aws SUMMIT ONLINE

DynamoDB를 게임에서 사용하기

김성수
솔루션즈 아키텍트
AWS Korea

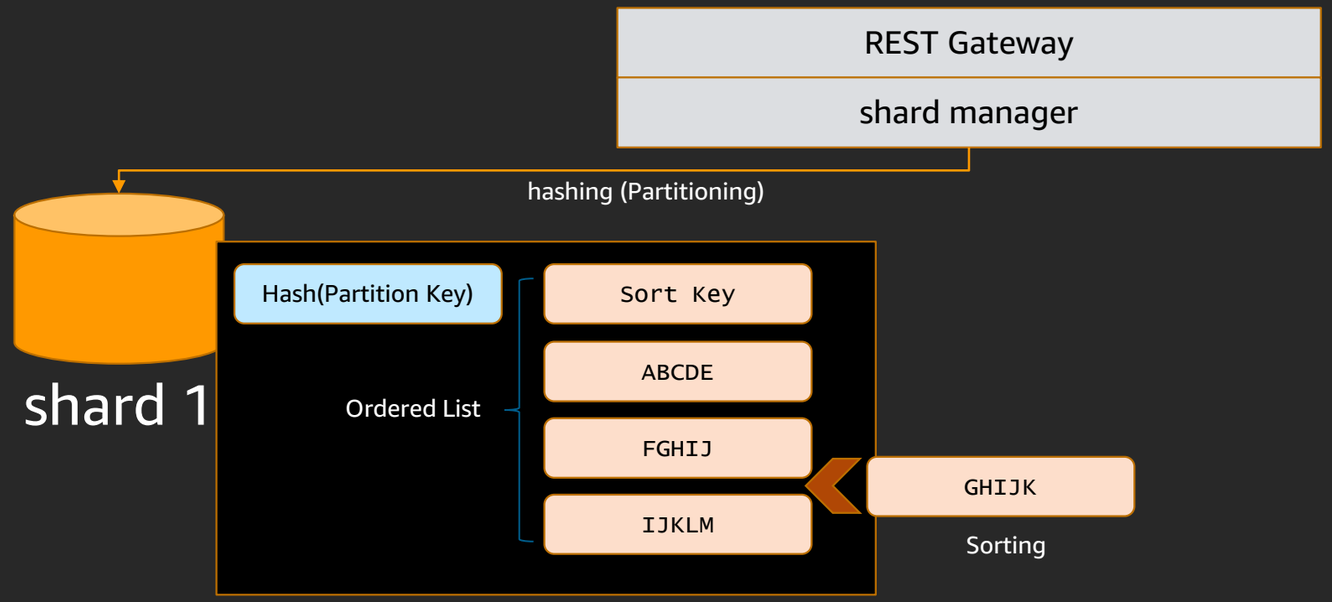
박경표
솔루션즈 아키텍트
AWS Korea

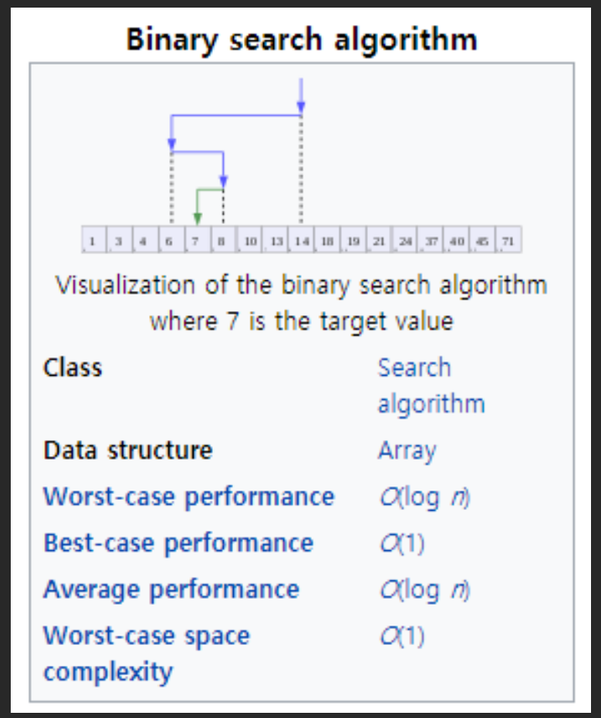
aws SUMMIT ONLINE

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

DynamoDB 저장의 개념

어떻게 대용량 데이터 세트에서 일정한 시간안에 쿼리를 수행할 수 있을까?
바로, Partitioning을 통한 용량 제어와, 저장 시점에 Sorting을 수행하여
쿼리의 수행 속도를 보장합니다.

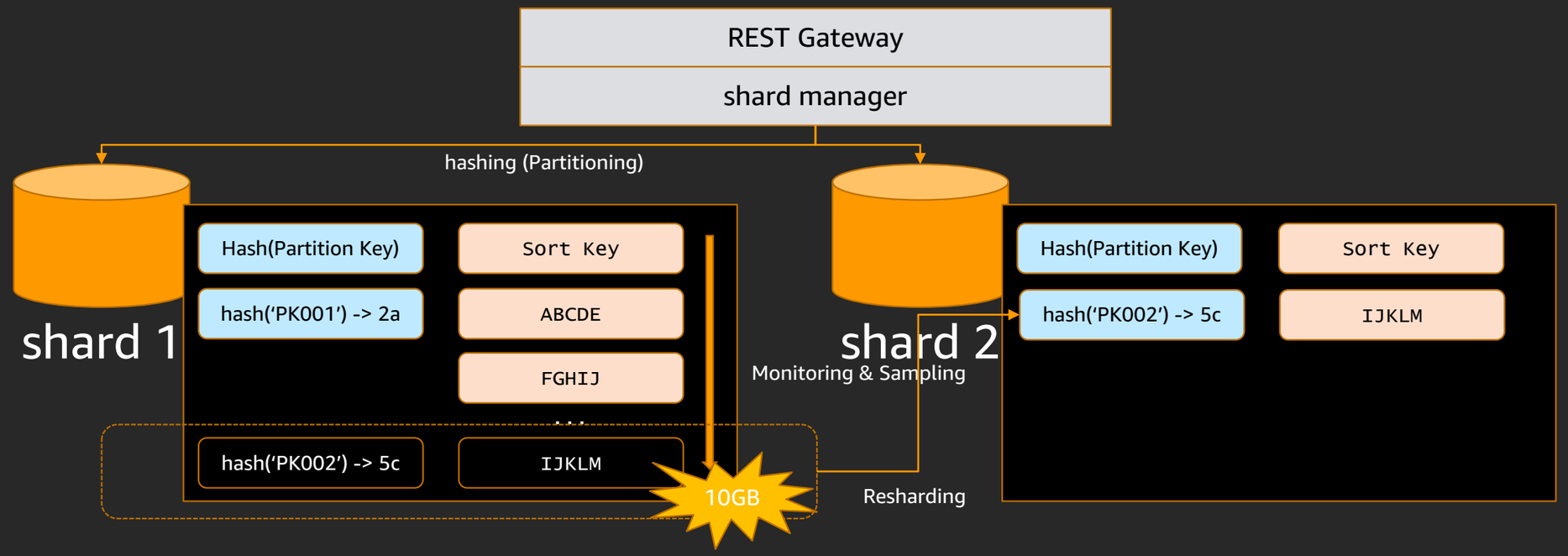




https://en.wikipedia.org/wiki/Binary_search_algorithm

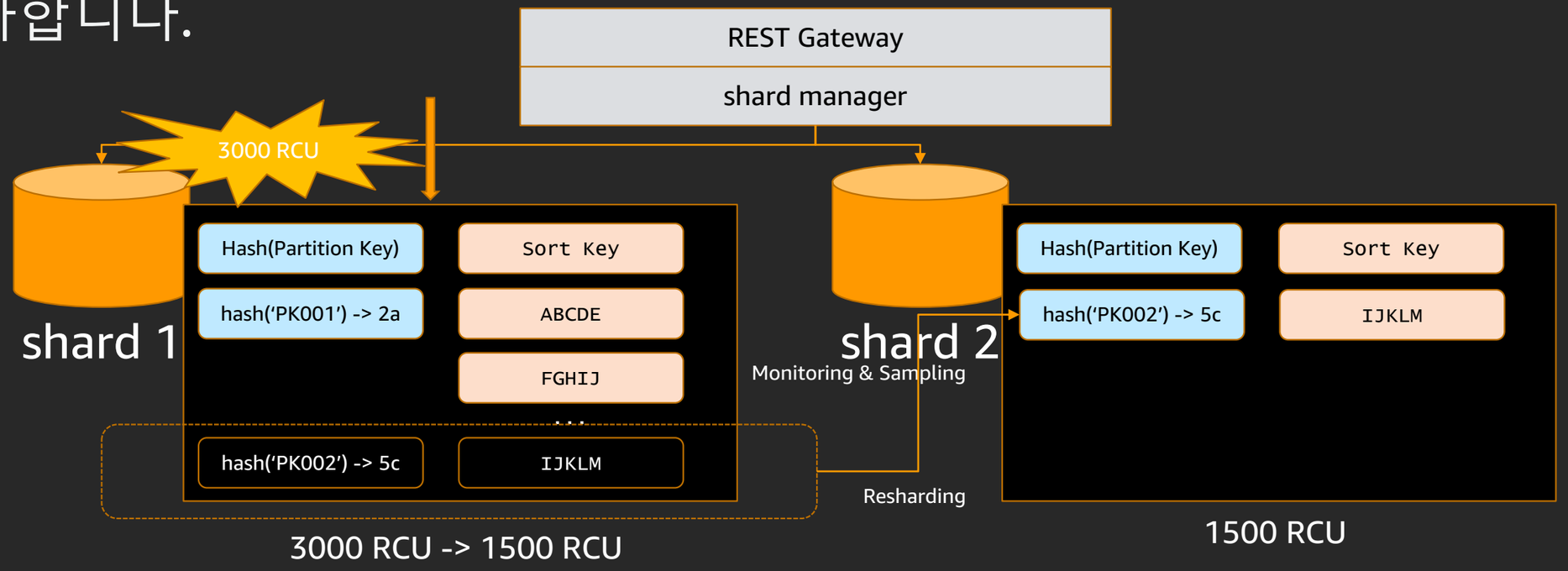
Repartitioning

DynamoDB는 내부적으로 Shard(partition) manager가 개별 파티션의 상태를 모니터링하고 있습니다.
만약, 특정 파티션이 개별 파티션의 한개치인 10GB에 도달하면 Repartitioning작업을 수행합니다.



Repartitioning (계속)

Repartitioning은 두 가지 요건에 의해 발생합니다. 첫번째는 위에서 설명한 크기가 10GB인 경우 그리고, 개별 파티션의 IOPS가 Max값에 도달할 경우 예를 들어, 파티션 하나가 3000 IOPS(WCU기준)에 도달했을 때 Shard manage는, 해당 새로운 파티션을 할당하고, 기존 레코드를 뒤에서 복사합니다.



DynamoDB Autoscaling (Provisioned)

Provisioned IOPS를 사용할 경우에는 아래 권고사항을 유념하여 설정하시기 바랍니다.

1. 반드시, WCU/RCU에 대한 개별적인 정책을 세우셔야 합니다.

2. GSI를 사용하신다면, 반드시 Master Table과 동일한 Autoscaling정책을 구성하십시요.

3. 만약, Global Table기능을 활용한다면, “모든 리전”에 동일한 Autoscaling 정책을 지정합니다.

4. 대형 이벤트를 준비하신다면, Scale-in 이벤트를 잠시 비활성화 할 수 있습니다.

5. Scale In-out이 발생하는 기준을 확인하십시요.
   [Scale out] trigger
   : Two continuous 1 minute windows over target utilization
   [Scale in] trigger
   : 15 continuous 1 minute windows under 80% target utilization

어떤 Price 모델을 사용할 것인가?

만약 꾸준하게 부하가 들어온다고 가정해 봅니다. (아래 표는 ICN On-Demand/Provisioned Price)

리전: 아시아 태평양(서울)

요금 유형	요금
쓰기 요청 유닛	쓰기 요청 유닛 100만 개당 1.3556 USD
읽기 요청 유닛	읽기 요청 유닛 100만 개당 0.271 USD

리전: 아시아 태평양(서울)

프로비저닝된 처리량 유형	시간당 요금
WCU(쓰기 용량 유닛)	WCU당 0.0007049 USD
RCU(읽기 용량 유닛)	RCU당 0.00014098 USD

한쪽은 Request단위로 또 다른 쪽은 WCU 단위로 과금을 하기 때문에 계산이 어렵습니다. WCU를 Request 로 변환해 보겠습니다.

1 RCU는 1초에 한번의 Request를 처리한다는 의미입니다. 즉 아래와 같이,

1 RCU * 3600 (Sec) * 24 (Hour) * 30 (Day)
= 2,592,000 (Request)

따라서, 한달 동안 1RCU로 꾸준하게 부하가 발생한다면, 약 2.6 백만건이며, 이를 On-Demand로 환산하면, 약 0.70$가 됩니다.

Provisioned 모델로 구성하셨다면, 시간당 요금이 0.00014098이므로,

1 RCU * 24 * 30 = 72 RCU / month. 72 RCU = 0.01$

즉, 0.01$가 됩니다.

무조건 Provisioned를 사용하라는 의미가 아니라, Workload특성을 봐야 합니다!

여러 논리적 테이블을 왜 하나의 DDB테이블에?

효율적인 JOIN을 하기 위해서는 하나의 DDB테이블로 구성하여야 한다.

효율적인 Query를 하기 위해서는 하나의 DDB테이블로 구성하여야 한다.

효율적인 Provisioning을 하기 위해서 하나의 DDB테이블로 구성하여야 한다.

여러 레코드를 Nested JSON으로 단일 Record표현이 가능하다.

유사한 형태의 조회를 단일 인덱스로 처리할 수 있다.

Join의 개념을 다르게 볼 수 있을까?

RDBMS에서 JOIN은 서로 다른 두개의 테이블과 레코드를 하나로 묶습니다.

ID	NAME	TEL	ADDR
USER001	HNK	821010xx	Seoul...

ACC_ID	BAL	USER_ID
03-240-X	1,000,000	USER001

```
SELECT A.ID, A.NAME, A.TEL, A.ADDR, B.ACC_ID, B.BAL FROM USER A JOIN ACCOUNT B ON (A.ID = B.USER_ID) WHERE A.ID = "USER001"
-----------------------------------------------------------------------------------------------------------------------------
      USER001, HNK, 821010XX, Seoul..., 03-240-X, 1000000

[Coding] String id = resultset.getString(1); String name = resultset.getString(2); String telno = resultset.getString(3); ...; String accounted = resultset.getString(5); ...
```

DynamoDB에서 JOIN 개념은 두개의 레코드를 한번에 조회하는 것입니다.

ID(Part)	SK(Sort)	NAME	TEL	ADDR	ACC_ID	BAL
USER001	USER	HNK	821010xx	Seoul...		
USER001	ACC				03-240-X	1,000,000

```
Query
{  "TableName" : "USERMASTER", "ProjectionExpression" : "id, sortkey, name, tel, addr, acc_id, bal", "KeyConditionExpression" : "id = :v1", ... }

[Coding] ItemCollection<UserMaster> items = table.query(spec); Iterator<Item> iterator = items.iterator(); Item item = null;
        while(iterator.hasNext()) { item = iterator.next(); if( item.get("sk").equals( "USER" ) classmap(targetBean, User.class, item ) else classmap(targetBean, Acc.class,item) }
```

Parent-Child Query

RDBMS에서 Parent-Child (Master-Detail)형태의 View를 구성할 경우, 보통 2번의 Query를 수행

Parent View

ID	NAME	TEL	ADDR
USER001	HNK	821010xx	Seoul...

Query 1 : Master Table Query

Child View

ACC_ID	BAL
03-240-X	1,000,000
01-210-X	500,000

Query 2 : Child Table Query

DynamoDB에서는, 하나의 Query에서 여러 관련된 테이블 Record를 가지고 올 수 있습니다.

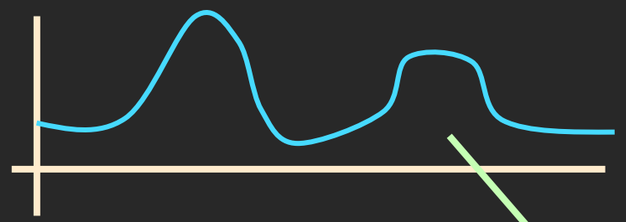
ID(Part)	SK(Sort)	NAME	TEL	ADDR	ACCID	BAL
USER001	USER	HNK	821010xx	Seoul...		
USER001	ACC				03-240-X	1,000,000
USER001	ACC				01-210-X	500,000

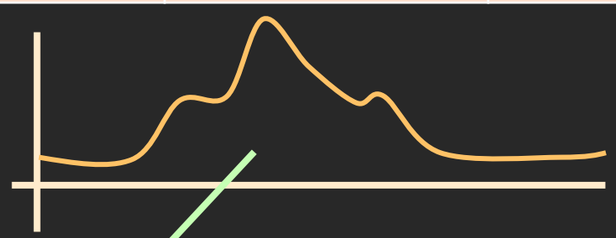
Provisioning

예를 들어, “사용자 테이블“, “어카운트 테이블“을 나누어서 설계를 하였다면 하면,

ID	NAME	TEL	ADDR
USER001	HNK	821010xx	Seoul...

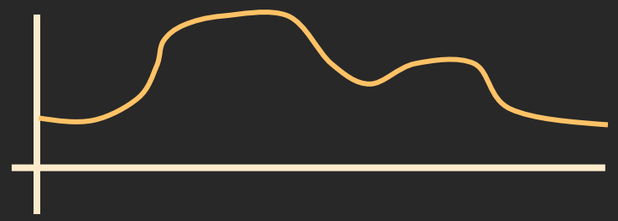
ACC_ID	BAL	USER_ID
03-240-X	1,000,000	USER001





계별적인, 테이블 사용량에 대한 Provisioned IOPS를 확인하고 설계하여야 합니다.
한개의 테이블로 구성할 경우, 서비스 전체에 대한 IOPS만을 설계하면 됩니다.

ID(Part)	SK(Sort)	NAME	TEL	ADDR	ACC_ID	BAL
USER001	USER	HNK	821010xx	Seoul...		
USER001	ACC				03-240-X	1,000,000
USER001	ACC				01-210-X	500,000



평탄화/상보관계

여러 레코드를 하나의 레코드로

예를 들어, “사용자 테이블“, “초기 접속 서버“를 테이블로 나누어서 설계를 하였다면 하면, 아래와 같이 구성할 수 있습니다.

ID	NAME	TEL	ADDR
USER001	HNK	821010xx	Seoul...

SRV_ID	USER_ID	REGION
SRV_001_LOBBY	USER001	SEOUL
SRV_002_WORLD	USER001	SEOUL
SRV_003_WORLD	USER001	TOKYO

만약, “초기 접속 서버” 정보가 **고정적이고, 사용자 정보 특히 로그인 시점의 정보와 밀접하게 연결**되어 있다면, 여러 레코드를 하나의 레코드로 묶어서 제공해 줄 수 있게 됩니다.

ID	NAME	TEL	ADDR	INIT_SERVER

USER001	HNK	821010xx	Seoul...	{ servers : [{
				server_id : “SRV_001_LOBBY”,
				region : “SEOUL”
				}, {
				server_id : “SRV_002_WORLD”,
				region : “SEOUL”
				}, {
				server_id : “SRV_003_WORLD”,
				region : “SEOUL”
				}]
				}

Attribute Overriding

RDBMS에서는 테이블과 컬럼은 고정적이며, 특히 컬럼의 경우, 비즈니스 목적에 맞는 Domain으로 식별하여 (예. 전화번호 VARCHAR(11), 주민번호 VARCHAR(13) 등) 다양한 테이블에, 준용하여 사용하도록 권장됩니다.

TABLE	TB_INTERNAL_WORKFORCE		
Column name	**Biz Domain**	**Physical Type**	**Comment**
EMPLOYEE_ID	ID_TYPE	VARCHAR(12)	
MOB_TEL_NO	MOB_TYPE	VARCHAR(11)	Excluded Dash
RESID_ID	RESIDID_TYPE	VARCHAR(13)	
ADDR	ADDR_TYPE	VARCHAR(255)	
WORK_TEL_NO	TELNO_TYPE	VARCHAR(11)	Excluded Dash
HOME_TEL_NO	TELNO_TYPE	VARCHAR(11)	Excluded Dash

TABLE	TB_INTERNAL_SERVICE		
Column name	**Biz Domain**	**Physical Type**	**Comment**
SERVICE_ID	SVCID_TYPE	VARCHAR(12)	
MANAGER_ID	ID_TYPE	VARCHAR(12)	
REPR_TEL_NO	TELNO_TYPE	VARCHAR(11)	Dash Excluded
ENTRY_POINT	URL_TYPE	VARCHAR(255)	
CREATE_DT	DT_TYPE	TIMESTAMP TZ	
MODIFIED_DT	DT_TYPE	TIMESTAMP TZ	

Attribute Overriding

DynamoDB 에서는 쿼리 특성에 맞추어, 일부 컬럼을 Biz Domain을 Overloading하여, 원래는 분리되어 있는 컬럼을 하나의 컬럼에 담을 수 있으며, 이를 이용하여 Index및 컬럼 갯수를 최소화할 수 있습니다.

TABLE	TB_INTERNAL_WORKFORCE		
Column name	**Biz Domain**	**Physical Type**	**Comment**
EMPLOYEE_ID	ID_TYPE	VARCHAR(12)	
MOB_TEL_NO	MOB_TYPE	VARCHAR(11)	Excluded Dash
RESID_ID	RESIDID_TYPE	VARCHAR(13)	
ADDR	ADDR_TYPE	VARCHAR(255)	
WORK_TEL_NO	TELNO_TYPE	VARCHAR(11)	Excluded Dash
HOME_TEL_NO	TELNO_TYPE	VARCHAR(11)	Excluded Dash

TABLE	TB_INTERNAL_SERVICE		
Column name	**Biz Domain**	**Physical Type**	**Comment**
SERVICE_ID	SVCID_TYPE	VARCHAR(12)	
MANAGER_ID	ID_TYPE	VARCHAR(12)	
REPR_TEL_NO	TELNO_TYPE	VARCHAR(11)	Dash Excluded
ENTRY_POINT	URL_TYPE	VARCHAR(255)	
CREATE_DT	DT_TYPE	TIMESTAMP TZ	
MODIFIED_DT	DT_TYPE	TIMESTAMP TZ	

Merged Table

TABLE	TB_SERVICE_WORKFORCE				
Column name	**Biz Domain**	**Physical Type**	**Comment**	**PK/SK**	**INDEX_NAME**
ID	ID_TYPE, SVCID_TYPE	String	Employee_id , Service_id overloaded	PK	
MOB_NO	MOB_TYPE, TELNO_TYPE	String	Mob_tel_no, repr_tel_no		
RES_MGR_ID	RESIDID_TYPE, ID_TYPE	String	Resid_id, manager_id	SK	
ADDR	ADDR_TYPE	String			
WORK_TEL_NO	TELNO_TYPE	String	Work_tel_no		
HOME_TEL_NO	TELNO_TYPE	String	Home_tel_no		
ENTRY_POINT	URL_TYPE	String	Entry_point		

예시를 통한 디자인 설계 리뷰

[DDB 설계 리뷰 전 준비]

1. 대상 테이블에 대한 용도가 명확해야 합니다.

2. 논리적인 요구사항(쿼리)을 명확하게 숙지하여야 합니다.

3. 기본적인 부하 패턴을 인식하여야 합니다.

4. 테이블의 초기 데이터량과 IOPS를 알고 있어야 합니다.
   (On-Demand 모델로 대치 가능)

안좋은 예시

현재 Application의 DDB접근 방법

안좋은 예시

로그인 부터 사용자 관련 데이터가 Instance (Structure)형태로 메모리에서 관리합니다.
여러건의 레코드의 경우 Collection 형태의 Attribute로 관리합니다.

```
Class UserInformation {
    private String name;
    private String password;
    private bool isValid;
    private accountId;
    private List<Logs> histLogs;
    private List<GameChar> characters;
    private int totalGold;
}
```

여러건의 레코드를 Collection Attribute로 처리

[위 레코드에 대한 처리 방식]

```
    private void storeToDDB(UserInformation userInfo) {
        List<dynamodb.Item> splittedItems = DynamoDBMapper.convertToItem(userInfo)
        splittedItems.foreach(x => table.putItem(x));
    }
    private UserInformation restoreFromDDB(String userid) {
        List<dynamodb.Item> splittedItems = new ArrayList<dynamodb.Item>();
        dynamodb.Item currentItem = table.getItem("UserId", userid); splittedItems.add(currentItem);
        while((String nextPos = currentItem.getString("NextPos")) != null) {
            currentItem = table.getItem("UserId", nextPos); splittedItems.add(currentItem);
        }
        return DynamoDBMapper.convertToUserItem(splittedItems);
    }
```

하나의 Instance를 DynamoDB Item으로 Split하여 한꺼번에 저장

여러개의 DynamoDB Item을 추적하여, 뭉친다음 하나의 Instance로 제공

내가 만든 DynamoDB Table의 형태

안좋은 예시

게임의 로그인 부터 사용자의 모든 데이터를 하나의 Row에 대응
개인별 로그성 데이터의 누적 및 관리

ID(Part)	VALID	NAME	PWD	LOGHIST	ACCID	기타등등
USER001	YES	HANK	**202cb...**	[“2019-12-31 14:..”, “2020-01-03 16:1...”,...]		
USER002	YES	IGUANA	**962ac...**	[“2020-01-14 15:..”, “2020-01-23 17:1...”,...]	03-240-X	
USER004	NO	LILY	**59075...**	[“2020-01-30 17:..”, “2020-02-03 16:1...”,...]	01-210-X	

한번에 모든 데이터를 읽어서 Application이 관리
쓰기는 중요도에 따라 Batch형태, 또는 실시간으로 수행
- Application의 Reliability가 중요

On the fly data loss에 대한 Reconstruction 을 위한 Application 로직 중요

내가 만든 DynamoDB 구조

안좋은 예시

[설계 원칙]

1 사용자 정보는 1 ROW 이상으로 구성됩니다.

400KB/ row – data split read/ write: 400KB가 넘어가면 끝에 continue할 Key값을 추가하여 다음 ROW를 알 수 있고, 이를 활용하여 이어 읽기가 가능해 집니다.

읽을 때는 모두 다 읽고, 쓸 때도 모든 데이터를 한번에 Flush 하는 형태가 됩니다.

ID(Part)	VALID	NAME	PWD	LOGHIST	기타데이터	NEXTROW
USER001	YES	HANK	**202cb…**	[“2019-12-31 14:..”, “2020-01-03 16:1…”, ..]	[“aaa”, …]	USER001-p13
ID(Part)	**VALID**	**DEX**	**GOLD**	**LASTPOS**		**NEXTROW**
USER001-p13	YES	16	**47782**	[12345,77863,43]		END

문제점

읽기 낭비: Workload에 맞는 쿼리 튜닝을 수행할 수 없습니다.

예) User ID/ PWD만 확인

오브젝트가 처음에는 작지만 게임이 진행되면서 커지는 문제

최초 5KB만 읽고 쓰다가 이젠 2MB씩 읽고 쓰고 있음

LOG성 자료 확인/제거가 Batch로 동작. 대량의 Read/ Write가 발생합니다.

Main Block이 아닌 Row에 있는 데이터를 한번에 가져올 방법이 없습니다.

사용자 데이터 크기를 어플리케이션에서 직접적으로 관리해주어야 합니다.

수정된 예시

개선 방향

데이터(컬럼)의 특성을 알아야 합니다.

데이터를 특성에 따라 분류해야 합니다.

- 자주 사용하는 데이터와 자주 사용하지 않는 데이터
- 그룹으로 처리되는 데이터
- 속성에 따라 분류 – Status / Count
- 로그성 데이터

컬럼을 레코드를 분리합니다.

분리된 레코드의 키 설계를 수행합니다.

- SORT Key의 디자인
- 데이터의 중복과 Composite Key

데이터를 알아야합니다.

데이터를 온전히 파악할 필요가 있습니다
데이터를 어떤 빈도로, 어떤 의도로 접근할지 알아야합니다

[예시]
읽기가 잦은 데이터 – ID/ PWD
쓰기/갱신이 잦은 데이터 – Character 속성, ITEM
유효기간이 있는 데이터 – LOG성 데이터들
기타 등등

데이터 컬럼 그룹핑

ID(Part)	VALID	NAME	PWD	LOGHIST	ITEMS	NEXTROW
USER001	YES	HANK	**202cb…**	[“2019-12-31 14:..”, “2020-01-03 16:1…”,…]	[“aaa”, …]	USER001-p13
ID(Part)	**VALID**	**DEX**	**GOLD**	**LASTPOS**		**NEXTROW**
USER001-p13	YES	16	**47782**	[12345,77863,43]		END

사용자 초기 로그인 시에 활용되는 컬럼

ID	PWD	NAME
USER001	YES	HANK

자주 내용이 변경되면서, 조회가 빈번하게 발생하는 경우.

ID	DEX	GOLD	ITEMS
USER001	YES	**47782**	[{itemcd : “guntlet”, amount : 2...}]

히스토리 성의 자료로, 추가가 되면서, 특정 기간이 지나면, 제거해야 하는 자료들

ID	LOGHIST
USER001	[“2019-12-31 14:..”, “2020-01-03 16:1…”,…]

레코드 분리 검증

단일 레코드를 여러개의 컬럼 그룹으로 구분하여, 레코드를 생성할 수도 있으며, LSI, GSI를 이용하여 구현할 수도 있습니다. 제일 중요한 판단 기준은 해당 자료의 생성/조회 시점에 같이 이용이 필요한지에 따라 다릅니다.

사용자 초기 로그인 시에 활용되는 컬럼

ID	PWD	NAME
USER001	YES	HANK

사용자 등록/로그인시에만 주로 활용됩니다. 즉 독립적인 레코드로 구성해도 좋습니다.

자주 내용이 변경되면서, 조회가 빈번하게 발생하는 경우.

ID	DEX	GOLD	ITEMS
USER001	YES	HANK	[{itemcd : “guntlet”, amount : 2...}]

게임 진행이 종료될 경우, 또는 주기적으로 저장할 경우, 활용됩니다. 다른 컬럼과의 종속관계가 낮아 독립적인 레코드 구성 O

히스토리 성의 자료로, 추가가 되면서, 특정 기간후 삭제

ID	LOGHIST
USER001	[“2019-12-31 14:..”, “2020-01-03 16:1...”,...]

시간대에 따른 추가 Record가 생성이 되며, TTL적용을 위하여 독립 레코드 구성에 적합함

분리된 레코드의 Sort Key 설계

일단, 분리된 레코드에 Sort Key를 제공하여, Primary Key를 구성하여, 개별 Record로 인식할 수 있게 하며, 조회 조건을 고려하여 Composite SortKey로 추가 조회 조건으로 활용할 수 있게 제공

사용자 초기 로그인 시에 활용되는 컬럼

ID	SK(SortKey)	PWD	NAME
USER001	“LOGIN”	YES	HANK

자주 내용이 변경되면서, 조회가 빈번하게 발생하는 경우.

ID	SK(SortKey)	DEX	GOLD	ITEMS
USER001	‘ServerId’+’:” + ‘CharId’	YES	HANK	[{itemcd : “guntlet”, amount : 2...}]

히스토리 성의 자료로, 추가가 되면서, 특정 기간후 삭제

ID	SK(SortKey)	LOGHIST
USER001	“LOG”+‘:’+‘ServerId’+’:”+‘Serial’	[“2019-12-31 14:..”, “2020-01-03 16:1...”,...]

데이터 읽기의 형태

사용자의 모든 데이터를 다 읽어오고 싶을때

aws dynamodb get-item --table-name MainTable --key ‘{“UserID": {"S": “myuserid"}}’

사용자의 특정 서버에 종속된 캐릭터 정보만 읽어오고 싶을때

aws dynamodb query –table-name MainTable –key-condition-expression “UserId = :id and SortKey = :sk " –expression-attribute-values "{ ₩":id₩" : { ₩"S₩" : ₩"myuserid₩" }, ₩":sk₩" : { ₩"S₩" : ₩“SRV01:CHR01₩" } }

사용자의 특정 데이터 그룹(LOG)을 다 읽어오고 싶을때

aws dynamodb query --table-name MainTable --key-condition-expression “UserId = :id and begins_with(SortKey, :sk)" --expression-attribute-values "{ ₩":id₩" : { ₩"S₩" : ₩"myuserid₩" }, ₩":sk₩" : { ₩"S₩" : ₩“LOG₩" } }"

Summary

세상에 동일한 워크로드는 존재하지 않는다.

DynamoDB에서는 레코드, 컬럼을 이동할 수 있는 자유를 느낄 수 있다

모든 문제는 바로 사용자의 패턴에 달려있다.

세부적인 DynamoDB 설계에 대한 조언을 얻고자 한다면...

DyanmoDB Workshop for workload

DynamoDB 설계 경험을 가지고 있는 Special SA가 투입되어, 실제 사용자 Workload를 같이 분석하고,

DynamoDB 레이아웃을 같이 분석하거나,

기존에 있던 DynamoDB 설계를 같이 Review해주는 Deep Dive Session

담당자 : Khim, Sungsoo (sungsook@amazon.com)
Park, KyungPyo (kyunp@amazon.com)

여러분의 소중한 피드백을 기다립니다!

강연 평가 및 설문 조사에 참여해 주세요.

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

감사합니다

aws SUMMIT ONLINE

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.