Report - 1

```
university_db=# \dt
                List of relations
 Schema |         Name          | Type  |  Owner
--------+-----------------------+-------+----------
 public | course_instructors    | table | postgres
 public | courses               | table | postgres
 public | departments           | table | postgres
 public | enrollments           | table | postgres
 public | instructors           | table | postgres
 public | students              | table | postgres
(6 rows)
```

```
university_db=# \d+ course_instructors
                                  Table "public.course_instructors"
    Column     |  Type   | Collation | Nullable | Default | Storage | Compression | Stats target | Description
---------------+---------+-----------+----------+---------+---------+-------------+--------------+-------------
 course_id     | integer |           | not null |         | plain   |             |              |
 instructor_id | integer |           | not null |         | plain   |             |              |
Indexes:
    "course_instructors_pkey" PRIMARY KEY, btree (course_id, instructor_id)
Foreign-key constraints:
    "course_instructors_course_id_fkey" FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE
    "course_instructors_instructor_id_fkey" FOREIGN KEY (instructor_id) REFERENCES instructors(instructor_id) ON DELETE CASCADE
Access method: heap
```

```
university_db=# \d+ courses
                                                        Table "public.courses"
    Column     |          Type          | Collation | Nullable |                 Default                  | Storage  | Compression | Stats target | Description
---------------+------------------------+-----------+----------+------------------------------------------+----------+-------------+--------------+-------------
 course_id     | integer                |           | not null | nextval('courses_course_id_seq'::regclass) | plain    |             |              |
 course_name   | character varying(100) |           | not null |                                          | extended |             |              |
 department_id | integer                |           |          |                                          | plain    |             |              |
 credits       | integer                |           | not null |                                          | plain    |             |              |
Indexes:
    "courses_pkey" PRIMARY KEY, btree (course_id)
Check constraints:
    "courses_credits_check" CHECK (credits > 0)
Foreign-key constraints:
    "courses_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(department_id) ON DELETE CASCADE
Referenced by:
    TABLE "course_instructors" CONSTRAINT "course_instructors_course_id_fkey" FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE
    TABLE "enrollments" CONSTRAINT "enrollments_course_id_fkey" FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE
Access method: heap
```

```
university_db=# \d+ departments
                                                        Table "public.departments"
     Column      |         Type          | Collation | Nullable |                   Default                    | Storage  | Compression | Stats target | Description
-----------------+-----------------------+-----------+----------+----------------------------------------------+----------+-------------+--------------+-------------
 department_id   | integer               |           | not null | nextval('departments_department_id_seq'::regclass) | plain    |             |              |
 department_name | character varying(50) |           | not null |                                              | extended |             |              |
Indexes:
    "departments_pkey" PRIMARY KEY, btree (department_id)
    "departments_department_name_key" UNIQUE CONSTRAINT, btree (department_name)
Referenced by:
    TABLE "courses" CONSTRAINT "courses_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(department_id) ON DELETE CASCADE
    TABLE "instructors" CONSTRAINT "instructors_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(department_id) ON DELETE SET NULL
    TABLE "students" CONSTRAINT "students_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(department_id) ON DELETE SET NULL
Access method: heap
```

```
university_db=# \d+ enrollments
                                                        Table "public.enrollments"
    Column     |     Type     | Collation | Nullable |                  Default                   | Storage  | Compression | Stats target | Description
---------------+--------------+-----------+----------+--------------------------------------------+----------+-------------+--------------+-------------
 enrollment_id | integer      |           | not null | nextval('enrollments_enrollment_id_seq'::regclass) | plain    |             |              |
 student_id    | integer      |           |          |                                            | plain    |             |              |
 course_id     | integer      |           |          |                                            | plain    |             |              |
 grade         | character(2) |           |          |                                            | extended |             |              |
Indexes:
    "enrollments_pkey" PRIMARY KEY, btree (enrollment_id)
Check constraints:
    "enrollments_grade_check" CHECK (grade = ANY (ARRAY['A'::bpchar, 'B'::bpchar, 'C'::bpchar, 'D'::bpchar, 'F'::bpchar, 'W'::bpchar]))
Foreign-key constraints:
    "enrollments_course_id_fkey" FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE
    "enrollments_student_id_fkey" FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE CASCADE
Access method: heap
```

```
university_db=# \d+ instructors
                                                                Table "public.instructors"
    Column     |          Type          | Collation | Nullable |                      Default                       | Storage  | Compression | Stats target | Description
---------------+------------------------+-----------+----------+----------------------------------------------------+----------+-------------+--------------+-------------
 instructor_id | integer                |           | not null | nextval('instructors_instructor_id_seq'::regclass) | plain    |             |              |
 first_name    | character varying(30)  |           | not null |                                                    | extended |             |              |
 last_name     | character varying(30)  |           | not null |                                                    | extended |             |              |
 email         | character varying(100) |           | not null |                                                    | extended |             |              |
 department_id | integer                |           |          |                                                    | plain    |             |              |
Indexes:
    "instructors_pkey" PRIMARY KEY, btree (instructor_id)
    "instructors_email_key" UNIQUE CONSTRAINT, btree (email)
Foreign-key constraints:
    "instructors_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(department_id) ON DELETE SET NULL
Referenced by:
    TABLE "course_instructors" CONSTRAINT "course_instructors_instructor_id_fkey" FOREIGN KEY (instructor_id) REFERENCES instructors(instructor_id) ON DELETE CASCADE
Access method: heap
```

```
university_db=# \d+ students
                                                            Table "public.students"
     Column      |          Type          | Collation | Nullable |                   Default                    | Storage  | Compression | Stats target | Description
-----------------+------------------------+-----------+----------+----------------------------------------------+----------+-------------+--------------+-------------
 student_id      | integer                |           | not null | nextval('students_student_id_seq'::regclass) | plain    |             |              |
 first_name      | character varying(30)  |           | not null |                                              | extended |             |              |
 last_name       | character varying(30)  |           | not null |                                              | extended |             |              |
 email           | character varying(100) |           | not null |                                              | extended |             |              |
 department_id   | integer                |           |          |                                              | plain    |             |              |
 enrollment_year | integer                |           |          |                                              | plain    |             |              |
Indexes:
    "students_pkey" PRIMARY KEY, btree (student_id)
    "students_email_key" UNIQUE CONSTRAINT, btree (email)
Check constraints:
    "students_enrollment_year_check" CHECK (enrollment_year >= 2000 AND enrollment_year::numeric <= EXTRACT(year FROM CURRENT_DATE))
Foreign-key constraints:
    "students_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(department_id) ON DELETE SET NULL
Referenced by:
    TABLE "enrollments" CONSTRAINT "enrollments_student_id_fkey" FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE CASCADE
Access method: heap
```

Report: Mapping from Relational Schema (PostgreSQL) to Document-Based Schema (MongoDB)

1. Relational Schema (PostgreSQL) Overview

The university's relational schema consists of the following tables:

- **courses:** Contains course information like `course_id`, `course_name`, `department_id`, and `credits`.
- **departments:** Stores department details, including `department_id` and `department_name`.
- **instructors:** Stores instructor details with `instructor_id`, `first_name`, `last_name`, `email`, and `department_id`.
- **students:** Contains student details like `student_id`, `first_name`, `last_name`, `email`, `department_id`, and `enrollment_year`.
- **course_instructors:** A linking table connecting `course_id` to `instructor_id` (many-to-many relationship).
- **enrollments:** Stores student enrollments with `enrollment_id`, `student_id`, `course_id`, and `grade`.

This normalized schema efficiently handles the relationships between students, courses, departments, and instructors, ensuring data consistency through foreign key constraints.

2. Document-Based Schema (MongoDB) Overview

In MongoDB, denormalization is commonly used to reduce the number of joins (which are expensive). This schema design replicates the relational structure while ensuring that querying is efficient for the required workload.

Schema Design:

courses (Document Schema)

json:

```
{
  "_id": 1,
  "course_name": "Data Structures",
  "department_id": 1,
  "credits": 3,
  "enrollment_count": 2,
  "instructors": [
    1
  ]
}
```

- **Justification:**
  - The `courses` document includes all the essential attributes from the relational schema: `course_name`, `department_id`, and `credits`.
  - The `instructors` field holds an array of instructor IDs (`instructors` array) to maintain the many-to-many relationship from the `course_instructors` table.
  - The `enrollment_count` field is added to optimize queries that require the number of students enrolled, avoiding joins with the `enrollments` table.

departments (Document Schema)

json:

```
{
  "_id": 1,
  "department_name": "Computer Science",
  "students": [
    1,
```

```
    11,
    21,
    31
  ],
  "courses": [
    1,
    2,
    22,
    23,
    24,
    25
  ]
}
```

- **Justification:**
  - The `departments` document embeds an array of `students` and `courses` IDs to reduce the need for joining with the `students` and `courses` tables.
  - This supports queries such as "listing all courses offered by a specific department" and "finding the total number of students per department."

instructors (Document Schema)

json:

```
{
  "_id": 1,
  "first_name": "Mark",
  "last_name": "Taylor",
  "email": "mark.taylor@example.com",
  "department_id": 1,
  "courses_taught": [
    1,
    2,
    13,
    29,
    22,
    23,
    24
  ]
}
```

- **Justification:**

    - The `instructors` document contains details of the instructor and an array of `courses_taught` IDs, which represents the many-to-many relationship from the `course_instructors` table.
    - This design enables efficient queries like "finding instructors who have taught all the BTech CSE core courses during their tenure."

students (Document Schema)

json:

```
{
  "_id": 1,
  "first_name": "John",
  "last_name": "Doe",
  "email": "john.doe@example.com",
  "department_id": 1,
  "enrollments": [
    {
      "course_id": 1,
      "grade": "A"
    },
    {
      "course_id": 2,
      "grade": "B"
    },
    {
      "course_id": 3,
      "grade": "A"
    }
  ]
}
```

- **Justification:**
    - The `students` document embeds `enrollments`, which is an array of sub-documents containing `course_id`and `grade`. This denormalization eliminates the need for frequent joins with the `enrollments` table.
    - It supports queries such as "fetching all students enrolled in a specific course" and calculating student performance based on grades.

---

3. MongoDB Validation Data Schema

courses

json:

```
{
  $jsonSchema: {
    bsonType: 'object',
    required: [
      '_id',
      'course_name',
      'department_id',
      'credits',
      'enrollment_count',
      'instructors'
    ],
    properties: {
      _id: {
        bsonType: 'int',
        description: 'must be an integer and is required'
      },
      course_name: {
        bsonType: 'string',
        description: 'must be a string and is required'
      },
      department_id: {
        bsonType: 'int',
        description: 'must be an integer and is required'
      },
      credits: {
        bsonType: 'int',
        minimum: 1,
        maximum: 6,
        description: 'must be an integer between 1 and 6'
      },
      enrollment_count: {
        bsonType: 'int',
        minimum: 0,
        description: 'must be a non-negative integer'
      },
      instructors: {
        bsonType: 'array',
        items: {
```

```
          bsonType: 'int',
          description: 'must be an array of integers representing
instructor IDs'
        },
        minItems: 1,
        description: 'must be a non-empty array'
      }
    }
  }
}
```

- **_id:** An integer that uniquely identifies the course.
- **course_name:** A string representing the name of the course, which is required.
- **department_id:** An integer linking the course to its department, required for maintaining relationships.
- **credits:** An integer indicating the number of credits awarded for the course, constrained to a minimum of 1 and a maximum of 6.
- **enrollment_count:** A non-negative integer representing the number of students enrolled in the course.
- **instructors:** An array of integers, each representing the ID of an instructor assigned to the course, with a requirement for at least one instructor.

departments

json:

```
{
  $jsonSchema: {
    bsonType: 'object',
    required: [
      '_id',
      'department_name',
      'students',
      'courses'
    ],
    properties: {
      _id: {
        bsonType: 'int',
        description: 'must be an integer and is required'
      },
      department_name: {
        bsonType: 'string',
        description: 'must be a string and is required'
      },
```

```
      students: {
        bsonType: 'array',
        items: {
          bsonType: 'int',
          description: 'must be an array of integers representing student
IDs'
        },
        minItems: 1,
        description: 'must be a non-empty array of student IDs'
      },
      courses: {
        bsonType: 'array',
        items: {
          bsonType: 'int',
          description: 'must be an array of integers representing course
IDs'
        },
        minItems: 1,
        description: 'must be a non-empty array of course IDs'
      }
    }
  }
}
```

- **department_name:** A string that names the department.
- **students:** An array of integers representing the IDs of students in the department. This field must contain at least one student.
- **courses:** An array of integers representing the IDs of courses offered by the department, also requiring at least one course.

instructors

json:

```
{
  $jsonSchema: {
    bsonType: 'object',
    required: [
      '_id',
      'first_name',
      'last_name',
      'email',
      'department_id',
```

```
      'courses_taught'
    ],
    properties: {
      _id: {
        bsonType: 'int',
        description: 'must be an integer and is required'
      },
      first_name: {
        bsonType: 'string',
        description: 'must be a string and is required'
      },
      last_name: {
        bsonType: 'string',
        description: 'must be a string and is required'
      },
      email: {
        bsonType: 'string',
        pattern: '^.+@.+\\..+$',
        description: 'must be a valid email format'
      },
      department_id: {
        bsonType: 'int',
        description: 'must be an integer and is required'
      },
      courses_taught: {
        bsonType: 'array',
        items: {
          bsonType: 'int',
          description: 'must be an array of integers representing course
IDs'
        },
        minItems: 1,
        description: 'must be a non-empty array'
      }
    }
  }
}
```

- **first_name and last_name:** Strings that contain the instructor's names, both required.
- **email:** A string that must match a specific email format pattern (`^.+@.+\\..+$`), ensuring valid email addresses.
- **department_id:** An integer linking the instructor to their respective department.

- **courses_taught:** An array of integers representing the IDs of courses taught by the instructor, requiring at least one course.

students

json:

```
{
  $jsonSchema: {
    bsonType: 'object',
    required: [
      '_id',
      'first_name',
      'last_name',
      'email',
      'department_id',
      'enrollments'
    ],
    properties: {
      _id: {
        bsonType: 'int',
        description: 'must be an integer and is required'
      },
      first_name: {
        bsonType: 'string',
        description: 'must be a string and is required'
      },
      last_name: {
        bsonType: 'string',
        description: 'must be a string and is required'
      },
      email: {
        bsonType: 'string',
        pattern: '^.+@.+\\..+$',
        description: 'must be a valid email format'
      },
      department_id: {
        bsonType: 'int',
        description: 'must be an integer and is required'
      },
      enrollments: {
        bsonType: 'array',
        items: {
```

```
          bsonType: 'object',
          required: ['course_id', 'grade'],
          properties: {
            course_id: {
              bsonType: 'int',
              description: 'must be an integer and is required'
            },
            grade: {
              bsonType: 'string',
              pattern: '^(A|B|C|D|F)$',
              description: 'must be a valid grade (A, B, C, D, F)'
            }
          }
        },
        minItems: 1,
        description: 'must be a non-empty array'
      }
    }
  }
}
```

- **first_name and last_name:** Required strings for the student's name.
- **email:** Must follow a valid email format.
- **department_id:** Links the student to their department.
- **enrollments:** An array of objects where each object represents a course enrollment. Each enrollment must contain:
    - **course_id:** An integer for the course ID.
    - **grade:** A string that must match specific grade patterns (A, B, C, D, F), ensuring only valid grades are recorded.

Key Benefits of Using Validation Schemas

1. **Data Integrity:** Ensures that only documents with the correct structure and data types are stored in the database.
2. **Consistent Data:** Prevents issues like missing required fields or incorrect data types, leading to a more reliable database.
3. **Better Query Performance:** When data is consistently structured, queries can be optimized and executed more efficiently.

---

4. Query Workload Mapping and Justifications

The schema was designed to efficiently support the following queries:

a) Fetching all students enrolled in a specific course

- **Query:** Use the `students` collection and filter by `enrollments.course_id`.
- **Justification:** The `enrollments` array in the `students` collection stores the course IDs directly, making this query fast without needing any joins.

b) Calculating the average number of students enrolled in courses offered by a particular instructor

- **Query:** Aggregate on the `instructors` collection to count the number of students enrolled in each course they teach.
- **Justification:** The `courses_taught` array in the `instructors` collection allows efficient retrieval of the courses taught by an instructor, and cross-referencing with the `courses` collection provides enrollment counts.

c) Listing all courses offered by a specific department

- **Query:** Query the `departments` collection and retrieve the `courses` array.
- **Justification:** The `courses` array is embedded directly in the `departments` collection, making this query efficient.

d) Finding the total number of students per department

- **Query:** Query the `departments` collection and retrieve the `students` array size.
- **Justification:** The `students` array is already embedded in the `departments` collection, making this query fast.

e) Finding instructors who have taught all the BTech CSE core courses

- **Query:** Query the `instructors` collection and filter by the `courses_taught` array, checking if it contains all the BTech CSE core course IDs.
- **Justification:** The `courses_taught` array allows for direct querying without needing joins with `courses`.

f) Finding the top 10 courses with the highest enrollments

- **Query:** Sort the `courses` collection by `enrollment_count` and limit to 10.

- **Justification:** The `enrollment_count` field is maintained in the `courses` collection to avoid expensive aggregation.

---

5. Denormalization Justifications

Denormalization in MongoDB is necessary for efficient querying, especially to avoid costly joins. Key areas where denormalization was applied include:

- **Instructors:** The `courses_taught` array directly stores course IDs.
- **Students:** Embedding `enrollments` within each student avoids multiple joins with the `enrollments` table.
- **Departments:** Embedding `students` and `courses` arrays within departments simplifies department-level queries.

---

6. Conclusion

The document-based MongoDB schema was designed to closely mimic the relational structure while enhancing the efficiency of specific query workloads. The use of denormalization, embedding, and optimized field design ensures that complex queries involving students, courses, instructors, and departments are handled efficiently in MongoDB.