**- Map-Reduce : 프로그래밍 적인 요소(자바스크립트, 라이브러리…) 많이 들어가면 사용. but. 왠만하면 Aggregate를 이용**

Map: 데이터 분류(키, value 지정). emit. 배열발생

Reduce: (Map에서의 배열을 넣어서 자동으로)데이터집계

|  |
| --- |
| **//1) 주문번호당 총 가격 (키값 별로 가격만 바꿔서 2번 총 4번 넣음)**  //1-1) map함수만들기  //map 함수는 데이터 분류 함수(반드시 emit을 사용한다)  var map\_function=function(){  //키: 주문번호, 값 : 가격  emit(this.cust\_id, this.price);  }  //1-2) reduce 함수만들기  //map 함수에서 emit으로 전달한 key와 value를 이용하여 집계처리  //reduce 함수의 첫 번째 매개변수로 emit에서 전달한 key가 전달됨  // 두 번째 매개변수로 emit에서 전달한 value가 전달됨  var reduce\_function=function(keyCustId, valuePrices){ // 배열로 들어옴  return Array.sum(valuePrices) // 결과물을 더 하겠다. 키가 2개라 2번 실행  }  // 맵 함수명 리듀스 함수명 출력할  db.order.mapReduce( map\_function, reduce\_function, {out: "order\_cust\_total"})  --- 결과 ---  /\* 1 \*/  {  "result" : "order\_cust\_total", // 집계 결과물이 저장된 컬렉션 이름  "timeMillis" : 207.0, //집계에 걸린 시간  "counts" : {  "input" : 2, //집계 결과물이  "emit" : 2,  "reduce" : 0,  "output" : 2  },  "ok" : 1.0,  "\_o" : {  "result" : "order\_cust\_total",  "timeMillis" : 207,  "counts" : {  "input" : 2,  "emit" : 2,  "reduce" : 0,  "output" : 2  },  "ok" : 1.0  },  "\_keys" : [  "result",  "timeMillis",  "counts",  "ok"  ],  "\_db" : {  "\_mongo" : {  "slaveOk" : true,  "host" : "localhost:27017",  "defaultDB" : "",  "\_readMode" : "commands",  "\_writeMode" : "commands"  },  "\_name" : "test"  },  "\_coll" : {  "\_mongo" : {  "slaveOk" : true,  "host" : "localhost:27017",  "defaultDB" : "",  "\_readMode" : "commands",  "\_writeMode" : "commands"  },  "\_db" : {  "\_mongo" : {  "slaveOk" : true,  "host" : "localhost:27017",  "defaultDB" : "",  "\_readMode" : "commands",  "\_writeMode" : "commands"  },  "\_name" : "test"  },  "\_shortName" : "order\_cust\_total",  "\_fullName" : "test.order\_cust\_total"  }  } |

|  |
| --- |
| //2) **제품별 주문수량 평균 집계**  // 하나의 도큐먼트(데이터)당 각각 맵이 실행  // 주문한번당 물품주문한 횟수.  // count는 구매한 횟수 .  // 즉. 평균 : qty 총 합 / count의 누적(총구매한 횟수)  //emit은 분류하고 쌓아 놓는 개념   * finalize * 평균은 집계 후 에 할 수 있음(map, reduce후..). 즉, finalize * 배열이 아님. 집계가 끝난 것이기 때문에.   결과)  /\* 1 \*/  {  "\_id" : "Bunny Boots",  "value" : {  "count" : 6.0,  "qty" : 60.0,  "average" : 10.0  }  }  /\* 2 \*/  {  "\_id" : "Sky Pole",  "value" : {  "count" : 6.0,  "qty" : 30.0,  "average" : 5.0  }  } |

**RDBMS** vs **mongoDB** 쓰이는 곳 )

RDBMS: 오래쓸때 사용. / mongoDB: DB에 접근, 검색이 많을 때 사용. 데이터 양이 많을 때.

**- Index**

인덱스 조건 : 검색이 잦은 필드. 필드마다 인덱스 부여됨으로 신중하게 결정.

createIndex : 인덱스 생성

dropIndex : 인덱스 삭제 / dropIndexs: 모든 인덱스 삭제(\_id제외)

reIndex : 인덱스 재배치(중요) -> 인덱스를 삭제 후 다시 만들기 . 주기적으로 검색이 느려질 때 쓰면 좋음.

getIndexes : 현재 컬렉션의 인덱스 정보 보기

sort(기준 인덱스로 잡으면 빨라짐) , limit : 성능 향상에 많은 도움을 준다.

**//1) Single-key Index (단일 인덱스)**

db.employees.createIndex({empno:1}); //숫자가 작은 것 부터 검색함 . -1은 내림차순

**//2) Compound Key Index (복합 인덱스)**

// 정렬의 기준을 잘 맞춰야 한다. (최대 31개 까지 가능 )

// 첫 번째로 empno로 정렬, 정렬된 결과물을 가지고 deptno 역순으로 정렬

db.employees.createIndex({empno: 1, deptno : -1});

**복합키와 싱글키 차이 문제 )**

db.employees.createIndex({empno: 1, deptno : -1});

// empno를 오름차순으로 검색 후에 그 결과물에서 deptno를 내림차순으로 놓고 검색

위와 아랜 다름.

db.employees.createIndex({empno: 1}) //empno를 이용하여 검색할 때 오름차순으로 검색

db.employees.createIndex({deptno: -1}) //deptno를 이용하여 검색 할 때 내림차순으로 검색

-복합키는 상세검색시 유리, 싱글키는 단순 순서대로 출력시 유리.

**//3) Unique Key, Non-Unique-Key**

// 기본적으로 인덱스를 만들게 되면 Non-Unique Key 옵션으로 생성된다.

// Unique Key로 설정하면 해당 필드가 유일한 키가 되기 때문에 중복된 데이터를 삽입 할 수 없다.

db.employees.createIndex({empno: 1}, {unique:true}) //empno를 unique key로 만들기

중복된 데이터가 있는 상태에서 unique로 걸 수 있는가? NO. 그러니

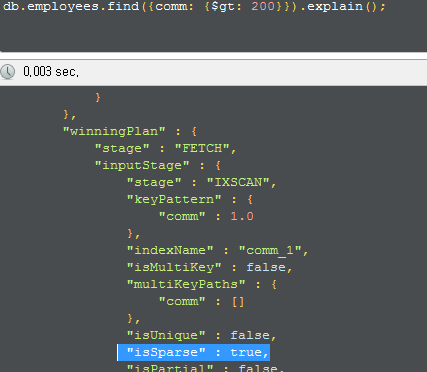
애초에 중복된 데이터가 들어있는 필드에는 unique 설정 불가능

**//4) Sharse 인덱스**

db.employees.createIndex({comm: 1}, {sparse:true})

// 검색 대상 필드 값이 전체 컬렉션에서 차지하는 밀도가 낮은 경우(필드 자체가 많이 없을 경우) 유리함

없으면 아예 검색대상에서 제외(ex: comm없으면 아예 검색x.)



explain()을 이용해 sparse확인

**//5) Partial 인덱스**

특정 필드에서 조건을 만족하는 데이터를 검색 후에 그 조건에서 다시 특정값만 가진 데이터들을 추가로 조건검색을 하는 경우 사용하면 좋다.

db.employees.find({deptno: 10, sal: {$gt: 2000}})

//deptno, ename 필드로 검색시, sal가 $gt: 500이상으로 검색이

//되면 추가 인덱스 발동

db.employees.createIndex({deptno: 1, ename: 1}, {partialFilterExpression: {sal: {$gt:500}}})

//ename은 부. deptno랑 ename 둘다 써야 됨.

db.employees.find({deptno: 10, sal:{$gt:{2000}}}) //partial 작동O

db.employees.find({deptno: 10}) //partial 작동X

db.employees.find({deptno:20, sal:{$lt:2000}}) //partial 작동X

**//지도 Geometry**

//mongoDB에서는 geoJSON을 사용 할 수 있다.

// 전세계 나라별 좌표를 JSON 형태로 만들어 놓은 자료

**//2차원 인덱스**

**//1) GeoSpatial 인덱스**

-> 2차원 형태의 지도에서 좌표를 효과적으로 찾을 수 있게 해주는 인덱스

$near: 기준점으로 가장 가까운 좌표

$center: 원

$within : 범위내의 자료들을 포함 할 때 사용된다.

$box : 네모그리기

$polygon :다각형 그리기

// 전화통화 위치추적 검색

db.tel\_pos.save({ mobile\_no: "01033336719", //통화한 전화번호

last\_pos : [

[127.1353452, 37.4576521], //통화한 위치

[127.1359081, 37.4512311],

[125.7823091, 36.3339801]

]

})

127.027577, 37.494725 //비트 좌표

37.494729, 127.028108 // cu

37.494725, 127.027577//세븐일레븐

37.494212, 127.028215//gs