BRITISH AI Lecture Intelligence System (v4.0 - TOTAL EXTRACTION)

1. THE LECTURER'S "GOLDEN RULES" (The Unspoken Marking Scheme)

- **The "Ambiguity" Rule:** Do not try to define "Big Data," "AI," or "NoSQL" perfectly.
  - *Quote:* "Spoiler alert a lot of my courses are not going to have very well defined terms... It's a bit of a theme of everything I teach."
  - *Application:* In your report, acknowledge that definitions are fluid. Don't present one definition as absolute truth.

- **The "Paper Hater" Rule:** You must view paper-based/legacy systems as the enemy ("Dinosaurs").
    - *Evidence:* She mocks the NHS for missing her second child in her notes due to paper silos.
    - *Application:* If asked to justify a database, contrast it against the failures of paper/siloed systems.
- **The "Visual" Rule:** She loves the visual aspect of Graph DBs.
    - *Instruction:* "Please do grab the nodes, move them around, look at those relationships... see how they join."
    - *Application:* Include screenshots of your graph visualizations in your coursework.

---

2. PART A: THEORY & CONCEPTS (Line-by-Line Extraction)

Big Data & NoSQL Definitions

- **Big Data:** Ill-defined. Some say it's about parallel computing; others say it's data size.
    - *The "USB Stick" Test:* "We don't laugh at people when they come to you and go, I've got six USB sticks full of data, that's big data. Then it is."
- **NoSQL Origins:**
    - Arrived by "accident" (Twitter hashtag 2009).
    - **Not** a clever acronym. Later retrofitted to mean "Not Only SQL".
    - **4 Features of NoSQL:**
        1. Non-relational (rejects tables/rows).
        2. No fixed schema (schema-less).
        3. Open Source / Customer Friendly (distributed easily).
        4. 21st Century Web (User-generated content).

Big Data Management Challenges

- **The 5 V's:** Mentioned but dismissed as "ill-defined."

- **Silos & Duplication:** "As soon as any organization gets past a certain point, there is horrible duplication."
    - *Risk:* Different levels of security in different silos.
- **Performance/Speed:**
    - *Example:* Poole Harbour project (AIS data). 800,000 entries take 2-3 minutes to load a chart. "Large data, you're going to have a slower system."
- **Quality & Trust:**
    - *Example:* Deep fakes. "How do you know that one of the videos you've got isn't a deep fake?"
    - *Sensors:* You can't manually monitor 10 sensors sending data every second.
- **Manpower:** Need for experts grows with data volume. (Good for your job prospects).

The "Dinosaurs" (Legacy Systems)

- **NHS Example (The Gallbladder Story):**
    - She had gallbladder surgery.
    - Hospital notes didn't list her second child (born at home).
    - They couldn't link her anaesthetic history to the birth.
    - *Verdict:* "Yay for the NHS and healthcare... It is scary."
    - *Lesson:* **Bad data = Bad decisions.**

Processing Paradigms (Coursework Vital)

- **Batch Processing:**
    - *Definition:* Collect data over time, send off in a group.
    - *Use Case:* Non-time critical evaluations.
    - *Coursework:* **"Essentially this is what you will be doing in your coursework."**
- **Stream Processing:**
    - *Definition:* Flows straight into the database.
    - *Use Case:* Time-critical (e.g., ER rooms).

Storage Types (The "Toolbox")

- **Distributed File Systems:** Stored across servers. Good for access, maybe bad for security.
- **Data Warehouses:** "Houses for data." Cleaning/Analytics happens here. High carbon footprint.
- **Object Storage:** Inspired by OOP. Data treated as a unit/object. Good for unstructured data.
- **NoSQL:** "Let's get loose and fast with data and store it however the hell we want."

Graph Databases (Neo4j)

- **Why She Loves It:** She has a **Mathematical Background** (Graph Theory).
- **Core Concept:** Store data based on **connections** , not content.
- **The "Ants" Analogy:** "Think of like little ants... following those paths." (Index-free adjacency).
- **Specific Use Cases:**
  - **Supply Chain:** (Sister's job example). Boss sees totals; Sister sees lead times/chains.
  - **Social Networks:** Twitter algorithms, "Who follows who."
  - **Recommender Systems:** Netflix, Amazon ("Scrolly bars at the bottom").
  - **Fraud/Cyber:** Spotting attack patterns.
- **When NOT to use Graph DBs:**
  - Heavy aggregations (Summing millions of rows).
  - Simple patterns.
  - *Analogy:* "Do you take the nutcracker or do you take the sledgehammer?" (Graph DB = Sledgehammer. Don't use it for a nut).

---

3. PART B: PRACTICAL CODING & EXECUTION (The "How-To")

1. The "Import Folder" Workflow (Mandatory)

She mandates this specific workflow to avoid file path errors.

1. **Do not start the database.**

2. Hover over the DB -> Click three dots -> **Open Folder** -> **Import** .
3. Paste CSV files there.
4. **Then** start the database.
5. *Why?* "I have a habit of starting my database, trying to import my data and it goes, I don't know what you're on about."

2. Loading Data: The "Simple" Method (Small Data)

- **Function:** `LOAD CSV WITH HEADERS`.
- **Protocol:**
  - **Protocol 1:** Use `file:///` (Three slashes).
  - **Protocol 2:** Explicit extensions (`.csv`). "Behave like a stubborn toddler."
  - **Protocol 3:** Case Sensitivity. Check your CSV headers.
- **Verification (The Math Check):**
  - Check the result summary.
  - Formula: `(Number of Rows) * (Number of Columns Imported) = (Total Properties Set)`.
  - If the math adds up, the load was successful.

3. Loading Data: The "Transaction" Method (Large Data)

- **Context:** For the "Netflix" dataset or anything large.
- **Reason:** Memory limits on Neo4j.
- **Command:** `CALL { ... } IN TRANSACTIONS OF X ROWS`.
- **The "Auto" Prefix:** If using the Desktop version, you likely need to prepend `:auto`.
  - *Quote:* "In the version I've got I need auto... that might not be true for the versions you're using."

4. Indexes (The Coursework Trap)

- **What they are:** Speed up searches but take space.
- **HER WARNING:** "If you decide to have indexes, you must use them in your queries. **You will be docked marks if you make an index and you don't use it.** "

- **Her Advice:** "Do not worry about indexes in your coursework."

---

4. PART C: COURSEWORK & EXAM INTEL (Hidden Hints)

The "Eurovision" Hint

She explicitly mentioned a dataset regarding **Eurovision** .

- **Data Structure:**
  - File 1: Host Countries.
  - File 2: Voting Results.
  - File 3: Winners.
- **The Task:** "I want you to find out... Who gives lots of votes to other countries quite commonly? You know, the whole **Greece-Cyprus** thing."
- **Action Required:** You need to learn how to query **Reciprocal Relationships** .

The "Twitter" Hint

- **Student Question:** Can you link tweets to hashtags?
- **Her Answer:** Yes. Do not store hashtags as text. Store them as **Nodes** .
- **Structure:** `(User)-[:TWEETED]->(Tweet)-[:TAGGED]->(Hashtag)`.

Privacy & Ethics (The "Pet Peeve")

- **Smart Meters:** She refuses to have one because she knows what companies do with the data.
- **Coursework Tip:** If writing a report, include a section on **Data Privacy/Ethics** . Mention that "just because we can collect it, doesn't mean we should." She will love this.

---

5. EXACT CYPHER CODE TEMPLATES (Based on her Lecture)

Here is the exact code she wants you to use, corrected for the errors she made during the live demo.

1. The "Clean" Load (Use for small files)

Cypher

```
// NOTE: Ensure file is in 'Import' folder first.
LOAD CSV WITH HEADERS FROM "file:///games.csv" AS csvLine
CREATE (g:Game {
    title: csvLine.title,   // Case sensitive!
    sales: toInteger(csvLine.sales) // Convert to number if needed
});
```

2. The "Safe" Load (Use for Coursework Main Data)

Cypher

```
// NOTE: Use :auto if on Desktop
:auto
LOAD CSV WITH HEADERS FROM "file:///netflix_titles.csv" AS row
CALL {
    WITH row
    // She explicitly mentioned repeating the variable "row" here
    CREATE (m:Movie {
        id: row.show_id,
        title: row.title,
        type: row.type
    })
} IN TRANSACTIONS OF 500 ROWS; // She used 250, but said "pick a nice
number"
```

3. The "Greece-Cyprus" Query (PREEMPTIVE STRIKE)

She hinted this is the task. Here is how you solve the "Reciprocal Voting" problem she mentioned:

Cypher

```
// Find countries that voted for each other (The Greece-Cyprus pattern)
MATCH (c1:Country)-[v1:VOTED_FOR]->(c2:Country)
MATCH (c2)-[v2:VOTED_FOR]->(c1)
WHERE v1.points > 10 AND v2.points > 10 // Assuming "lots of votes" means
high points
RETURN c1.name, c2.name, v1.points, v2.points
```