



# 데이터 처리 성능 최적화를 통한 카드데이터 분석 대시보드 구현

하나금융티아이 교육생  
광명융합기술교육원 데이터분석과



# 1. 목표



빅데이터를 처음 마주했었던 기억



미래 빅데이터 엔지니어의 꿈



수많은 데이터가 쌓이는 카드업계 선택

## 2. 주요 기능



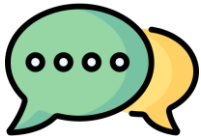
### 데이터 시각화

- 일일 집계 / 기간 집계



### 군집별 집계

- 기존 손님의 데이터로 군집분석
- 군집 별 특징 요약



### 대출만기 안내 카카오톡 메시지 서비스



### 월 카드결제내역 리포트 메일링 서비스

- 한 달 간의 고객 소비 데이터를 리포팅
- 유사 군집과의 비교 제공



### 제플린 서버

- 다양한 인터프리터를 제공하는 제플린을 활용,  
실무자가 직접 다양한 언어로 쿼리 작성 가능

### 3. 요구사항

번호	구분	내용
L01	대출정보페이지	전일에 발생한 신규 대출(리볼빙, 현금서비스, 카드론) 및 연체 발생 건 수를 조회
L02	대출정보페이지	직전 7일 간의 신규 대출(리볼빙, 현금서비스, 카드론) 및 연체 발생 건 수의 추이 조회
L03	대출정보페이지	직전 7일 간의 신규 대출(리볼빙, 현금서비스, 카드론) 의 규모 조회
L04	대출정보페이지	대출(리볼빙, 현금서비스, 카드론) 만기일이 7일 이내인 손님 조회, 카카오톡 안내 발송
P01	고객/결제정보페이지	전일에 가입한 손님의 수, 남/여/연령대 비율 조회
P02	고객/결제정보페이지	전주(D-15 ~ D-8) 대비 직전 7일(D-8 ~ D-1)의 카드결제금액 추이 조회
P03	고객/결제정보페이지	직전 7일 간의 일시불/할부/해외/국내/온라인/오프라인 결제 비율 조회
P04	고객/결제정보페이지	전일 가장 많이 사용된 카드 TOP 5 조회
P05	고객/결제정보페이지	전일 연령대별 카드 결제금액 조회
P06	고객/결제정보페이지	전일 가장 많이 사용된 업종 TOP 6 조회
P07	고객/결제정보페이지	전일 일시불/할부/해외/국내/온라인/오프라인 결제 별 가장 많이 사용된 카드 조회
P08	고객/결제정보페이지	전일 시도별 사용된 금액 조회

### 3. 요구사항

번호	구분	내용
C01	군집정보페이지	군집별 평균 소득/연령/사용금액 및 주 사용 카드, 사용처와 사용횟수 조회
R01	메일링서비스페이지	손님의 지난 사용일까지의 결제내역, 잔여할부내역, 결제해야할 금액 조회
R02	메일링서비스페이지	손님과 동일한 군집의 평균 결제금액, 주 사용카드 및 하나카드 이벤트 조회
T01	공통	사용자가 메모할 수 있는 메모 기능 및 메모 삭제, 조회
T02	공통	Service를 포인트컷으로 하는 디버깅 로그 기록
T03	공통	사용자가 직접 쿼리를 사용할 수 있는 제플린 서버 연결

# 4. 개발환경 / 도구

## 개발 환경

JDK 1.8

Apache Tomcat 9

ORACLE 12c EE

Anaconda 2020.02 (base = python 3.7)

Ubuntu 20.04 LTS

Apache Spark 2.4.6



## 개발 도구

VMware Workstation Pro

ECLIPSE

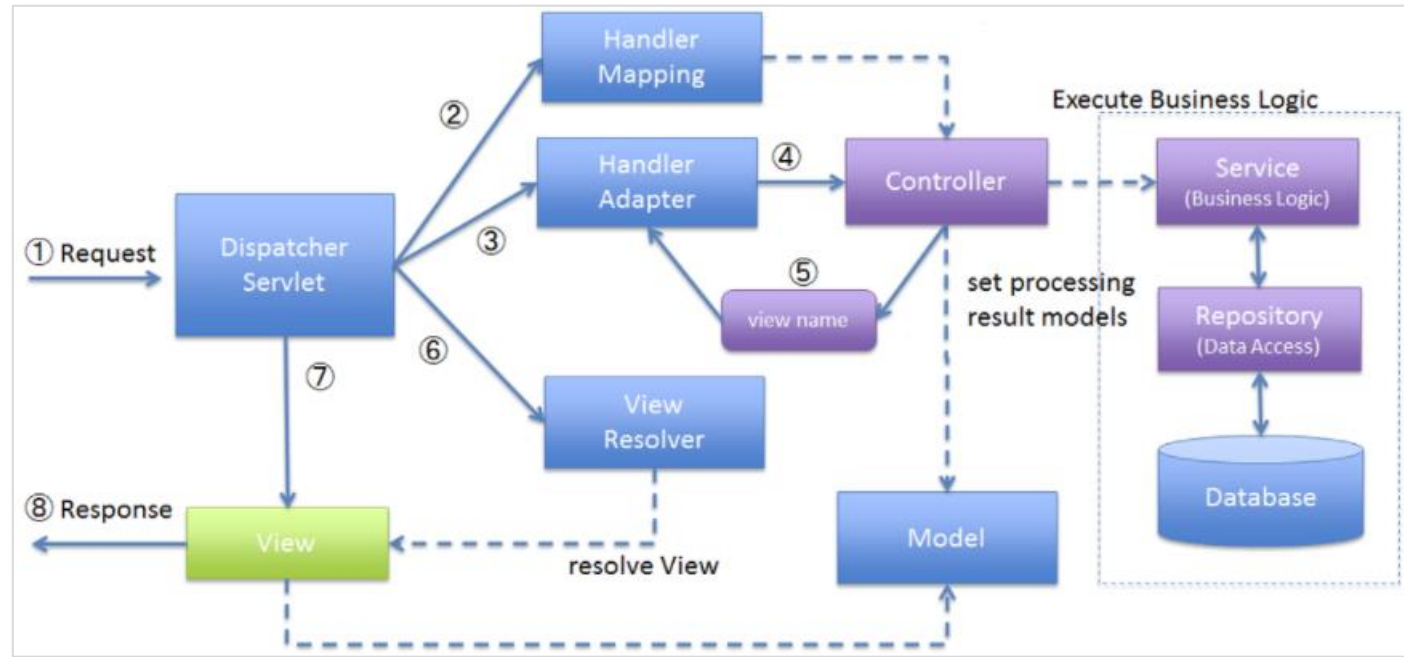
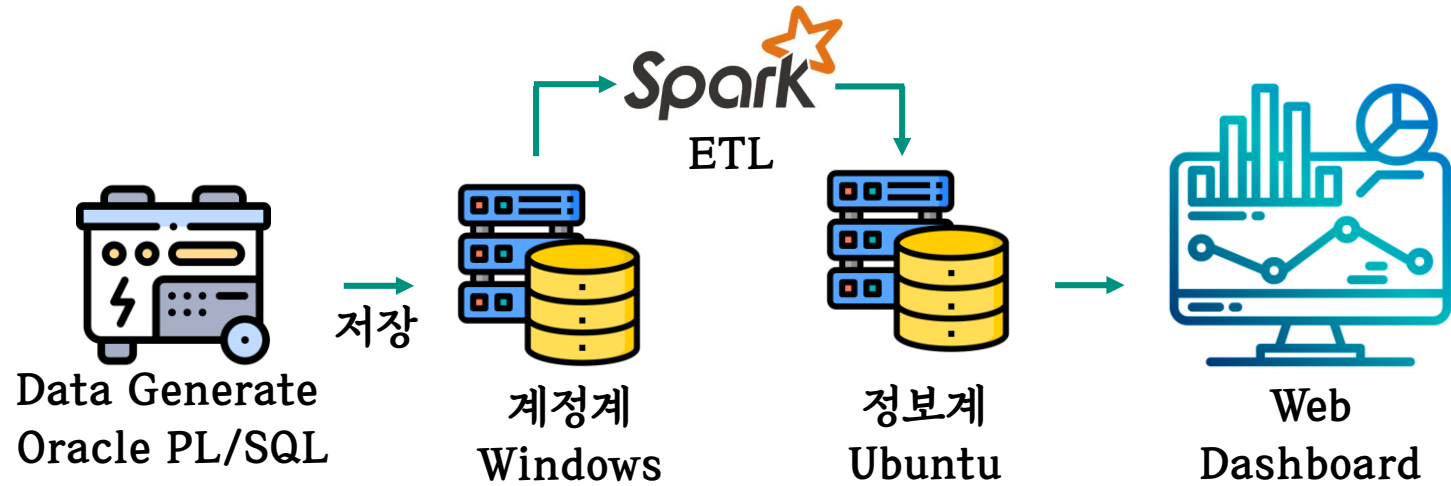
SQL Developer

Docker

Apache Zeppelin



# 5. 설계도



# 목차

1. 계정계 모델링



2. 정보계 모델링



3. ETL & 최적화



4. 쿼리, DB 튜닝



5. 웹 기능 시연



# 5. 계정계 모델링 & 데이터 생성



## 미니멀한 계정계 모델링

+

2020년 3월 - 2020년 10월 / 고객 60만 명이 40회 이상을 결제

⇒ 2800만 행의 카드 결제 내역 데이터 생성

FUNCTION을 사용한 불균등한 데이터 생성

Bulk Binding / Insert 활용

예상시간	실제 소요 시간	절약한 시간
48.7시간	43.1시간	<b>5.6시간</b>

# 목차

1. 계정계 모델링



2. 정보계 모델링



3. ETL & 최적화

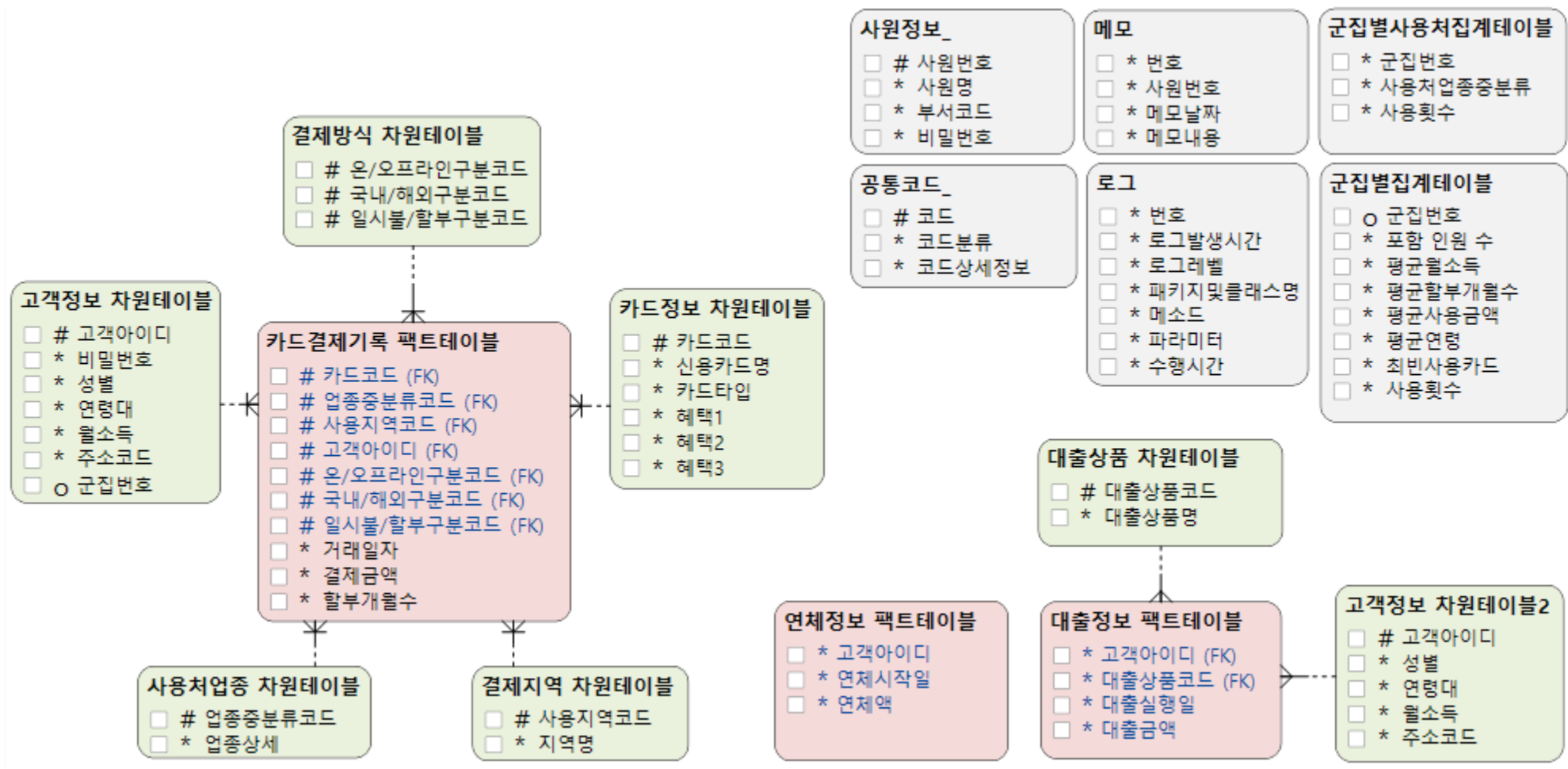


4. 쿼리, DB 튜닝



5. 웹 기능 시연

# 6. 정보계 (DW) 스타스키마



# 목차

1. 계정계 모델링



2. 정보계 모델링



**3. ETL & 최적화**

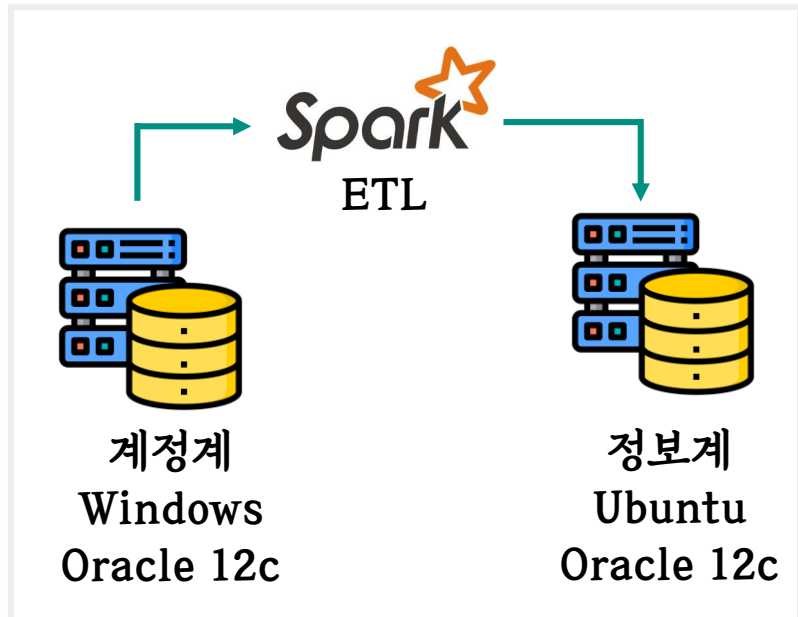


4. 쿼리, DB 튜닝



5. 웹 기능 시연

# 7. ETL & 최적화



많은 데이터 중 원하는 데이터를 추출 (Extract)

요구하는 형태로 변형 (Transform)

새로운 저장소로 적재 (Load)



인메모리 처리로 빠른 Spark 사용

Standalone 환경 구축



6G RAM

3 Thread



6G RAM

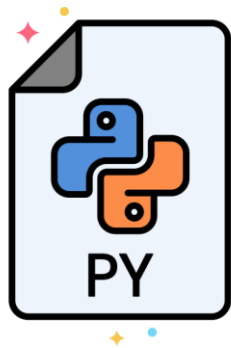
3 Thread



6G RAM

3 Thread

# 7. ETL & 최적화



Star\_Schema\_ETL.py

Clustering.py

파이썬 스크립트 작성



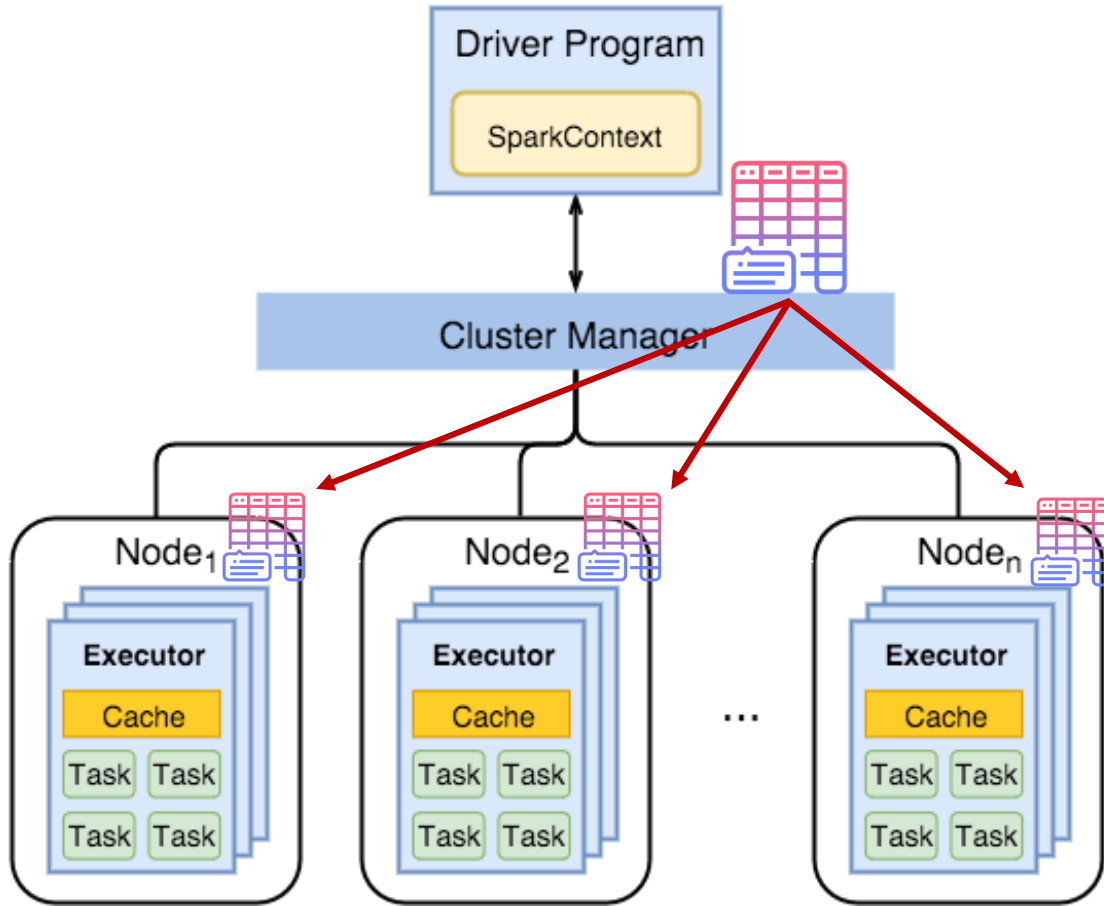
Processing time : 2635.967

ETL 작업이 **약 44분 소요**

생각보다 빠르진 않다?

⇒ 성능을 올릴 수 있는 방안 모색

# 7. ETL & 최적화



Oracle Default Fetch Size = 10 → 10000

Batch Size = 1000 → 10000

spark.sql.shuffle.partitions = 200 → 9

(집계, shuffle 시 사용될 파티션 수, 1 파티션 = 1 태스크)

불필요하게 데이터를 나누는 작업에 소요되는 I/O 시간 ↓

모든 파티션을 스케줄링 하는 시간 등 부가적 시간 ↓

Processing\_time : 263.308

약 44분 ⇒ 약 4분 30초

# 목차

1. 계정계 모델링



2. 정보계 모델링



3. ETL & 최적화



4. 쿼리, DB 튜닝



5. 웹 기능 시연



# 8. 쿼리 최적화

👤 전일 가입자 수

**2,486명**

♂ 남성 가입자 수 (비율)

**1,484명 (59.7)%**

♀ 여성 가입자 수 (비율)

**1,002명 (40.3)%**

👤 20대 비율

**10.0%**

전체 가입자수 중 248명

👤 30대 비율

**13.8%**

전체 가입자수 중 344명

👤 40대 비율

**36.2%**

전체 가입자수 중 901명

👤 50대 비율

**24.0%**

전체 가입자수 중 597명

👤 60대 비율

**9.9%**

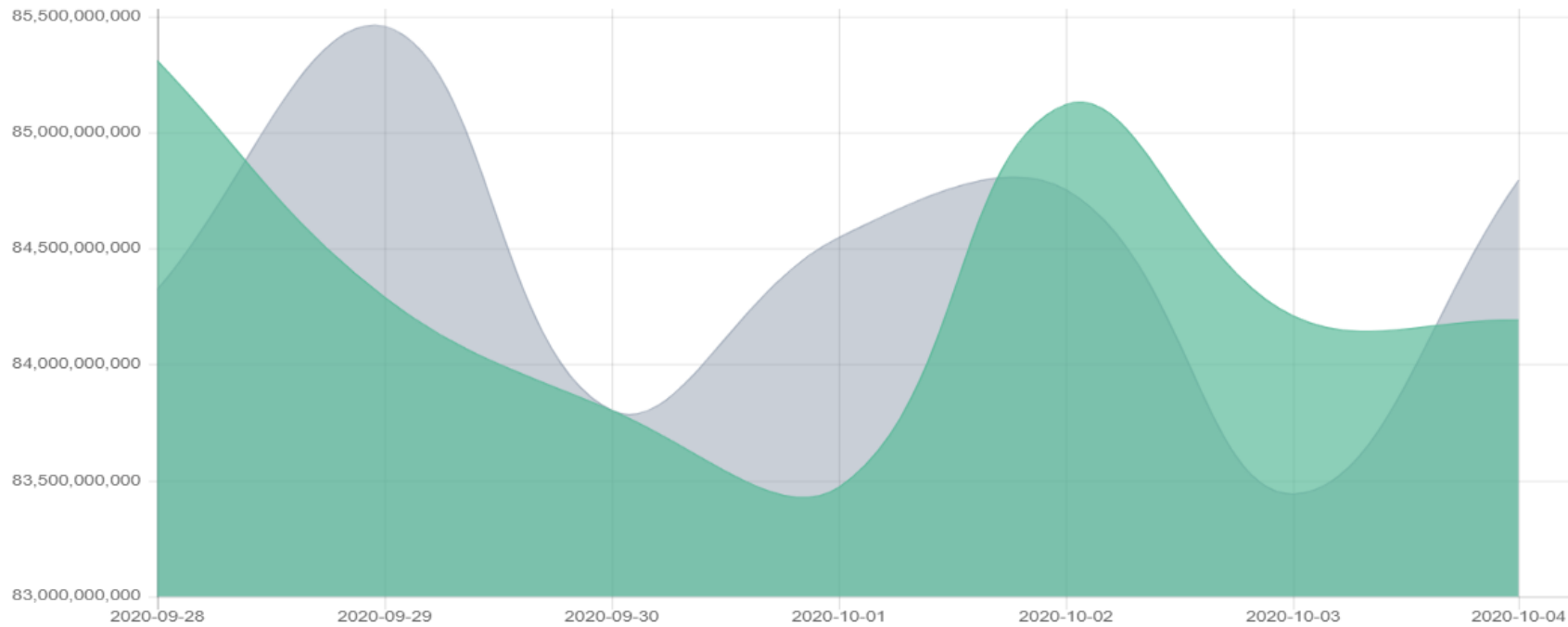
전체 가입자수 중 245명

👤 70대 비율

**6.1%**

전체 가입자수 중 151명

전 주 대비 최근 7일 카드결제금액 추이 녹색: 이번 주



직전 7일간 결제별 비율

일시불결제 비율 (56%)

할부결제 비율 (44%)

해외결제 비율 (23%)

국내결제 비율 (77%)

온라인결제 비율 (63%)

오프라인결제 비율 (37%)



# 8. 쿼리 최적화

전일 가입자 수  
2,486명

남성 가입자 수 (비율)  
1,484명 (59.7)%

여성 가입자 수 (비율)  
1,002명 (40.3)%

20대 비율  
10.0%  
전체 가입자수 중 248명

30대 비율  
13.8%  
전체 가입자수 중 344명

40대 비율  
36.2%  
전체 가입자수 중 901명

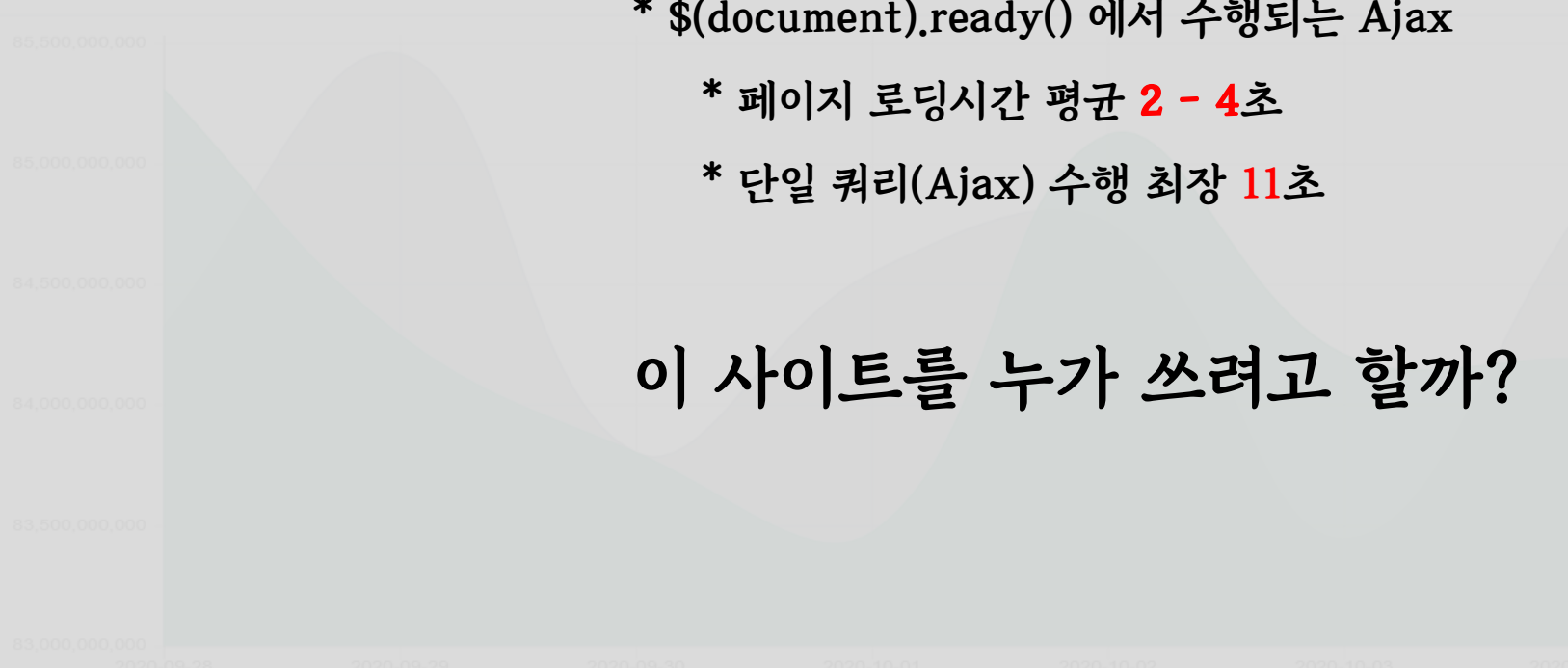
50대 비율  
24.0%  
전체 가입자수 중 597명

60대 비율  
9.9%  
전체 가입자수 중 245명

70대 비율  
6.1%  
전체 가입자수 중 151명

## 페이지 로딩 시간!

전 주 대비 최근 7일 카드결제금액 추이



\* `$(document).ready()` 에서 수행되는 Ajax

\* 페이지 로딩시간 평균 **2 - 4초**

\* 단일 쿼리(Ajax) 수행 최장 **11초**

이 사이트를 누가 쓰려고 할까?

## 직전 7일간 결제별 비율

일시불결제 비율 (56%)

할부결제 비율 (44%)

해외결제 비율 (23%)

국내결제 비율 (77%)

온라인결제 비율 (63%)

오프라인결제 비율 (37%)



## 8. 쿼리 최적화

```
--온/오프라인/할부/일시불/국내/해외
select card_code, max_cnt
from (select card_code, count(*) as max_cnt from fact_payment_hist where on_offline_code = 4000 and transaction_date = to_char(sysdate-1, 'yyyy-mm-dd') group by card_code order by max_cnt desc)
where rownum <= 1
UNION ALL
select card_code, max_cnt
from (select card_code, count(*) as max_cnt from fact_payment_hist where on_offline_code = 4001 and transaction_date = to_char(sysdate-1, 'yyyy-mm-dd') group by card_code order by max_cnt desc)
where rownum <= 1
UNION ALL
select card_code, max_cnt
from (select card_code, count(*) as max_cnt from fact_payment_hist where payment_typecode = 4002 and transaction_date = to_char(sysdate-1, 'yyyy-mm-dd') group by card_code order by max_cnt desc)
where rownum <= 1
UNION ALL
select card_code, max_cnt
from (select card_code, count(*) as max_cnt from fact_payment_hist where payment_typecode = 4003 and transaction_date = to_char(sysdate-1, 'yyyy-mm-dd') group by card_code order by max_cnt desc)
where rownum <= 1
UNION ALL
select card_code, max_cnt
from (select card_code, count(*) as max_cnt from fact_payment_hist where dom_overseas_code = 4004 and transaction_date = to_char(sysdate-1, 'yyyy-mm-dd') group by card_code order by max_cnt desc)
where rownum <= 1
UNION ALL
select card_code, max_cnt
from (select card_code, count(*) as max_cnt from fact_payment_hist where dom_overseas_code = 4005 and transaction_date = to_char(sysdate-1, 'yyyy-mm-dd') group by card_code order by max_cnt desc)
where rownum <= 1;
```

# 8. 쿼리 최적화

```
--온/오프라인/할부/일시불/국내/해외
select card code, max cnt
from (select card_code, count(*) as max_cnt from fact_payment_hist where dom_overseas_code = 4005 and trans
where rownum <= 1;
code order by max_cnt desc)
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				6
UNION-ALL				360016
COUNT		STOPKEY		
Filter Predicates ROWNUM<=1				
VIEW			90993	60004
SORT		ORDER BY STOPKEY	90993	60004
Filter Predicates ROWNUM<=1				
HASH		GROUP BY	90993	60004
TABLE ACCESS	FACT_PAYMENT_HIST	FULL		59998
Filter Predicates				
AND		ON_OFFLINE_CODE=4000 TRANSACTION_DATE=TO_CHAR(SYSDATE@!-1,'yyyy-mm-dd')		

1. AND 연산  
2. TABLE FULL SCAN  
3. HASH GROUP BY (Sort 보장X)  
X 5번 (UNION ALL)

**COST = 1,800,080**

**수행시간 = 11.5초**

# 8. 쿼리 최적화

```
select MIN((decode(online_rank, 1, code_info))) as top_online, MAX(online_cnt) as max_online_cnt,  
MIN((decode(offline_rank, 1, code_info))) as top_offline, MAX(offline_cnt) as max_offline_cnt,  
MIN((decode(inst_rank, 1, code_info))) as top_inst, MAX(inst_cnt) as max_inst_cnt,  
MIN((decode(lumb_sum_rank, 1, code_info))) as top_lumb_sum, MAX(lumb_sum_cnt) as max_lumb_sum_cnt,  
MIN((decode(dom_rank, 1, code_info))) as top_dom, MAX(dom_cnt) as max_dom_cnt,  
MIN((decode(overseas_rank, 1, code_info))) as top_overseas, MAX(overseas_cnt) as max_overseas_cnt  
from (  
select code_info, online_cnt, ROW_NUMBER() OVER (order by online_cnt) as online_rank,  
offline_cnt, ROW_NUMBER() OVER (order by offline_cnt) as offline_rank,  
inst_cnt, ROW_NUMBER() OVER (order by inst_cnt) as inst_rank,  
lumb_sum_cnt, ROW_NUMBER() OVER (order by lumb_sum_cnt) as lumb_sum_rank,  
dom_cnt, ROW_NUMBER() OVER (order by dom_cnt) as dom_rank,  
overseas_cnt, ROW_NUMBER() OVER (order by overseas_cnt) as overseas_rank  
from (  
select code_info, count(decode(on_offline_code, 4000, 1)) as online_cnt,  
count(decode(on_offline_code, 4001, 1)) as offline_cnt,  
count(decode(payment_typecode, 4002, 1)) as inst_cnt,  
count(decode(payment_typecode, 4003, 1)) as lumb_sum_cnt,  
count(decode(dom_overseas_code, 4004, 1)) as dom_cnt,  
count(decode(dom_overseas_code, 4005, 1)) as overseas_cnt  
from fact_payment_hist a, code_tbl b  
where transaction_date = to_char(sysdate-1, 'yyyy-mm-dd')  
and to_char(a.card_code) = to_char(b.code)  
group by code_info  
)  
)
```

4

3

2

1

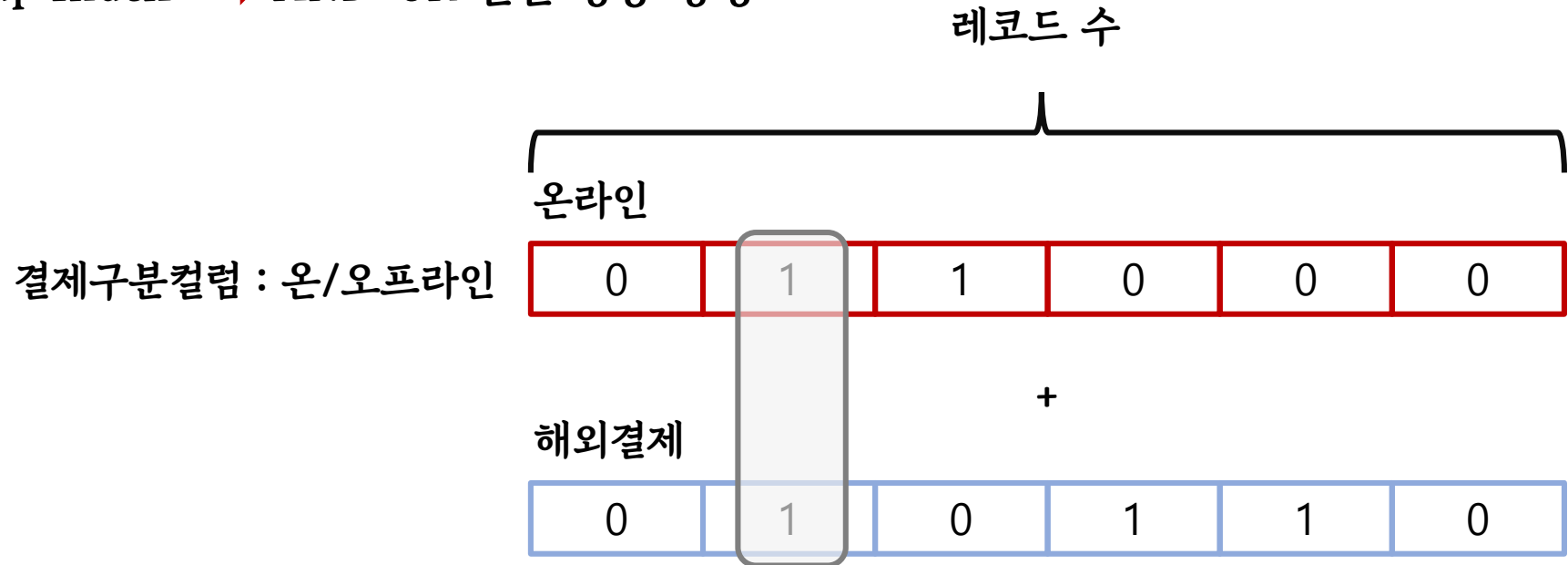
# 8. 쿼리 최적화

분포도가 넓은 컬럼 - index (ex. 날짜 데이터)

Unique한 값이 적고, 분포도가 좁은 컬럼 - Bitmap Index (ex. 성별, 온/오프라인코드 등)

TABLE FULL SCAN → INDEX FULL/RANGE SCAN

Bitmap Index → AND-OR 연산 성능 향상



# 8. 쿼리 최적화

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				59745
SORT		AGGREGATE	1	59745
VIEW			2426	59745
WINDOW			2426	59745
WINDOW			2426	59745
WINDOW			2426	59745
WINDOW			2426	59745
WINDOW			2426	59745
WINDOW			2426	59745
WINDOW			2426	59745
HASH			2426	59745
GROUP BY			2426	59745
HASH JOIN			2426	59737
Access Predicates				
B.CODE=TO_CHAR(A,CARD_CODE)				
TABLE ACCESS	CODE_TBL	FULL	337	3
TABLE ACCESS	FACT_PAYMENT_HIST	BY INDEX ROWID BATCHED	2426	59734
INDEX	IDX_FACT_PAYMENT_HIST_DATE	RANGE SCAN	104124	341
Access Predicates				
SYS_OP_D				
Filter Predica				
SYS_OP_U				

**COST : 1,800,080 → 59,745 (약 97% 감소)**  
**수행시간 : 11.5초 → 1.8초 (약 84% 감소)**

# 목차

1. 계정계 모델링



2. 정보계 모델링



3. ETL & 최적화



4. 쿼리, DB 튜닝



5. 웹 기능 시연



# 하나카드대시보드



## 보완/느낀점

### 1. RT / NRT (Near Real Time)

\* 과거 데이터를 조회하는 대시보드  
-> 카프카 / Spark 스트리밍 보완

### 2. 기획의 중요성

- 첫 기획이 상세하지 않았음  
- 기획이 잘 되어야 짜임새 있을 수 있다.

### 3. 도메인 지식 부족

- 도메인 지식이 있어야  
현업에서 의미있는 집계를 기획 가능

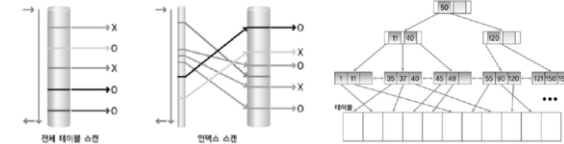
### 4. 알고 넓은 개발

- 대체적으로 알고 넓다고 생각  
- 깊이있는 개발을 하고 싶다

## 비트맵 인덱스 적용

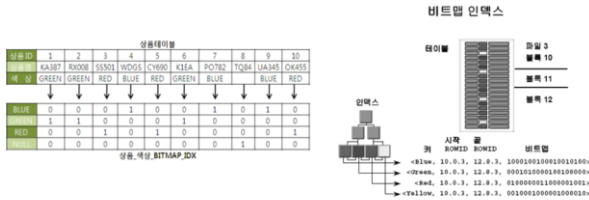


## 인덱스




출처 : 로무는 개발자, DBA 커뮤니티 구루비

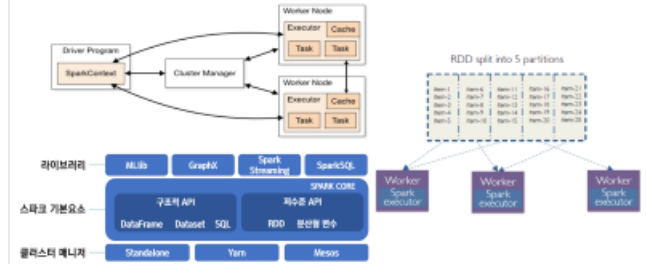
## 비트맵 인덱스



# 감사합니다!

 데이터분석과 소재원

## 스파크



# 보완/느낀점

## 1. RT / NRT (Near Real Time)

- \* 과거 데이터를 조회하는 대시보드
- > 카프카 / Spark 스트리밍 보완

## 2. 기획의 중요성

- 첫 기획이 상세하지 않았음
- 기획이 잘 되어야 짜임새 있을 수 있다.

## 3. 도메인 지식 부족

- 도메인 지식이 있어야
- 현업에서 의미있는 집계를 기획 가능

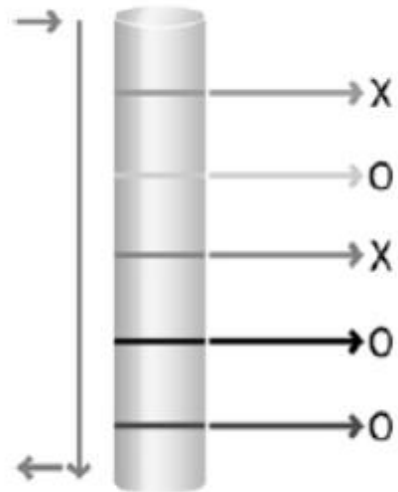
## 4. 얇고 넓은 개발

- 대체적으로 얇고 넓다고 생각
- 깊이있는 개발을 하고 싶다!

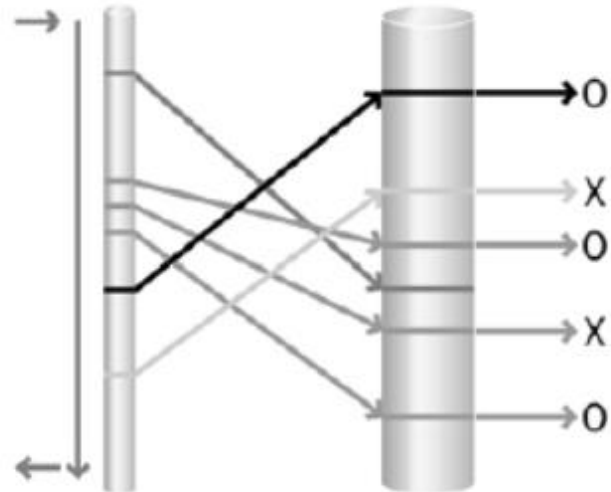
# 비트맵 인덱스 적용



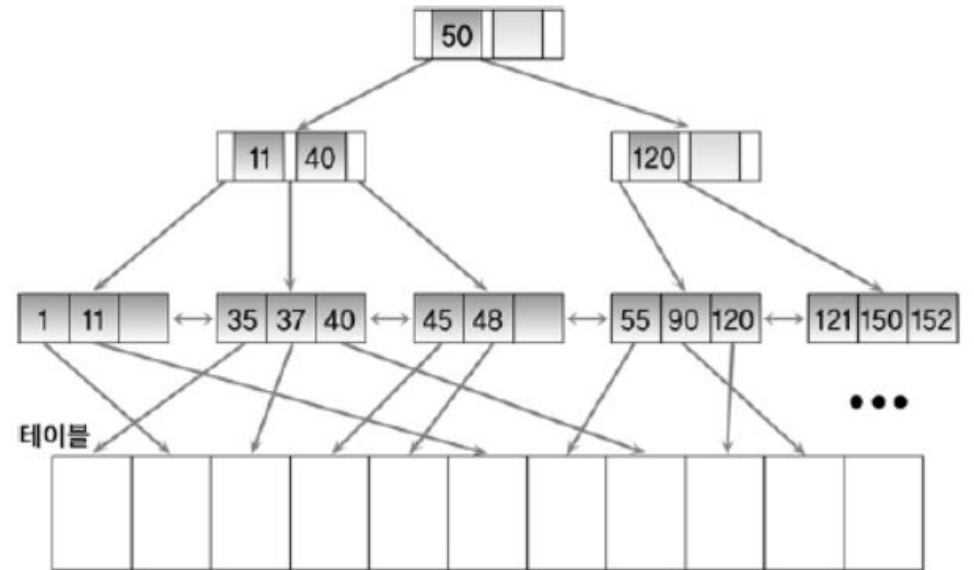
# 인덱스



전체 테이블 스캔



인덱스 스캔



# 비트맵 인덱스

상품테이블

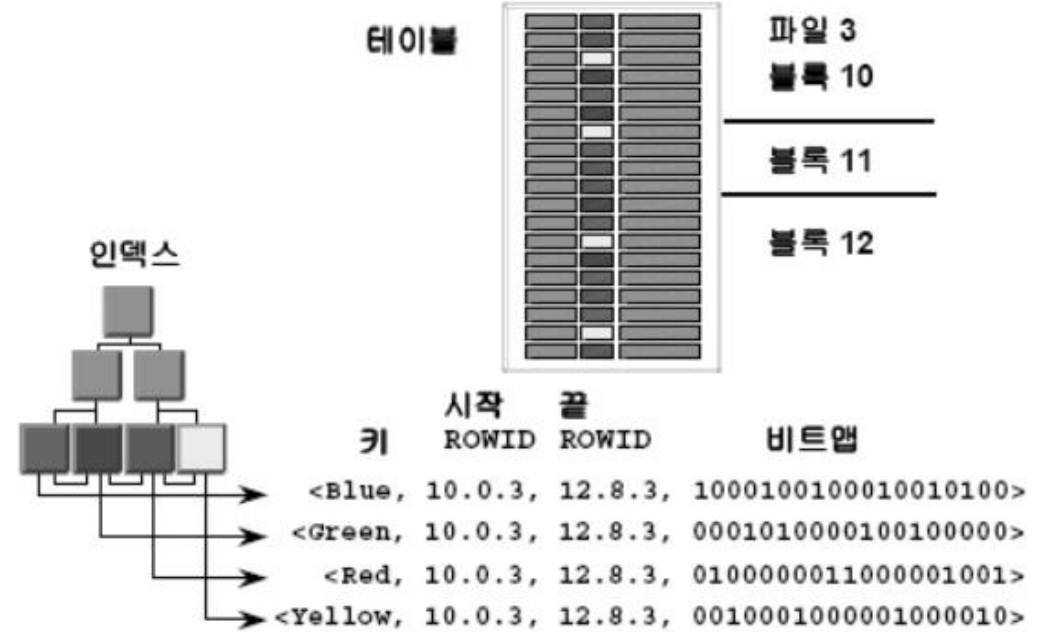
상품 ID	1	2	3	4	5	6	7	8	9	10
상품명	KA387	RX008	SS501	WDGS	CY690	K1EA	PO782	TQ84	UA345	OK455
색 상	GREEN	GREEN	RED	BLUE	RED	GREEN	BLUE		BLUE	RED

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

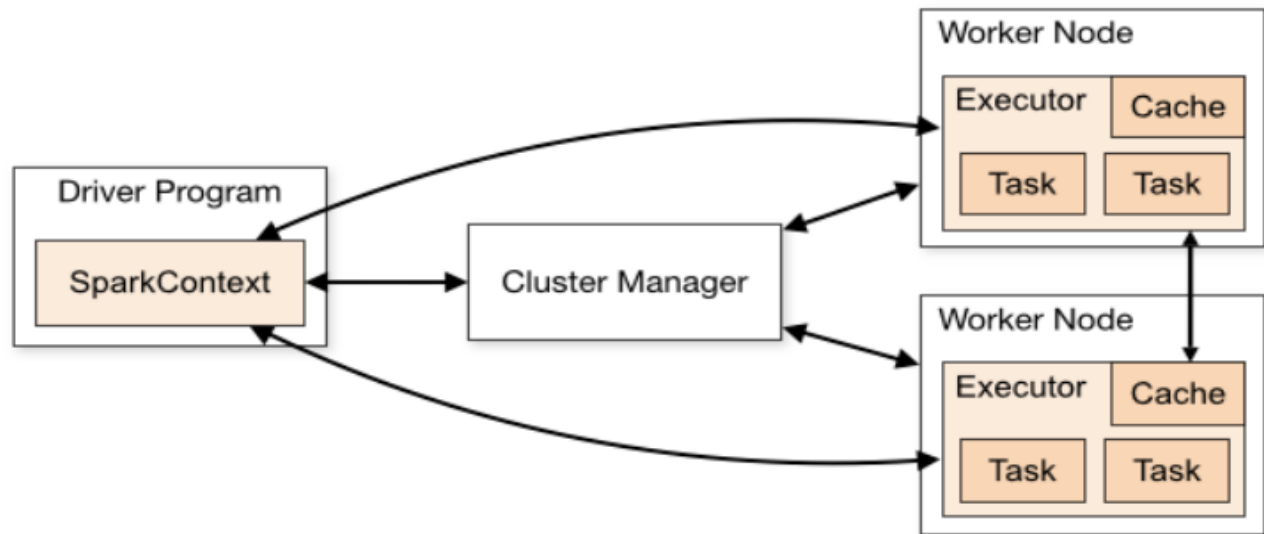
색상	1	2	3	4	5	6	7	8	9	10
BLUE	0	0	0	1	0	0	1	0	1	0
GREEN	1	1	0	0	0	1	0	0	0	0
RED	0	0	1	0	1	0	0	0	0	1
NULL	0	0	0	0	0	0	0	1	0	0

상품\_색상\_BITMAP\_IDX

## 비트맵 인덱스



# 스파크



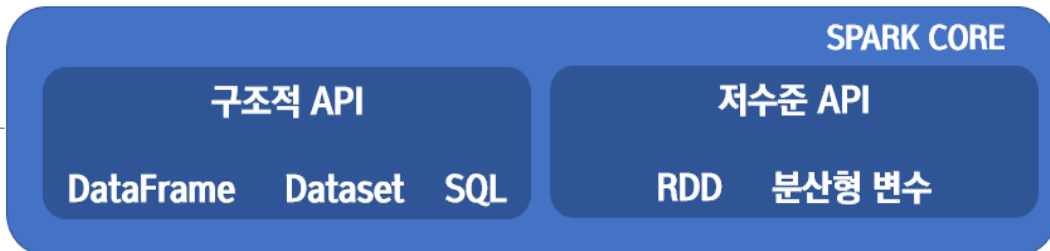
RDD split into 5 partitions



라이브러리



스파크 기본요소



클러스터 매니저

