Fiona Nicdao, Josh Honig, Drew Patterson, Hannah Serio

FRIDATING

Friend Dating Database

FRIDATING

Allows like-minded individuals to connect and form lasting friendships!

Each user can create an account

Account contains a biography, location, interests, etc.

Accounts can manage and/or RSVP to events

Social events facilitate friendships moving from messaging to real-life

An account can match with other accounts

Two accounts are considered a match when both users accept the status

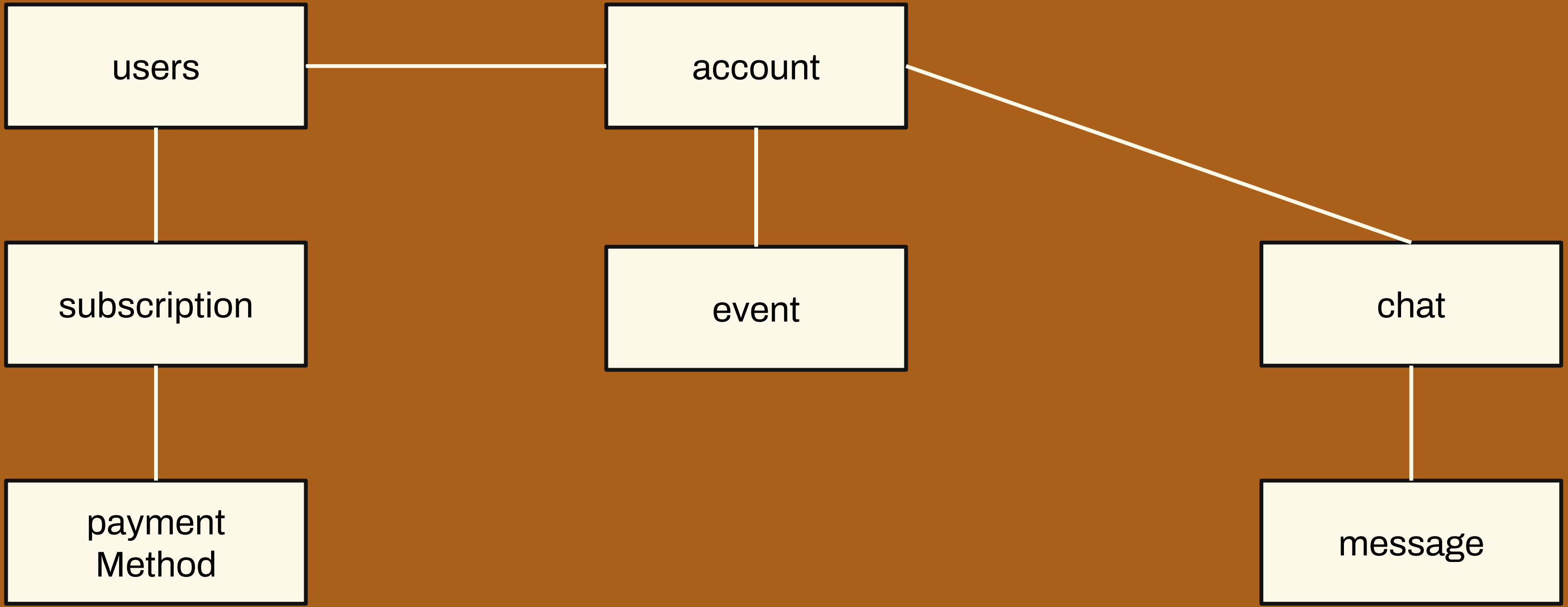
Accounts can message their matches

A chat is created when two accounts match to allow messaging between users

Each user manages their subscription

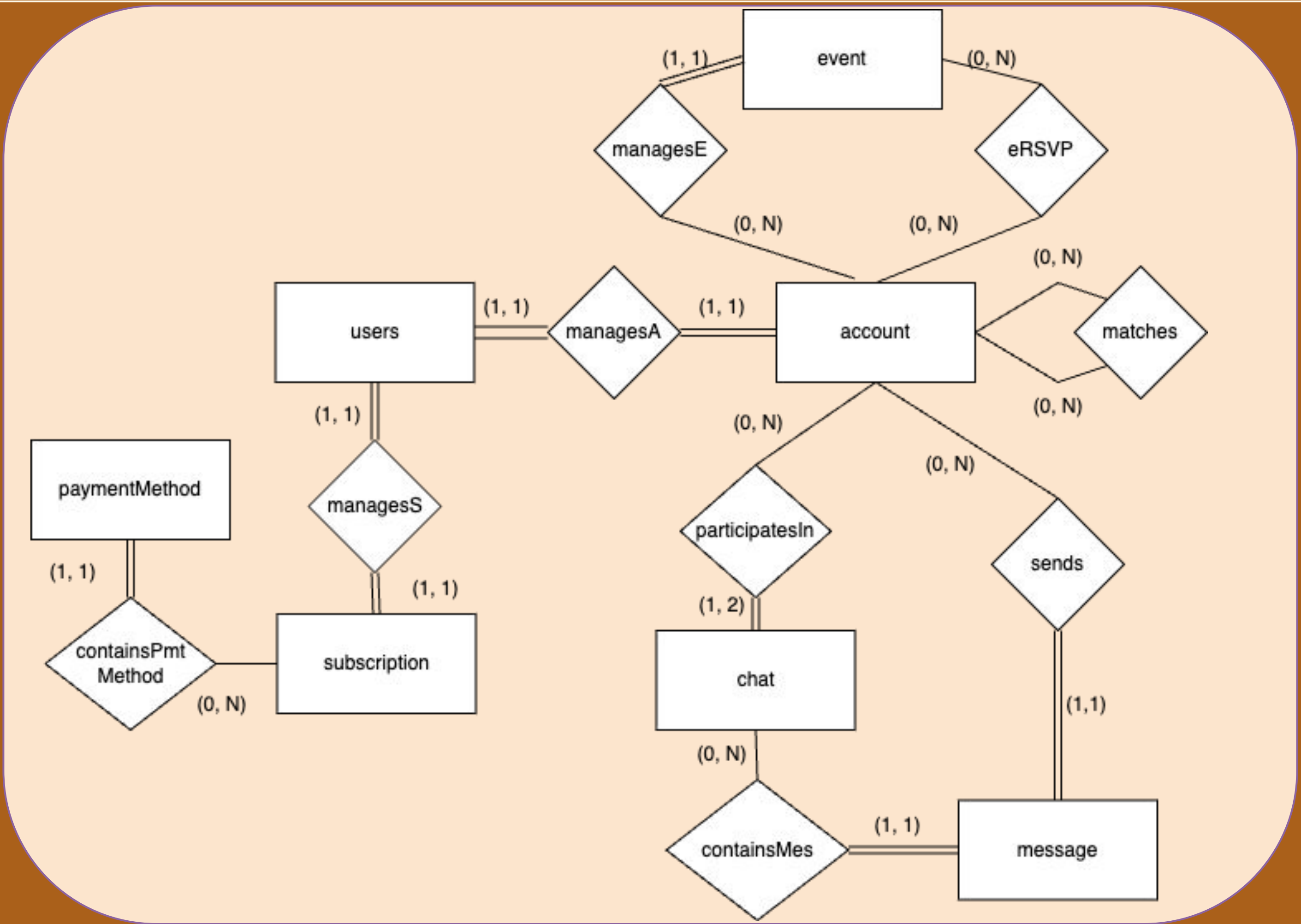
Users manage their subscription type and payment methods

ER Model



---LINK---

ER Model



---LINK---

Relational Schema

- event(<u>evtID</u>, evtType, evtDateStart, evtDateEnd, evtName, evtDetails, evtLocGPS, evtLocRange, evtManager)
  foreign key (evtManager) references account(actID)

- eRSVP(<u>actID, evtID</u>, response, rsvDate)
  foreign key (actID) references account(actID)
  foreign key (evtID) references event(evtID)

- subscription(<u>subID</u>, subTier, subPrice, outstandingBalance, nextDueDate, annualBilling, userID)
  foreign key (userID) references users(usrID)

- paymentMethod(<u>pmtID</u>, subID, pmtType, num, cvv, expiration, pmtStreetAddr, pmtCity, pmtState, pmtZipcode, androidPay, applePay, payPal)
  foreign key (subscription) references subscription(subID)

- matches(<u>actID1, actID2</u>, mchDate, compatabilityScore, act1status, act2status)
  foreign key (actID1) references account(actID)
  foreign key (actID2) references account(actID)

Friend Dating Database

ORM

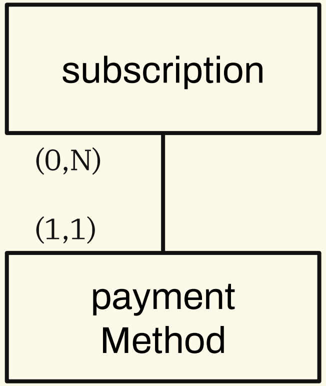
Object-Relational Mapping

subscription

- Subscription tier
- Tier price*
- Billing mode*
- Due date*
- User ID
- Payment Methods

paymentMethod

- Type
- Card Information
  - Number
  - CVV
  - Expiration
  - Billing Address
- 3rd Party Information
- Subscription

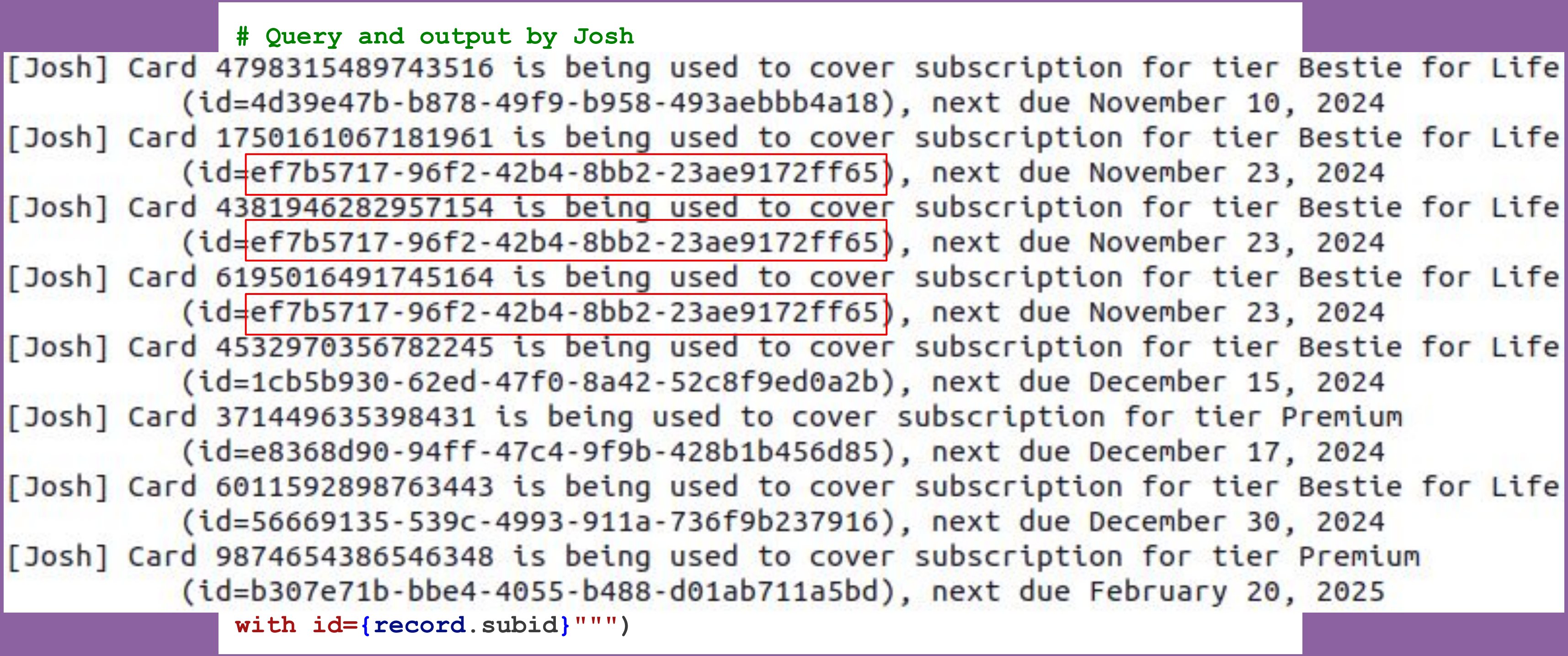


**if paid*

```
# Class created by Josh
class paymentmethod(Base):
    __tablename__ = "paymentmethod"
    pmtid: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True), default=uuid.uuid4,
primary_key=True)
    subid: Mapped[uuid.UUID] = mapped_column(ForeignKey("subscription.subid"))
    pmttype = mapped_column(Enum(payment_type))
    num: Mapped[Numeric] = mapped_column(Numeric, nullable=True)
    cvv: Mapped[Numeric] = mapped_column(Numeric, nullable=True)
    expiration: Mapped[str] = mapped_column(String(16), nullable=True)
    pmtstreetaddr: Mapped[str] = mapped_column(String(128), nullable=True)
    pmtcity: Mapped[str] = mapped_column(String(64), nullable=True)
    pmtstate: Mapped[str] = mapped_column(String(64), nullable=True)
    pmtzipcode: Mapped[str] = mapped_column(String(64), nullable=True)
    androidpay: Mapped[str] = mapped_column(String(128), nullable=True)
    applepay: Mapped[str] = mapped_column(String(128), nullable=True)
    paypal: Mapped[str] = mapped_column(String(128), nullable=True)
    subscription: Mapped["Subscription"] = relationship(back_populates="paymentmethods")
    def __repr__(self) -> str:
        return f"""paymentMethod(
            pmtid={self.pmtid!r}, subid={self.subid!r}, pmtType={self.pmttype!r},
            num={self.num!r}, cvv={self.cvv!r}, expiration={self.expiration!r},
            pmtstreetaddr={self.pmtstreetaddr!r}, pmtcity={self.pmtcity!r},
            pmtstate={self.pmtstate!r}, pmtzipcode={self.pmtzipcode!r},
            androidpay={self.androidpay!r}, applepay={self.applepay!r},
paypal={self.paypal!r})"""
```

```
class Subscription(Base):
    __tablename__ = "subscription"
    subid: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True), default=uuid.uuid4,
primary_key=True)
    subtier: Mapped[str] = mapped_column(String(64))
    subprice: Mapped[Numeric] = mapped_column(Numeric, nullable=True)
    outstandingbalance: Mapped[Numeric] = mapped_column(Numeric, nullable=True)
    nextduedate: Mapped[datetime] = mapped_column(DateTime(timezone=False), nullable=True)
    annualbilling: Mapped[Optional[bool]]
    userid: Mapped[uuid.UUID] = mapped_column(ForeignKey("user.userID"))
    paymentmethods: Mapped[List["paymentmethod"]] = relationship(
        back_populates="subscription", cascade="all, delete-orphan"
    )
    def __repr__(self) -> str:
        return f"""Subscription(
            id={self.subid!r}, subtier={self.subtier!r}, subprice={self.subprice!r},
            nextduedate={self.nextduedate!r}, annualbilling={self.annualbilling!r},
            userid={self.userid!r})"""
```

Credit Cards for Each Subscription



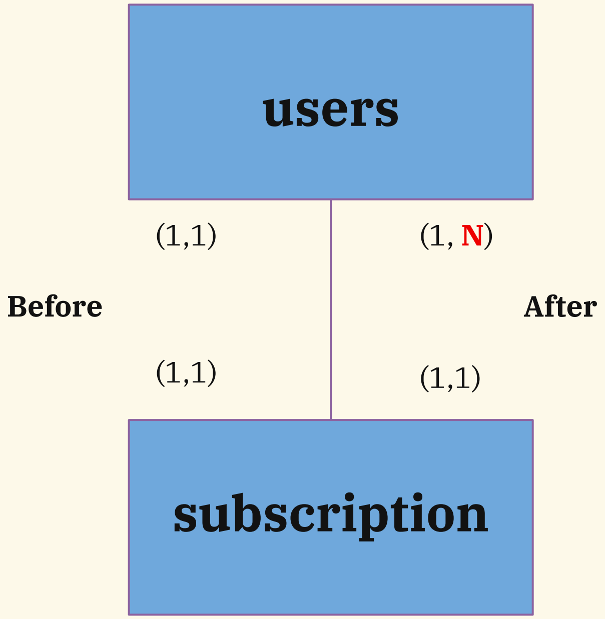
```
# Query and output by Josh
[Josh] Card 4798315489743516 is being used to cover subscription for tier Bestie for Life
          (id=4d39e47b-b878-49f9-b958-493aebbb4a18), next due November 10, 2024
[Josh] Card 1750161067181961 is being used to cover subscription for tier Bestie for Life
          (id=ef7b5717-96f2-42b4-8bb2-23ae9172ff65), next due November 23, 2024
[Josh] Card 4381946282957154 is being used to cover subscription for tier Bestie for Life
          (id=ef7b5717-96f2-42b4-8bb2-23ae9172ff65), next due November 23, 2024
[Josh] Card 6195016491745164 is being used to cover subscription for tier Bestie for Life
          (id=ef7b5717-96f2-42b4-8bb2-23ae9172ff65), next due November 23, 2024
[Josh] Card 4532970356782245 is being used to cover subscription for tier Bestie for Life
          (id=1cb5b930-62ed-47f0-8a42-52c8f9ed0a2b), next due December 15, 2024
[Josh] Card 371449635398431 is being used to cover subscription for tier Premium
          (id=e8368d90-94ff-47c4-9f9b-428b1b456d85), next due December 17, 2024
[Josh] Card 6011592898763443 is being used to cover subscription for tier Bestie for Life
          (id=56669135-539c-4993-911a-736f9b237916), next due December 30, 2024
[Josh] Card 9874654386546348 is being used to cover subscription for tier Premium
          (id=b307e71b-bbe4-4055-b488-d01ab711a5bd), next due February 20, 2025
with id={record.subid}""")
```

users

- User ID
- Username
- Email
- Passwordhash
- Passwordsalt
- MFAtoken

subscription

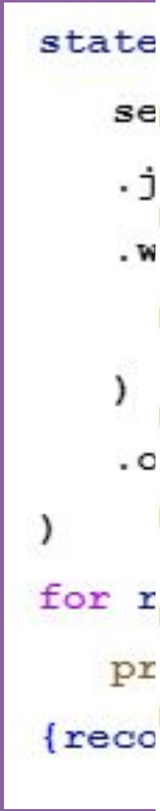
- Subscription tier
- Tier price*
- Billing mode*
- Due date*
- User ID
- Payment Methods



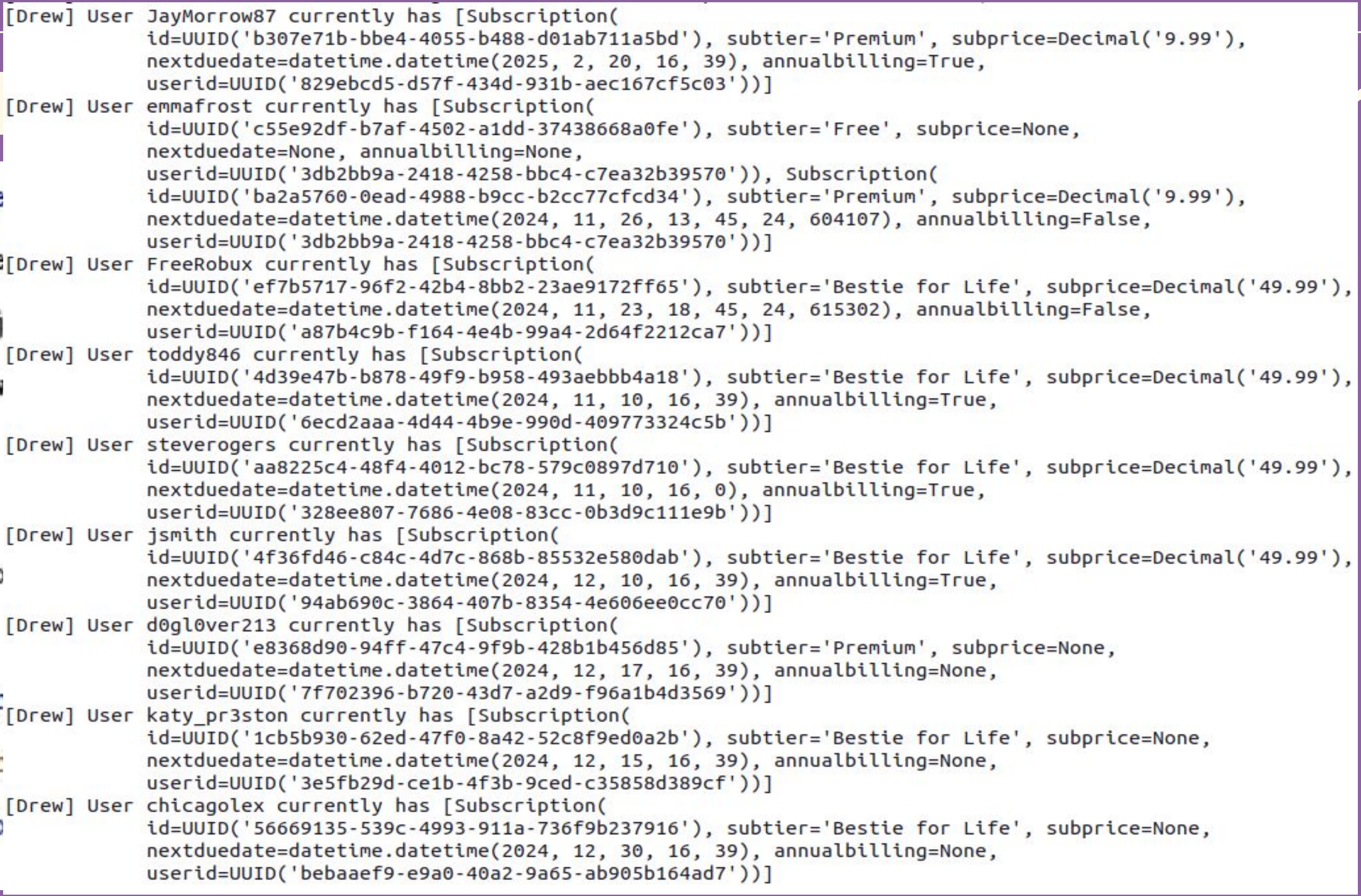
**if paid*

```
# Class created by Drew; Defining User class/table
class Users(Base):
    __tablename__ = "users"

    usrid: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
default=uuid.uuid4, primary_key=True)
    username: Mapped[str] = mapped_column(String(32))
    email: Mapped[str] = mapped_column(String(96))
    passwordhash: Mapped[str] = mapped_column(String(256))
    passwordsalt: Mapped[str] = mapped_column(String(256))
    mfatoken: Mapped[str] = mapped_column(String(64))
    subscription: Mapped[List["Subscription"]] =
relationship("Subscription", back_populates="user")

    def __repr__(self) -> str:
        return f"User(id={self.userID!r},
username={self.username!r}, email={self.email!r})"
```

Sul rs



```
state
    se
    .j
    .w
    )
    .c
)
for r
    pr
{reco
```



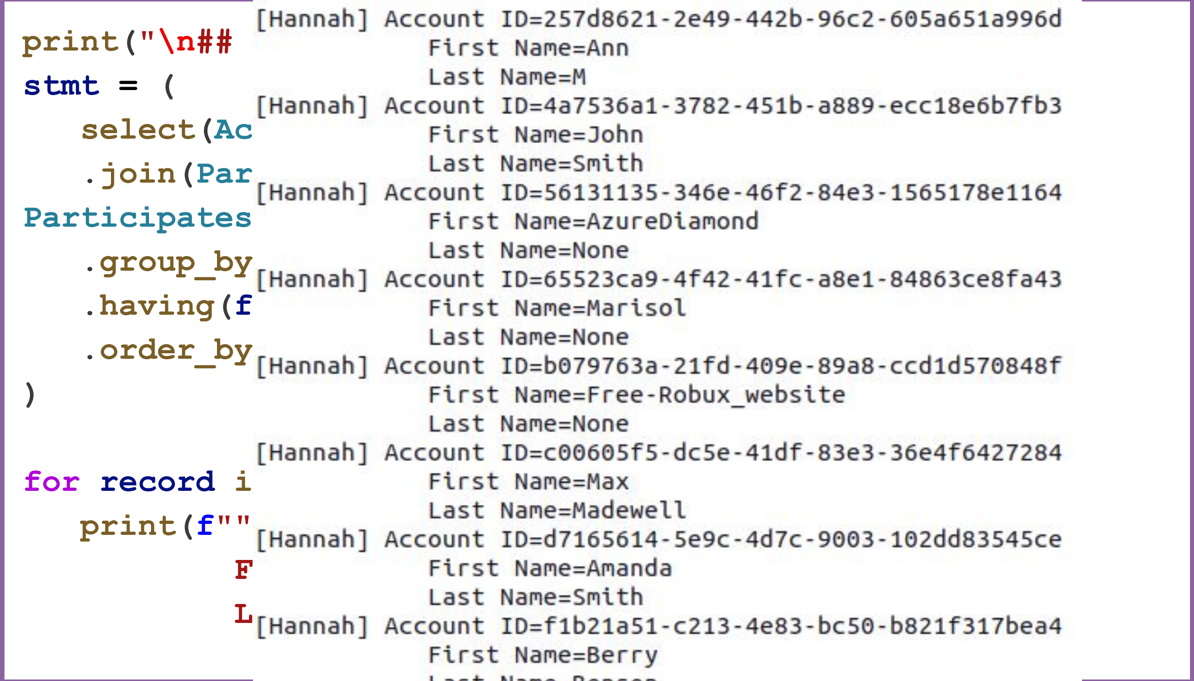
```
[Drew] User JayMorrow87 currently has [Subscription(
            id=UUID('b307e71b-bbe4-4055-b488-d01ab711a5bd'), subtier='Premium', subprice=Decimal('9.99'),
            nextduedate=datetime.datetime(2025, 2, 20, 16, 39), annualbilling=True,
            userid=UUID('829ebcd5-d57f-434d-931b-aec167cf5c03'))]
[Drew] User emmafrost currently has [Subscription(
            id=UUID('c55e92df-b7af-4502-a1dd-37438668a0fe'), subtier='Free', subprice=None,
            nextduedate=None, annualbilling=None,
            userid=UUID('3db2bb9a-2418-4258-bbc4-c7ea32b39570')), Subscription(
            id=UUID('ba2a5760-0ead-4988-b9cc-b2cc77cfcd34'), subtier='Premium', subprice=Decimal('9.99'),
            nextduedate=datetime.datetime(2024, 11, 26, 13, 45, 24, 604107), annualbilling=False,
            userid=UUID('3db2bb9a-2418-4258-bbc4-c7ea32b39570'))]
[Drew] User FreeRobux currently has [Subscription(
            id=UUID('ef7b5717-96f2-42b4-8bb2-23ae9172ff65'), subtier='Bestie for Life', subprice=Decimal('49.99'),
            nextduedate=datetime.datetime(2024, 11, 23, 18, 45, 24, 615302), annualbilling=False,
            userid=UUID('a87b4c9b-f164-4e4b-99a4-2d64f2212ca7'))]
[Drew] User toddy846 currently has [Subscription(
            id=UUID('4d39e47b-b878-49f9-b958-493aebbb4a18'), subtier='Bestie for Life', subprice=Decimal('49.99'),
            nextduedate=datetime.datetime(2024, 11, 10, 16, 39), annualbilling=True,
            userid=UUID('6ecd2aaa-4d44-4b9e-990d-409773324c5b'))]
[Drew] User steverogers currently has [Subscription(
            id=UUID('aa8225c4-48f4-4012-bc78-579c0897d710'), subtier='Bestie for Life', subprice=Decimal('49.99'),
            nextduedate=datetime.datetime(2024, 11, 10, 16, 0), annualbilling=True,
            userid=UUID('328ee807-7686-4e08-83cc-0b3d9c111e9b'))]
[Drew] User jsmith currently has [Subscription(
            id=UUID('4f36fd46-c84c-4d7c-868b-85532e580dab'), subtier='Bestie for Life', subprice=Decimal('49.99'),
            nextduedate=datetime.datetime(2024, 12, 10, 16, 39), annualbilling=True,
            userid=UUID('94ab690c-3864-407b-8354-4e606ee0cc70'))]
[Drew] User d0gl0ver213 currently has [Subscription(
            id=UUID('e8368d90-94ff-47c4-9f9b-428b1b456d85'), subtier='Premium', subprice=None,
            nextduedate=datetime.datetime(2024, 12, 17, 16, 39), annualbilling=None,
            userid=UUID('7f702396-b720-43d7-a2d9-f96a1b4d3569'))]
[Drew] User katy_pr3ston currently has [Subscription(
            id=UUID('1cb5b930-62ed-47f0-8a42-52c8f9ed0a2b'), subtier='Bestie for Life', subprice=None,
            nextduedate=datetime.datetime(2024, 12, 15, 16, 39), annualbilling=None,
            userid=UUID('3e5fb29d-ce1b-4f3b-9ced-c35858d389cf'))]
[Drew] User chicagolex currently has [Subscription(
            id=UUID('56669135-539c-4993-911a-736f9b237916'), subtier='Bestie for Life', subprice=None,
            nextduedate=datetime.datetime(2024, 12, 30, 16, 39), annualbilling=None,
            userid=UUID('bebaaef9-e9a0-40a2-9a65-ab905b164ad7'))]
```

```
# Class created by Fiona
class Message(Base):
  __tablename__ = "message"
  mesid: Mapped[int] = mapped_column(primary_key=True,
autoincrement=True)
  mestext: Mapped[str] = mapped_column(String(200))
  mesdate : Mapped[datetime] =
mapped_column(DateTime(timezone=True))
  chatid : Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
ForeignKey("chat.chatid"))
  senderid: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
ForeignKey("account.actid"))
  chat : Mapped["Chat"] = relationship(back_populates="messages")

   def __repr__(self) -> str :
      return f"Message(id={self.id!r}), mesText={self.mestext!r},
mesDate={self.mesdate!r}"
```

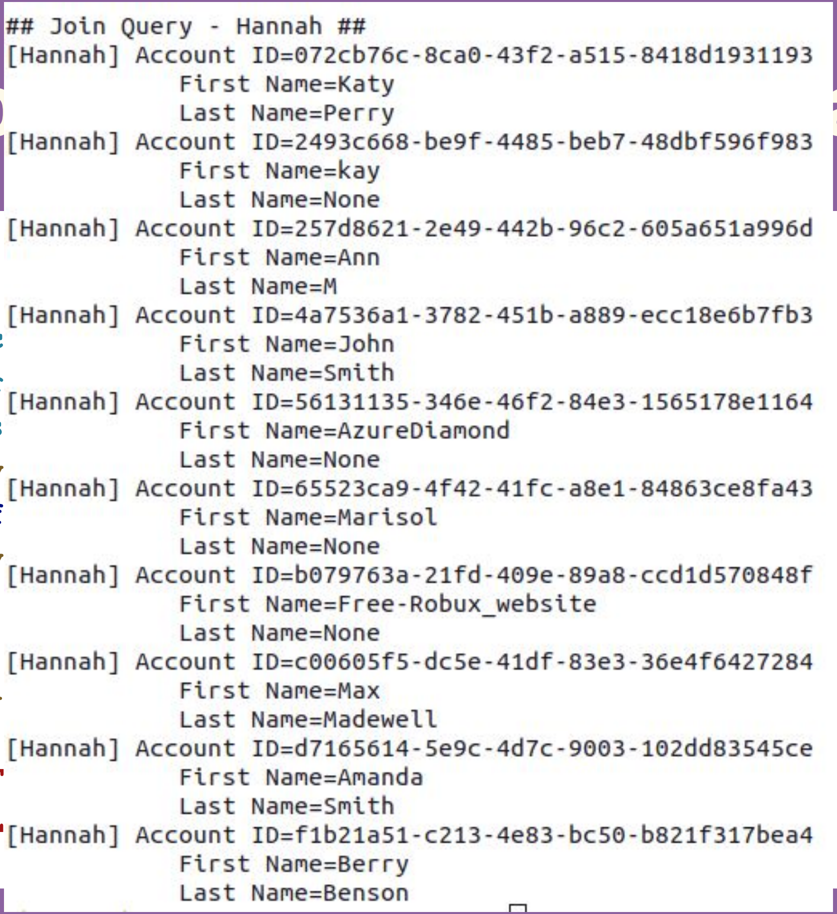
```
## Join Query -  Fiona  ##
message id=03a9c586-a4d5-4089-aa34-1198ea8fa3bc chat_id=294441f3-87d3-4989-96bd-9dfb7c9d5325
  message date=2024-11-18 15:04:25.461850-06:00
  message Text=Hi Vision! Do you play Magic the Gathering ?


message id=081195c2-9058-4e10-8316-ed2a5cd69856 chat_id=294441f3-87d3-4989-96bd-9dfb7c9d5325
  message date=2024-11-19 15:04:25.463839-06:00
  message Text=Thats awesome! Do you want to get together and play MTG commandard ?


message id=0c6a58a0-5fc0-44a6-8c39-54c14a8d553f chat_id=294441f3-87d3-4989-96bd-9dfb7c9d5325
  message date=2024-11-19 15:04:25.463824-06:00
  message Text=What is up Wanda! Ya! I just got the new duskmourn deck.


message id=549e76df-c525-4153-a450-51d81142bb5d chat_id=294441f3-87d3-4989-96bd-9dfb7c9d5325
  message date=2024-11-19 15:04:25.463858-06:00
  message Text=Great! Is it okay for me to invite two of my friends for the game?


message id=8fdcd137-5917-4507-9c46-d139503e76ea chat_id=294441f3-87d3-4989-96bd-9dfb7c9d5325
  message date=2024-11-19 15:04:25.463850-06:00
  message Text=Yes! That sounds like fun!


message id=f4bde6c9-4999-45b7-b9b2-577574421334 chat_id=294441f3-87d3-4989-96bd-9dfb7c9d5325
  message date=2024-11-19 15:04:25.463867-06:00
  message Text=Ya! Commander is fun with a group of 4 people
```

```
# Classes created by Hannah
class ParticipatesIn(Base):
    __tablename__ = "participatesin"

    actid: Mapped[uuid.UUID] = mapped_column(UUID( as_uuid=True),
ForeignKey("account.actid", ondelete="CASCADE", onupdate="CASCADE"),
primary_key=True)
    chatid: Mapped[uuid.UUID] = mapped_column(UUID( as_uuid=True),
ForeignKey("chat.chatid", ondelete="CASCADE", onupdate="CASCADE"),
primary_key=True)
    startdate: Mapped[datetime] = mapped_column(DateTime( timezone=False),
nullable=True)

    # Relationships
    account: Mapped["Account"] = relationship("Account",
back_populates="participates_in")
    chat: Mapped["Chat"] = relationship("Chat", back_populates="participants")

    def __repr__(self) -> str: return f"ParticipatesIn(actID={self.actid!r},
chatID={self.chatid!r})"
```

Accounts p   at



```
print("\n##
stmt = (
    select(Ac
    .join(Par
Participates
    .group_by
    .having(f
    .order_by
)

for record i
    print(f""
             F
             L
```



```
## Join Query - Hannah ##
[Hannah] Account ID=072cb76c-8ca0-43f2-a515-8418d1931193
            First Name=Katy
            Last Name=Perry
[Hannah] Account ID=2493c668-be9f-4485-beb7-48dbf596f983
            First Name=kay
            Last Name=None
[Hannah] Account ID=257d8621-2e49-442b-96c2-605a651a996d
            First Name=Ann
            Last Name=M
[Hannah] Account ID=4a7536a1-3782-451b-a889-ecc18e6b7fb3
            First Name=John
            Last Name=Smith
[Hannah] Account ID=56131135-346e-46f2-84e3-1565178e1164
            First Name=AzureDiamond
            Last Name=None
[Hannah] Account ID=65523ca9-4f42-41fc-a8e1-84863ce8fa43
            First Name=Marisol
            Last Name=None
[Hannah] Account ID=b079763a-21fd-409e-89a8-ccd1d570848f
            First Name=Free-Robux_website
            Last Name=None
[Hannah] Account ID=c00605f5-dc5e-41df-83e3-36e4f6427284
            First Name=Max
            Last Name=Madewell
[Hannah] Account ID=d7165614-5e9c-4d7c-9003-102dd83545ce
            First Name=Amanda
            Last Name=Smith
[Hannah] Account ID=f1b21a51-c213-4e83-bc50-b821f317bea4
            First Name=Berry
            Last Name=Benson
```

THANK YOU

Brought to you by The Matchmakers