be confusing. You are given two timestamps of one such post that a user can see on his newsfeed in the following format:

`Day dd Mon yyyy hh:mm:ss +xxxx`

Here +xxxx represents the time zone. Your task is to print the absolute difference (in seconds) between them.

Input Format

The first line contains T, the number of testcases.
Each testcase contains 2 lines, representing time t_1 and time t_2.

Constraints

- Input contains only valid timestamps
- $year \leq 3000$.

Output Format

Print the absolute difference $(t_1 - t_2)$ in seconds.

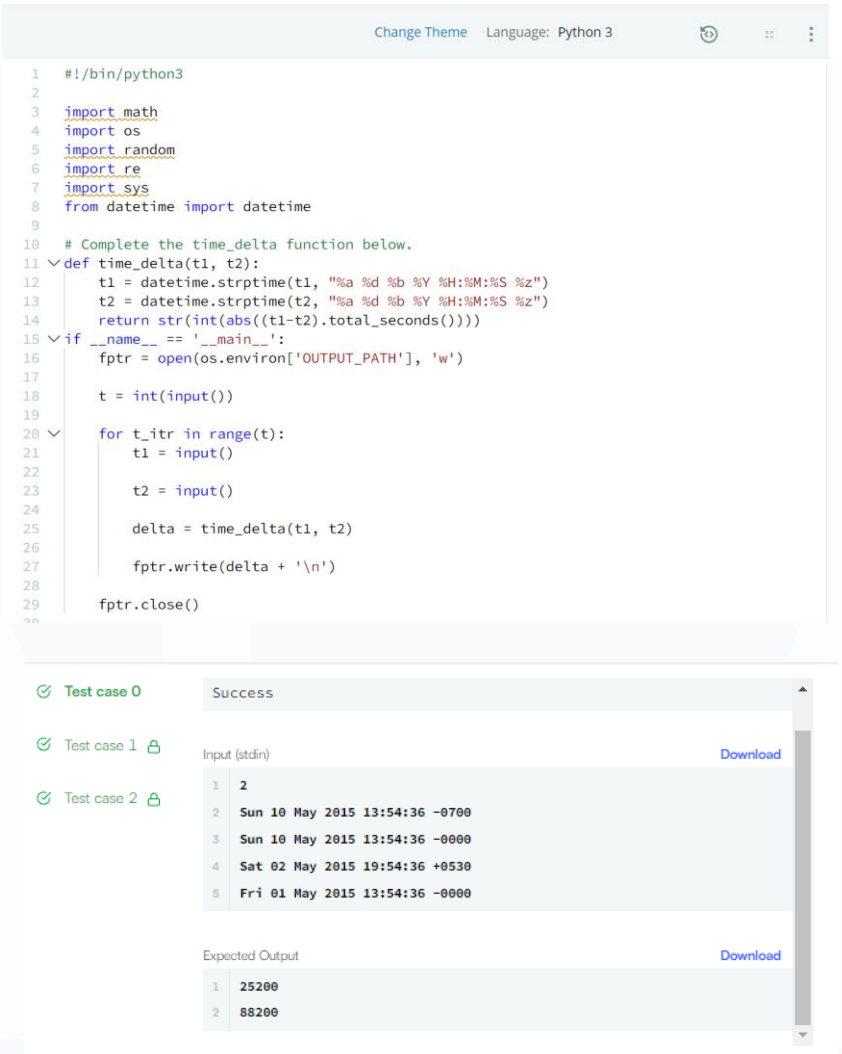
Sample Input 0

```
2
Sun 10 May 2015 13:54:36 -0700
Sun 10 May 2015 13:54:36 -0000
Sat 02 May 2015 19:54:36 +0530
Fri 01 May 2015 13:54:36 -0000
```

Sample Output 0

```
25200
88200
```

Gambar 3. 5 Contoh Soal Python



Gambar 3. 6 Solusi Soal Python

Dalam contoh soal di Gambar 3.7, terdapat satu buah tabel yang terdiri dari dua kolom, yakni *Name* dan *Occupation*, Dimana *Name* berisi nama seseorang dan *Occupation* berisi pekerjaannya. Diminta untuk melakukan pivot pada *Occupation*, sehingga setiap nama ditampilkan dibawah Occupation yang sesuai. Urutan kolom yang diharapkan adalah Doctor, Professor, Singer, dan Actor, jika tidak ada nama lagi dalam sebuah pekerjaan, ganti dengan nilai NULL.

Pivot the Occupation column in **OCCUPATIONS** so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

Note: Print **NULL** when there are no more names corresponding to an occupation.

Input Format

The **OCCUPATIONS** table is described as follows:

Column	Type
Name	String
Occupation	String

Occupation will only contain one of the following values: **Doctor**, **Professor**, **Singer** or **Actor**.

Sample Input

Name	Occupation
Samantha	Doctor
Julia	Actor
Maria	Actor
Meera	Singer
Ashely	Professor
Ketty	Professor
Christeen	Professor
Jane	Actor
Jenny	Doctor
Priya	Singer

Sample Output

```
Jenny   Ashley   Meera  Jane
Samantha Christeen  Priya  Julia
NULL    Ketty    NULL  Maria
```

Explanation

The first column is an alphabetically ordered list of Doctor names.
The second column is an alphabetically ordered list of Professor names.
The third column is an alphabetically ordered list of Singer names.
The fourth column is an alphabetically ordered list of Actor names.
The empty cell data for columns with less than the maximum number of names per occupation (in this case, the Professor and Actor

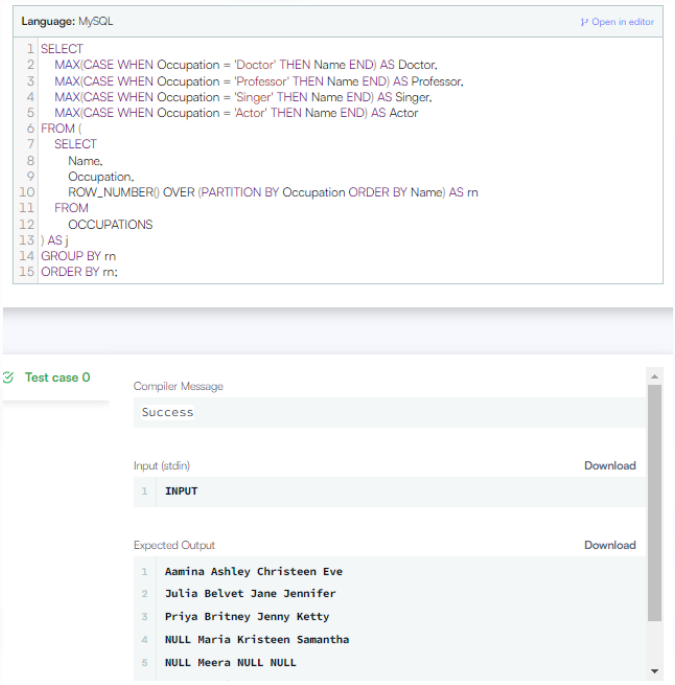
Gambar 3. 7 Contoh 2 Soal SQL

Untuk melakukan hal tersebut, dibuat *query* berikut:

```
SELECT
   MAX(CASE WHEN Occupation = 'Doctor' THEN Name END) AS Doctor,
   MAX(CASE WHEN Occupation = 'Professor' THEN Name END) AS Professor,
   MAX(CASE WHEN Occupation = 'Singer' THEN Name END) AS Singer,
   MAX(CASE WHEN Occupation = 'Actor' THEN Name END) AS Actor
FROM (
   SELECT
      Name,
      Occupation,
      ROW_NUMBER() OVER (PARTITION BY Occupation ORDER BY Name)
AS rn
   FROM
      OCCUPATIONS
) AS j
```

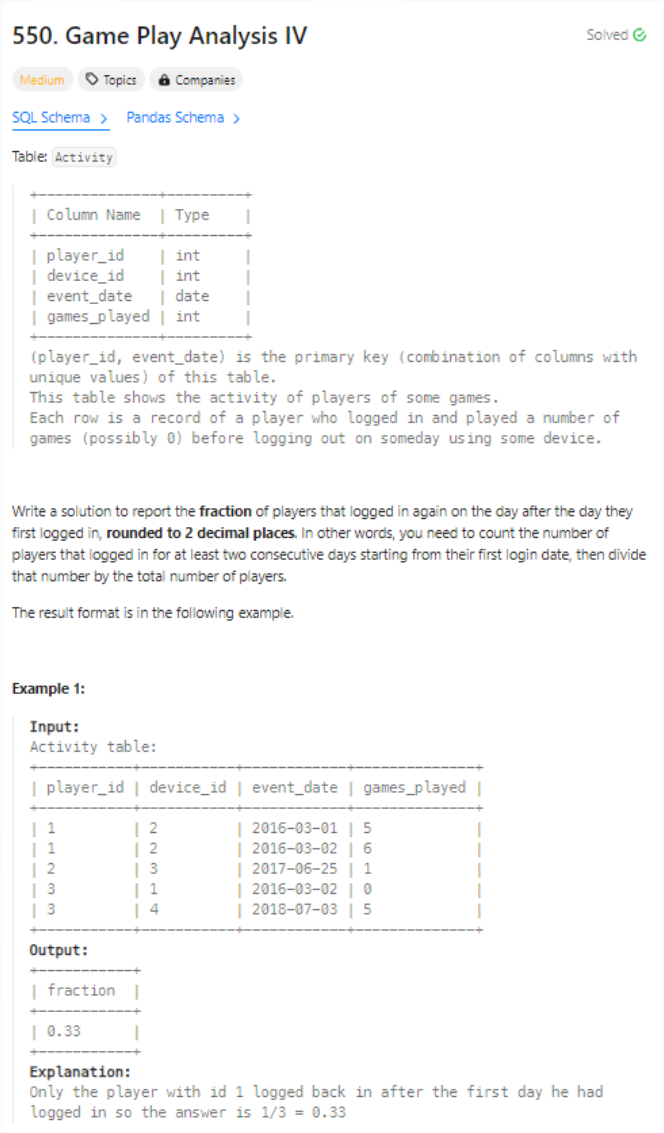
```
GROUP BY rn
ORDER BY rn;
```

Query tersebut pertama akan melakukan *subquery select* kolom *Name, Occupation*, serta membuat kolom baru yang akan memberikan nomor bagi setiap baris setelah dibagi menjadi grup berdasarkan pekerjaan menggunakan kalimat PARTITION BY dan diurutkan menggunakan kalimat ORDER BY berdasarkan nama (*ascending*) yang diberi nama kolom *rn*. Dari *subquery* tersebut, nama ditampilkan menggunakan fungsi MAX() dan kalimat CASE, yang akan menghasilkan kolom sesuai dengan urutan yang diharapkan. Jika tidak ada nama dalam pekerjaan tertentu dalam sebuah grup, maka nilai akan menjadi NULL. Fungsi MAX() diperlukan karena dilakukan GROUP BY berdasarkan *rn*. Jika tidak, maka hanya akan menampilkan satu baris (rn nomor 1). Dalam Gambar 3.9, terdapat beberapa nilai NULL, oleh karena jumlah orang terbanyak dalam sebuah pekerjaan adalah 5 orang (Professor), dan pekerjaan lainnya kurang dari 5 orang. Solusi tersebut tercantum dalam Gambar 3.8.



Gambar 3. 8 Solusi Soal 2 SQL

Dalam soal di gambar 3.9, Terdapat table *Activity* dengan kolom *player_id, device_id, event_date, dan games_played*. *Player_id* merupakan nomor identifikasi pemain, *device_id* merupakan nomor identifikasi perangkat yang digunakan pemain, *event_date* merupakan tanggal mereka *login* ke dalam akun mereka, dan *games_played* merupakan jumlah permainan yang dimainkan pada hari tersebut. Hasil yang diharapkan adalah jumlah pemain yang *login* minimal dua hari berturut-turut dibagi dengan jumlah pemain keseluruhan.



550. Game Play Analysis IV

Solved

Medium | Topics | Companies

SQL Schema > | Pandas Schema >

Table: `Activity`

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

(player_id, event_date) is the primary key (combination of columns with unique values) of this table.
This table shows the activity of players of some games.
Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

Write a solution to report the **fraction** of players that logged in again on the day after the day they first logged in, **rounded to 2 decimal places**. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.

The result format is in the following example.

Example 1:

Input:
Activity table:

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

Output:

fraction
0.33

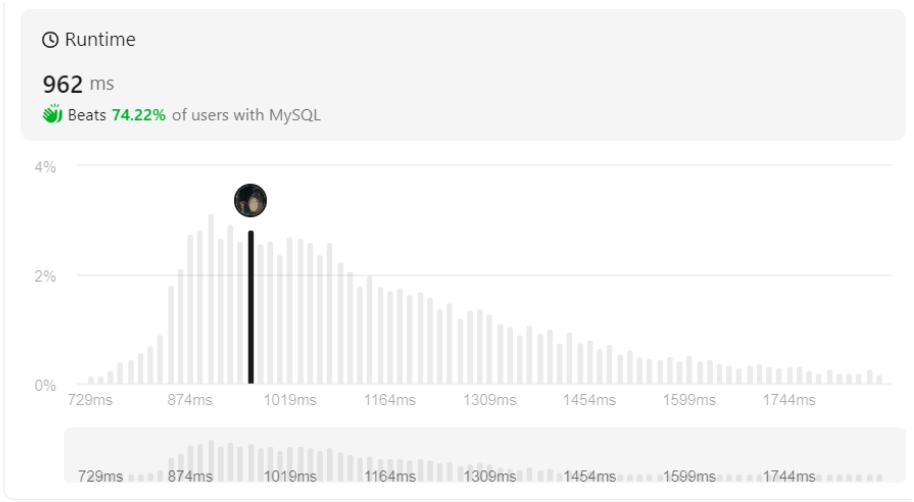
Explanation:
Only the player with id 1 logged back in after the first day he had logged in so the answer is 1/3 = 0.33

Gambar 3. 9 Contoh 3 Soal SQL

Untuk melakukan hal tersebut, digunakan *query* berikut:

```
SELECT
 ROUND(COUNT(DISTINCT player_id) / (SELECT COUNT(DISTINCT player_id) FROM Activity), 2) AS fraction
FROM
 Activity
WHERE
 (player_id, DATE_SUB(event_date, INTERVAL 1 DAY))
 IN (
  SELECT player_id, MIN(event_date) AS first_login FROM Activity GROUP BY player_id)
```

Query tersebut menghitung jumlah pemain, yang ditentukan menggunakan *subquery* yang men-*select* pemain dengan kombinasi *player_id* dan DATE_SUB(event_date, INTERVAL 1 DAY) terdapat di dalam *query* SELECT *player_id*, dan MIN(*event_date)* sesuai dengan grup berdasarkan *player_id*. MIN(*event_date*) berfungsi untuk menentukan tanggal paling awal *login* bagi seluruh pemain. DATE_SUB(*event_date*, INTERVAL 1 DAY) akan menghasilkan tanggal sehari sebelum tanggal yang diinput. Maka dari itu, jika ada hasil DATE_SUB(*event_date*, INTERVAL 1 DAY) yang sama dengan MIN(*event_date*) dari setiap pemain, dapat diartikan bahwa pemain tersebut pernah *login* dua kali berturut-turut. Jumlah pemain yang memenuhi kondisi tersebut kemudian dibagi dengan jumlah pemain keseluruhan, lalu didesimalkan dengan 2 angka dibelakang koma. Solusi tersebut tercantum dalam Gambar 3.10.



Gambar 3. 10 Solusi Soal 3 SQL

Dalam soal di gambar 3.11, terdapat tabel *Customer* dengan kolom *customer*_id yang adalah nomor identifikasi pelanggan, *name* yang adalah nama pelanggan, *visited_on* yang merupakan tanggal pelanggan mengunjungi restoran, dan *amount* yakni total biaya yang dibayar pelanggan. Hasil yang diharapkan dari tabel tersebut adalah moving average dalam jangka waktu 7 hari.

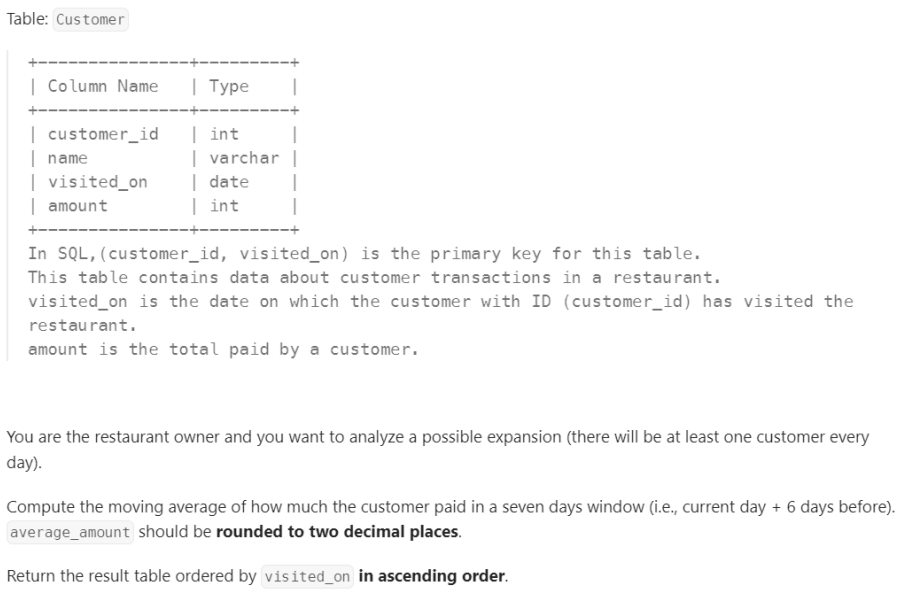


Table: `Customer`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| customer_id   | int     |
| name          | varchar |
| visited_on    | date    |
| amount        | int     |
+---------------+---------+
In SQL,(customer_id, visited_on) is the primary key for this table.
This table contains data about customer transactions in a restaurant.
visited_on is the date on which the customer with ID (customer_id) has visited the
restaurant.
amount is the total paid by a customer.
```

You are the restaurant owner and you want to analyze a possible expansion (there will be at least one customer every day).

Compute the moving average of how much the customer paid in a seven days window (i.e., current day + 6 days before). `average_amount` should be **rounded to two decimal places**.

Return the result table ordered by `visited_on` **in ascending order**.

Gambar 3. 11 Contoh 4 Soal SQL

Untuk melakukan hal tersebut, digunakan *query* berikut:

```
SELECT visited_on,
    (SELECT SUM(amount)
    FROM Customer
    WHERE visited_on BETWEEN DATE_SUB(c.visited_on, INTERVAL 6 DAY)
AND c.visited_on) as amount,
    ROUND((SELECT SUM(amount)/7
    FROM Customer
    WHERE visited_on BETWEEN DATE_SUB(c.visited_on, INTERVAL 6 DAY)
AND c.visited_on),2) as average_amount
FROM Customer c
WHERE visited_on >= (SELECT DATE_ADD(MIN(visited_on), INTERVAL 6 DAY)
```

```
FROM Customer)
GROUP BY visited_on
```

Pertama dibuat *subquery* dalam kalimat FROM dimana diambil baris dimana kolom *visited_on* lebih atau sama dengan 6 hari setelah tanggal paling awal di dalam tabel *Customer* yang digrup berdasarkan *visited_on*, yang kemudian dinamakan sebagai tabel *c*, hal ini berfungsi agar mampu menghitung rata-rata bergerak dalam jangka waktu 7 hari. Dalam kalimat SELECT pertama, dibuat kolom *amount* yang dihasilkan dari *subquery* yang merupakan jumlah kolom *amount* dari baris yang memenuhi kondisi dimana pelanggan berkunjung diantara tanggal *visited_on* dan 6 hari sebelumnya. Lalu dibuat kolom *average_amount* yang adalah rata-rata *amount* dalam satu minggu, dengan menggunakan *subquery* yang sama dengan kolom *amount* sebelumnya. Solusi tersebut tercantum dalam Gambar 3.12.

Input

Customer =

customer_id	name	visited_on	amount
1	Jhon	2019-01-01	100
2	Daniel	2019-01-02	110
3	Jade	2019-01-03	120
4	Khaled	2019-01-04	130
5	Winston	2019-01-05	110
6	Elvis	2019-01-06	140

View more

Output

visited_on	amount	average_amount
2019-01-07	860	122.86
2019-01-08	840	120
2019-01-09	840	120
2019-01-10	1000	142.86

Expected

visited_on	amount	average_amount
2019-01-07	860	122.86
2019-01-08	840	120
2019-01-09	840	120
2019-01-10	1000	142.86

Gambar 3. 12 Solusi Soal 4 SQL

Dalam soal di gambar 3.13, terdapat tabel *Insurance* dengan kolom *pid*, yang merupakan nomor identifikasi pemegang polis, *tiv_2015* yang merupakan jumlah investasi yang dimiliki pemegang polis pada tahun 2015, *tiv_2016* yang merupakan jumlah investasi yang dimiliki pemegang polis pada tahun 2016, *lat* yang merupakan *latitude* kota asal pemegang polis, dan *lon* yang merupakan *longitude* kota asal pemegang polis. Hasil yang diharapkan adalah jumlah investasi dari pemegang polis yang memiliki nilai investasi yang sama pada tahun 2015 namun tinggal di kota yang berbeda.

Table: `Insurance`

```
+-------------+-------+
| Column Name | Type  |
+-------------+-------+
| pid         | int   |
| tiv_2015    | float |
| tiv_2016    | float |
| lat         | float |
| lon         | float |
+-------------+-------+
```

pid is the primary key (column with unique values) for this table.
Each row of this table contains information about one policy where:
pid is the policyholder's policy ID.
tiv_2015 is the total investment value in 2015 and tiv_2016 is the total investment value in 2016.
lat is the latitude of the policy holder's city. It's guaranteed that lat is not NULL.
lon is the longitude of the policy holder's city. It's guaranteed that lon is not NULL.

Write a solution to report the sum of all total investment values in 2016 `tiv_2016`, for all policyholders who:

- have the same `tiv_2015` value as one or more other policyholders, and
- are not located in the same city as any other policyholder (i.e., the `(lat, lon)` attribute pairs must be unique).

Round `tiv_2016` to **two decimal places**.

Example 1:

Input:
Insurance table:

```
+-----+----------+----------+-----+-----+
| pid | tiv_2015 | tiv_2016 | lat | lon |
+-----+----------+----------+-----+-----+
| 1   | 10       | 5        | 10  | 10  |
| 2   | 20       | 20       | 20  | 20  |
| 3   | 10       | 30       | 20  | 20  |
| 4   | 10       | 40       | 40  | 40  |
+-----+----------+----------+-----+-----+
```

Output:

```
+----------+
| tiv_2016 |
+----------+
| 45.00    |
+----------+
```

Explanation:
The first record in the table, like the last record, meets both of the two criteria.
The tiv_2015 value 10 is the same as the third and fourth records, and its location is unique.

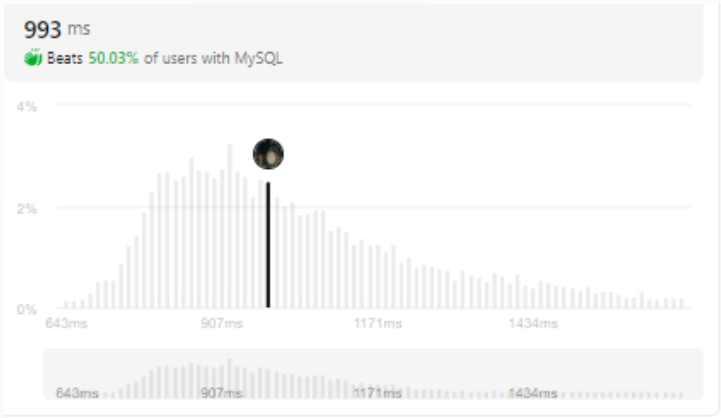
The second record does not meet any of the two criteria. Its tiv_2015 is not like any other policyholders and its location is the same as the third record, which makes the third record fail, too.
So, the result is the sum of tiv_2016 of the first and last record, which is 45.

Gambar 3. 13 Contoh 5 Soal SQL

Untuk melakukan itu, digunakan *query* berikut:

```
SELECT ROUND(SUM(tiv_2016),2) as tiv_2016
FROM Insurance
WHERE tiv_2015
IN (SELECT
tiv_2015 FROM Insurance
GROUP BY tiv_2015
HAVING COUNT(tiv_2015) > 1)
AND (lat,lon)
IN (SELECT lat,lon
FROM Insurance
GROUP BY lat, lon
HAVING COUNT(*) = 1)
```

Kalimat SELECT utama menghasilkan jumlah dari *tiv_2016* yang dibulatkan dengan dua angka dibelakang koma. Untuk mem-*filter* baris yang memenuhi kondisi, dibuat dua buah *subquery* yang dihubungkan dengan kata AND, Dimana *subquery* pertama mencari grup nilai *tiv_2015* yang memiliki lebih dari satu baris, yang artinya ada setidaknya dua orang dengan *tiv_2015* yang sama. *Subquery* kedua mencari grup kombinasi *lat* dan *lon* yang memiliki tepat satu baris, yang berarti hanya ada satu orang yang tinggal di kota tersebut. Oleh karena dihubungkan dengan kata AND, baris yang dihasilkan adalah yang memenuhi dua kondisi tersebut. Solusi tersebut tercantum dalam Gambar 3.14.



ode | MySQL

```mysql
# Write your MySQL query statement below
SELECT ROUND(SUM(tiv_2016),2) as tiv_2016
FROM Insurance
WHERE tiv_2015
IN (SELECT
tiv_2015 FROM Insurance
GROUP BY tiv_2015
HAVING COUNT(tiv_2015) > 1)

AND (lat,lon)
IN (SELECT lat,lon
FROM Insurance
GROUP BY lat, lon
HAVING COUNT(*) = 1)
```

Input

Insurance =

pid	tiv_2015	tiv_2016	lat	lon
1	10	5	10	10
2	20	20	20	20
3	10	30	20	20
4	10	40	40	40

Output

tiv_2016
45

Expected

tiv_2016
45

Gambar 3. 14 Solusi Soal 5 SQL

3.3.3 Minggu 7: Mempelajari Dataset

Dataset routing yang dianalisis terdiri dari 4 jenis dataset yang saling berkaitan, yakni *routing journal*, *routing master data*, *bill of material (BOM)* dan *consumption*. Jurnal berisi data tentang detail proses yang dilakukan terkait item tertentu, seperti ID Produk, Nomor Routing, serta waktu *setup* dan *runtime*. *Master data* berfungsi sebagai 'standard' terkait waktu yang diperlukan untuk membuat sebuah barang. Proyek yang dilakukan memiliki 5 tujuan, yakni membuat rekomendasi *run time* dan *setup time* data master, menentukan jumlah *wait time* jurnal, membuat pohon *bom* dan menentukan total waktu produksi sebuah item, membuat rekomendasi *work center*, serta membuat rekomendasi *raw material*. Data jurnal terdiri dari 42 kolom, namun dalam analisis, terdapat 7 kolom utama yang digunakan, yakni:

Tabel 3. 2 Deskripsi kolom data jurnal

Kolom	Deskripsi
Item No_	Nomor identifikasi untuk produk tertentu
Routing No_	Nomor routing untuk produk tertentu.
Setup Time	Waktu untuk persiapan suatu proses
Run Time	Waktu suatu proses berjalan
Work Center No.	Tempat dimana suatu proses dikerjakan
Description	Nama proses yang dilakukan
OrderNo	Nomor dokumen *work order*

Satu nomor routing terdiri dari beberapa proses, seperti *inspection, assembling, dan receiving*. Setup Time dan Run Time menggunakan satuan menit. *Master Data* memiliki 35 kolom, namun hanya 7 kolom yang digunakan, yakni Routing No., Operation No., Next Operation No., Work Center No., Description, Setup Time, dan Run Time.

Data master terdiri dari 35 kolom, namun dalam analisis digunakan 4 kolom utama, yakni:

Tabel 3. 3 Deskripsi Kolom Master Data

Kolom	Deskripsi
Routing No_	Nomor routing untuk produk tertentu.

Kolom	Deskripsi
Setup Time	Waktu untuk persiapan suatu proses
Run Time	Waktu suatu proses berjalan
Operation No	Urutan suatu proses dalam routing
Work Center No	Tempat proses dijalankan

Data BOM terdiri dari 23 kolom, namun dalam analisis digunakan 3 kolom utama, yakni:

Tabel 3. 4 Deskripsi Kolom data BOM

Kolom	Deskripsi
Production BOM No.	Nomor identifikasi item
No.	Nomor identifikasi material
Description	Nama material untuk membuat item

Item dalam BOM dapat dibagi menjadi 2 jenis berdasarkan angka depan Production BOM No., dimana jika angka depannya 4, maka Item tersebut adalah *finished good*, sedangkan angka depan 2 merupakan *component*. Sebuah *finished good* terdiri dari gabungan *component* dan/atau *accessories* (angka depan 3, tidak diproduksi).

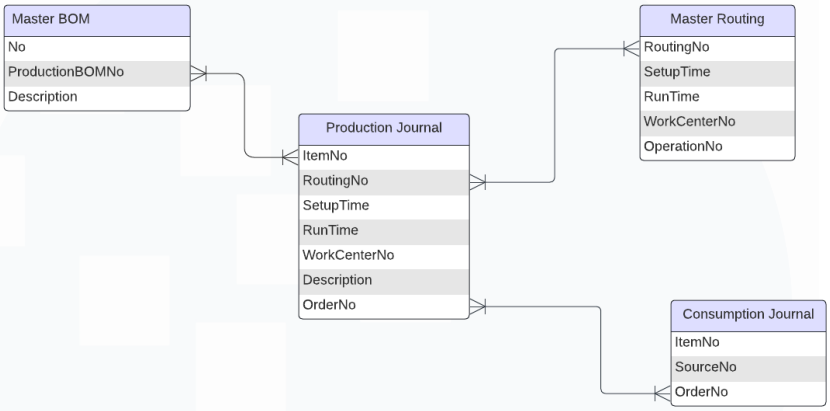
Data *consumption* berisi tentang material yang digunakan dalam proses pembuatan sebuah *item* / komponen. *Consumption* memiliki 13 kolom, namun hanya digunakan 3 kolom dalam analisis, yakni:

Tabel 3. 5 Deskripsi kolom data consumption

Kolom	Deskripsi
ItemNo	Nomor identifikasi material
SourceNo	Nomor identifikasi *item*
OrderNo	Nomor identifikasi surat perintah

ItemNo merupakan ID *raw material* yang digunakan, SourceNo merupakan ID *item*/ komponen yang dibuat, dan OrderNo merupakan ID *work order*. BOM terhubung dengan jurnal melalui ProductionBOMNo dan ItemNo, *master data* terhubung dengan jurnal melalui RoutingNo, dan *consumption*

terhubung dengan jurnal melalui OrderNo. Hubungan antara masing-masing file digambarkan dalam *Entity Relationship Diagram* pada Gambar 3.15.



Gambar 3. 15 Entity Relationship Diagram

3.3.4 Minggu 8: Importing dan Merging Dataset

Dalam Gambar 3.16, digunakan beberapa *library* yang digunakan dalam proyek. Pandas digunakan untuk memanipulasi data, glob digunakan untuk mengakses file excel yang terdapat di komputer, pandasql digunakan untuk membuat query dalam pembuatan variabel, dan anytree digunakan untuk membuat visualiasi *bom* tree.

```
import pandas as pd

import glob

from pandasql import sqldf

from anytree import Node, RenderTree
```

Gambar 3. 16 Import Library

Terdapat 8 file journal, yang disatukan dengan cara mengisi *list* dengan nilai yang terdapat pada semua file, lalu diubah menjadi format Excel kembali dengan code dalam Gambar 3.17:

Import Journal

```
path = "E:/Arga/Capacity/2017-2024"
```

```
file_list = glob.glob(path + "/*.xlsx")
file_list
```

```
excl_list = []

for file in file_list:
    excl_list.append(pd.read_excel(file))
```

```
excl_merged = pd.concat(excl_list, ignore_index=True)
```

```
excl_merged.to_excel('Capacity_Merged.xlsx', index=False)
```

```
journal = pd.read_excel('Capacity_Merged.xlsx')
```

```
journal.info()
```

```
journal.head()
```

Gambar 3. 17 Menyatukan File Journal

Hanya ada 1 file *master data*, BOM, dan *consumption* sehingga tidak perlu melakukan merging. Tanda titik, garis bawah dan spasi pada nama kolom dihapus agar memudahkan untuk dibaca, serta kolom PostingDate pada variabel **journal** diubah format dari yyyy-MM-dd HH:mm:ss A menjadi yyyy-MM-dd dengan code dalam Gambar 3.18.

Remove Spaces and Underscores

```
journal.columns = journal.columns.str.replace(' ', '')
journal.columns = journal.columns.str.replace('_', '')
# So column can be queried using sql
```

Extract Date

```
journal['PostingDate'] = journal['PostingDate'].dt.date
```

Save to quickly import

```
journal.to_pickle('journal.pkl')
```

```
journal = pd.read_pickle('journal.pkl')
```

Gambar 3. 18 Transformasi Data Jurnal

Dibuat variabel **master** untuk dokumen *master data*, yang kemudian tanda spasi dan titik dihilangkan, serta kolom OperationNo diubah menjadi *data type string*, lalu variabel disimpan dalam format .pkl agar saat di import kembali proses menjadi lebih cepat dengan code dalam Gambar 3.19.

Gambar 3. 19 Import Master Data

Dibuat variabel **bom** untuk dokumen *master data*, yang kemudian tanda spasi, titik dan garis bawah dihilangkan, lalu variabel disimpan dalam format .pkl agar saat di import kembali proses menjadi lebih cepat dengan code dalam Gambar 3.20:

Gambar 3. 20 Transformasi Master Data

Dibuat dua kolom baru pada variabel **journal**, yakni **TotalTime** yang merupakan jumlah RunTime dan SetupTime, kolom **AverageProcessTime**, yang merupakan **TotalTime** dibagi dengan kolom **OutputQuantity**, serta kolom **RunTimePerItem** dan SetupTimePerItem yang merupakan masing-masing RunTime dan SetupTime yang dibagi dengan **OutputQuantity,** hal ini dilakukan untuk mengetahui waktu yang diperlukan per *item* pada setiap

proses. Dalam kolom **AverageProcessTime** terdapat nilai *inf*, sehingga semua row yang memiliki nilai tersebut akan dihapus, karena tidak bisa digunakan sebagai perbandingan. Selain itu **journal** difilter menjadi data dari tahun 2019 keatas dengan code dalam Gambar 3.21.

```
journal['TotalTime'] = journal['SetupTime'] + journal['RunTime']

journal['AverageProductionTime'] = journal['TotalTime']/journal['OutputQuanti

journal['RunTimePerItem'] = journal['RunTime']/journal['OutputQuantity']

journal['SetupTimePerItem'] = journal['SetupTime']/journal['OutputQuantity']

journal = journal.replace(np.inf,np.nan)

journal = journal.dropna(subset = ['AverageProductionTime'])

journal = journal[journal['OutputQuantity'] > 0]

journal['PostingDate'] = pd.to_datetime(journal.PostingDate, format='%Y-%m-%d

journal = journal[journal['PostingDate'].dt.year >= 2019]
```

Gambar 3. 21 Menambah kolom dan filter journal

Karena tidak semua routing di jurnal memiliki informasi di master data, dibuat variabel **journal_in_master** dengan code berikut:

```
journal_in_master = sqldf('Select * from journal WHERE (RoutingNo, OperationNo, Description, WorkCenterNo) IN (SELECT RoutingNo,OperationNo,Description,WorkCenterNo FROM master)')
```

Query tersebut menggunakan kombinasi RoutingNo, OperationNo, Description, WorkCenterNo untuk memastikan bahwa setiap routing di journal memiliki hubungan dengan informasi di master data. Lalu variabel disimpan dalam format pkl.

Consumption diimpor dengan membaca *file excel*. Tanda titik, garis bawah dan spasi pada nama kolom dihapus agar memudahkan untuk dibaca, dan difilter menjadi tahun 2019 keatas dengan code dalam Gambar 3.22.

Gambar 3. 22 Impor data consumption

3.3.5 Minggu 9: Rekomendasi Setup Time & Runtime Master Data

Jurnal digrup berdasarkan kolom ItemNo, RoutingNo, OperationNo, WorkCenterNo, dan Description. Hal ini dilakukan untuk mencari rata-rata waktu setiap proses. Dicari rata-rata dari RunTimePerItem dan SetupTimePerItem dari setiap grup dengan code dalam Gambar 3.23. Nilai ini akan menjadi rekomendasi bagi *master data*. Nilai tersebut di gabungkan dengan *dataframe* **master** yang diberikan nama baru yakni **recommendation.** Misalnya dalam Gambar 3.24, rekomendasi untuk *routing* nomor 21A000700A02 dengan nomor operasi 15 di *work center* 2PROD adalah 0.00 menit untuk SetupTime dan 3.28 menit untuk RunTime.

Gambar 3. 23 Mencari Rata-Rata SetupTime dan RunTime Jurnal

	RoutingNo	OperationNo	WorkCenterNo	Description	SetupTime	RunTime	SetupTimeRecommendation	RunTimeRecommendation
0	21A000700A02	15	2PROD	P005 Pilot Hydrolic Prees	5.0	3.090	0.00	3.28
1	21A000700A02	13	2PROD	P023 Cina Mechanical Power Prees	20.0	2.720	0.00	2.97
2	21A000700A02	14	2PROD	P309 Pedastal Drilling Model M1-25 2HP	5.0	2.354	0.01	3.53
3	21A000700A02	16	2PROD	P006 Rolling Machine (Allen Wes Co Ltd 5.2 Hp)	5.0	3.120	0.00	6.23
4	21A000700A02	17	2PROD	P379 Cigweld W 1003501 Machine Tackweld	5.0	2.040	0.00	2.47

Gambar 3. 24 Rekomendasi SetupTime dan RunTime

3.3.6 Minggu 10: Penentuan Wait Time Jurnal

Dibuat variabel **journal_added_wait_time**, yang merupakan **journal** dengan beberapa kolom yang dihapus. Dibuat variabel **extracted** yang merupakan beberapa kolom dari **master.** Kolom **MasterTotalTime** yang merupakan jumlah dari MasterSetupTime dan MasterRunTime.

Dataframe **journal_added_wait_time** di filter, sehingga hanya tersisa row dengan nilai TotalTime yang lebih besar dari MasterTotalTime. Setelah itu dibuat kolom **JournalWaitTime**, yang merupakan selisih dari TotalTime dan MasterTotalTime dengan code dalam Gambar 3.25. Misalnya dalam Gambar 3.26, *wait time* untuk salah satu proses item dan routing nomor 22Z021600A00 adalah 40 menit. Namun, terdapat *wait time* yang sangat tinggi, misalnya 3290 menit, hal tersebut disebabkan OutputQuantity yang lebih kecil daripada BOM namun dengan *run time* dan *setup time* yang tidak jauh berbeda dari proses sejenisnya sehingga AverageProcessTime menghasilkan nilai yang jauh lebih besar. Kasus seperti ini perlu diselidiki lebih lanjut.

```
journal_added_wait_time = pd.read_pickle('journal_in_master.pkl')

journal_added_wait_time = journal_added_wait_time.reset_index(drop = True)

journal_added_wait_time = journal_added_wait_time[['ItemNo','RoutingNo','OperationNo','WorkCenterNo','Description','AveragePr

extracted = master[['RoutingNo','OperationNo','WorkCenterNo','Description','SetupTime','RunTime']].rename(columns = {'SetupTi

extracted['MasterTotalTime'] = extracted['MasterSetupTime'] + extracted['MasterRunTime']

journal_added_wait_time = pd.merge(journal_added_wait_time, extracted, left_on = ['RoutingNo','OperationNo','WorkCenterNo','D

ime = journal_added_wait_time[(journal_added_wait_time['AverageProcessTime']) > (journal_added_wait_time['MasterTotalTime'])]

t_time.loc[:, 'JournalWaitTime'] = journal_added_wait_time['AverageProcessTime'] - journal_added_wait_time['MasterTotalTime']
```

Gambar 3. 25 Pembuatan variabel journal_added_wait_time

ItemNo	RoutingNo	OperationNo	WorkCenterNo	Description	AverageProcessTime	MasterSetupTime	MasterRunTime	MasterTotalTime	JournalWaitTime
1600A00	22Z021600A00	15	1PROD	First Vertical Manual Milling Machine Lc-20Vhn	250.000000	15.0	195.0	210.0	40.000000
1600A00	22Z021600A00	15	1PROD	First Vertical Manual Milling Machine Lc-20Vhn	330.000000	15.0	195.0	210.0	120.000000
1600A00	22Z021600A00	15	1PROD	First Vertical Manual Milling Machine Lc-20Vhn	480.000000	15.0	195.0	210.0	270.000000
1600A00	22Z021600A00	15	1PROD	First Vertical Manual Milling Machine Lc-20Vhn	614.035088	15.0	195.0	210.0	404.035088
1600A00	22Z021600A00	15	1PROD	First Vertical Manual Milling Machine Lc-20Vhn	3500.000000	15.0	195.0	210.0	3290.000000

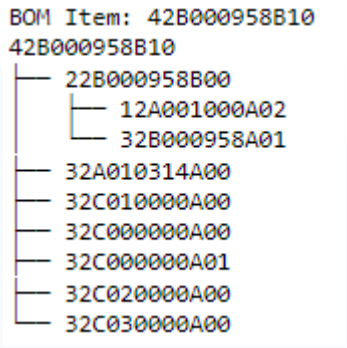
Gambar 3. 26 Hasil Wait Time Jurnal

3.3.7 Minggu 11 - 13: Penentuan Waktu Produksi Item dan BOM Tree

Dibuat fungsi **add_nodes** dan **item_bom** yang berfungsi untuk memvisualisasikan routing dengan code dalam Gambar 3.27. **Item_bom** mengambil data BOM sebuah *item* dengan pertama mencari **bom** sesuai dengan input user yang disimpan sebagai **data1**. Oleh karena sebuah *item* dapat terdiri dari beberapa *component/accessories*, dicari lagi BOM untuk *component/accessories* tersebut, yang disimpan sebagai **data2**. **Data3** merupakan gabungan **data1** dan **data2** secara vertikal. Visualisasi dibuat menggunakan *library* **anytree**, yang dapat membuat *object Node* yang disimpan dalam *dictionary*. Kolom **ProductionBOMNo** merupakan kode sebuah item, dan kolom **No** merupakan material dari item tersebut.

```
def add_nodes(nodes, parent, child):
    if parent not in nodes:
        nodes[parent] = Node(parent)
    if child not in nodes:
        nodes[child] = Node(child)
    nodes[child].parent = nodes[parent]

def item_bom(item):
    data1 = bom[bom['ProductionBOMNo'] == item]
    data2 = bom[bom['ProductionBOMNo'].isin(data1['No'])]
    data3 = pd.concat([data1,data2])
    data3 =  data3[['ProductionBOMNo','No']]

    nodes = {}
    for parent, child in zip(data3["ProductionBOMNo"],data3["No"]):
        add_nodes(nodes, parent, child)

    roots = list(data3[~data3["ProductionBOMNo"].isin(data3["No"])]["ProductionBOMNo"].unique())
    for root in roots:
        for pre, fill, node in RenderTree(nodes[root]):
            print("%s%s" % (pre, node.name))

search = input('BOM Item: ')
item_bom(search)
```

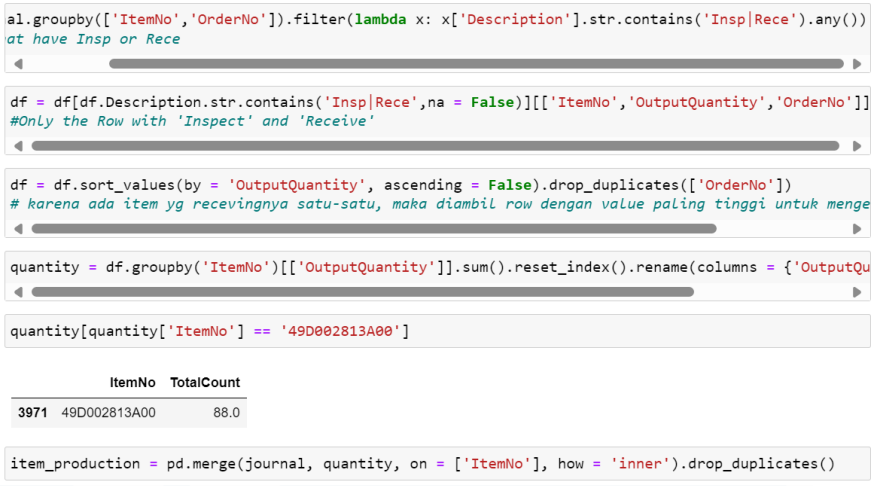
Gambar 3. 27 Fungsi visualisasi BOM Tree



Gambar 3. 28 Visualisasi BOM Tree

Dapat dilihat dalam Gambar 3.28, *item* 42B000958B10 menggunakan komponen 22B000958B00, aksesoris 32A010314A00,32C010000A00, 32C000000A00, 32C000000A01, 32C020000A00, 32C30000A00. Untuk membuat komponen 22B000958B00 membutuhkan *raw material* 12A001000A02 dan 32B000958A01.

Untuk menentukan total waktu yang diperlukan untuk membuat sebuah item, pertama perlu dicari jumlah barang yang diproduksi. Sebuah proses routing diakhiri dengan proses *Inspection* atau *Receiving*. Maka dari itu dibuat variabel **df** yang merupakan **journal** yang digrup berdasarkan ItemNo dan OrderNo, yang kemudian difilter sehingga berisi *work order* dengan proses *Inspection* atau *Receiving*. Kemudian, hanya diambil row yang berisi proses *Inspection* atau *Receiving* agar jumlah barang per *work order* bisa dihitung. Hanya satu row dari setiap *work order* sebuah item yang diambil, yakni yang memiliki OutputQuantity yang paling tinggi, yang menunjukkan total barang yang di produksi dalam sebuah *work order.* Jumlah dari semua *work order* disimpan dalam variabel **quantity**. Kolom OutputQuantity kemudian diubah nama menjadi TotalCount. Misalnya dalam Gambar 3.29, ada 88 buah *item* 49D002813A00 yang diproduksi. Jumlah setiap *item* yang diproduksi kemudian digabungkan dengan **journal** sesuai dengan grup masing-masing sebagai variabel **item_production**.



Gambar 3. 29 Menghitung jumlah produksi item

Dibuat fungsi time_per_product yang berfungsi untuk membuat kolom TimePerProduct, yang merupakan jumlah TotalTime dari semua proses dibagi dengan jumlah *item yang* diproduksi (TotalCount). Fungsi item_wo_time berfungsi untuk menggabungkan hasil time_per_product sesuai dengan input user. Fungsi ini menggabungkan ID *item* yang dicari beserta komponen yang digunakan sesuai dengan BOM. Misalnya dalam Gambar 3.30, *item* 42B000312N11 memerlukan waktu 834 menit untuk setiap produknya. Nilai tersebut merupakan jumlah dari rata-rata waktu semua proses produksi *component* yang berkaitan dengan *item* 42B000312N11 dan *item* itu sendiri. Hal yang perlu diperhatikan dari hasil ini adalah bahwa perhitungan waktu dilakukan dengan asumsi bahwa proses dilakukan tidak secara bersamaan. Jika semua proses waktu produksi dilakukan bersamaan, kemungkinan akan memerlukan waktu yang lebih singkat.

```
def time_per_product(df):
    total_time = df['TotalTime'].sum()
    count = df['TotalCount'].mean()
    time_per_item = (total_time/count).round(2)
    df['TimePerProduct'] = time_per_item
    return df
```

```
production_time = item_production.groupby('ItemNo').apply(time_per_product)
```

```
#production_time.head()
```

```
def item_wo_time(item):
    df = bom[bom['ProductionBOMNo'] == item]
    return production_time[(production_time['ItemNo'] == item) | (production_time['ItemNo'].isin(df
```

```
search_item = input('Search for Item: ')
result = item_wo_time(search_item)

result = result[['ItemNo','TimePerProduct']].drop_duplicates(subset = 'ItemNo')
total_time_mean = result['TimePerProduct'].sum().round(2)

print('Total time to make this item: ' + str(total_time_mean) + ' minutes')

result.sort_values(by = 'ItemNo', ascending = True).reset_index(drop = True)
```

```
Search for Item: 42B000312N11
Total time to make this item: 834.89 minutes
```

	ItemNo	TimePerProduct
0	22B000312N01	443.00
1	22Z000312A01	71.00
2	22Z020312A14	147.20
3	22Z030312A01	77.50
4	22Z040000A01	18.86
5	42B000312N11	77.33

Gambar 3. 30 Total waktu produksi item

3.3.8 Minggu 14: Penentuan Alternatif Work Center

Dibuat variable **journal_2** yang berisikan detail jurnal, namun berbeda dengan variable **journal_in_master** yang menggunakan kombinasi RoutingNo, Description dan Work Center, **journal_2** hanya menggunakan kombinasi RoutingNo dan Description. Hal ini dilakukan agar WorkCenter yang tidak terdapat dalam routing di **master** namun sebenarnya bisa digunakan di **journal** dapat dianalisis. Variabel tersebut dibuat dengan code berikut:

```
journal_2 = sqldf('Select * from journal WHERE (RoutingNo, Description) IN (SELECT RoutingNo,Description FROM master)')
```

Dibuat dua kolom baru pada variabel **journal_2**, yakni **TotalTime** yang merupakan jumlah RunTime dan SetupTime, kolom **AverageProcessTime**, yang merupakan **TotalTime** dibagi dengan kolom **OutputQuantity**, serta kolom **RunTimePerItem** dan SetupTimePerItem yang merupakan masing-masing RunTime dan SetupTime yang dibagi dengan **OutputQuantity,** hal ini

dilakukan untuk mengetahui waktu yang diperlukan per *item* pada setiap proses. Selain itu, row dengan nilai *inf* pada kolom AverageProcessTime dihapus dengan code dalam Gambar 3.31.

```
journal_2['TotalTime'] = journal_2['SetupTime'] + journal_2['RunTime']
journal_2['AverageProcessTime'] = journal_2['TotalTime']/journal_2['OutputQuantity']
journal_2 = journal_2.replace(np.inf,np.nan)
journal_2 = journal_2.dropna(subset = ['AverageProcessTime'])
```

Gambar 3. 31 Membuat kolom baru dan menghapus nilai inf

Rata-rata **AverageProcessTime** proses dalam setiap WorkCenter dihitung dan digabungkan bersama **journal_2** sebagai *dataframe* **journal_routing_multi_workcenter. AverageProcessTime** diubah nama menjadi **GroupAverageProcessTime**, kemudian *dataframe* **journal_routing_multi_workcenter** difilter sehingga hanya ada proses yang memiliki 2 atau lebih WorkCenterNo. Misalnya pada Gambar 3. 32, Operation 14 pada *routing* 21A001600D00 dapat dilakukan pada *work center* LO.B23 atau 2PROD, dengan rata-rata waktu di LO.B23 adalah 3 menit dan di 2PROD adalah 6 menit.

```
journal_2 = journal_2[['ItemNo','RoutingNo','OperationNo','WorkCenterNo','Description','AverageProc
group_average = journal_2.groupby(['ItemNo','RoutingNo','OperationNo','Description','WorkCenterNo']
rNo','Description'], right_on = ['RoutingNo','OperationNo','WorkCenterNo','Description'], how = 'inr
journal_routing_multi_workcenter = journal_routing_multi_workcenter.drop('AverageProcessTime_x', ax
journal_routing_multi_workcenter.rename(columns = {'AverageProcessTime_y':'GroupAverageProcessTime'
','OperationNo','Description']).filter(lambda x: x['WorkCenterNo'].nunique() > 1).drop_duplicates()
journal_routing_multi_workcenter.sort_values(by = ['RoutingNo','OperationNo'])
```

	ItemNo	RoutingNo	OperationNo	WorkCenterNo	Description	GroupAverageProcessTime
7409	21A001600D00	21A001600D00	14	LO.B23	Pilot Hydrolic Prees	3.00
54629	21A001600D00	21A001600D00	14	2PROD	Pilot Hydrolic Prees	6.00

Gambar 3. 32 Menentukan proses dengan work center lebih dari satu

WorkCenter terbaik ditentukan dengan cara memberikan label ‘Best’ pada row dengan nilai **GroupAverageProcessTime** paling kecil pada setiap grupnya. Misalnya dalam Gambar 3.33, proses Pilot Hydraulic Press dari

routing 21A001600D00 dapat dilakukan di WorkCenter 2PROD dan LO.B23, namun yang lebih cepat adalah LO.B23. Yang perlu diperhatikan dari rekomendasi adalah bahwa waktu menjalankan sebuah proses dapat dipengaruhi oleh faktor lainnya, seperti *wait time*, sehingga perlu peninjauan secara langsung yang terkontrol untuk menentukan yang *work center* terbaik di lapangan.

```
# Function to label rows with lowest value as 'best'
def best_workcenter(df):
    min_index = df['GroupAverageProductionTime'].idxmin()
    df.loc[min_index, 'Label'] = 'Best'
    return df
```

```
workcenter_recommendation = journal_routing_multi_workcenter.groupby(['ItemNo','RoutingNo','OperationNo','Description']).appl
```

```
workcenter_recommendation
```

outingNo	OperationNo	Description		ItemNo	RoutingNo	OperationNo	WorkCenterNo	Description	GroupAverageProductionTime	Label
1600D00	14	Pilot Hydrolic Prees	7409	21A001600D00	21A001600D00	14	LO.B23	Pilot Hydrolic Prees	3.00	Best
			54629	21A001600D00	21A001600D00	14	2PROD	Pilot Hydrolic Prees	6.00	NaN
	21	Final Inspection / Living Up	7351	21A001600D00	21A001600D00	21	LO.QC01	Final Inspection / Living Up	0.20	NaN
			54766	21A001600D00	21A001600D00	21	2PROD	Final Inspection / Living Up	0.10	Best

Gambar 3. 33 Rekomendasi Work Center

3.3.9 Minggu 15 - 17: Penentuan Alternatif Raw Material

Dibuat variabel **bom_uses_material** yakni *item* di data BOM yang menggunakan *raw material* sebagai bahan pembuatannya. Hal ini dilakukan dengan cara mencari kolom No yang dimulai dengan angka ‘1’. Selain itu, routing yang mengandung kata ‘SERVICE’ dihapus, hal ini dilakukan sehingga baris yang dihasilkan hanyalah *item* yang dapat diproduksi. Dilakukan filtering yang sama pada dataset *consumption*, ditambah kolom SourceNo yang ada didalam ProductionBOMNo dataset BOM, yang kemudian disimpan dalam variabel **consumption_uses_material** dengan code dalam Gambar 3.34.

Gambar 3. 34 Pembuatan bom_uses_material dan consumption_uses_material

Dilakukan grouping pada variabel **bom_uses_material** berdasarkan ProductionBOMNo, lalu kolom No diubah menjadi set. Dilakukan juga grouping pada **consumption_uses_material** berdasarkan SourceNo dan OrderNo, lalu kolom ItemNo diubah menjadi set. Hal ini dilakukan agar dapat digunakan fungsi **issubset()**, yang mengecek apakah ItemNo (*raw material*) yang digunakan dalam *consumption* sama dengan No (*raw material*) di BOM. Jika ya, maka *consumption* menggunakan *raw material* yang sama dengan BOM, jika tidak, maka menggunakan *raw material* yang berbeda. Dibuat kolom Bernama Label, dengan nilai 'Same As Master' untuk penggunaan *raw material* yang sama, dan 'Alternative' untuk penggunaan *raw material* yang berbeda. Setelah itu kolom No dan ItemNo diubah nama menjadi MasterMaterial dan MaterialUsed, serta datatype kolom diubah menjadi string. Semua ini disimpan dalam variabel **merged_df.** Misalnya dalam Gambar 3.35, MaterialUsed merupakan bagian dari MasterMaterial, sehingga Label memiliki nilai 'Same As Master'.

```
# Group df1 by Item
df1 = bom_uses_material.groupby('ProductionBOMNo')['No'].apply(set).reset_inc

# Group df2 by Item and OrderNo
df2 = consumption_uses_material.groupby(['SourceNo', 'OrderNo'])['ItemNo'].ap
```

```
# Merge the two grouped DataFrames on column B
merged_df = pd.merge(df1, df2, left_on = 'ProductionBOMNo',right_on = 'Source
merged_df

# Check if column A in df1 is subset of column A in df2 after group by B
merged_df['Label'] = merged_df.apply(lambda x: 'Same As Master' if x['ItemNo'
merged_df = pd.DataFrame(merged_df)
```

```
merged_df
```

```
= {'No':'MasterMaterial','ItemNo':'MaterialUsed'}).drop('SourceNo',axis = 1)
```

```
merged_df = merged_df.astype(str)
```

```
merged_df[merged_df['ProductionBOMNo'] == '22B000312N01']
```

	ProductionBOMNo	MasterMaterial	OrderNo	MaterialUsed	Label
1811	22B000312N01	{'14A190412A00', '12A010400A00'}	WO-CF/2309/1700	{'14A190412A00', '12A010400A00'}	Same As Master

Gambar 3. 35 Menentukan label berdasarkan material yang digunakan

Variabel **journal** digabungkan dengan **merged_df** berdasarkan ItemNo-ProductionBOMNo dan OrderNo-OrderNo. Untuk mencari waktu yang dibutuhkan untuk membuat sebuah item dalam sebuah *work order*, pertama **merged_journal** digrup berdasarkan ItemNo dan OrderNo, lalu difilter sehingga hanya ada yang memiliki ‘Inspection’ atau ‘Receiving’ dalam kolom Description. Setelah difilter, hanya diambil row dengan ‘Inspection’ atau ‘Receiving’. Hal ini dilakukan untuk mengetahui berapa jumlah *item* yang diproduksi dalam setiap *work order*, yang disimpan dalam variabel **quantity**. OutputQuantity dari setiap *work order* juga dijumlahkan sesuai grupnya, yang disimpan dalam variabel wo_time. Setelah **quantity** dan **wo_time** digabungkan, TotalTime dibagi dengan OutputQuantity untuk mendapatkan waktu yang diperlukan per item. Misalnya dalam Gambar 3.36, waktu untuk sebuah *work order* yang menggunakan *raw material* yang sama dengan *master data* adalah 443 menit.

```
df = merged_journal.groupby(['ItemNo','OrderNo']).filter(lambda x: x['Description'].str.contains('Insp|Rece').any())
#Groups that have Insp or Rece

df = df[['EntryNo','ItemNo','Description','OutputQuantity','OrderNo','MasterMaterial','MaterialUsed','Label']].drop_duplicate

df = df[df.Description.str.contains('Insp|Rece',na = False)]
#Only the Row with 'Inspect' and 'Receive'

df = df.sort_values(by = 'OutputQuantity', ascending = False).drop_duplicates(['OrderNo'])
# karena ada item yg recevingnya satu-satu, maka diambil row dengan value paling tinggi untuk mengetahui jumlah total

quantity = df.groupby(['ItemNo','OrderNo'])[['OutputQuantity']].sum().reset_index()

# Time Per WO
wo_time = journal.groupby(['ItemNo','OrderNo'])[['TotalTime']].sum().reset_index()

df = pd.merge(df,wo_time, on = ['ItemNo','OrderNo'])

df['TimePerItem'] = df['TotalTime']/df['OutputQuantity']

df = df.drop('Description', axis = 1)

df[df['ItemNo'] =='22B000312N01']
#WO-CF/2309/1700 memproduksi 3 item dengan total waktu 1329 menit, maka per item 443 menit
```

	EntryNo	ItemNo	OutputQuantity	OrderNo	MasterMaterial	MaterialUsed	Label	TotalTime	TimePerItem
9486	876682	22B000312N01	3.0	WO-CF/2309/1700	{'14A190412A00', '12A010400A00'}	{'14A190412A00', '12A010400A00'}	Same As Master	1329.0	443.0

Gambar 3. 36 Menentukan waktu work order

Dicari rata-rata AverageProcessTime dari setiap grup berdasarkan ItemNo, Label, MasterMaterial, dan kolom MaterialUsed yang diubah nama menjadi **GroupAverageProcessTime**, yang disimpana dalam variabel **material_recommendation**. Lalu dibuat fungsi **best_material**, yang akan menciptakan kolom Recommendation yang nilai nya adalah ‘Best’ bagi baris dengan GroupAverageProcessTime paling kecil disetiap grup yang dibagi berdasarkan item dan raw material yang digunakan. Misalnya dalam Gambar 3.37, item 21A000700A02 memiliki empat variasi Material yang digunakan, Dimana tiga diantaranya berbeda dengan master data. Grup yang menggunakan raw material 11B020000A05 dan 11E020000A05 memiliki waktu yang paling singkat, maka memiliki nilai ‘Best’ pada kolom Recommendation.

```
material_recommendation = df.groupby(['ItemNo','Label','MasterMaterial','MaterialUsed'])[['TimePerI
```

```
material_recommendation = material_recommendation.reset_index()
```

```
material_recommendation
```

```
def best_material(df):
    min_index = df['GroupAvgTime'].idxmin()
    df.loc[min_index, 'Recommendation'] = 'Best'
    return df
```

```
material_recommendation = material_recommendation.groupby(['ItemNo']).apply(best_material)
```

```
material_recommendation = material_recommendation.reset_index(drop = True)
```

```
material_recommendation
```

```
material_recommendation = material_recommendation.groupby('ItemNo').filter(lambda x: x['Label'].nun
```

```
material_recommendation
```

	ItemNo	Label	MasterMaterial	MaterialUsed	GroupAvgTime	Recommendation
1	21A000700A02	Alternative	{'11B020000A05'}	{'11B020000A03'}	29.801597	NaN
2	21A000700A02	Alternative	{'11B020000A05'}	{'11B020000A05', '11E020000A05'}	10.343333	Best
3	21A000700A02	Alternative	{'11B020000A05'}	{'11B020000A06'}	30.883333	NaN
4	21A000700A02	Same As Master	{'11B020000A05'}	{'11B020000A05'}	23.203627	NaN

Gambar 3. 37 Rekomendasi raw material

3.3 Kendala yang Ditemukan

1) Penafsiran Dataset

Tidak semua kolom memiliki arti atau hubungan yang jelas antar satu sama lain. Hal ini dapat menciptakan kesalahpahaman dalam menganalisis data. Misalnya dalam dataset BOM Terdapat kolom Production BOM No. dan kolom No, Sehingga kedua kolom tersebut menjadi ambigu.

2) Perbedaan antara jurnal dan *master data*

File jurnal diambil dari tahun 2017 sampai dengan 2024, yang didalamnya terdapat nomor *routing*, nomor *operation*, deskripsi ataupun *Work Center* yang sudah tidak ada di *master data*. Terdapat nomor *routing* yang tidak terdapat di *master data*. Hal ini menyebabkan adanya proses *routing* di *journal* yang tidak bisa dibandingkan dengan *master data*.

3) Informasi waktu produksi yang terpisah-pisah

Proses produksi sebuah item jurnal produksi terpisah kedalam beberapa work order, sehingga waktu untuk memproduksi finished product tidak bisa

diketahui secara langsung. Lalu dalam pembuatan component, di jurnal tidak langsung diketahui raw material yang digunakan.

4) Tidak adanya label penggunaan material alternatif

Dalam journal, tidak ada kolom yang menjelaskan apakah material yang digunakan sama dengan master data, sehingga waktu yang dibutuhkan untuk work order dengan material alternatif dan material sesuai dengan master data tidak bisa langsung diketahui.

3.4 Solusi atas Kendala yang Ditemukan

Bagian ini berisi solusi atas kendala yang ditemukan selama proses kerja magang

1) Komunikasi dengan Supervisor

Kolom yang dianggap memiliki arti yang ambigu ditanyakan kepada supervisor untuk penjelasan lebih lanjut. Misalnya dalam dataset BOM terdapat kolom ProductionBOMNo dan kolom No. Production BOM No memiliki arti *item* atau komponen yang ingin dibuat, sedangkan kolom No berisi material yang digunakan untuk membuatnya. Description merupakan nama dari material dalam kolom No.

2) Memfilter Jurnal Sesuai Dengan Analisis

Karena dataset master tidak memiliki semua informasi proses routing yang terdapat di jurnal, maka row dalam dataset jurnal di filter sesuai dengan analisis yang ingin dilakukan melalui query SQL dengan kombinasi beberapa kolom. Dalam pembuatan rekomendasi Run Time dan Setup Time, variabel **journal_in_master** dibuat dengan kombinasi kolom RoutingNo, OperationNo, Description, dan WorkCenterNo. Hal ini akan memastikan bahwa setiap row di jurnal juga ada informasinya dalam dataset master. Untuk analisis alternatif Work Center, variabel **journal_2** dibuat dengan kombinasi RoutingNo, OperationNo, dan Description. Hal ini memastikan bahwa nomor *routing* di jurnal ada di dalam dataset master, namun tetap dapat menampilkan alternatif Work Center di jurnal yang tidak ada di master.

3) Menggabungkan waktu produksi Work Order sesuai dengan BOM

Untuk mengetahui jumlah waktu produksi suatu item, dihitung semua work order yang berkaitan dengan BOM finished product, lalu dijumlahkan. Namun hal ini memberikan hasil waktu dengan asumsi work order dikerjakan tidak secara bersamaan.

4) Menggabungkan waktu produksi Work Order sesuai dengan BOM

Consumption dan master data digabungkan untuk menentukan apakah kombinasi material yang digunakan dalam consumption sama dengan master data. Jika minimal ada satu material yang berbeda, maka diberi label ‘Alternative’ dan jika sama diberi label ‘Same As Master’, sehingga dapat digabungkan dengan jurnal untuk mengetahui waktu sesuai dengan material yang digunakan.