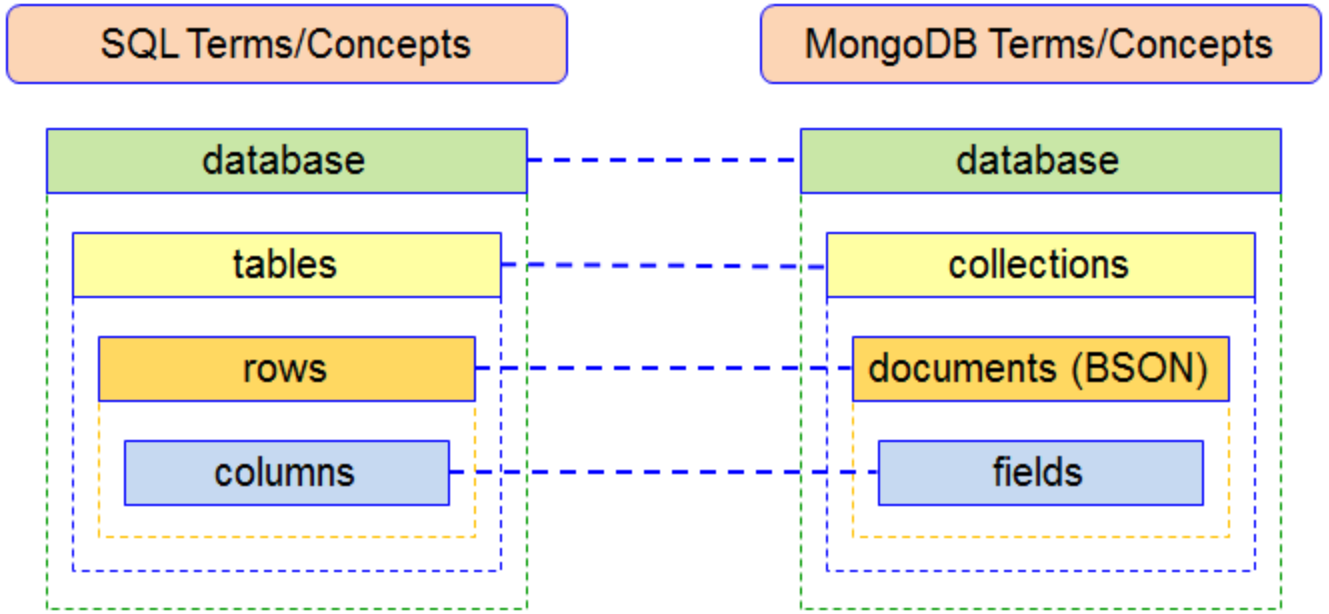
# MongoDB III

CS 341 Database Systems

# Basic Comparison



# Schema Design Best Practices

- Usually, flexible
- Major Considerations
  - Storage of data
  - Good query performance
  - Reasonable amount of hardware

# Relations

- **One-to-One** - Prefer key value pairs within the document
  - Department and Manager
- **One-to-Few** - Prefer embedding
  - Customer and Addresses
- **One-to-Many** - Prefer embedding
  - Department and Staff
- **Many-to-Many** - Referencing or 1-way embedding or 2-way embedding
  - Courses and Students

# Consider a Scenario

- Each SalesRep is assigned to one department. Each department has many sales representatives
- Each department is headed by a manager who is also a SalesRep
- Each customer can be dealt by multiple SalesRep and different SalesRep can deal with different customers

# Consider a Scenario

- Each **SalesRep** is assigned to one **department**. Each department has many sales representatives
- Each department is headed by a manager who is also a SalesRep
- Each **customer** can be dealt by multiple SalesRep and different SalesRep can deal with different customers

Department – SalesRep
1-M (Assigned)

Department – SalesRep
1-1 (Manager)

Customer – SalesRep
M-M (Deals)

# Collections

- **Department**
  - SalesRep as key:value pair or embedded document (Manager)
- **SalesRep**
  - Department as embedded document
  - Embedded array of customers dealt by this salesRep
- **Customer**
  - Embedded array of SalesRep dealing this customer

## Scenario

Refer to the scenario create collections with appropriate validation rules.

- Each SalesRep is assigned to one department. Each department has many sales representatives.
- Each department is headed by a manager who is also a SalesRep
- Each customer can be dealt by multiple SalesRep and different SalesRep can deal with different customers

The Sales Rep collection stores the s_name, s_email and s_performance score. The customer collection contains the cust_name, cust_add and cust_email. The department collection contains the dept_name, dept_region, and grade.

**Limitations:**

- A SalesRep can only deal with maximum 10 customers
- A customer is dealt by maximum 5 SalesRep.
- The performance_score is given as a number between 1 and 10.
- The dept_region must be from among East, West, Central
- Grade is a single character string.
- The fields in red are mandatory fields.
- To implement the M-M relation, both sides will keep embedded documents. We only require name and email address in the array. *Hint: sub-schema*

# Departments

```
db.createCollection("departments",{
 validator: {
  $jsonSchema: {
   bsonType: "object",
   required: ["dept_name", "dept_region","dept_manager"],
   properties: {
    dept_name: {
     bsonType: "string",
     description: "Department name is required and must be
unique."
    },
    dept_region: {
     bsonType: "string",
     enum: ["East", "West", "Central"],
     description: "Department region must be East, West, or
Central."
    },
    grade: {
     bsonType: "string",
     maxLength: 1,
     description: "Grade must be a single character."
    },
```

```
dept_manager: {
        bsonType: "object",
        required: ["s_name", "s_email"],
        properties: {
         s_name: {
          bsonType: "string",
          description: "Manager's name is required."
         },
         s_email: {
          bsonType: "string",
          description: "Manager's email is required."
         }
        },
        description: "Manager who is also a SalesRep."
      }
     }
    }
  }
});
```

## Scenario

Refer to the scenario create collections with appropriate validation rules.

- Each SalesRep is assigned to one department. Each department has many sales representatives.
- Each department is headed by a manager who is also a SalesRep
- Each customer can be dealt by multiple SalesRep and different SalesRep can deal with different customers

The Sales Rep collection stores the s_name, s_email and s_performance score. The customer collection contains the cust_name, cust_add and cust_email. The department collection contains the dept_name, dept_region, and grade.

**Limitations:**

- A SalesRep can only deal with maximum 10 customers
- A customer is dealt by maximum 5 SalesRep.
- The performance_score is given as a number between 1 and 10.
- The dept_region must be from among East, West, Central
- Grade is a single character string.
- The fields in red are mandatory fields.
- To implement the M-M relation, both sides will keep embedded documents. We only require name and email address in the array. *Hint: sub-schema*

# SalesRep

```
db.createCollection("salesreps", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["s_name", "s_email", "departmentId"],
      properties: {
        s_name: {
          bsonType: "string",
          description: "SalesRep name is required."
        },
        s_email: {
          bsonType: "string",
          description: "SalesRep email is required unique."
        },
        s_performance: {
          bsonType: "number",
          minimum: 1,
          maximum: 10,
          description: "SalesRep performance score must be a number between 1 and 10."
        },
        departmentId: {
          bsonType: "objectId",
          description: "SalesRep must be assigned to a department."
        },
```

Linked

Array of embedded documents.

```
        customers: {
          bsonType: "array",
          maxItems: 10,
          items: {
            bsonType: "object",
            required: ["cust_name", "cust_email"]
            properties: {
              cust_name: {
                bsonType: "string",
                description: "Customer name is required."
              },
              cust_email: {
                bsonType: "string",
                description: "Customer email is required."
              }
            }
          },
          description: "List of customers associated with the sales rep."
        }
      }
    }
  }
});
```

# SalesRep

```
db.createCollection("salesreps", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["s_name", "s_email", "department"],
      properties: {
        s_name: {
          bsonType: "string",
          description: "SalesRep name is required."
        },
        s_email: {
          bsonType: "string",
          description: "SalesRep email is required."
        },
        s_performance: {
          bsonType: “number",
          minimum: 1,
          maximum: 10,
          description: "SalesRep performance score must be a number between 1 and 10."
        },
        department: {
          bsonType: “string",
          description: "SalesRep must be assigned to a department."
        },
```

Key-value

```
customers: {
        bsonType: "array",
        maxItems: 10,
        items: {
          bsonType: "object",
          required: [“cust_name", “cust_email"],
          properties: {
            cust_name: {
              bsonType: "string",
              description: "Customer name is required."
            },
            cust_email: {
              bsonType: "string",
              description: "Customer email is required."
            }
          }
        },
        description: "List of customers associated with the sales rep."
      }
    }
  }
});
```

# SalesRep

```
db.createCollection("salesreps", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["s_name", "s_email", "department"],
      properties: {
        s_name: {
          bsonType: "string",
          description: "SalesRep name is required."
        },
        s_email: {
          bsonType: "string",
          description: "SalesRep email is required."
        },
        s_performance: {
          bsonType: "number",
          minimum: 1,
          maximum: 10,
          description: "SalesRep performance score must be a number between
1 and 10."
        },
        department: {
          bsonType: "object",
          properties: {
          department_name: {
            bsonType: "string"
          },
            description: "SalesRep must be assigned to a department.
"} , ... other fields as needed
        },
```

Embedded

```
customers: {
        bsonType: "array",
        maxItems: 10,
        items: {
          bsonType: "object",
          required: ["cust_name", "cust_email"],
          properties: {
            cust_name: {
              bsonType: "string",
              description: "Customer name is required."
            },
            cust_email: {
              bsonType: "string",
              description: "Customer email is required."
            }
          }
        },
        description: "List of customers associated with
the sales rep."
      }
    }
  }
});
```

## Scenario

Refer to the scenario create collections with appropriate validation rules.

- Each SalesRep is assigned to one department. Each department has many sales representatives.
- Each department is headed by a manager who is also a SalesRep
- Each customer can be dealt by multiple SalesRep and different SalesRep can deal with different customers

The Sales Rep collection stores the s_name, s_email and s_performance score. The customer collection contains the cust_name, cust_add and cust_email. The department collection contains the dept_name, dept_region, and grade.

**Limitations:**

- A SalesRep can only deal with maximum 10 customers
- A customer is dealt by maximum 5 SalesRep.
- The performance_score is given as a number between 1 and 10.
- The dept_region must be from among East, West, Central
- Grade is a single character string.
- The fields in red are mandatory fields.
- To implement the M-M relation, both sides will keep embedded documents. We only require name and email address in the array. *Hint: sub-schema*

# Customers

```
db.createCollection("customers",{
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["cust_email"],
      properties: {
        cust_name: {
          bsonType: "string",
          description: "Customer name is required."
        },
        cust_add: {
          bsonType: "string"
        },
        cust_email: {
          bsonType: "string",
          description: "Customer email is required."
        },
        salesReps: {
          bsonType: "array",
          maxItems: 5,
          items: {
            bsonType: "object",
            required: ["s_name", "s_email"],
            properties: {
              s_name: {
                bsonType: "string",
                description: "SalesRep name is required."
              },
```

```
s_email: {
                bsonType: "string",
                description: "SalesRep email is required."
              }
            }
          },
          description: "List of sales reps associated with the
customer."
        }
      }
    }
  }
});
```

# Querying Linked Documents

# SalesRep Linked

```
db.createCollection("salesreps", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["s_name", "s_email", "departmentId"],
      properties: {
        s_name: {
          bsonType: "string",
          description: "SalesRep name is required."
        },
        s_email: {
          bsonType: "string",
          description: "SalesRep email is required unique."
        },
        s_performance: {
          bsonType: "number",
          minimum: 1,
          maximum: 10,
          description: "SalesRep performance score must be a number between 1 and 10."
        },
        departmentId: {
          bsonType: "objectId",
          description: "SalesRep must be assigned to a department."
        },
```

```
customers: {
        bsonType: "array",
        maxItems: 10,
        items: {
          bsonType: "object",
          properties: {
            cust_name: {
              bsonType: "string",
              description: "Customer name is required."
            },
            cust_email: {
              bsonType: "string",
              description: "Customer email is required."
            }
          }
        },
        description: "List of customers associated with the sales rep."
      }
    }
  }
});
```

# $lookup

- The **$lookup** operator allows you to perform a **join** between two collections in MongoDB.

```
db.salesreps.aggregate([
 {
   $lookup: {
     from: "departments",  // The collection to join with
     localField: "departmentId",  // The field in `salesreps` that stores the department's ObjectId
     foreignField: "_id",  // The field in `departments` that contains the ObjectId
     as: "departmentInfo"  // The field where the resulting department information will be stored
   }
 },
 {
   $unwind: "$departmentInfo"  // Flatten the array created by `$lookup` so each sales rep gets a single department
 }
]);
```

**SALES REP**

```
{
  "_id": ObjectId("..."),
  "s_name": "John Doe",
  "s_email": "john.doe@example.com",
  "departmentId":
ObjectId("60b8a9bb1d52c0d2f7a8c8d9")
}
```

**DEPARTMENT**

```
{
  "_id": ObjectId("60b8a9bb1d52c0d2f7a8c8d9"),
  "dept_name": "Sales",
  "dept_region": "East",
  "grade": "A",
  "manager_name": "Alice Smith"
}
```

# After $lookup stage

```
{
  "_id": ObjectId("..."),
  "s_name": "John Doe",
  "s_email": "john.doe@example.com",
  "departmentId":
ObjectId("60b8a9bb1d52c0d2f7a8c8d9"),
  "departmentInfo": [
    {
      "_id": ObjectId("60b8a9bb1d52c0d2f7a8c8d9"),
      "dept_name": "Sales",
      "dept_region": "East",
      "grade": "A",
      "manager_name": "Alice Smith"
    }
  ]
}
```

# After $unwind stage

```
{
  "_id": ObjectId("..."),
  "s_name": "John Doe",
  "s_email": "john.doe@example.com",
  "departmentId":
ObjectId("60b8a9bb1d52c0d2f7a8c8d9"),
  "departmentInfo": {
    "_id":
ObjectId("60b8a9bb1d52c0d2f7a8c8d9"),
    "dept_name": "Sales",
    "dept_region": "East",
    "grade": "A",
    "manager_name": "Alice Smith"
  }
}
```

# Orders and Inventory

```
db.orders.insertMany( [
{ "_id" : 1, "item" : "almonds", "price" : 12, "quantity" : 2 },
{ "_id" : 2, "item" : "pecans", "price" : 20, "quantity" : 1 },
{ "_id" : 3 }
] )
```

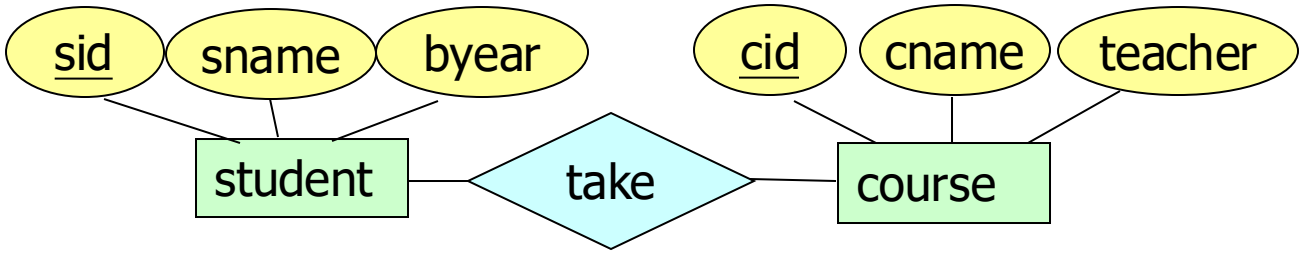
```
db.inventory.insertMany( [
{ "_id" : 1, "sku" : "almonds", "description": "product 1", "instock" : 120 },
{ "_id" : 2, "sku" : "bread", "description": "product 2", "instock" : 80 },
{ "_id" : 3, "sku" : "cashews", "description": "product 3", "instock" : 60 },
{ "_id" : 4, "sku" : "pecans", "description": "product 4", "instock" : 70 },
{ "_id" : 5, "sku": null, "description": "Incomplete" },
{ "_id" : 6 }
] )
```

```
db.orders.aggregate( [
  {
    $lookup:
     {
       from: "inventory",
       localField: "item",
       foreignField: "sku",
       as: "inventory_docs"
     }
  }
] )
```

## Output

```
{
  "_id" : 1,
  "item" : "almonds",
  "price" : 12,
  "quantity" : 2,
  "inventory_docs" : [
    { "_id" : 1, "sku" : "almonds", "description" : "product 1",
"instock" : 120 }
  ]
}
{
  "_id" : 2,
  "item" : "pecans",
  "price" : 20,
  "quantity" : 1,
  "inventory_docs" : [
    { "_id" : 4, "sku" : "pecans", "description" : "product 4", "instock" : 70 }
  ]
}
{
  "_id" : 3,
  "inventory_docs" : [
    { "_id" : 5, "sku" : null, "description" : "Incomplete" },
    { "_id" : 6 }
  ]
}
```

# Consider a Relational Schema translated to Collections → *The Linking Approach*




student (sid, sname, byear)
course (cid, cname, teacher)
take (sid, cid)

# Find the course IDs of all courses taken by the student with the name “Raymond”

student (sid, sname, byear)
course (cid, cname, teacher)
take (sid, cid)

It involves a “join” operation between two collections.

```
db.take.aggregate(
[
  {
    $lookup: {
                    localField: "sid",
                     from: "student",
                     foreignField: "sid",
                     as: "student_info"
                  }
  },
 { $unwind: "$student_info"},
{ $match: { "student_info.sname": "Raymond" } },
  { $project: { cid: 1, _id: 0 } }
]
)
```

Natural Join

```
SELECT cid
FROM take JOIN student ON take.sid=student.sid
WHERE student.sname = “Raymond”
```

Output

{ cid : "90123" }
{ cid : "90696" }
{ cid : "95270" }

In the output, there can be multiple occurrences of the same course ID.

It could be regarded as "correct" since we do not require "distinct" course IDs in the output.

Suppose that we require the "distinct" requirement. What should we do?

```
db.take.aggregate(
[
  {
     $lookup: {
                        localField: "sid",
                         from: "student",
                         foreignField: "sid",
                         as: "student_info"
                    }
   },
   { $unwind: "$student_info"},
   { $match: { "student_info.sname": "Raymond" } },
   { $project: { cid: 1, _id: 0 } },
   { $group: { _id: "$cid" } }
]
)
```

# Find the student IDs of all students who take the course taught by "Prof. Charles"

student (sid, sname, byear)
course (cid, cname, teacher)
take (sid, cid)

```
db.take.aggregate(
[
  {
    $lookup: {
                    localField: "cid",
                     from: "course",
                     foreignField: "cid",
                     as: "course_info"
                  }
   },

   { $unwind: "$course_info"},
   { $match: { "course_info.teacher": "Prof. Charles" } },
   { $project: { sid: 1, _id: 0 } }
]
)
```

# Names of all students taking the course by “Prof. Charles”

```
db.student.aggregate([
  {
    $lookup: {
                localField: "sid",
                from: "take",
                foreignField: "sid",
                as: "student_take"
              }
  },
  { $unwind: "$student_take" },
   {
     $lookup: {
                  localField: "student_take.cid",
                  from: "course",
                  foreignField: "cid",
                  as: "student_take_course",
                }
    },
    { $unwind: "$student_take_course" },
    { $match: { "student_take_course.teacher": "Prof. Charles" }  },
    { $project: { sname: 1, _id: 0}  }
])
```

For distinct values, add another stage
{ $group: { _id: "$sname" }}

# Find the class size of each course. Show the course ID, the course name and the class size for each course.

student (sid, sname, byear)
course (cid, cname, teacher)
take (sid, cid)

```
db.take.aggregate(
[
  { $group : {_id : "$cid",  classSize : {$sum : 1}}  },
   { $lookup: {
                        localField: "_id", // from previous stage
                         from: "course",
                         foreignField: "cid",
                         as: "course_info"
                    }    },
   { $unwind: "$course_info" },
    { $project: { _id: 1, cname: "$course_info.cname", classSize: 1 }  } // Rename course name

]
);
```

# Computing Values

$add

$subtract

$multiply

$divide

# Output the field “sid” and a new field called “newSum” which is equal to “a+2” for each document

```
{sid: 1, a:100, b:150}
{sid: 2, a:70, b:200}
{sid: 3, a:500, b:300}
```


```
db.temp4.aggregate([
{ $project: {newSum: {$add: ["$a", 2]}, sid: 1, _id: 0 } }
])
```

Output

```
{ sid : 1, newSum : 102 }
{ sid : 2, newSum : 72 }
{ sid : 3, newSum : 502 }
```

# Output the field “sid” and a new field called “newProduct” which is equal to 2 times “a” for each document

```
{sid: 1, a:100, b:150}
{sid: 2, a:70, b:200}
{sid: 3, a:500, b:300}
```

```
db.temp4.aggregate([
{ $project: {newProduct: {$multiply: ["$a", 2]}, sid: 1, _id: 0 } }
])
```

Output

```
{ sid : 1, newProduct : 200 }
{ sid : 2, newProduct : 140 }
{ sid : 3, newProduct : 1000 }
```