Household Internet of Things Database Design

1.1 Business functions and use case of Household IoT app

To realize home automation and smart homes, the Household Internet of Things app collects a large amount of data through various smart devices in the house. The data of the home Internet of Things reflects the real state of the user and the house under certain conditions (for example, smart blood pressure monitor and environmental detectors), which contains a lot of valuable information that can potentially provide personalized services for the home user.

However, the value of home IoT data cannot be realized without the storage, management, and processing of data in the back end. We need to design data storage solutions to ensure continuous data collection and provide data support for various home service applications of the Household IoT. In IoT app, we focus on how to store and analyze the data collected by a user's device.

1.2 The choice of databases

Characteristics of data:

1) **Huge amount:** Home IoT app is aimed at thousands of home users, and there are several smart devices in each home, which are used to collect data on the behaviors and living conditions of households. Air purifiers, for example, collect air quality data 24 hours a day through built-in sensors. So, there's a huge amount of data in the home IoT.
2) **Multi-source heterogeneous:** Smart devices in the Home Internet of Things are different, each device is a source of information, so the data in the home Internet of Things is multi-source. At the same time, the format and semantics of data collected by different devices are different, so the data is heterogeneous. Multi-source heterogeneous is the most prominent feature of IoT data.
3) **Spatial and temporal correlation:** The perceived data of the home IoT reflects the real state of the user at a certain moment and the real state of the family under certain conditions. Therefore, these perceived data have Spatial and temporal correlation attributes.

Why not just use RDBMS

The relational database is not suitable for multi-source heterogeneous data collected by the device. If we choose an RDBMS to store heterogeneous data from multiple sources, there are two main approaches: one is to store tables from different devices independently, and the other is to store all the data in a single table. If we take the first approach: when a new smart device is added to the home IoT app, we need to create a new table for that smart device and define the fields of the table based on the collected data. However, with the development of the IoT, different smart devices are rapidly increasing, and this storage method will increase the difficulty of table management. If we take the second approach: when a new device is added, the original table is expanded by adding new fields (explicitly changing the table structure). However, in the case of a large number of devices, the redundant data is too much, and we have to know the new device information and the data collected by the new device in advance.

Why use MongoDB

Therefore, no matter what storage scheme is adopted for the relational database, it is difficult to meet the data storage requirements of IoT app. MongoDB in the NoSQL database provides a Schema-free storage mode, which is different from the RDBMS. This storage mode can easily meet the storage requirements of multi-source heterogeneous data in the IoT app.

1) MongoDB supports storage and query of huge amounts of data: MongoDB supports Master/Slave mode, replica-set mode, and sharding technology. With these technologies, MongoDB can realize Distributed Cloud Storage which can solve the storage and processing problem of high Concurrency. In addition, MongoDB can solve the problem of access efficiency of huge amounts of data through sharding technology. According to official documents, when the amount of data exceeds 50G, MongoDB's database access speed is more than 10 times that of MySQL, which can well meet the real-time access of data by the IoT app. And the query language supported by MongoDB is very powerful, which can easily query the embedding objects and arrays in the documents in depth. It also supports the establishment of indexes for these embedding data, such as compound index and TTL index.
2) MongoDB supports the storage of spatial and temporal correlation data: the time series model in MongoDB can store massive stream data such as environmental quality and GPS in the home Internet of Things in real time.
3) MongoDB supports the storage of multi-source heterogeneous data: In the data model of MongoDB, a document is a group of ordered key-value pairs. The value of key-value pairs can be arrays, objects, or even other documents, so complex data types can be stored. The document is the basic unit of data in MongoDB. During the database design, there is no need to define the attribute values of the document in advance, and key-values can be flexibly added and deleted in the document as required.

Therefore, MongoDB can meet the storage requirements of data collected by devices in the IoT app.

Why combine MongoDB and SQLite

Data between users, addresses and devices have the characteristics of multi-row transactions and complex joins. However, MongoDB does not support transactions. And when the data is highly relational and structured, MongoDB is not the best choice. Relational database SQLite is more suitable for databases that require a large number of atomically complex transactions. Because of its explicit architecture, SQLite creates reliable database structures by using tables, making the corresponding ones sufficiently query and easy to search. In addition, SQLite ensures consistency of the data and enables complex queries through JOIN. Data redundancy can be reduced through normal forms.

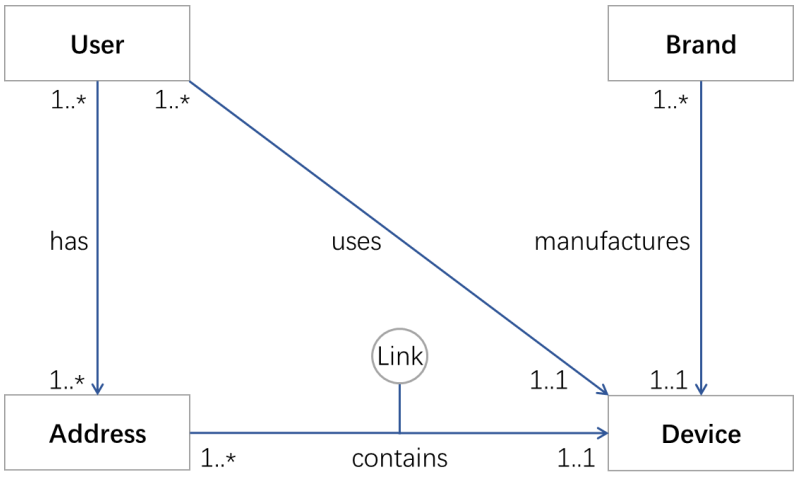
Therefore, we store the data of users and devices in SQLite and store the massive multi-source and heterogeneous data collected by devices in MongoDB.

1.3 Application example

This IoT database can be used to analyze and improve indoor environment quality, to predict the electricity consumption of the home in the next week based on time series, and to intelligently adjust the brightness of indoor lighting based on the user's habits. This paper takes environmental quality data as an example to discuss how to store user, device, address,

environmental quality, and other data, then establish React Program to improve indoor environmental quality and generate environmental quality reports for users through data analysis.

2.1 RDBMS database ER diagram design



Suggested SQLite ER Diagram for Household IoT (User-Device data)

2.2 Logical Database Design

Entity	attribute name	Description	Domain attribute and/or Data type	Null	Multi values
User	**user_id {PK}**	Uniquely identifies each user	Number	No	No
	user_name	Name of the user	Text (30 characters)	No	No
	user_gender	Gender of the user	M(male) or F(female)	Yes	No
	user_phone [AK]	mobile-phone number	Maximum of 10 characters	No	No
	user_email[AK]	Email of user	Text (30 characters) and must include '@'	No	No
	user_dob	Date of birth	Date format	Yes	No
	user_height	Height of user	Height number (cm)	Yes	No
	user_weight	Weight of user	Weight number (kg)	Yes	No

Device	**device_id {PK}**	Uniquely identifies each device	Number	No	No
	device_type	Type of device	Text (30 characters)	No	No
	device_model	Model of device	Text (30 characters)	No	No
	device_name	Custom name of device	Text (30 characters)	Yes	No
	device_status	Status of device	0 or 1 (connected or disconnected)	No	No

Address	**address_id {PK}**	Uniquely identifies each address	Number	No	No
	address_city	City of address	Text (30 characters)	No	No
	address_postcode	Postcode of adress	Text (10 characters)	Yes	No
	address_detail	detailed address	Text (30 characters)	Yes	No

Brand	**brand_id {PK}**	Uniquely identifies each brand	Number	No	No
	brand_name	Name of brand	Text (30 characters)	No	No
	brand_adress	Adress of brand	Text (30 characters)	Yes	No
	brand_phone [AK]	Phone number of brand	Maximum of 10 characters	No	No

DBDL

User (uses) Device - 1:* relationship

User (user_id, user_name, user_gender, user_mobile, user_email, user_dob, user_height, user_weight)
Primary Key user_id
Alternate Key user_mobile, user_email

Device (device_id, device_type, device_model, device_name, device_status, user_id, brand_id, address_id)
Primary Key device_id
Foreign Key user_id references User (user_id)
Foreign Key address_id references Address (address_id)
Foreign Key brand_id references Brand (brand_id)

User (has) Adress - *:* relationship

User (user_id, user_name, user_gender, user_mobile, user_email, user_dob, user_height, user_weight)
Primary Key user_id
Alternate Key user_mobile, user_email

Adress (address_id, address_city, address_postcode, address_detail, user_id)
Primary Key address_id
Foreign Key user_id references User (user_id)

Brand (manufactures) Device - 1:* relationship

Brand (brand_id, brand_name, brand_adress, brand_phone)
Primary Key brand_id
Alternate Key brand_phone

Device (device_id, device_type, device_model, device_name, device_status, user_id, brand_id, address_id)
Primary Key device_id
Foreign Key user_id references User (user_id)
Foreign Key address_id references Address (address_id)
Foreign Key brand_id references Brand (brand_id)

Adress (contains) Device - 1:* relationship

Adress (address_id, address_city, address_postcode, address_detail, user_id)
Primary Key address_id
Foreign Key user_id references User (user_id)

Device (device_id, device_type, device_model, device_name, device_status, user_id, brand_id, address_id)
Primary Key device_id
Foreign Key user_id references User (user_id)
Foreign Key address_id references Address (address_id)
Foreign Key brand_id references Brand (brand_id)

Link Table Identified
Device_Address (address_id, device_id, adStarttime, adWorkhours)
Primary Key address_id, device_id
Foreign Key address_id references Address (address_id)
Foreign Key device_id references Device (device_id)

Include referential integrity in the DBDL statements

Table 1

User (user_id, user_name, user_gender, user_mobile, user_email, user_dob, user_height, user_weight)
Primary Key user_id
Alternate Key user_mobile, user_email

Table 2

Adress (address_id, address_city, address_postcode, address_detail, user_id)
Primary Key address_id
Foreign Key user_id references User (user_id) on **Update Cascade** on **Delete Cascade**

Table 3

Brand (brand_id, brand_name, brand_adress, brand_phone)
Primary Key brand_id
Alternate Key brand_phone

Table 4

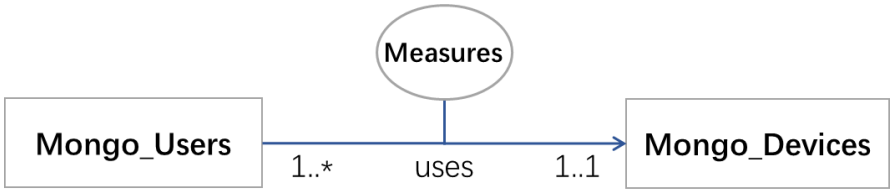
Device (device_id, device_type, device_model, device_name, device_status, user_id, brand_id, address_id)
Primary Key device_id
Foreign Key user_id references User (user_id) on **Update Cascade** on **Delete Cascade**
Foreign Key address_id references Address (address_id) on **Update Cascade** on **Delete Set Null**
Foreign Key brand_id references Brand (brand_id) on **Update Cascade** on **Delete Set Null**

Table 5

Device_Address (address_id, device_id, adStarttime, adWorkhours)
Primary Key address_id, device_id
Foreign Key address_id references Address (address_id) on **Update Cascade** on **Delete Cascade**
Foreign Key device_id references Device (device_id) on **Update Cascade** on **Delete Cascade**

3.1 MongoDB database ER Diagram design

The household IoT app uses MongoDB to store all data of connected smart devices, as well as data collected by devices, and analyze this data in multiple dimensions. In household IoT app, the relationship between users and devices is one to many. So we built the following MongoDB ER Diagram.



Suggested MongoDB ER Diagram for Household IoT (Measures data)

3.2 Document model design

According to the principle of entity transformation, user collection and device collection can be obtained. The following table shows the document data model for the household IoT app. In this model, the user collection used the Embedded Document Pattern stores the device collection, and The measure collection used The Parent References Pattern stores user collection and device collection. In this document model, when we need to query which devices the user owns, we only need to query once to get the result.

Collection	Relationship	format
Mongo_Devices	Device (by) User - 1:1 relationship	{device_id: '40001'}
Mongo_Users	User (uses) Device - 1:* relationship Embedded Document pattern	{user_id: '10001', Mongo_Devices: [{device_id: '40001'}, {device_id: '40002'}, {device_id: '40003'}]}
Measures	The Parent References pattern	{Mongo_Users: user_id('10001'), Mongo_Devices: device_id ('40001'), datas: {}}

In MongoDB, we created Measures Collection to store the massive multi-source heterogeneous data collected by devices. In the application of IoT, the environmental quality detector generate a continuous data stream. This paper takes the measured data of the environmental quality detector as an example to design the storage of measured streaming data based on time series.

The model consists of groups of embedded documents. In the IoT app, it is assumed that the environmental quality detector collects PM2.5, temperature, humidity, noise, carbon dioxide concentration, and other data every 200ms, that is, the minimum time granularity of the measured data is 200ms. To make data management and acquisition more natural, the embedded layers can be divided into units of hours, minutes, and seconds. The specific document structure is as follows:

1) One document per hour.

2) Field: user_id, device_id, Timestamp, datas.

3) Value: Embedded set of subdocuments, 60*5 subdocuments per minute, where 5

subdocuments are generated every second, and one document is generated every 200ms containing an array of key-value pairs for PM2.5, temperature, humidity, noise, CO2 concentration, and AIR quality level.

Measures (example):

```
{
Mongo_Users: user_id('10001'),
Mongo_Devices: device_id ('40001'),
Timestamp: ISODate("2021-07-01T00:00:00Z"),
datas: {
   m1: {
      s1: {pm25: 90, tem: 19, hum: 35, voc: 9, co2: 14, level: 79}
      …,
      s300: {pm25: 95, tem: 17, hum: 35.5, voc: 9, co2: 17, level: 79}
      }
   m2: {
      s1: {pm25: 90, tem: 19, hum: 35, voc: 9, co2: 14, level: 79}
      …,
      s300: {pm25: 95, tem: 17, hum: 35.5, voc: 9, co2: 17, level: 79}
      }
   …,
   m60: {
      s1: {pm25: 90, tem: 19, hum: 35, voc: 9, co2: 14, level: 79}
      …,
      s300: {pm25: 98, tem: 17, hum: 35.5, voc: 9, co2: 17, level: 79}
      }
   }
}
```

The advantage of this format is that each document can correspond to a unit of time, which makes the management and retrieval of data more natural. And the document-oriented design pattern enables the document to be inferred from the current time so that data insertion performance is more efficient. In addition, in this time series model, all documents within that hour can be read according to the time hour, so this batch reading of documents is also more efficient.

4. Implementation of polyglot persistence

Establish the React Program to improve indoor environmental quality and generate environmental quality reports for users through data analysis.

Goal One: Improve the indoor environmental quality of the 20001 address of 10001 users through the React Program:

1. Use Device table in SQLite to query the device ID of "Purifier" where address_id is 20001.

```
# find the Purifier device_id of address 20001 in Sqlite
# Prepare the query String
qry="SELECT device_id FROM Device where address_id=20001 and device_type='Purifier';"

# Execute query on SQLite
cur.execute(qry)

# Fetch and display one row
row=cur.fetchall()

print (row)
```

```
[(40001,)]
```

2. In MongoDB's Measure Collection, find the environmental quality data (embedded document) through Purifier device ID queried in step 1.

```
# Query the device_id 40001(Purifier)  data
# from 2021-07-01T00:00:00Z to 2021-07-01T01:00:00Z in Mongodb
myresult=IoTdb.Measures.aggregate(
    [{"$match": {'Mongo_Devices': 'device_id("40001")',}},
     {"$project": {"datas": '$datas',"_id": 0,'Timestamp':1}}
    ])
```

3. Query the latest environmental quality data record in the Embedded document and convert the document to list format.

```
# Converting cursor to the list of dictionaries
list_cur = list(myresult)
# Query the latest environment quality data
latest_data= list_cur[1]['datas']['m60']['s300']
latest_data
```

```
{'pm25': 45, 'tem': 16.1, 'hum': 29.5, 'voc': 2, 'co2': 60, 'level': 79}
```

4. Use Python to create a React Program to improve the current environment quality immediately when detecting the latest environment quality data deviating from the normal range.

Indicators	Description	Condition	Reaction
pm25	PM2.5 concentration	>37.5	Enable the PM2.5 filter function
		<=37.5	Indoor PM2.5 concentration is normal
tem	Temperature	<23	Enable the heating function
		>=23, <=28	Indoor temperature is comfortable
		>28	Enable the cooling function
hum	Humidity	<30	Enable the humidifier function
		>=30, <=60	Indoor Humidity is normal
		>60	Enable the dehumidification function
co2	Co2 concentration	>80	Remind users of ventilation
		<=80	Indoor co2 concentration is normal

```python
# Design a program to react on different environmental quality data
def ReactProgram(data):
    if data['pm25']>37.5:
        print('Enable the PM2.5 filter function')
    else:
        print('Indoor PM2.5 concentration is normal')
    if data['tem']<23:
        print('Enable the heating function')
    elif data['tem']<28:
        print('Indoor temperature is comfortable')
    else:
        print('Enable the cooling  function')
    if data['hum']<30:
        print('Enable the humidifier function')
    elif data['hum']<60:
        print('Indoor Humidity is normal')
    else:
        print('Enable the dehumidification function')
    if data['co2']>80:
        print('Remind users of ventilation')
    else:
        print('Indoor co2 concentration is normal')
    return
```

```python
# Apply the ReactProgram
ReactProgram(latest_data)
```

```
Enable the PM2.5 filter function
Enable the heating function
Enable the humidifier function
Indoor co2 concentration is normal
```

Goal Two: Generate an indoor environmental quality report

1. Convert the dictionary format document obtained in Goal 1 step2 to the data frame using the 'for' loop. Next, transpose the data frame to fit the Seaborn diagram, and then change the horizontal axis of the data frame to a sequence of numbers to get the final data frame.

```python
# Convert Query data to Dataframe and analyze the Dataframe
dict_cur1 = list_cur[0]['datas']
dict_cur2 = list_cur[1]['datas']
a = []
for t in dict_cur1:
    a.append(pd.DataFrame.from_dict(dict_cur1[t]))
for t in dict_cur2:
    a.append(pd.DataFrame.from_dict(dict_cur2[t]))

result1 = pd.concat(a, axis=1)
result1.columns = range(1, 19)
result1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
pm25	42.0	42.0	43.0	43.5	43.0	42.5	42.5	44.0	42.0	42.0	43.0	44.0	43.0	41.0	42.0	44.0	46.0	45.0
tem	16.5	16.5	16.6	16.5	16.5	16.8	16.6	16.7	16.6	16.5	16.4	16.8	16.7	16.5	16.4	16.3	16.2	16.1
hum	29.0	29.8	29.5	29.9	30.0	29.6	29.3	29.9	30.0	29.0	28.0	29.0	29.6	29.5	29.4	29.3	29.8	29.5
voc	9.0	9.0	4.0	8.0	9.0	8.0	3.0	3.0	3.0	9.0	9.0	9.0	10.0	2.0	2.0	2.0	2.0	2.0
co2	60.0	60.0	58.0	60.0	60.0	62.0	59.0	60.0	60.0	60.0	62.0	63.0	65.0	64.0	62.0	65.0	63.0	60.0
level	79.0	79.0	79.0	79.0	79.0	79.0	79.0	79.0	79.0	79.0	79.0	80.0	79.0	79.0	79.0	79.0	79.0	79.0

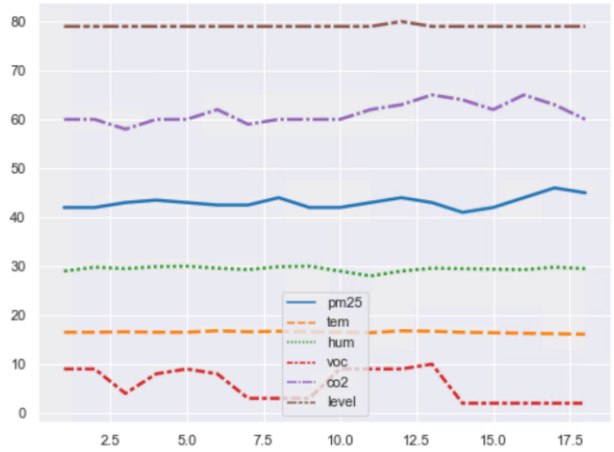
```
result2 = result1.T
result2
```

	pm25	tem	hum	voc	co2	level
1	42.0	16.5	29.0	9.0	60.0	79.0
2	42.0	16.5	29.8	9.0	60.0	79.0
3	43.0	16.6	29.5	4.0	58.0	79.0
4	43.5	16.5	29.9	8.0	60.0	79.0
5	43.0	16.5	30.0	9.0	60.0	79.0
6	42.5	16.8	29.6	8.0	62.0	79.0
7	42.5	16.6	29.3	3.0	59.0	79.0
8	44.0	16.7	29.9	3.0	60.0	79.0
9	42.0	16.6	30.0	3.0	60.0	79.0
10	42.0	16.5	29.0	9.0	60.0	79.0
11	43.0	16.4	28.0	9.0	62.0	79.0
12	44.0	16.8	29.0	9.0	63.0	80.0
13	43.0	16.7	29.6	10.0	65.0	79.0
14	41.0	16.5	29.5	2.0	64.0	79.0
15	42.0	16.4	29.4	2.0	62.0	79.0
16	44.0	16.3	29.3	2.0	65.0	79.0
17	46.0	16.2	29.8	2.0	63.0	79.0
18	45.0	16.1	29.5	2.0	60.0	79.0

2. According to the data frame, use Seaborn to draw the line chart of environmental quality change.

```
#  Draw a line chart of air environment changes by the data of Purifier
sns.set_style(style="whitegrid")
sns.set(rc={"figure.figsize":(8, 6)})
sns.lineplot(data=result2, palette="tab10", linewidth=2.5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e34eda7438>
```



3. Join Device and Address table in SQLite to query the detailed information of user 10001 and summarize it as user_id, device_id, address_id, address_city. And convert the retrieved list to a data frame

```
# find the address information of 40001 device
# Prepare the query String
qry='''SELECT Address.user_id, Address.address_id, address_city, device_id
       FROM Address JOIN Device on Address.address_id = Device.address_id
       where device_id=40001;'''

# Execute query on SQLite
cur.execute(qry)

# Fetch and display one row
row=cur.fetchall()

print (row)
```

```
[(10001, 20001, 'Exeter', 40001)]
```

```
data = pd.DataFrame(row)
data.columns = ['user_id', 'address_id', 'address_city', 'device_id']
data
```

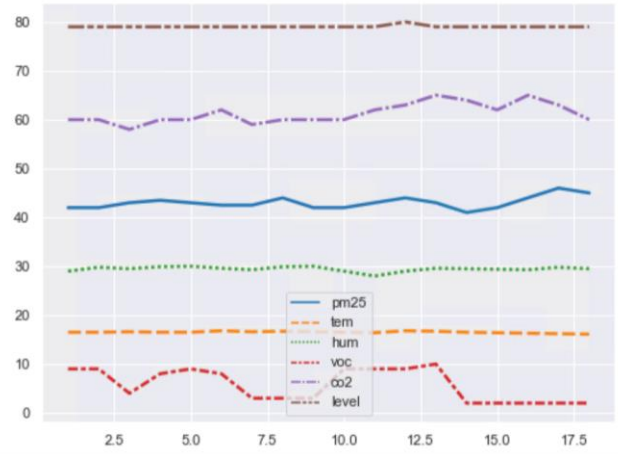
	user_id	address_id	address_city	device_id
0	10001	20001	Exeter	40001

4. Calculate the average value of each indicator with the data frame obtained in Goal 2 step2, and combine it with the user information data frame.

```
# Calculate the average value of each indicator, and insert the value to the final dataframe
data['avg_pm25'] = result2['pm25'].mean()
data['avg_hum'] = result2['hum'].mean()
data['avg_voc'] = result2['voc'].mean()
data['avg_co2'] = result2['co2'].mean()
data['avg_level'] = result2['level'].mean()
data
```

	user_id	address_id	address_city	device_id	avg_pm25	avg_hum	avg_voc	avg_co2	avg_level
0	10001	20001	Exeter	40001	43.027778	29.45	5.722222	61.277778	79.055556

5. Combine the final data frame and line chart to generate the average environmental quality report.



	user_id	address_id	address_city	device_id	avg_pm25	avg_hum	avg_voc	avg_co2	avg_level
0	10001	20001	Exeter	40001	43.027778	29.45	5.722222	61.277778	79.055556