Data 101/CS 187: Data Engineering Fall 2026 Discussion 7 – October 23rd, 2026

Entity Resolution, ER Diagram, Normalization, Hampel X84

Instructor: Aditya Parameswaran *Covers Lecture(s): 13 – 16*

I Entity Resolution

In this section, we will work with an album sales dataset. Here is a sneak peek:

name	artist	track_count
Red (Deluxe Edition)	Taylor Swift	22
thank u, next	Ariana Grande	12
Bloom	Red Velvet	11
After LIKE	IVE	2

albums

name	day	count
After LIKE	1	2
After Like	2	-1
After like (cd)	4	7
After LIKE - ive	6	7
Red - Deluxe	5	-1

sales

One problem with this dataset is that the album names are not standardized. For example, "After Like", "After LIKE", "After Like (CD)" all refer to the same album, but their string representations are not exactly the same.

1. Write a query to join the *albums* table with the *sales* using strict equality condition. What is going to go wrong with this approach?

```
SELECT a.name, artist, day, count
FROM albums AS a INNER JOIN sales AS s ON a.name = s.name;
```

Problem: Only rows whose name exactly matches the album name (i.e., "After LIKE") will appear in the join result, while all other variants will get dropped

2. One solution is to use the Levenshtein distance. The `levenshtein()` distance function can be helpful to find strings that are similar but not identical. This algorithm computes the minimal number of insertions, deletions, and mutations. Let us apply this to our dataset. Write a query to compute the Levenshtein distance between every pair of names in the two tables, sorted by this distance.

```
SELECT a.name, s.name, levenshtein(a.name, s.name) AS distance
FROM albums AS a, sales AS s
ORDER BY levenshtein(a.name, s.name) ASC
```

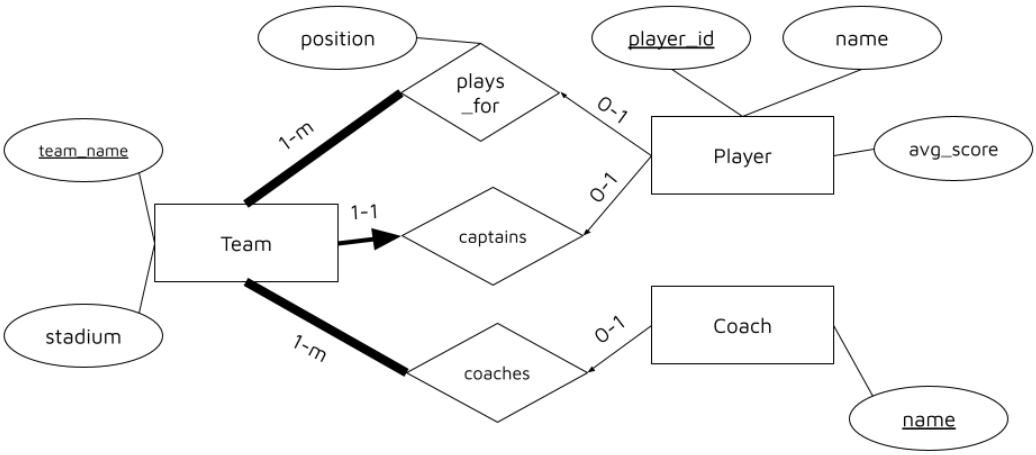
3. Create a materialized view `joined_sales` that joins the tables on the condition that two entries match if their Levenshtein distance is less than 10.

```
CREATE MATERIALIZED VIEW joined_sales AS (
 SELECT a.name, artist, day, count
 FROM albums AS a INNER JOIN sales AS s
 ON levenshtein(a.name, s.name) < 10
)
```

II Entity Relationship Diagrams

4. We want to store sports teams and their players in our database. Draw an ER diagram corresponding to data given below:

   - Every **team** has a unique team_name and a stadium where they play.
   - Each **coach** has a unique name.
   - Each **player** has a unique player_id, a name and an average score.
   - A **player** *plays for* a **team**. This relationship also records the position of the player. A player can play at most one team, but may also be a free agent and not play on any team. A team needs at least 1 player.
   - A **player** can *captain* at most one **team**. A team has exactly one captain.
   - A **coach** can *coach* at most one **team**. A team may have many coaches, and needs at least 1 coach.



5. Translate the ER diagram into a relational schema.

```
CREATE TABLE Teams (
team_name TEXT PRIMARY KEY,
stadium TEXT,
captain_id TEXT,
FOREIGN KEY (captain_id) REFERENCES Players(player_id)
);

CREATE TABLE Players (
player_id TEXT PRIMARY KEY,
name TEXT,
avg_score FLOAT,
position TEXT,
team_name TEXT,
```

```
FOREIGN KEY (team_name) REFERENCES Teams(team_name)
);

CREATE TABLE Coaches (
name TEXT PRIMARY KEY,
team_name TEXT,
FOREIGN KEY (team_name) REFERENCES Teams(team_name)
);
```

III Functional Dependency and Normalization

A functional dependency (FD) is a form of constraint between 2 sets of attributes in a relation.

- Two attributes satisfy $X \to Y$ if and only if for every pair of tuples (t, u), the following holds: if $t.X = u.X$, then $t.Y = u.Y$.
- Three attributes satisfy $XY \to Z$ if and only if for every pair of tuples (t, u), the following holds: if $t.X = u.X$ and $t.Y = u.Y$, then $t.Z = u.Z$.

For this section, consider the student_course_info table.

student_id	**student_name**	**course_id**	**course_name**	**professor**
101	Alice	C001	Math	Dr. Smith
102	Bob	C001	Math	Dr. Smith
103	Charlie	C002	English	Dr. Brown
104	Dave	C002	English	Dr. Brown
105	Eve	C003	History	Dr. Clark

Table 1: student_course_info

6. Write down three functional dependencies that **may hold** based on the schema and the sample data.

(a) **student_id** $\to$ **student_name**

(b) **course_id** $\to$ **course_name**, **professor**

(c) **student_id**, **course_id** $\to$ **professor**

7. Perform normalization on this table. Your result should have 3 tables: `students`, `courses`, `enrollments`.

Table 1: students

student_id	student_name
101	Alice
102	Bob
103	Charlie
104	Dave
105	Eve

`student_id` uniquely determines `student_name`, and `student_id` is the primary key.

Table 2: courses

course_id	course_name	professor
C001	Math	Dr. Smith
C002	English	Dr. Brown
C003	History	Dr. Clark

`course_id` uniquely determines both `course_name` and `professor`, so `course_id` is set as the primary key. This structure avoids repeating the course name and `professor` information across multiple rows.

Table 3: enrollments

student_id	course_id
101	C001
102	C001
103	C002
104	C002
105	C003

The `enrollments` table is created to represent the many-to-many relationship between students and courses. Each (`student_id`, `course_id`) pair identifies one enrollment, and together they form the primary key of this table. Both `student_id` and `course_id` are foreign keys referencing the primary keys in `students` and `courses` respectively. Including the `professor` column in the enrollment table would be redundant because `professor` information for each course is already stored in the course table. If we want to find the `professor` for a specific enrollment, we can perform a join.

8. Suppose we know that $X \to Y$ and $Y \to Z$. Does $X \to Z$ hold? Prove your answer using contradiction.

To prove that $X \to Z$ holds, we can use a proof by contradiction.

Assume that $X \to Z$ does *not* hold. That means there exist two tuples t and u such that:

- $t.X = u.X$ (they agree on X),
- but $t.Z \neq u.Z$ (they disagree on Z).

From $X \to Y$, since $t.X = u.X$, it must follow that $t.Y = u.Y$.

Then from $Y \to Z$, since $t.Y = u.Y$, it must follow that $t.Z = u.Z$.

This contradicts our assumption that $t.Z \neq u.Z$.

Therefore, our assumption must be false, and we conclude that $X \to Z$ holds.

IV [Optional] Hampel X84

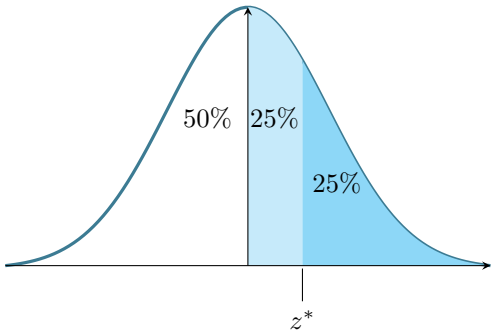
Hampel X84 is a robust outlier detection technique because it has an ideal breakdown point of 50% (the proof of the breakdown point is beyond the scope of this course).

9. In this exercise, we will prove the following statement: selecting outliers 1.4826 MAD away from the median is equivalent to selecting outliers 1 standard deviation away from the mean, if the data follows a normal distribution.

   (a) Write down the definition of Median Absolute Deviation (MAD). What is the intuitive meaning of MAD?

> MAD = median($|X_i - \text{median}(X)|$).
> Intuitively, this is the median distance between any data point and the median (center) value.

   (b) Let us try to derive the magic number 1.4826. First, let us find the MAD of a standard normal distribution. Let Z be a random variable following a standard normal distribution with $\mu = 0$ and $\sigma = 1$, find z^* such that $P(Z < z^*) = 0.75$. Is z^* the MAD? Hint: Use `scipy.stats.norm`.




> ```
> from scipy.stats import norm
> >>> norm.ppf(0.75)
> 0.6744897501960817
> ```
> Why does this give the MAD? First, since median(Z) = 0, MAD = median($|Z|$). Next, since the standard normal distribution is symmetric around $Z = 0$, median($|Z|$) equals the median of Z restricted to $Z > 0$. As shown in the plot, $P(0 < Z < z^*) = P(Z > z^*) = 0.25$, so z^* splits the $Z > 0$ half evenly. Hence $z^* = median(|Z|)$, i.e., z^* is the MAD As shown in the plot, for all $Z > 0$, there are exactly as many $Z < z^*$ as many $Z > z^*$. Hence $z^* = \text{median}(|Z|)$, hence z^* is the MAD.

   (c) Continuing the derivation of 1.4826, what is the mathematical relation between the MAD and the standard deviation in a normal distribution?

> Since standard normal distribution has $\sigma = 1$, and MAD = 0.6745, for any other normal distribution with standard deviation σ, MAD = $\sigma * 0.6745$, or, σ = 1.4826MAD. This concludes our proof.

10. Now let us apply the Hampel X84 method on the following dataset to detect outliers: $\{2, 4, 6, 8, 9, 11, 30\}$.

   (a) Compute the median.

The median is 8.

(b) Compute the MAD.

First compute the distances: $|2-8|, |4-8|, |6-8|, |8-8|, |9-8|, |11-8|, |30-8| = 6, 4, 2, 0, 1, 3, 22$. Then sort them: $0, 1, 2, 3, 4, 6, 22$. Hence the MAD is 3.

(c) Find the outliers using the Hampel X84 method.

First calculate the radius: $2 \times 1.4826 \times 3 = 8.8956$.

Then calculate the bounds: lower bound $8 - 8.8956 = -0.8956$, upper bound $8 + 8.8956 = 16.8956$.

30 is outside of the bounds, so it is the outlier.