```
"""
Harmony Database Schema
Stores family data, conversations, and memory indices
"""

from sqlalchemy import create_engine, Column, Integer, String, Text, DateTime, Float, Boolean,
ForeignKey, JSON
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker
from datetime import datetime
import uuid

Base = declarative_base()

class Family(Base):
    """Core family entity"""
    __tablename__ = 'families'

    id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
    family_name = Column(100), nullable=False)
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)

    # Memory storage paths
    memory_path = Column(String(255))  # Path to MemoRAG memory file
    retrieval_index_path = Column(String(255))  # Path to FAISS index

    # Family settings
    settings = Column(JSON, default={})

    # Relationships
    members = relationship("FamilyMember", back_populates="family")
    conversations = relationship("Conversation", back_populates="family")
    memories = relationship("FamilyMemory", back_populates="family")
    insights = relationship("FamilyInsight", back_populates="family")
    data_records = relationship("DataRecord", back_populates="family")


class FamilyMember(Base):
    """Individual family members"""
    __tablename__ = 'family_members'

    id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
    family_id = Column(String(36), ForeignKey('families.id'), nullable=False)

    # Member info
    name = Column(String(100), nullable=False)
    profile_name = Column(String(100))  # e.g., "Ella Faith Reed"
```

```
    role = Column(String(50))  # e.g., "parent", "child", "extended"

    # Profile data
    hobbies = Column(JSON, default=[])
    interests = Column(JSON, default=[])
    values = Column(JSON, default=[])

    # Dates
    joined_date = Column(DateTime, default=datetime.utcnow)
    birthdate = Column(DateTime, nullable=True)

    # Relationships
    family = relationship("Family", back_populates="members")
    messages = relationship("Message", back_populates="member")


class Conversation(Base):
    """Conversation threads"""
    __tablename__ = 'conversations'

    id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
    family_id = Column(String(36), ForeignKey('families.id'), nullable=False)

    title = Column(String(200))
    started_at = Column(DateTime, default=datetime.utcnow)
    last_message_at = Column(DateTime, default=datetime.utcnow)

    # Context for this conversation
    context_summary = Column(Text)

    # Relationships
    family = relationship("Family", back_populates="conversations")
    messages = relationship("Message", back_populates="conversation")


class Message(Base):
    """Individual messages in conversations"""
    __tablename__ = 'messages'

    id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
    conversation_id = Column(String(36), ForeignKey('conversations.id'), nullable=False)
    member_id = Column(String(36), ForeignKey('family_members.id'), nullable=True)  # Null if
from Harmony

    # Message content
    role = Column(String(20), nullable=False)  # "user" or "assistant"
    content = Column(Text, nullable=False)

    # Metadata
    created_at = Column(DateTime, default=datetime.utcnow)
```

```
    tokens_used = Column(Integer, default=0)

    # If this message used memory retrieval
    used_memory = Column(Boolean, default=False)
    retrieved_chunks = Column(JSON, default=[])

    # Relationships
    conversation = relationship("Conversation", back_populates="messages")
    member = relationship("FamilyMember", back_populates="messages")


class FamilyMemory(Base):
    """Stored family memories and important information"""
    __tablename__ = 'family_memories'

    id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
    family_id = Column(String(36), ForeignKey('families.id'), nullable=False)

    # Memory content
    memory_type = Column(String(50))  # "event", "tradition", "milestone", "profile", "general"
    title = Column(String(200))
    content = Column(Text, nullable=False)

    # Metadata
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)

    # Tags for categorization
    tags = Column(JSON, default=[])

    # Related members
    related_members = Column(JSON, default=[])

    # Relationships
    family = relationship("Family", back_populates="memories")


class FamilyInsight(Base):
    """AI-generated insights about the family"""
    __tablename__ = 'family_insights'

    id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
    family_id = Column(String(36), ForeignKey('families.id'), nullable=False)

    # Insight details
    insight_type = Column(String(50))  # "values", "strengths", "opportunities", "growth"
    title = Column(String(200))
    content = Column(Text, nullable=False)

    # Generation info
```

```
    generated_at = Column(DateTime, default=datetime.utcnow)
    prompt_used = Column(Text)

    # Relationships
    family = relationship("Family", back_populates="insights")


class DataRecord(Base):
    """
    Anonymized data records for the ethical data marketplace
    This is what gets sold - fully anonymized
    """
    __tablename__ = 'data_records'

    id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
    family_id = Column(String(36), ForeignKey('families.id'), nullable=False)

    # Anonymized data
    data_type = Column(String(50))  # "mood", "activity", "milestone", "interaction"
    anonymized_data = Column(JSON, nullable=False)

    # Metadata (no identifying info)
    family_size_range = Column(String(20))  # e.g., "2-4", "5-7"
    family_stage = Column(String(50))  # e.g., "young_children", "teenagers",
"multigenerational"
    timestamp_bucket = Column(String(20))  # e.g., "2024-Q1", not exact dates

    # Data marketplace
    is_marketable = Column(Boolean, default=True)
    value_cents = Column(Integer, default=0)  # Micro-payment value
    sold_at = Column(DateTime, nullable=True)

    created_at = Column(DateTime, default=datetime.utcnow)

    # Relationships
    family = relationship("Family", back_populates="data_records")


class FamilyFund(Base):
    """Family fund account - earnings from data"""
    __tablename__ = 'family_funds'

    id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
    family_id = Column(String(36), ForeignKey('families.id'), unique=True, nullable=False)

    # Balance
    balance_cents = Column(Integer, default=0)
    staked_balance_cents = Column(Integer, default=0)
    total_earned_cents = Column(Integer, default=0)
```

```
    # Staking info
    stake_apr = Column(Float, default=0.0)
    last_interest_calc = Column(DateTime, default=datetime.utcnow)

    # Tracking
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)

    # Relationships
    transactions = relationship("FundTransaction", back_populates="fund")


class FundTransaction(Base):
    """Transaction history for family fund"""
    __tablename__ = 'fund_transactions'

    id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
    fund_id = Column(String(36), ForeignKey('family_funds.id'), nullable=False)

    # Transaction details
    transaction_type = Column(String(50))  # "data_sale", "interest", "withdrawal", "stake",
"unstake"
    amount_cents = Column(Integer, nullable=False)
    description = Column(Text)

    # Balance after transaction
    balance_after_cents = Column(Integer)

    created_at = Column(DateTime, default=datetime.utcnow)

    # Relationships
    fund = relationship("FamilyFund", back_populates="transactions")


# Database initialization
class HarmonyDatabase:
    def __init__(self, db_url: str = "sqlite:///harmony_families.db"):
        self.engine = create_engine(db_url, echo=True)
        Base.metadata.create_all(self.engine)
        self.Session = sessionmaker(bind=self.engine)

    def create_family(self, family_name: str) -> Family:
        """Create a new family"""
        session = self.Session()
        try:
            family = Family(family_name=family_name)
            session.add(family)

            # Create associated fund account
            fund = FamilyFund(family_id=family.id)
```

```
        session.add(fund)

        session.commit()
        return family
    finally:
        session.close()

def add_family_member(self, family_id: str, name: str, **kwargs) -> FamilyMember:
    """Add a member to a family"""
    session = self.Session()
    try:
        member = FamilyMember(
            family_id=family_id,
            name=name,
            **kwargs
        )
        session.add(member)
        session.commit()
        return member
    finally:
        session.close()

def record_data_for_marketplace(
    self,
    family_id: str,
    data_type: str,
    data: dict,
    family_size_range: str,
    family_stage: str
) -> DataRecord:
    """Record anonymized data for marketplace"""
    session = self.Session()
    try:
        record = DataRecord(
            family_id=family_id,
            data_type=data_type,
            anonymized_data=data,
            family_size_range=family_size_range,
            family_stage=family_stage,
            timestamp_bucket=f"{datetime.now().year}-Q{(datetime.now().month-1)//3 + 1}"
        )
        session.add(record)
        session.commit()
        return record
    finally:
        session.close()

def add_fund_transaction(
    self,
    family_id: str,
```

```
        transaction_type: str,
        amount_cents: int,
        description: str = None
    ):
        """Add a transaction to family fund"""
        session = self.Session()
        try:
            # Get fund
            fund = session.query(FamilyFund).filter_by(family_id=family_id).first()
            if not fund:
                raise ValueError(f"No fund found for family {family_id}")

            # Update balance
            fund.balance_cents += amount_cents
            fund.total_earned_cents += max(0, amount_cents)

            # Create transaction record
            transaction = FundTransaction(
                fund_id=fund.id,
                transaction_type=transaction_type,
                amount_cents=amount_cents,
                description=description,
                balance_after_cents=fund.balance_cents
            )

            session.add(transaction)
            session.commit()
        finally:
            session.close()


# Usage example
if __name__ == "__main__":
    # Initialize database
    db = HarmonyDatabase()

    # Create a family
    adams_family = db.create_family("Adams Family")
    print(f"Created family: {adams_family.id}")

    # Add members
    jeramy = db.add_family_member(
        adams_family.id,
        "Jeramy",
        role="parent",
        values=["faith", "leadership", "stability"]
    )

    ella = db.add_family_member(
        adams_family.id,
```

```
    "Ella",
    profile_name="Ella Faith Reed",
    role="partner",
    hobbies=["caring for family", "bringing love"]
)

addy = db.add_family_member(
    adams_family.id,
    "Addy",
    role="child",
    hobbies=["pink and purple art projects"]
)

print("Family members added!")

# Record some anonymized data
db.record_data_for_marketplace(
    family_id=adams_family.id,
    data_type="family_activity",
    data={
        "activity_type": "creative_time",
        "frequency": "weekly",
        "participation": "all_members"
    },
    family_size_range="2-4",
    family_stage="young_children"
)

# Add earnings from data sale
db.add_fund_transaction(
    family_id=adams_family.id,
    transaction_type="data_sale",
    amount_cents=50,  # $0.50 earned
    description="Weekly activity data sold to research company"
)

print("Data recorded and fund updated!")
```