AWS Database

AWS Cloud Club

DDWU ACC Crew

정채원

RDS

관계형 데이터베이스

AWS RDS

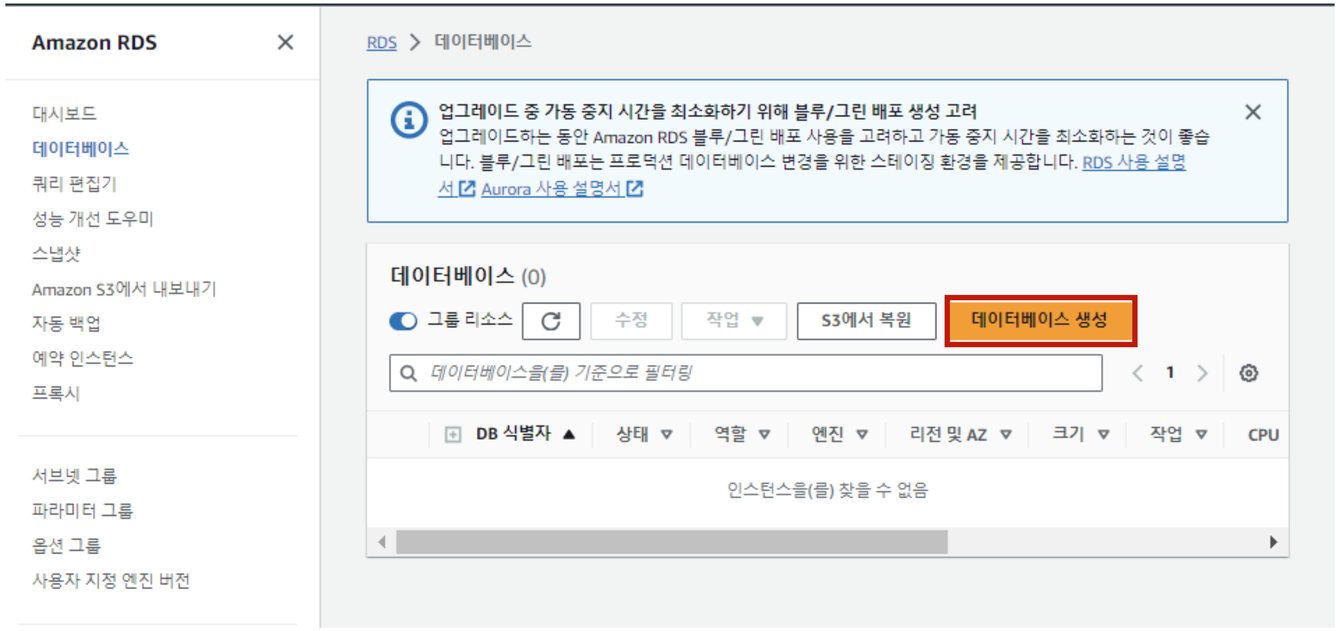
1) 관계형 데이터베이스
2) 가상머신 위에서 동작
3) 다양한 엔진 옵션 제공

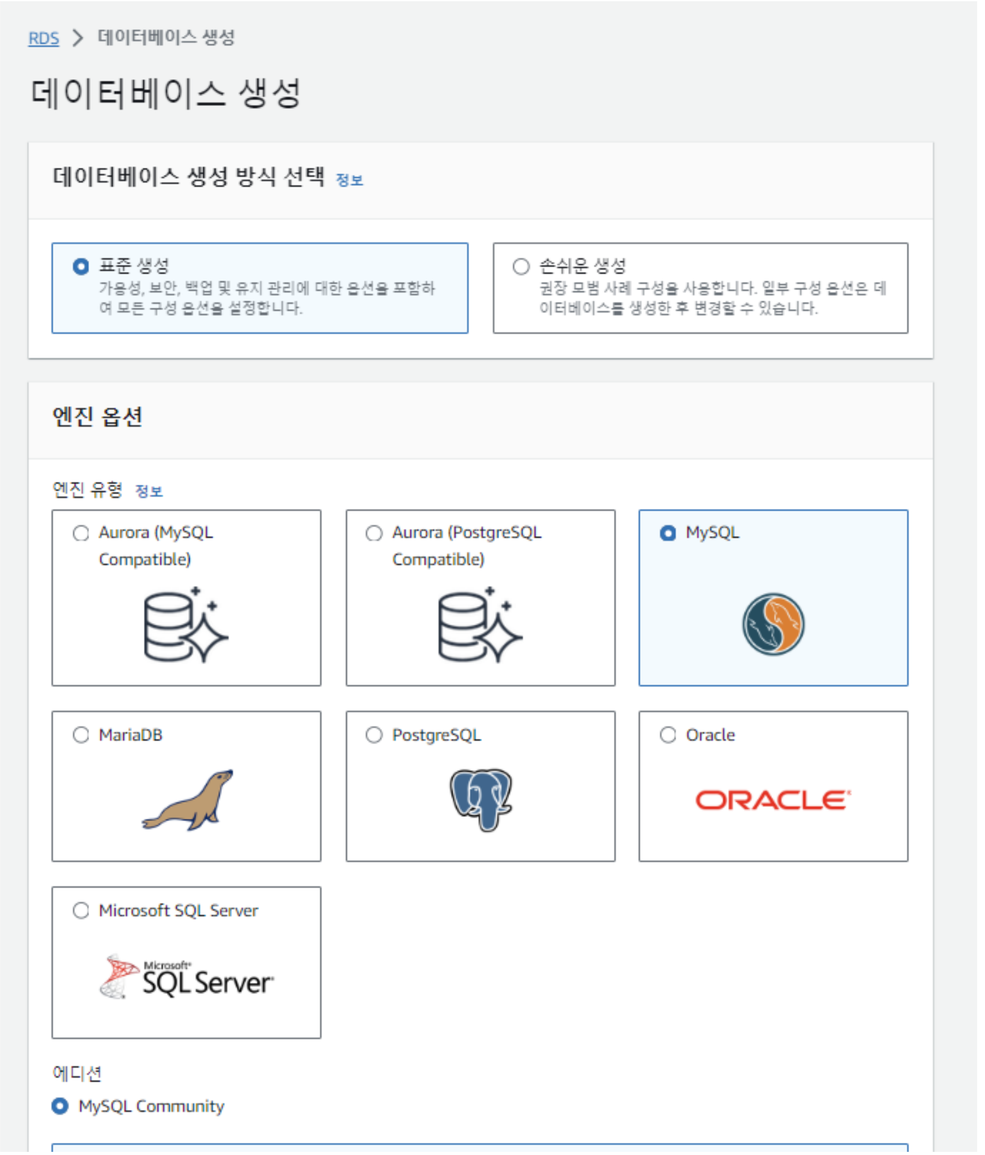
시간 소모적인 데이터베이스 관리 작업을 처리하는 한편
비용 효율적이고 크기를 조정할 수 있는 용량을 제공

고객은 애플리케이션과 비즈니스에 좀 더 집중 가능

RDB

관계형 데이터베이스 - 데이터베이스 생성



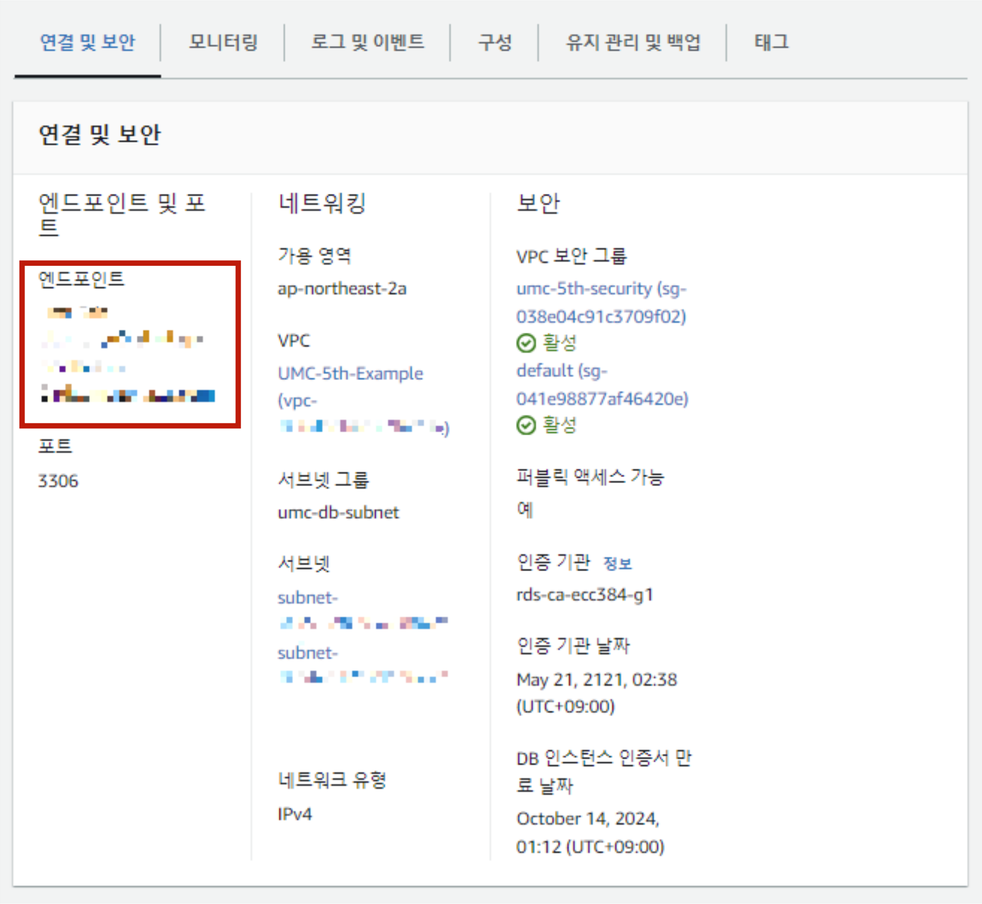


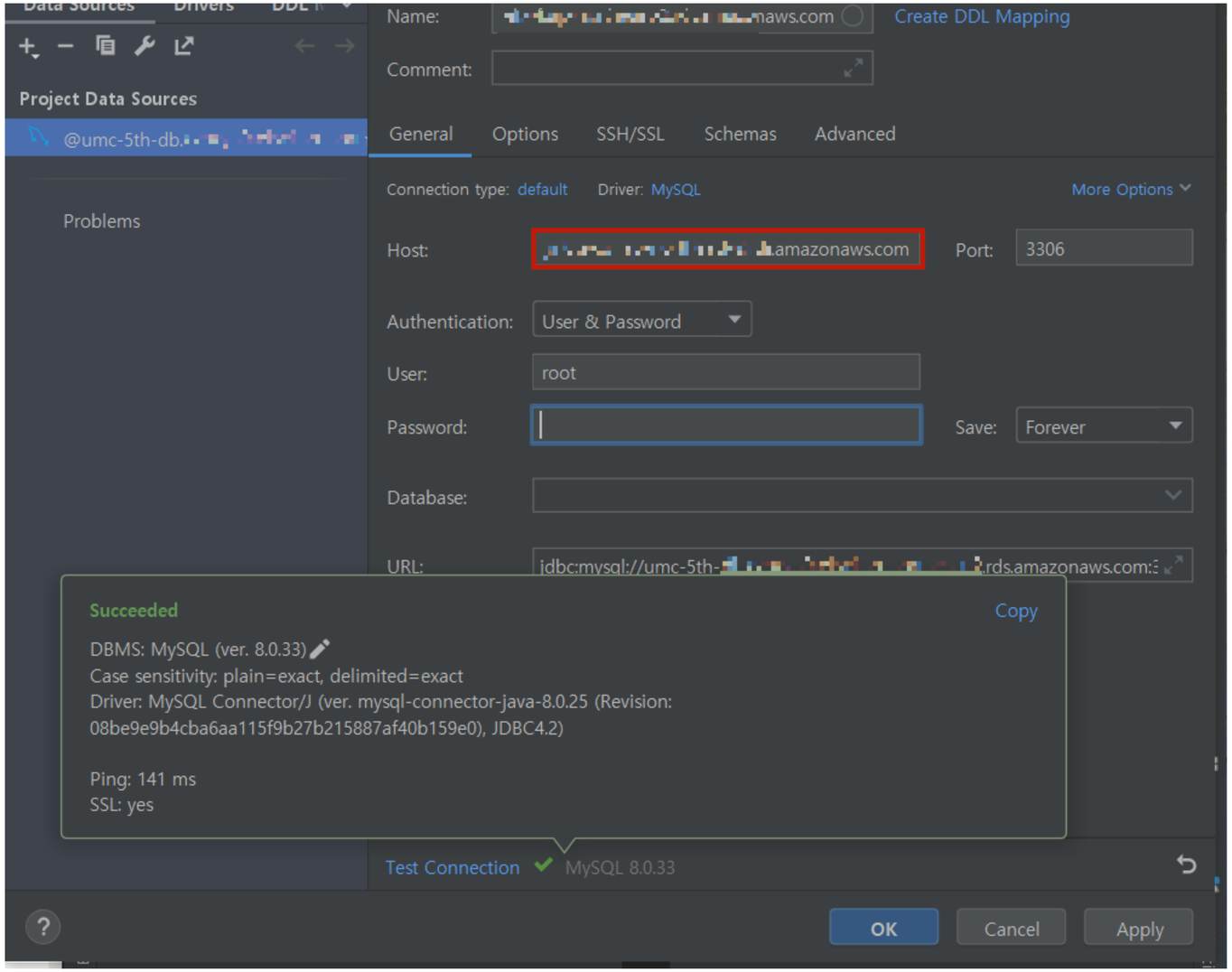
RDB

관계형 데이터베이스 - 데이터베이스 생성

RDB

관계형 데이터베이스 - 데이터베이스 접속





DynamoDB

NoSQL 완전관리형 데이터베이스

AWS DynamoDB

1) 서버리스 기반 **Key-Value NoSQL** 데이터베이스
2) HTTP로 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

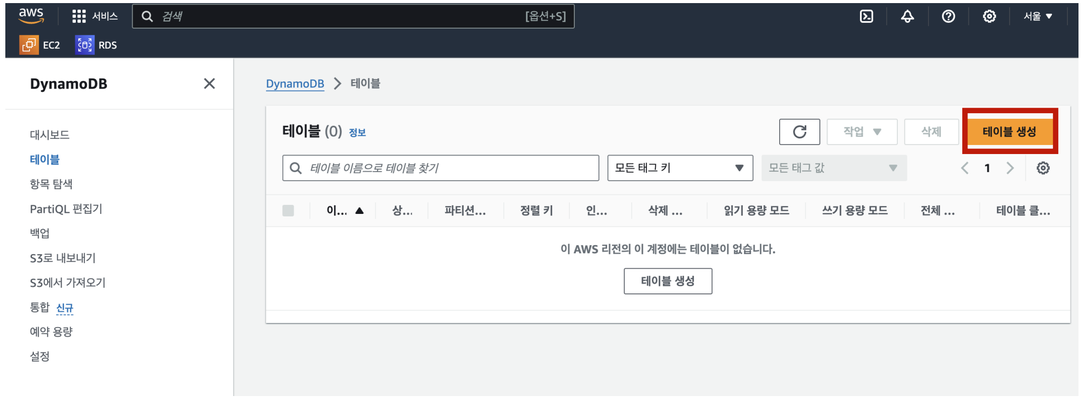
: 첫 번째 속성은 파티션키, 두 번째 속성은 정렬키

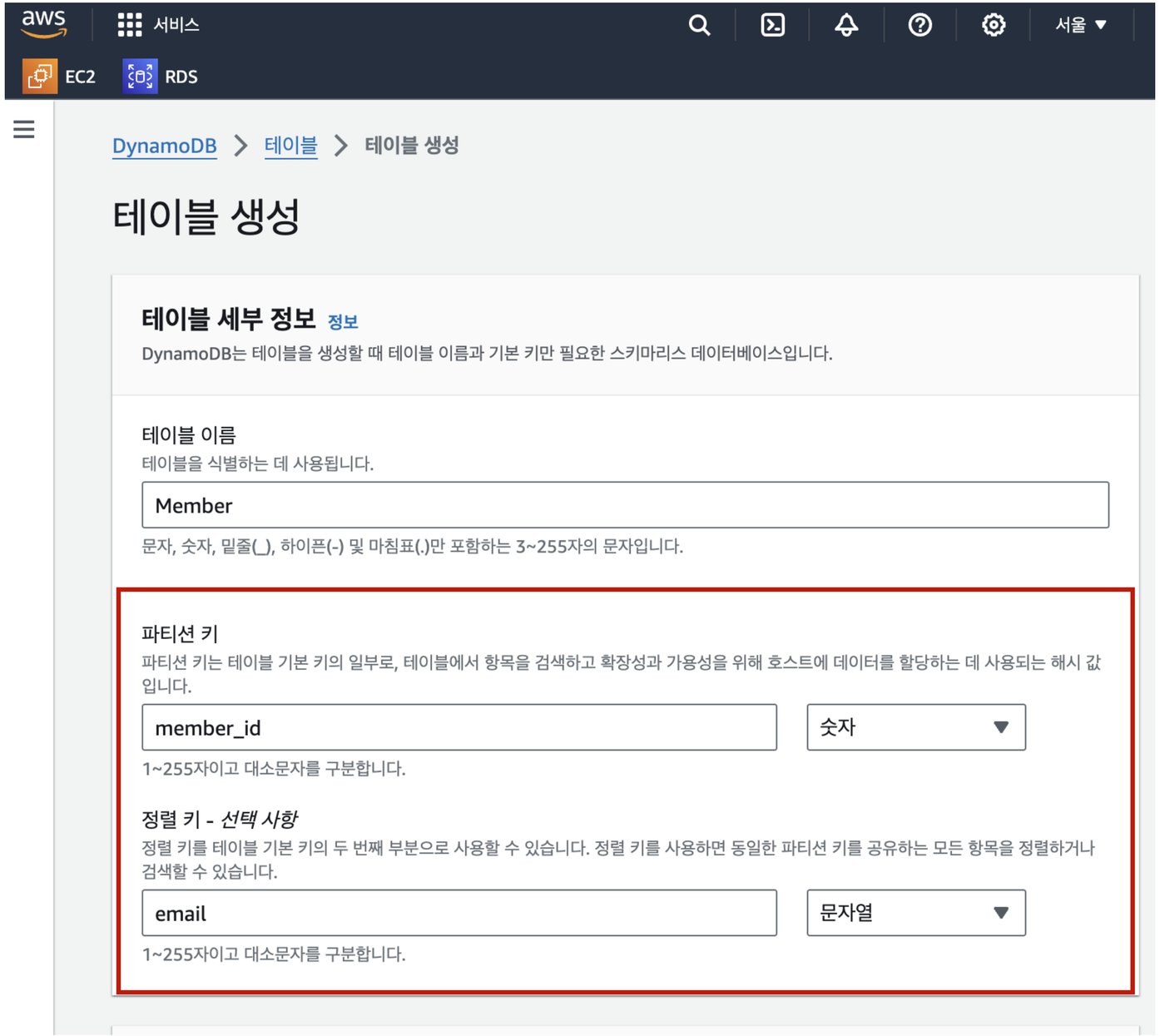
Music

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

DynamoDB

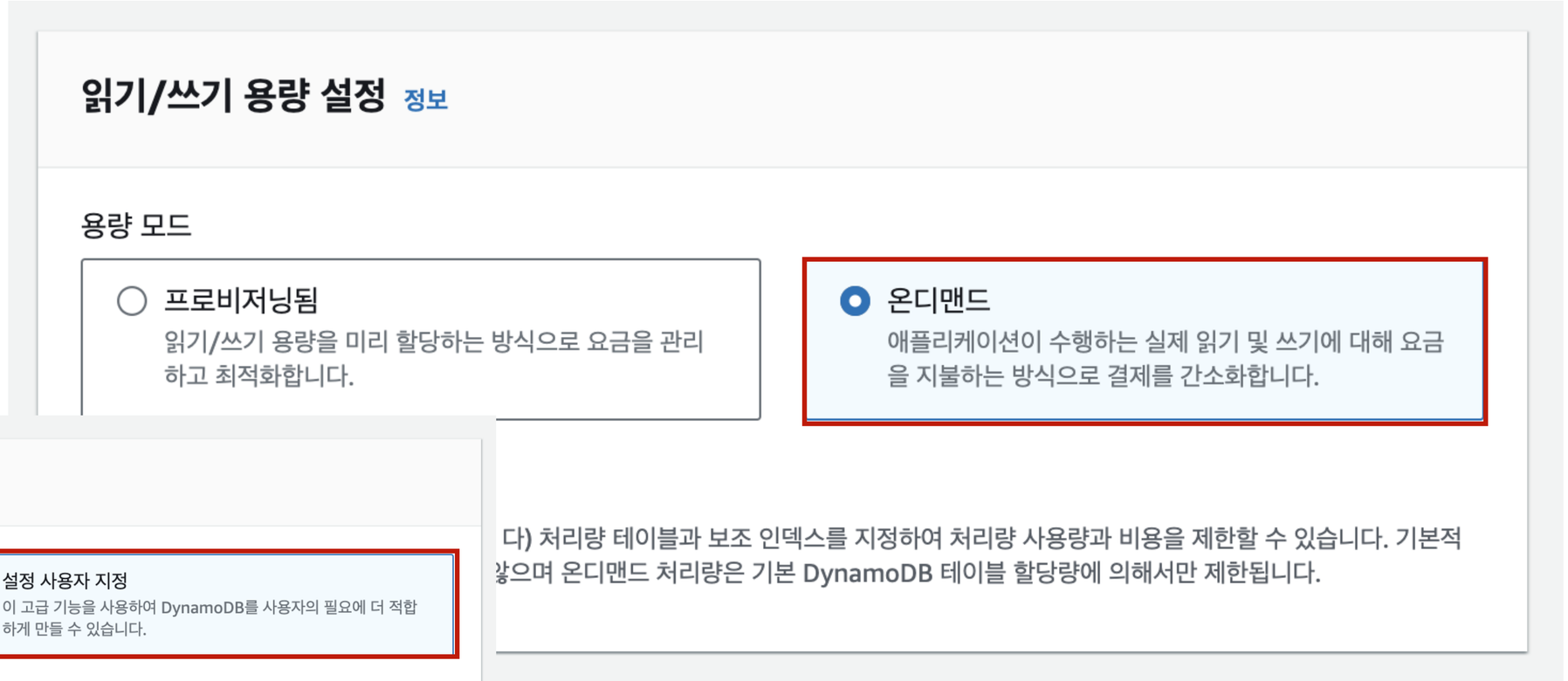
NoSQL 완전관리형 데이터베이스 - 데이터베이스 생성

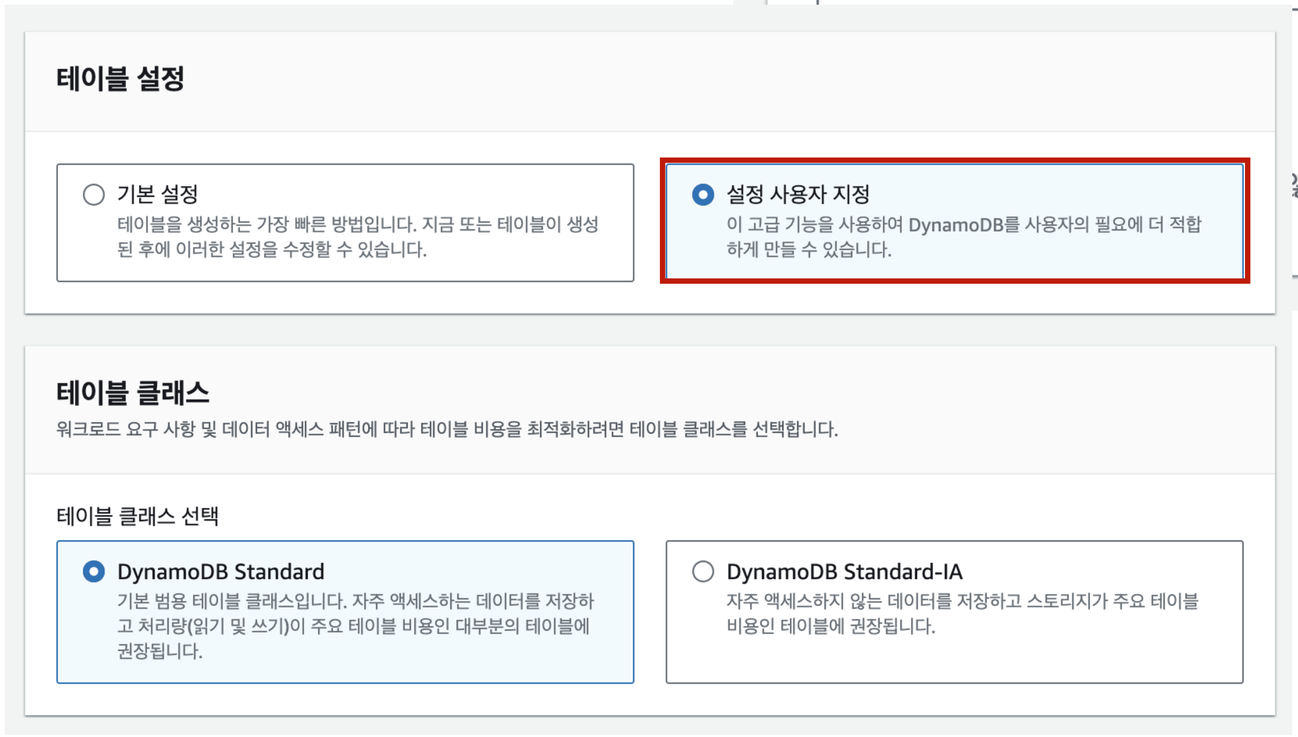




DynamoDB

NoSQL 완전관리형 데이터베이스 - 데이터베이스 생성

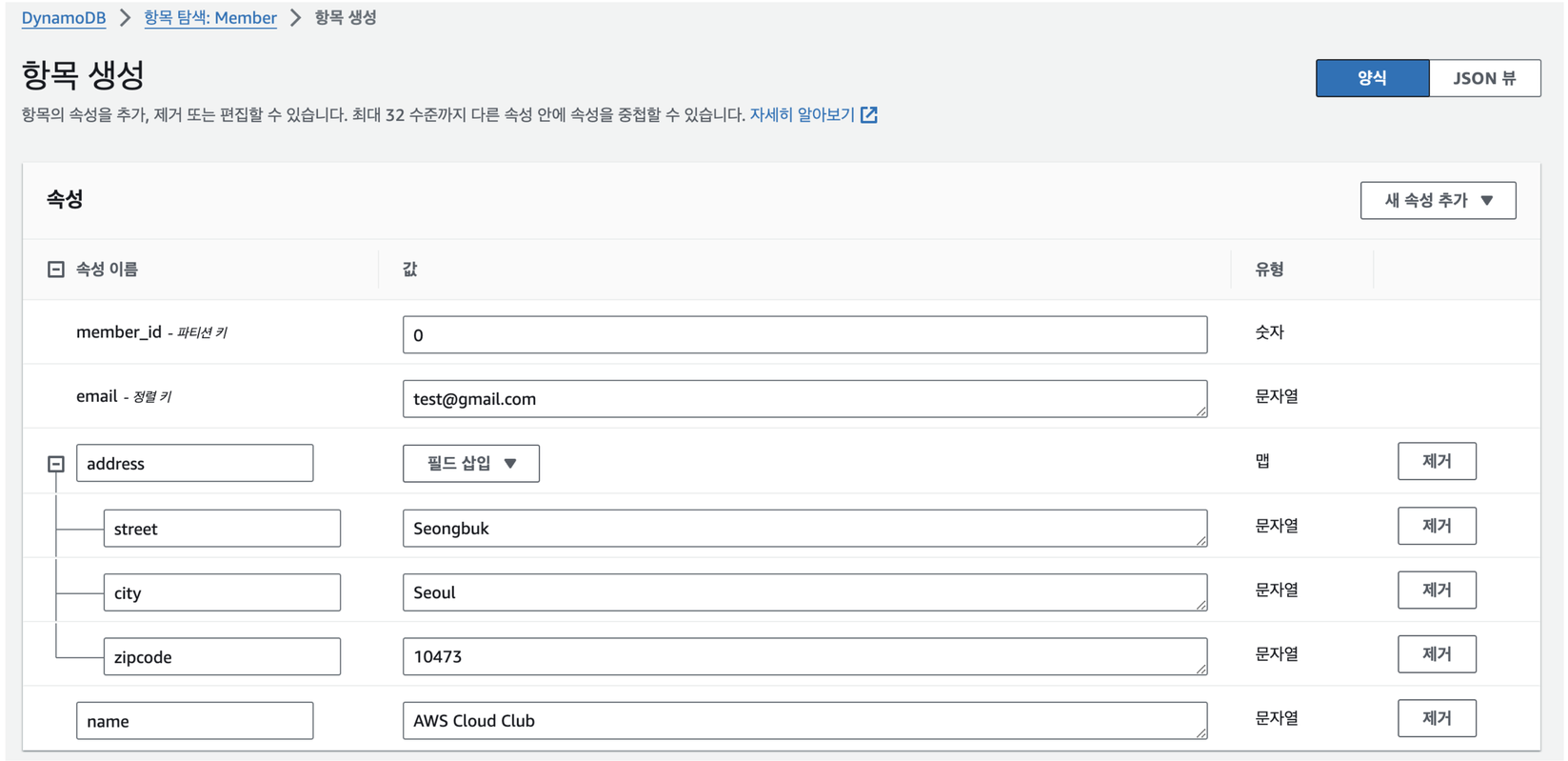




DynamoDB

NoSQL 완전관리형 데이터베이스 - 데이터베이스 삽입

(1) Member 테이블

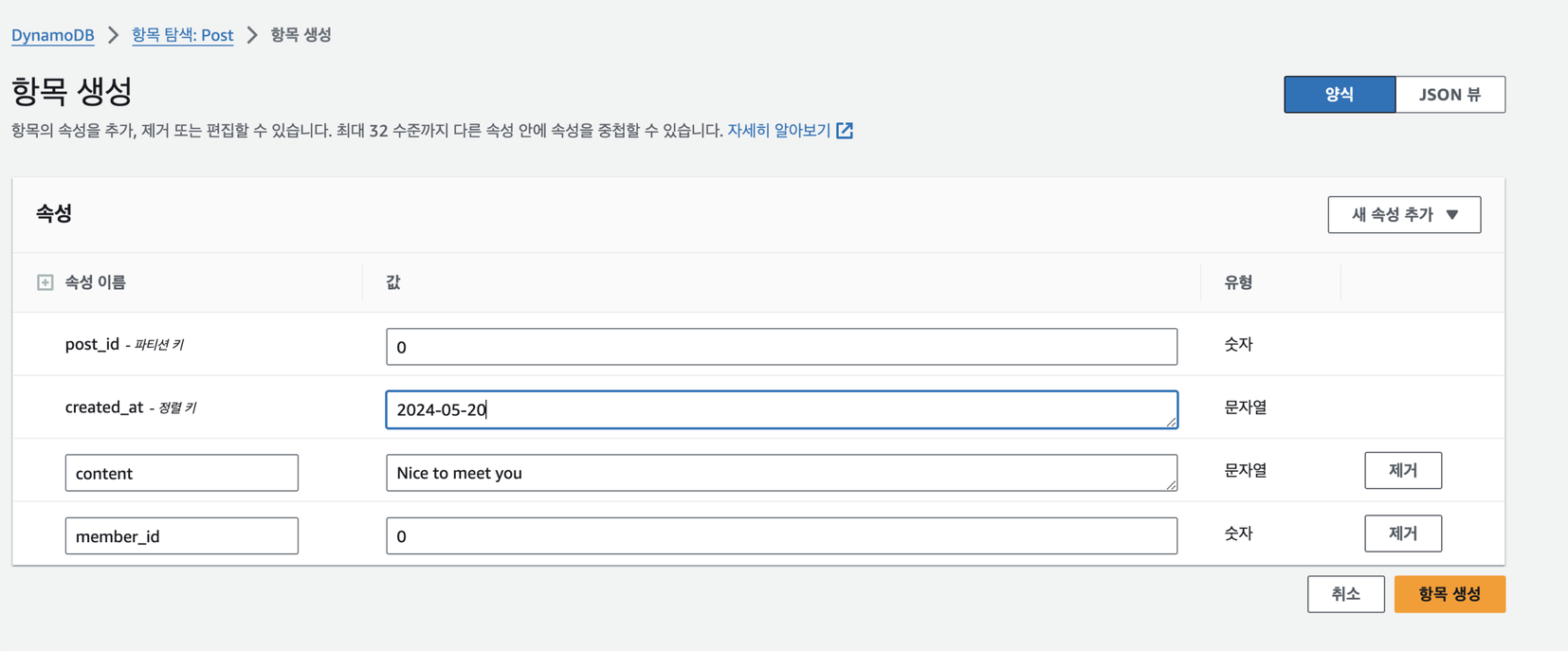



```
{
 "member_id": {
  "N": "0"
 },
 "email": {
  "S": "test@gmail.com"
 },
 "address": {
  "M": {
   "city": {
    "S": "Seoul"
   },
   "street": {
    "S": "Seongbuk"
   },
   "zipcode": {
    "S": "10473"
   }
  }
 },
 "name": {
  "S": "AWS Cloud Club"
 }
}
```

DynamoDB

NoSQL 완전관리형 데이터베이스 - 데이터베이스 삽입

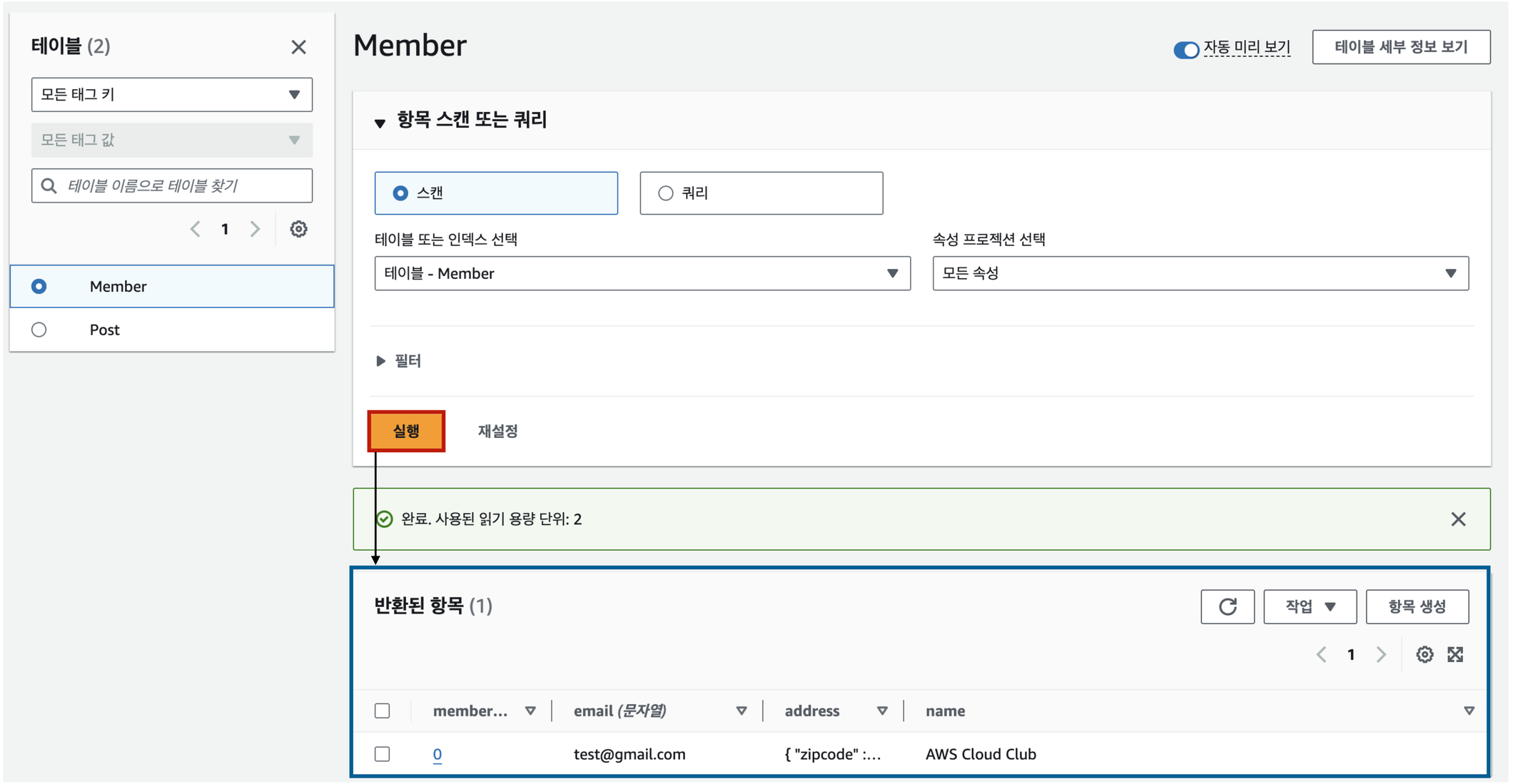
(2) Post 테이블



```
{
  "post_id": {
    "N": "0"
  },
  "created_at": {
    "S": "2024-05-20"
  },
  "content": {
    "S": "Nice to meet you"
  },
  "member_id": {
    "N": "0"
  }
}
```

DynamoDB

NoSQL 완전관리형 데이터베이스 - 데이터베이스 조회



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

```
import mongoose from "mongoose";
import jwt from "jsonwebtoken";
import dotenv from "dotenv";

const envFile = process.env.NODE_ENV === "prod" ? ".env.prod" : ".env.dev";
dotenv.config({ path: envFile }); // .env 파일 사용 (환경 변수 관리)
const secret = process.env.JWT_SECRET;

const { Schema } = mongoose;
const {
  Types: { ObjectId },
} = Schema;

const userSchema = new Schema(
  {
    email: {
      type: String,
      required: true,
      unique: true,
      index: true,
    },
    name: {
      type: String,
      required: true,
    },
    nickname: {
      type: String,
      required: true,
      unique: true,
    },
```

```
const chatRoomSchema = new Schema(
  {
    // 채팅 참여자
    joiner: [
      {
        type: ObjectId,
        ref: "User",
      },
    ],
    // 채팅방 종류
    type: {
      type: String,
      /*
        free = 자유 라이브
        ing = 나 지금 헌혈 중 라이브
        story = 헌혈 이야기 공유 라이브
        crew = 크루
        blood = 지정헌혈
      */
      enum: ["free", "ing", "story", "crew", "blood"], //
      required: true,
    },
    // 채팅방 이름
    title: {
      type: String,
      required: true,
    },
    // 생성일
    created_at: {
      type: Date,
      default: Date.now,
      required: true,
    },
```

```
const chatSchema = new Schema(
  {
    // 채팅방
    chatRoom: {
      type: ObjectId,
      required: true,
      ref: "ChatRoom",
    },
    // 채팅 작성자
    writer: {
      id: {
        type: ObjectId,
        required: true,
        ref: "User",
      },
      nickname: {
        type: String,
        required: true,
      },
    },
    // 메시지
    message: {
      type: String,
      required: false,
    },
    // 이미지 (S3 업로드 URL)
    image: {
      type: String,
      required: false,
    },
    // 생성일
    created_at: {
      type: Date,
      default: Date.now,
      required: false,
    },
  },
  { versionKey: false } // 데이터 삽입 시 __v 칼럼 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 = customErrResponse(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", saevdChatRoom) -> 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 도큐먼트 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 도큐먼트 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)