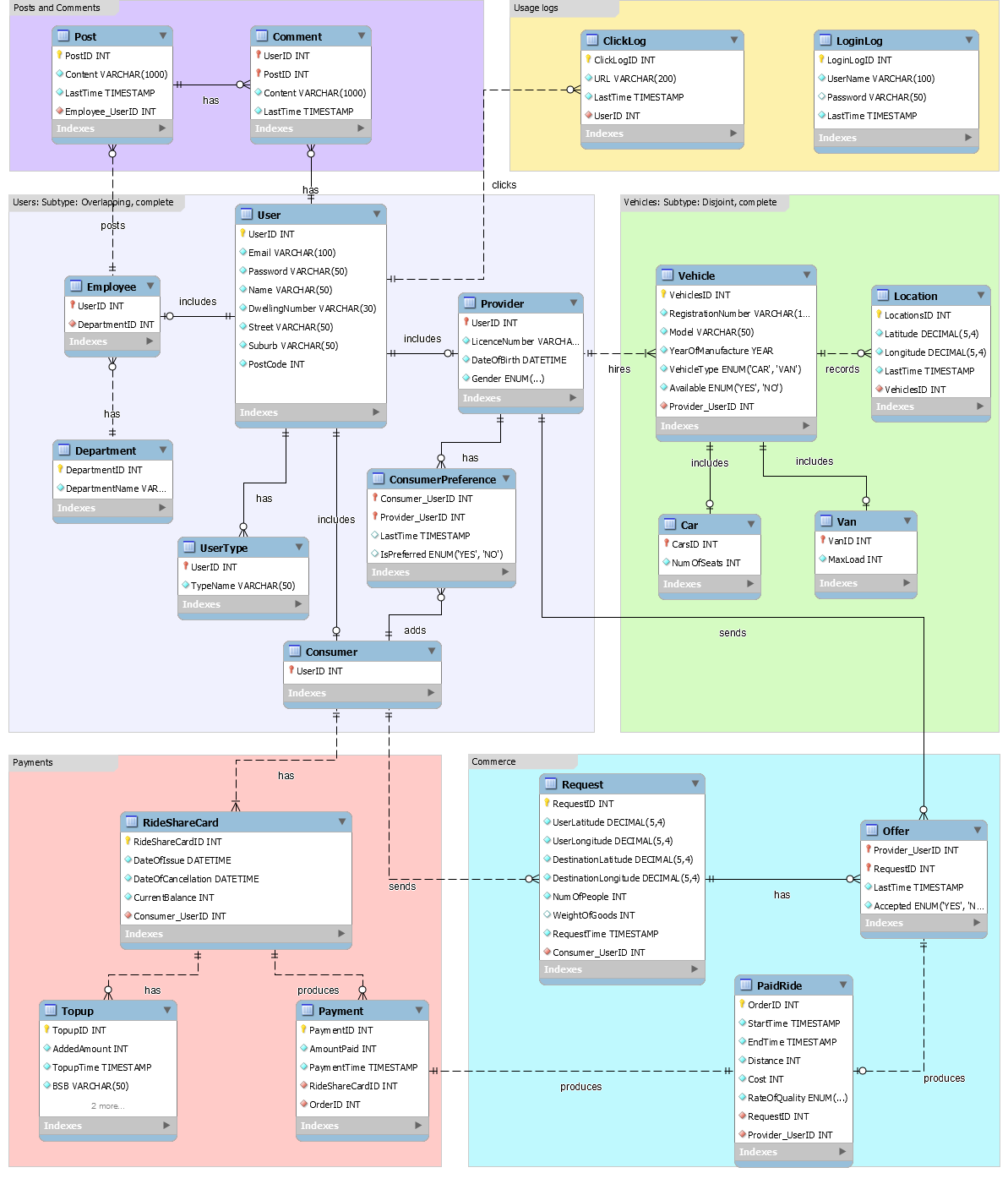
INFO90002 2015 semester 2: Assignment 1

Case study: “RideShare”

**1. Entity Relationship Diagram**



**2. Assumptions**

**1) About Users**

According to the case, there are three kinds of users that are mentioned. Therefore we build the ‘User’ table and three more tables are going to be established which are ‘Employee’, ‘Provider’ and ‘Consumer’. We assume that only these three roles can be involved in, so the relationship between supertype and subtype belongs to complete. These three tables have the same primary key with ‘User’. In addition, a table named ‘Department’ is below ‘Employee’. Moreover, we plan to create a ‘ConsumerPreference’ table to show the information of consumers’ preference.

Due to the instruction that a person can hold more than one role, we used to add an attribute called ‘UserType’ to represent this kind of features, and marked the entity ‘User’ with two primary keys (‘UserType’, ‘UserID’) as well as an one-to-many self-relationship by using the principle of overlapping. However, this plan will generate data redundancy violating the principle of Second Normal Form, because each person would hold three roles in this system, and the profile information of each role could be the same. Therefore, we separate the entity ‘User’ into two entities, ‘User’ and ‘UserType’ to implement the feature that a person can simultaneously hold more than one role.

**2) About Vehicles**

From the case, it is not difficult to draw four tables, which are ‘Vehicle’, ‘Car’, ‘Van’ and ‘Location’. The table ‘Vehicle’ is the main table in the part of vehicles. As the recorded location use both latitude and longitude, so we considered to use DECIMAL as its data type.

Furthermore, in this data model, only two kinds of vehicles can be classified, so we assume that the relationship between supertype and subtype is complete. Then we decided to create two tables, which are ‘Car’ and ‘Van’ with different attributes. Also, by using the principle of disjoint, a vehicle can only be one of subtypes in this data model. Therefore, the part of vehicles needs to establish four tables in order to describe clearly the relationship of location and vehicles.

**3) About Commerce**

According to the aim of this application, allowing people to arrange their travel or transport with some private vehicles on very short notice, we assume that this system should be real-time to provide some instant and valid notices to users or providers. The three assumptions of commerce we considered are given as some real cases.

For sending requests, first of all, the rideshare users should use their smart mobile devices to send request instead of using computers or laptops, and those mobile devices should accurately access users’ current locations showing them on a map. In this case, 4 attributes are designed for representing the user’s location and destination’s location in Request entity.

Secondly, we assume that the nearby providers have to send offers to each request within 2 minutes, and users who sent a request within this 2 minutes need to stay at the current location until one of those providers offers them. If one of requests has no any responses within the 2 minutes, this request will be invalid and cannot be offered any more. The client application will prompt users to send a new request again or show a dialog to notice that there are no available rideshare services nearby. Also, if providers intend to offer this kind of invalid requests, the server will prompt them that the current request has been canceled and discard the relevant offers. The reason for this assumption is considered, as the majority of users usually do not have patience to wait those offers sent more than 2 minutes approximately. The Server can use current system time to subtract the recorded request time to calculate which requests are within the given patient time (2 minutes).

Thirdly, for sending offers, the nearby providers must have at least one of their vehicles that are available for the current request. If one of offers has been accepted, the number of people who will be travelling and the weight of extra goods can not be changed any more. Otherwise, this match should be judged as incorrect data, and providers can apply server to cancel the offer.

**4) About Payments**

According to this study case, we assume that each ride contract should be paid by using only one RideCard, and can not be paid by several cards.

Also, the current balance of each card can be negative by some reasons. For example, since the current balance can not afford the cost of once ride contract, the payment can also be approved. In this case, the current balance will be negative, and users can not use this application for requesting any services until the next top-up. The relevant attribute, ‘CurrentBalance’ in Topup entity, is designed as an unsigned value.

**5) About Blog**

For designing this part of entities’ relationships, we used to write one entity, ‘Post’, to represent the relationship related to Employee entity and User entity, and to make a self-relation to implement the feature that many users can comment one post simultaneously. The reason for this consideration is related to reduce data redundancy according to the principles of normal form. However, according to the case, only employee can send a post in common, and any users can comment posts. Therefore, we separate the ‘Post’ entity to two tables, ‘Post’ and ‘Comment’.

Additionally, we assume that the length of each post or comment will be less than 1000 varchar type, because this length can be sufficient to represent the meaning of the case, a paragraph or two of text.

**6) About Usage logs**

As can be seen from the above ER diagram, the entity ‘LoginLog’, it does not have any relationship to refer to other entities. The reason for this design is to consider the security of the system. According to the case, the entity ‘LoginLog’ needs to record every single login attempts, even using the invalid usernames to attempt the login action. If the relationship between ‘User’ and ‘LoginLog’ exists, hackers could easily attack the database and steal the data via simulating the login action. Therefore, we decide to release the relationship between these two entities for protecting the database.

**3. Data Dictionary**

**List of entities**

|  |  |  |  |
| --- | --- | --- | --- |
| Entities | Entity Type | Primary Key | # Attributes |
| User | Independent | UserID | 8 |
| UserType | Dependent | UserID | 2 |
| Employee | Dependent | UserID | 2 |
| Provider | Dependent | UserID | 4 |
| Consumer | Dependent | UserID | 1 |
| ConsumerPreference | Dependent | Consumer\_UserID, Provider\_UserID | 4 |
| Department | Independent | DepartmentID | 2 |
| Vehicle | Independent | VehiclesID | 7 |
| Car | Dependent | CarsID | 2 |
| Van | Dependent | VanID | 2 |
| Location | Independent | LocationsID | 5 |
| Request | Independent | RequestID | 9 |
| Offer | Dependent | Provider\_UserID, RequestID | 4 |
| PaidRide | Independent | OrderID | 8 |
| RideShareCard | Independent | RideShareCardID | 5 |
| Topup | Independent | TopupID | 6 |
| Payment | Independent | PaymentID | 5 |
| Post | Independent | PostID | 4 |
| Comment | Dependent | UserID, PostID | 4 |
| ClickLog | Independent | ClickLogID | 4 |
| LoginLog | Independent | LoginLogID | 4 |

**Entity ‘User’**

**Description:**

The entity of users is one of the main entities in this case. It shows several of information about users. The information is ‘Not Null’ and very significant for users’ login. When a user login, he or she need to fill in some personal information and the email address as well as the password are used for login operation. Because this is a vehicle sharing application, the living address is necessary such as dwelling number, street and postcode.

**Attributes:**

**Entity ‘UserType’**

**Description:**

This entity is used for recording the type of users. According to the case, there are three kinds of users, which are employee, provider and consumer. Sometimes one user can have more than one role, so this entity is very essential to establish.

**Attributes:**

**Entity ‘Employee’**

**Description:**

This entity has two attributes, which are ‘UserID’ and ‘DepartmentID’. It is one of the user types. The entities of three user types are all have the same primary key.

**Attributes:**

**Entity ‘Provider’**

**Description:**

This entity of provider is to record each provider’s personal information, including ‘LicenceNumber’, ‘DateOfBirth’, ‘Gender’ and ‘UserID’. It is one of the user types. The entities of three user types are all have the same primary key.

**Attributes:**

**Entity ‘Consumer’**

**Description:**

This entity has only one attribute that is ‘UserID’. It is one of the user types. The entities of three user types are all have the same primary key.

**Attributes:**

**Entity ‘ConsumerPreference’**

**Description:**

The entity of consumer preference is to record whom the consumers prefer or not prefer. It has two primary keys. The consumer can add a new provider to their willing list and it will be recorded the time when this change happened.

**Attributes:**

**Entity ‘Department’**

**Description:**

This entity records the information of each department that the employees work for. Several departments are added so far and more new departments are going to be increased in the future. Both of two attributes need to be ‘Not Null’.

**Attributes:**

**Entity ‘Vehicle’**

**Description:**

This entity shows some necessary attributes that provider should provide, such as registration number, model, vehicle type and so on. All of the attributes should be ‘Not Null’ because they are all important information not only for providers but also for employees. Only the attribute of ‘VehicleID’ needs to be unique.

**Attributes:**

**Entity ‘Car’**

**Description:**

The entity of car is a small entity just for recording the ‘CarsID’ which is unique and the number of seats that the car have in order to hire a car with different demands conveniently.

**Attributes:**

**Entity ‘Van’**

**Description:**

The entity of van is a small entity for recording the ‘VansID’, which is unique, and the maximum loading weight in kilograms.

**Attributes:**

**Entity ‘Location’**

**Description:**

This entity is used for recording different kinds of information for vehicles’ location so that the employees and consumers can know where these vehicles are and which one is the closest. Some attributes, such as ‘latitude’ and ‘longitude’, need change to ‘DECIMAL’. All of the attributes are ‘Not Null’ and only ‘LocationID’ is unique.

**Attributes:**

**Entity ‘Request’**

**Description:**

This entity represents a request sent from a user who is going to request a ride service. It will hold a history list of where each request goes over time. Therefore, the relationship between ‘User’ and ‘Request’ is one-to-many. Also, it will be related to the ‘Offer’ entity to represent a one-to-many relationship.

**Attributes:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Attribute | Data Type | NN | UQ | UN | Description |
| PK | RequestID | INT | YES | YES | NO | This is the primary key of ‘Request’, unique in the lifetime of this database. When a user sends a ride request, one series for integer type will be inserted into this cell. The generated algorithm of series should be encrypted for security, or it can be an automatically increasing number for storing simply. |
|  | UserLatitude | DECIMAL(5,4) | YES | NO | NO | Holds the current latitude of users, considering 4 decimal places that can be sufficient to identify the data accurately, generated from the local device’s GPS service. |
|  | UserLongitude | DECIMAL(5,4) | YES | NO | NO | Holds the current longitude of users, combining with the latitude to represent where users are, and to notice the server to find the nearby providers. |
|  | DestinationLatitude | DECIMAL(5,4) | YES | NO | NO | Stores the latitude of the destination. The format is the same as the user’s latitude. |
|  | DestinationLongitude | DECIMAL(5,4) | YES | NO | NO | Stores the longitude of the destination, combining with the latitude to represent the location where users will travel. |
|  | NumOfPeople | INT | YES | NO | YES | The number of ride people must be given when users send a request, because the value will be a key word for searching appropriate providers by the server. Also, according to our above assumptions, this value can not be changed after sending a request. It is unsigned, and the default value is 1. |
|  | WeightOfGoods | INT | NO | NO | YES | Like the numbers of ride people, this value will also be a key word for searching appropriate providers by the server. It can be null or an unsigned integer. |
|  | RequestTime | TIMESTAMP | YES | NO | NO | Records the request time. Also, according to the assumptions, it can be calculated a patient time by the server to identify whether the request is available or not for providers. |
| FK | Consumer\_UserID | INT | YES | NO | NO | A foreign\_key, coming from the entity ‘User’ |

**Entity ‘Offer’**

**Description:**

This is an Associate Entity between ‘Provider’ and ‘Request’, allowing for multiple providers per request or vice versa, using their respective primary keys. Also, it will be related to the entity ‘PaidRide’ for a one-to-one relationship.

**Attributes:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Attribute | Data Type | NN | UQ | UN | Description |
| PFK | Provider\_UserID | INT | YES | YES | NO | A foreign\_key, coming from the entity ‘Provider’ |
| PFK | RequestID | INT | YES | YES | NO | A foreign\_key, coming from the entity ‘Request’ |
|  | LastTime | TIMESTAMP | YES | NO | NO | This value represents the last update time recording from the server’s system time. So, this data type of this value is TIMESTAMP. For first generated time, it records the time provider sent. When one of the offers is accepted, it will be updated. |
|  | Accepted | ENUM('YES', 'NO') | YES | NO | NO | This value is designed as an ENUM type. For each offer sent from a provider, this value represents the status of it. Once the offer is accepted, it will be marked with ‘YES’. Otherwise it will still be ‘NO’. The default of this value is ‘NO’. |

**Entity ‘PaidRide’**

**Description:**

For each accepted offer by users, it will generate one unpaid contract to users after travelling, guiding users to pay the contract. Then, a paid ride order will be produced into the database. Also, this entity will be only related to one payment recorded in ‘Payment’ entity.

**Attributes:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Attribute | Data Type | NN | UQ | UN | Description |
| PK | OrderID | INT | YES | YES | NO | This is the primary key of ‘PaidRide’. Once users successfully make a payment for the ride service, it will be generated by the server automatically. |
|  | StartTime | TIMESTAMP | YES | NO | NO | This attribute holds the start time of the current ride service. |
|  | EndTime | TIMESTAMP | YES | NO | NO | This attribute holds the end time of the current ride service. |
|  | Distance | INT | YES | NO | YES | The travelled distance of the current ride service will be calculated by the server automatically. The minimum unit of this value is meter, because this is sufficient resolution to track the distance of each vehicle. For display, it can be formatted as kilometer. Also, this value is not null and unsigned. |
|  | Cost | INT | YES | NO | YES | Records the number of cost for each service, and the unit of this value is Australian Dollar, according to the case that only Australian users can use this application to arrange their travel. |
|  | RateOfQuality | ENUM('1', '2', '3', '4', '5', '6', '7', '8', '9', '10') | YES | NO | NO | This attribute represents a way to describe or comment the quality of the current service, rated by users. Using ENUM type to present 10 kinds of degrees. ‘1’ is the lowest, which means a worst service, while ‘10’ is the highest, which means a best service. However, users usually are not interested in rating or forget to rate. In this case, considering the benefits of providers from rating, ‘10’ is placed as a default value. |
| FK | RequestID | INT | YES | NO | NO | A foreign\_key, coming from the entity ‘Request’ |
| FK | Provider\_UserID | INT | YES | NO | NO | A foreign\_key, coming from the entity ‘Provider’ |

**Entity ‘RideShareCard’**

**Description:**

This entity has records for storing the specific information of each membership card, which is called ‘RideCard’. Consumers can hold several cards at the same time, so the relationship between ‘Consumer’ and ‘RideShareCard’ is one-to-many, and can only use at least one of cards to make a payment for each unpaid contract. Also, each card has a one-to-many relationship related to the entity ‘Topup’ and the entity ‘Payment’ respectively.

**Attributes:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Attribute | Data Type | NN | UQ | UN | Description |
| PK | RideShareCardID | INT | YES | YES | YES | This is the primary key of ‘RideShareCard’, and unique in the database. Like the primary key of ‘Request’, this value should also be encrypted for security. Do not recommend to use an automatically increasing key. |
|  | DateOfIssue | DATETIME | YES | NO | NO | Stores the date when the card is firstly dealt with. This value can not be null. |
|  | DateOfCancellation | DATETIME | YES | NO | NO | Holds the date when the card is finally canceled, means the expired date. |
|  | CurrentBalance | INT | YES | NO | NO | This attribute represents the account balance, which means how many funds are available in the current membership card. According to the above assumptions, this value can be negative. When the available funds can not afford the current unpaid contract, it would be firstly deducted, and then need to top-up the account to pay the debts. |
| FK | Consumer\_UserID | INT | YES | NO | NO | A foreign\_key, coming from the entity ‘Consumer’ |

**Entity ‘Topup’**

**Description:**

This entity has only one relationship to the entity ‘RideShareCard’, which records the history top-up list of each card. Especially, according to the case, the RideCard can only be transferred via electronic funds transfer, so the bank information need to be stored in this entity.

**Attributes:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Attribute | Data Type | NN | UQ | UN | Description |
| PK | TopupID | INT | YES | YES | NO | This is the primary key of ‘Topup’, generated by the DBMS, which is marked with an automatically increasing number. |
|  | AddedAmount | INT | YES | NO | YES | Holds the added amount of each top-up. The unit of this value is Australian Dollar. |
|  | TopupTime | TIMESTAMP | YES | NO | NO | Records the time of each top-up, and can not be null. |
|  | BSB | VARCHAR(50) | YES | NO | NO | This attribute represents the number of the Bank-State-Branch for recording where the added amount is coming from. |
|  | BankAccount | VARCHAR(50) | YES | NO | NO | Records the number of each bank account to identify which account the amount is transferred. |
| FK | RideShareCardID | INT | YES | NO | NO | A foreign\_key, coming from the entity ‘RideShareCard’ |

**Entity ‘Payment’**

**Description:**

This entity is designed for representing each payment. Each RideCard has many payments, while one paid contract is only related to one payment in the database. Regarding to the above assumptions, each contract can only be paid by using one card. Therefore, it will generate only one payment record for each contract.

**Attributes:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Attribute | Data Type | NN | UQ | UN | Description |
| PK | PaymentID | INT | YES | YES | NO | This is the primary key of ‘Payment’, generated by the DBMS, which is marked with an automatically increasing number. |
|  | AmountPaid | INT | YES | NO | YES | Records the paid amount. Also, this attribute is absolute positive, so it is unsigned integer type. If the payment is a refund, this value would be negative, but the case does not mention any information related to refund. Therefore, a positive amount in this case can be convinced. |
|  | PaymentTime | TIMESTAMP | YES | NO | NO | Holds the time of each payment. |
| FK | RideShareCardID | INT | YES | NO | NO | A foreign\_key, coming from the entity ‘RideShareCard’ |
| FK | OrderID | INT | YES | NO | NO | A foreign\_key, coming from the entity ‘PaidRide’ |

**Entity ‘Post’**

**Description:**

This entity represents a public post to all users in the Rideshare system. Only employees can send a post to the system blog, so there is a one-to-many relationship related to the entity ‘Employee’. Also, each post has many comments from all users.

**Attributes:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Attribute | Data Type | NN | UQ | UN | Description |
| PK | PostID | INT | YES | YES | NO | This is the primary key of ‘Payment’, generated by the DBMS, which is marked with an automatically increasing number. |
|  | Content | VARCHAR(1000) | YES | NO | NO | This attribute records the content of the post. Only sending text feature is offered in the blog system, and employee can not send any emotions or images. Also, according to the assumptions, 1000 size is sufficient for all kinds of text posts. |
|  | LastTime | TIMESTAMP | YES | NO | NO | Records the latest update time for each post. |
| FK | Employee\_UserID | INT | YES | NO | NO | A foreign\_key, coming from the entity ‘Employee’ |

**Entity ‘Comment’**

**Description:**

This is an Associate Entity between ‘Post’ and ‘User’, allowing for multiple comments per post. Also each user has many comments in the blog system.

**Attributes:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Attribute | Data Type | NN | UQ | UN | Description |
| PFK | UserID | INT | YES | NO | NO | A foreign\_key, coming from the entity ‘User’ |
| PFK | PostID | INT | YES | NO | NO | A foreign\_key, coming from the entity ‘Post’ |
|  | Content | VARCHAR(1000) | YES | NO | NO | Records the content of the post. Only sending text feature is offered. According to the assumptions, 1000 size is also sufficient for all kinds of text posts. |
|  | LastTime | TIMESTAMP | YES | NO | NO | Records the latest update time for each comment. |

**Entity ‘ClickLog’**

**Description:**

**Attributes:**

**Entity ‘LoginLog’**

**Description:**

**Attributes:**