

AWS Database

AWS Cloud Club

DDWU ACC Crew

정채원

RDS

관계형 데이터베이스



AWS RDS

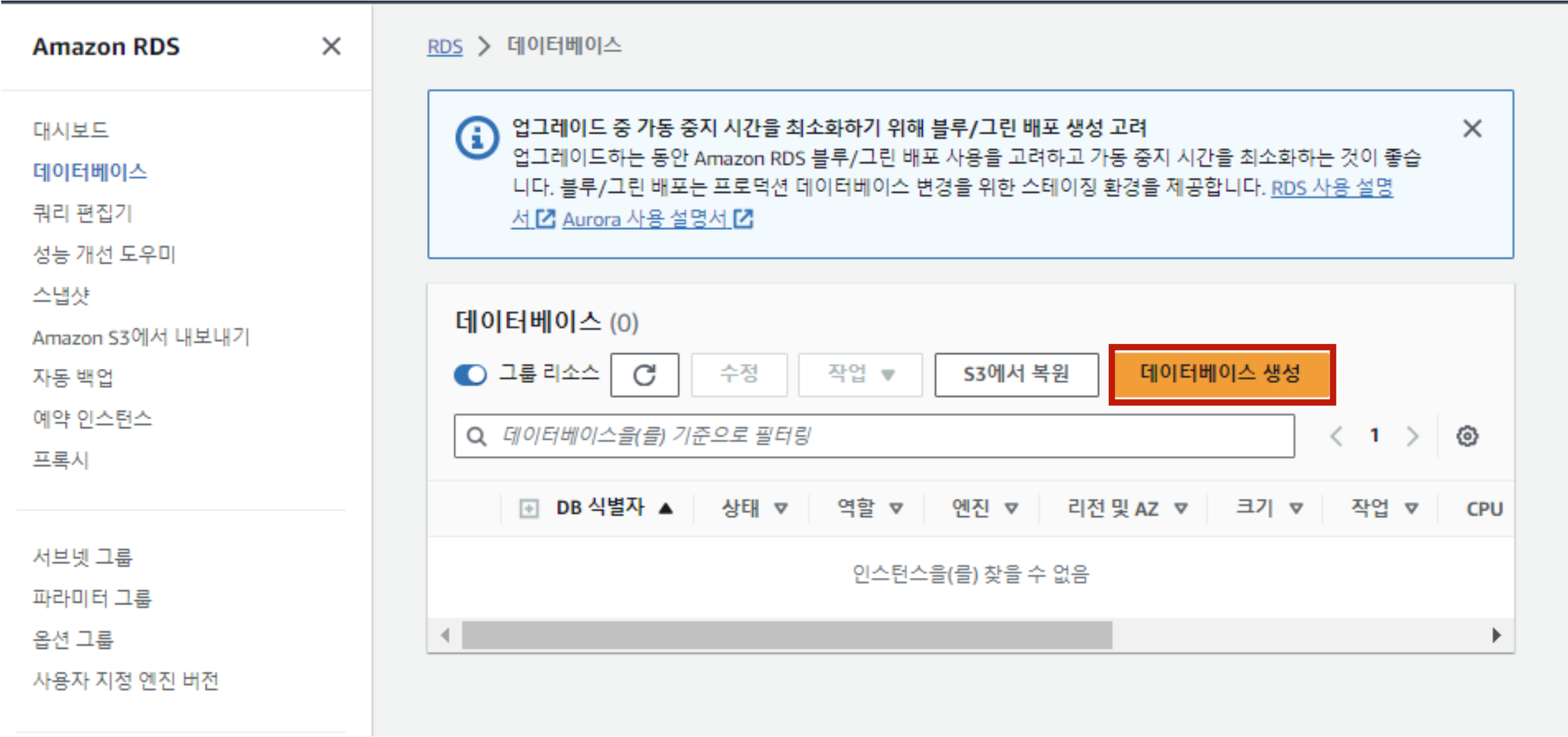
- 1) 관계형 데이터베이스
- 2) 가상머신 위에서 동작
- 3) 다양한 엔진 옵션 제공

시간 소모적인 데이터베이스 관리 작업을 처리하는 한편
비용 효율적이고 크기를 조정할 수 있는 용량을 제공

고객은 애플리케이션과 비즈니스에 좀 더 집중 가능

RDB

관계형 데이터베이스 - 데이터베이스 생성



RDB

관계형 데이터베이스 - 데이터베이스 생성

템플릿

해당 사용 사례를 충족하는 샘플 템플릿을 선택하세요.

☐ 프로덕션

고가용성 및 빠르고 일관된 성능을 위해 기본값을 사용하세요.

☐ 개발/테스트

이 인스턴스는 프로덕션 환경 외부에서 개발 용도로 마련되었습니다.

☒ 프리 티어

RDS 프리 티어를 사용하여 새로운 애플리케이션을 개발하거나, 기존 애플리케이션을 테스트하거나 Amazon RDS에서 실무 경험을 쌓을 수 있습니다. [정보](#)

가용성 및 내구성

배포 옵션 정보

아래의 배포 옵션은 위에서 선택한 엔진에서 지원하는 배포 옵션으로 제한됩니다.

☒ 다중 AZ DB 클러스터 - 신규

기본 DB 인스턴스와 읽기 가능한 예비 DB 인스턴스 2개가 있는 DB 클러스터를 생성합니다. 각 DB 인스턴스는 서로 다른 가용 영역(AZ)에 있습니다. 고가용성, 데이터 이중화를 제공하고 읽기 워크로드를 처리하기 위한 용량을 늘립니다.

☐ 다중 AZ DB 인스턴스(다중 AZ DB 클러스터 스냅샷에는 지원되지 않음)

다른 AZ에 기본 DB 인스턴스와 예비 DB 인스턴스를 생성합니다. 고가용성 및 데이터 이중화를 제공하지만 예비 DB 인스턴스는 읽기 워크로드에 대한 연결을 지원하지 않습니다.

☒ 단일 DB 인스턴스(다중 AZ DB 클러스터 스냅샷에는 지원되지 않음)

예비 DB 인스턴스가 없는 단일 DB 인스턴스를 생성합니다.

설정

DB 인스턴스 식별자 정보

DB 인스턴스 이름을 입력하세요. 이름은 현재 AWS 리전에서 AWS 계정이 소유하는 모든 DB 인스턴스에 대해 고유해야 합니다.

umc-5th-db

DB 인스턴스 식별자는 대소문자를 구분하지 않지만 'mydbinstance'와 같이 모두 소문자로 저장됩니다. 제약: 1~60자의 영숫자 또는 하이픈으로 구성되어야 합니다. 첫 번째 문자는 글자여야 합니다. 하이픈 2개가 연속될 수 없습니다. 하이픈으로 끝낼 수 없습니다.

▼ 자격 증명 설정

설정

DB 인스턴스 식별자 정보

DB 인스턴스 이름을 입력하세요. 이름은 현재 AWS 리전에서 AWS 계정이 소유하는 모든 DB 인스턴스에 대해 고유해야 합니다.

umc-5th-db

DB 인스턴스 식별자는 대소문자를 구분하지 않지만 'mydbinstance'와 같이 모두 소문자로 저장됩니다. 제약: 1~60자의 영숫자 또는 하이픈으로 구성되어야 합니다. 첫 번째 문자는 글자여야 합니다. 하이픈 2개가 연속될 수 없습니다. 하이픈으로 끝낼 수 없습니다.

▼ 자격 증명 설정

마스터 사용자 이름 정보

DB 인스턴스의 마스터 사용자에게 로그인 ID를 입력하세요.

root

1~16자의 영숫자. 첫 번째 문자는 글자여야 합니다.

☐ AWS Secrets Manager에서 마스터 보안 인증 정보 관리

Secrets Manager에서 마스터 사용자 보안 인증을 관리합니다. RDS는 사용자 대신 암호를 생성하고 수명 주기 동안 이를 관리할 수 있습니다.

📌 Secrets Manager에서 마스터 사용자 보안 인증 정보를 관리하는 경우 일부 RDS 기능은 지원되지 않습니다. [자세히 알아보기](#)

☐ 암호 자동 생성

Amazon RDS에서 사용자를 대신하여 암호를 생성하거나 사용자가 직접 암호를 지정할 수 있습니다.

마스터 암호 정보

제약 조건: 8자 이상의 인쇄 가능한 ASCII 문자. 다음은 포함할 수 없습니다. /(슬래시), `(작은따옴표), `(큰따옴표) 및 @(앳 기호).

마스터 암호 확인 정보

RDB

관계형 데이터베이스 - 데이터베이스 접속

연결 및 보안

모니터링

로그 및 이벤트

구성

유지 관리 및 백업

태그

연결 및 보안

엔드포인트 및 포트

엔드포인트

포트

3306

네트워킹

가용 영역

ap-northeast-2a

VPC

UMC-5th-Example (vpc-)

서브넷 그룹

umc-db-subnet

서브넷

subnet-

subnet-

네트워크 유형

IPv4

보안

VPC 보안 그룹

umc-5th-security (sg-038e04c91c3709f02)

✓ 활성화

default (sg-041e98877af46420e)

✓ 활성화

퍼블릭 액세스 가능

예

인증 기관 정보

rds-ca-ecc384-g1

인증 기관 날짜

May 21, 2121, 02:38 (UTC+09:00)

DB 인스턴스 인증서 만료 날짜

October 14, 2024, 01:12 (UTC+09:00)

Data Sources

Drivers

DDL Mapping

Project Data Sources

@umc-5th-db

General

Options

SSH/SSL

Schemas

Advanced

Name:

Comment:

Connection type:

default

Driver:

MySQL

More Options

Host:

amazonaws.com

Port:

3306

Authentication:

User & Password

User:

root

Password:

Save:

Forever

Database:

URL:

jdbc:mysql://umc-5th-.rds.amazonaws.com:3306/

Succeeded

Copy

DBMS: MySQL (ver. 8.0.33)

Case sensitivity: plain=exact, delimited=exact

Driver: MySQL Connector/J (ver. mysql-connector-java-8.0.25 (Revision: 08be9e9b4cba6aa115f9b27b215887af40b159e0), JDBC4.2)

Ping: 141 ms

SSL: yes

Test Connection

MySQL 8.0.33

OK

Cancel

Apply

DynamoDB

NoSQL 완전관리형 데이터베이스



AWS DynamoDB

- 1) 서버리스 기반 **Key-Value NoSQL** 데이터베이스
- 2) **HTTP**로 통신
- 3) 보안, 백업, 다중 리전 복제, 인메모리 캐시 지원

적은 비용으로 고성능 애플리케이션 실행 가능

완전 관리형 서비스이므로 운영 부담이 발생 **X**

DynamoDB

NoSQL 완전관리형 데이터베이스

DynamoDB 구성요소

1) 테이블

2) 항목

3) 속성

People

```
{
  "PersonID": 101,
  "LastName": "Smith",
  "FirstName": "Fred",
  "Phone": "555-4321"
}

{
  "PersonID": 102,
  "LastName": "Jones",
  "FirstName": "Mary",
  "Address": {
    "Street": "123 Main",
    "City": "Anytown",
    "State": "OH",
    "ZIPCode": 12345
  }
}
```

Music

```
{
  "Artist": "The Acme Band",
  "SongTitle": "Still in Love",
  "AlbumTitle": "The Buck Starts Here",
  "Price": 2.47,
  "Genre": "Rock",
  "PromotionInfo": {
    "RadioStationsPlaying": [
      "KHCR",
      "KQBX",
      "WTNR",
      "WJJH"
    ],
    "TourDates": {
      "Seattle": "20150622",
      "Cleveland": "20150630"
    },
    "Rotation": "Heavy"
  }
}
```

DynamoDB

NoSQL 완전관리형 데이터베이스

DynamoDB 프라이머리 키

1) 파티션 키

: 하나의 속성으로 구성되는 단순 기본 키

2) 파티션 키 및 정렬 키

: 복합 기본 키로 지칭

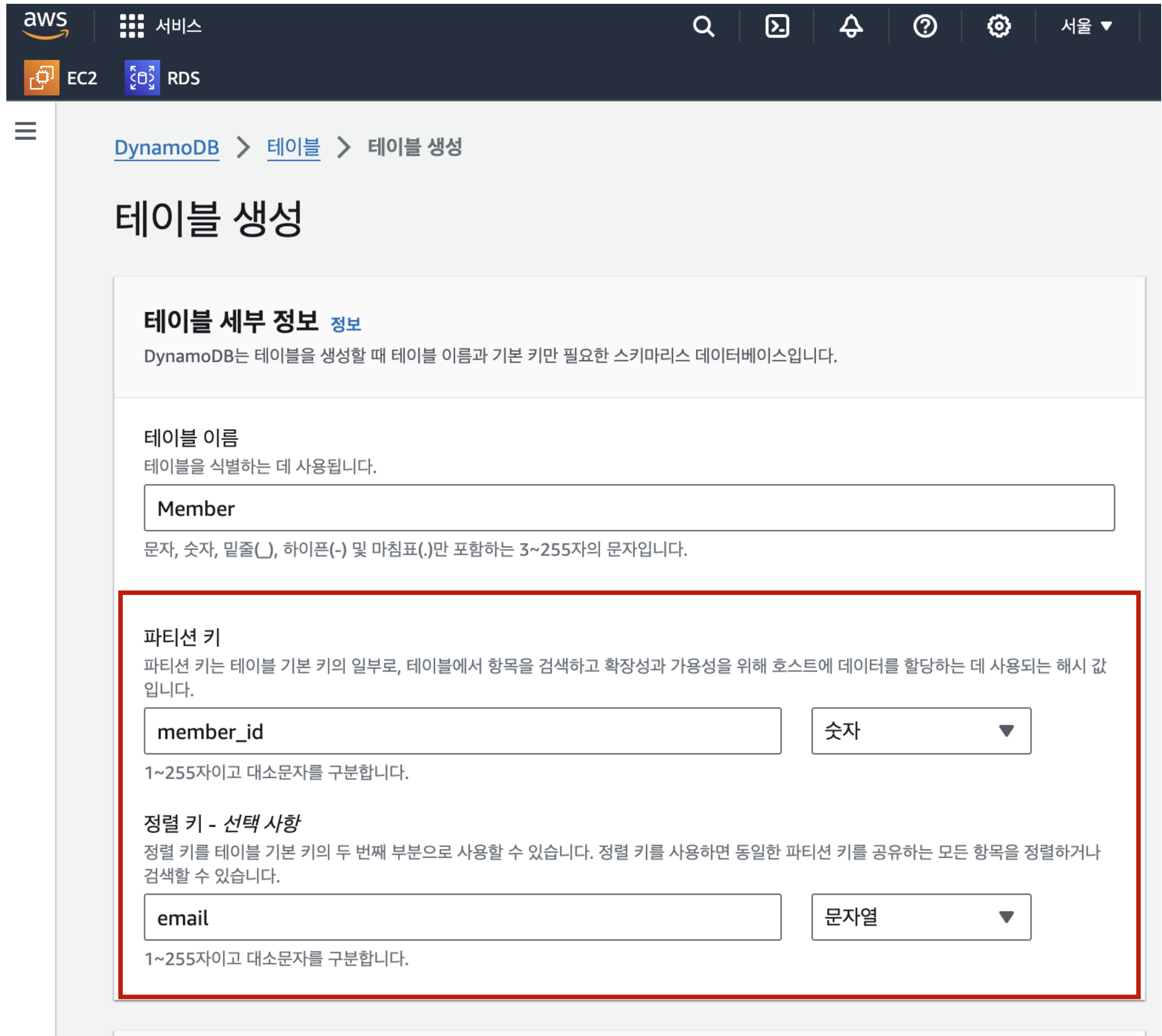
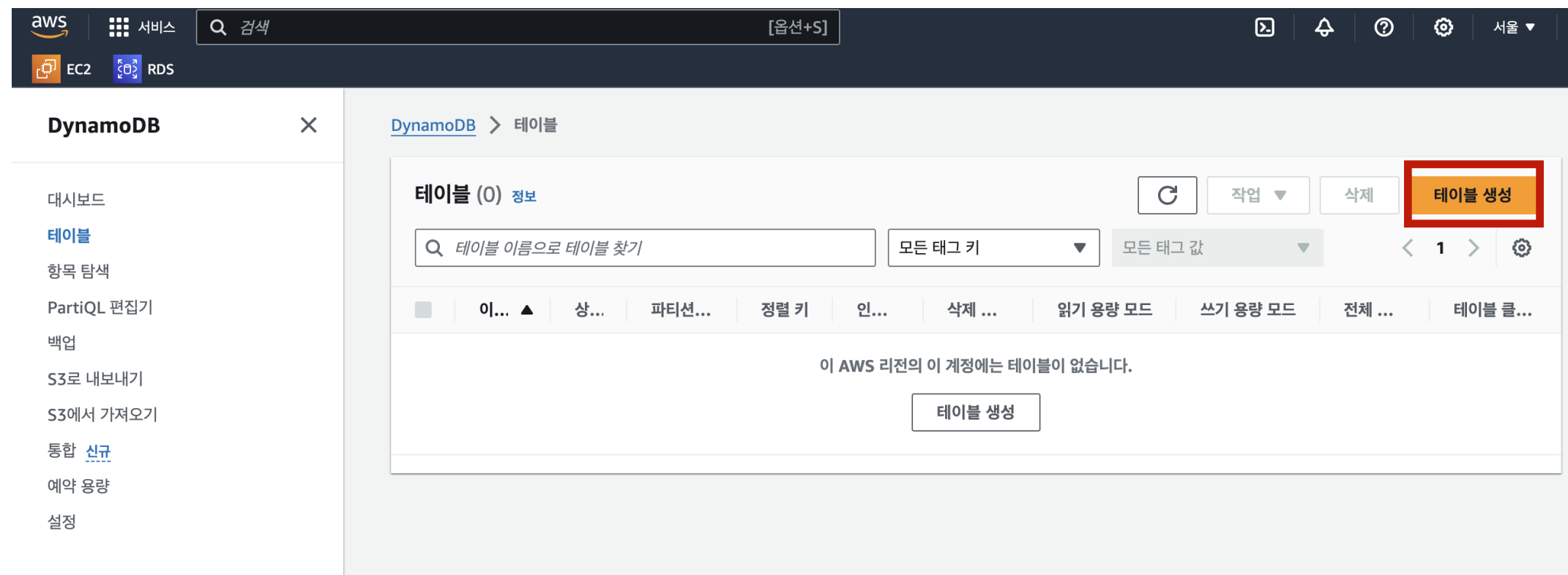
: 첫 번째 속성은 파티션키, 두 번째 속성은 정렬키

Music

```
{  
  "Artist": "No One You Know",  
  "SongTitle": "My Dog Spot",  
  "AlbumTitle": "Hey Now",  
  "Price": 1.98,  
  "Genre": "Country",  
  "CriticRating": 8.4  
}
```


DynamoDB

NoSQL 완전관리형 데이터베이스 - 데이터베이스 생성



DynamoDB

NoSQL 완전관리형 데이터베이스 - 데이터베이스 생성

읽기/쓰기 용량 설정 정보

용량 모드



프로비저닝됨

읽기/쓰기 용량을 미리 할당하는 방식으로 요금을 관리하고 최적화합니다.



온디맨드

애플리케이션이 수행하는 실제 읽기 및 쓰기에 대해 요금을 지불하는 방식으로 결제를 간소화합니다.

테이블 설정



기본 설정

테이블을 생성하는 가장 빠른 방법입니다. 지금 또는 테이블이 생성된 후에 이러한 설정을 수정할 수 있습니다.



설정 사용자 지정

이 고급 기능을 사용하여 DynamoDB를 사용자의 필요에 더 적합하게 만들 수 있습니다.

다) 처리량 테이블과 보조 인덱스를 지정하여 처리량 사용량과 비용을 제한할 수 있습니다. 기본적으로 온디맨드 처리량은 기본 DynamoDB 테이블 할당량에 의해서만 제한됩니다.

테이블 클래스

워크로드 요구 사항 및 데이터 액세스 패턴에 따라 테이블 비용을 최적화하려면 테이블 클래스를 선택합니다.

테이블 클래스 선택



DynamoDB Standard

기본 범용 테이블 클래스입니다. 자주 액세스하는 데이터를 저장하고 처리량(읽기 및 쓰기)이 주요 테이블 비용인 대부분의 테이블에 권장됩니다.



DynamoDB Standard-IA

자주 액세스하지 않는 데이터를 저장하고 스토리지가 주요 테이블 비용인 테이블에 권장됩니다.

DynamoDB

NoSQL 완전관리형 데이터베이스 - 데이터베이스 삽입

(1) Member 테이블

DynamoDB > 항목 탐색: Member > 항목 생성

항목 생성

양식JSON 뷰

항목의 속성을 추가, 제거 또는 편집할 수 있습니다. 최대 32 수준까지 다른 속성 안에 속성을 중첩할 수 있습니다. 자세히 알아보기

속성		새 속성 추가 ▼	
속성 이름	값	유형	
member_id - 파티션 키	0	숫자	
email - 정렬 키	test@gmail.com	문자열	
address	필드 삽입 ▼	맵	제거
street	Seongbuk	문자열	제거
city	Seoul	문자열	제거
zipcode	10473	문자열	제거
name	AWS Cloud Club	문자열	제거

```
1 {
2   "member_id": {
3     "N": "0"
4   },
5   "email": {
6     "S": "test@gmail.com"
7   },
8   "address": {
9     "M": {
10      "city": {
11        "S": "Seoul"
12      },
13      "street": {
14        "S": "Seongbuk"
15      },
16      "zipcode": {
17        "S": "10473"
18      }
19    }
20  },
21  "name": {
22    "S": "AWS Cloud Club"
23  }
24 }
```

DynamoDB

NoSQL 완전관리형 데이터베이스 - 데이터베이스 삽입

(2) Post 테이블

[DynamoDB](#) > [항목 탐색: Post](#) > 항목 생성

항목 생성

양식JSON 뷰

항목의 속성을 추가, 제거 또는 편집할 수 있습니다. 최대 32 수준까지 다른 속성 안에 속성을 중첩할 수 있습니다. [자세히 알아보기](#)

속성		새 속성 추가 ▼	
속성 이름	값	유형	
post_id - 파티션 키	0	숫자	
created_at - 정렬 키	2024-05-20	문자열	
content	Nice to meet you	문자열	제거
member_id	0	숫자	제거

취소

항목 생성

```
1 {
2   "post_id": {
3     "N": "0"
4   },
5   "created_at": {
6     "S": "2024-05-20"
7   },
8   "content": {
9     "S": "Nice to meet you"
10  },
11  "member_id": {
12    "N": "0"
13  }
14 }
```

DynamoDB

NoSQL 완전관리형 데이터베이스 - 데이터베이스 조회

테이블 (2) ×

모든 태그 키 ▼

모든 태그 값 ▼

테이블 이름으로 테이블 찾기

< 1 > ⚙

☒ Member

☐ Post

Member

자동 미리 보기

테이블 세부 정보 보기

▼ 항목 스캔 또는 쿼리

☒ 스캔

☐ 쿼리

테이블 또는 인덱스 선택

속성 프로젝션 선택

테이블 - Member ▼

모든 속성 ▼

▶ 필터

실행

재설정

✓ 완료. 사용된 읽기 용량 단위: 2

반환된 항목 (1)

↺

작업 ▼

항목 생성

< 1 > ⚙

<input type="checkbox"/>	member... ▼	email (문자열) ▼	address ▼	name ▼
<input type="checkbox"/>	0	test@gmail.com	{ "zipcode" :...	AWS Cloud Club

DocumentDB

완전관리형 기본 JSON 도큐먼트 데이터베이스



AWS DocumentDB

- 1)클라우드에서 **MongoDB** 호환 데이터베이스 쉽게 운영 가능
- 2) 데이터를 문서(대부분 json) 형태로 저장
- 3) 보안, 백업, 다중 리전 복제, 인메모리 캐시 지원

애플리케이션을 신속하게 개발하고 개선 가능

인프라를 관리하지 않고도 규모와 관계 없이 문서 워크로드 운영

DocumentDB

완전관리형 기본 JSON 도큐먼트 데이터베이스 - 데이터 삽입

(1) 데이터베이스 스키마

```
1  import mongoose from "mongoose";
2  import jwt from "jsonwebtoken";
3  import dotenv from "dotenv";
4
5  const envFile = process.env.NODE_ENV === "prod" ? ".env.prod" : ".env.dev";
6  dotenv.config({ path: envFile }); // .env 파일 사용 (환경 변수 관리)
7  const secret = process.env.JWT_SECRET;
8
9  const { Schema } = mongoose;
10 const {
11   Types: { ObjectId },
12 } = Schema;
13
14 const userSchema = new Schema(
15   {
16     email: {
17       type: String,
18       required: true,
19       unique: true,
20       index: true,
21     },
22     name: {
23       type: String,
24       required: true,
25     },
26     nickname: {
27       type: String,
28       required: true,
29       unique: true,
30     },
```

```
7
8   const chatRoomSchema = new Schema(
9     {
10       // 채팅 참여자
11       joiner: [
12         {
13           type: ObjectId,
14           ref: "User",
15         },
16       ],
17       // 채팅방 종류
18       type: {
19         type: String,
20         /*
21          free = 자유 라이브
22          ing = 나 지금 헌혈 중 라이브
23          story = 헌혈 이야기 공유 라이브
24          crew = 크루
25          blood = 지정헌혈
26          */
27         enum: ["free", "ing", "story", "crew", "blood"], //
28         required: true,
29       },
30       // 채팅방 이름
31       title: {
32         type: String,
33         required: true,
34       },
35       // 생성일
36       created_at: {
37         type: Date,
38         default: Date.now,
39         required: true,
40       },
```

```
8   const chatSchema = new Schema(
9     {
10       // 채팅방
11       chatRoom: {
12         type: ObjectId,
13         required: true,
14         ref: "ChatRoom",
15       },
16       // 채팅 작성자
17       writer: {
18         id: {
19           type: ObjectId,
20           required: true,
21           ref: "User",
22         },
23         nickname: {
24           type: String,
25           required: true,
26         },
27       },
28       // 메시지
29       message: {
30         type: String,
31         required: false,
32       },
33       // 이미지 (S3 업로드 URL)
34       image: {
35         type: String,
36         required: false,
37       },
38       // 생성일
39       created_at: {
40         type: Date,
41         default: Date.now,
42         required: false,
43       },
44     },
45     { versionKey: false } // 데이터 삽입 시 __v 컬럼 생성 X);
```

DocumentDB

완전관리형 기본 JSON 도큐먼트 데이터베이스 - 데이터 삽입

(2) 데이터 삽입

```
// 댓글 작성하기
export const postComment = async (req, res, next) => {
  try {
    const postId = req.params.id;
    const comment = req.body.comment;
    const { _id, email } = req.user; // body에서 가져오지 않고 댓글 내용과 작성자 저장하기
    const commenter = await User.findById(_id);

    if (comment == "") {
      const error = customErrorResponse(status.BAD_REQUEST, "댓글을 작성해주세요.");
      return next(res.send(error));
    }

    const newComment = new Comment({
      post: postId,
      commenter: {
        id: _id,
        nickname: commenter.nickname
      },
      comment,
    });

    await newComment.save();
    return res.send(response(status.SUCCESS, newComment));
  } catch (error) {
    return res.send(errResponse(status.INTERNAL_SERVER_ERROR))
  }
};
```

```
/*
 * API No. 1
 * API Name : 채팅방 생성 API
 * [POST] /chatRoom
 */
export const postChatRoom = async (req, res, next) => {
  try {
    const { _id, email } = req.user;
    const newChatRoom = new ChatRoom({
      joiner: _id,
      type: req.body.type,
      title: req.body.title,
    });

    const savedChatRoom = await newChatRoom.save();
    const io = req.app.get("io"); // io 가져오기

    /*
     io.emit 현재 소켓 서버에 접속한 모든 사용자에게 메시지 전송

     1. io.of("/room") -> /room 네임스페이스의 Socket.IO 객체 반환
     2. emit("newRoom", savedChatRoom) -> newRoom 이벤트 발생 + 방 정보 savedChatRoom 클라이언트에게 전송
     3. 클라이언트에서는 채팅방 목록 업데이트
    */
    io.of("/chatRoom").emit("newRoom", savedChatRoom);
    const chatRoomURI = "/chatRoom/" + savedChatRoom._id;

    console.log("chatRoomURI:" + chatRoomURI);
    return res.send(response(status.SUCCESS, chatRoomURI)); // 채팅방 주소 반환
  } catch (err) {
    console.log(err);
    return res.send(errResponse(status.INTERNAL_SERVER_ERROR));
  }
};
```

DocumentDB

완전관리형 기본 JSON 문서 데이터베이스 - 데이터 삽입

(2) 데이터 조회

```
/*
 * API No. 5
 * API Name : 지정헌혈글 좋아요 API
 * [POST] /blood/:bloodId/like
 */
export const likeBlood = async (req, res, next) => {
  const { _id, email } = req.user;
  const bloodId = req.params.bloodId;

  try {
    const blood = await Blood.findOneAndUpdate(
      {
        _id: bloodId,
        likes: { $ne: _id }, // 좋아요 누르지 않은 경우에만 업데이트
      },
      {
        $push: { likes: _id }, // likes에 현재 사용자의 _id 추가
        $inc: { likeCount: 1 }, // 좋아요 개수 1 증가
      },
      { new: true } // 업데이트 된 문서 반환
    );

    if (!blood) {
      return res.send(errResponse(status.BLOOD_ALREADY_LIKED));
    }

    return res.send(response(status.SUCCESS, "지정헌혈글 좋아요 성공"));
  } catch (err) {
    console.log(err);
    return res.send(errResponse(status.INTERNAL_SERVER_ERROR));
  }
};
```

```
/*
 * API No. 2
 * API Name : 내가 참여중인 채팅방 전체 조회 API
 * [GET] /chatRoom/all
 */
export const getChatRooms = async (req, res, next) => {
  try {
    const { _id, email } = req.user;
    const chatRooms = await ChatRoom.find({ joiner: _id });

    if (!chatRooms || chatRooms.length === 0) {
      return res.send(
        errResponse(
          status.BAD_REQUEST,
          "해당 유저가 참여하고 있는 채팅방이 존재하지 않습니다."
        )
      );
    }

    const chatRoomsWithChat = [];

    // 채팅방 별 가장 최근 메시지 or 이미지 찾기
    for (const chatRoom of chatRooms) {
      const recentChat = await Chat.findOne({ chatRoom: chatRoom._id })
        .sort({ created_at: -1 })
        .select("message image");

      chatRoomsWithChat.push({
        chatRoom: chatRoom.toObject(),
        recentChat: recentChat ? recentChat.message || recentChat.image : null,
      });
    }
    return res.send(response(status.SUCCESS, { chatRoomsWithChat }));
  } catch (err) {
    console.log(err);
    return res.send(errResponse(status.INTERNAL_SERVER_ERROR));
  }
};
```

DocumentDB

완전관리형 기본 JSON 문서 데이터베이스

AWS DynamoDB

- 키밸류 **vs** 문서
- 하이퍼 스케일 (용량에 상관없이 서비스 가능)
- 최대 **400KB** 아이템 크기
- **글로벌 테이블과 같은 차별화된 기능 제공**
- 관리 및 가격 책정
 - 서버리스
 - 자동 샤드 및 오토스케일링
 - 가격 책정 (온디맨드 **vs** 프로비전)
- **프리티어 사용 가능**

VS

AWS DocumentDB

- 문서
- 쉬운 확장
- 풍부한 조회 연산자
- 최대 **16MB** 문서 크기
- **MongoDB API 호환**, 손쉬운 마이그레이션
- 관리 및 가격 책정
 - 인스턴스 기반
 - 스토리지 자동 확장
 - 가격 (인스턴스, **IO**, 스토리지, 백업 스토리지)