Solution 1.

- $$\pi_{SId}(\sigma_{Sprice<PUsualPrice/2}(Sale \bowtie Part))$$
- $$\pi_{SId}(\sigma_{Sprice<PUsualPrice/2 \wedge Sale.SId=Part.SId}(Sale \times Part))$$

Solution 2.

1. - It means that in the *Crew* table, every combination of *flightCode* and *date* uniquely determines a single *tailCode*.
   - This would violate the functional dependency because a flight on a specific date cannot be assigned to more than one aircraft.
     - ('QF141', 'Dubai', '2019-10-31', 'VH-EAB', 'City of Canberra', 'Qantas', 65098, 'Joe Bloggs', 'Navigator')
     - ('QF141', 'Dubai', '2019-10-31', 'VH-Different', 'City of Canberra', 'Qantas', 65098, 'Joe Bloggs', 'Navigator')
2. Let the initial result set $S = \{flightcode, empid\}$. We calculate $\{flightcode, empid\}$by repeatedly searching for existing functional dependencies:
   - For $flightCode \rightarrow destination$: $\{flightcode\} \subset S, \{destination\} \not\subset S$. Thus, we add *destination* into S. Now, $S = \{flightcode, emp_id, destination\}$.
   - For $flightCode, date \rightarrow tailCode$: $\{flight, date\} \not\subset S$. We do nothing.
   - For $tailCode \rightarrow acName, airline$: $\{tailCode\} \not\subset S$. We do nothing.
   - For $empid \rightarrow name, title$: $\{empid\} \subset S$, $\{name, title\} \not\subset S$. We add *name* and *tile* into S. Now, $S = \{flightcode, empid, destination, name, title\}$

   There are no more attributes that can be added. Thus, $(flightcode, empid)^+ = (flightcode, empit, destination, name, title)$
3. The relation **Crew** is not in **BCNF**.
   - The FD: $flightcode \rightarrow destination$ violates the **BCNF** because $flightCode^+ = \{flightCode, destination\} \neq R$. In other words, $flightCode$ is not a superkey of R and this dependency is non-trivial.
   - The FD: $flightCode, date \rightarrow tailCode$ violates the **BCNF** because $\{flightCode, date\}^+ = \{flightCode, date, tailCode, destination, acName, airline\} \neq R$. $flightCode, date$ is not a superkey of R and this dependency is non-trivial.
   - The FD: $tailCode \rightarrow acName, airline$ violates the **BCNF** because $\{tailCode\}^+ = \{tailCode, acName, airline\} \neq R$. $tailCode$ is not a superkey of R and this dependency is non-trivial.
   - The FD: $empId \rightarrow name, title$ viaolates the **BCNF** because $\{empId\}^+ = \{empId, name, title\} \neq R$. $empId$ is not a superkey of R and this dependency is non-trivial.$empId$ is not a superkey of R and this dependency is non-trivial.

Since not every functional dependency (FD) satisfies the conditions of **BCNF**, the relation **Crew** is not in **BCNF**.

4. Decomposition:

   - R_1 :
     - $X = \{empId,\ name,\ title\}$
     - Primary Key: $empId$
     - FDs:
       - $empId \rightarrow name,\ title$
   - R_2 :
     - $Y = \{flightCode,\ destination,\ date,\ tailCode,\ acName,\ airline,\ empId\}$
     - Primary Key: $\{flightCode,\ date,\ empId\}$
     - FDs:
       - $flightCode \rightarrow destination$
       - $flightCode,\ date \rightarrow tailCode$
       - $tailCode \rightarrow acName,\ airline$

   Justification:

   (a) $X \cap Y = \{empId\}$

   (b) $\{empId\}^+ = \{empId,\ name,\ title\} = X$

   (c) That is, $X \cap Y \rightarrow X$

   Hence, this decomposition of R into R_1 based on X and R_2 based on columns Y is lossless-join.

Solution 3.

1. The query cannot be answered effectively using the primary index because the primary index is based on *(CId, CPhone)*. Since the query in the *WHERE* clause uses only *CPhone*, which is not the leading attribute of the primary key, the database cannot effectively use the primary index. As a result, the query would be answered through a full table scan. This means the database would go through all the rows, checking each record for the conditions *CPhone* = *'0414441777'* and *CPhoneFee* > *500*, which is inefficient.

2.
```
CREATE INDEX cphone_cphonefee
ON Cust_Phone (CPhone, CPhoneFee);
```

   Explanation: This index will first sort by *CPhone* and then by *CPhoneFee*. When the query runs, the database can quickly locate the rows where *CPhone* = *'0414441777'* and then filter those rows based on *CPhoneFee* > *500*. This significantly reduces the number of records that need to be scanned, leading to faster execution.

Solution 4.

1. The database account password is stored in plain text in the *config.ini* file. In the *database_connect()* function within *database.py*, a *ConfigParser* is first initialized and used to read information such as the hostname and password from the *config.ini* file. Finally, this information is passed to the DBMS to establish a connection.

   Security

   - The password is stored in the *config.ini* file instead of being hardcoded in the program code, which avoids directly exposing the password in the code.
   - The client needs to be on the school network or connected through the school VPN to access the DBMS. The VPN will encrypt the transmitted data, so even if an attacker intercepts it, they won't be able to easily understand the content.

2. weakness:

   - The password is stored in plain text in the *config.ini* file. If someone gains access to this file, they can directly read the password.

   How to improve:

- Store the password in an encrypted format in the *config.ini file*, rather than in plain text. The password can be decrypted in the *database_connect()* before being passed to *pg8000.connect()*.
- Ensure that the database connection uses SSL/TLS encryption to protect the password during transmission.