**[빅데시 중간고사 예상문제]**

1. Structured Data와 Unstructured Data의 차이점을 5가지 쓰시오.
- Structured: 행/열/RDB, 문자열/숫자/일시, 저장공간 적게, 레거시 솔루션O, 미리 정의된 데이터 구조
- Unstructured: 행/열,RDB X, 사진/영상, 저장공간 많이, 레거시 솔루션 X, 미리 정의된 데이터 모델 X

2. Big Data의 정의를 특성 4가지를 들어 쓰시오.
- Volume(방대한 용량), Variety(정형화된 데이터+반정형화, 비구조화)
- Velocity(데이터 생성/수집/처리 속도), Veracity(데이터 타당성, 신뢰성)

3. Big Data 분석학의 목표 4가지를 쓰시오.
- 숨겨진 패턴, 알려지지 않은 연관 관계, 시장 트렌드, 소비자 선호

4. Big Data 프레임워크의 정의와 특징 4가지를 쓰시오.
- 방대한 용량의 데이터를 실시간으로 처리하고, 비용 효율성과 내고장성을 충족하는 프레임워크
- 비용 효율성: Scale-Out(HW 분산), Horizontal Scaling으로 비용 효율성 증가, 주어진 과업에 적합할 정도로 구성하고 초기 구축 비용 최소화하여 비용 최소화
- 내고장성: HW 고장, SW 에러, 자원 부족 등으로 오류가 발생하더라도 기능 작동에 영향이 없도록 함
- 호환성: 빅데이터를 수집하기 위한 기존 시스템과의 호환성 (새로 개발 vs. 기존 시스템에 기능 추가)
- 오픈소스

5. 스트리밍 데이터의 정의와 예시, 빅데이터 시스템의 정의와 2가지 방식에 대해 쓰시오.
- 스트리밍 데이터: 다양한 데이터 소스에서 실시간, 지속적으로 생성되는 데이터(영상, GPS 정보 등)
- 빅데이터 시스템: 방대한 용량의 데이터를 처리, 관리하기 위한 분산, 병렬화된 시스템
  (Batch Procesing System vs. Real-time Processing System)

6. Centralized Storage와 Decentralized Storage의 차이점을 쓰시오.
- Centralized: 단일 기기의 DB에 모든 데이터 저장, 사용자는 해당 기기와 직접 통신, 주로 RDB
- Decentralized: 여러 기기의 DB에 분산되어 동시에 실행, 사용자는 하나의 기기와 통신하는지, 아니면 여러 기기와 통신하는지 모름, 주로 NoSQL DB

7. Relation Model의 특징과 단점 4가지를 쓰시오.
- Table = Schema = Relation, 연관된 데이터가 여러 테이블에 걸쳐 생성, 테이블 관계 형성으로 연결
- 유연성 부족(정형화된 데이터에만 적합), 속도 ↓(데이터의 안정적인 보존 목표 설계), 확장성 부족(방대한 데이터 용량에 비적합), 특정 종류의 기본 쿼리문 SQL로 구현 X

8. NoSQL의 정의 3가지를 쓰시오.
"No SQL", "Not Only SQL", "Non Relational“

9. NoSQL의 특징 3가지를 쓰시오.
- 유연성: Schema Free or 느슨한 구조의 스키마 사용, 데이터의 스키마 구조 필요 X
- 확장성: Scale-Out(Horizontal Scaling)을 통한 하드웨어 분산, 특정 시점 이후 비용 효율적
- 내고장성: 여러 기기의 조합이 단일 기기보다 내고장성 뛰어남, 시스템 가용성 향상

10. NoSQL의 유형 4가지를 그림과 함께 설명하시오.
- Key-Value DB: Key, Value 별도로 저장, 대부분의 언어에서 사용하는 데이터 구조
- Column Store Database: Column 모델 기반의 데이터 저장하기 위한 DB
  Column Family 안에 Row 여러개, Row 안에 Column 여러개 (Column 개수, 자료형 자유로움)
  Row Key, Column Name, Column Value, Time Stamp 4개로 이루어짐
- Document Store Database: XML, JSON 등 쿼리문을 사용할 수 있는 반정형화된 데이터를
  Document 구조에 저장하는 DB, 각 Record는 함께 연관된 데이터를 하나의 Document에 저장
- Graph Database: 데이터 저장->Node, Node간 관계 -> 화살표, 연결된 데이터 다루는데 적합

11. Document Store DB와 Relational DB의 차이점을 쓰시오.
- Table: 모든 데이터를 주어진 Entity에 따라 하나의 Document에 저장
- Schema: 각 Document는 서로 다른 구조와 자료형의 데이터 저장 가능 (스키마 구조 자유로움)
- Scalability: Horizontal Scaling 용이함 (Scale-Out, 데이터 분산)
- Relationship: 하나의 Record에 연관된 모든 데이터는 같은 Document에 저장됨

12. 빅데이터 프로세스와 MongoDB의 활용 범위를 쓰시오.
- 데이터 수집 -> 데이터 정제, 변형

13. MongoDB 구조에 대해 쓰시오.
- Database -> Collection -> Document

14. Relational DB와 MongoDB의 차이점을 쓰시오.
- RDB: 하나의 테이블이 여러 개의 Row 가짐 // - MongoDB: 하나의 Collection이 여러 Document

15. MongoDB Document의 1 : N 패턴을 구현할 수 있는 2가지 방법을 쓰시오.

16. MongoDB Document의 M : N 패턴을 구현할 수 있는 2가지 방법을 쓰시오.

17. MongoDB Document의 트리 패턴을 구현할 수 있는 2가지 방법과 특성을 쓰시오.

18. MongoDB Document의 Dynamic Field 패턴 정의와 특성을 쓰시오.

19. Embedded Document와 Linked Document의 차이점을 5가지 쓰시오.

20. $regex를 활용한 정규식 표현 양식 3가지를 쓰시오.
{<field>: {$regex: /pattern/<options>}}//{<field>: {$regex: /pattern/, $options:'<options>'}}

**[MongoDB 쿼리문]**

1. DB 생성

use database_name

2. 현재 DB

db

3. DB 목록

show dbs

4. DB Collection Document 삽입

db.collection_name.insert( {name: "영재"} )

5. 현재 DB 삭제

db.dropDatabase()

6. Collection 생성

db.createCollection("collection_name", options)

7. Collection 목록

show collections

8. Collection 삭제

db.collection_name.drop()

9. Capped Collection 정의를 쓰시오.

Collection 내부 Document 개수와 데이터 용량 제한, 원형 큐 구조

10. Capped Collection 생성

db.createCollection("capped_collection_name", {size: 10000})

11. Collection의 Document 개수 제한 생성

db.createCollection("capped_collection_name", {size: 10000, max: 1000})

12. Capped 여부 확인

db.collection_name.isCapped()

13. 기존 Collection Capped로 전환

db.runCommand({"convertToCapped": "collection_name", size: 10000})

14. Capped Collection 옵션 변경

db.runCommand({"collMod": "collection_name", cappedSize: 10000})

15. Collation 정의를 쓰시오.

string을 비교하고 정렬하기 위해 정의한 규칙

16. Time Series Collection 정의와 특성을 쓰시오.

시간의 변화에 따른 insights를 얻을 수 있는 일련의 데이터를 효율적으로 다루기 위한 Collection

Time Series Data 작업 복잡도 감소, 쿼리 효율성 상승, 디스크 사용량 절감, 탐색 위한 I/O 감소

17. Clustered Collection 정의와 특성을 쓰시오.

그룹화된 인덱스로 생성되는 Collection, 이미 정렬된 상태로 물리적 저장, 쿼리 속도 상승, 용량절감

18. Document 1개 삽입

```
db.collection_name.insertOne( {name: "영재"} )
```

19. Document 여러 개 삽입

```
db.collection_name.insertMany( [
  {name: "영재", grade: "4"}
  {name: "혁수", grade: "4"}
] )
```

20. Document에서 _id 필드의 역할 쓰시오.

Unique Key 역할, 별도로 지정하지 않으면 자동 설정

21. Document 전체 쿼리

```
db.collection_name.find()
```

22. Document 쿼리 형식에 맞춰 출력

```
db.collection_name.find().pretty()
```

23. 'inventory' Collection에서 'qty' 필드 값이 25인 Document 쿼리

```
db.inventory.find( {qty: 25} )
```

24. 'inventory' Collection에서 'qty' 필드 값이 25 이상인 Document 쿼리

```
db.inventory.find( {qty: {$gte: 25}} )
```

25. 'inventory' Collection에서 'qty' 필드 값이 25가 아닌 Document 쿼리

```
db.inventory.find( {qty: {$ne: 25}} )
```

26. 'inventory' Collection에서 'status' 필드 값이 [A, D] 안에 있는 Document 쿼리

```
db.inventory.find( {status: {$in: ["A", "D"]}} )
```

27. 'status'가 D, 'qty'가 75 이하인 Document 쿼리

```
db.inventory.find( {status: "D", qty: {$lte: 75}} )
```

28. 'status'가 A이거나, 'qty'가 30 미만인 Document 쿼리

```
db.inventory.find( {$or: [{status: "A"}, {qty: {$lt: 30}}]} )
```

29. 'qty'가 75보다 큰 것이 아닌 Document 쿼리

```
db.inventory.find({qty: {$not: {$gt: 75}}} )
```

30. Embedded Document 쿼리 시 Dot Notation 사용이 권장되는 이유를 쓰시오.

Dot Notation을 사용하지 않을 경우 필드 순서, 개수와 함께 모든 필드가 정확히 일치해야 리턴하기 때문

31. Projection의 정의와 값에 따른 사용법

쿼리문에서 일치하는 Document에 대해 표시할 필드를 별도로 지정할 때 사용

32. 'inventory' Collection에서 'item' 필드가 paper인 Document의 size만 출력

```
db.inventory.find( {item: "paper"}, {_id: 0, size: 1} )
```

33. 'item' 필드가 paper인 Document의 값을 paperless로 수정 (첫 번째 Document만)

```
db.inventory.update( {item: "paper"}, {$set: {item: "paperless"}} )
```

33. 'item' 필드가 paper인 Document의 값을 paperless로 수정 (Document 전체)

```
db.inventory.update( {item: "paper"}, {$set: {item: "paperless"}}, {multi: true} )
```

34. Document 삭제

db.inventory.remove( {item: "paper"} )

35. Document 중 값이 일치하는 첫 번째 Document만 삭제

db.inventory.remove( {item: "paper"}, 1)

36. "restaurants" Collection에서 모든 Document에 대해 restaurant_id, name, borough, cuisine 필드 출력

db.restaurants.find( { }, {restaurant_id: 1, name: 1, borough: 1, cuisine: 1, _id: 0} )

37. "restaurants" Collection에서 모든 Document에 대해 _id 필드 제외한 …, zip code(Nested) 필드 출력

db.restaurants.find( { }, {_id: 0, restaurant_id: 1, name: 1, …, "address.zipcode": 1} )

38. borough가 "Queens"에 속하는 식당의 restaurant_id, name, borough, cuisine 필드 출력

db.restaurants.find( {borough: "Queens"}, {restaurant_id: 1, name: 1, borough: 1, cuisine: 1} )

39. borough가 "Queens"에 속하지 않는 식당의 restaurant_id, name, borough, cuisine 필드 출력

db.restaurants.find( {borough: {$ne: "Queens"}}, {restaurant_id: 1, name: 1, borough: 1, cuisine: 1} )

40. "grades.grade" 값이 "A"인 식당의 restaurant_id, name, borough, cuisine 필드 출력

db.restaurants.find( {"grades.grade": A}, {restaurant_id: 1, name: 1, borough: 1, cuisine: 1} )

41. "American" cuisine을 제공하지 않고 grades.score 값이 70 초과인 식당 출력

db.restaurants.find( {"cuisine: {$ne: "American"}, "grades.score": {$gt: 70}} )

42. borough가 "Staten Island", "Queens", "Bronx", "Brooklyn" 중 하나에 속하는 식당의 …, cuisine 필드 출력

db.restaurants.find( {"borough": {$in: ["Staten Island", "Queens", "Bronx", "Brooklyn"]}}, {…, cuisine: 1} )

43. borough가 "Staten Island", "Queens", "Bronx", "Brooklyn"에 속하지 않는 식당의 …, cuisine 필드 출력

db.restaurants.find( {"borough": {$nin: ["Staten Island", …, "Brooklyn"]}}, {…, cuisine: 1} )

44. "American" cuisine 제공하지 않고 "grades.score" 70 초과하고 borough가 "Brooklyn" 속하지 않는 식당

db.restaurants.find( {cuisine: {$ne: "American"}, "grades.score": {$gt: 70}, borough: {$ne: "Brooklyn"}} )

45. "Hamburgers" cuisine 제공하고 "grades.grade" 값이 "A"이고 borough가 "Manhattan", "Queens", "Staten Island", "Bronx"에 속하지 않는 식당의 _id 필드 제외한 restaurant_id, …, cuisine 필드 출력

db.restaurants.find( {cuisine: "Hamburgers", "grades.grade": "A", borough: {$nin: ["Manhattan", "Queens", "Staten Island", "Bronx"]}}, {_id: 0, restaurant_id: 1, …, cuisine: 1} )

- db.inventory2.insert([
    {item: "journal", qty: 25, tags: ["blank", "red"], dim_cm: [14, 21]},
    {item: "notebook", qty: 50, tags: ["red", "blank"], dim_cm: [14, 21]},
    {item: "paper", qty: 100, tags: ["red", "blank", "plain"], dim_cm: [14, 21]},
    {item: "planner", qty: 75, tags: ["blank", "red"], dim_cm: [22.85, 30]},
    {item: "paper", qty: 45, tags: ["blue"], dim_cm: [10, 15.25]}
  ])

45. tags가 red, blank인 Document (Dot Notation 사용 X)

db.inventory2.find( {tags: ["red", "blank"]} )

46. tags에 red가 포함되어 있는 Document

db.inventory2.find( {tags: "red"} )

47. dim_cm 필드의 두 번째 Element 값이 25보다 큰 Document

db.inventory2.find( {"dim_cm.1": {$gt: 25}} )

- db.inventory3.insert([
    {item: "NORWAY", instock: [{warehouse: "A", qty: 5}, {warehouse: "C", qty: 15}]},
    {item: "notebook", instock: [{warehouse: "C", qty: 5}]},
    {item: "paper", instock: [{warehouse: "A", qty: 60}, {warehouse: "B", qty: 15}]},
    {instock: [{warehouse: "A", qty: 40}, {warehouse: "B", qty: 5}]},
    {item: null, instock: [{warehouse: "B", qty: 15}, {warehouse: "C", qty: 35}]}
  ])

45. warehouse가 A, qty가 5인 Document (Dot Notation 사용 X)

db.inventory3.find( {instock: {warehouse: "A", qty: 5}} )

46. qty 값이 20 이상인 Document

db.inventory3.find( {"instock.qty": {$gte: 20}} )

47. instock 배열 첫 번째 Embedded Document 요소에 'qty' 필드 존재하고 값이 5인 Document 반환

db.inventory3.find( {"instock.0.qty": 5} )

48. instock 배열 첫 번째 Embedded Document 요소에 'qty' 필드 존재하고 값이 20 이하인 Document 반환

db.inventory3.find( {"instock.0.qty": {$lte: 20}} )

49. item 필드가 없거나 null인 Document

db.inventory3.find( {item: null} )

50. item 필드의 값이 존재하고 그 값이 null이 아닌 Document

db.inventory3.find( {item: {$ne: null}} )

51. item 필드가 존재하지 않는 Document

db.inventory3.find( {item: {$exists: false}} )

52. item 필드가 존재하는 Document (null 상관없이)

db.inventory3.find( {item: {$exists: true}} )

- db.grades.insertMany([
    {"_id": 1, name: "Alice King", classAverage: 87.33333},  // double
    {"_id": 2, name: "Bob Jenkins", classAverage: "83.52"},
    {"_id": 1, name: "Cathy Hart", classAverage: "94.06"},
    {"_id": 1, name: "Drew Williams", classAverage: NumberInt("93")},  // 32-bit Integer

53. classAverage 필드 자료형이 string인 Document

db.grades.find( {classAverage: {$type: "string"}} )

54. classAverage 필드 자료형이 number인 Document

db.grades.find( {classAverage: {$type: "number"}} )

54. classAverage 필드 자료형이 string 혹은 double인 Document

db.grades.find( {classAverage: {$type: ["string", "double"]}} )

- db.inventory.insertMany([
    {item: "journal", qty: 25, size: {h: 14, w: 21, uom: "cm"}, status: "A"},
    {item: "notebook", qty: 50, size: {h: 8.5, w: 11, uom: "in"}, status: "A"},
    {item: "paper", qty: 100, size: {h: 8.5, w: 11, uom: "in"}, status: "D"},
    {item: "planner", qty: 75, size: {h: 22.85, w: 30, uom: "cm"}, status: "D"},
    {item: "paper", qty: 45, size: {h: 10, w: 15.25, uom: "cm"}, status: "A"}
  ])

55. item 필드 값에 'paper' 키워드 포함한 Document

db.inventory.find( {item: {$regex: "paper"}} )

56. item 필드 값이 'note' 키워드로 시작하는 Document

db.inventory.find( {item: {$regex: "^note"}} )

57. item 필드 값이 'nal' 키워드로 끝나는 Document

db.inventory.find( {item: {$regex: "nal$"}} )

58. item 필드 값이 'PAPER' 키워드 포함한 Document, 대소문자 구분 X

db.inventory.find( {item: {$regex: "PAPER", $options: 'i'}} )

59. 'status' 값이 A이고, 'qty' 값이 30보다 작거나 'item' 값이 'p' 키워드로 시작하는 Document

db.inventory.find( {status: "A", $or: [{qty: {$lt: 30}}, {item: {$regex: "^p"}}]} )




```
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 }
,
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
```

60. "restaurants" Collection에서 "grades.grade" 값이 "A"인 식당의 restaurant_id, …, cuisine 필드를 출력

db.restaurants.find( {"grades.grade": "A"}, {restaurant_id: 1, …, "grades.grade": 1, cuisine: 1} )

61. "American" cuisine을 제공하지 않고 grades.score 값이 70 초과인 식당을 출력

db.restaurants.find( {cuisine: {$ne: "American"}, "grades.score": {$gt: 70}} )

62. "address.coord" 필드에서 배열 요소인 latitude = -73.85이고, longitude = 40.84인 모든 식당을 출력

db.restaurants.find( {"address.coord": [-73.85, 40.84]} )

63. "American" 또는 "Chinese" cuisine을 제공하고, "grades.score"이 60보다 크고, latitude가 -74 미만

db.restaurants.find({cuisine: {$in: ["American", "Chinese"]}, "grades.score": {$gt: 60}, "address.coord.0": {$lt: -74}})

64. "grades" 배열의 두 번째 요소가 "grade" 필드 값 "A"를 포함하고 "score"이 9, ISODate가 …인 식당

db.restaurants.find({"grades.1.grade": "A", "grades.1.score": 9, "grades.1.date": ISODate("2014…")})

65. "grades" 배열의 8번째 요소가 "score" 값 30을 초과하는 식당의 restaurant_id, name, score 출력

db.restaurants.find( {"grades.7.score": {$gt: 30}}, {restaurant_id: 1, name: 1, score: 1} )

66. "name" 필드에 "Pizza" 키워드를 포함하는 식당 출력

db.restaurants.find( {name: {$regex: "Pizza"}} )

67. "borough" 필드가 "Staten Island" 또는 "Queens"이며, "name" 필드가 "Wen" 세 글자로 시작하는 식당

db.restaurants.find( {borough: {$in: ["Staten Island", "Queens"]}, name: {$regex: "^Wen"}} )

68. 모든 Document의 "address" 필드가 "street" 필드를 포함하고 있는지 아닌지 확인

db.restaurants.find( {"address.street": {$exists: false}} )

69. 모든 Document의 "address.coord" 필드 값이 배열 형식인지 확인

db.restaurants.find( {"address.coord": {$not: {$type: "array"}}} )

70. AirBnb DB의 "reviews" Collection에서 "weekly_price" 필드가 존재하고,
"name" 필드가 "Pr" 키워드로 시작하며, "review_scores.review_scores_rating" 값이 80 이상인 Document

db.reviews.find( {"weekly_price": {$exists: true}, "name": {$regex: "^Pr"},
"reviews_scores.review_scores_rating": {$gte: 80}} )