# Beat

Cameron Keene
Dillan Maraj
Thomas Pena
Charles Richardson

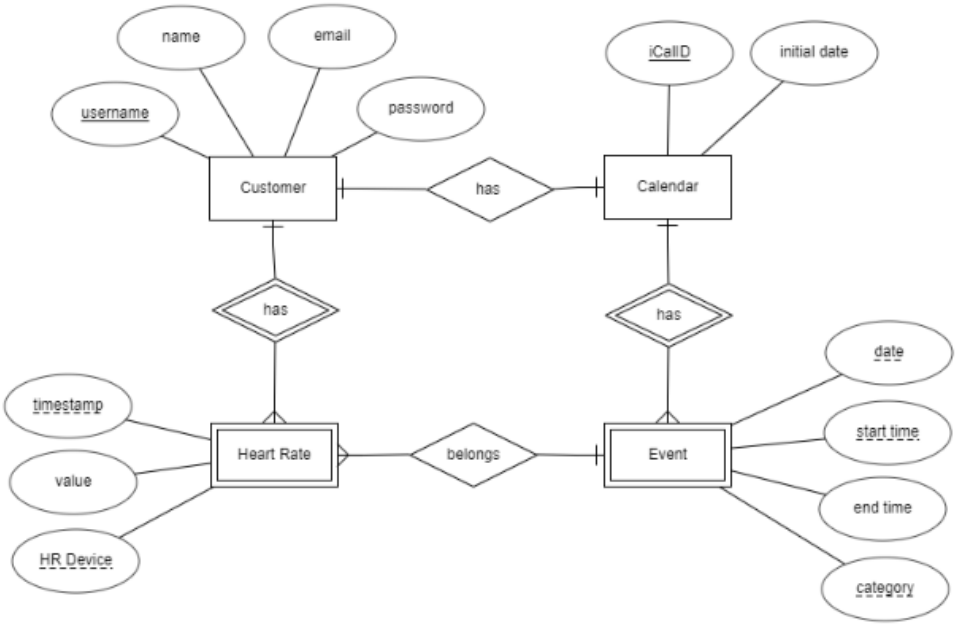
CIS4301 - Dr. Markus Schneider

University of Florida

**Table of Contents**



# Transformation of the ER diagram into a collection of relation schemas




**Entity-set Schema:**
Customer(username:string, name:string, email:string, password:string)
Calendar(iCalID:string, initial date:string)
Heart Rate(timestamp:string, HR device:string, value: integer)
Event(date:string, start time:string, end time:string, category:string)
**Relationship-set Schema:**
Has(username:string, iCalID:string)
has(username:string)
has(iCalID:string)
belongs(iCalID: string, username: string, category: string, value: integer)

## Transformation Process:

Customer:

All of the attributes defined in the Customer entity are defined as strings. By working with string we ensure that customers are able to use numbers and special characters when entering their usernames, passwords, emails, and names. Furthermore, by using strings the process of user authentication and security of the application can be better managed since it allows more combinations and we can later use this information for processes such as calendar API authorization and synchronization.

Calendar:

All calendar data will be retrieved using Google Calendar. Given this constraint, all of the attributes for this entity were defined considering the documentation available on Google Calendar API. The following figure defines the data type used for a Google calendar id and start.date.

| | | | |
|---|---|---|---|
| id | string | Identifier of the calendar. To retrieve IDs call the calendarList.list() method. | |
| start.date | date | The date, in the format "yyyy-mm-dd", if this is an all-day event. | writable |

Figure 1.1: Google Calendar API - ID & Start Date

https://developers.google.com/calendar/api/v3/reference/calendars#resource-representations

Considering the need to use simple data types, the date and ID will be stored as a string. Using strings allows for simple and seamless manipulation and comparison.

Heart Rate:

The data that we are using for the heart rate data is sourced from the Apple Health app. Through the settings, users are able to export all of their data from the app. From the files exported, there exists a file called export_data.xml. This file contains all the records of the user such as heart rate, calories, running time, etc. From this file we will only be analyzing the heart rate data. Figures 1.2, 1.3, and 1.4 show example records from the XML file. From this record we derived the three attributes for the heart rate entity. The two weak key attributes timestamp and HR Device can be seen in figure 1.2 and 1.3. The HRDevice attribute was derived from the sourceName values in the record entry. The timestamp attribute is derived from the record creationDate values. The value attribute is directly related to the value of the record value, which is the measure of a user's heart rate.

```
<Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Cam's Apple Watch" sourceVersion="6.1.1" device="&lt;&lt;HKDevice: 0x28390a490&gt;,
 <MetadataEntry key="HKMetadataKeyHeartRateMotionContext" value="1"/>
</Record>
```

Figure 1.2: Apple Health Record Part 1

```
name:Apple Watch, manufacturer:Apple Inc., model:Watch, hardware:Watch5,2, software:6.1.1&gt;" unit="count/min" creationDate="2020-06-22 05:08:39 -0500"
```

Figure 1.3: Apple Health Record Part 2

```
startDate="2020-06-22 05:03:38 -0500" endDate="2020-06-22 05:03:38 -0500" value="46">
```

Figure 1.4: Apple Health Record Part 3

Event**:**

The event entity is a weak entity that inherits iCalID (an identifying attribute) from the calendar entity – via the *has* identifying relationship. The event attributes are derived from the common values of the Google Calendar API. The weak key attribute *date* is derived from the start nested object which contains a date value, this can be seen in figure 1.5. The second weak key attribute *start time* is also derived from the same start nested object, which contains *dateTime*. This presents the time as a combined date-time value, this can also be seen in figure 1.5. The final weak key attribute for the event weak entity is *category*. This takes a little more work to develop. It is derived from the summary value of the calendar API, which contains the title of the event. Each event summary will begin with one of the six categories of event types we previously specified in Deliverable 1 and 2. The summary is a string, so we will have to parse the string to get the leading text. This will then be the value used in the *category* weak key attribute, this can be seen in figure 1.6. The final attribute *end time* is derived from the *end* nested object. The *end* nested object contains a value *dateTime*, which contains the time as a combined date-time value, this can be seen in figure 1.5. This will be used for the *end time* attribute of the event attribute.

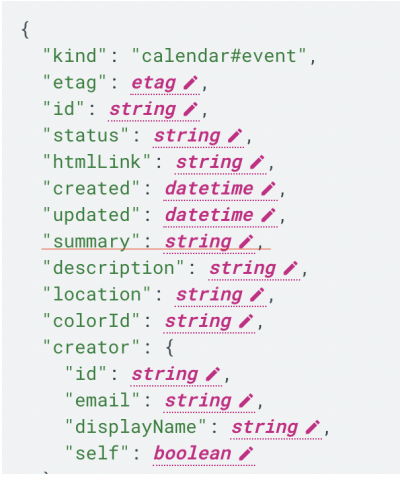



Figure 1.5: Google Calendar API


Figure 1.6: Google Calendar API - Start & End Nested Objects

Our ER Diagram contains multiple relationships. The *has* relationship combines the *Customer* and *Calendar* entities. To properly identify this relationship two attributes are used, *username* from the Customer entity and *iCalID* from the *Calendar* entity. The second *has* relationship is an identifying relationship between the *Customer* entity and the *Heart Rate* weak entity. The attribute of this relationship is the *username* of the *Customer* entity. The third *has* identifying relationship connects the *Calendar* entity and the Event weak entity. The attribute of this relationship is the *iCalID*, inheriting from the *Calendar* entity. The last relationship *belongs* connects both weak entities, *Heart Rate* and *Event*. The key attributes for this are *iCalID* and *username*. The *iCalID* is inherited from *Calendar* and *username* is inherited from *Customer*.

# Transformation of the collection of relation schemas into a collection of SQL table schemas

Mapping the collection of relational schema created above to the SQL table schemas was straightforward. The table names were one to one, as with the majority of the attribute names. There were some attribute names that conflicted with SQL keywords which required some underscores to be added to words or some concatenation of words.

Deciding the data types was done for us, the deliverable requirements state that SQL data types could not be used, so we went with the classic "string" type – varchar2 – all the way through, minus the *HRvalue* entity, which is restricted to **int** given its numerical measurement. Deciding which attributes would be constrained to **not null** was also simple. Since the majority of the data was of high importance for our queries, there was not much that could be left null. Further, primary keys cannot be null, which further limited which values could be null.

The screen capture of all the table creation commands is below, followed by the table tree with its attributes shown:

```
create table beat_customer (
    username   varchar2(100) not null,
    fullname   varchar2(100),
    email      varchar2(100),
    pass       varchar2(100) not null,
    primary key (username));

create table beat_calendar (
    iCalID     varchar2(100) not null,
    start_date varchar2(100),
    primary key (iCalID));

create table beat_heartrate (
    username   varchar2(100) not null,
    time_stamp varchar2(100) not null,
    deviceID   varchar2(100) not null,
    HRvalue    int           not null,
    primary key (username, time_stamp),
    foreign key (username) references beat_customer(username));

create table beat_event (
    event_date varchar2(100) not null,
    start_time varchar2(100) not null,
    end_time   varchar2(100) not null,
    cat        varchar2(100) not null,
    primary key (iCalID, event_date, start_time, cat),
    foreign key (iCalID) references beat_calendar(iCalID));
```

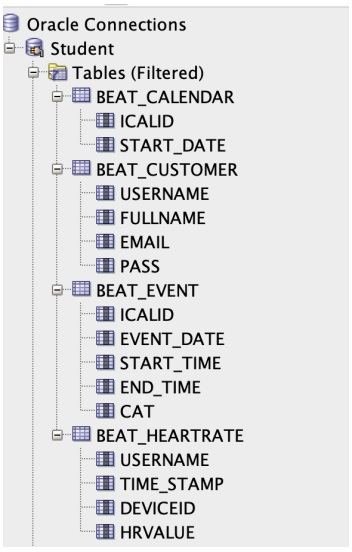



Figure 2.1 and 2.2: SQL table schema creation worksheet and tables tree including attributes