Fondamenti di Basi di Dati per Intelligenza Artificiale Generativa

Programma Dettagliato - 8 Settimane (60 ore totali)

FILOSOFIA DEL CORSO

Messaggio centrale: "L'AI generativa non è magia - è matematica applicata a DATI. Questo corso vi insegna a gestire il carburante che alimenta l'AI."

Approccio: Ogni concetto database → Applicazione AI immediata

---

SETTIMANA 1: Database e AI - Il Fondamento Necessario

Lezione 1: Perché l'AI Ha Bisogno di Database (3 ore)

Parte 1: Il Ruolo dei Dati nell'AI (1.5h)

- **Opening provocatorio**:
  - Demo live: ChatGPT che "allucina" vs ChatGPT con RAG
  - Domanda: "Cosa fa la differenza?" → DATI BEN ORGANIZZATI
- **Training data pipeline**:
  - Come GPT-4 è stato addestrato: miliardi di righe in database
  - Stable Diffusion: database di immagini + metadata
  - Come Netflix raccomanda film: database + AI
- **Types of data in AI**:
  - Training data (storico, immutabile)
  - Production data (real-time, transazionale)
  - User data (conversazioni, preferenze)
  - Model metadata (versioni, performance)
- **Case study**:
  - Midjourney: 100M+ immagini organizzate in database
  - Come cercano "un gatto steampunk" tra miliardi di dati?

Parte 2: Panoramica Ecosistema Database (1.5h)

- **Traditional SQL Databases**:

    - Quando: dati strutturati, transazioni, consistency
    - AI use case: user management, analytics, logs
- **NoSQL Databases**:
    - Quando: dati non strutturati, scalabilità, flessibilità
    - AI use case: training data storage, JSON responses
- **Vector Databases** (preview):
    - Quando: similarity search, embeddings
    - AI use case: RAG systems, semantic search
    - Demo rapida: Pinecone o Chroma
- **Cloud Database Services**:
    - AWS RDS, DynamoDB, Azure Cosmos DB
    - Perché le aziende AI usano cloud database

Lab (45 min):

- Setup ambiente: DB Browser for SQLite
- Esplorare sample database "AI Training Logs"
- Identificare strutture dati tipiche AI apps

Homework: Research un'AI application famosa e identificare che tipo di database potrebbe usare (es: Spotify, Netflix, ChatGPT)

Lezione 2: Organizzazione Dati - Tabelle e Relazioni (3 ore)

Parte 1: Database Relazionali Basics (1.5h)

- **Concetti fondamentali**:
    - Tabelle = spreadsheets strutturati
    - Righe (records) vs Colonne (fields)
    - Schema: la "blueprint" del database
- **AI Context**:
    - Tabella "Users" per AI chatbot
    - Tabella "Conversations" con user_id
    - Tabella "Prompts" con metadata

  - Tabella "Model_Outputs" per logging
- **Tipi di dato rilevanti AI**:
  - TEXT: prompt, responses, descriptions
  - INTEGER: IDs, counters, ratings
  - FLOAT: scores, embeddings (accenno)
  - DATETIME: timestamps cruciali per AI
  - BLOB: immagini, file (cenni)

Parte 2: Relazioni tra Tabelle (1.5h)

- **One-to-Many**:
  - Un utente → molte conversazioni
  - Un model → molti outputs
  - Una categoria → molte immagini generate
- **Many-to-Many**:
  - Utenti ↔ Progetti AI
  - Prompts ↔ Tags
  - Training examples ↔ Labels
- **Primary Keys e Foreign Keys**:
  - Perché cruciali per integrità dati
  - Come linkare tabelle
  - Esempi AI pratici

Lab (45 min):

- Design su carta: Database per "AI Image Generator"
  - Tabelle: Users, Images, Prompts, Styles, Generations
- Identificare relazioni
- Peer review dei design

Homework: Design database per un chatbot AI (Users, Conversations, Messages, Feedback)

Lezione 3: DBMS e Componenti (2 ore)

Parte 1: Cos'è un DBMS (1h)

- **Definizione e funzioni**:
  - Database Management System
  - Query processor
  - Storage engine
  - Transaction manager
- **Perché non usare file Excel/CSV**:
  - Concurrent access (multi-user)
  - Data integrity
  - Query optimization
  - Backup e recovery
- **DBMS popolari**:
  - SQLite: embedded, perfetto per prototipare
  - PostgreSQL: open source, potente, estensibile
  - MySQL: diffuso, web applications
  - MongoDB: NoSQL, flessibile

Parte 2: DBMS nel Contesto AI (1h)

- **PostgreSQL + pgvector**:
  - Extensione per vector embeddings
  - Usato da molte startup AI
- **MongoDB**:
  - Storage JSON responses da LLM
  - Flexible schema per dati AI
- **Redis**:
  - Caching responses AI (velocità!)
  - Session management chatbot
- **Cloud-managed DBMS**:
  - AWS RDS (gestito, scalabile)
  - Supabase (PostgreSQL + API instant)
  - MongoDB Atlas

Demo Live:

- Creare database SQLite locale
- Inserire dati sample "AI conversations"
- Mostrare come query veloce vs file search

Lezione 4: Progettazione ER - Modellare i Dati (3 ore)

Parte 1: Entity-Relationship Diagrams (1.5h)

- **Entities**: "cose" nel nostro dominio
  - User, Prompt, Image, Model, Conversation
- **Attributes**: proprietà delle entities
  - User: id, email, created_at, tier
  - Prompt: id, text, user_id, timestamp, tokens
- **Relationships**: come entities si collegano
  - User "has many" Prompts
  - Prompt "generates" Image
  - User "rates" Image
- **Cardinalità**:
  - 1:1, 1:N, N:M
  - Esempi pratici AI

Parte 2: ER per AI Applications (1.5h)

- **Case Study 1: AI Chatbot**
  - Entities: User, Conversation, Message, Feedback
  - ER diagram completo
- **Case Study 2: Image Generation Platform**
  - Entities: User, Project, Image, Prompt, Style
  - Relazioni complesse
- **Case Study 3: RAG System**
  - Entities: Document, Chunk, Embedding, Query, Response
  - Come modellare il flusso RAG

Lab (45 min):

- Gruppi di 3: Design ER per una delle seguenti app AI:
    1. AI Video Generator con prompts e styles
    2. Code Assistant con project context
    3. AI Content Moderation system
- Presentazione rapida (5 min per gruppo)

Homework: Completare ER diagram del progetto gruppo + documentazione

---

Lezione 5: Normalizzazione e Best Practices (2 ore)

Parte 1: Concetti di Normalizzazione (1h)

- **Perché normalizzare**:
    - Ridurre ridondanza
    - Evitare anomalie (insert, update, delete)
    - Ottimizzare storage
- **1NF, 2NF, 3NF** (concetti, non formule):
    - 1NF: Atomic values, no repeating groups
    - 2NF: No partial dependencies
    - 3NF: No transitive dependencies
- **AI Context**:
    - Prompt template separato da prompt instances
    - User profile separato da user activity
    - Model metadata separato da model runs

Parte 2: Design Patterns per AI Data (1h)

- **Temporal data**:
    - Tracking changes nel tempo (versioning prompts)
    - Audit logs per AI decisions
- **Metadata management**:
    - Tags, categories per AI assets
    - Flexible schema con JSONB (PostgreSQL)

- **Performance considerations**:
  - Quando denormalizzare per speed
  - Trade-offs in AI applications (speed vs consistency)

Lab pratico:

- Analizzare database "mal progettato" per AI app
- Identificare problemi
- Proporre normalizzazione

WEEKEND PROJECT - Settimana 1

Deliverable: Database Design Document per un'AI Application

Scegliere uno scenario:

1. AI Writing Assistant con history e templates
2. AI Art Gallery con generations e collections
3. AI Code Review Tool con projects e feedback

Documento include:

- ER Diagram completo
- Schema tabelle (nomi, tipi dato, constraints)
- Descrizione relazioni
- Giustificazione design choices
- Casi d'uso principali

Presentazione: Lunedì Settimana 2 (10 min per studente/gruppo)

SETTIMANA 2: SQL Fundamentals - Data Definition Language

Lezione 1: Introduzione a SQL e CREATE (3 ore)

Parte 1: SQL Overview (1h)

- **Cos'è SQL**:
  - Structured Query Language

  - Standard per database relazionali
  - Linguaggio dichiarativo (cosa, non come)
- **SQL Categories**:
  - DDL: Data Definition (CREATE, ALTER, DROP)
  - DML: Data Manipulation (INSERT, UPDATE, DELETE)
  - DQL: Data Query (SELECT)
  - DCL: Data Control (GRANT, REVOKE)
- **SQL per AI Engineers**:
  - Setup schema per AI apps
  - CRUD operations su training data
  - Query analytics su model performance
  - Backup e migration

Parte 2: CREATE DATABASE e CREATE TABLE (2h)

- **CREATE DATABASE** (SQLite: attach database)

sql

```
-- Per PostgreSQL/MySQL
CREATE DATABASE ai_chatbot_prod;
```

- **CREATE TABLE syntax completa**:

sql

```
```

```
CREATE TABLE users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    email TEXT NOT NULL UNIQUE,
    username TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    subscription_tier TEXT DEFAULT 'free'
);

CREATE TABLE conversations (
    conversation_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    title TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    model_version TEXT DEFAULT 'gpt-4',
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

CREATE TABLE messages (
    message_id INTEGER PRIMARY KEY AUTOINCREMENT,
    conversation_id INTEGER NOT NULL,
    role TEXT NOT NULL CHECK(role IN ('user', 'assistant', 'system')),
    content TEXT NOT NULL,
    tokens_used INTEGER,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (conversation_id) REFERENCES conversations(conversation_id)
);
```

Lab (1h):

Esercizio Guidato: Creare database per "AI Image Generator"

- Tabella users
- Tabella prompts (con user_id FK)
- Tabella generated_images (con prompt_id FK)
- Tabella image_ratings (user feedback)

Challenge: Aggiungere tabelle per:

- Style presets
- Generation history con parametri (steps, cfg_scale, seed)

Lezione 2: Data Types e Constraints (3 ore)

Parte 1: Tipi di Dato SQL (1.5h)

Numerici:

- INTEGER: IDs, counters, ratings (1-5)
- REAL/FLOAT: scores (0.0-1.0), probabilities
- DECIMAL: prezzi (se monetization)

Testo:

- TEXT: prompts, responses, descriptions (unlimited in SQLite)
- VARCHAR(n): email, username (limited)
- CHAR(n): fixed (codes, tags)

Temporali:

- DATETIME: timestamps cruciali per AI
- DATE: solo data
- TIME: solo ora

Binari (cenni):

- BLOB: embeddings, immagini small
- Meglio: store path, file in object storage (S3)

JSON (PostgreSQL/MySQL):

- JSONB: metadata flessibili AI

```sql
  generation_params JSONB -- {steps: 50, cfg_scale: 7.5, seed: 42}
```

Parte 2: Constraints per Data Integrity (1.5h)

PRIMARY KEY:

```sql
user_id INTEGER PRIMARY KEY AUTOINCREMENT
```

FOREIGN KEY:

```sql
FOREIGN KEY (user_id) REFERENCES users(user_id)
  ON DELETE CASCADE  -- Delete conversations if user deleted
  ON UPDATE CASCADE
```

NOT NULL:

```sql
email TEXT NOT NULL  -- Email obbligatoria
```

UNIQUE:

```sql
email TEXT UNIQUE  -- No duplicate emails
```

CHECK:

```sql
rating INTEGER CHECK(rating BETWEEN 1 AND 5)
model_name TEXT CHECK(model_name IN ('gpt-4', 'claude-3', 'llama-2'))
```

DEFAULT:

```sql
created_at DATETIME DEFAULT CURRENT_TIMESTAMP
status TEXT DEFAULT 'pending'
```

AI-Specific Constraints Examples:

```sql
```

```
CREATE TABLE ai_generations (
    generation_id INTEGER PRIMARY KEY,
    user_id INTEGER NOT NULL,
    prompt TEXT NOT NULL CHECK(length(prompt) >= 3),
    negative_prompt TEXT,
    steps INTEGER DEFAULT 50 CHECK(steps BETWEEN 1 AND 150),
    cfg_scale REAL DEFAULT 7.5 CHECK(cfg_scale BETWEEN 1.0 AND 20.0),
    seed INTEGER,
    model_version TEXT DEFAULT 'stable-diffusion-xl',
    status TEXT DEFAULT 'pending' CHECK(status IN ('pending', 'processing', 'completed', 'failed')),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    completed_at DATETIME,
    image_path TEXT,
    generation_time_seconds REAL,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

Lab (1h):

Hands-on: Creare schema completo per AI Text-to-Speech service

- Users table
- Voice models table
- Text inputs table
- Audio outputs table (con constraints appropriati)

Lezione 3: INSERT, UPDATE, DELETE (3 ore)

Parte 1: INSERT - Popolare Dati (1h)

INSERT single row:

```sql
INSERT INTO users (email, username, subscription_tier)
VALUES ('alice@example.com', 'alice_ai', 'pro');
```

INSERT multiple rows:

```sql
```

```sql
INSERT INTO users (email, username, subscription_tier) VALUES
  ('bob@example.com', 'bob_dev', 'free'),
  ('charlie@example.com', 'charlie_creator', 'enterprise'),
  ('diana@example.com', 'diana_artist', 'pro');
```

INSERT with auto-generated ID:

```sql
INSERT INTO conversations (user_id, title, model_version)
VALUES (1, 'Debug my Python code', 'gpt-4');
-- conversation_id auto-generato
```

INSERT ... SELECT (advanced):

```sql
-- Copy successful generations to archive
INSERT INTO archived_generations
SELECT * FROM ai_generations
WHERE status = 'completed' AND created_at < '2024-01-01';
```

Parte 2: UPDATE - Modificare Dati (1h)

UPDATE syntax:

```sql
UPDATE table_name
SET column1 = value1, column2 = value2
WHERE condition;
```

AI Examples:

```sql
```

```
-- Update generation status
UPDATE ai_generations
SET status = 'completed',
    completed_at = CURRENT_TIMESTAMP,
    generation_time_seconds = 12.5,
    image_path = '/outputs/img_12345.png'
WHERE generation_id = 12345;

-- Upgrade user subscription
UPDATE users
SET subscription_tier = 'pro',
    updated_at = CURRENT_TIMESTAMP
WHERE user_id = 42;

-- Update model version per conversation
UPDATE conversations
SET model_version = 'gpt-4-turbo'
WHERE created_at > '2024-06-01' AND model_version = 'gpt-4';
```

⚠️ WARNING: Always use WHERE!

```
sql

-- BAD: Updates ALL rows!
UPDATE users SET subscription_tier = 'free';

-- GOOD: Updates specific user
UPDATE users SET subscription_tier = 'free' WHERE user_id = 99;
```

Parte 3: DELETE - Rimuovere Dati (1h)

DELETE syntax:

```
sql

DELETE FROM table_name
WHERE condition;
```

AI Examples:

```
sql
```

```
-- Delete failed generations older than 7 days
DELETE FROM ai_generations
WHERE status = 'failed'
  AND created_at < datetime('now', '-7 days');

-- Delete spam feedback
DELETE FROM generation_ratings
WHERE rating = 1 AND comment LIKE '%spam%';

-- Delete test users
DELETE FROM users
WHERE email LIKE '%@test.com';
```

CASCADE DELETE (via FK constraint):

```
sql

-- Deleting user also deletes their conversations and messages
DELETE FROM users WHERE user_id = 999;
-- FK ON DELETE CASCADE handles the rest
```

TRUNCATE (delete all, fast):

```
sql

-- SQLite doesn't have TRUNCATE, use:
DELETE FROM temp_generations;
VACUUM;  -- Reclaim space
```

⚠️ **Safety practices**:

1. Always use WHERE unless truly deleting all
2. Use transactions for important deletes
3. Backup before bulk deletes
4. Test with SELECT first:

```
sql

  -- First: see what you'll delete
  SELECT * FROM ai_generations WHERE status = 'failed';
  -- Then: actually delete
  DELETE FROM ai_generations WHERE status = 'failed';
```

Lab (1h):

Scenario-based exercises:

1. **Populate AI Chatbot Database**:
   - Insert 5 users
   - Insert 10 conversations (spread across users)
   - Insert 30 messages (mix of user/assistant/system)
2. **Update Operations**:
   - Mark pending generations as completed
   - Update user subscription tiers
   - Fix model version names
3. **Cleanup Operations**:
   - Delete failed generations older than 30 days
   - Remove inactive users (no conversations)
   - Clear test data

Challenge: Write a "data sanitization" script:

- Anonymize user emails (replace with "user_XXX@anonymized.com")
- Clear sensitive prompts containing "password" or "credit card"
- Archive old data before deletion

Lezione 4: ALTER TABLE e Schema Evolution (2 ore)

Parte 1: ALTER TABLE Operations (1h)

Add Column:

```
sql
```

```
-- Add feature flag
ALTER TABLE users
ADD COLUMN beta_features_enabled INTEGER DEFAULT 0;

-- Add embedding column (for vector search later)
ALTER TABLE prompts
ADD COLUMN embedding_vector TEXT;

-- Add usage tracking
ALTER TABLE conversations
ADD COLUMN total_tokens_used INTEGER DEFAULT 0;
```

Rename Column (SQLite 3.25+):

```
sql

ALTER TABLE ai_generations
RENAME COLUMN cfg_scale TO guidance_scale;
```

Drop Column (SQLite 3.35+, PostgreSQL):

```
sql

ALTER TABLE users
DROP COLUMN legacy_password_hash;
```

Rename Table:

```
sql

ALTER TABLE old_generations
RENAME TO archived_generations;
```

Parte 2: Schema Evolution in AI Applications (1h)

Common scenarios:

1. Adding Model Versioning:

```
sql
```

```
-- Initially: single model
CREATE TABLE conversations (id INTEGER, user_id INTEGER, ...);

-- Evolution: track model versions
ALTER TABLE conversations
ADD COLUMN model_version TEXT DEFAULT 'gpt-3.5-turbo';

ALTER TABLE conversations
ADD COLUMN model_provider TEXT DEFAULT 'openai';
```

2. Adding Analytics Columns:

```
sql

ALTER TABLE ai_generations
ADD COLUMN user_rating INTEGER CHECK(user_rating BETWEEN 1 AND 5);

ALTER TABLE ai_generations
ADD COLUMN reported_as_inappropriate INTEGER DEFAULT 0;

ALTER TABLE conversations
ADD COLUMN average_response_time REAL;
```

3. Migrating to Support New Features:

```
sql

-- Initially: simple prompts
CREATE TABLE prompts (id INTEGER, text TEXT, ...);

-- Add support for system prompts
ALTER TABLE prompts
ADD COLUMN system_prompt TEXT;

-- Add support for few-shot examples
ALTER TABLE prompts
ADD COLUMN example_conversations TEXT;  -- JSON array
```

Migration Strategy:

1. Add column with DEFAULT value (safe for existing data)
2. Backfill data if needed (UPDATE)
3. Make NOT NULL if required
4. Add indexes if needed for performance

Example migration:

```sql
-- Step 1: Add nullable column
ALTER TABLE messages ADD COLUMN token_count INTEGER;

-- Step 2: Backfill (estimate tokens for old messages)
UPDATE messages
SET token_count = length(content) / 4  -- rough estimate
WHERE token_count IS NULL;

-- Step 3: Make NOT NULL for future inserts
-- (SQLite limitation: can't add NOT NULL after creation easily)
```

Lab (30 min):

Evolution Exercise: Given initial schema for AI assistant, evolve it:

- Add support for voice mode (audio input/output)
- Add conversation sharing feature (share_token, is_public)
- Add content moderation flags
- Add cost tracking (cost_usd per generation)

WEEKEND PROJECT - Settimana 2

Deliverable: SQL Scripts + Populated Database

Task: Implement il database design della Settimana 1

Requirements:

1. CREATE TABLE statements con tutti i constraints
2. INSERT sample data (realistico per AI app):
   - Almeno 10 users
   - 30+ conversations/generations
   - 100+ messages/outputs
3. UPDATE statements (esempio: mark completed, update ratings)
4. DELETE statements (cleanup operations)
5. ALTER TABLE statements (evolve schema)

Bonus:

- Script di seed data automatico
- Comments in SQL spiegando scelte
- Sample data realistico (use ChatGPT per generare!)

Presentazione: Lunedì Settimana 3 - Live demo del database funzionante

---

SETTIMANA 3: SQL Queries - SELECT e Data Retrieval

Lezione 1: SELECT Basics e WHERE (3 ore)

Parte 1: SELECT Fundamentals (1h)

Basic SELECT:

sql

```sql
-- All columns
SELECT * FROM users;

-- Specific columns
SELECT user_id, email, username FROM users;

-- Column aliases
SELECT
  user_id AS id,
  email AS user_email,
  created_at AS registration_date
FROM users;

-- Calculated columns
SELECT
  prompt,
  length(prompt) AS prompt_length,
  length(prompt) * 0.75 AS estimated_tokens
FROM prompts;
```

AI Context Queries:

sql

```sql
-- Get all AI generations with metadata
SELECT
  generation_id,
  prompt,
  model_version,
  steps,
  cfg_scale,
  status,
  generation_time_seconds AS gen_time_sec
FROM ai_generations;

-- Get conversation with message count estimate
SELECT
  conversation_id,
  title,
  model_version,
  created_at,
  -- (actual count requires subquery, preview for later)
FROM conversations;
```

Parte 2: WHERE Clause - Filtering (1h)

Comparison operators:

sql

```sql
-- Equals
SELECT * FROM users WHERE subscription_tier = 'pro';

-- Not equals
SELECT * FROM ai_generations WHERE status != 'failed';

-- Numeric comparisons
SELECT * FROM ai_generations WHERE steps > 50;
SELECT * FROM users WHERE created_at >= '2024-01-01';

-- Text patterns (LIKE)
SELECT * FROM prompts WHERE prompt LIKE '%cat%';
SELECT * FROM users WHERE email LIKE '%@gmail.com';
SELECT * FROM prompts WHERE prompt LIKE 'a photo of%';
```

Logical operators:

sql

```
-- AND
SELECT * FROM users
WHERE subscription_tier = 'pro'
  AND created_at > '2024-01-01';

-- OR
SELECT * FROM ai_generations
WHERE status = 'failed' OR status = 'pending';

-- NOT
SELECT * FROM prompts
WHERE NOT (prompt LIKE '%nsfw%');

-- IN operator
SELECT * FROM ai_generations
WHERE model_version IN ('sd-xl', 'sd-2.1', 'midjourney-v6');

-- BETWEEN
SELECT * FROM ai_generations
WHERE steps BETWEEN 20 AND 80;
```

NULL handling:

```
sql

-- Find conversations without titles
SELECT * FROM conversations WHERE title IS NULL;

-- Find generations with completion time
SELECT * FROM ai_generations WHERE completed_at IS NOT NULL;
```

Parte 3: AI-Specific Query Patterns (1h)

Find problematic generations:

```
sql

SELECT * FROM ai_generations
WHERE status = 'failed'
  AND created_at > datetime('now', '-24 hours');
```

Find power users:

```
sql
```

```
-- Users with many generations (preview - needs aggregation)
SELECT user_id, email FROM users
WHERE user_id IN (
  SELECT user_id FROM ai_generations
  GROUP BY user_id
  HAVING COUNT(*) > 100
);
```

Find expensive prompts:

```
sql

SELECT prompt, steps, cfg_scale, generation_time_seconds
FROM ai_generations
WHERE generation_time_seconds > 30
ORDER BY generation_time_seconds DESC
LIMIT 10;
```

Content moderation queries:

```
sql

-- Flagged content
SELECT * FROM ai_generations
WHERE reported_as_inappropriate = 1;


-- Suspicious patterns
SELECT * FROM prompts
WHERE prompt LIKE '%violence%'
   OR prompt LIKE '%illegal%';
```

Lab (1h):

Query Writing Practice - AI Image Generator database:

1. Find all pro users registered in last 30 days
2. Find all failed generations from yesterday
3. Find all prompts containing "cyberpunk" or "sci-fi"
4. Find generations that took longer than 60 seconds
5. Find users with no generations yet
6. Find all NSFW-flagged content from last week

Challenge: Write queries for:

- Most common model versions used
- Average generation time by model
- Identify potential bot users (>100 generations per day)

Lezione 2: DISTINCT, LIMIT, ORDER BY (3 ore)

Parte 1: DISTINCT - Unique Values (45min)

Basic DISTINCT:

```
sql

-- All unique model versions used
SELECT DISTINCT model_version FROM ai_generations;

-- Unique subscription tiers
SELECT DISTINCT subscription_tier FROM users;

-- Unique combinations
SELECT DISTINCT model_version, steps
FROM ai_generations
ORDER BY model_version, steps;
```

AI Applications:

```
sql

-- What models have users tried?
SELECT DISTINCT model_version FROM conversations;

-- What are all the unique styles users requested?
SELECT DISTINCT style_preset FROM ai_generations
WHERE style_preset IS NOT NULL;

-- Unique error types
SELECT DISTINCT error_message FROM failed_generations;
```

COUNT DISTINCT:

```
sql
```

```
-- How many unique users have generated content?
SELECT COUNT(DISTINCT user_id) as active_users
FROM ai_generations;

-- How many different models are being used?
SELECT COUNT(DISTINCT model_version) as models_in_use
FROM conversations;
```

Parte 2: ORDER BY - Sorting (1h)

Basic sorting:

sql

```
-- Ascending (default)
SELECT * FROM users ORDER BY created_at;
SELECT * FROM users ORDER BY created_at ASC;

-- Descending
SELECT * FROM ai_generations
ORDER BY created_at DESC;

-- Multiple columns
SELECT * FROM users
ORDER BY subscription_tier DESC, created_at ASC;
```

AI Query Examples:

sql

```
-- Most recent generations
SELECT * FROM ai_generations
ORDER BY created_at DESC
LIMIT 10;

-- Slowest generations
SELECT prompt, generation_time_seconds
FROM ai_generations
WHERE status = 'completed'
ORDER BY generation_time_seconds DESC;

-- Best rated content
SELECT * FROM ai_generations
WHERE user_rating IS NOT NULL
ORDER BY user_rating DESC, created_at DESC;

-- Alphabetical prompt listing
SELECT DISTINCT prompt FROM ai_generations
ORDER BY prompt;
```

NULL ordering:

```
sql

-- SQLite: NULLs come first in ASC, last in DESC
SELECT * FROM conversations
ORDER BY title;  -- NULL titles first

-- Force NULLs last (SQLite trick)
SELECT * FROM conversations
ORDER BY title IS NULL, title;
```

Parte 3: LIMIT e OFFSET - Pagination (1h15)

LIMIT basics:

```
sql
```

```
-- Top 10 most recent
SELECT * FROM ai_generations
ORDER BY created_at DESC
LIMIT 10;

-- Just one (latest)
SELECT * FROM conversations
ORDER BY created_at DESC
LIMIT 1;
```

OFFSET for pagination:

```
sql

-- Page 1 (first 20)
SELECT * FROM ai_generations
ORDER BY created_at DESC
LIMIT 20 OFFSET 0;

-- Page 2 (next 20)
SELECT * FROM ai_generations
ORDER BY created_at DESC
LIMIT 20 OFFSET 20;

-- Page 3
SELECT * FROM ai_generations
ORDER BY created_at DESC
LIMIT 20 OFFSET 40;

-- General formula: LIMIT page_size OFFSET (page_number - 1) * page_size
```

AI Application - Infinite Scroll:

```
sql
```

```
-- Initial load: 30 generations
SELECT generation_id, prompt, image_path, user_rating
FROM ai_generations
WHERE status = 'completed'
ORDER BY created_at DESC
LIMIT 30;

-- Load more (next 30)
SELECT generation_id, prompt, image_path, user_rating
FROM ai_generations
WHERE status = 'completed'
ORDER BY created_at DESC
LIMIT 30 OFFSET 30;
```

Performance consideration:

- Large OFFSETs are slow (database still scans all rows)
- Better: cursor-based pagination

```
sql

 -- Instead of OFFSET, use last seen ID
 SELECT * FROM ai_generations
 WHERE generation_id < 12345  -- last seen
 ORDER BY generation_id DESC
 LIMIT 30;
```

Lab (1h):

Analytics Queries:

1. Top 10 most active users (by generation count - hint: needs COUNT)
2. List all unique model versions, sorte