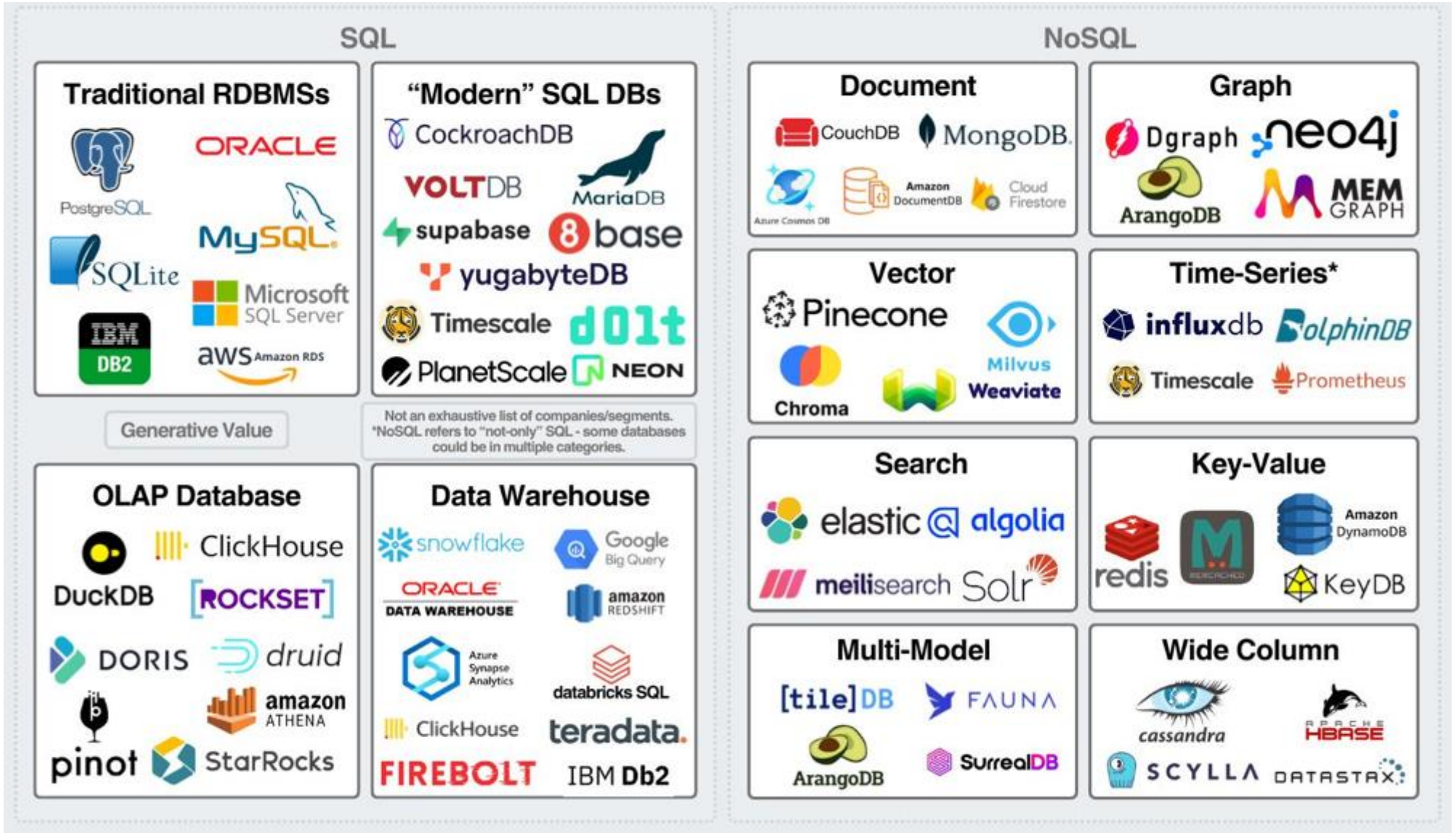
Couchbase

2교시.
NoSQL 모델링, SDK 개요, 실습

1 SQL, NoSQL 모델링
2 SDK 개요
3 SDK 실습 : Python, Java
A SQL Samples

2-1.
SQL, NoSQL 모델링

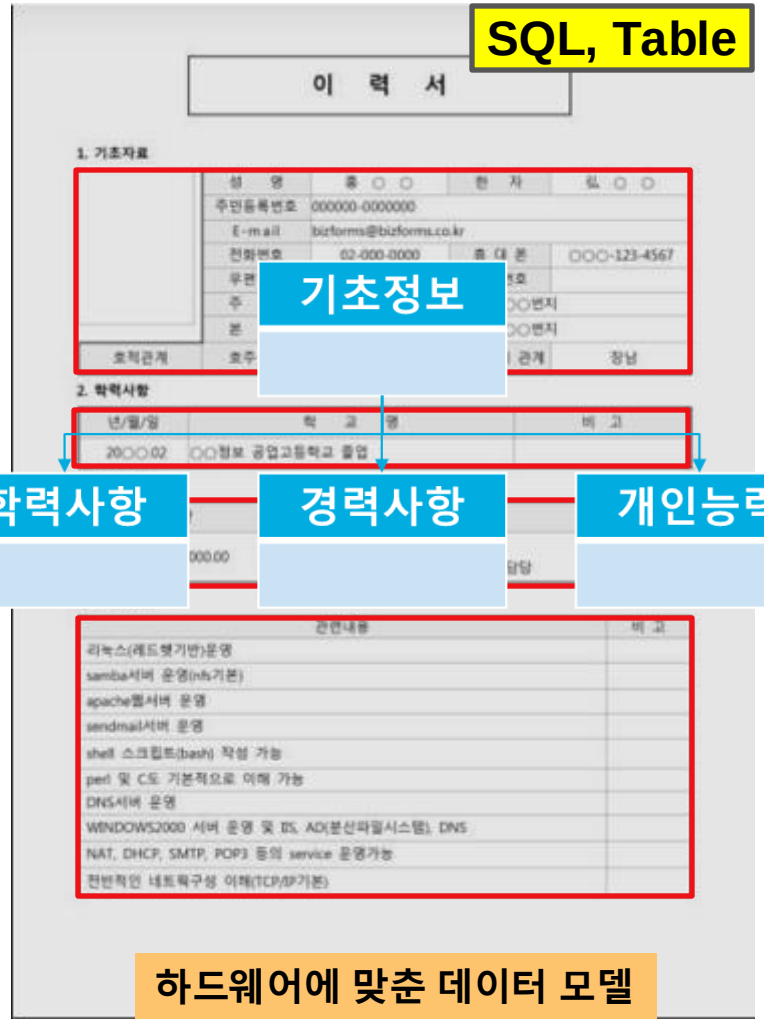
데이터 모델 : SQL(Table)과 NoSQL(Real World)



데이터 모델 : JSON 도큐먼트

JSON은 텍스트로 이루어져 있으므로, 사람과 기계 모두 읽고 쓰기 쉽다. 프로그래밍 언어와 플랫폼에 독립적이므로, 서로 다른 시스템 간에 객체를 교환하기에 좋다.

- JSON 도큐먼트의 장점
  - 단일 도큐먼트 내에 다양한 정보를 계층 구조를 활용하여 저장
  - 정보 추가/삭제가 유연한 구조 제공
  - 데이터 전달을 위한 표준 인터페이스 역할
- RDB와 차별점
  - 여러 테이블로 분리, 저장되는 데이터를 단일 도큐먼트에 저장
  - 테이블 간 조인을 최소화하여 데이터 처리 속도 향상



NoSQL, JSON 도큐먼트

```
KEY : 1001
{
    “성명”: “홍길동”,
    “주소”: “서울시 OO구 OO동 OOO-OOO”,
    “E-mail”: “HongKildong@couchbase.com”,
    …
    “학력사항”: [
        {
            “졸업년도”: “2019년”,
            “학교명”: “OO정보 공업고등학교”
        }
    ],
    “경력사항”: [
        {
            “기간”: “2019 ~ 현재”,
            “관련내용”: “XX글로벌 IT팀 Unix 서버 담당”
        }
    ]
    “개인능력”: [
        {
            “관련내용”: “리눅스 운영”
            “비고”: NULL
        },
        …
    ]
}
```

실제 세계에 맞춘 데이터 모델

논리 / 물리 모델

- RDBMS와 유사한 구조의 논리 계층 구조로 구성하여 편리한 데이터 관리
- Data 서비스를 완전 메모리DB로 사용도 가능하며 용도에 따라 물리 저장 방식을 선택할 수 있음

RDBMS	Couchbase
Server	Cluster
Database	**Bucket**
Schema	**Scope**
Table	**Collection**
Row	**Document (JSON)**
Value	Sub-Document, Array

Feature	Ephemeral Bucket	Couchbase Bucket	Magma Bucket
Bucket memory quota (per node)	Min 256MB	Min 256MB	Min 1024MB
Max Object Size	20MB	20MB	20MB
Persistence	no	yes	yes
Replication and XDCR	yes	yes	yes
Encrypted data access	yes	yes	yes
Rebalance	yes	yes	yes
N1QL, Seach, Analytics, Eventing	yes	yes	yes
Indexing	yes	yes	yes
Backup	yes	yes	yes

관계형 vs. 다큐멘터 데이터 모델

관계형 데이터 모델	다큐멘트 데이터 모델
***Required* 정규화**	***Relaxed* 정규화**
• Schema enforced by the database (스키마가 데이터베이스에 의해 강제짐) • Same fields required in all records (모든 레코드는 같은 필드를 가져야 함)	• Schema inferred from structure (시키마는 구조로 부터 추론됨) • Fields may vary, be duplicate or missing (필드에 어떤 제약도 없음)
• Optimized for data entry(데이터 구성에 최적화) • Reduced duplicated data (중복 감소,제거) • Minimize data inconsistencies (데이터 불일치 최소화)	• Optimized based on access patterns (사용 패턴에 최적화) • Flexible and agile development(유연하고 빠른 개발) • Supports clustered, scalable architecture(확장성 보장)

Real World Data 를 JSON으로 표현

Customer

CustomerID	Name	DoB
CBL2017	Jane Smith	1990-01-30

Customer Document Key: CBL2017

```
{
    "Name" : "Jane Smith",
    "DOB"  : "1990-01-30”
}
```

OR

Customer Document Key: CBL2017

```
{
    "Name" : {
         "fname": "Jane",
         "lname": "Smith"
     }
    "DOB"  : "1990-01-30"
}
```

- **The Primary Key becomes the Document Key**
- **Column name-Column value become KEY-VALUE pair**

고객의 카드 정보를 JSON으로 표현

Customer

CustomerID	Name	DoB
CBL2017	Jane Smith	1990-01-30

Billing

CustomerID	Type	CardNum	Expiry
CBL2017	Visa	5827-2842-2847-3222	2025-03
CBL2017	Master	6274-2542-5847-3949	2024-12

Customer Document Key: CBL2017

```
{
    "Name" : "Jane Smith",
    "DOB"  : "1990-01-30",
    "Billing" : [
        {
            "type"    : "visa",
            "cardnum" : "5827-2842-2847-3222",
            "expiry"  : "2025-03"
        },
        {
            "type"    : "master",
            "cardnum" : "6274-2542-5847-3949",
            "expiry"  : "2024-12"
        }
    ]
}
```

- **Denormalization simplifies data access and offers the best performance**
  비정규화는 데이터 사용을 단순화 해주고, 최고의 성능을 제공함
- **Value evolution: Simply add additional array element or update a value**

테이블(RDBMS)을 컬렉션(NoSQL)에 매핑 | eCommerce 예제

Orders	
id	200
customer	40
status	Paid

Customers	
id	40
name	Jane Smith
email	jane@couchbase.com
address	11-21 Paul Street
city	London
postcode	EC2Q 4JU
telephone	44-20-3837-9130

Orders Details	
id	3000
order	200
item	1
quantity	3

Products	
id	1
name	T-shirt
description	Red Couchbase T-shirt
image	tshirt_red.jpg

Bucket: ecommerce | Scope:_default

Collection: **'Products'**

DocID:1

```
{
 "name": "T-shirt",
 "description": "Red Couchbase T-shirt",
 "image": "tshirt_red.jpg"
}
```

Collection: **'Customers'**

DocID:40

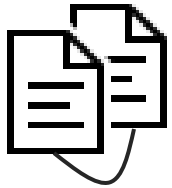
```
{
 "name": "Jane Smith",
 "email": "jane@couchbase.com",
 "address": "11-21 Paul Street",
 "city": "London",
 "postCode": "EC2A 4JU",
 "telephone": "44-20-3837-9130"
}
```

Collection: **'Orders'**

DocID:200

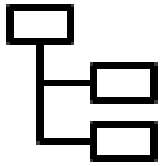
```
{
 "customer": {
  "id": 40,
   "name": "Jane smith",
   "email": "jane@couchbase.com"
 },
 "status":"Paid",
 "orderDetails": [
  { "productId" : 1, "name": "T-shirt","quantity": 3},
 ]
}
```

JSON 데이터 모델시 고려사항



EMBED WHEN

- There is an Ownership Relationship
  오너쉽 관계가 명확할 때
- Both docs are frequently accessed together
  두 문서가 동시에 사용되는 경우가 많을 때
- Reads greatly outnumber writes
  읽기가 쓰기에 비해 많을 때
- Data is small
  데이터 사이즈가 작을 때



REFER WHEN

- There is not an Ownership Relationship
  오너쉽 관계가 명확하지 않을 때
- Both docs are not frequently accessed together
  두 문서가 동시에 사용되는 경우이 작을 때
- Document is updated frequently
  문서가 자주 변경될 때
- Need to reduce the document size
  문서의 크기를 줄일 필요가 있을 때

Try to embed first, refer when it makes sense

먼저 Embed를 시도해 보고, 그 다음 Refer 고려

Relationships에 따른 Embed 와 Refer

- **1-1**

  Embed Example: Satellite and Manufacturer

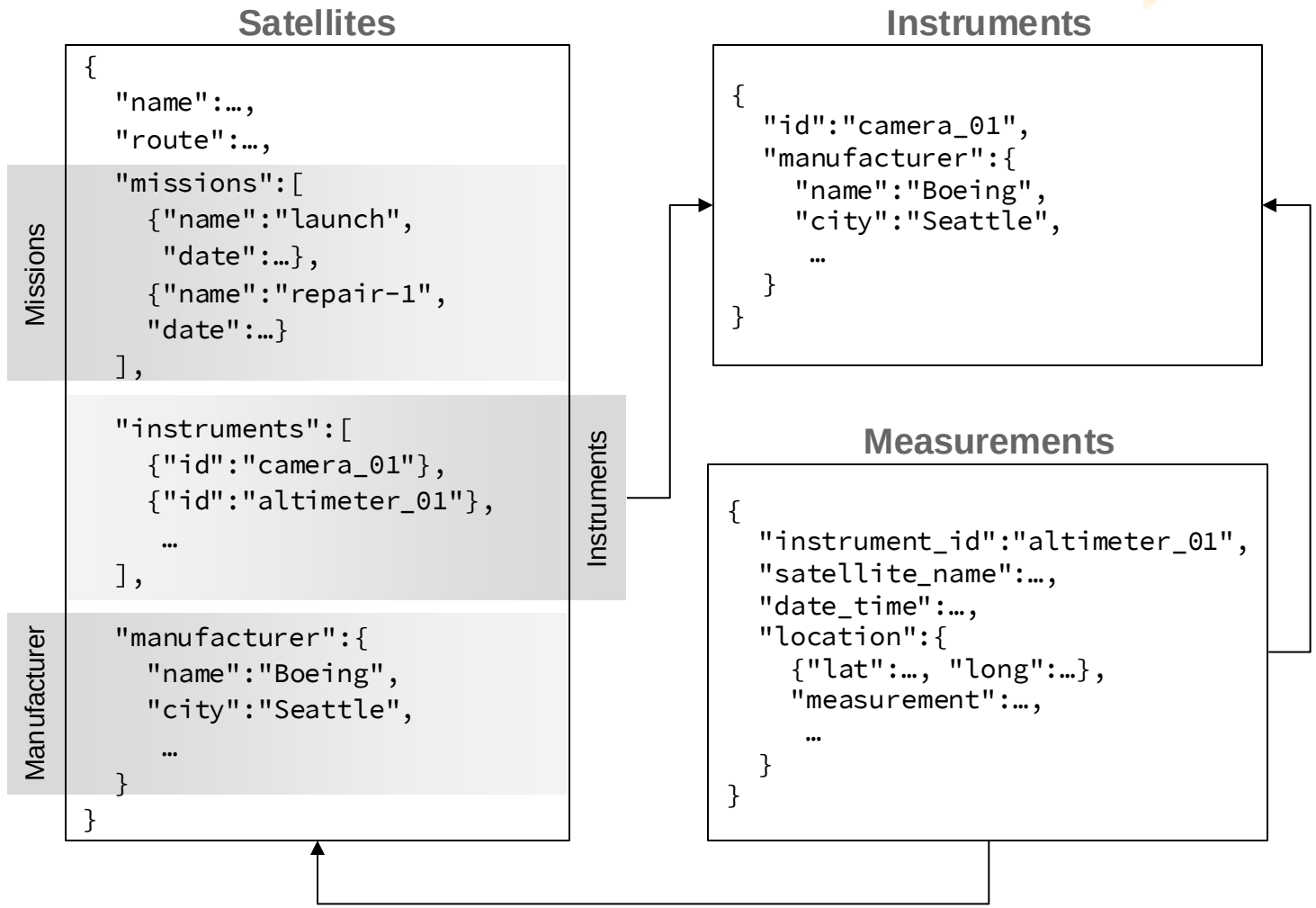
- **1-Many**

  Embed Example: Satellite and Missions

  Reference Example: Measurements, Satellites and Instruments

- **Many-Many**

  Reference Example: Satellite and Instruments



2-2.
SDK 개요

SDK 개요

10 개 개발 언어에 대한 Couchbase 클러스터 접근을 위한 SDK(Software Development Kit) 제공

Enterprise Support

v4.2

V4.3

v3.5

v2.8

v3.6

v4.2

v3.3

v1.6

v3.5

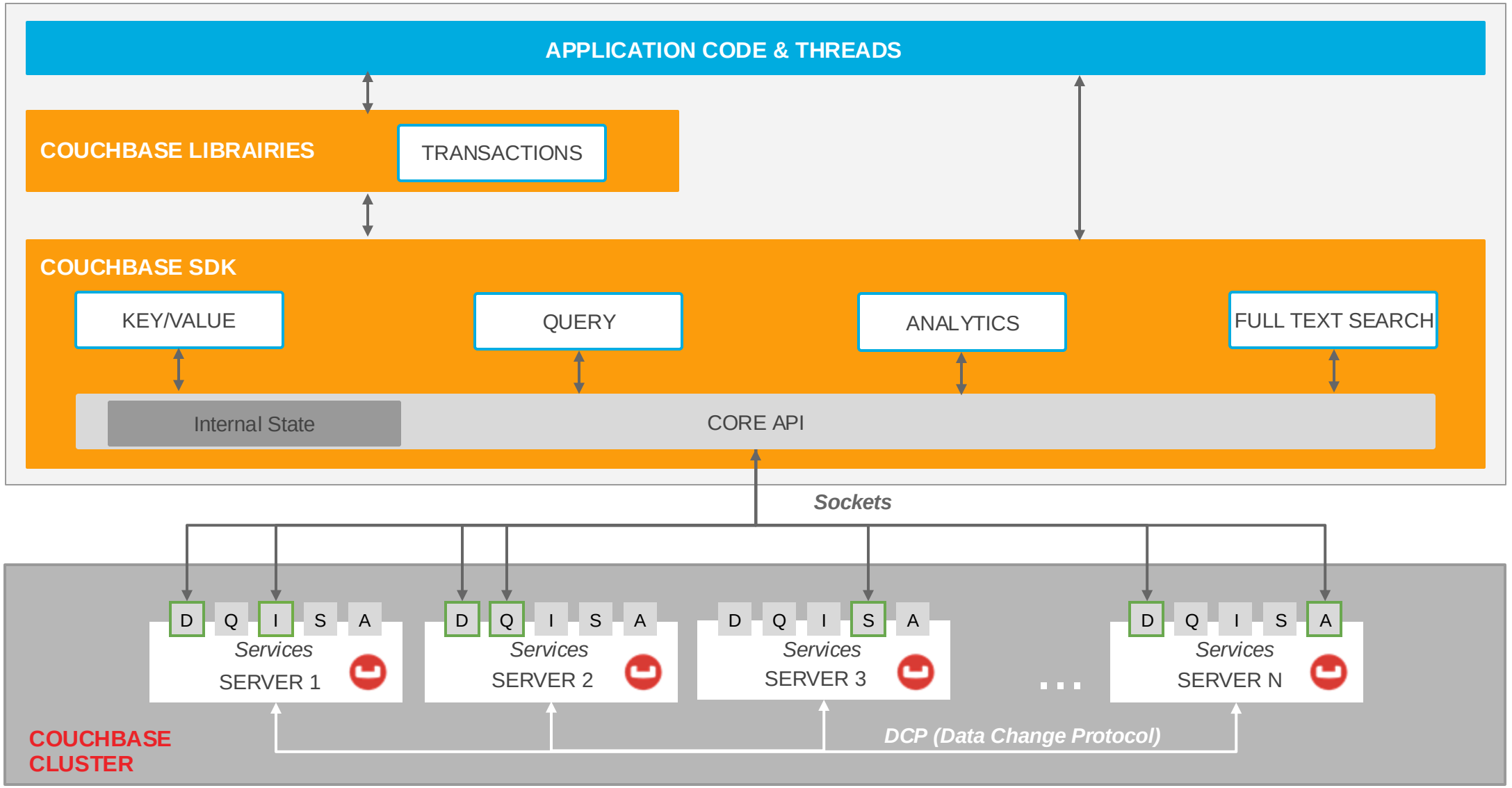
Mobile Support

Java for Android

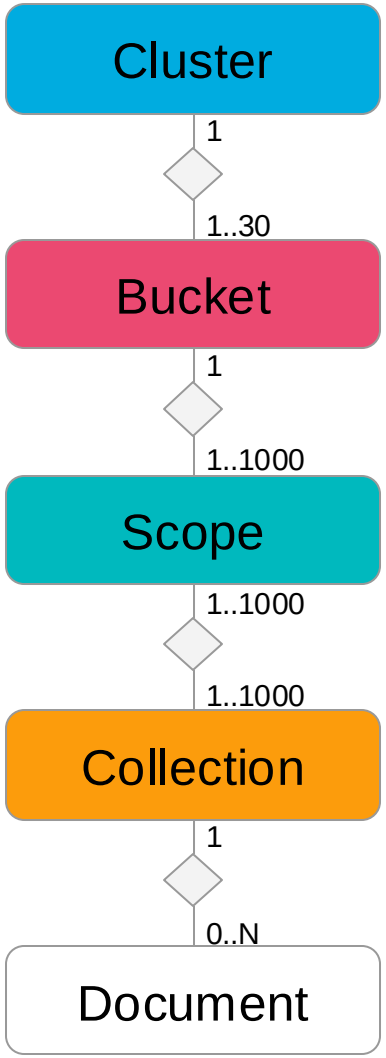
Community Support

v1.0-pre

SDK 아키텍처 : 통합 API 제공



SDK 아키텍처 : 엔티티 계층구조



- Provides access to **cluster-level** operations(인스턴스)
- Manages bootstrapping, fault tolerance, rebalancing

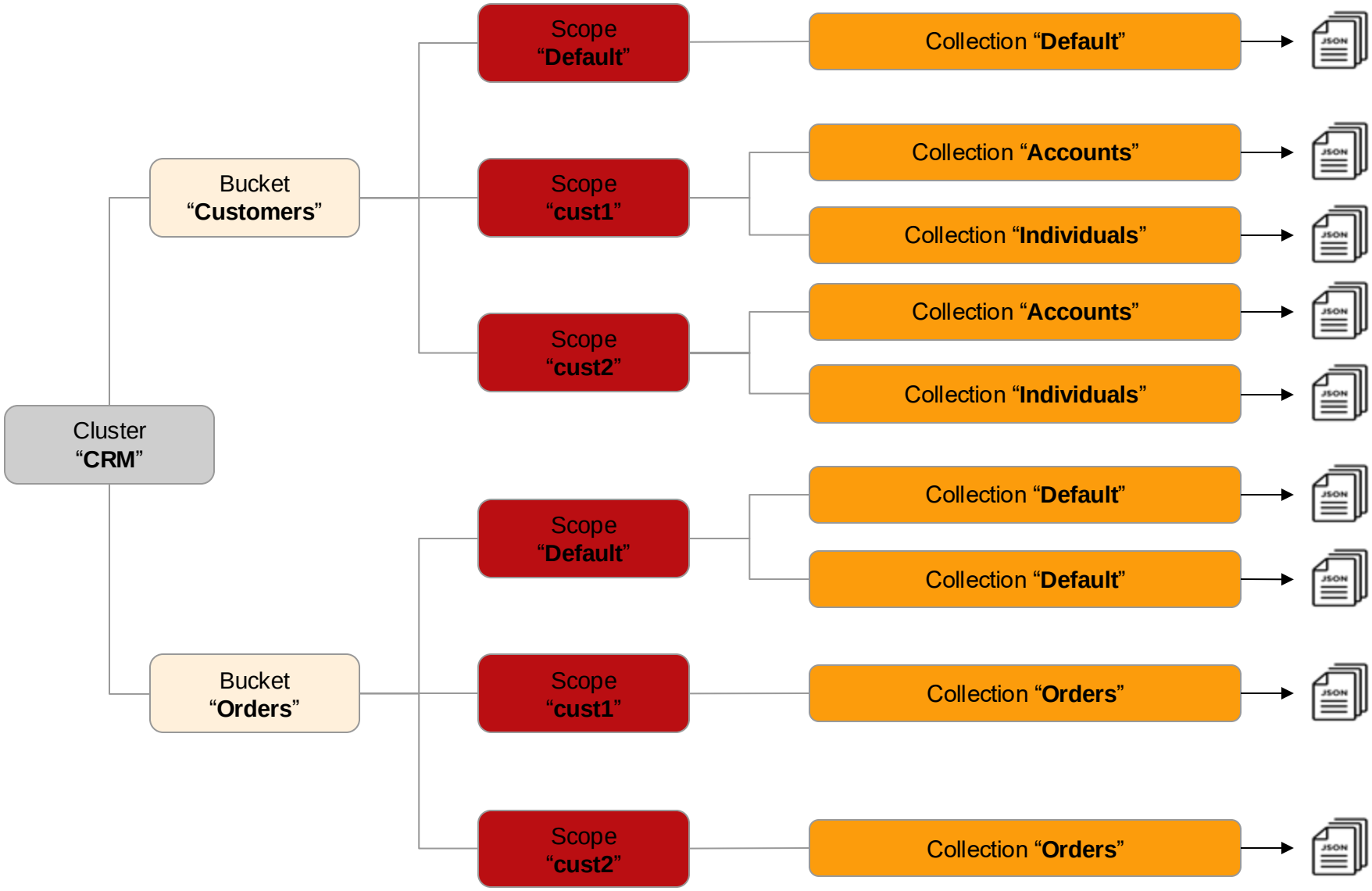
- Provides access to **bucket level** operations(데이터베이스)
- Manages persistence, replicas, TTL, compression mode, XDCR conflict resolution, ejection policy, compaction frequency

- Unique **keyspace of Collections**(스키마)
- Applications can be assigned per-scope access-rights

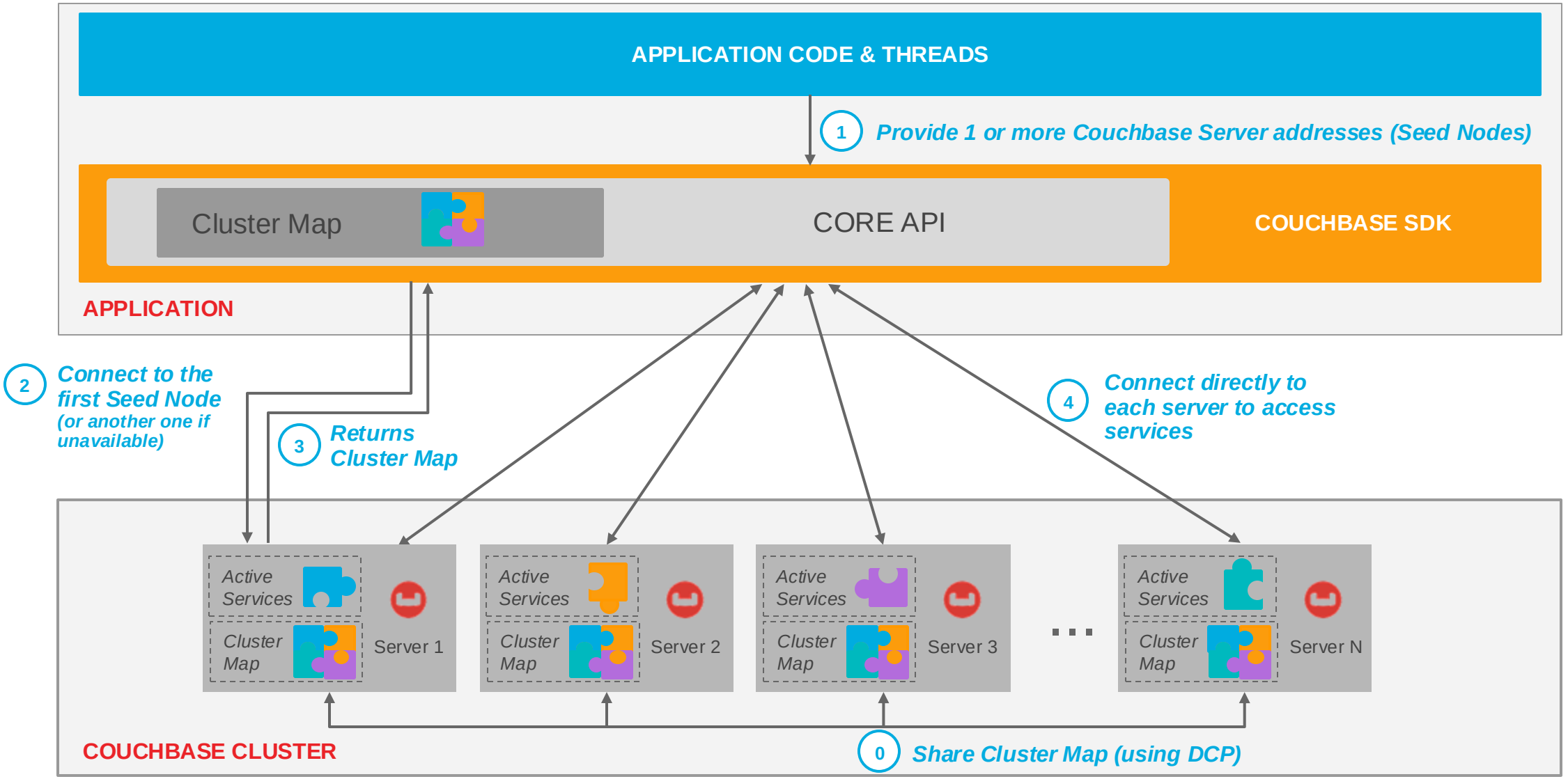
- Unique **keyspace of Documents**(테이블)
- Collections might be assigned to different scopes.

- JSON: numbers, strings, embedded documents and arrays(로우)
- Non JSON: UTF8, buffer, bytearray, serialized objects

SDK 아키텍처 : 엔티티 계층구조



SDK 아키텍처 : Connection 세부 단계(Bootstrapping)



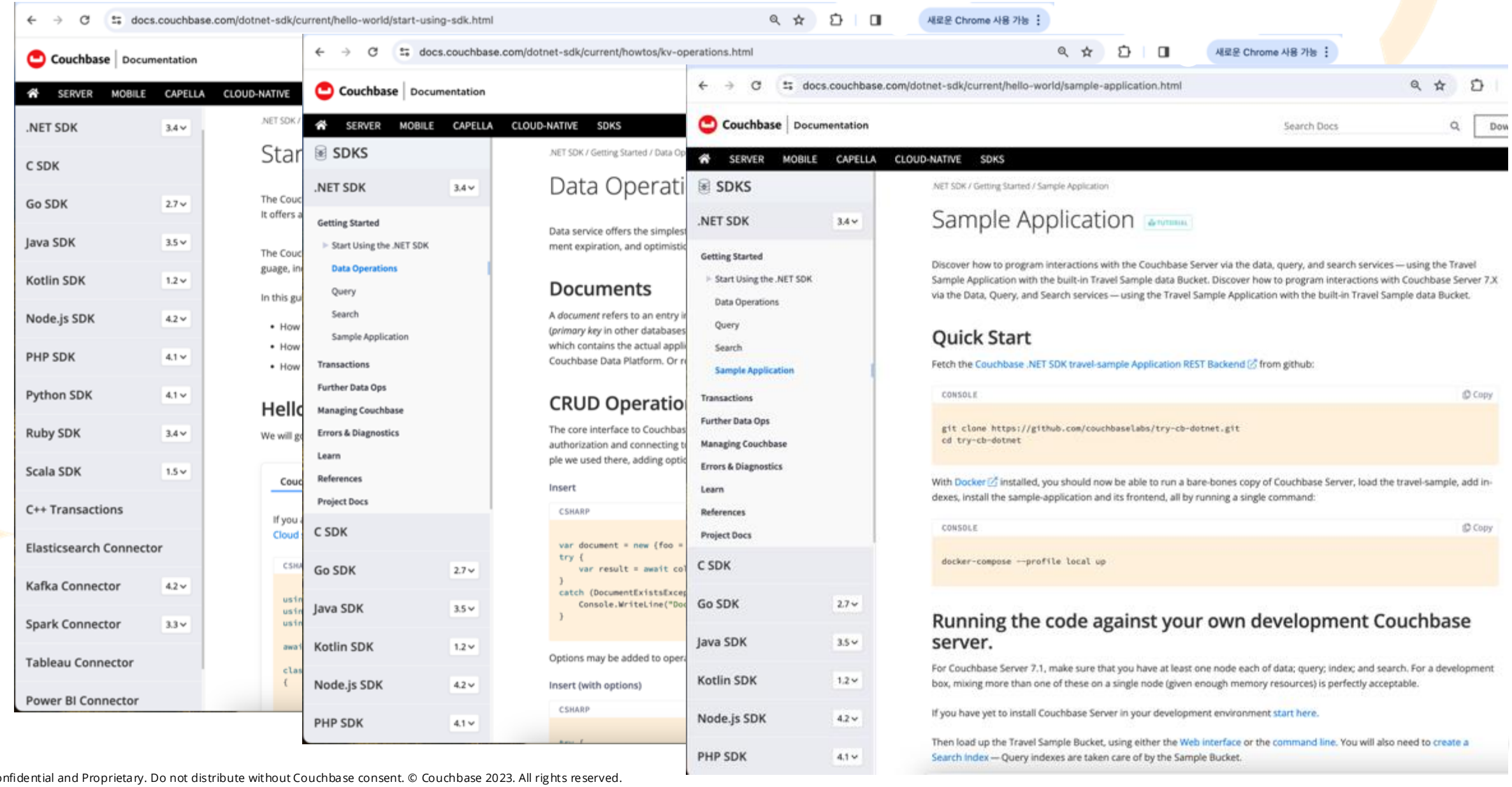
Sample Code : Synchronous

```
Cluster c = Cluster.connect("localhost", "username","password");
Bucket b = c.bucket("travel-sample");
Collection col = b.defaultCollection();

// KV
MutationResult mr = col.upsert("doc1", JsonObject.create().put("name","mike"));
GetResult d1 = col.get("doc1");
String name = d1.contentAsObject().getString("name");
System.out.println(name); // name == "mike"

// SQL++ query
final QueryResult qr = cluster.query("select * from `travel-sample` limit 10",
                          queryOptions().metrics(true));
for (JsonObject row : qr.rowsAsObject()) {System.out.println("Row: " + row);}
System.out.println("Exec time:“+qr.metaData().metrics().get().executionTime());

// Search query
final SearchResult sr = cluster.searchQuery("airportIdx",
                        SearchQuery.prefix("LAX"),
                        searchOptions().fields("name"));
for (SearchRow row : sr.rows()){
      System.out.println("Score/ID: "+row.score(+”/”)+row.id());
}

// Analytics query
AnalyticsResult ar = cluster.analyticsQuery(
  "select count(*) from airports where country = \”France\”");
for (JsonObject row : ar.rowsAsObject()) { System.out.println("Row: "+row);}
```

Sample Code : Connections

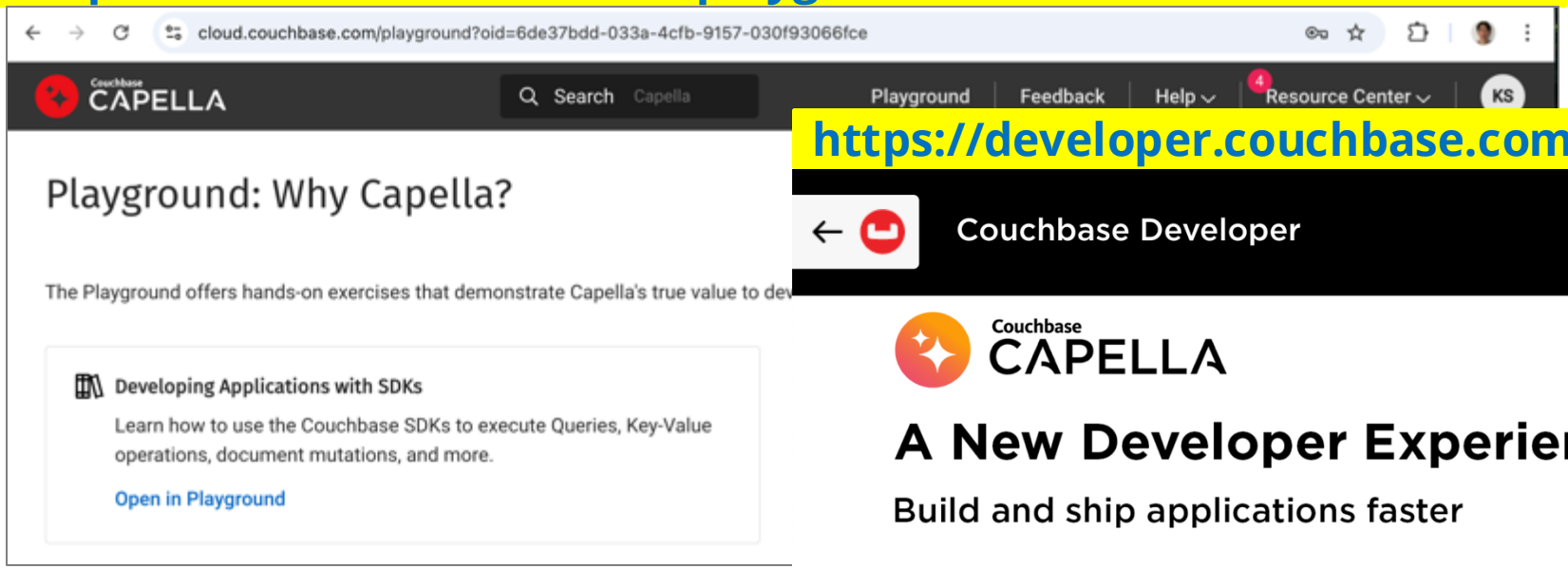
```
static String endpoint = "couchbase://10.0.0.183,10.0.10.92";
static String username = "Administrator";
static String password = "P@ssw0rd";
static String bucketName = ”travel-sample";
static String scopeName = ”inventory";
static String collectionName = ”airport";

ClusterEnvironment env = ClusterEnvironment.builder()
                    .timeoutConfig(TimeoutConfig.kvTimeout(Duration.ofSeconds(15))
                                   .kvDurableTimeout(Duration.ofSeconds(15)))
                    .thresholdLoggingTracerConfig(config)
                    .build();

Cluster cluster = Cluster.connect(endpoint, ClusterOptions.clusterOptions(username, password)
        // Use the pre-configured profile below to avoid latency issues with your connection.
      .environment(env)
    );

// get a bucket reference
Bucket bucket = cluster.bucket(bucketName);
bucket.waitUntilReady(Duration.ofSeconds(30));
Scope scope = bucket.scope(scopeName);
Collection collection = scope.collection(collectionName);
```

개발 지원 문서

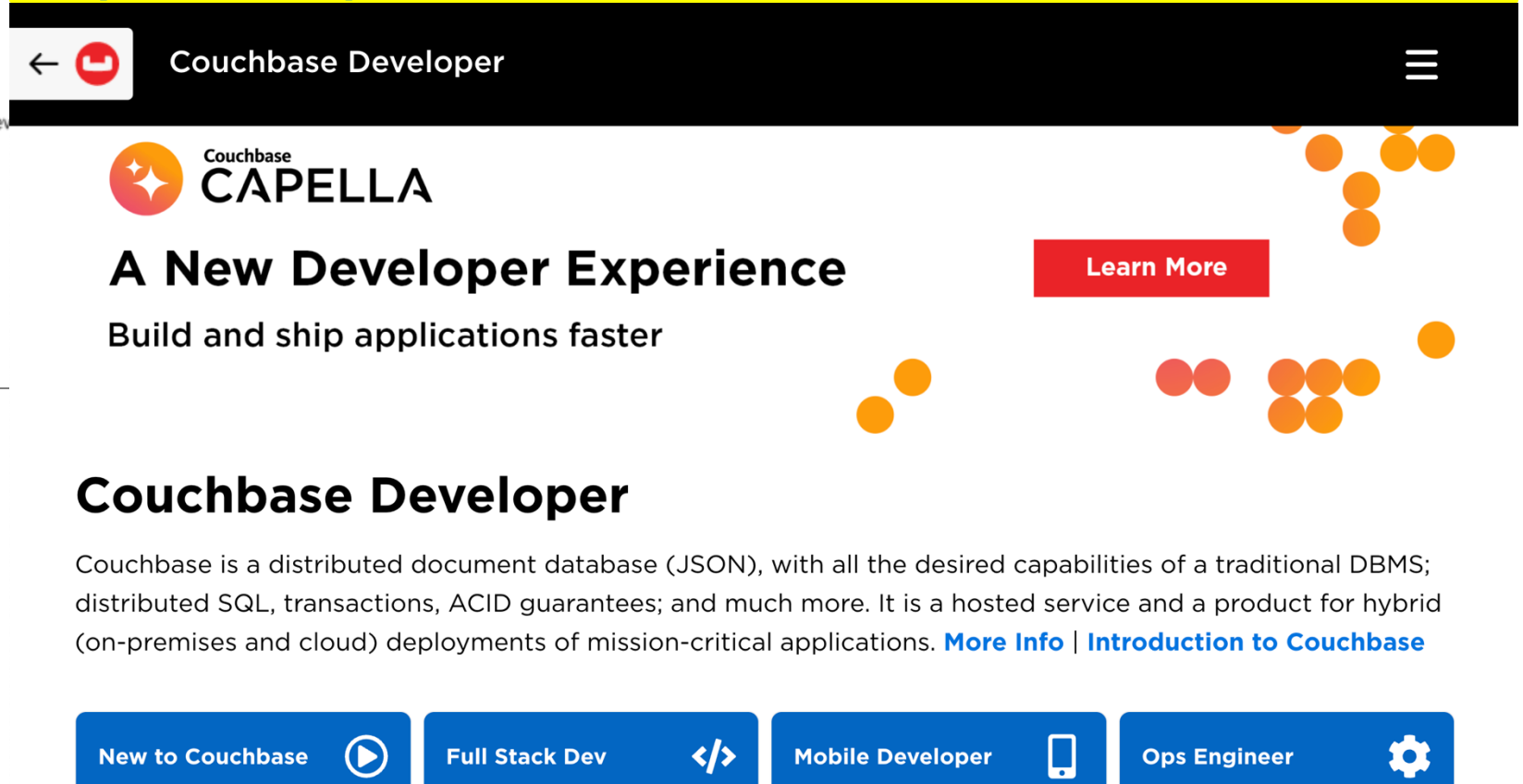


개발자 포털/플레이그라운드

https://cloud.couchbase.com/playground



https://developer.couchbase.com/tutorials

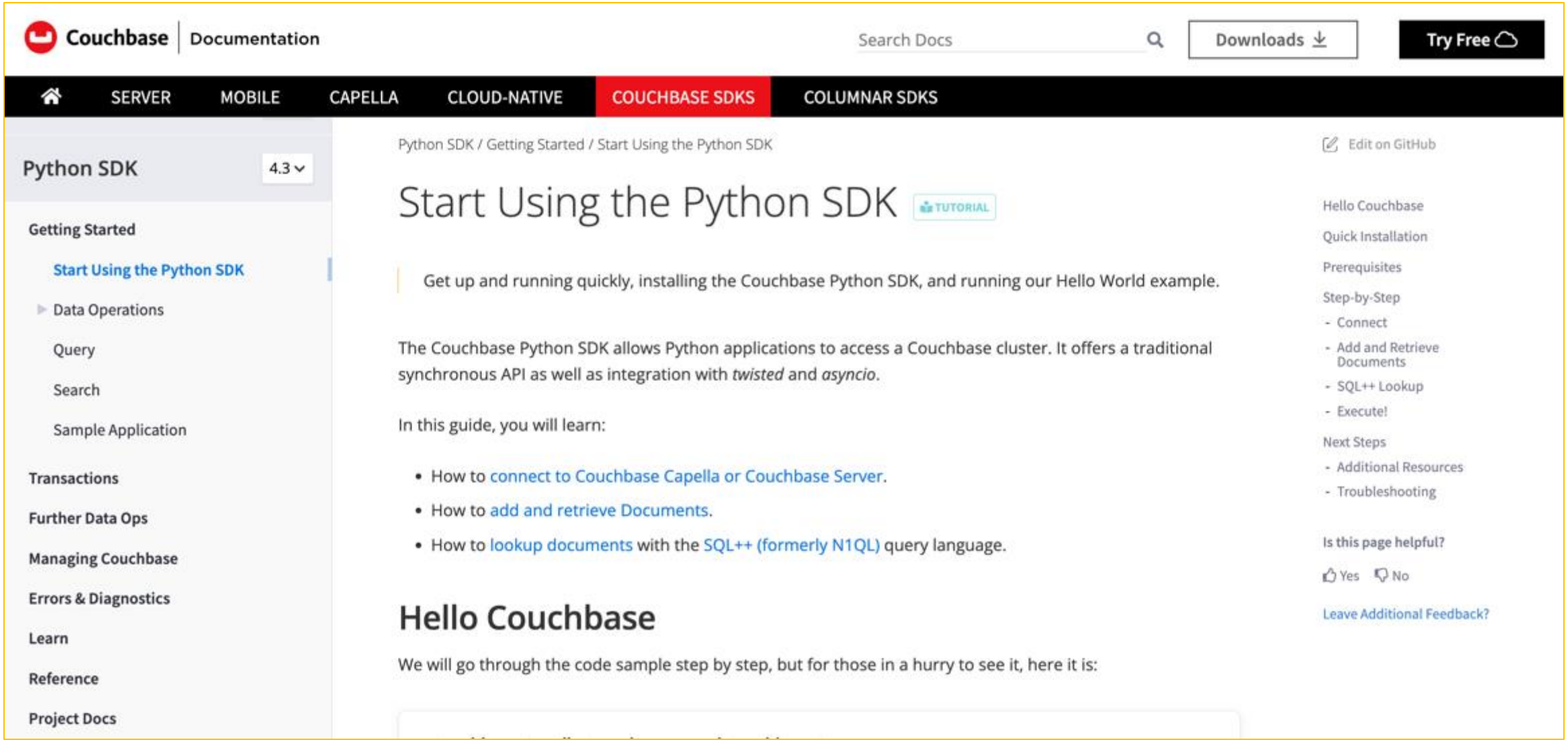


2-3.
SDK 실습 (Python, Java)

실습

https://docs.couchbase.com/java-sdk/current/hello-world/start-using-sdk.html
https://docs.couchbase.com/python-sdk/current/hello-world/start-using-sdk.html

KV Rangescan/Prefixscan
동일한 SQL Query



SQL Sample from Capella Playground

```
SELECT route.airline, route.schedule, route.sourceairport
FROM route
WHERE route.sourceairport = "JFK"
ORDER BY route.airline LIMIT 5;
```

```
SELECT h.city, AVG(r.ratings.Overall) AS hotel_rating
FROM hotel AS h
UNNEST h.reviews AS r
WHERE h.country = "United Kingdom"
GROUP BY h.city
HAVING AVG(r.ratings.Overall) > 4.9
```

```
SELECT DISTINCT airline.name, airline.callsign, route.schedule,
route.sourceairport, route.destinationairport
FROM route
INNER JOIN airline
ON route.airlineid = META(airline).id
WHERE route.sourceairport = "JFK"
AND route.destinationairport = "SFO"
AND airline.callsign = "UNITED";
```

```
SELECT h.name, h.address
FROM hotel h
WHERE h.city = 'San Francisco'
AND ANY r IN h.reviews
SATISFIES r.ratings.Overall > 3 END
```

```
SELECT h.city, h.name,
AVG(r.ratings.Overall) AS avg_rating,
RANK() OVER (PARTITION BY h.city ORDER BY AVG(r.ratings.Overall) DESC) as city_rank
FROM hotel as h
UNNEST h.reviews r
WHERE h.country = "France"
GROUP BY h.city, h.name
```

SQL Sample from Capella Playground

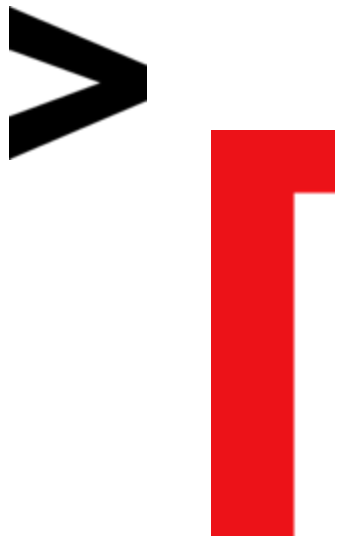
```
WITH AirlineDestinations AS (
  SELECT r.destinationairport,
  ARRAY_COUNT(r.schedule) AS flights
  FROM `travel-sample`.inventory.route AS r
  WHERE r.airline = "UA"
  GROUP BY r.destinationairport, r.schedule
)
SELECT ad.destinationairport, SUM(ad.flights) AS total_flights
FROM AirlineDestinations AS ad
GROUP BY ad.destinationairport
ORDER BY ad.destinationairport;
```

```
SELECT h.city,
 ARRAY_AGG(h.name)
  FILTER (WHERE ANY AND EVERY r IN h.reviews
  SATISFIES r.ratings.Overall >= 3 END) AS good_hotels,
 ARRAY_AGG(h.name)
  FILTER (WHERE ANY r IN h.reviews
  SATISFIES r.ratings.Overall < 3 END) AS other_hotels
FROM hotel as h
GROUP BY h.city;
```

```
select a.name, count(1) as numRoutes
from route r
join airline a on r.airlineid = meta(a).id
group by a.name
having count(1) > 2000
order by count(1) desc;
```

```
select meta().id as _id, * from route
```

Appendix.
SQL Samples



SQL++ Queries design

Modelling data using Collections simplifies queries and indexes

```
CREATE BUCKET CRM;
set query_context = "default:CRM._default"

CREATE COLLECTION Orders;
CREATE COLLECTION  Individuals;
```

Set query context. Here `_default` scope of bucket `CRM`.

Create collections in that context

```
INSERT INTO Individuals (key,value)
VALUES ("Ind:110", { "Id": "110","Name": "Mary Joe",
                     "Email": "mj@email.com" });

INSERT INTO Orders (key,value)
VALUES ("Ord:123", { "Id": "123", "Indid": "110",
        "Items": [ "Qty": "1","Name": "Shoes XX"}] });
```

Insert Json docs into collections.
There is no `type` field.

```
CREATE INDEX Orders_Indid ON Orders(Indid);
CREATE INDEX Individuals_Id ON Individuals(Id);
```

Create indexes into a collection.
There is no `WHERE type='Orders'`.

```
SELECT i.Name, o.Items
FROM Orders o
JOIN Individuals i ON i.Id = o.Indid;
```

Query with JOIN between collections

SQL-like queries | MERGE

Finds all routes for airline BA whose source airport is in France.

- *If any flights are using equipment 319, they are updated to use equipment 797.*
- *If any flights are using equipment 757, they are deleted.*

Collection: **'route'**

```
{
  "id": "14452",
  "type": "route",
  "airline": "BA",
  "airlineid": "airline_1355",
  "sourceairport": "BOD",
  "destinationairport": "LGW",
  "equipment": "734 319",
  "stops": "0",
  "distance": 99.89861063028253
  "schedule": [
      {"day": 0, "utc": "19:41:00", "flight": "BA977"},
      {"day": 0, "utc": "14:03:00", "flight": "BA938"},
      {"day": 1, "utc": "08:40:00", "flight": "BA394"},
      {"day": 1, "utc": "06:54:00", "flight": "BA427"},
      {"day": 1, "utc": "06:00:00", "flight": "BA916"}
  ]
}
```

Collection: **'airport'**

```
{
  "id": 1264,
  "type": "airport",
  "airportname": "Merignac",
  "city": "Bordeaux",
  "country": "France",
  "faa": "BOD",
  "icao": "LFBD",
  "tz": "Europe/Paris",
  "geo": {
    "lat": 44.828335,
    "lon": -0.715556,
    "alt": 162
  }
}
```

***Before updating or deleting** some routes,*
***you first need to JOIN** route and airport because the country of the source airport is in the airport collection*

SQL-like queries | MERGE

```
MERGE INTO `travel-sample`.inventory.route
USING `travel-sample`.inventory.airport
ON route.sourceairport = airport.faa
WHEN MATCHED THEN
  UPDATE
  SET route.old_equipment = route.equipment,
      route.equipment = "797"
  WHERE airport.country = "France" AND route.airline = "BA" AND CONTAINS(route.equipment, "319")
WHEN MATCHED THEN
  DELETE
  WHERE airport.country = "France" AND route.airline = "BA"AND CONTAINS(route.equipment, "757")
RETURNING route.old_equipment, route.equipment, airport.faa;
```

Collection to modify

Collection to join with

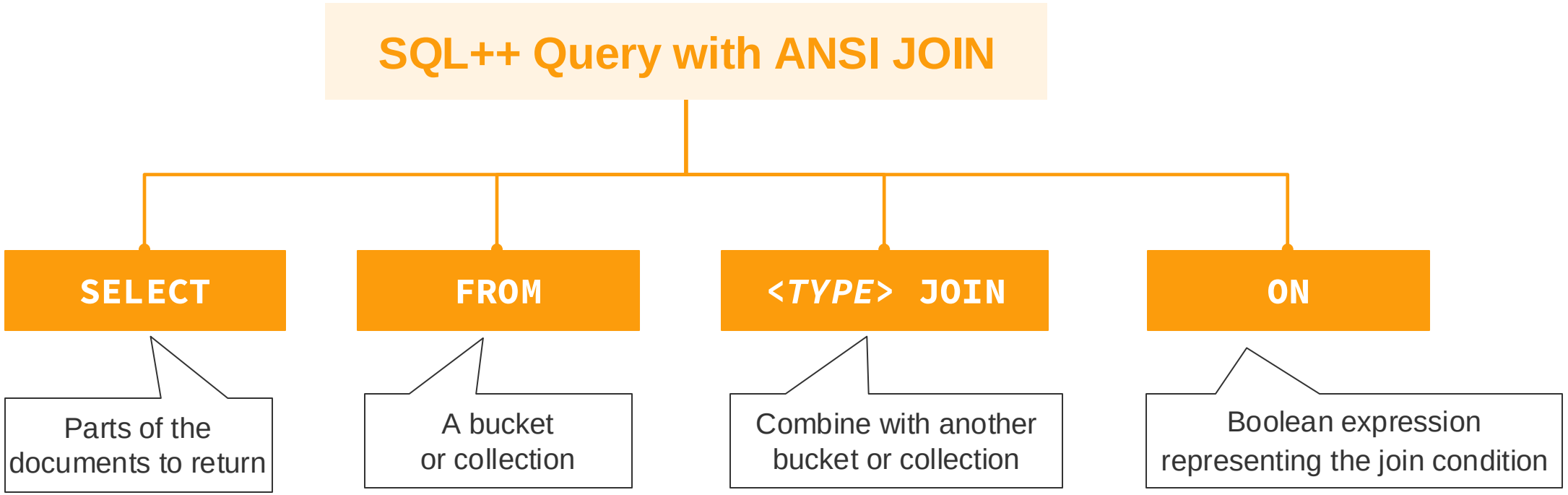
JOIN predicate

Update the routes using equipment 319

Delete routes having 757 equipment

- Provides the ability to modify a collection based on the results of a join with another collection.
- Based on a match or no match in the join, actions can be INSERT, UPDATE, DELETE.
- Multiple actions can be specified.

SQL-like queries | ANSI JOINS



SQL-like queries | ANSI JOINS

JOIN TYPE	Examples
ANSI JOIN	SELECT .. FROM customer c **JOIN** orders o **ON** o.customer_id = c.id
ANSI JOIN Complex	SELECT .. FROM airline **JOIN** route **ON** route.airlineid = "airline_" || tostring(airline.id) **AND** route.type = "route"
ANSI JOIN with IN CLAUSE	SELECT .. FROM `travel-sample` route **JOIN** airport **ON** airport.faa **IN** [route.sourceairport, route.destinationairport] **AND** airport.type = "airport"
ANSI LEFT OUTER	SELECT .. FROM airport **LEFT JOIN** route **ON** airport.faa = route.sourceairport **AND** route.type = "route"
ANSI JOIN with HASH JOIN	SELECT .. FROM airport **JOIN** route **USE HASH(**build**)** **ON** airport.faa = route.sourceairport **AND** route.type = "route"

ANSI JOIN can join arbitrary fields of the documents and can be chained together.

SQL-like queries | Arrays

Retrieve the details of KL flight schedules from Albuquerque (ABQ) to Atlanta (ATL) if any of the flights are after 23:40.

```
SELECT *
FROM `travel-sample`.inventory.route
WHERE airline="KL"
  AND sourceairport="ABQ"
  AND destinationairport="ATL"
  AND ANY departure IN schedule
      SATISFIES departure.utc > "23:40" END;
```


```
[
  {
    "travel-sample": {
      "airline": "KL",
      "airlineid": "airline_3090",
      "destinationairport": "ATL",
      "distance": 2038.3535078909663,
      "equipment": "757 320",
      "id": 36159,
      "schedule": [
        {"day": 0, "flight": "KL938", "utc": "03:54:00"},
        // ...
        {"day": 5, "flight": "KL169", "utc": "23:41:00"},
        // ...
        {"day": 6, "flight": "KL636", "utc": "17:40:00"}
      ],
      "sourceairport": "ABQ",
      "stops": 0,
      "type": "route"
    }
  }
]
```


- Range predicates (ANY, EVERY) enable you to evaluate expressions over every element in an array
- They are particularly useful when used to evaluate expressions over an array of objects

SQL-like queries | Subqueries

Find total number of airports by country where each city has more than 5 airports.

```
SELECT t1.country,
       array_agg(t1.city),
       sum(t1.city_cnt) as apnum
FROM
    (SELECT city,
            city_cnt,
            array_agg(airportname) as apnames,
            country
     FROM `travel-sample`.inventory.airport
     GROUP BY city,
            country LETTING city_cnt = count(city)
     ) AS t1
WHERE t1.city_cnt > 5
GROUP BY t1.country;
```


```
[
  {
    "$1": [
      "Paris"
    ],
    "apnum": 9,
    "country": "France"
  },
  {
    "$1": [
      "London"
    ],
    "apnum": 13,
    "country": "United Kingdom"
  },
  {
    "$1": [
      "Houston",
      "New York",
      "San Diego"
    ],
    "apnum": 22,
    "country": "United States"
  }
]
```


- A subquery is a query within another query
- Subqueries can be embedded anywhere a valid expression can go

SQL-like queries | Built-in Functions

Category	Functions
Aggregate functions	ARRAY_AGG() ARRAY_AGG(DISTINCT) AVG(), AVG(DISTINCT) COUNT() COUNT(DISTINCT) MAX() MIN() SUM() SUM(DISTINCT)
Object functions	OBJECT_LENGTH() OBJECT_NAMES() OBJECT_PAIRS() OBJECT_INNER_PAIRS() OBJECT_VALUES() OBJECT_INNER_VALUES() OBJECT_ADD() OBJECT_PUT() OBJECT_REMOVE() OBJECT_UNWRAP()
Array functions	ARRAY_APPEND() ARRAY_AVG() ARRAY_CONCAT() ARRAY_CONTAINS() ARRAY_COUNT() ARRAY_DISTINCT() ARRAY_IFNULL() ARRAY_LENGTH() ARRAY_MAX() ARRAY_MIN() ARRAY_POSITION() ARRAY_PREPEND() ARRAY_PUT() ARRAY_RANGE() ARRAY_REMOVE() ARRAY_REPEAT() ARRAY_REPLACE() ARRAY_REVERSE() ARRAY_SORT() ARRAY_SUM()
Comparison functions	GREATEST() LEAST()
Conditional functions	IFMISSING() IFMISSINGORNULL() IFNULL() MISSINGIF() NULLIF() IFINF() IFNAN() IFNANORINF() NANIF() NEGINFIF() POSINFIF()
Number functions	ABS() ACOS() ASIN() ATAN() ATAN2() CEIL() COS() DEGREES() E() EXP() LN() LOG() FLOOR() PI() POWER() RADIANS() RANDOM() ROUND() SIGN() SIN() SQRT() TAN() TRUNC()
Date functions	CLOCK_MILLIS() CLOCK_STR() DATE_ADD_MILLIS() DATE_ADD_STR() DATE_DIFF_MILLIS() DATE_DIFF_STR() DATE_PART_MILLIS() DATE_PART_STR() DATE_TRUNC_MILLIS() DATE_TRUNC_STR() STR_TO_MILLIS() MILLIS_TO_STR() MILLIS_TO_UTC() MILLIS_TO_ZONE_NAME() NOW_MILLIS() NOW_STR() STR_TO_MILLIS() STR_TO_UTC() STR_TO_ZONE_NAME()
JSON functions	DECODE_JSON() ENCODE_JSON() ENCODED_SIZE() POLY_LENGTH()
Meta and UUID functions	BASE64() BASE64_ENCODE() BASE64_DECODE() META() UUID()
Pattern-matching functions	REG_CONTAINS() REG_LIKE() REG_POSITION() REG_REPLACE()
String functions	CONTAINS() INITCAP() LENGTH() LOWER() LTRIM() POSITION() REPEAT() REPLACE() RTRIM() SPLIT() SUBSTR() TITLE() TRIM() UPPER()
Type-Checking Functions	ISARRAY() ISATOM() ISBOOLEAN() ISNUMBER() ISOBJECT() ISSTRING() TYPE()
Type-Conversion Functions	TOARRAY() TOATOM() TOBOOLEAN() TONUMBER() TOOBJECT() TOSTRING()

SQL-like queries | User Defined Functions (UDF)

Simple Inline Example

```
CREATE FUNCTION to_meters(...) {
  args[0] * 0.3048
};
```

```
SELECT airportname, ROUND(to_meters(geo.alt)) AS mamsl
FROM `travel-sample`.inventory.airport
LIMIT 5;
```


```
[
  {"airportname": "Calais Dunkerque","mamsl": 4},
  {"airportname": "Peronne St Quentin","mamsl": 90},
  {"airportname": "Les Loges","mamsl": 130},
  {"airportname": "Couterne", "mamsl": 219},
  {"airportname": "Bray","mamsl": 111}
]
```


Inline Example with SQL++ query inside

```
CREATE FUNCTION locations(vActivity) {
  (SELECT id, name, address, city
  FROM `travel-sample`.inventory.landmark
  WHERE activity = vActivity) };
```

```
SELECT l.name, l.city
FROM locations("eat") AS l
WHERE l.city = "Gillingham";
```


```
[
  {"city": "Gillingham","name": "Hollywood Bowl"},
  {"city": "Gillingham","name": "Thai Won Mien"},
  {"city": "Gillingham","name": "Spice Court"},
  {"city": "Gillingham","name": "Beijing Inn"},
  {"city": "Gillingham","name": "Ossie's Fish and
Chips"}
]
```

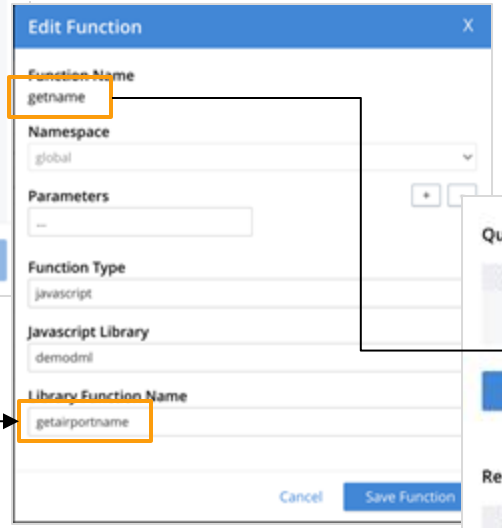

- Inline functions are defined using SQL++ expressions, including subqueries.
- You can name and reuse complex or repetitive expressions in order to simplify your queries.

SQL-like queries | UDF with JavaScript & SQL++

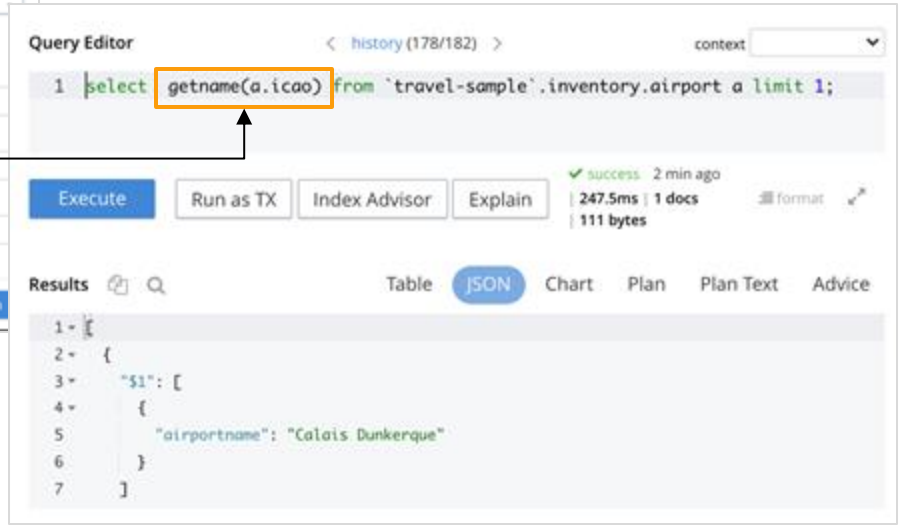
Create a Javascript function in a Library

Javascript library

Create a UDF Function that references it



Use the UDF function in SQL++ statements



- UDFs with JavaScript allows developers to provide custom functions to extend SQL++ capabilities

SQL-like queries | Search Functions

Find the name of the hotels in United Kingdom where the reviews match the term 'bathrobes'

```
SELECT t1.name, meta().id
FROM `travel-sample`.inventory.hotel AS t1
WHERE SEARCH(t1, {
  "match": "bathrobes",
  "field": "reviews.content",
  "analyzer": "standard"
})
AND country="United Kingdom"
LIMIT 3;
```

```
[
  {
    "id": "hotel_12068",
    "name": "Castle Hotel"
  },
  {
    "id": "hotel_18819",
    "name": "Bistro Prego With Rooms"
  },
  {
    "id": "hotel_3622",
    "name": "Premier Inn Birmingham Central East"
  }
]
```

- Search functions enable you to use full text search (FTS) queries directly within a SQL++ query.
- It is recommended that you create suitable full text indexes for the searches that you need to perform.

SQL-like queries | Prepared Statement

Positional Parameters

```
PREPARE NumParam AS
SELECT * FROM `travel-sample`.inventory.hotel
WHERE city=$1 AND country=$2;
```

```
EXECUTE NumParam
USING ["Paris", "France"];
```

Named Parameters

```
PREPARE NameParam AS
SELECT * FROM `travel-sample`.inventory.hotel
WHERE city=$city AND country=$country;
```

```
EXECUTE NameParam
USING {"city": "Paris", "country": "France"};
```

- You can add placeholder parameters to a statement, so that you can supply variable values when you run the statement.
- If you need to run a statement more than once, you can prepare the execution plan for the statement.

SQL-like queries | Transactions

Ensures database consistency when multiple documents are updated in a single or multiple SQL++ statements

```
START TRANSACTION;
UPDATE customer SET balance = balance + 100 WHERE cid = 4872;
SELECT cid, name, balance  FROM customer;
SAVEPOINT s1;
UPDATE customer SET balance = balance - 100 WHERE cid = 1924;
SELECT cid, name, balance FROM customer;
ROLLBACK WORK TO SAVEPOINT s1;
SELECT cid, name, balance FROM customer;
COMMIT;
```

Couchbase provides the following statements for transactions:

- BEGIN TRANSACTION
- SET TRANSACTION (optional)
- SAVEPOINT
- ROLLBACK TRANSACTION
- COMMIT TRANSACTION

수고하셨습니다.

paul.son@couchbase.com

www.couchbase.com

cloud.couchbase.com

Couchbase