Fondamenti di Basi di Dati per AI Generativa

Programma 60 ore - Contestualizzato per Intelligenza Artificiale

---

CONTESTO DEL CORSO

Posizionamento nel Percorso

Questo modulo da 60 ore è parte di un percorso completo su **Intelligenza Artificiale Generativa** che include:

- Agenti AI e automazione
- Gen AI: UX/UI e Content Generation
- Vibe Coding e Testing
- Context Engineering e RAG
- Deep Learning e Reti Neurali
- ModelOps e Machine Learning
- Embodied AI
- Sviluppo Full Stack (Python BE, JS/React FE)

Obiettivo Strategico

Non formare Database Administrator, ma AI Engineer che capiscono i dati.

Gli studenti devono: ✔️ Comprendere come i dati alimentano l'AI
✔️ Saper progettare database per applicazioni AI
✔️ Essere pronti per i moduli RAG, Context Engineering, ModelOps
✔️ Gestire training data, embeddings, logging di modelli

STRUTTURA CORSO: 60 ORE

Formato: 12 sessioni da 5 ore ciascuna (teoria + pratica integrata)

Distribuzione:

- 30% Teoria (18h) - Concetti fondamentali
- 50% Pratica (30h) - Lab ed esercitazioni
- 20% Progetto (12h) - Applicazione AI-focused

Approccio: Ogni concetto database = caso d'uso AI immediato

SESSIONE 1: Database e AI - Il Foundation Layer (5h)

Parte 1: Perché i Database nell'Era AI (2h)

Teoria:

- L'AI non è magia: è matematica + DATI
- Caso studio: Come funziona ChatGPT dietro le quinte
  - Training data: Petabyte di testo organizzato
  - Fine-tuning data: Conversazioni strutturate
  - User data: Database di interazioni
- Caso studio: Stable Diffusion
  - 5+ miliardi di immagini con metadata
  - Database di coppie testo-immagine
  - Embedding storage per ricerca semantica

Esempi Concreti AI:

- Netflix recommendation: Database + ML models
- Autonomous driving: Sensor data in database real-time
- Medical AI: Database pazienti per diagnostica

Domanda chiave: "Vorreste creare un chatbot AI? Dove salvate le conversazioni, le preferenze utente, i feedback?"

Parte 2: Panorama Database per AI (1.5h)

Tipi di Database:

1. **SQL/Relazionale** (focus del corso)
   - Structured data, transazioni, integrità
   - Uso AI: Training data, user profiles, logs
2. **NoSQL** (overview)
   - Document DB (MongoDB): Flexible schemas
   - Uso AI: Unstructured data, rapid prototyping
3. **Vector Databases** (preview per Context Engineering)
   - Pinecone, Chroma, Weaviate

- Uso AI: RAG, semantic search, embeddings

Setup Environment (1.5h):

- SQLite (locale, no server, perfetto per learning)
- DB Browser for SQLite
- Python + sqlite3
- Prima query: `SELECT * FROM ai_training_data`

Lab:

- Explore pre-populated database: "ChatbotConversations"
- Tabelle: Users, Conversations, Messages, Feedback
- Query semplici per capire la struttura

SESSIONE 2: Database Design per AI Applications (5h)

Parte 1: Entities e Relationships nel Mondo AI (2h)

Caso Studio Guidato: Chatbot Training System

Entities da modellare:

- Users (chi usa il chatbot)
- Conversations (sessioni di chat)
- Messages (singoli messaggi)
- Intents (intenzioni riconosciute dall'AI)
- Feedback (thumbs up/down)
- TrainingExamples (dati per fine-tuning)

ER Diagram on Board:

- One-to-Many: User → Conversations
- One-to-Many: Conversation → Messages
- Many-to-Many: Messages ↔ Intents

Normalization Pratica:

- Perché non mettere tutto in una tabella?

- Come evitare ridondanza nei training data
- Quando denormalizzare per performance

Parte 2: Data Types per AI Data (1.5h)

SQL Data Types con focus AI:

- `TEXT`: Prompts, risposte, training text
- `INTEGER`: IDs, counters, ratings
- `FLOAT`: Confidence scores, embeddings components
- `TIMESTAMP`: Quando è stata generata una risposta
- `BOOLEAN`: Is_successful, needs_review
- `BLOB`: Immagini, audio (cenni)

Constraints Critici per AI:

- `PRIMARY KEY`: Identificare univocamente training examples
- `FOREIGN KEY`: Mantenere integrità relazionale
- `NOT NULL`: Campi essenziali (es: prompt text)
- `UNIQUE`: Evitare training data duplicati
- `CHECK`: Validare range (es: rating 1-5)

Parte 3: Lab Design (1.5h)

Esercizio Pratico: Progettare database per:

1. **Image Generation App** (tipo Midjourney)
    - Users, Prompts, GeneratedImages, Styles, Ratings
2. **Voice Assistant** (tipo Alexa)
    - Users, Commands, Responses, Devices, UsageLog

Deliverable: ER Diagram + lista tabelle con campi e tipi

SESSIONE 3: SQL DDL - Creare Strutture Database (5h)

Parte 1: CREATE TABLE per AI Systems (2h)

Syntax e Best Practices:

```sql
CREATE TABLE training_prompts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    prompt_text TEXT NOT NULL,
    category TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    token_count INTEGER,
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Casi AI Reali:

- Tabella per embeddings (text + vector representation)
- Tabella per model outputs con metadata
- Tabella per A/B testing risultati

Parte 2: Alter, Drop, Index (1.5h)

ALTER TABLE:

- Aggiungere colonna `sentiment_score` a tabella messaggi
- Modificare colonna per supportare prompts più lunghi

INDEX Creation:

- Perché indexare `user_id` in tabella messages
- Index su `created_at` per analisi temporali
- Composite index per query complesse

DROP con cautela:

- Mai droppare in produzione senza backup
- Scenario: Re-design database dopo feedback utenti

Parte 3: Lab Pratico (1.5h)

Progetto: AI Chatbot Database

1. Creare 5 tabelle correlate
2. Inserire constraints appropriati
3. Creare indici strategici

4. Documentare design choices

SESSIONE 4: SQL DML - Popolare il Database (5h)

Parte 1: INSERT - Training Data Loading (1.5h)

Single e Bulk Insert:

```sql
-- Singolo training example
INSERT INTO training_data (prompt, expected_output, quality_score)
VALUES ('Explain quantum computing', 'Quantum computing is...', 4.5);

-- Bulk insert per dataset
INSERT INTO training_data VALUES
  (1, 'prompt1', 'output1', 4.2),
  (2, 'prompt2', 'output2', 3.8),
  -- ... 1000s of rows
```

AI Context:

- Importare dataset da CSV (Kaggle, HuggingFace)
- Preparare dati per fine-tuning
- Versioning dei training data

Parte 2: UPDATE - Refining Data (1.5h)

Update Patterns:

```sql
-- Aggiornare quality score dopo review
UPDATE training_data
SET quality_score = 5.0, reviewed = TRUE
WHERE id = 123;

-- Batch update per categorization
UPDATE prompts
SET category = 'coding'
WHERE prompt_text LIKE '%code%' OR prompt_text LIKE '%program%';
```

AI Use Cases:

- Aggiornare feedback utenti
- Marcare dati come "verified" dopo human review
- Correggere labels errati

Parte 3: DELETE e Data Hygiene (1h)

Safe Deletion:

```sql
-- Rimuovere low-quality examples
DELETE FROM training_data WHERE quality_score < 2.0;

-- Soft delete (preferito in AI)
UPDATE training_data
SET is_active = FALSE, deleted_at = CURRENT_TIMESTAMP
WHERE quality_score < 2.0;
```

AI Context:

- GDPR: Cancellare dati utente
- Rimuovere toxic/biased examples
- Data retention policies

Parte 4: Lab (1h)

Esercizio:

1. Popolare database chatbot con 100+ record
2. Simulare user feedback (UPDATE ratings)
3. Cleanup low-quality data (DELETE/soft delete)

SESSIONE 5: SQL Query Base - Estrarre Insights (5h)

Parte 1: SELECT Fundamentals (1.5h)

Basic SELECT:

```sql
```

```sql
-- Tutti i prompt di un utente
SELECT * FROM prompts WHERE user_id = 42;

-- Prompts più recenti
SELECT prompt_text, created_at
FROM prompts
ORDER BY created_at DESC
LIMIT 10;

-- Unique categories
SELECT DISTINCT category FROM prompts;
```

AI Analytics:

- Quali prompt generano più engagement?
- Quali categorie sono più popolari?
- Quanti utenti attivi oggi?

Parte 2: Filtering e Pattern Matching (1.5h)

WHERE Clause Avanzato:

```sql
-- Prompts con alta quality
SELECT * FROM training_data
WHERE quality_score >= 4.0 AND token_count < 500;

-- Search in prompts
SELECT * FROM prompts
WHERE prompt_text LIKE '%AI%' OR prompt_text LIKE '%machine learning%';

-- Date range analysis
SELECT * FROM user_interactions
WHERE created_at BETWEEN '2024-01-01' AND '2024-12-31';
```

Pattern per AI:

- Trovare esempi specifici per fine-tuning
- Analizzare trend temporali
- Filtrare per quality thresholds

Parte 3: Aggregations per Statistics (1.5h)

Aggregate Functions:

```sql
-- Statistiche base
SELECT
    COUNT(*) as total_prompts,
    AVG(quality_score) as avg_quality,
    MAX(token_count) as longest_prompt,
    MIN(created_at) as first_prompt
FROM training_data;

-- Group by category
SELECT
    category,
    COUNT(*) as count,
    AVG(quality_score) as avg_quality
FROM prompts
GROUP BY category
HAVING count > 10
ORDER BY avg_quality DESC;
```

AI Insights:

- Distribution delle categorie
- Average response time del modello
- User engagement metrics

Parte 4: Lab Analytics (0.5h)

Esercizio: Rispondere con SQL:

1. Quanti utenti hanno usato il chatbot questa settimana?
2. Qual è la categoria più popolare?
3. Qual è la quality score media per categoria?
4. Quali utenti hanno dato più feedback?

SESSIONE 6: JOIN - Connettere i Dati (5h)

Parte 1: INNER JOIN per Relazioni (2h)

Basic JOIN:

```sql
-- Users con le loro conversazioni
SELECT u.username, c.title, c.created_at
FROM users u
INNER JOIN conversations c ON u.id = c.user_id;

-- Messages con user info
SELECT u.username, m.message_text, m.timestamp
FROM messages m
INNER JOIN conversations c ON m.conversation_id = c.id
INNER JOIN users u ON c.user_id = u.id;
```

AI Context:

- Collegare prompts a generated outputs
- User behavior analysis
- Training example con metadata completo

Parte 2: LEFT JOIN per Completeness (1.5h)

Include Missing Data:

```sql
-- Tutti gli utenti, anche senza conversazioni
SELECT u.username, COUNT(c.id) as conversation_count
FROM users u
LEFT JOIN conversations c ON u.id = c.user_id
GROUP BY u.id;

-- Prompts senza feedback (da revieware)
SELECT p.prompt_text
FROM prompts p
LEFT JOIN feedback f ON p.id = f.prompt_id
WHERE f.id IS NULL;
```

AI Use Cases:

- Identificare utenti inattivi

- Training data non ancora validated
- Missing labels in dataset

Parte 3: Multi-Table Joins (1h)

Complex Relationships:

```sql
-- Complete conversation view
SELECT
    u.username,
    c.title,
    m.message_text,
    m.is_ai_generated,
    f.rating
FROM users u
JOIN conversations c ON u.id = c.user_id
JOIN messages m ON c.id = m.conversation_id
LEFT JOIN feedback f ON m.id = f.message_id
WHERE c.created_at > '2024-01-01';
```

Parte 4: Lab JOIN (0.5h)

Esercizio:

1. Report: User + loro prompts + ratings
2. Find: Prompts generati ma mai rated
3. Analysis: Categorie più popolari per user type

SESSIONE 7: Subqueries e Query Avanzate (5h)

Parte 1: Subqueries in WHERE (1.5h)

Nested Queries:

```sql
```

```sql
-- Utenti più attivi della media
SELECT username FROM users
WHERE id IN (
    SELECT user_id FROM prompts
    GROUP BY user_id
    HAVING COUNT(*) > (SELECT AVG(prompt_count)
                      FROM (SELECT COUNT(*) as prompt_count
                            FROM prompts GROUP BY user_id))
);

-- Prompts con quality sopra la media della categoria
SELECT * FROM training_data t1
WHERE quality_score > (
    SELECT AVG(quality_score)
    FROM training_data t2
    WHERE t2.category = t1.category
);
```

AI Applications:

- Trovare outliers in training data
- Identificare power users per beta testing
- Quality control automatico

Parte 2: CTE (Common Table Expressions) (1.5h)

Leggibilità e Riutilizzo:

```sql
-- User statistics con CTE
WITH user_stats AS (
    SELECT
        user_id,
        COUNT(*) as total_prompts,
        AVG(token_count) as avg_tokens
    FROM prompts
    GROUP BY user_id
)
SELECT u.username, us.total_prompts, us.avg_tokens
FROM users u
JOIN user_stats us ON u.id = us.user_id
WHERE us.total_prompts > 100;
```

AI Context:

- Pipeline di data processing
- Multi-step analysis per model evaluation
- Reporting complesso

Parte 3: Window Functions (1.5h)

Advanced Analytics:

```sql
-- Ranking prompts per quality
SELECT
    prompt_text,
    quality_score,
    RANK() OVER (ORDER BY quality_score DESC) as quality_rank
FROM training_data;

-- Rolling average per trend analysis
SELECT
    date,
    daily_prompts,
    AVG(daily_prompts) OVER (
        ORDER BY date
        ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
    ) as week_avg
FROM daily_stats;
```

AI Use Cases:

- Time series analysis
- Ranking training examples
- Moving averages per model performance

Parte 4: Lab Avanzato (0.5h)

Challenge Queries:

1. Top 10 users per engagement score (composite metric)
2. Trend analysis: prompts per giorno ultimi 30 giorni
3. Category performance comparison

SESSIONE 8: Database per RAG Systems (5h)

Parte 1: Cosa è RAG e Perché Serve (1h)

Retrieval-Augmented Generation:

- LLM limitations: knowledge cutoff, hallucinations
- Solution: Retrieve relevant docs → Augment prompt → Generate
- Database role: Store e retrieve documents efficiently

Architecture:

```
User Query → Embedding → Vector Search in DB →
Retrieve Top-K Docs → Context + Query → LLM → Response
```

Parte 2: Structuring Data for RAG (2h)

Document Storage:

```sql
CREATE TABLE documents (
    id INTEGER PRIMARY KEY,
    title TEXT NOT NULL,
    content TEXT NOT NULL,
    source TEXT,
    created_at TIMESTAMP,
    chunk_id INTEGER,
    parent_document_id INTEGER,
    metadata JSON
);

CREATE TABLE embeddings (
    id INTEGER PRIMARY KEY,
    document_id INTEGER,
    embedding_vector TEXT, -- JSON array for SQLite
    model_version TEXT,
    FOREIGN KEY (document_id) REFERENCES documents(id)
);
```

Chunking Strategy:

- Perché dividere documents in chunks
- Overlap tra chunks per contesto

- Metadata per filtering

Parte 3: Querying for Context (1.5h)

Similarity Search Simulation:

```sql
-- Retrieve documents per keyword (simplified)
SELECT d.title, d.content, d.source
FROM documents d
WHERE d.content LIKE '%quantum computing%'
ORDER BY created_at DESC
LIMIT 5;

-- With metadata filtering
SELECT d.content
FROM documents d
WHERE
    d.content LIKE '%AI%' AND
    json_extract(d.metadata, '$.category') = 'technical'
LIMIT 3;
```

Note: Vector similarity in pratica richiede vector DB, ma comprendiamo il pattern

Parte 4: Lab RAG Database (0.5h)

Progetto:

1. Creare database per knowledge base
2. Popolare con 50+ document chunks
3. Query simulation per retrieval
4. Discussion: SQL vs Vector DB trade-offs

SESSIONE 9: NoSQL Overview per AI (5h)

Parte 1: Document Databases (2h)

MongoDB Concepts:

- JSON-like documents
- Schema flexibility per AI experiments
- Quando usare vs SQL

Use Cases AI:

```javascript
// MongoDB document per AI experiment
{
  "_id": ObjectId("..."),
  "experiment_name": "gpt4_finetuning_v2",
  "model_config": {
    "temperature": 0.7,
    "max_tokens": 500,
    "top_p": 0.9
  },
  "training_data": [
    {"prompt": "...", "completion": "..."},
    // ... flexible structure
  ],
  "results": {
    "accuracy": 0.92,
    "loss": 0.15
  },
  "created_at": ISODate("2024-01-15")
}
```

Vantaggi per AI:

- Rapid prototyping
- Nested data structures
- Schema evolution

Parte 2: Vector Databases Intro (2h)

Perché Esistono:

- Embeddings: Text → 1536-dim vector (OpenAI)
- Need: Find similar vectors FAST
- SQL limitation: No native vector similarity

Vector DB Examples:

- **Pinecone**: Managed, scalable
- **Chroma**: Open-source, local
- **Weaviate**: Open-source, self-hosted

Conceptual Example:

```python
# Pseudo-code
embedding = model.encode("What is AI?")  # → [0.23, -0.15, ...]

# Vector DB query
results = vectordb.query(
    embedding=embedding,
    top_k=5,
    filter={"category": "AI"}
)
# Returns 5 most similar documents
```

Parte 3: SQL vs NoSQL vs Vector (0.5h)

Decision Matrix:

Use Case	Best Choice
User profiles, transactions	SQL
Flexible AI experiment logs	NoSQL (Mongo)
Semantic search, RAG	Vector DB
Training data structured	SQL
Embeddings storage	Vector DB
Hybrid: Metadata in SQL, vectors in VectorDB	Both

Parte 4: Hands-on Demo (0.5h)

Live Demo:

- MongoDB: Insert flexible AI experiment
- Chroma DB: Basic similarity search
- Discussion: Integration strategies

SESSIONE 10: Python + Database Integration (5h)

Parte 1: sqlite3 Module (1.5h)

Connection e Basic Operations:

```python
import sqlite3

# Connect
conn = sqlite3.connect('ai_chatbot.db')
cursor = conn.cursor()

# Query
cursor.execute("SELECT * FROM prompts WHERE user_id = ?", (user_id,))
results = cursor.fetchall()

# Insert with safety
cursor.execute("""
    INSERT INTO prompts (user_id, prompt_text, created_at)
    VALUES (?, ?, ?)
""", (user_id, prompt, datetime.now()))

conn.commit()
conn.close()
```

SQL Injection Prevention:

- NEVER: `f"SELECT * FROM users WHERE id = {user_id}"`
- ALWAYS: Parametrized queries con `?`

Parte 2: Context Managers e Best Practices (1h)

Clean Code:

```python
def get_user_prompts(user_id):
    with sqlite3.connect('db.db') as conn:
        conn.row_factory = sqlite3.Row  # Dict-like access
        cursor = conn.cursor()
        cursor.execute("""
            SELECT prompt_text, created_at, rating
            FROM prompts
            WHERE user_id = ?
            ORDER BY created_at DESC
        """, (user_id,))
        return [dict(row) for row in cursor.fetchall()]
```

Parte 3: Building AI Data Pipeline (1.5h)

Example: Training Data Manager:

```python
class TrainingDataManager:
    def __init__(self, db_path):
        self.db_path = db_path

    def add_example(self, prompt, completion, quality):
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO training_data
                (prompt, completion, quality_score, created_at)
                VALUES (?, ?, ?, ?)
            """, (prompt, completion, quality, datetime.now()))
            return cursor.lastrowid

    def get_high_quality_batch(self, batch_size=100):
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            cursor.execute("""
                SELECT prompt, completion
                FROM training_data
                WHERE quality_score >= 4.0
                ORDER BY RANDOM()
                LIMIT ?
            """, (batch_size,))
            return cursor.fetchall()
```

Parte 4: Lab Python Integration (1h)

Progetto:

1. Creare classe DatabaseManager
2. CRUD operations per AI chatbot
3. Analytics functions (stats, reports)
4. Error handling completo

SESSIONE 11: Database in Production per AI (5h)

Parte 1: Performance e Optimization (2h)

Query Optimization:

- EXPLAIN QUERY PLAN
- Index usage analysis
- Query refactoring examples

Bottlenecks Comuni AI:

```sql
-- SLOW: Full table scan
SELECT * FROM training_data WHERE quality_score > 4.0;

-- FAST: With index
CREATE INDEX idx_quality ON training_data(quality_score);
SELECT * FROM training_data WHERE quality_score > 4.0;
```

Batching per Large Datasets:

```python
def process_large_dataset(batch_size=1000):
    offset = 0
    while True:
        batch = fetch_batch(offset, batch_size)
        if not batch:
            break
        process(batch)
        offset += batch_size
```

Parte 2: Backup, Recovery, Versioning (1.5h)

Data è Critico:

- Backup strategies per training data
- Version control per datasets (DVC intro)
- Disaster recovery planning

Practical Backup:

```bash
```

```
# SQLite backup
sqlite3 ai_db.db ".backup backup_2024-01-15.db"

# Automated daily backup
cron: 0 2 * * * /scripts/backup_db.sh
```

Parte 3: Logging e Monitoring (1h)

Production Monitoring:

```python
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def query_with_logging(query, params):
    start = time.time()
    try:
        result = execute_query(query, params)
        duration = time.time() - start
        logger.info(f"Query executed in {duration:.2f}s")
        return result
    except Exception as e:
        logger.error(f"Query failed: {e}")
        raise
```

Metrics to Track:

- Query latency
- Database size growth
- Error rates
- Connection pool usage

Parte 4: Security e GDPR (0.5h)

Data Protection:

- Encrypt sensitive data
- Access control
- Audit trails
- GDPR compliance: right to deletion

```python
def gdpr_delete_user_data(user_id):
    """Complete user data deletion"""
    with transaction():
        delete_from_prompts(user_id)
        delete_from_feedback(user_id)
        delete_from_conversations(user_id)
        log_deletion(user_id)
```

SESSIONE 12: Progetto Finale - AI Application Database (5h)

Opzioni Progetto (Scegliere 1)

Opzione A: RAG Chatbot Backend

Requirements:

- Database per documents (chunks + metadata)
- User conversations storage
- Query history e analytics
- Simple retrieval simulation
- Python integration completa

Deliverables:

- Database design (ER + SQL)
- Populated database (50+ documents)
- Python CRUD functions
- Analytics queries (usage stats)
- Documentation

Opzione B: AI Training Data Manager

Requirements:

- Training examples storage
- Quality scoring system
- Versioning dei datasets

- Filtering e export functions
- Batch processing capability

Deliverables:

- Database schema
- Python pipeline per data ingestion
- Quality control queries
- Export to JSONL for fine-tuning
- Stats dashboard (SQL queries)

Opzione C: AI Experiment Tracker

Requirements:

- Experiments metadata storage
- Model configurations
- Results logging
- Comparison queries
- Best model selection logic

Deliverables:

- Database per MLOps
- Python logger per experiments
- Comparison queries
- Leaderboard generation
- Visualizzazioni (optional)

Evaluation Criteria

- Database design quality (30%)
- SQL query efficiency (25%)
- Python integration (25%)
- AI relevance (10%)
- Documentation (10%)

Presentation (30 min per team)

- 5 min: Problem e design
- 10 min: Demo live
- 5 min: Code walkthrough
- 10 min: Q&A

MATERIALI E RISORSE

Software

- SQLite + DB Browser
- Python 3.11+
- sqlite3 module (built-in)
- VS Code con SQLite extension

Dataset per Esercitazioni

- HuggingFace datasets (CSV export)
- Kaggle AI/ML datasets
- Synthetic chatbot conversations (generated)

Reference Documentation

- SQLite docs: sqlite.org
- SQL tutorial: w3schools.com/sql
- Python sqlite3: docs.python.org

Additional Resources

- Pinecone learning center (vector DB)
- ChromaDB docs
- RAG tutorials
- DVC (Data Version Control)

COLLEGAMENTI CON ALTRI MODULI

Pre-requisiti da Altri Corsi

- **Fondamenti Programmazione**: Python basics
- **Fondamenti Version Control**: Git per dataset versioning

Preparazione per Moduli Successivi

- **Context Engineering**: Database per RAG
- **ModelOps**: Training data management
- **Python Backend**: API + database integration
- **Agenti AI**: State persistence, logging

SUCCESS METRICS

Il corso ha successo se gli studenti:

✅ Sanno progettare database per AI applications
✅ Scrivono query SQL efficienti per data retrieval
✅ Integrano database in Python AI projects
✅ Comprendono SQL vs NoSQL vs Vector DB trade-offs
✅ Vedono database come enabler, non ostaco