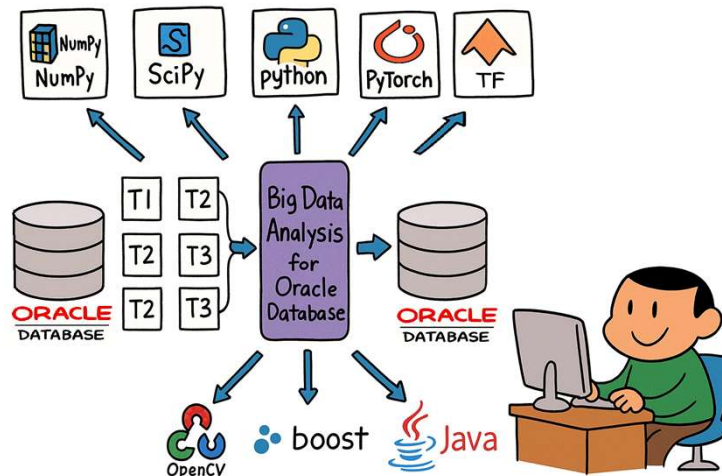


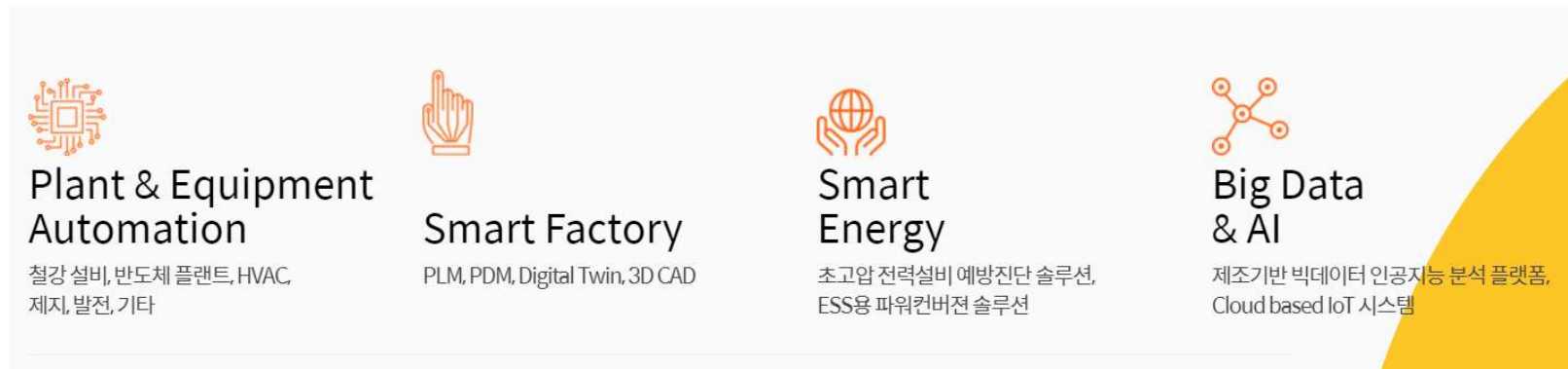
# Big Data Analysis Enabler(BDAE™) 매뉴얼



Raymond,  
2025/05/16

# 1 | 개요

**BDAE<sup>(TM)</sup>** 는 **Oracle Database<sup>(TM)</sup>** 를 사용하는 모든 분야에서 사용 가능 함.  
무정지 설치, 알고리즘이나 로직을 즉시 적용하여 기존 솔루션에도 무중단, 바로 적용 가능.

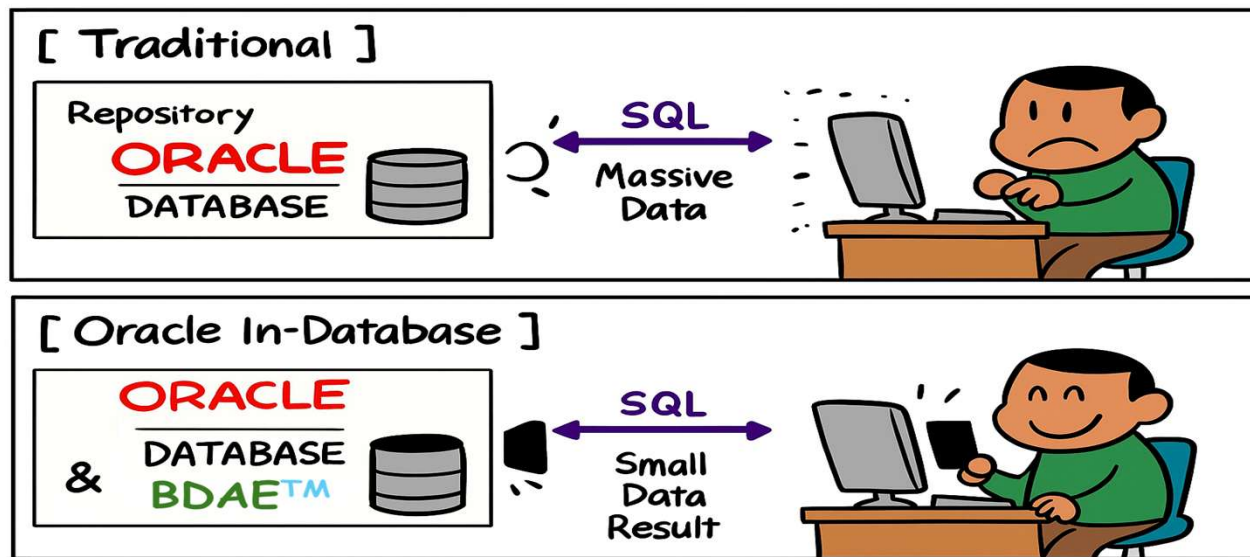


**BDAE<sup>(TM)</sup>** 는 SQL 문으로 실행되고 그 결과도 일반 SQL 문의 결과 처럼 조회되므로 별도의 어플리케이션 서버가 불필요하며, Python 과 R 로 모든 로직(백엔드 포함)을 구현할 수 있게 한다.

제조, 금융, 에너지 분야 등의 MES, SPC, FDC, YMS 및 Smart Factory 전 구성 시스템에 사용 가능 함.

## 2 | BDAE<sup>TM</sup>의 목적

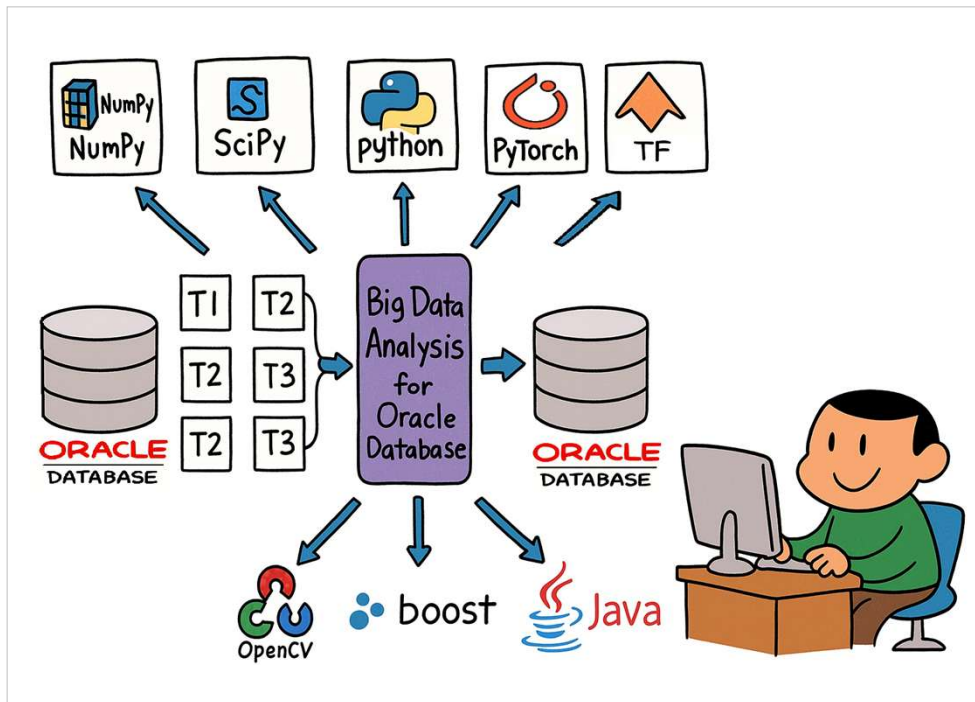
BDAE<sup>TM</sup>는 Oracle In-Database 기술을 바탕으로 만들어졌으며 일반적인 AI 업무에서 Oracle Database<sup>TM</sup>를 단순 저장소 역할로만 사용하는 것을 떠나 학습과 추론 시, 데이터 이동에 대한 Overhead 없이 무정지 운영 환경으로 가져오도록 하는 플랫폼 특징을 가지고 있음.



AI 업무 뿐 아니라, 스마트팩토리의 각종 전처리 가공, 조회 업무에서 일반적인 SQL 문으로 할 수 없는 일들을 Python 과 R 모듈을 통해서 SQL 문의 형태로 별도 서버 없이 Query 만으로 수행할 수 있게 한다는 의미.

BDAE<sup>TM</sup>의 목적은 개발 생산성(Python, R)과 병렬 처리, 성능을 실시간 영역으로 가져온 것.

### 3 | BDAE<sup>(TM)</sup> 의 SW 구성 및 Architecture

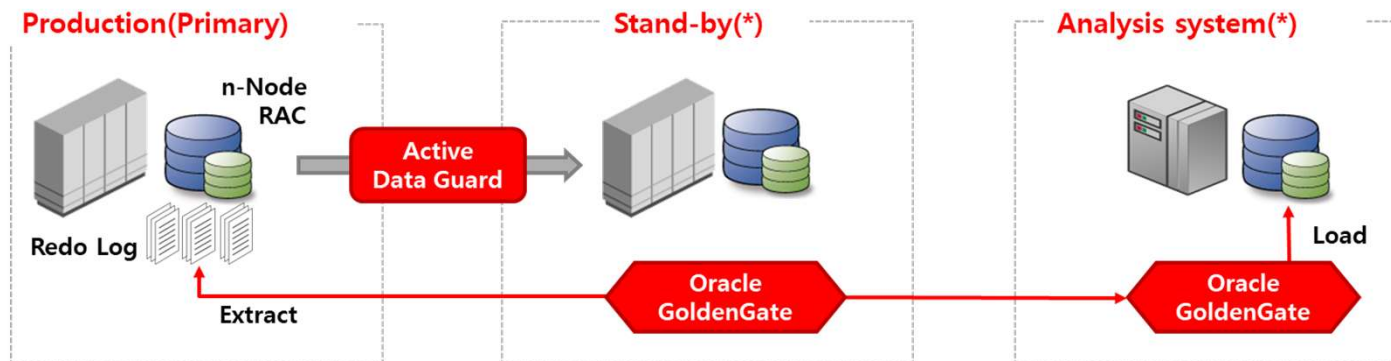


Oracle In-Database 형태에서 BDAE<sup>(TM)</sup> 의 동작 위치를 보여 준다.

병렬 분산 처리는 Oracle In-Database 의 기능이며, 분석가들을 자신의 모듈에서 이를 고려할 필요가 없어, **로직의 재 활용성이 증대**된다.

또한 다양한 분석 엔진과 융합 될 수 있다는 점도 BDAE<sup>(TM)</sup> 의 장점이라고 볼 수 있다.

BDAE<sup>(TM)</sup> 는 Oracle Database<sup>(TM)</sup> 의 어떠한 아키텍처 구성 형태에서도 동작이 가능하다.



## 4 | 클라우드 기반 데모 환경

**BDAE™** 는 아래 `oracle_bdae` 와 `bdae_web` 2개로 구성된다.

`bdae_web` 은 Spring Boot + JSP 로 되어 있고 `oracle_bdae` 는 오라클에서 배포한 이미지에 **BDAE™** 의 설치한 다음, `export` 파일로 배포하면 된다. (USB 저장장치로 전달할 수 있음)

The screenshot shows the Docker Desktop interface. The left sidebar contains navigation options: Containers, Images, Volumes, Builds, Docker Hub, Docker Scout, and Extensions. The main area is titled 'Containers' and displays a table of running containers. Above the table, it shows 'Container CPU usage' as 2.18% / 1600% (16 CPUs available) and 'Container memory usage' as 2.26GB / 6.28GB. A search bar and a toggle for 'Only show running containers' are present. The table lists four containers: `bdae_web`, `oracle-23ai-free`, `oracle_bdae`, and `rabbitmq`. Below the table, there are 'Walkthroughs' for 'Multi-container applications' and 'Containerize your application'. The bottom status bar indicates 'Engine running', 'RAM 4.04 GB', 'CPU 0.06%', and 'Disk: 65.30 GB used (limit 1006.85 GB)'.

Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
bdae_web	6d037339823d	bdae_web:0.50	8080:8080	0%	2 days ago	[Start] [Stop] [Restart] [Logs]
oracle-23ai-free	440f9f4fdd90	database/free:latest	1521:1521 <a href="#">Show all ports (2)</a>	2.18%	1 minute ago	[Start] [Stop] [Restart] [Logs]
oracle_bdae	1e2adb9b4951	oracle_bdae:latest	8521:1521 <a href="#">Show all ports (2)</a>	0%	17 hours ago	[Start] [Stop] [Restart] [Logs]
rabbitmq	-	-	-	0%	22 days ago	[Start] [Stop] [Restart] [Logs]

## 5 | BDAE<sup>TM</sup> 의 빌드

BDAE<sup>TM</sup> 는 Oracle Linux<sup>TM</sup> (OL, 예전에는 OEL : Oracle Enterprise Linux<sup>TM</sup>) 기반에서 빌드 되었음. Oracle Database<sup>TM</sup> 가 지원하는 어떠한 Linux, UNIX 환경에서 BDAE<sup>TM</sup> 는 운영 가능.

현재까지 BDAE<sup>TM</sup> 는 RedHat 계열(Oracle Linux 포함)의 버전 6, 7, 8, 9 에서 모두 빌드가 되어 있으며 한번 빌드되면 Library 파일 형태로 PL/SQL 등과 함께 단 한번 제공 됨.

Oracle In-Database 는 반드시 C with PL/SQL, Function 등으로 작성되어야 하기 때문에 빌드가 필요한 것이며 R, Python 등의 엔진도 모두 C 로 되어 있으며, 운영 체제의 디바이스 드라이버와 흡사한 형태로 보면 됨.

※ BDAE<sup>TM</sup> 를 직접 설치할 일은 거의 없지만 고객의 알고리즘이 Python 버전에 민감할 경우는 직접 설치해 볼 수 있음.

※ Python, R 의 패키지들은 고객이 직접 설치하면 되며 Nexus 기반에서는 Private Network 으로 패키지 위치를 변경 할 수 있음.

## 6 | BDAE<sup>TM</sup>의 사용자 설치(#1)

GitHub 와 블로그에 나름 자세히 적어 두었다.

빌드는 Docker 에서도 가능하기 때문에, 크게 무리가 없지만, 개발 환경은 부족하기 때문에 Oracle Linux 8, Oracle Linux 9 으로 각각 개발 환경을 꾸며 빌드를 한 후 바이너리 형태로 배포 된다.

```
docker pull container-registry.oracle.com/database/free:latest
```

```
명령 프롬프트
C:\AtlasProj\Docke_Work\DockeBuild\Oracle>docker images
REPOSITORY                                TAG                IMAGE ID           CREATED           SIZE
container-registry.oracle.com/database/free  latest            5c5b8d8499c4      2 months ago     13GB
rabbitmq                                    3.7.14-management-alpine  9d10763ec9e3      5 years ago      208MB

C:\AtlasProj\Docke_Work\DockeBuild\Oracle>
```

```
docker run --name oracle-23ai-free -d -p 1521:1521 -p 5500:5500 -e ORACLE_PWD=oracle container-registry.oracle.com/database/free:latest
```









```
명령 프롬프트
C:\AtlasProj\Docke_Work\DockeBuild\Oracle>docker images
REPOSITORY                                TAG                IMAGE ID           CREATED           SIZE
container-registry.oracle.com/database/free  latest            5c5b8d8499c4      2 months ago     13GB
rabbitmq                                    3.7.14-management-alpine  9d10763ec9e3      5 years ago      208MB

C:\AtlasProj\Docke_Work\DockeBuild\Oracle>docker ps -a
CONTAINER ID   IMAGE                                COMMAND              CREATED        STATUS
6b085569a1b6   container-registry.oracle.com/database/free:latest   "/bin/bash -c $ORACL...  16 minutes ago   Up 1
6 minutes (healthy)   0.0.0.0:8521->1521/tcp, 0.0.0.0:8500->5500/tcp
cab71ee7a4cb   rabbitmq:3.7.14-management-alpine   "docker-entrypoint.s...  41 hours ago    Exit
ed (0) 41 hours ago                                rabbitmq_test
```



## 6 | BDAE<sup>(TM)</sup> 의 설치 (#2)

준비되어 있는 것을 *Google Cloud* 에서 다운 받는다.  
( *oel8\_runtime* 폴더명으로 되어 있을 것이다. )

 Anaconda3-2024.10-1-Linux-x86_64.sh	2025-04-25 오후 5:07	SH 원본 파일	1,076,656KB
 extproc.ora	2025-04-27 오전 10:41	ORA 파일	3KB
 libODCI_Python_AnyDataSet.so	2025-04-26 오후 1:56	SO 파일	1,675KB
 libODCI_R_AnyDataSet.so	2025-04-27 오전 10:11	SO 파일	2,444KB
 ora19c.dmp	2025-05-08 오전 9:33	DMP 파일	1,355,536KB
 Readme.txt	2025-04-27 오후 12:11	텍스트 문서	3KB
 RInside_0.2.15.tar.gz	2025-04-23 오후 3:03	압축된 보관 폴더	78KB
 which	2025-04-26 오후 2:44	파일	30KB

이것을 *Docker* 명령을 이용해서 복사 한다. 폴더 전체를 가져가는 것이 낫다.

```
docker cp oel8_runtime oracle_bdae:/home/oracle
```

※ 이하 설치는 *GitHub* 참조 하기 바람 (*GitHub* 는 메일로 공지 했음, 한달만 유효 예정)

※ 데모 시스템 역시 메일로 공지 (*Google Cloud* 파일 URL 공지 함, 한달만 유효 예정)



## 7 | BDAE<sup>TM</sup>의 사용에 앞서 ..

\* 상용 제품인 (주)오라클의 **Oracle R Enterprise<sup>TM</sup>** 은 R 언어만을 지원한다.  
물론 훌륭한 분석가들이 주요 알고리즘을 R 패키지로 내장하였지만, 현실은 많은 코드를 짜야 한다.  
그리고 짠 코드는 DB 테이블로 저장 되어야 하며 GUI 를 제공하지 않는다.

그런데, 어떠한 GUI 도 제공되지 않기 때문에 많은 양의 R 스크립트 코드를 해당 DB 테이블에 넣는 것이 쉽지 않고 디버깅은 사실 한숨이 좀 나온다. 이 부분은 R 의 한계이기도 함.

\* 반면, **BDAE<sup>TM</sup>** 는 Python, R 및 ETL (주로 Hadoop 등)을 목적으로 한 JAVA 를 지원하며, 간단한 Web 화면을 가지고 있고, 이 Web의 Editor 를 통해서 고객의 알고리즘을 DB 로 입력을 하게 된다.

통상 개발은 R의 경우 R studio 나 Python 의 경우 Jupyter notebook 등에서 하는 경우가 많고, 여기에서 개발된 것을 **BDAE<sup>TM</sup>** Web 을 통해서 Copy & Paste 해서 입력하면 된다.

※ **BDAE<sup>TM</sup>** Web 은 번들이며, 개선점이 많지만 이러한 목적으로 사용되기엔 크게 부족함이 없다. (**Apache NIFI<sup>TM</sup>** 와 함께 사용하면 배치작업 등에도 효과적이다.)

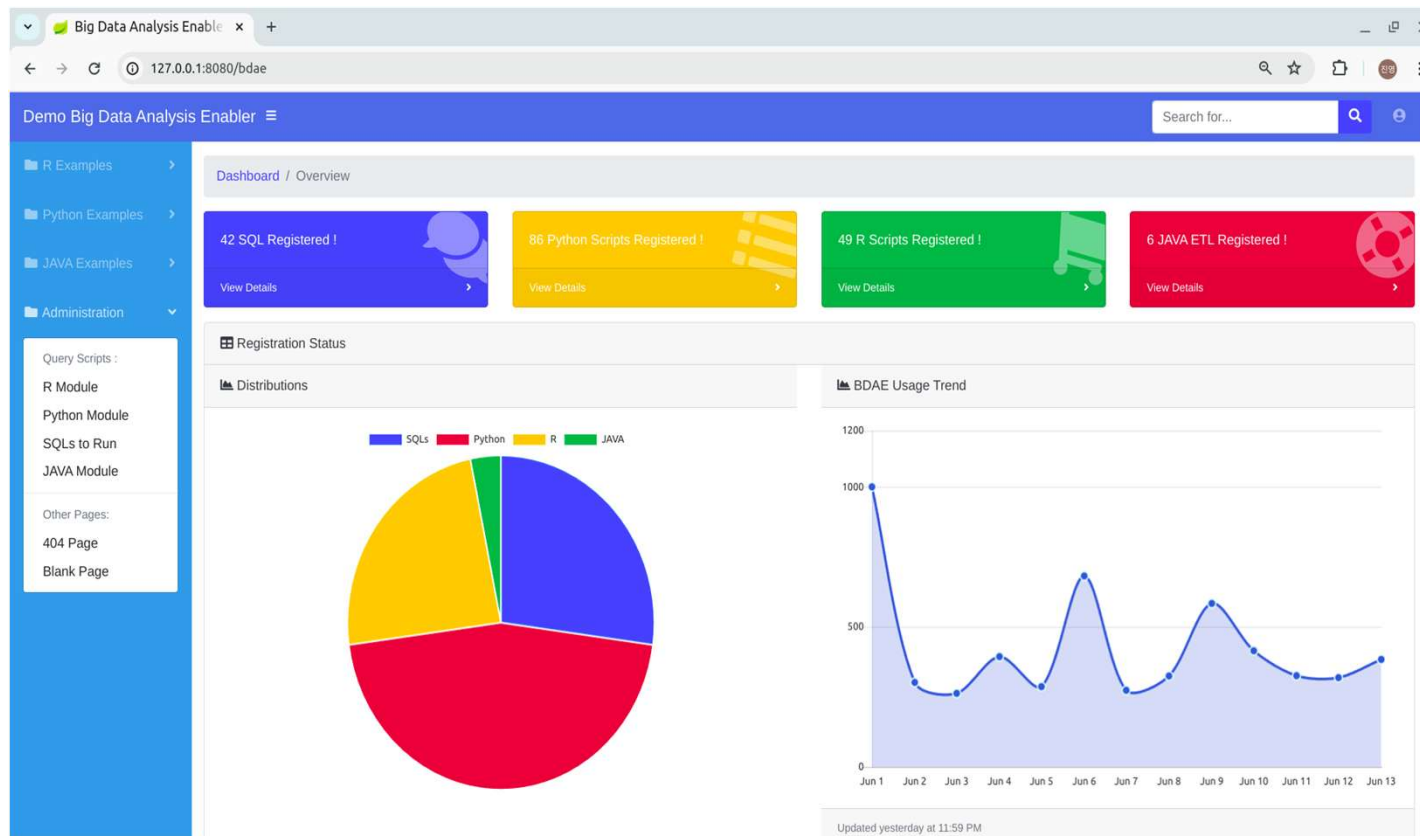
※ Jupyter notebook 에서도 R 은 실행 시킬 수 있으며 **BDAE<sup>TM</sup>** 는 통합 되었다.

※ 단일 Windows 에서의 R 패키지 설치에 GitHub 의 Troubleshooting 부분을 참조하기 바람.

## 8 | BDAE<sup>TM</sup> Web구성

주로 좌측의 Administration 메뉴를 통해서 R Module, Python Module 을 등록, 조회, 삭제 하고 데이터와 호출을 위한 SQL 을 등록한 뒤 즉시 실행하면서 결과를 확인 해 볼 수 있다.

고객의 자산인 Python, R 의 알고리즘 코드를 DB 에 저장하는 장점은 언제든지 무정지 패치가 가능하고 DB 에서 버전 관리 할 수 있다는데 있다.



## 9 | Oracle R Enterprise™의 형태

아래는 Oracle R Enterprise™에서 제공되는 Embedded Script Execution 형태이다.  
맨 아래 2개 (sys.rqScriptCreate, rqScriptDrop)은 모듈 입력, 삭제에 대한 것이며 좀 불편하다.  
BDAE™의 경우 Web Editor에서 생성, 수정, 삭제가 이뤄지기 때문에 편리하다.

- ※ Oracle R Enterprise™는 R 코드의 버전 관리 기능이 없다.  
반면 BDAE™는 버전 관리를 할 수 있는 스키마 구성이 되어 있으며 커스터마이징 가능하다.
- ※ rqScriptCreate을 보면 INSERT INTO ... 인데 Python은 들여쓰기가 매우 중요하며 Python, R 모두 " ", ' ' 등을 사용하기 때문에 SQL INSERT 구분을 작성하려면 더 많은 시간이 소모함.  
특히 Python의 경우 "''", '"', ' '를 마구 사용하기 때문에, INSERT 구문으로는 거의 불가능함.

### Embedded Script Execution – SQL Interface

SQL Interface function	Purpose
rqEval()	Invoke stand-alone R script
rqTableEval()	Invoke R script with full table as input
rqRowEval()	Invoke R script on one row at a time, or multiple rows in chunks
"rqGroupEval()"	Invoke R script on data partitioned by grouping column
sys.rqScriptCreate	Create named R script
sys.rqScriptDrop	Drop named R script

## 9 | BDAE<sup>TM</sup> SQL 문의 형태와 R, Python 모듈

BDAE<sup>TM</sup> 는 R, Python 이 모두 되므로 Oracle R Enterprise<sup>TM</sup> 의 2배, 총 8개의 함수가 제공된다. 그 Argument 는 동일하며 Concept 및 Oracle In-Database 모두 동일한 아키텍처를 따른다.

SQL Interface function	Description
asEval()	Same rqEval()
asTableEval()	Same rqTableEval()
asRowEval()	Same rqRowEval()
asGroupEval()	Same rqGroupEval()
apEval()	Invoke stand-alone Python module
apTableEval()	Invoke Python module with full table as input
apRowEval()	Invoke Python module on one row at a time, or multiple rows in chunks
apGroupEval()	Invoke Python module on data partitioned by grouping column

## 10 | BDAE<sup>TM</sup> Web 을 이용한 등록 및 실행

BDAE<sup>TM</sup> 는 아래의 3 Step 으로 등록, 실행할 수 있다.

1. R 또는 Python 모듈 등록 (재활용 가능)
2. DB 데이터와 등록되어 있는 R 또는 Python 모듈과 융합을 위한 앞장의 SQL 문의 작성
3. SQL 문의 실행 및 결과 보기

이것이 전부이다. 실시간에서 즉시 패치하고 기존 버전의 동작 중인 것에 영향을 주지 않는다.

Oracle R Enterprise<sup>TM</sup> 나 BDAE<sup>TM</sup> 모두 분석 및 출력 하고자 하는 데이터는 Oracle Database<sup>TM</sup> 내에 존재하기 때문에 모든 것은 SQL 문으로 기술되어야 한다.

즉, 입력과 출력은 모두 Oracle Database<sup>TM</sup> 가 이해할 수 있는 테이블 형태의 데이터 포맷으로 구성되지만, 그 내부의 Temporary 부분은 마음껏 다양한 포맷과 코드로 구성해도 무방하다는 뜻이다.

※ 통상적으로 대용량 분석이나 하둡(Hadoop) 또는 파일 기반의 분석 작업 후 결과를 RDBMS 에 넣는 작업을 하지만 BDAE<sup>TM</sup> 는 자동으로 이러한 작업들이 수행될 수 있다.

※ Oracle Optimizer 는 SQL 실행 시 입력과 출력에 대해서 통계 정보 및 Data Dictionary 를 이용하여 확인 후 Plan 을 잡기 때문에, 명시적으로 입.출력에 대해서 반드시 알려 주어야 한다.

Oracle R Enterprise<sup>TM</sup> 나 BDAE<sup>TM</sup> SQL 형태에서 입.출력을 명시한 이유가 바로 이것이다.

## 11 | R 코딩 스타일(#1)

일반적으로 R 코딩은 대략 2가지이며 아래와 같은 함수 형태가 있다.

이때 **BDAE<sup>TM</sup>** 는 자동으로 함수 argument 인 `data`, `args` 이름을 찾아서 그 데이터를 넣어서 호출해 준다. 따라서 `data`, `args` 변수명은 분석가가 임의로 지정해도 무방하다.

```
function(data, args) {  
  # 각종 의존성 라이브러리의 등록  
  library(xts)  
  library(quantmod)  
  library(RCurl)  
  library(logr)  
  library(xts)  
  library(quantmod)  
  library(RCurl)  
  
  ...  
  ... < R 이 지원하는 어떠한 데이터 타입, 모델, 알고리즘 코드를 수용할 수 있음 > ...  
  ...  
  
  df <- data.frame(col1=c(...), col2=c(...), ... ,stringsAsFactors=FALSE)  
  return (df)  
}
```

## 10 | R 코딩 스타일(#2) - BDAE<sup>TM</sup> 만의 특징

또 다른 R 코딩 스타일은 다음과 같은 서술형이다.

이때는 BDAE<sup>TM</sup> 에서 **data, args** 라는 **명시적인 변수명에 데이터를 넣어서 호출**하기 때문에, 반드시 **data, args** 라는 두가지 변수에 각각 입력 데이터, 부가적인 데이터가 들어간다고 보면 된다.

**이 두가지 변수명은 Read-Only 이며 다른 목적으로 사용되어서는 안된다.**

맨 마지막에 리턴이 없기 때문에 **df** 라는 것을 적어주어야 하며, 이 변수명(df)은 임의로 해도 된다.

( ※ BDAE<sup>TM</sup> 의 RSCRIPT 테이블의 컬럼 중 STYLE\_TYPE=Normal 이 아니어야 한다. )

```
library(xts)
library(quantmod)
library(RCurl)
library(logr)
library(xts)
library(quantmod)
library(RCurl)
```

```
x <- data # 입력되는 처리하려는 원시 데이터가 들어 있는 변수는 data 임
```

```
y <- args # 부가적인 데이터가 들어 있는 데이터는 args 임
```

```
...
```

```
... < R 이 지원하는 어떠한 데이터 타입, 모델, 알고리즘 코드를 수용할 수 있음 > ...
```

```
...
```

```
df <- data.frame(col1=c(...), col2=c(...), ... ,stringsAsFactors=FALSE)
```

```
df
```



## 10 | R 코딩 스타일(#3)

Syntax 오류 체크와 Runtime 오류

분석가는 **BDAE<sup>(TM)</sup>** 모듈에서 예외 처리를 하지 않아도 된다. **BDAE<sup>(TM)</sup>** 에서 문제가 되면 SQL 의 오류코드와 R 오류 메시지로 반환된다.

**BDAE<sup>(TM)</sup>** 는 R 의 경우 먼저 Syntax 체크를 수행한 후 문제가 없는 경우에만 데이터를 가공해서 해당 모듈을 호출한다. 문제가 되면 분석가는 이를 해결한 후 다시 실행하도록 한다.

※ R 은 Python 에 비해서 예외 처리가 빈약한 편이다. C 언어로 되어 있는 R 의 API 가 한계가 있다.

## 11 | Python 코드(#1)

Python 은 R 과 다르다. Python 은 모듈 이름 (통상 파일 명)과 많은 함수들 중 시작 함수 명이 있어야 호출 가능하다. ( \_\_main\_\_ 을 호출하는 일반적인 방식이 아니다. )

```
import pandas as pd
import numpy as np
import seaborn as sns
from StreamToLogger import StreamToLogger
import sys
import logging, logging.handlers

def describe(df):
    logger = logging.getLogger('TitanicDesc:describe')
    logger.setLevel(logging.DEBUG)
    socketHandler = logging.handlers.SocketHandler('localhost',
        logging.handlers.DEFAULT_TCP_LOGGING_PORT)
    logger.addHandler(socketHandler)

    sys.stdout = StreamToLogger(logger, logging.INFO)
    sys.stderr = StreamToLogger(logger, logging.ERROR)

    print('----- start -----')
    columns_ = df.columns.tolist()
    print(str(columns_))
    df.columns = list(map(str.lower, columns_))
    df_desc = df.describe()
    print(str(df_desc))
    df_desc.reset_index(inplace=True)
    df_desc.columns = ['vars', 'passengerid', 'survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']
    df_melt = pd.melt(df_desc, id_vars=['vars'])
    print('----- end -----')

    socketHandler.close()
    logger.removeHandler(socketHandler)

    return df_melt
```

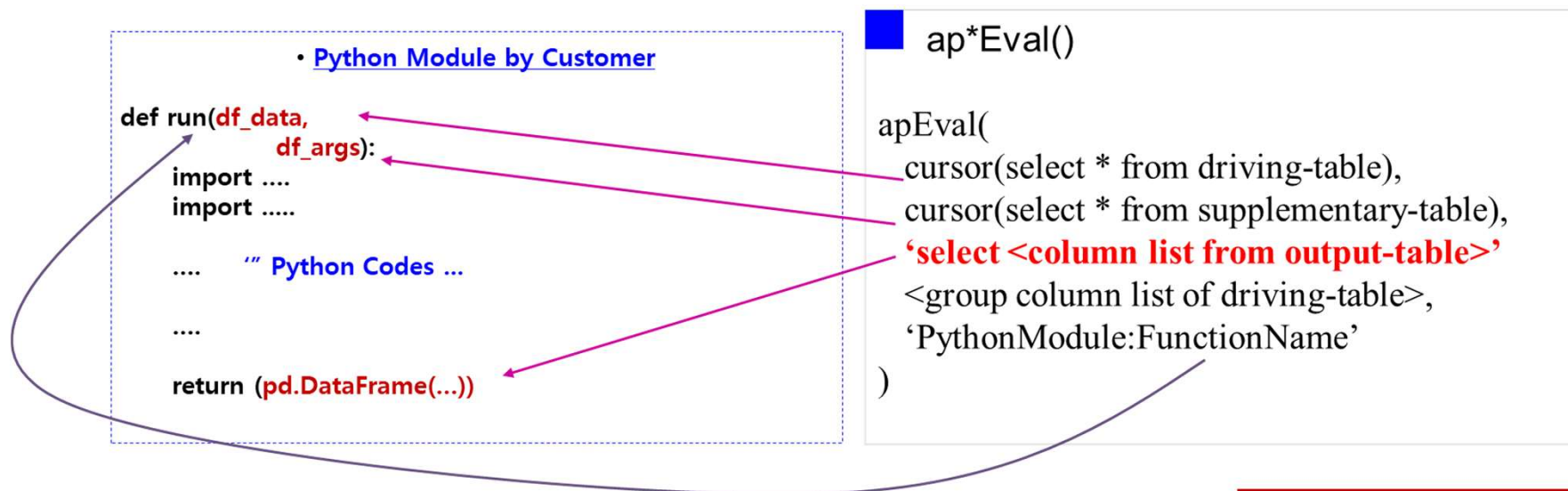
## 11 | Python 코드(#2)

Python 모듈의 시작 함수의 argument 는 R 과 동일하게 2개이다.

앞의 것이 입력 데이터, 뒤의 것이 부가적인 데이터이다. 이름은 마음대로 정하면 된다.

R 과 달리, 서술형태는 지원하지 않는다.

- ※ 병렬 처리의 경우는 함수 명(ap(s)GroupEval)이 다르고, 병렬처리 컬럼명들을 반드시 제공해야 한다.
- ※ Oracle Database<sup>TM</sup> 의 SQL 실행 메커니즘을 이해할 필요가 있다. Oracle Optimizer 는 입력과 출력 형태를 보관 중인 통계 정보를 기반으로 Plan 을 만든다. 최종 Python 출력을 Oracle 이 자동으로 인지하지 못하므로 이를 알려 주어야 한다. (우측 빨간 색)
- ※ 출력이 복잡한 경우 Oracle View 로 만들고 'VIEW\_NAME' 만 적어 주면 된다. BDAE<sup>TM</sup> 기능
- ※ Python, R 출력을 SELECT 구문으로 자동으로 만드는 부분은 GitHub 에 있고, 이를 View 로 만들 것



## 12 | BDAE<sup>TM</sup> Web Editor

BDAE<sup>TM</sup> Web Editor 부분은 오픈소스 패키지를 импорт 해서 사용했다. 여기에서 개발하라는 뜻은 아니며, 기존 개발 툴 (R studio, Jupyter notebook 등)에서 개발한 것을 복사해서 넣으면 된다.

- ※ **Syntax** 오류 시에 라인 번호가 있는 경우 이것을 활용하면 좌측의 라인번호가 바로 그것이다. BDAE<sup>TM</sup> 가 정의한 **ORA-20000** 대의 **Oracle** 오류 메시지 안에 포함되어 있다.

```
R Modification / For enhancing your model.

R Module Name : kspi_Demo_img_3rd Script Type : Normal Description :

function(data, args1) {
2
3   library(xts)
4   library(quantmod)
5   library(RCurl)
6   library(logr)
7   library(xts)
8   library(quantmod)
9   library(RCurl)
10
11   sma1 = args1$SMA1
12   sma2 = args1$SMA2
13
14   data2 <- data.frame(data$open, data$high, data$low, data$close, data$volume, data$adjusted, row.names=data$row.names)
15   s1 <- as.xts(data2)
16   png(tfl <- tempfile(fileext = ".png"), width=1920, height=1080)
17   taS <- sprintf("addMACD();addBBands();addSMA(%d);addSMA(%d,col='blue')", sma1[[1]],sma2[[1]])
18   chartSeries(s1['2016-08-10:'], up.col='red', dn.col='blue', theme='white', name="Samsung", TA=taS)
```

- ※ R 은 Python 에 비하면 예외처리 등이나 디버깅이 매우 불리하다. Python 은 실행 시의 오류에 대해서 **traceback()** 이 매우 정확하며 **API** 가 잘 정리되어 있다.

## 13 | BDAE<sup>TM</sup> Python

Python 의 경우 Module Name 과 Function Name (시작 함수) 두가지가 반드시 존재해야 하며, BDAE<sup>TM</sup> 는 DB 에 있는 경우와 File 구성 두가지를 모두 실행해 주되, DB 가 우선적으로 실행된다.

즉, DB 에 없다면 PYTHONPATH 위치에서 Module 을 찾는다는 의미이다. (설정은 고객이 함)

※ 파일로 되어 있는 Python 모듈의 문제점은 버전 관리가 취약하다는 점이며, Oracle RAC 환경에서는 모든 Node 에 해당 파일을 설치해 주어야 한다는 점이다.

Python Script Modification / You can modify your model ...

Python Module Name :  Python Function Name :

Description

```
1 from ultralytics import YOLO
2 import pandas as pd
3 import numpy as np
4 import os
5 os.environ['KMP_DUPLICATE_LIB_OK']='TRUE'
6 os.environ['PYTHONIOENCODING']='UTF-8'
7 import logging, logging.handlers
8 import sys
9 import base64
10 from os import listdir
11 from os.path import isfile, join
12 from StreamToLogger import StreamToLogger
13 import logging, logging.handlers
14
15 file_loc = []
16
17 def yolo_callback(x):
18     print("yolo callback called !")
19     file_loc.append(str(x))
```

## 14 | Python DataFrame, R data.frame

모든 RDBMS 의 테이블은 2D Tensor 이기 때문에 이것에서 벗어나질 못한다.  
따라서 입력 값과 출력 값은 모두 Python Pandas DataFrame 과 R 의 data.frame 포맷으로 작성해 주어야 한다. (R 의 경우 data.frame, data.table 2가지 출력 모두를 지원한다. )

입력은 BDAE(TM) 가 자동으로 해 주지만, 출력은 분석가가 리턴할 때 만들어서 리턴해 주어야 한다.  
입력 시에 대 소문자가 문제인데, 아래 처럼 하면 대소문자가 섞인 컬럼도 가능하다.

SELECT 의 "col1", .. 로 되어 있다면 대소문자가 섞인 형태도 그대로 BDAE(TM) 가 넘겨줄 것이다.

이것은 분석가들이 알아야 한다. Pandas DataFrame 컬럼을 이용할 경우 주의를 기울여야 한다.  
차라리 컬럼들을 모두 대문자, 또는 소문자로 바꾼다음에 작업해도 될 것이다.

SELECT "Passenger" as Passenger 라고 하면 대문자로 나올 것이다.  
이건 BDAE(TM) 가 하는 것이 아니라 Data Dictionary API 때문이니 알고 넘어가야 한다.

출력의 경우도 "" 로 대소문자를 섞을 경우에는 그대로 나오지만, 아닌 경우는 대문자로 나온다.

- ※ 이 부분은 BDAE(TM) 때문이 아니라 Oracle Database(TM) SQL 엔진 특성이다.
- ※ SELECT \* FROM .. 으로 하면 테이블 생성 당시의 컬럼 대.소문자가 그대로 적용 된다.

## 15 | NA, NaN, Inf (#1)

분석에서는 NA (Not Available), NaN (NULL), Inf (Infinity) 가 중요하다.  
그러나, **Oracle Database™**에서는 NA, NaN 는 동일하게 값이 NULL 이며,  
Inf 는 다음과 같이 제공된다. 아래 R 의 Infinity 는 1.0/0.0 은 양의 Infinity, -1.0/0.0 은 음의 Infinity.

### 1. R 의 경우

```
X <- c(1,2,3,NA,5)
Y <- c(1.1,-1.0/0,1.0/0,4.0,5.34)
```

```
df <- data.frame(X,Y)
df
```

### 2. Python 의 경우

```
import pandas as pd
import numpy as np
```

```
def returnNaN():
    df = pd.DataFrame([[ 'motor type',1, np.inf],
                        [np.nan, 2, 3.2],
                        ['RF', np.nan, 4.5]],
                        columns = list('abc'))

    return df
```



## 15 | NA, NaN, Inf (#2)

Oracle에서는 Infinity를 `binary_double_infinity`로 리턴하지만, JAVA에서 이 SQL을 단순 호출하면 Overflow가 발생하니 주의해야 한다. (이 부분은 **Oracle Database<sup>TM</sup>**나 **BDAE<sup>TM</sup>** 문제가 아님.)

```
SELECT A, case when B=-binary_double_infinity
               then '-Infinity'
               when B=binary_double_infinity
               then '+Infinity'
               else TO_CHAR(B) end AS B
FROM (
  SELECT *
  FROM
  table(asEval(
    NULL,
    'SELECT 1 as A, 1.0 as B FROM dual',
    'R_infinity'))
);
```

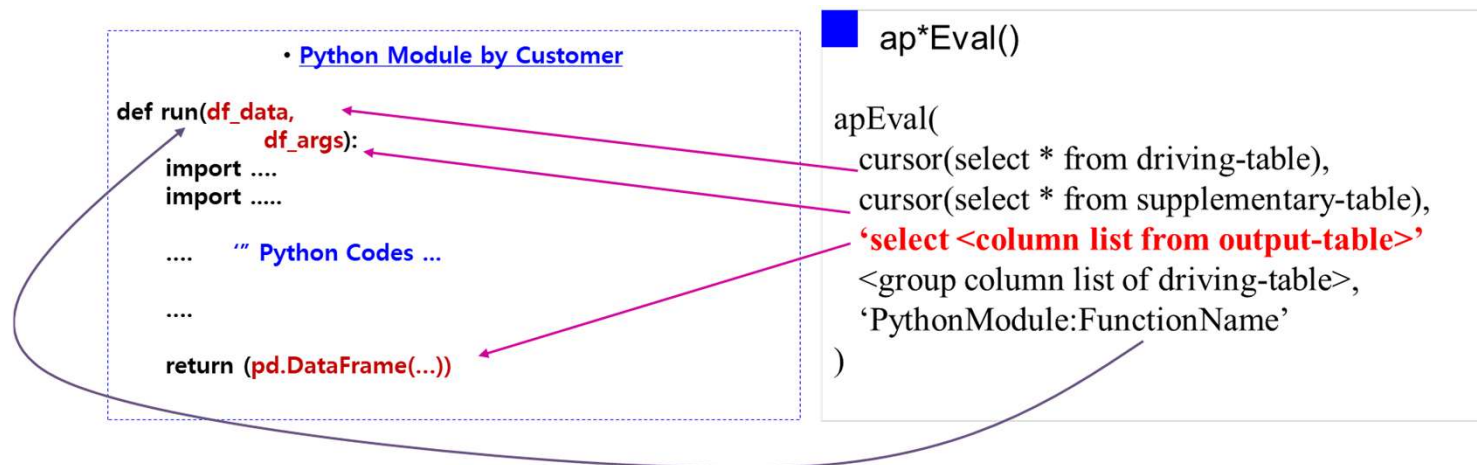
## 16 | 리턴 DataFrame과 Oracle 매칭

우측의 빨간 색 'select <column list from output-table>' 부분은 보통 다음과 같이 dual 을 사용한다. 그런데, 아래와 같이 이 컬럼이 많을 경우에는 짜증스럽다. 따라서 이를 View 로 만들어 사용하면 편리하다. 'V\_OUTPUT1' 으로 입력하면 된다. ( ※ 이 부분은 BDAE<sup>(TM)</sup> 의 기능이다. )

```
'SELECT CAST(''AA'' AS VARCHAR2(40)) EQP,  
      CAST(''AA'' AS VARCHAR2(40)) UNIT,  
      CAST(''AA'' AS VARCHAR2(40)) LOT,  
      CAST(''AA'' AS VARCHAR2(40)) WAFER,  
      CAST(''AA'' AS VARCHAR2(40)) RECIPE,  
      CAST(''AA'' AS VARCHAR2(40)) PARAM,  
      CAST(''AA'' AS VARCHAR2(100)) LOCATION  
FROM DUAL'
```



```
CREATE VIEW V_OUTPUT1 AS  
SELECT CAST(''AA'' AS VARCHAR2(40)) EQP,  
      CAST(''AA'' AS VARCHAR2(40)) UNIT,  
      CAST(''AA'' AS VARCHAR2(40)) LOT,  
      CAST(''AA'' AS VARCHAR2(40)) WAFER,  
      CAST(''AA'' AS VARCHAR2(40)) RECIPE,  
      CAST(''AA'' AS VARCHAR2(40)) PARAM,  
      CAST(''AA'' AS VARCHAR2(100)) LOCATION  
FROM DUAL;
```



## 17 | 개발 시, 실시간 로깅

시스템 오류가 나면 Python, R 모듈 안의 로그는 당연히 볼 수 없다.

(※ **BDAE<sup>(TM)</sup>** 는 Python, R 개발환경 처럼 한 줄씩 실행되는 것이 아니기 때문이다. )

그리고, 파일로 매번 하는 것 보다는 개발 시 원격 로깅을 사용하는 것이 좋다.  
GitHub 에 넣어 두었다.

**Python 의 경우 LogServer.py 파일을 실행 시키면 된다.**

## 18 | Python 모듈 개발 할 때 (Input 에 대한 부분 확인)

**BDAE<sup>(TM)</sup> 없이 개발 환경에서는 Python sqlalchemy 패키지를** 사용할 가능성이 있다.

pip install sqlalchemy, cx\_Oracle 만 설치한다고 동작하지 않는다.

Oracle Client 가 있어야 하는데, 이것은 Naver Cloud 에 올려 두었다. (설치 파일 폴더)

사용법은 아래와 같다. 이것으로 컬럼을 확인하면서 모듈 개발을 하면 된다.

```
import sqlalchemyimport pandas as pd
import pandas as pdimport cx_Oracle
import osfrom sqlalchemy
import create_engine
LOCATION = r"C:\Users\Admin\Downloads\instantclient-basic-windows.x64-23.8.0.25.04\instantclient_23_8"
os.environ["PATH"] = LOCATION + ";" + os.environ["PATH"]

oracle_connection_string = 'oracle+cx_oracle://{username}:{password}@{hostname}:{port}'
DATABASE = "FREE"
SCHEMA = "rquser"
PASSWORD = "0000"
engine = create_engine( oracle_connection_string.format(username=SCHEMA,
    password= PASSWORD, hostname='177.175.54.97', port='1521', database='FREE', ))

conn = engine.connect()
SQL = \
"""
SELECT ...
"""
df = pd.read_sql_query(SQL, conn)
```

## 18 | R 모듈 개발 할 때 (Input 에 대한 부분 확인)

**BDAE<sup>(TM)</sup>** 없이 개발 환경에서는 **ROracle** 패키지를 사용할 가능성이 있다.

ROracle 설치 는 Github 나 인터넷을 활용하고, 다만 Windows 에서 설치 는 GitHub 에 넣어두었음.

Oracle Client 가 있어야 하는데, 이것은 Cloud 에 올려 두었다. (설치 파일 폴더)

사용법은 아래와 같다. 이것으로 **BDAE<sup>(TM)</sup>** 입력을 예상해서 개발 한다.

```
# 아래는 Oracle 설치 위치 임. PATH 잡혀 있을 경우 무시 가능

Sys.setenv("ORACLE_HOME"="/u01/app/oracle/product/12.2.0.1/db_1")
library(DBI)
library(ROracle)

driv <- dbDriver("Oracle")

connect.string <- paste(
  "(DESCRIPTION=",
  "(ADDRESS=(PROTOCOL = TCP)(HOST = 177.175.54.97)(PORT = 1521))",
  "(CONNECT_DATA=(SERVER = DEDICATED))",
  "(SERVICE_NAME = FREE)))", sep = "")

conn <- dbConnect(driv, username="rquser", password="0000", dbname=connect.string)
df <- dbGetQuery(conn,"SELECT * FROM FDC_TRACE WHERE ROWNUM < 10")
```

## 19 | 모듈 개발 할 때 출력 부분 확인(#1)

결국 **BDAE<sup>TM</sup>**의 `SELECT ... FROM DUAL` 부분을 매번 수작업 하기 어렵기 때문에 유틸리티를 만들었다.

앞의 `Output` 부분인데, 아래를 사용하면 만들어 주고, 그것을 다시 `View` 로 만들면 편리하다.

```
def dtype_to_dbtype(typestr):
    return {
        'int64': lambda: '1',
        'object': lambda: "CAST('AA' AS VARCHAR2(40))",
        'float32': lambda: '1.0',
        'float64': lambda: '1.0',
        'datetime64[ns]': lambda: 'TO_TIMESTAMP(NULL)',
        'byte': lambda: 'TO_BLOB(NULL)'
    }.get(typestr, lambda: typestr + "not defined type.")()

def space_fill_underbar(column_name):
    return ' '.join(column_name.split(' '))
```

## 19 | 모듈 개발 할 때 출력 부분 확인(#2)

아래 `get_bdae_output_format()` 을 호출하면 "SELECT ... FROM dual" 문을 만들어준다.

```
def get_bdae_output_format(df):
    types = df.dtypes
    column_name_list = []
    column_type_list = []
    for i in range(len(types.index.tolist())):
        column_name_list.append(types.index.tolist()[i])
        column_type_list.append(str(types[types.index.tolist()[i]]))
        print("%-40s %s" %(types.index.tolist()[i], str(types[types.index.tolist()[i]])))

    sql = 'SELECT '
    last_index = len(column_name_list) - 1
    for i in range(last_index + 1):
        if i == last_index:
            sql = sql + ' ' + dtype_to_dbtype(column_type_list[i]) + ' ' + space_fill_underbar(column_name_list[i]) + '\nFROM dual'
        else:
            sql = sql + ' ' + dtype_to_dbtype(column_type_list[i]) + ' ' + space_fill_underbar(column_name_list[i]) + ',\n'

    return (sql)

sql = get_bdae_output_format(df)
print(sql)
```



## 20 | SQL 문

\* Dynamic SQL 문을 사용하지 않는다면 이는 고객사의 스키마에 의존성을 갖게 되는 것이며 이는 매번 고객사에 맞도록 코드를 변경해야 한다는 뜻이다.

그러나, **BDAE<sup>(TM)</sup>** 는 Dynamic SQL 이 가능하도록 구현되었고, 이는 차별화된 핵심적인 기능이다. **Oracle R Enterprise<sup>(TM)</sup>** 와 같은 수준으로 **BDAE<sup>(TM)</sup>** 도 가능하도록 구현 됨.

( ※ Database Dictionary 를 완벽하게 지원하는 RDBMS 는 오직 **Oracle Database<sup>(TM)</sup>** 뿐이다.  
Dynamic SQL 의 핵심은 SubQuery 의 SubQuery .. 가 되어도 최상위 컬럼 정보의 Dictionary 를 투명하게 지원하지 않으면 안되기 때문이다. )

\* 병렬 처리 부분

1. Oracle Optimizer 는 입력 Query 와 출력 Query 및 통계정보를 확인하고 병렬 처리를 결정한다.
2. 따라서, 이미지(BLOB) 이나 4000 바이트 이상의 문자열 (CLOB) 이 포함된 것을 확인 하면 병렬 처리를 **Oracle Database<sup>(TM)</sup>** 는 포기한다.
3. **BDAE<sup>(TM)</sup>** 를 이용할 때도 마찬가지로 적용 된다. 단, 이것을 회피하고 병렬 처리 가능하도록 공수를 부릴 수 있는 방안도 얼마든지 있지만, 일반 SQL 문이 아닌 **BDAE<sup>(TM)</sup>** 를 써야 가능하다.
4. **BDAE<sup>(TM)</sup>** 의 Python 모듈을 한번 사용하면 PL/SQL 을 잘 사용하지 않게 된다. 개발 생산성 때문.

## 21 | 분석가들이 준비하는 일

**BDAE<sup>(TM)</sup>** SQL 을 만들기 전에 분석가들은 다음과 같이 하면 된다.

병렬 처리를 고려하지 말고, 단일 모듈을 만들면 된다.

`import sqlalchemy` 를 기반으로,

```
SQL1 = "SELECT .." # 이 SQL 문은 분석 하고자 하는 Query 를 만들어 사용한다.  
df_data = pd.read_sql_query(SQL1, conn)
```

만약 만들고자 하는 *Class* 의 함수가 인자가 필요한 경우는 2가지가 있다.

1) 단순 *Scalar* 의 조합

```
SQL2 = "SELECT 1 as ARG1, 0.5 as ARG2 FROM DUAL"
```

2) 참조되는 데이터 (참조 테이블, 예를 들면 유사도 측정을 위한 *Reference* 테이블, 추론 테이블)

```
SQL2 = "SELECT ... FROM TABLE ...."
```

```
df_args = pd.read_sql_query(SQL2, conn)
```

그 이후 함수를 만들면 되고 그 모듈 명과 함수명을 **BDAE<sup>(TM)</sup>** 에 등록 한다.

(※ R 은 ROracle 을 이용해서 작업하면 되고 역시 과정은 비슷하다.)

## 22 | R 예제 참고 사항

R 은 앞서 언급한 바와 같이 코딩 스타일이 두가지 이다.

1. `function()` 형태 : 입력 변수는 임의의 이름으로 사용 가능, 명시적 리턴 `return df`
2. 서술 형태는 분석하려는 입력 데이터명은 `data` 이고, 부가적인 데이터는 `args` 로 고정되어 있다.  
`return` 구문 없이 `df` 를 한번만 써 주면 되며 `df` 이름 말고 다른 이름도 된다.  
단, 1,2 모두 리턴은 `data.frame` 포맷이며 `stringsAsFactor=FALSE` 로 해야 한다.

`stringsAsFactor` 부분은 R 분석가들은 당연히 잘 알고 있을 것이다.

무심코 `data.frame` 을 만들면 `Category` 성 컬럼은 1,2,3 등의 `Factor` 로 저장된다는 것을 ...

3. Python 처럼 리턴되는 `data.frame` 의 속성은 보편적으로 문자열, 숫자, `DateTime` 정도이다.  
`as.character()`, `as.numeric()`, ..

## 23 | Visualization 과 Serialization/DeSerialization

R/Python 모두 3가지 형태로 차트를 만들 수 있다.

1. 이미지 (GitHub 참조)
2. 이미지, base64 로 인코딩 한 String (GitHub 참조)
3. plotly 를 사용한 Interactive Java Script String (GitHub 참조)

2 번은 추천하지 않는다. 사이즈만 커지기 때문이다. 3 번은 대용량일 경우 추천하지 않지만, 어느 정도 작을 때에는 Mouse 이벤트를 받으면서 Interactive 해서 매우 직관적이다.

이미지나 모델의 바이너리를 `data.frame` 으로 만드는 것은 익혀두는 것이 좋다. 특히 모델을 학습 시킨 후에 `data.frame` 으로 만들어 DB 에 저장 후 추후에 이것을 다시 추론에 사용하는 기법은 보편적이다.

AI 학습 시에는 많은 검증 데이터 결과들, 각종 시각화 차트들, 파라미터들이 나오고 이를 보고 싶어 한다. SQL 문의 리턴은 단 1회이기 때문에, 이 모든 것들을 `DataFrame`, `data.frame` 에 모두 담을 수 있으며 그렇게 하면 된다.

특히 `Model` 의 경우는 `Serialization` 해야 하기 때문에 결국 `Oracle` 의 `BLOB` 으로 저장해야 하고, 추후 운영 시 실시간 추론(`Inference`) 를 할 때 이를 이용해 다시 `DeSerialization` 해서 사용해야 한다.

**BDAE<sup>(TM)</sup>** 를 이용하면 충분히 쉽고 가능하다.

## 24 | BDAE<sup>(TM)</sup> 는 실행 시 저장 공간을 사용하지 않는다.

BDAE<sup>(TM)</sup> 는 실행 시와 Python, R 엔진을 통해서 분석 후에 리턴 시 별도의 저장 공간이 아닌 모든 것이 메모리에서 이뤄진다.

BDAE<sup>(TM)</sup> 가 필요한 저장 공간은 오직 고객사의 Python, R 모듈을 관리 할 때만 사용하며 이는 거의 공간을 차지한다고 볼 수 없다. 실제 운영 데이터를 핸들링하고 분석 결과 등은 모두 메모리이다.

BDAE<sup>(TM)</sup> 의 리턴 결과를 DB 에 저장하는 방법은 Python, R 내에서 분석가가 별도의 세션으로 하는 방법, 그리고 다음과 같은 보편적인 Database 기법을 이용하면 된다.

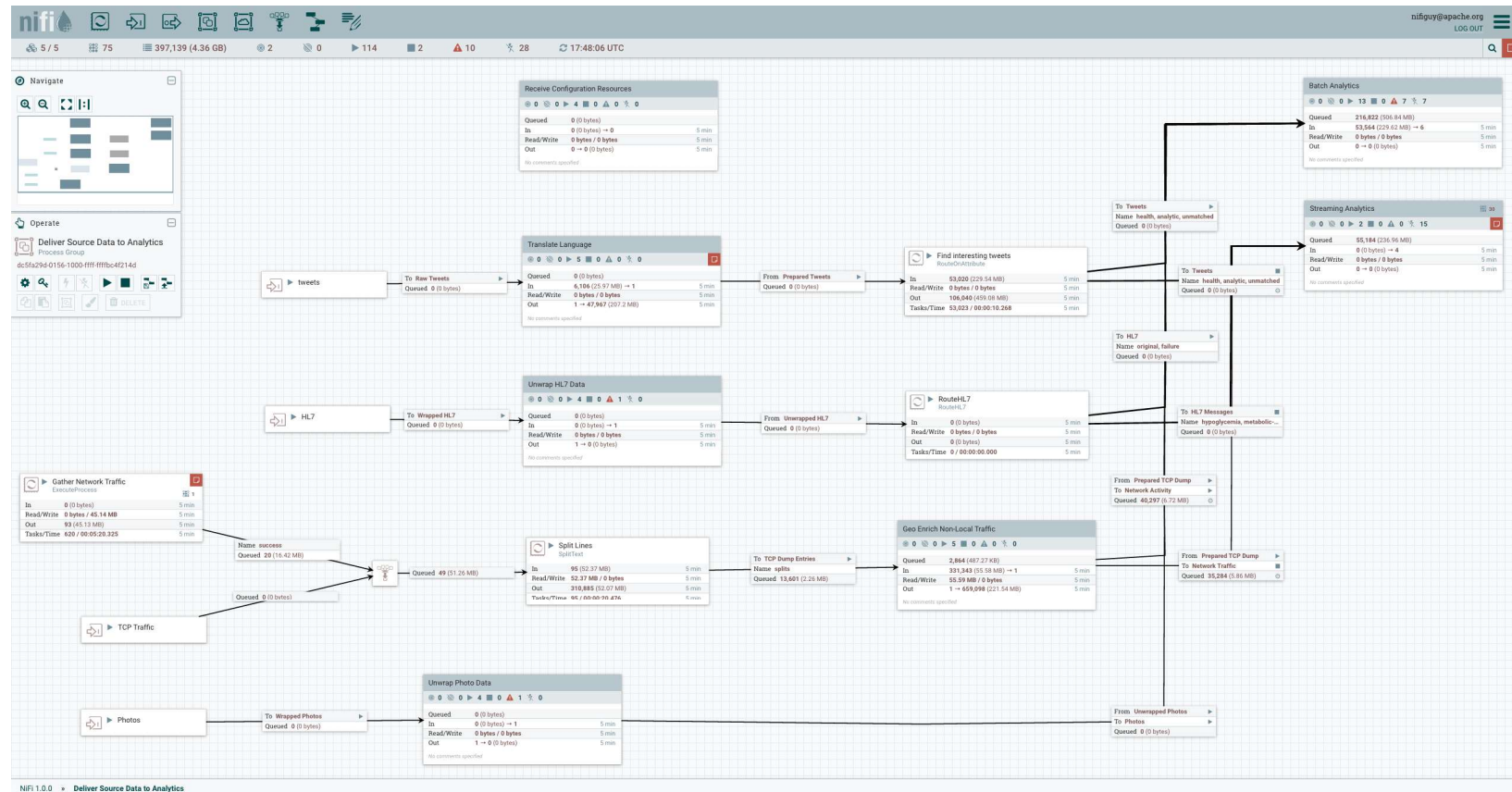
```
CREATE TABLE RESULT_01 AS SELECT ... ;  
INSERT INTO RESULT_02 SELECT ... ;
```

### ※ Oracle In-Database 프로그램들의 문제점과 BDAE<sup>(TM)</sup> 의 대책

1. Oracle Database<sup>(TM)</sup> 의 SGA, PGA 영역 등과는 별개로 관리되기 때문에, Python, R 내부 코드에서 방만하게 많은 메모리를 사용 하게 되면 Oracle Instance 가 있는 서버 자체를 죽일 수 있다. (OS 측면에서 Out of Memory)
2. 실제 Oracle R Enterprise 운영 시 발생하는 것을 목격했기 때문에, BDAE<sup>(TM)</sup> 는 하나의 세션 메모리에 대한 최대 가용 메모리를 설정하고, 실행 시 감시하게 된다. (매 10 초마다 해당 세션에서 감시됨.)
3. 또한, Oracle Instance 가 있는 노드의 메모리의 총량이 80 % 이상 일 경우 BDAE<sup>(TM)</sup> 는 새로운 세션을 거부하도록 되어 있다. 이 부분도 설정할 수 있도록 되어 있다.

## 24 | BDAE<sup>(TM)</sup> 와 Apache NIFI<sup>(TM)</sup> 연계

Apache NIFI<sup>(TM)</sup>와 BDAE<sup>(TM)</sup> 를 함께 사용하면 보다 안정적인 Workflow 로 배치 작업을 구성할 수 있다. 특히 파일 기반보다 SQL 기반으로 하면 효과적이며 깔끔하게 정리될 수 있다.



## BDAE(TM) 활용



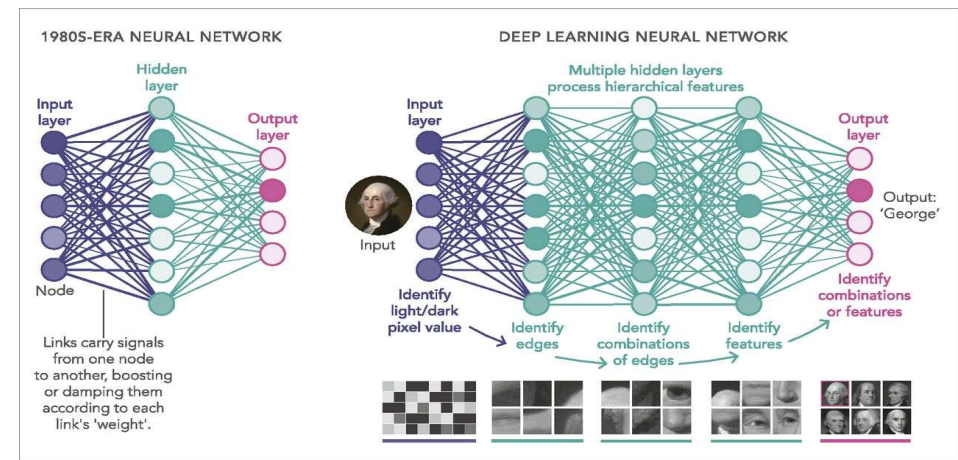
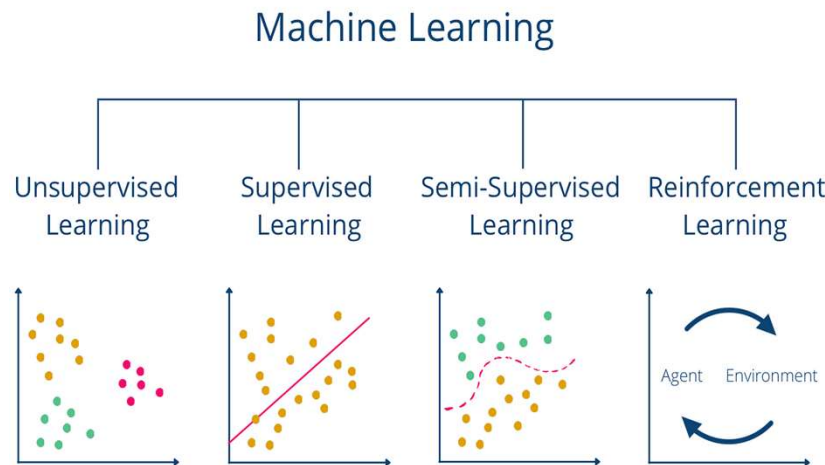
## 1 | BDAE(TM) 활용 (#1. Python, R 로 수행하는 다양한 분석 업무)

BDAE(TM)의 최초 개발 목적은 제조 분야의 시스템들, 예를 들면 SPC, FDC, YMS 등에 기존 솔루션에 손쉽게 최신 알고리즘을 적용할 수 있는 방안을 찾는 데에 있었다.

특히 패턴 인식을 통한 불량 분석, SPC, 기술 통계량, ANOVA, .. 등을 JAVA 등의 백엔드 등에서 수행하거나, 별도의 고가의 분석 제품을 구입하는 것에 대한 것이 과연 바람직한가? 에 대한 의구심.

실제 하이테크 현장에서는 Raw 데이터가 Oracle Database(TM) 이고, R 이나 Python 스크립트를 별도의 어플리케이션 서버에서 돌리는 형태가 보편적이다.

그러나, 별도의 서버에서 알고리즘 소스 관리의 문제, 복잡한 프로토콜 등을 BDAE(TM) 는 해결해 준다.

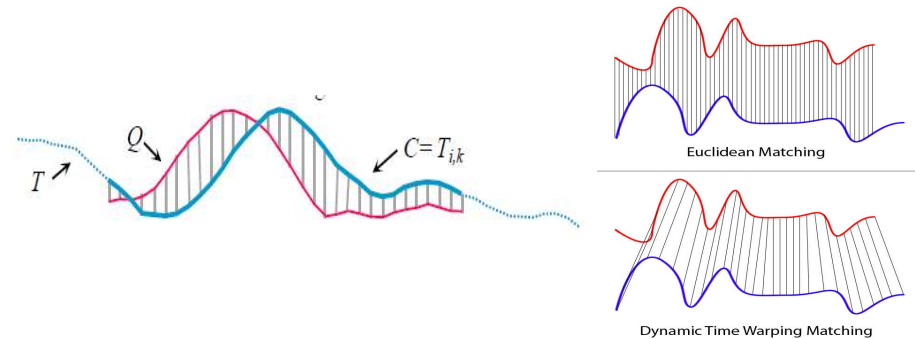


※ Python, R 이 분석 업무 (AI) 에는 가장 적합하며, 생산성이 가장 높은 언어들이다.

## 2 | BDAE<sup>(TM)</sup> 활용 (#1. Python, R 로 수행하는 다양한 분석 업무)

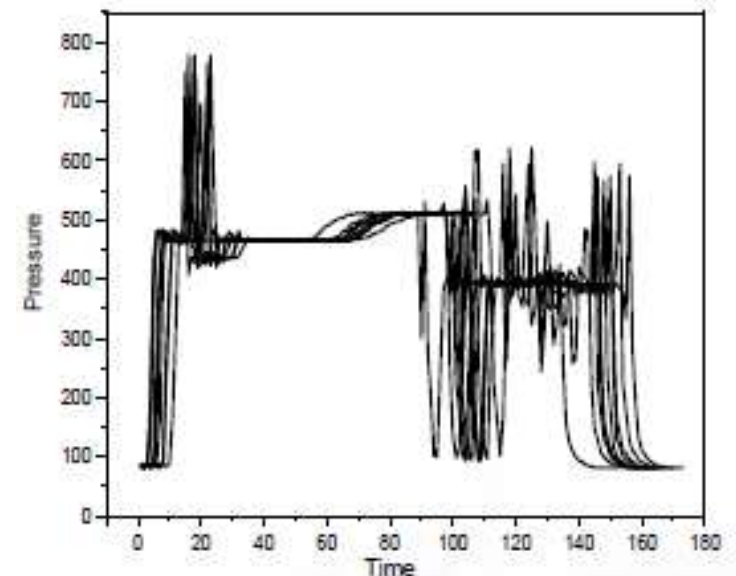
실시간의 분석을 빠르게 수행하여 그 결과를 아래와 같은 보편적인 SQL Query 결과로 받는다면 어플리케이션은 매우 간단한 구조가 된다.

※ BDAE<sup>(TM)</sup> 의 두번째 argument 의 용도는 비교를 위한 Query Data, 추론을 위한 Model 데이터 또는 Python, R 에서 사용하는 함수의 Argument 이다.



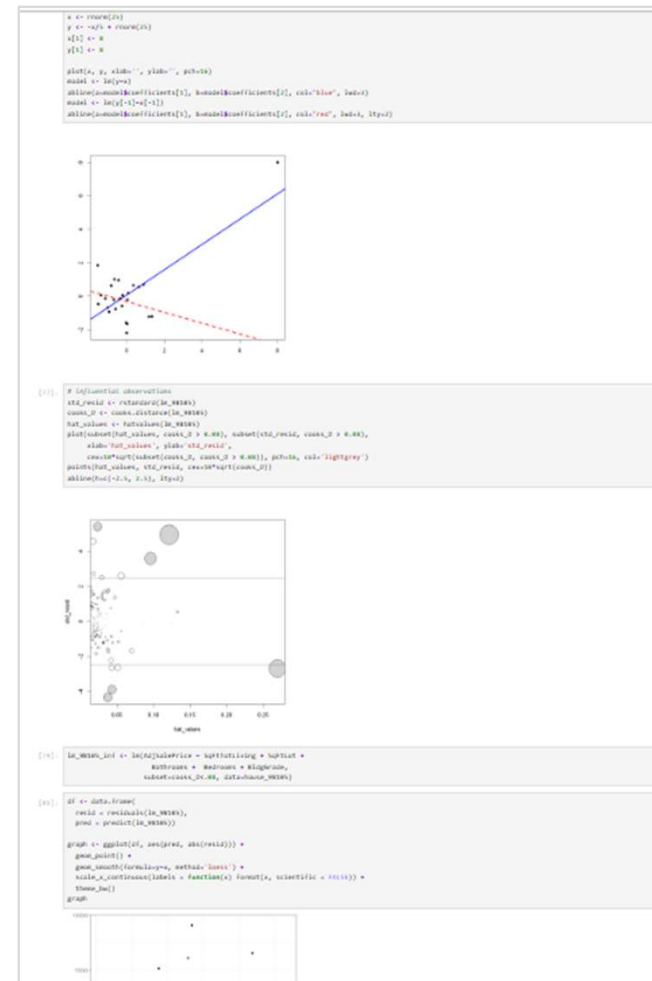
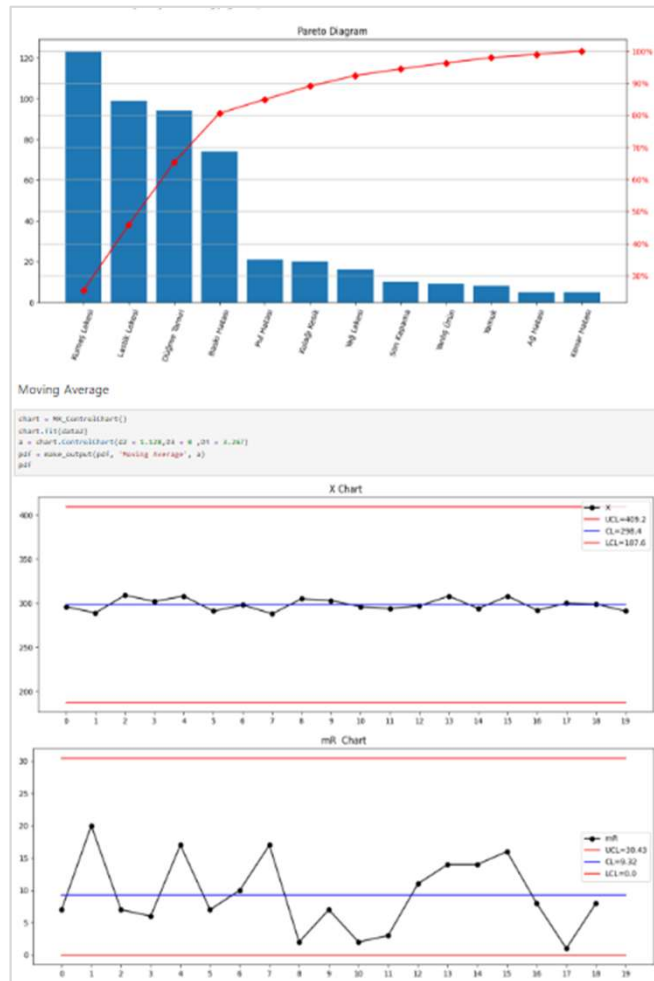
```
SQL> SELECT *
2 FROM TABLE (apGroupEvalParallel (
3     cursor (
4         SELECT *
5         FROM TRACE_DATA
6         WHERE EQP_ID = 'EPS001'
7           AND LOT_ID = 'LOT001'
8           AND ETC = '.....'
9     ),
10    cursor(SELECT * FROM GOLDEN_EQUIPMENT ...),
11    'SELECT CAST("A" AS VARCHAR2(40)) ITEM_ID,
12      1.0 SIMILARITY FROM DUAL',
13    'EQP_ID, LOT_ID, ....',
14    'DefectUtil:FastDTW');
```

PARAMETER_ID	SIMILARITY
RF_POWER_1	2.23
O2_PUMP_1	0.5
Ch1_TEMP_1	2.1
.....	.....
.....	.....



### 3 | BDAE<sup>(TM)</sup> 활용 (#2. 분석 업무의 배치 작업 및 통합)

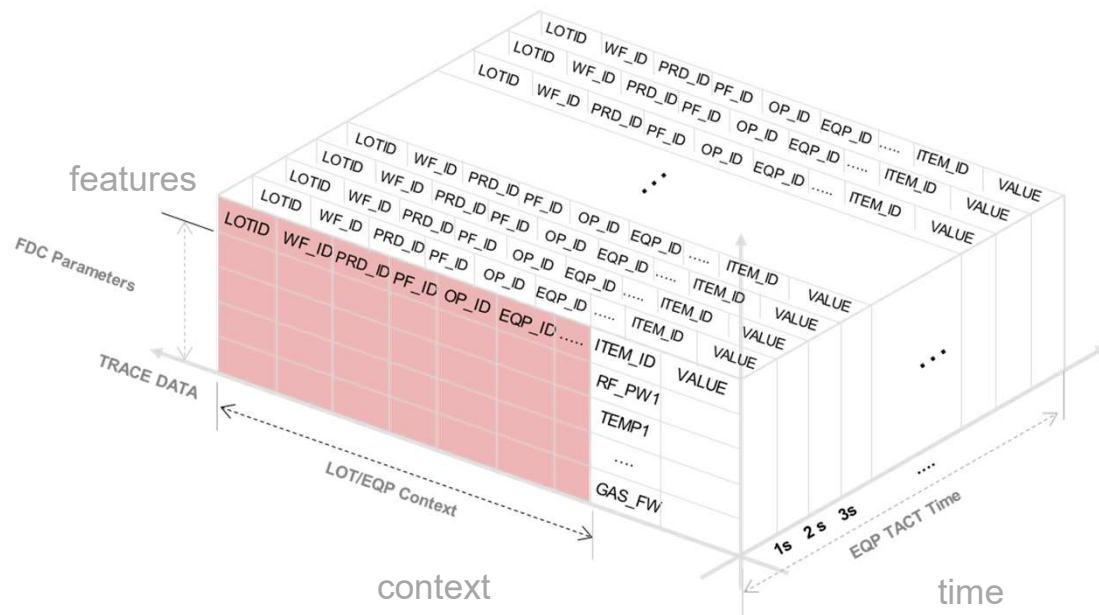
앞의 SQL 문을 Python 또는 R 로 Wrapping 하여 클래스로 만들면 다양한 분석 업무의 통합하여 한번에 수행할 수 있으며, 그 결과 또한 한번에 받을 수 있다.



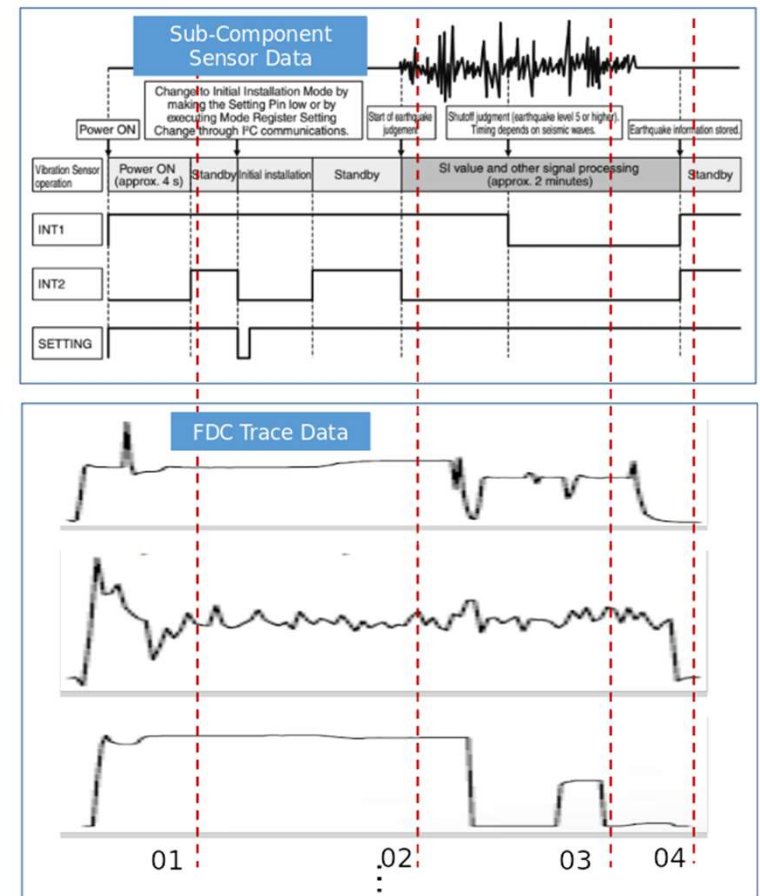
## 4 | BDAE<sup>TM</sup> 활용 (#4. 병렬 분산 처리)

대용량의 테이블을 그룹핑하여 동일한 알고리즘을 한번에 적용하여 결과를 받을 때에는 Python, R 의 병렬 처리를 이용하면 재활용성과 성능, 메모리를 많이 사용하게 된다.

이러한 병렬처리 부분은 Oracle In-Database 에 맡기고, Python, R 모듈은 병렬 처리를 고려하지 않는 단순한 형태를 가지게 하는 것이 BDAE<sup>TM</sup> 의 역할이고 Concept 이며 위의 문제점들은 해결된다.



Trace Data per 1 LOT/1 EQP



## 4 | BDAE<sup>TM</sup> 활용 (#4. 병렬 분산 처리)

다음의 Query 는 대용량 테이블의 병렬 분산 처리를 빨간 글씨의 그룹 컬럼들의 단위로 수행하는 예제이다. Oracle Hint (`/*+ parallel(20) */`) 부분으로 총 20개로 나누어서 병렬 처리 되며, 그 때 Python 모듈은 병렬 처리를 고려하지 않은 단순한 모듈 형태를 띄게 된다.

BDAE<sup>TM</sup> 는 이를 가능하게 만들어 준다.

```
SELECT /*+ parallel(5) */*
FROM table(apGroupEvalParallel(
  CURSOR(
    WITH TARGET_TBLE AS
    (
      SELECT * from FDC_TRACE
      WHERE 1=1
      AND EQP_ID='EQP-200'
      AND UNIT_ID='UNIT-02'
    )
    SELECT EQP_ID, UNIT_ID, LOT_ID, WAFER_ID, RECIPE, PARAM_ID,
      (VALUE - (AVG(VALUE) OVER (PARTITION BY PARAM_ID)))
      / (STDDEV(VALUE) OVER (PARTITION BY PARAM_ID)) AS NORMALIZED_VALUE
      FROM TARGET_TBLE
  ),
  NULL,
  'SELECT CAST(''A'' AS VARCHAR2(40)) EQP_ID,
    CAST(''A'' AS VARCHAR2(40)) UNIT_ID,
    CAST(''A'' AS VARCHAR2(40)) LOT_ID,
    CAST(''A'' AS VARCHAR2(40)) WAFER_ID,
    CAST(''A'' AS VARCHAR2(40)) RECIPE,
    CAST(''A'' AS VARCHAR2(40)) RESULT
    FROM DUAL',
  'EQP_ID,UNIT_ID,LOT_ID,WAFER_ID,RECIPE,PARAM_ID',
  'Standardization:normalize'));
```

LOTID	WF_ID	PRD_ID	OP_ID	EQP_ID	UNIT_ID	...	TIME	ITEM_ID	VALUE
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_001	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_002	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_003	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_004	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_005	653
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_006	78
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_007	112
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_008	8
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_009	212
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_010	3
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_011	7
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_012	22
...									
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_683	112.22
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_684	0.212

LOTID	WF_ID	PRD_ID	OP_ID	EQP_ID	UNIT_ID	...	TIME	ITEM_ID	VALUE
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_001	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_002	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_003	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_004	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_005	653
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_006	78
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_007	112
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_008	8
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_009	212
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_010	3
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_011	7
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_012	22
...									
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_683	112.22
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_684	0.212

:

LOTID	WF_ID	PRD_ID	OP_ID	EQP_ID	UNIT_ID	...	TIME	ITEM_ID	VALUE
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_001	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_002	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_003	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_004	21
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_005	653
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_006	78
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_007	112
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_008	8
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_009	212
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_010	3
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_011	7
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_012	22
...									
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_683	112.22
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_684	0.212



## 5 | BDAE<sup>TM</sup> 활용 (#5. 전처리)

많은 입력 데이터 부분의 전처리를 Python, R 이 수행하면 메모리, 성능의 문제가 발생한다.

BDAE<sup>TM</sup> 는 다음과 같은 복잡한 SQL 문을 수용하여 전처리 할 수 있게 하며 이는 Oracle Database<sup>TM</sup> 가 가장 앞선 SQL 엔진을 가지고 있기 때문이다.

타사의 DB 나 오픈소스, 하둡 에코 시스템 (Hive, Phoenix, Impala, ... ) 등은 ANSI-SQL 의 Subset 이기 때문에 이러한 고도화된 전처리를 할 수 없다.

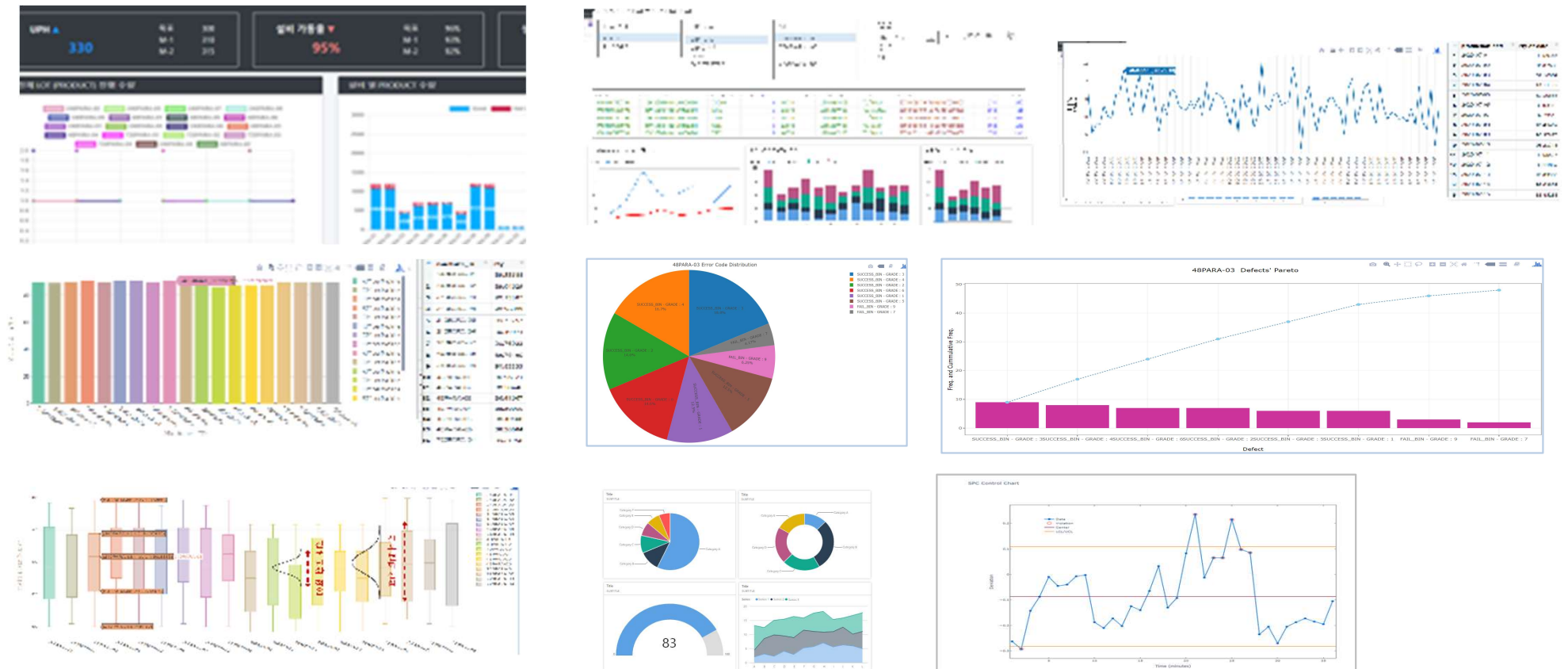
```
SELECT eqp_id, recipe_id, ..., time, parameter_name, sma, ema
FROM (
    SELECT eqp_id, recipe_id, time, ..., parameter_name, parameter_value
    FROM trace_data
    WHERE 1=1
        AND time between ... and ...
        AND step_seq = ' ...'
    ...
) a
MODEL
PARTITION BY (a.parameter_name, 2 / (1 + count(*) over (partition by a.parameter_name))
              smoothing_constant )
DIMENSION BY (row_number() over (partition by a.parameter_name order by a.time) rn)
MEASURES (a.time, a.parameter_value, sma, 0 ema)
(
    ema[1] = a.parameter_value[1],
    ema[rn > 1] order by rn = ( cv(smoothing_constant) * (parameter_value[cv()] - ema[cv() - 1]) ) + ema[cv() -
1]
)
ORDER BY eqp_id, a.recipe_id, ..., a.time, a.parameter_name;
```



## 6 | BDAE<sup>(TM)</sup> 활용 (#6. Smart Factory 분야)

통계적 품질 분석이나, 이상 감지, SPC, ... 등의 별도의 솔루션 도입 없이 Python 과 R 이 제공하는 각종 패키지들을 BDAE<sup>(TM)</sup> 에서 사용하면 별도 비용 없이 유연한 기능들을 SQL 문으로 손쉽게 기존 솔루션에 Embedding 할 수 있다.

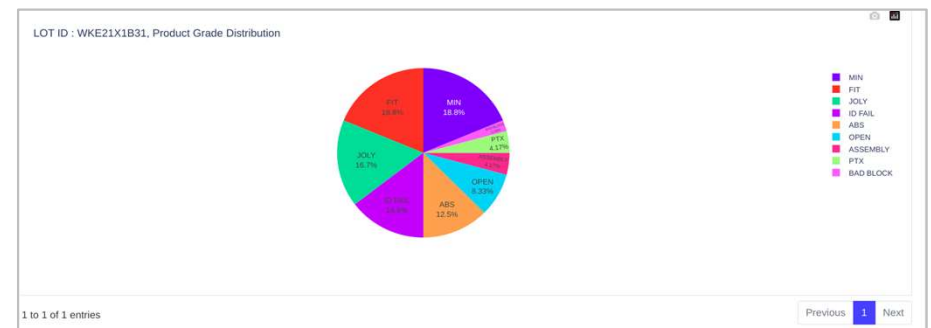
※ BDAE<sup>(TM)</sup> 는 SQL 문으로, 메모리 상에서 수행되기 때문에 어떠한 솔루션에도 추가 될 수 있다.



## 6 | BDAE<sup>(TM)</sup> 활용 (#6. Smart Factory 분야)

다음과 같은 LOT 에 포함된 Product 의 분류 및 시각화에는 단지 5 줄의 분석 코드와 SQL 문이 필요할 뿐이다.

```
WITH LOT_SUM_ONE AS (  
  SELECT LOT_ID,  
    CASE WHEN PROD_GRADE = 9 THEN 'BAD BLOCK'  
      WHEN PROD_GRADE = 8 THEN 'ASSEMBLY'  
      WHEN PROD_GRADE = 7 THEN 'PTX'  
      WHEN PROD_GRADE = 6 THEN 'ID FAIL'  
      WHEN PROD_GRADE = 5 THEN 'OPEN'  
      WHEN PROD_GRADE = 4 THEN 'ABS'  
      WHEN PROD_GRADE = 3 THEN 'JOLY'  
      WHEN PROD_GRADE = 2 THEN 'FIT'  
      WHEN PROD_GRADE = 1 THEN 'MIN'  
      ELSE  
        'N/A'  
      END PROD_GRADE_DESC  
    , COUNT(PROD_GRADE) CNT from table (  
      productExplodeEvalClob(cursor(  
        SELECT *  
        FROM LOT_SUM  
        WHERE LOT_ID = 'WKE21X1B31'  
          AND MACHINE_ID = '48PARA-03' ...DURABLE_ID = 'Z718'))))  
    GROUP BY LOT_ID, PROD_GRADE  
  )  
SELECT *  
FROM table(apTableEval(  
  cursor(  
    SELECT * FROM LOT_SUM_ONE  
  ),  
  NULL,  
  'XML',  
  'LOTSUM_ERR_PIE:display'))
```



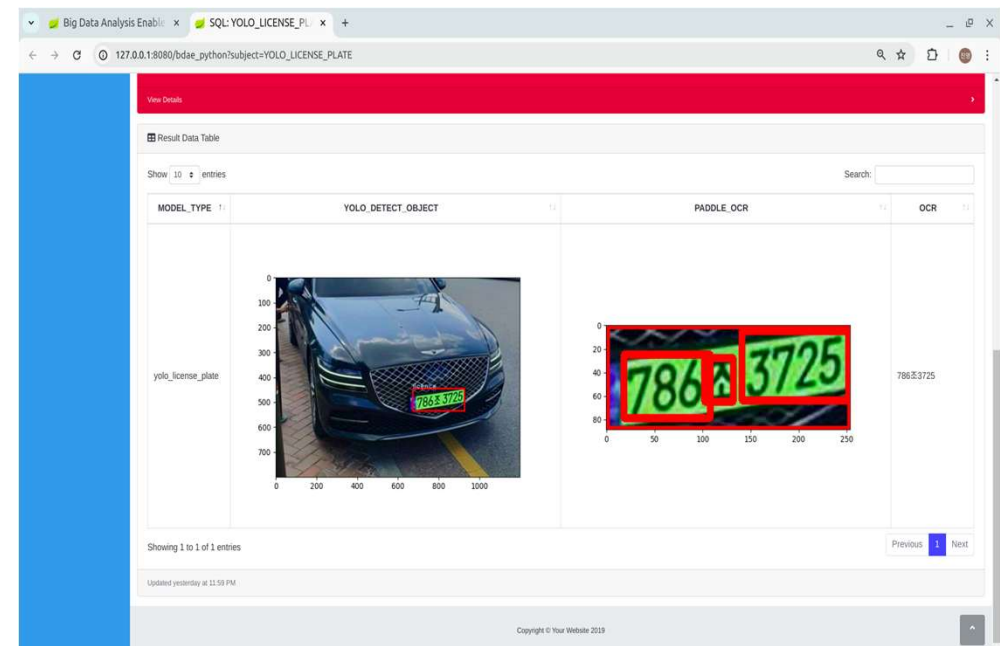
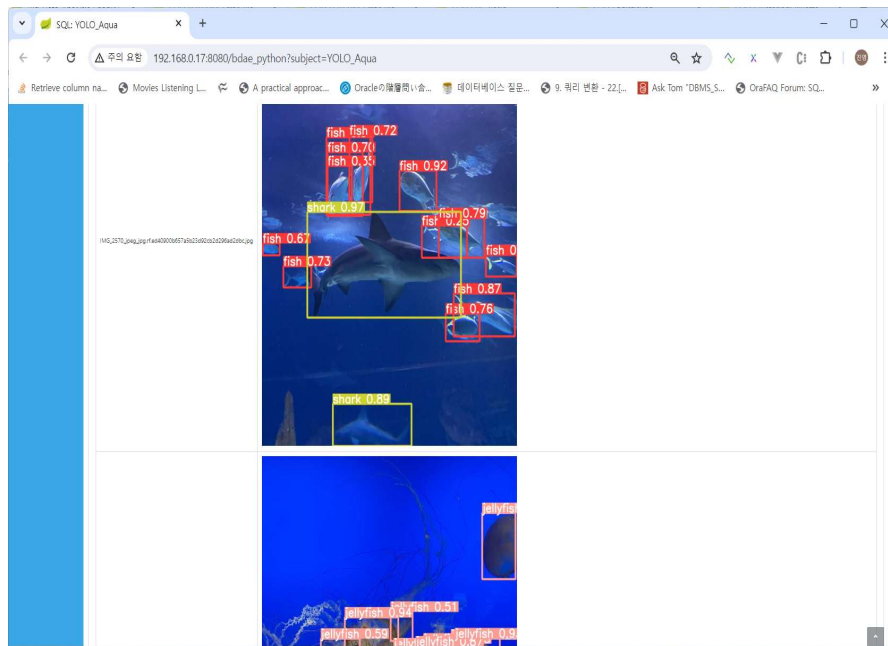


## 7 | BDAE<sup>(TM)</sup> 활용 (#7. Deep Learning 추론)

비용이나 GPU 에 의존적인 알고리즘의 학습을 위해 BDAE<sup>(TM)</sup> 가 설치되어 있지 않은 곳에서 Model 을 학습하고 그것을 실시간에 BDAE<sup>(TM)</sup> 를 이용하여 추론에 사용할 수 있다.

( 설비 별, 제품 별 Model 이 다를 수 있기 때문이며 BDAE<sup>(TM)</sup> 는 DeSerialization 을 지원하기 때문 )

※ 시계열 데이터 (1D Tensor) 를 이미지로 변환 후 이것을 Deep Learning 이미지 합성곱으로 불량 검증을 하는 것이 더 효과적이라는 논문이 있고, 이를 실제로 활용하는 경우가 있음. BDAE<sup>(TM)</sup> 는 배치작업으로 많은 이미지를 한번에 생성할 수 있다.



## 8 | BDAE<sup>(TM)</sup> Web 에 대해서

BDAE<sup>(TM)</sup>의 장점은 SQL 문으로 호출된다는 것이고, 따라서 어떠한 Oracle Connectivity 를 제공하는 클라이언트에서 호출될 수 있다.

BDAE<sup>(TM)</sup> Web 은 Spring Boot + JSP (Tiles) 기반이며 Sprint Boot 에서 제공하는 jdbcTemplate 을 사용해서 BDAE<sup>(TM)</sup> SQL 을 호출 한다. (매우 단순하고, 결과는 jQuery, dataTable 하나로 되어 있음.)

그 결과가 String, BLOB, CLOB, .. 인지 호출 결과 즉시 알기 때문에 화면 처리가 매우 간단할 수 있다. 특히 시각화의 경우 <div> 형태로 오기 때문에 적절한 GUI 구성이 손쉽다.

다만, 이 경우 Spring Boot 프로젝트의 static 부분에 이미지의 javascript 관련 라이브러리, css 등이 있어야 Interactive 한 것이 동작할 수 있지만, 이 경우는 그리 세팅이 어려운 부분은 아님.

BDAE<sup>(TM)</sup> 모듈에서 이미지 외에 각종 바이너리들, 예를 들면 학습된 모델, 제너레이션 된 문서 등을 리턴하는 것은 가능하다. 이 경우 Python, R 모듈에서 Serialization 되어야 하고 그것을 BLOB 으로 BDAE<sup>(TM)</sup> 이 받아서 자동으로 컨버팅 후 처리하게 된다.

이 부분은 GitHub 예제를 참조하면 된다.