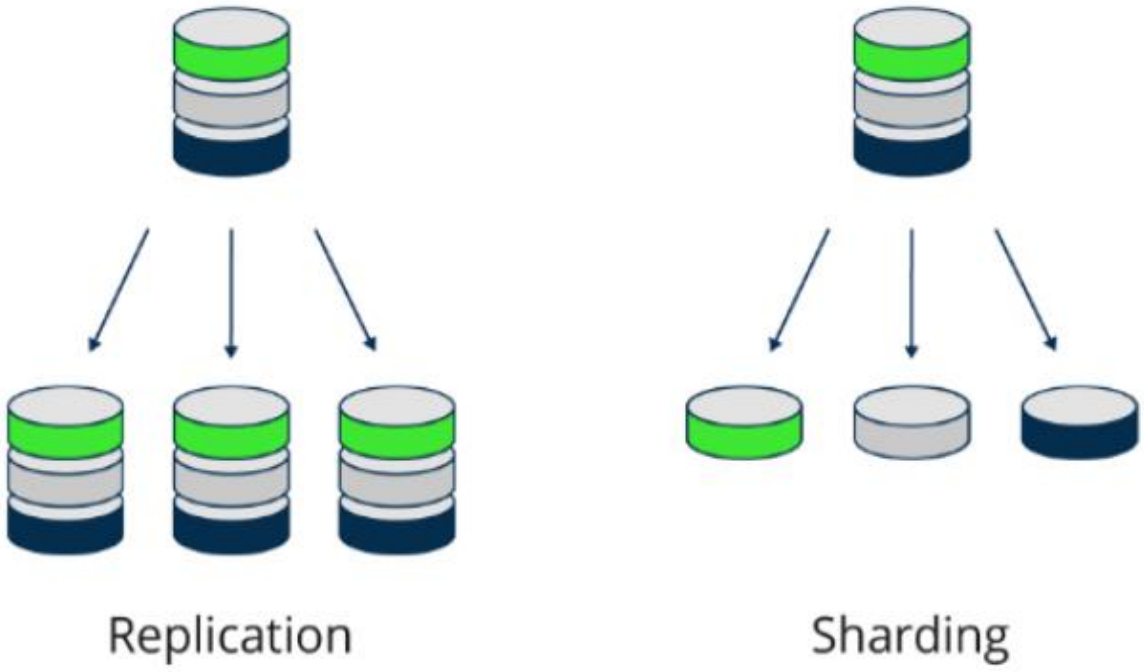
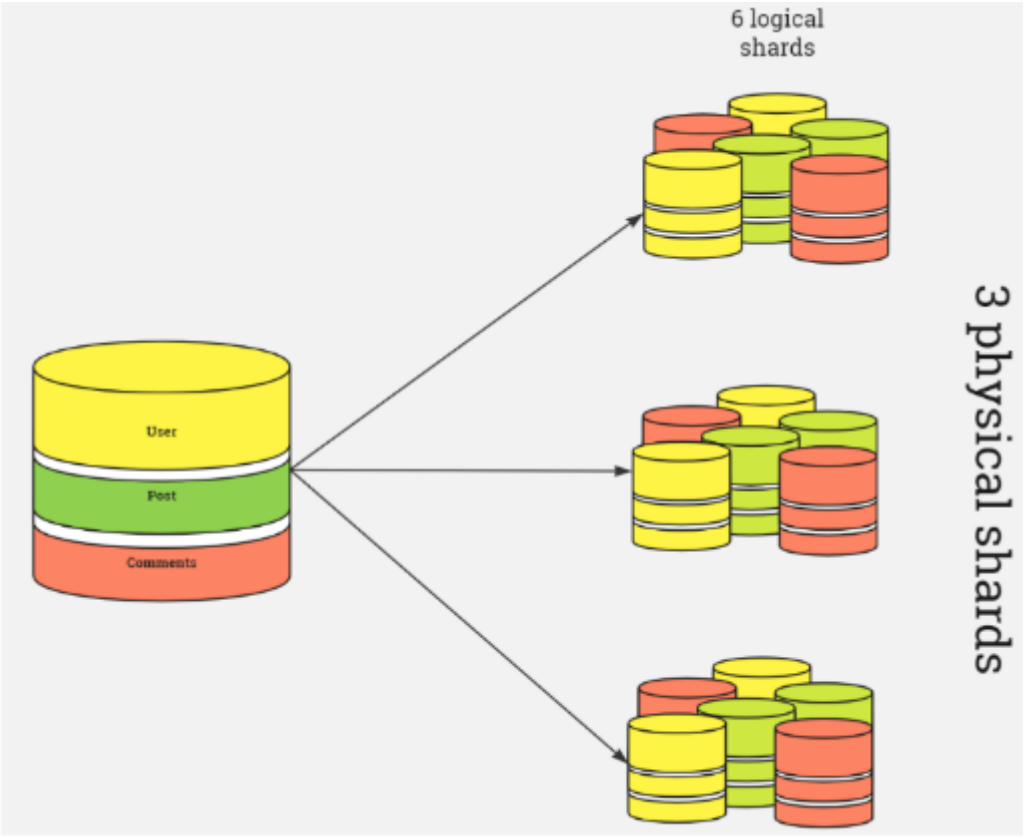
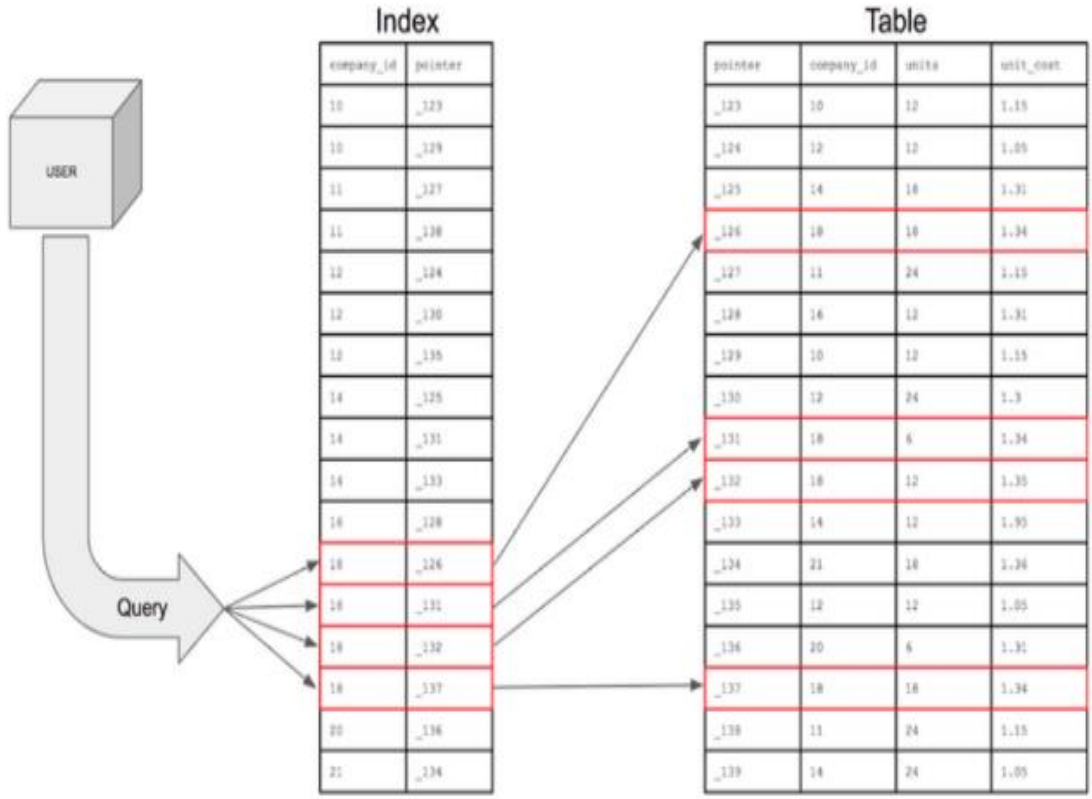
6. Replication VS Sharding:



7. Replication + Sharding:



8. Indexes:



Types of Indexes:

Basic Index Types

- **Single Field Index:**
  - Indexes a single field within a document.
  - Example: `db.collection.createIndex({ field1: 1 })`
- **Compound Index:**
  - Indexes multiple fields in a specified order.
  - Useful for range-based queries involving multiple fields.
  - Example: `db.collection.createIndex({ field1: 1, field2: -1 })`
- **Multikey Index:**
  - Indexes array elements individually.
  - Enables efficient queries on array elements.
  - Example: `db.collection.createIndex({ arrayField: 1 })`

Specialized Index Types

- **Text Index:**
  - Indexes text content for full-text search capabilities.
  - Supports text search operators like $text and $search.
  - Example: `db.collection.createIndex({ text: "text" })`
- **Geospatial Index:**
  - Indexes geospatial data (coordinates) for efficient proximity-based queries.
  - Supports 2dsphere and 2d indexes for different use cases.
  - Example: `db.collection.createIndex({ location: "2dsphere" })`
- **Hashed Index:**
  - Creates a hashed index for the specified field.
  - Primarily used for the `_id` field for performance optimization.
  - Example: `db.collection.createIndex({ _id: "hashed" })`

Additional Considerations

- **Sparse Indexes:**
  - Only index documents where the indexed field exists.
  - Can improve performance for sparse datasets.
- **Unique Indexes:**
  - Ensure that the indexed field has unique values across all documents.
- **TTL Indexes:**
  - Automatically expire documents after a specified time.

Choosing the right index type depends on your specific data structure, query patterns, and performance requirements. Careful index design can significantly improve query performance, but excessive indexing can impact write performance.

Would you like to delve deeper into a specific index type or discuss index creation strategies for a particular use case?

Index program example:

```
db> db.products.insertMany([
...   { _id: 1, name: "Product A", category: "Electronics", price: 99.99, tatags: ["electronics", "gadget"] },
...   { _id: 2, name: "Product B", category: "Clothing", price: 49.99, tags: ["clothing", "fashion"] },
...   { _id: 3, name: "Product C", category: "Electronics", price: 199.99, ttags: ["electronics", "gadget"] },
...   { _id: 4, name: "Product D", category: "Books", price: 29.99 }, // No tags
...   { _id: 5, name: "Product E", category: "Electronics", price: 149.99, ttags: ["electronics"] }
... ]);
{
  acknowledged: true,
  insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4, '4': 5 }
}
db> db.products.createIndex({ name: 1 }, { unique: true });
name_1
db> db.products.createIndex({ tags: 1 }, { sparse: true });
tags_1
db> db.products.createIndex({ category: 1, price: -1 });
category_1_price_-1
```

```
db> db.products.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { name: 1 }, name: 'name_1', unique: true },
  { v: 2, key: { tags: 1 }, name: 'tags_1', sparse: true },
  {
    v: 2,
    key: { category: 1, price: -1 },
    name: 'category_1_price_-1'
  }
]
```