backend\app\memory\models.py

```python
from datetime import datetime
from sqlalchemy import (
    Column, Integer, String, Text, ForeignKey, DateTime, Boolean, Index, JSON
)
from sqlalchemy.orm import declarative_base, relationship

Base = declarative_base()

__all__ = [
    "Base",
    "Role",
    "Project",
    "RoleProjectLink",
    "MemoryEntry",
    "CanonItem",
    "PromptTemplate",
    "AuditLog",
    "Attachment",
]

# ✅ Role table
class Role(Base):
    __tablename__ = "roles"

    id = Column(Integer, primary_key=True)
    name = Column(String(50), unique=True, nullable=False, index=True)
    description = Column(String(255), nullable=True)

    # Relationships (no cascading deletes to MemoryEntry/CanonItem; DB handles SET NULL)
    memories = relationship(
        "MemoryEntry",
        back_populates="role",
        passive_deletes=True,
    )
    role_projects = relationship("RoleProjectLink", back_populates="role", cascade="all, delete")
    prompt_templates = relationship("PromptTemplate", back_populates="role", cascade="all, delete")
    canon_items = relationship(
        "CanonItem",
        back_populates="role",
        passive_deletes=True,
    )

    def __repr__(self) -> str:
        return f"<Role id={self.id} name={self.name!r}>"

# ✅ Project table
class Project(Base):
    __tablename__ = "projects"

    id = Column(Integer, primary_key=True)
    name = Column(String(100), unique=True, nullable=False, index=True)
    description = Column(Text, nullable=True)
    project_structure = Column(Text, nullable=True)

    # View-only mirrors for FK-int join convenience
    memories = relationship(
        "MemoryEntry",
        back_populates="project",
        primaryjoin="foreign(MemoryEntry.project_id_int)==Project.id",
        viewonly=True,
    )
    role_projects = relationship("RoleProjectLink", back_populates="project", cascade="all, delete")
    canon_items = relationship(
        "CanonItem",
        back_populates="project",
        primaryjoin="foreign(CanonItem.project_id_int)==Project.id",
        viewonly=True,
    )

    def __repr__(self) -> str:
        return f"<Project id={self.id} name={self.name!r}>"

# ✅ Link table for Role ↔ Project (many-to-many)
class RoleProjectLink(Base):
    __tablename__ = "role_project_link"

    id = Column(Integer, primary_key=True)
    role_id = Column(Integer, ForeignKey("roles.id", ondelete="CASCADE"), nullable=False)
    project_id = Column(Integer, ForeignKey("projects.id", ondelete="CASCADE"), nullable=False)

    role = relationship("Role", back_populates="role_projects")
    project = relationship("Project", back_populates="role_projects")

    __table_args__ = (
        Index("ix_role_project_unique", "role_id", "project_id", unique=True),
    )

    def __repr__(self) -> str:
        return f"<RoleProjectLink role_id={self.role_id} project_id={self.project_id}>"

# ✅ Memory entries (chat rows + summaries)
class MemoryEntry(Base):
    __tablename__ = "memory_entries"

    id = Column(Integer, primary_key=True)

    # Dual key: string for API filtering, int for FK joins/analytics
    project_id = Column(String(255), nullable=False, index=True)
    project_id_int = Column(Integer, ForeignKey("projects.id", ondelete="CASCADE"), nullable=True)

    # NOTE: ON DELETE SET NULL aligns with passive_deletes on Role.memories
    role_id = Column(Integer, ForeignKey("roles.id", ondelete="SET NULL"), nullable=True, index=True)
    chat_session_id = Column(String(255), nullable=True, index=True)

    timestamp = Column(DateTime, default=datetime.utcnow, index=True)
    tokens = Column(Integer, nullable=True)
    summary = Column(Text, nullable=True)
    raw_text = Column(Text, nullable=True)

    # Flags
    is_summary = Column(Boolean, default=False, nullable=False, index=True)
    is_ai_to_ai = Column(Boolean, default=False, nullable=False)
```

```python

    deleted = Column(Boolean, default=False, nullable=False, index=True)
    updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)

    role = relationship(
        "Role",
        back_populates="memories",
        passive_deletes=True,
    )
    project = relationship(
        "Project",
        back_populates="memories",
        primaryjoin="foreign(MemoryEntry.project_id_int)==Project.id",
        viewonly=True,
    )

    __table_args__ = (
        Index("ix_mem_role_proj_sess_time", "role_id", "project_id", "chat_session_id", "timestamp"),
        Index("ix_mem_proj_role_is_summary_time", "project_id", "role_id", "is_summary", "timestamp"),
    )

    attachments = relationship(
        "Attachment",
        back_populates="message",
        cascade="all, delete-orphan",
        passive_deletes=True,
    )

    def __repr__(self) -> str:
        return f"<MemoryEntry id={self.id} role_id={self.role_id} proj={self.project_id} sess={self.chat_session_id}>"

# ✅ Canonical items (ADR, CHANGELOG, BACKLOG, GLOSSARY, PMD)
class CanonItem(Base):
    __tablename__ = "canon_items"

    id = Column(Integer, primary_key=True)

    project_id = Column(String(255), nullable=False, index=True)
    project_id_int = Column(Integer, ForeignKey("projects.id", ondelete="CASCADE"), nullable=True)

    role_id = Column(Integer, ForeignKey("roles.id", ondelete="SET NULL"), nullable=True, index=True)

    type = Column(String(32), nullable=False, index=True)
    title = Column(String(256), nullable=False, index=True)
    body = Column(Text, nullable=False)

    tags = Column(JSON, nullable=True)   # SQLite stores as TEXT; SQLAlchemy handles it
    terms = Column(Text, nullable=True)

    created_at = Column(DateTime, default=datetime.utcnow, index=True)
    updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)
    is_active = Column(Boolean, default=True, nullable=False, index=True)

    role = relationship(
        "Role",
        back_populates="canon_items",
        passive_deletes=True,
    )
    project = relationship(
        "Project",
        back_populates="canon_items",
        primaryjoin="foreign(CanonItem.project_id_int)==Project.id",
        viewonly=True,
    )

    __table_args__ = (
        Index("ix_canon_proj_role_type_time", "project_id", "role_id", "type", "created_at"),
        Index("ix_canon_title_terms", "title", "terms"),
    )

    def __repr__(self) -> str:
        return f"<CanonItem id={self.id} type={self.type} title={self.title!r}>"

# ✅ Prompt templates tied to a role
class PromptTemplate(Base):
    __tablename__ = "prompt_templates"

    id = Column(Integer, primary_key=True)
    role_id = Column(Integer, ForeignKey("roles.id", ondelete="CASCADE"), nullable=False, index=True)
    name = Column(String(100), nullable=False)
    content = Column(Text, nullable=False)
    is_default = Column(Boolean, default=False)

    role = relationship("Role", back_populates="prompt_templates")

    __table_args__ = (
        Index("ix_prompt_unique_per_role", "role_id", "name", unique=True),
    )

    def __repr__(self) -> str:
        return f"<PromptTemplate id={self.id} role_id={self.role_id} name={self.name!r}>"

# ✅ Audit log entries with model version
class AuditLog(Base):
    __tablename__ = "audit_logs"

    id = Column(Integer, primary_key=True)

    project_id = Column(String(255), nullable=False, index=True)
    role_id = Column(Integer, nullable=False, index=True)
    chat_session_id = Column(String(255), nullable=True, index=True)

    provider = Column(String(50), nullable=False, index=True)
    action = Column(String(50), nullable=False)
    query = Column(Text, nullable=True)
    timestamp = Column(DateTime, default=datetime.utcnow, index=True)
    model_version = Column(String(50), nullable=True)

    def __repr__(self) -> str:
        return f"<AuditLog id={self.id} provider={self.provider} action={self.action}>"

# ✅ Attachments for file uploads
class Attachment(Base):
    __tablename__ = "attachments"

    id = Column(Integer, primary_key=True, index=True)

    # Link to message
    message_id = Column(Integer, ForeignKey("memory_entries.id", ondelete="CASCADE"), nullable=False, index=True)

    # File info
    filename = Column(String(255), nullable=False)
    original_filename = Column(String(255), nullable=False)
    file_type = Column(String(50), nullable=False)  # image, document, data
    mime_type = Column(String(100), nullable=False)
    file_size = Column(Integer, nullable=False)

    # File path
    file_path = Column(String(500), nullable=False)

    # Metadata
    uploaded_at = Column(DateTime, default=datetime.utcnow, nullable=False)

    # Relations
    message = relationship(
        "MemoryEntry",
        back_populates="attachments",
        passive_deletes=True,
    )

    def __repr__(self) -> str:
        return f"<Attachment(id={self.id}, filename={self.filename}, type={self.file_type})>"
```