# Low-Level Design (LLD) – Student Exam Performance Analyzer (PySpark)

**Difficulty Level**: Easy | **Total Marks**: 20
**Standards Followed**: 6 Functions | 6 Visible Test Cases

---

### Summary of Design Requirements

- Load CSV files using `spark.read.csv`
- Join student and scores datasets
- Apply groupBy and aggregation for average and top scorers
- Filter students who missed exams
- Sort results using `orderBy()`
- Output clean values with correct schema and types

---

### Concepts Tested

- Reading CSV files with `header=True` and `inferSchema=True`
- Joining DataFrames on common keys
- Grouping and aggregating with `.groupBy().agg()`
- Sorting with `.orderBy()`
- Filtering with `.isNull()` or `.isNotNull()`
- Calculating max/avg using `F.max`, `F.avg`

---

# Problem Statement

You're given two CSV files containing student information and their exam scores:

- `students.csv` – contains `student_id`, `name`, `class`
- `scores.csv` – contains `student_id`, `subject`, `score`

You must analyze exam performance using PySpark to answer key queries about student participation, performance, and top scoring.

---

### Operations

---

### 1. Load Data

Load both CSVs (`students.csv` and `scores.csv`) into DataFrames.

**Function Prototype:**

```
python
CopyEdit
def load_data(students_path: str, scores_path: str) -> tuple:
```

Output: Tuple – `(students_df, scores_df)`

Use: `spark.read.csv(path, header=True, inferSchema=True)`

---

### 2. Join DataFrames

Inner join on `student_id` between students and scores.

**Function Prototype:**

```
def join_data(students_df, scores_df) -> DataFrame:
```

Output: joined DataFrame

Use `.join()` with `how="inner"`

---

### 3. Top Scorer per Subject

Return student with highest score in each subject.

**Function Prototype:**

```
def top_scorers_by_subject(df) -> DataFrame:
```

Output: DataFrame with `subject`, `student_id`, `score`

Use: `groupBy("subject").agg(F.max("score"))`
Then join back to original to get full student info

---

**4. Average Score per Class**

Return class-wise average score.

**Function Prototype:**

```
def average_score_by_class(df) -> DataFrame:
```

Output: DataFrame with `class`, `avg_score`

Use: `groupBy("class").agg(F.avg("score"))`

---

**5. Students Missing Scores**

List students who didn't take any exam.

**Function Prototype:**

```
def students_with_no_scores(students_df, scores_df) -> DataFrame:
```

Output: DataFrame with only `student_id`, `name`, `class`

Use: Left join and filter with `.isNull()`

---

**6. Highest Average Scorer**

Return student with highest overall average.

**Function Prototype:**

```
def highest_average_scorer(df) -> Row:
```

Output: Single Row or dict with student info

Use: `groupBy("student_id").agg(avg)` + `orderBy(desc)` + `.first()`

---

# Test Cases & Marks Allocation

| Test Case ID | Description | Function | Marks |
|---|---|---|---|
| TC1 | Load both CSVs | `load_data()` |  3 |

| Test Case ID | Description | Function | Marks |
|---|---|---|---|
| TC2 | Join datasets | `join_data()` | □ 3 |
| TC3 | Top scorer per subject | `top_scorers_by_subject()` | □ 4 |
| TC4 | Class-wise average score | `average_score_by_class()` | □ 3 |
| TC5 | Students who missed exams | `students_with_no_scores()` | □ 3 |
| TC6 | Highest average scorer | `highest_average_scorer()` | □ 4 |

**Total Marks: 20**

---

# Visible Test Cases (6)

- □ TC1: Valid paths to student and scores CSVs, should return valid DataFrames
- □ TC2: Join operation must return rows only where student_id exists in both
- □ TC3: Validate top scorer per subject matches correct ID and score
- □ TC4: Validate class average with correct rounding
- □ TC5: Return all students who don't appear in scores.csv
- □ TC6: Return student with highest average across all subjects