

PROJECT

# Complex Filters & Projections

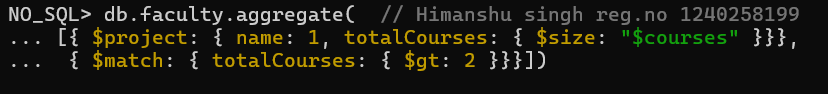
**Q1: -** List the names and departments of students who have more than 85% attendance and are s skilled inboth "MongoDB" and "Python".

## Query: -

db.students.find(

{ attendance: { $gt: 85 }, skills: { $in: ["MongoDB", "Python"] }})

## Output: -



▪ Nothing will show up because there aren’t any students who have both ‘MongoDB’ and ‘Python’ skills and more than 85% attendance.

* Use **comparison operators** like $gt (greater than).

* Apply **array matching** with $all to ensure multiple elements exist.

* Use **projection** to show only required fields.

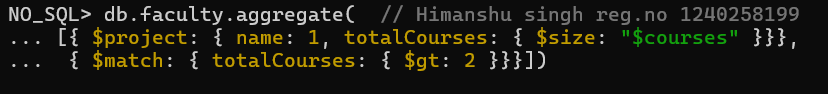
* Build **compound filters** using multiple conditions.

**Q2: -** Show all faculty who are teaching more than 2 courses. Display their names and the total number of courses they teach.

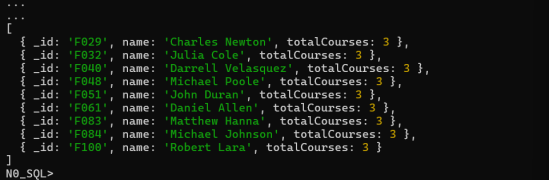
## Query: -

db.faculty.aggregate(

[{ $project: { name: 1, totalCourses: { $size: "$courses" }}}, { $match: { totalCourses: { $gt: 2 }}}])



## Output: -



* Use $project to create computed fields.

* Use $size to count array elements.

* Combine $match after projection for conditional filtering.

* Understand aggregation pipelines.

# Joins ($lookup) and Aggregations

**Q3: -** Write a query to show each student’s name along with the course titles they are enrolled in (use $lookup between enrollments, students, and courses).

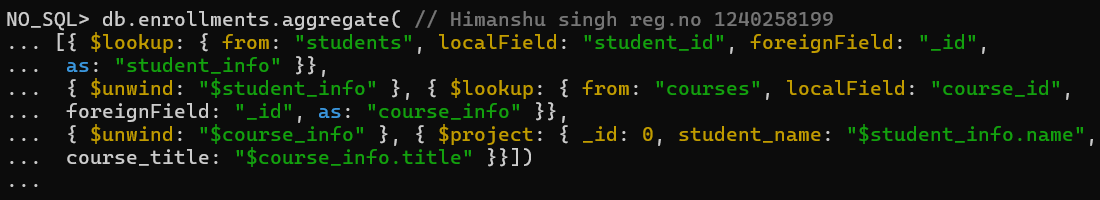
## Query: -

db.enrollments.aggregate(

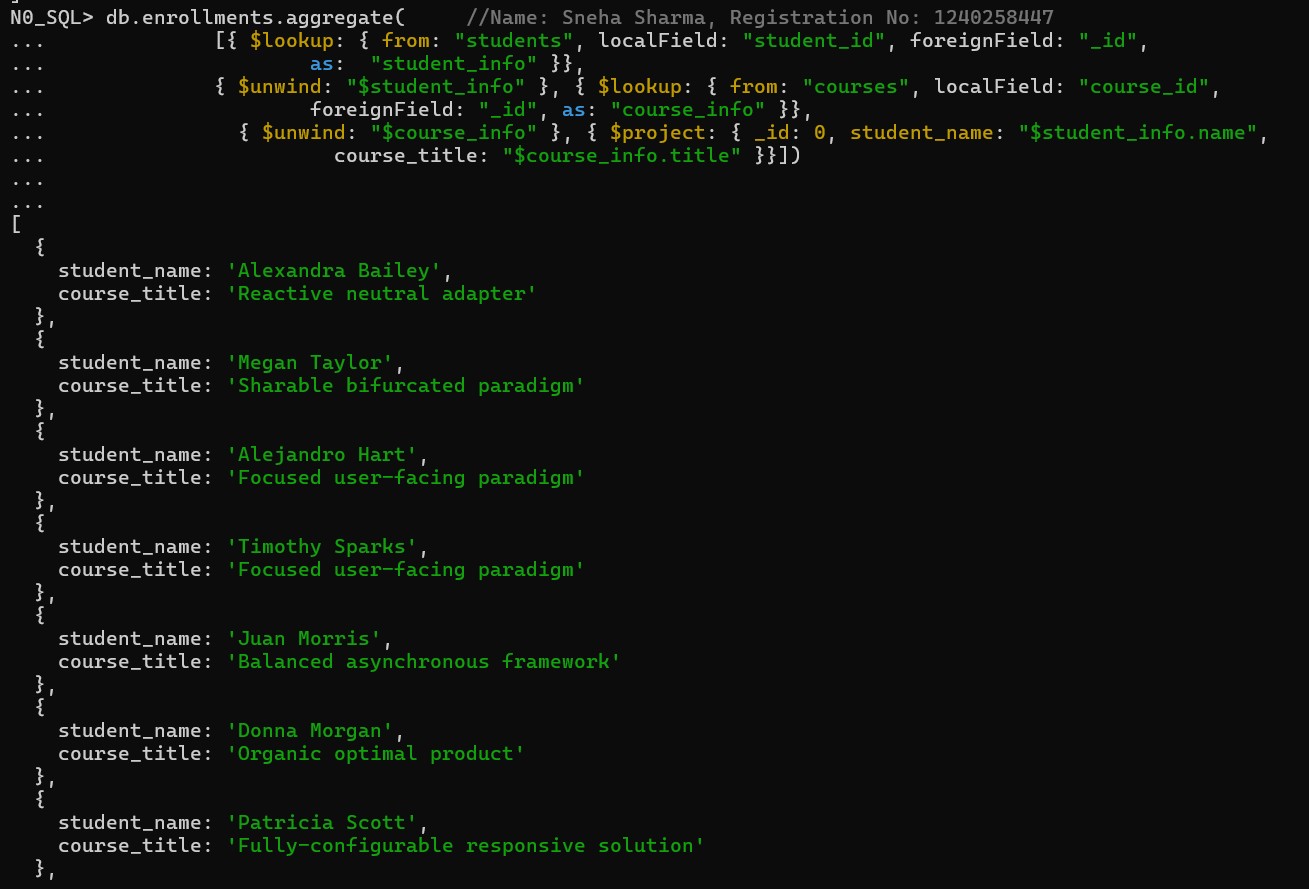
[{ $lookup: { from: "students", localField: "student\_id", foreignField: "\_id", as: "student\_info" }},

{ $unwind: "$student\_info" }, { $lookup: { from: "courses", localField: "course\_id", foreignField: "\_id", as: "course\_info" }},

{ $unwind: "$course\_info" }, { $project: { \_id: 0, student\_name: "$student\_info.name", course\_title: "$course\_info.title" }}])

****

## Output:-



* Use $lookup for joins between collections.

* Combine multiple $lookups for complex relationships.

* Use $arrayElemAt to extract single values from arrays.

* Understand MongoDB’s relational-like linking.

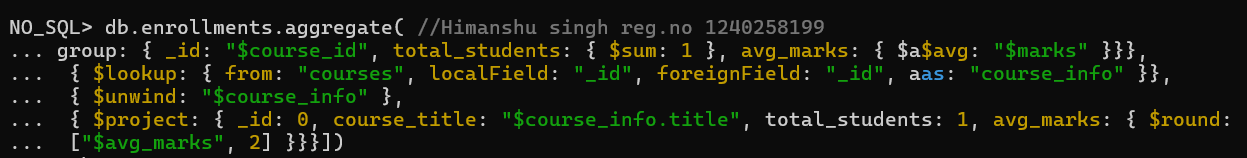
**Q4: -** For each course, display the course title, number of students enrolled, and average marks (use $group).

## Query: -

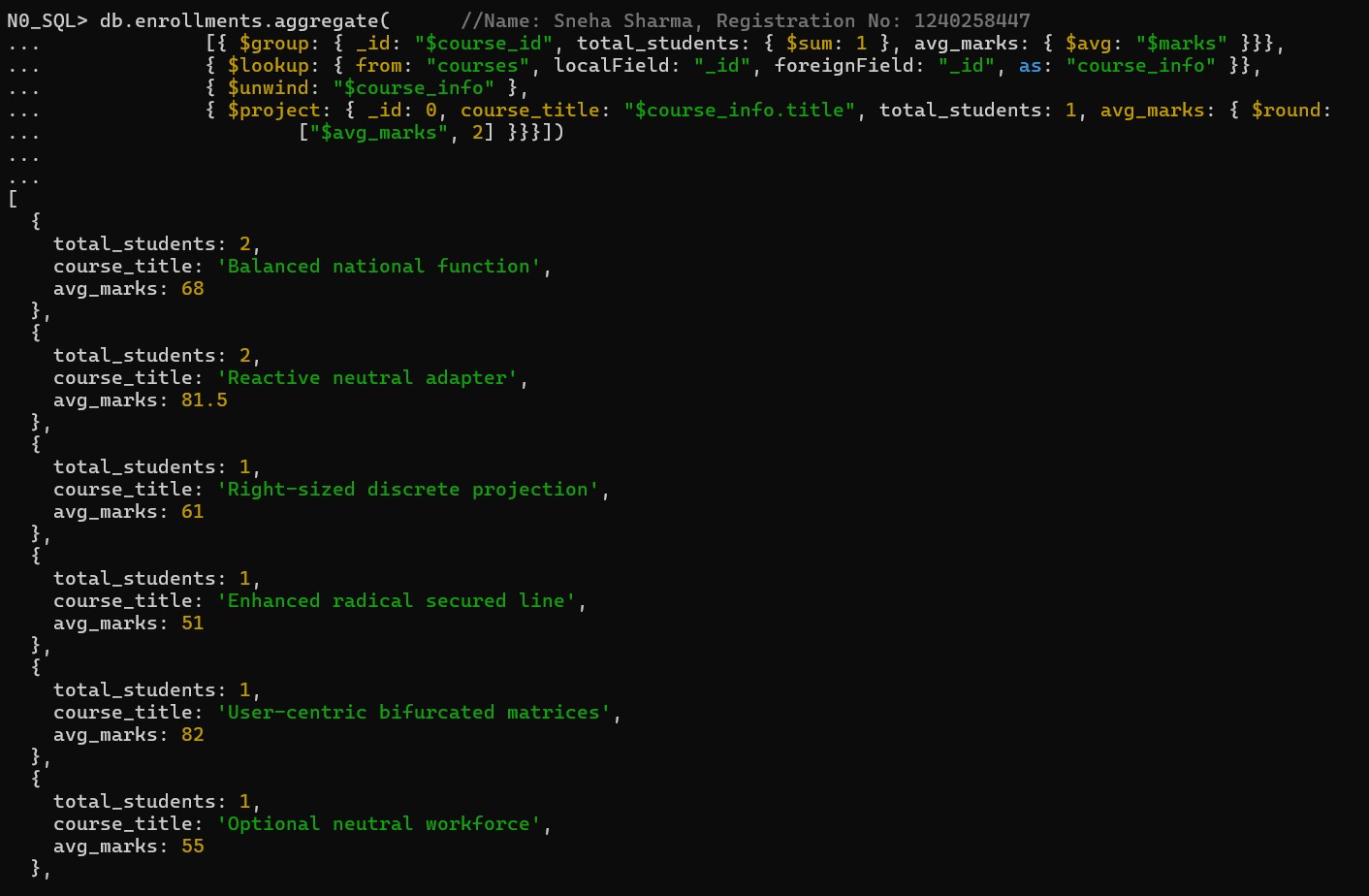
db.enrollments.aggregate(

[{ $group: { \_id: "$course\_id", total\_students: { $sum: 1 }, avg\_marks: { $avg: "$marks" }}}, { $lookup: { from: "courses", localField: "\_id", foreignField: "\_id", as: "course\_info" }}, { $unwind: "$course\_info" },

{ $project: { \_id: 0, course\_title: "$course\_info.title", total\_students: 1, avg\_marks: { $round: ["$avg\_marks", 2] }}}])



## Output:-



* Use $group for summarizing data.

* Use $avg and $sum to calculate aggregates.

* $unwind helps to deconstruct arrays.

* $project to rename and structure output.

# Grouping, Sorting, and Limiting

**Q5: -** Find the top 3 students with the highest average marks across all enrolled courses.

## Query: -

db.enrollments.aggregate(

[{ $group: { \_id: "$student\_id", avg\_marks: { $avg: "$marks" } } }, { $sort: { avg\_marks: -1 } },

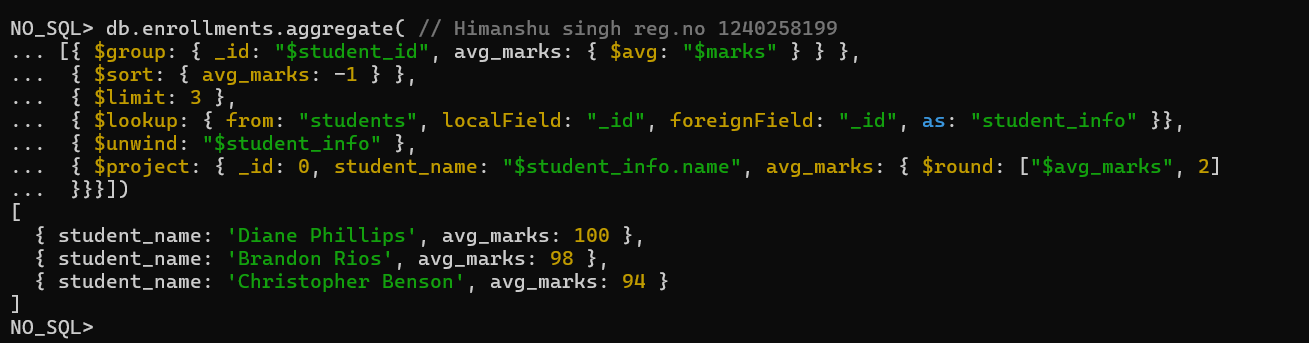
{ $limit: 3 },

{ $lookup: { from: "students", localField: "\_id", foreignField: "\_id", as: "student\_info" }}, { $unwind: "$student\_info" },

{ $project: { \_id: 0, student\_name: "$student\_info.name", avg\_marks: { $round: ["$avg\_marks", 2]

}}}])

## Output:-



* $sort sorts data in ascending/descending order.

* $limit restricts results to top records.

* $group for calculating averages.

* Combining joins with grouping.

**Q6: -** Count how many students are in each department. Display the department with the highest number of students.

## Query: -

db.students.aggregate(

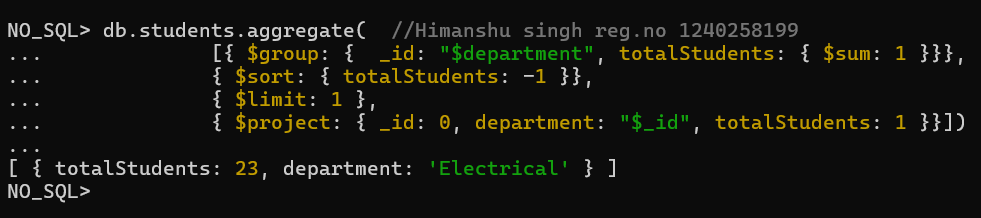
[{ $group: { \_id: "$department", totalStudents: { $sum: 1 }}},

{ $sort: { totalStudents: -1 }},

{ $limit: 1 },

{ $project: { \_id: 0, department: "$\_id", totalStudents: 1 }}])

## Output:-



* Count items per category with $sum: 1.

* Use $sort to rank results.

* Identify top-performing or most populated groups.

* Apply $limit to get top results.

# Update, Upsert, and Delete

**Q7: -** Update attendance to 100% for all students who won any "Hackathon".

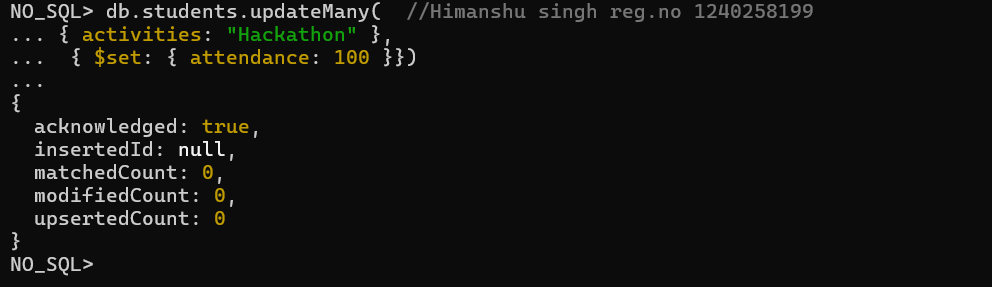
## Query: -

db.students.updateMany(

{ activities: "Hackathon" },

{ $set: { attendance: 100 }})

## Output:-



* Use updateMany() for bulk updates.

* $set modifies specific fields.

* Target documents via **nested fields**.

* Understand bulk updates with filters.

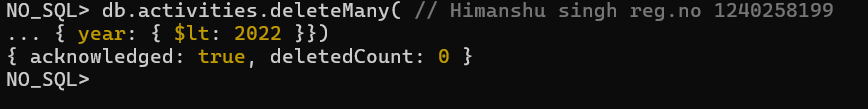
**Q8: -** Delete all student activity records where the activity year is before 2022.

## Query: -

db.activities.deleteMany(

{ year: { $lt: 2022 }})

## Output:-



* $lt filters by less than a value.

* Delete records conditionally using deleteMany().

* Manage dataset cleanup.

* Apply conditional data management

**Q9: -** Upsert a course record for "Data Structures" with ID "C150" and credits 4—if it doesn’t exist, insert it; otherwise update its title to "Advanced Data Structures".

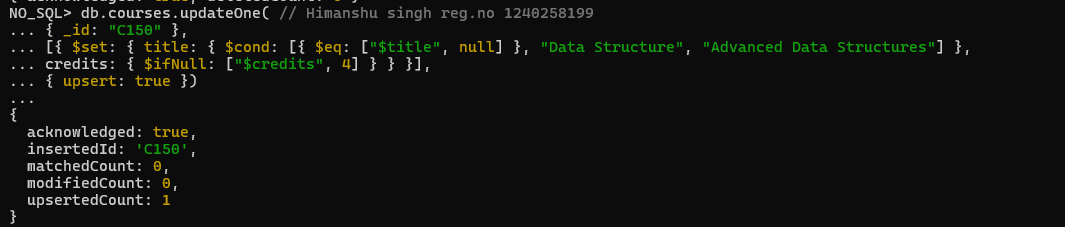
## Query: -

db.courses.updateOne( { \_id: "C150" },

[{ $set: { title: { $cond: [{ $eq: ["$title", null] }, "Data Structure", "Advanced Data Structures"] }, credits: { $ifNull: ["$credits", 4] } } }],

{ upsert: true })

## Output:-



* upsert: true inserts if no match is found.

* $setOnInsert applies only when inserting new data.

* $set updates fields if record exists.

* Handle both **insert and update** in one command.

# Array & Operator Usage

**Q10: -** Find all students who have "Python" as a skill but not "C++".

## Query: -

db.students.find(

{ $and: [{ skills: "Python" }, { skills: { $ne: "C++" }}]})

## Output:-



* $in checks for presence in arrays.

* $nin checks for absence in arrays.

* Combine both for exclusive conditions.

* Operate effectively on array fields.

**Q11: -** Return names of students who participated in "Seminar" and "Hackathon" both.

## Query: -

db.activities.aggregate(

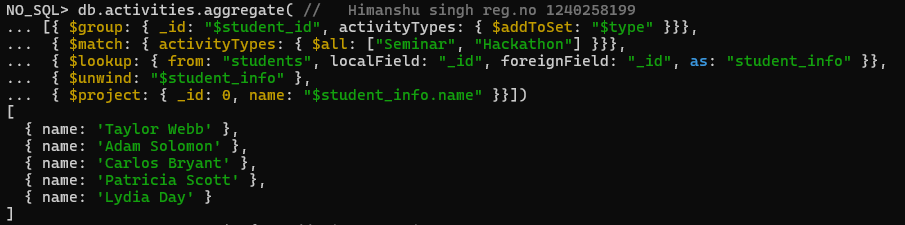
[{ $group: { \_id: "$student\_id", activityTypes: { $addToSet: "$type" }}},

{ $match: { activityTypes: { $all: ["Seminar", "Hackathon"] }}},

{ $lookup: { from: "students", localField: "\_id", foreignField: "\_id", as: "student\_info" }}, { $unwind: "$student\_info" },

{ $project: { \_id: 0, name: "$student\_info.name" }}])

## Output:-



* $all ensures all specified elements exist in an array.

* Simple array querying in MongoDB.

* Combine multiple filters in a single query.

* Efficient participation tracking.

# Subdocuments and Nested Conditions

**Q12: -** Find students who scored more than 80 in "Web Development" only if they belong to the "Computer Science" department.

## Query: -

db.enrollments.find(

{ course\_title: "Web Development", marks: { $gt: 80 }, department: "Computer Science" })

## Output:-



▪ Nothing will show up because there are no students in the Computer Science department who scored more than 80 in 'Web Development'.

* Access nested fields using dot notation.

* Combine multiple field conditions.

* Query subdocuments efficiently.

* Focused filtering by department and performance.

# Advanced Aggregation (Challenge Level)

**Q13: -** For each faculty member, list the names of all students enrolled in their courses along with average marks per student per faculty.

## Query: -

db.faculty.aggregate(

[{ $lookup: { from: "courses", localField: "courses", foreignField: "\_id", as: "courseInfo" }}, { $unwind: "$courseInfo" },

{ $lookup: { from: "enrollments", localField: "courseInfo.\_id", foreignField: "course\_id", as: "enrolledStudents" }},

{ $unwind: "$enrolledStudents" },

{ $lookup: { from: "students", localField: "enrolledStudents.student\_id", foreignField: "\_id", as: "studentInfo" }},

{ $project: { \_id: 0, facultyName: "$name", studentName: { $arrayElemAt:

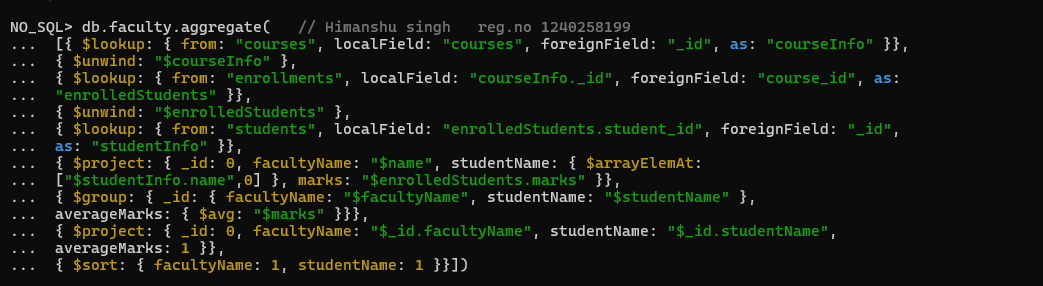
["$studentInfo.name",0] }, marks: "$enrolledStudents.marks" }},

{ $group: { \_id: { facultyName: "$facultyName", studentName: "$studentName" }, averageMarks: { $avg: "$marks" }}},

{ $project: { \_id: 0, facultyName: "$\_id.facultyName", studentName: "$\_id.studentName", averageMarks: 1 }},

{ $sort: { facultyName: 1, studentName: 1 }}])

## Output: -



* Multi-level joins using $lookup.
* $addToSet to avoid duplicate student names.
* $avg to compute average marks per faculty.
* Real-world aggregation chaining.

**Q14: -** Show the most popular activity type (e.g., Hackathon, Seminar, etc.) by number of student participants.

## Query: -

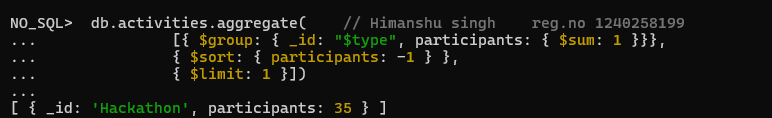
db.activities.aggregate(

[{ $group: { \_id: "$type", participants: { $sum: 1 }}},

{ $sort: { participants: -1 } },

{ $limit: 1 }])

## Output: -



* $unwind to count array elements.

* $group and $sum for totals.

* $sort to rank results.

* Identify “most popular” entities.