pg_onnx

PostgreSQL extension 개발기

pgday.Seoul 2023 신기배

pg_onnx

PostgreSQL extension 개발기

- pg_onnx PostgreSQL extension 소개
- Background Worker 활용 방법
- pg_regress로 테스트 자동화

pg_onnx

ML 추론을 DB에서

PostgreSQL + ONNX Runtime

https://github.com/kibae/pg_onnx

pg_onnx

ML 추론을 DB에서

PostgreSQL + ONNX Runtime

ONNX 파일을 실행하는 포터블 런타임

https://onnxruntime.ai/

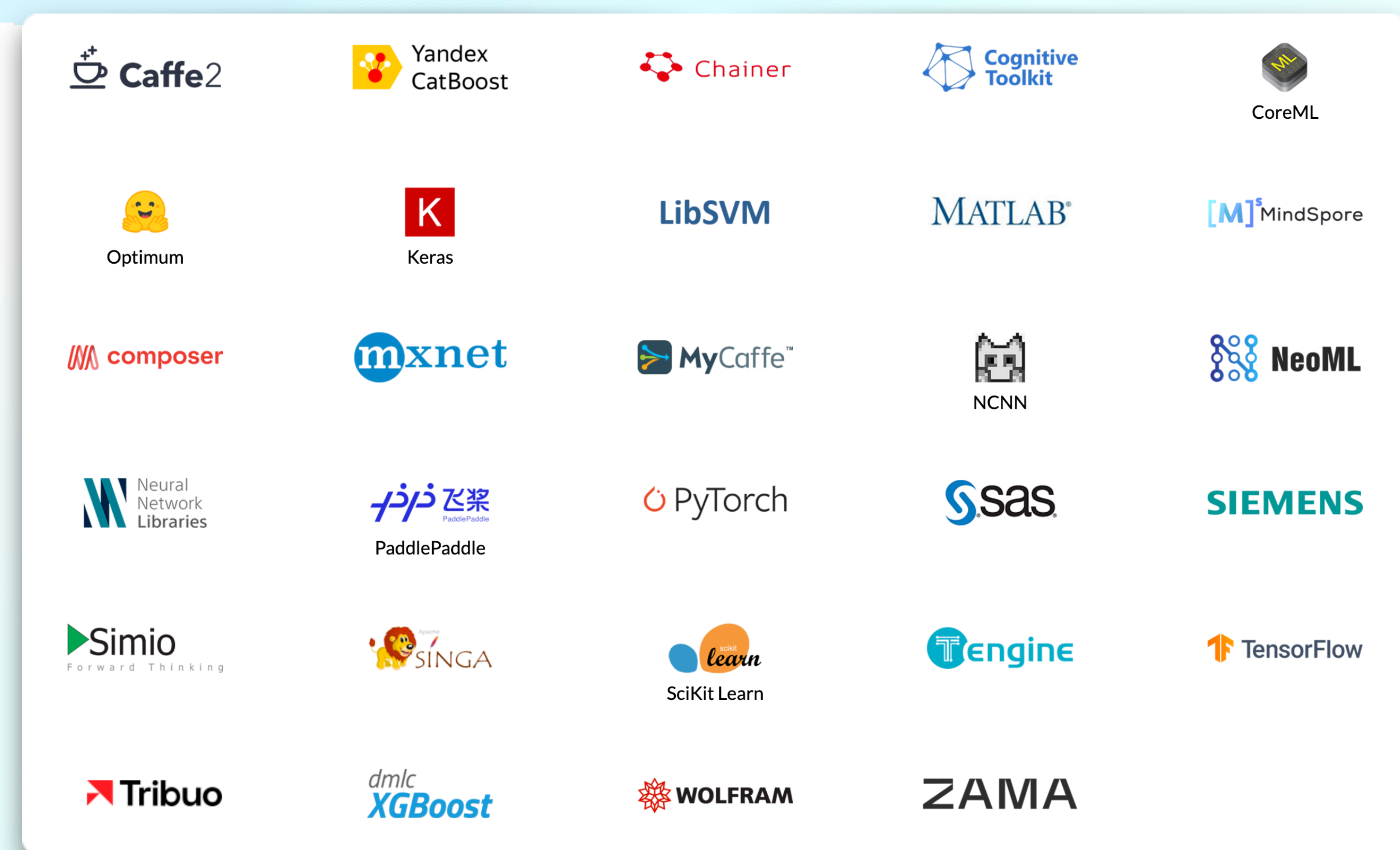
pg_onnx

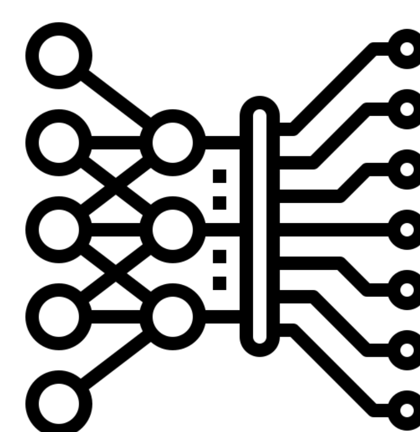
ML 추론을 DB에서

PostgreSQL + **ONNX** Runtime

Open Neural Network Exchange

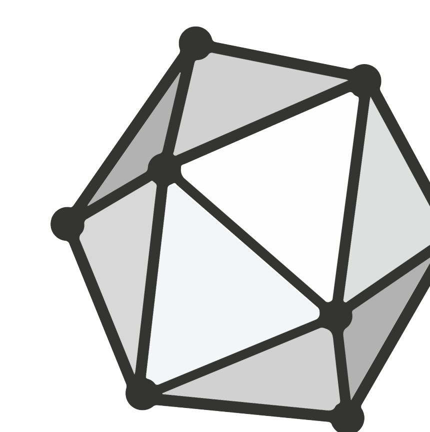
The open standard for machine learning interoperability



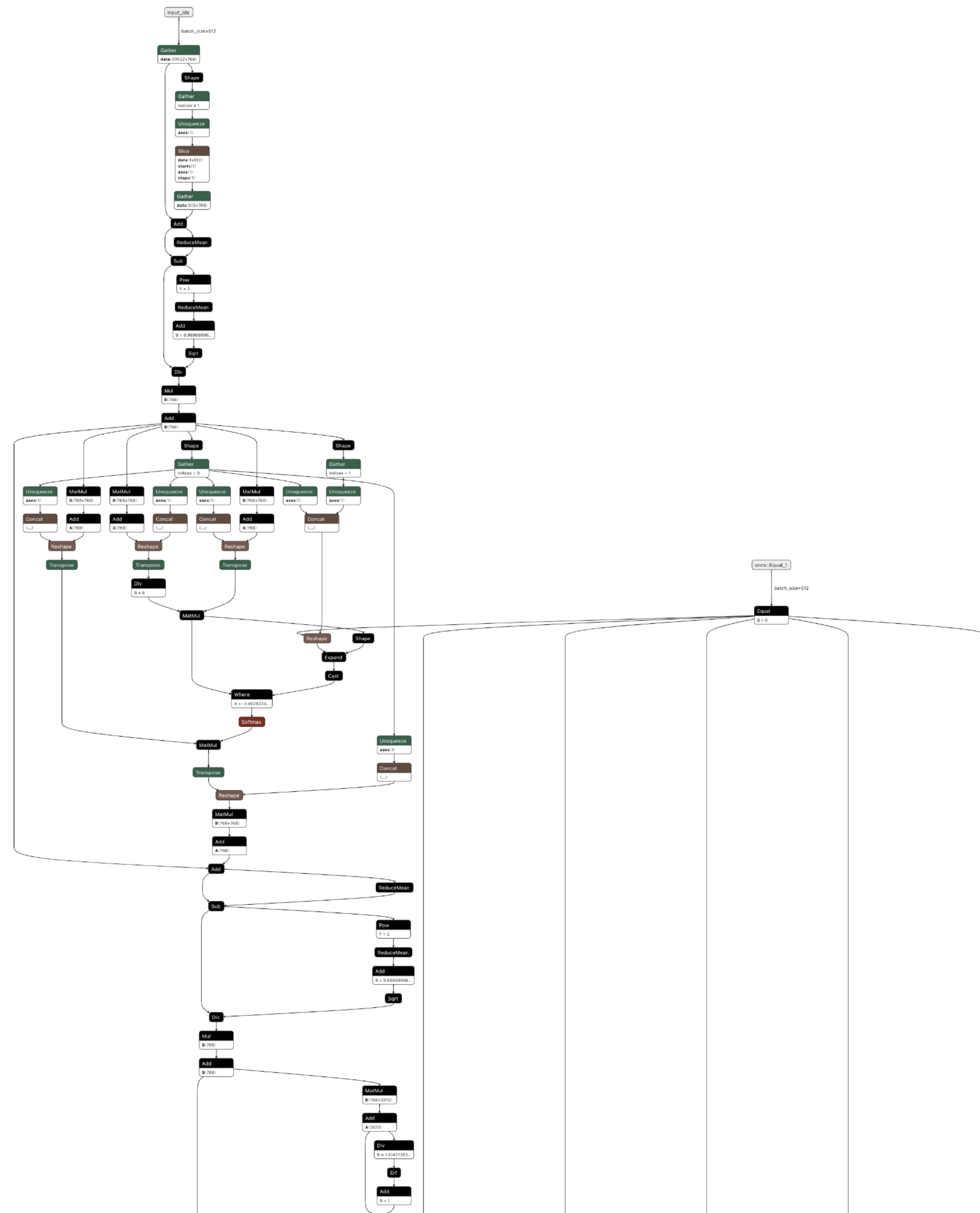


ML 모델

변환



ONNX 파일



input_ids
batch_size×512
Gather
data(30522×768)
Shape
Gather
indices = 1
Unsqueeze
axes(1)
Slice
data(1×512)
starts(1)
axes(1)
steps(1)
Gather
data(512×768)
Add
ReduceMean
Sub
Pow
Y = 2
ReduceMean
Add
B = 9.99999996...
Sqrt
Div
Mul
B(768)
Add
B(768)
Shape
Gather
indices = 0
onnx::Equal_1
batch_size×512
Equal
Softmax

pg_onnx

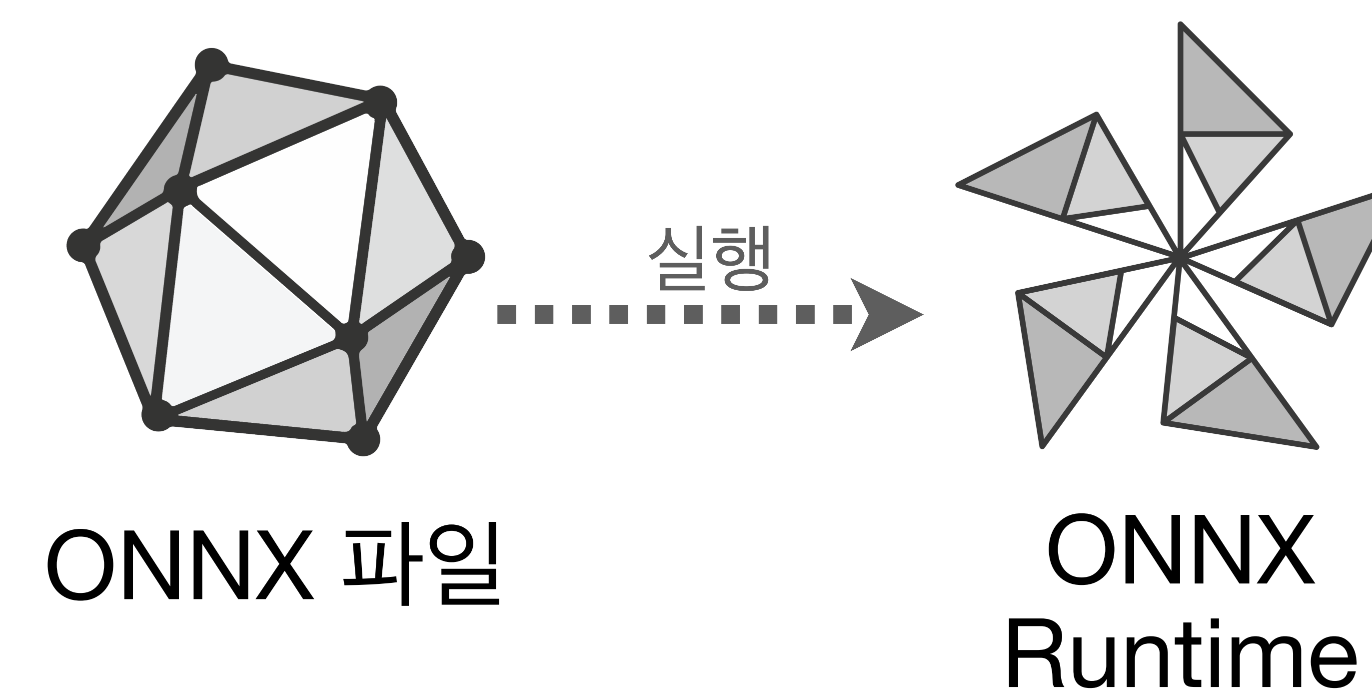
ML 추론을 DB에서

PostgreSQL + **ONNX Runtime**

ONNX 파일을 실행하는 포터블 런타임

https://onnxruntime.ai/

Platform	Windows	Linux	Mac	Android	iOS	Web Browser		
API	Python	C++	C#	C	Java	JS	Obj-C	WinRT
Architecture	X64	X86	ARM64	ARM32	IBM Power			
Hardware Acceleration	Default CPU	CoreML	CUDA	DirectML				
	MIGraphX	NNAPI	oneDNN	OpenVINO				
	ROCm	QNN	TensorRT	ACL (Preview)				
	ArmNN (Preview)	Azure (Preview)	CANN (Preview)	Rockchip NPU (Preview)				
	TVM (Preview)	Vitis AI (Preview)	XNNPACK (Preview)					



pg_onnx

Motivation

- ML 모델러와 백엔드 개발자 간 경험의 괴리
  - ML 모델러
    - Python 주로 활용. Transformers, scikit-learn, pandas 등 AI 관련 라이브러리는 잘 다룸
    - 학습을 위한 소스 데이터(input)가 주로 CSV 파일 형식임
    - 학습 후 데이터를 받아서 추론(inference)를 수행할 수는 있으나 이를 API 형태로 제공하는 것에 미숙
  - 백엔드 개발자
    - ML이나 학습된 모델을 실행하는 방법을 잘 모름
    - 외부 API나 DBMS 등을 호출하는 것에 익숙해져 있고 구조화된 응답(JSON 등)으로 받기를 원함
    - ML 모델같은 블랙박스가 자원을 소모하며 생겨나는 잠재적 위험으로 인한 장애 발생에 대한 우려
    - GPU라는 자원을 다뤄본 경험이 거의 없음

pg_onnx

Motivation

- 학습된 모델을 추론(inference)할 때 DB의 데이터를 활용하려면?
- 추론된 결과를 DB에 저장하고 이를 정렬이나 필터링에 활용하려면?
    - 그리고 이를 자동화 하려면?

pg_onnx

onnxruntime-server

- 좋은 DX(developer experience)를 위해 개발
  - No-Build, Zero Configuration
  - Ready-To-Run Docker Image 제공
- ONNX 추론을 위한 TCP, HTTP/HTTPS REST API
  - 내장된 Swagger Document 제공하여 테스트 및 협업자에게 정보 제공 용이
- CUDA를 통해 GPU 가속 지원
- C++, Async IO, Managed Thread Pool

ONNX Runtime Session

GET /api/sessions List sessions

POST /api/sessions Create sessions

Create a new session

Parameters

No parameters

Request body

```
{
  "model": "sample",
  "version": "2"
}
```

Execute

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/sessions' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "model": "sample",
  "version": "2"
}'
```

Request URL

http://localhost:8080/api/sessions

Server response

Code Details

200

Response body

```
{
  "created_at": 1694439555,
  "execution_count": 0,
  "inputs": {
    "input_ids": "int64[-1,512]",
    "onnx::Equal_1": "int64[-1,512]"
  },
  "last_executed_at": 0,
  "model": "sample",
  "option": {
    "cuda": false
  },
  "outputs": {
    "output": "int64[-1]"
  },
  "version": "2"
}
```

Response headers

content-length: 225

POST /api/sessions/{model}/{version} Execute session

Execute a session

Parameters Cancel Reset

Name	Description
model * required string (path)	Model name: sample
version * required string (path)	Model version: 2

Request body application/json

```
{
  "input_ids": [
    [
      101, 11834, 21600, 2102, 9005, 12098, 8566, 5740, 6853, 1999, 1996, 2806, 1997, 15262, 19699, 14663, 1005, 1055,
      3203, 8447, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/sessions/sample/2' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "input_ids": [
    [
      101, 11834, 21600, 2102, 9005, 12098, 8566, 5740, 6853, 1999, 1996, 2806, 1997, 15262, 19699, 14663, 1005, 1055,
      3203, 8447, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Request URL

http://localhost:8080/api/sessions/sample/2

Server response

Code Details

200

Response body

```
{
  "output": [
    13,
    14,
    11
  ]
}
```

Download

Response headers

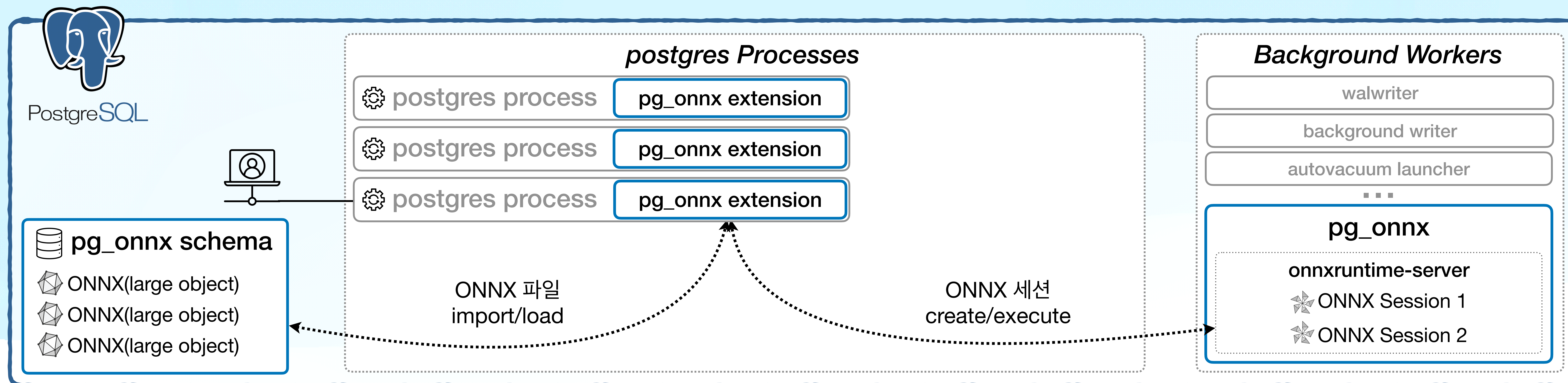
content-length: 21

pg_onnx

ML 추론을 DB에서

```
PostgreSQL +
    pg_onnx extension {
        Functions +
        onnxruntime-server {
            onnxruntime
        }
    }
```

pg_onnx



ONNX Model Functions

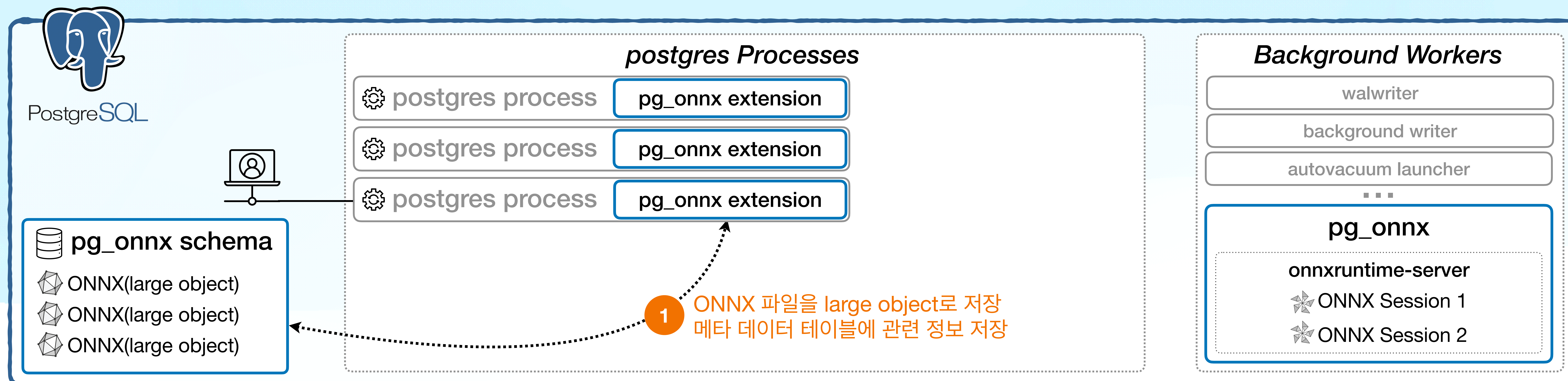
- pg_onnx_import_model(TEXT, TEXT, BYTEA, JSONB, TEXT)
- pg_onnx_drop_model(TEXT, TEXT)
- pg_onnx_list_model()
- pg_onnx_inspect_model_bin(BYTEA)

ONNX Session Functions

- pg_onnx_create_session(TEXT, TEXT)
- pg_onnx_describe_session(TEXT, TEXT)
- pg_onnx_execute_session(TEXT, TEXT, JSONB)
- pg_onnx_destroy_session(TEXT, TEXT, JSONB)
- pg_onnx_list_session()

pg_onnx

pg_onnx_import_model(TEXT, TEXT, BYTEA, JSONB, TEXT)



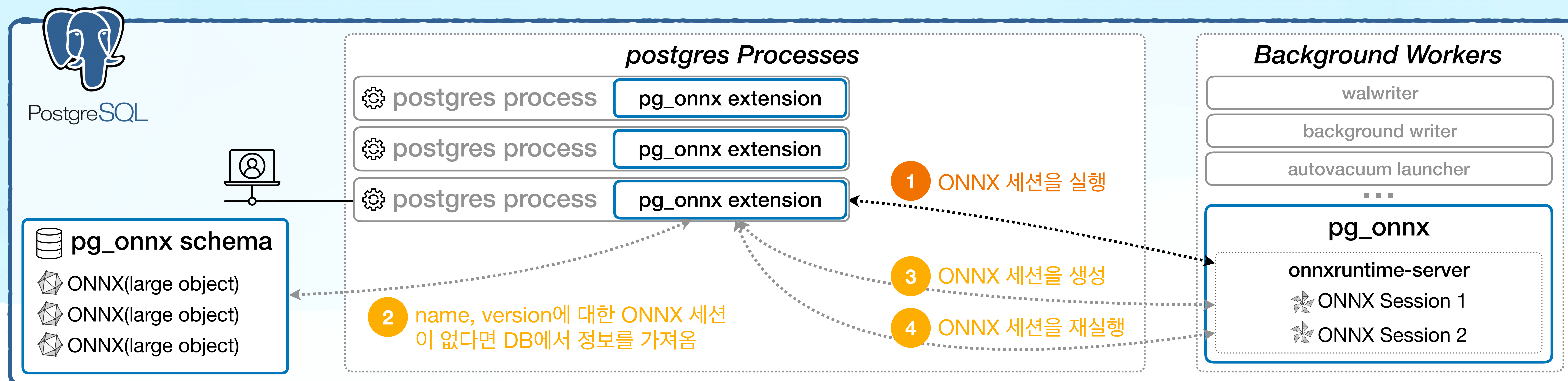
Arguments

- name(TEXT): Model 이름
- version(TEXT): Model 버전
- model(BYTEA): ONNX 파일 바이너리 데이터
- option(JSONB): 옵션(GPU 지원 등)
- description(TEXT): Model 설명

```
SELECT pg_onnx_import_model(
    'sample_model',
    'v20230101',
    PG_READ_BINARY_FILE('/your_model_path/model.onnx')::bytea,
    '{"cuda": true}'::jsonb,
    'sample model'
);
```

pg_onnx

pg_onnx_execute_session(TEXT, TEXT, JSONB)



Arguments

- name(TEXT): Model 이름
- version(TEXT): Model 버전
- inputs(JSONB): ONNX 추론을 위한 입력값들

```
SELECT pg_onnx_execute_session('sample_model', 'v20230101', '{
  "x": [[1], [2], [3]],
  "y": [[3], [4], [5]],
  "z": [[5], [6], [7]]
}');
```

pg_onnx

Trigger 활용

```sql
CREATE TABLE trigger_test_table (
  id         SERIAL PRIMARY KEY,
  value1     INT,
  value2     INT,
  value3     INT,
  prediction FLOAT);
```

```sql
SELECT pg_onnx_import_model(
  'sample_model',                -- Model 이름
  'v20230101',                   -- Model 버전
  PG_READ_BINARY_FILE('/model_path/model.onnx')::bytea,
  '{"cuda": true}'::jsonb, -- 옵션
  'sample model'                 -- Model 설명
);
```

ONNX 모델 Import

```sql
-- trigger function
CREATE OR REPLACE FUNCTION trigger_test_new_data()
  RETURNS TRIGGER AS
$$
DECLARE
  result jsonb;
BEGIN
  result := pg_onnx_execute_session(
    'sample_model', 'v20230101',
    JSONB_BUILD_OBJECT(
      'x', ARRAY [[NEW.value1]],
      'y', ARRAY [[NEW.value2]],
      'z', ARRAY [[NEW.value3]]));

  -- output shape: float[-1,1]
  -- eg: {"output": [[0.6492120623588562]]}
  NEW.prediction := result -> 'output' -> 0 -> 0;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger 함수

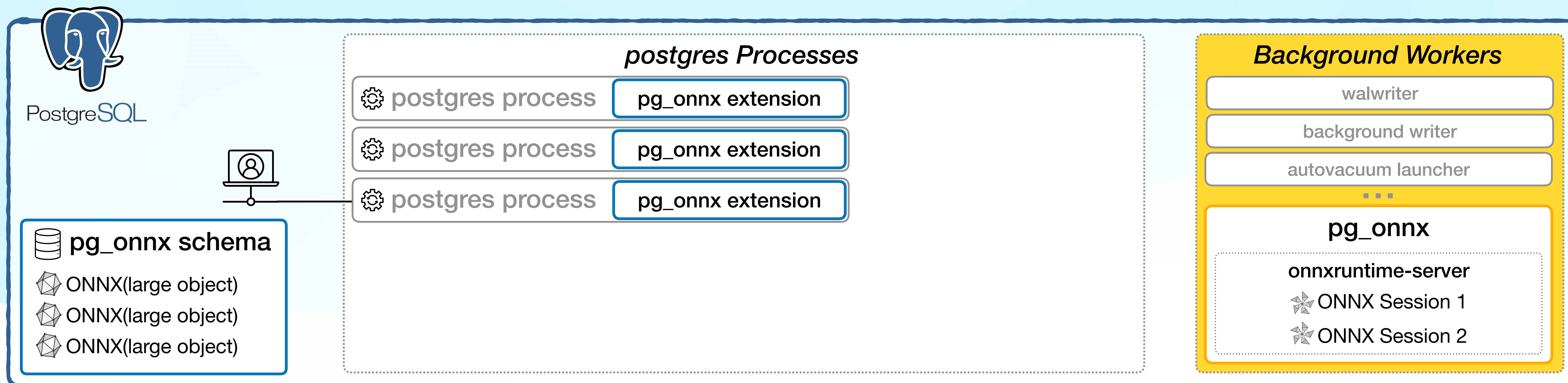
1. NEW record의 값들을 활용하여 ONNX 입력값 JSON 생성
2. ONNX 세션 실행 결과를 NEW record의 결과값 컬럼에 입력

```sql
-- trigger
CREATE TRIGGER trigger_test_insert
  BEFORE INSERT ON trigger_test_table FOR EACH ROW
EXECUTE PROCEDURE trigger_test_new_data();
```

Insert 전 처리되는 Trigger

pg_onnx

Background Worker



- 모든 postgres 프로세스마다 ML 모델을 메모리에 올리면 자원 고갈
- postgres 프로세스 내의 pg_onnx extension <- TCP/IP 통신 -> Background Worker[onnxruntime-server]

Background Worker

프로세스 생성하기

void _PG_init(void) {

1. BackgroundWorker 구조체 설정
2. RegisterDynamicBackgroundWorker 함수 호출
3. WaitForBackgroundWorkerStartup 로 시작되었는지 체크(optional)

}

```c
void _PG_init(void) {
    // Background Worker 생성, 설정을 위한 구조체
    BackgroundWorker worker;
    // Background Worker 핸들. Background Worker를 생성하면 이 값이 채워진다. 이 값으로 Background Worker를 제어할 수 있다.
    BackgroundWorkerHandle *handle;

    // Background Worker에서 실행된 것이라면 종료
    if (IsBackgroundWorker) return;

    // Background Worker 구조체 초기화
    MemSet(&worker, 0, sizeof(BackgroundWorker));
    // Background Worker flag 설정.
    // BGWORKER_SHMEM_ACCESS는 Background Worker가 공유 메모리에 접근할 수 있도록 한다.
    // BGWORKER_BACKEND_DATABASE_CONNECTION는 Background Worker가 데이터베이스 연결(SPI)을 할 수 있도록 한다.
    worker.bgw_flags = BGWORKER_SHMEM_ACCESS | BGWORKER_BACKEND_DATABASE_CONNECTION;
    // Background Worker가 실행되는 시기를 설정한다. extension마다 다르겠지만 보통 Recovery가 완료된 후 실행되도록 설정한다.
    // BgWorkerStart_PostmasterStart, BgWorkerStart_ConsistentState, BgWorkerStart_RecoveryFinished
    worker.bgw_start_time = BgWorkerStart_RecoveryFinished;
    // Background Worker 이름 설정. 프로세스 목록에 표시되고 고유해야 함.
    snprintf(worker.bgw_name, BGW_MAXLEN, "pg_onnx");
    // Background Worker 타입 설정. Background Worker를 구분하는데 사용된다.
    snprintf(worker.bgw_type, BGW_MAXLEN, "pg_onnx");
    // Background Worker가 실행될 라이브러리 이름 설정. 확장자는 제외한다.
    snprintf(worker.bgw_library_name, BGW_MAXLEN, "pg_onnx");
    // Background Worker가 실행될 함수 이름 설정.
    snprintf(worker.bgw_function_name, BGW_MAXLEN, "worker_main");
    // Background Worker가 재실행 될 시간 간격 설정. 초 단위. BGW_NEVER_RESTART로 설정하면 재실행되지 않는다.
    worker.bgw_restart_time = BGW_NEVER_RESTART;
    // Background Worker가 실행될 때 신호를 전달받을 PID. 0으로 설정하면 신호를 받지 않는다.
    // MyProcPid를 설정하면 Background Worker 실행 상태가 변경될 때 신호를 받을 수 있다.
    // 아래 WaitForBackgroundWorkerStartup 함수를 호출해서 정상적인 응답을 받으려면 이 값을 설정해야 한다.
    worker.bgw_notify_pid = MyProcPid;

    if (!RegisterDynamicBackgroundWorker(&worker, &handle))
        elog(ERROR, "could not start background process");

    pid_t pid; // Background Worker의 프로세스 ID
    BgwHandleStatus status = WaitForBackgroundWorkerStartup(handle, &pid);
    if (status == BGWH_STOPPED || status == BGWH_POSTMASTER_DIED)
        elog(ERROR, "could not start background process");

    Assert(status == BGWH_STARTED);
}
```

```
// Background Worker 생성, 설정을 위한 구조체
BackgroundWorker worker;
// Background Worker 핸들. Background Worker를 생성하면 이 값이 채워진다. 이 값으로 Background Worker를 제어할 수 있다.
BackgroundWorkerHandle *handle;

// Background Worker에서 실행된 것이라면 종료
if (IsBackgroundWorker) return;
```

```c
// Background Worker 구조체 초기화
MemSet(&worker, 0, sizeof(BackgroundWorker));

// Background Worker flag 설정.
// BGWORKER_SHMEM_ACCESS는 Background Worker가 공유 메모리에 접근할 수 있도록 한다.
// BGWORKER_BACKEND_DATABASE_CONNECTION는 Background Worker가 데이터베이스 연결(SPI)을 할 수 있도록 한다.
worker.bgw_flags = BGWORKER_SHMEM_ACCESS | BGWORKER_BACKEND_DATABASE_CONNECTION;

// Background Worker가 실행되는 시기를 설정한다. extension마다 다르겠지만 보통 Recovery가 완료된 후 실행되도록 설정한다.
// BgWorkerStart_PostmasterStart, BgWorkerStart_ConsistentState, BgWorkerStart_RecoveryFinished
worker.bgw_start_time = BgWorkerStart_RecoveryFinished;

// Background Worker 이름 설정. 프로세스 목록에 표시되고 고유해야 함.
snprintf(worker.bgw_name, BGW_MAXLEN, "pg_onnx");

// Background Worker 타입 설정. Background Worker를 구분하는데 사용된다.
snprintf(worker.bgw_type, BGW_MAXLEN, "pg_onnx");

// Background Worker가 실행될 라이브러리 이름 설정. 확장자는 제외한다.
snprintf(worker.bgw_library_name, BGW_MAXLEN, "pg_onnx");

// Background Worker가 실행될 함수 이름 설정.
snprintf(worker.bgw_function_name, BGW_MAXLEN, "worker_main");

// Background Worker가 재실행 될 시간 간격 설정. 초 단위. BGW_NEVER_RESTART로 설정하면 재실행되지 않는다.
worker.bgw_restart_time = BGW_NEVER_RESTART;

// Background Worker가 실행될 때 신호를 전달받을 PID. 0으로 설정하면 신호를 받지 않는다.
// MyProcPid를 설정하면 Background Worker 실행 상태가 변경될 때 신호를 받을 수 있다.
// 아래 WaitForBackgroundWorkerStartup 함수를 호출해서 정상적인 응답을 받으려면 이 값을 설정해야 한다.
worker.bgw_notify_pid = MyProcPid;
```

```c
if (!RegisterDynamicBackgroundWorker(&worker, &handle))
   elog(ERROR, "could not start background process");

pid_t pid; // Background Worker의 프로세스 ID
BgwHandleStatus status = WaitForBackgroundWorkerStartup(handle, &pid);
if (status == BGWH_STOPPED || status == BGWH_POSTMASTER_DIED)
   elog(ERROR, "could not start background process");

Assert(status == BGWH_STARTED);
```

```
|-+- 55714 kibae /opt/homebrew/Cellar/postgresql@14
| |--= 55718 kibae postgres: checkpointer
| |--= 55722 kibae postgres: stats collector
| \--= 55748 kibae postgres: pg_onnx
```

```c
void _PG_init(void) {
  BackgroundWorker worker;
  BackgroundWorkerHandle *handle;

  if (IsBackgroundWorker) return;

  MemSet(&worker, 0, sizeof(BackgroundWorker));
  worker.bgw_flags = BGWORKER_SHMEM_ACCESS | BGWORKER_BACKEND_DATABASE_CONNECTION;
  worker.bgw_start_time = BgWorkerStart_RecoveryFinished;
  snprintf(worker.bgw_name, BGW_MAXLEN, "pg_onnx");
  snprintf(worker.bgw_type, BGW_MAXLEN, "pg_onnx");
  snprintf(worker.bgw_library_name, BGW_MAXLEN, "pg_onnx");
  snprintf(worker.bgw_function_name, BGW_MAXLEN, "worker_main");
  worker.bgw_restart_time = BGW_NEVER_RESTART;
  worker.bgw_notify_pid = MyProcPid;

  if (!RegisterDynamicBackgroundWorker(&worker, &handle))
    elog(ERROR, "could not start background process");

  pid_t pid;
  BgwHandleStatus status = WaitForBackgroundWorkerStartup(handle, &pid);
  if (status == BGWH_STOPPED || status == BGWH_POSTMASTER_DIED)
    elog(ERROR, "could not start background process");

  Assert(status == BGWH_STARTED);
}
```

https://github.com/kibae/pg_onnx/blob/main/pg_onnx/pg_background_worker.cpp#L40

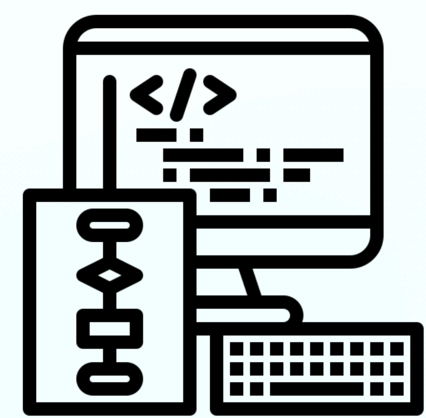
Background Worker

프로세스 생성하기

- RegisterBackgroundWorker
  - 설정 파일에서 shared_preload_libraries 를 통해 라이브러리가 로딩될 때만 사용할 수 있음
- SPI (Server Programming Interface)
  - postgres 프로세스에서의 extension과는 다르게 SPI를 이용한 DB 접근이 불가능함
  - 일부 카탈로그 데이터에만 readonly로 접근 가능
    - *BackgroundWorkerInitializeConnection(char *dbname, char *username, uint32 flags)* 를 실행하면 된다고 문서에는 나와있지만 실제로 쿼리 실행 시 실패함
    - https://www.postgresql.org/docs/current/bgworker.html
    - (성공하신 분 계시면 연락 좀...)

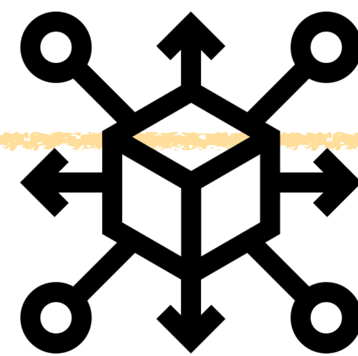
Extension 개발할 때 Test는?

Extension 개발할 때 Test는?

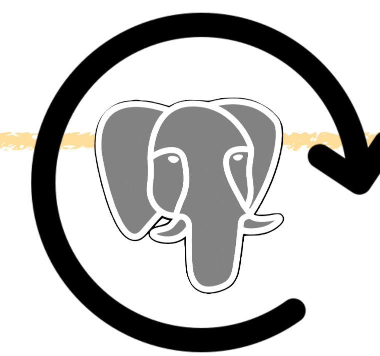


코딩

컴파일



배포/설치



PostgreSQL
재실행

SQL 실행
결과 확인

Test 자동화

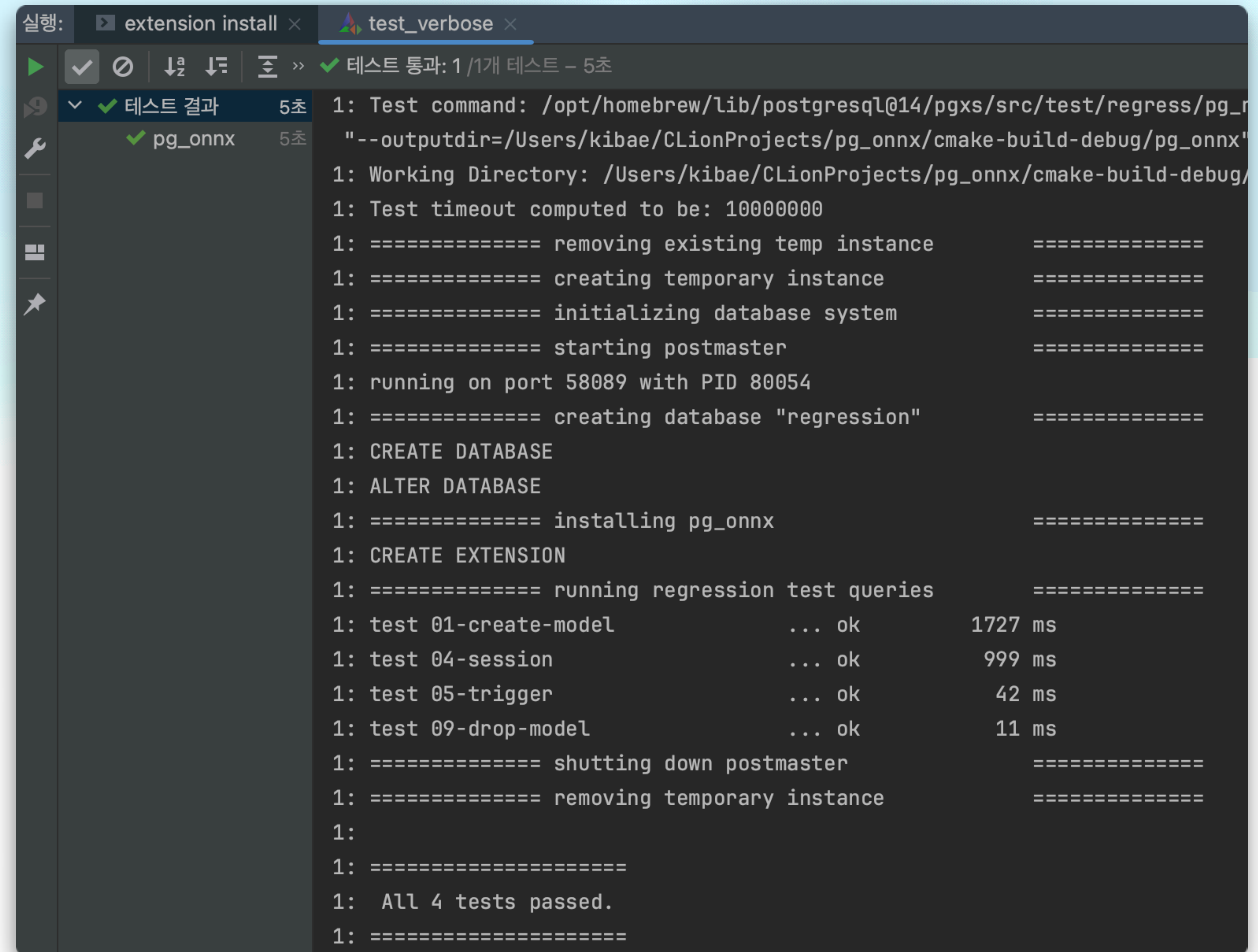
onnxruntime-server

- CMake의 ctest + GoogleTest
  - Unit Test
  - E2E Test
- GitHub Actions로 Build+Test 자동화
  - 예제

Test 자동화

pg_onnx

- CMake의 ctest + **pg_regress**
  - 격리된 PostgreSQL 데몬을 실행
  - 필요한 경우 특정 extension 자동 로드
  - 입력 SQL 파일들을 실행한 후 출력된 값들이 예측과 동일한 지 비교
- GitHub Actions로 Build+Test 자동화
  - 예제



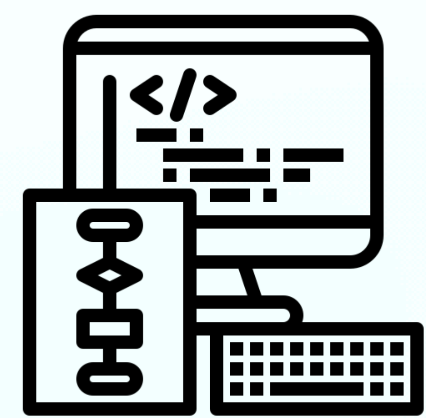
pg_regress

Regression Tests

- Regression test authoring
- Options *pg_regress --help*
  - **--temp-instance**
    - 격리된 PostgreSQL이 사용할 데이터 디렉토리 경로(PG_DATA)
  - **--inputdir** 기본값: 현재 경로
    - 입력할 SQL과 결과가 담긴 파일들이 위치한 경로
    - sql(입력), expected(예측) 하위 디렉토리가 필요
  - **--load-extension** (optional)
    - 격리된 PostgreSQL을 실행한 후 로드할 extension
- 주의할 점
  - 출력 예측 데이터에 현재 시간이나 랜덤값 같은 것들이 포함되면 SQL 실행할 때마다 결과가 달라지기 때문에 테스트는 항상 실패하게 됨

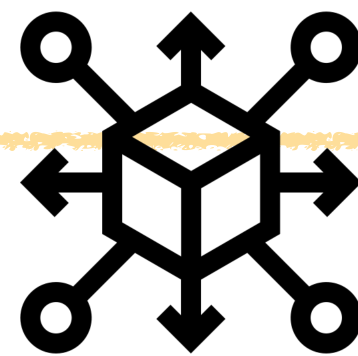
```
$ ls sql/ expected/
sql/:
test_name_1.sql   test_name_2.sql

expected/:
test_name_1.out   test_name_2.out


$ pg_regress \
  --temp-instance=/tmp/regress_test \
  --inputdir=/your/input_test_files_path \
  --load-extension=pg_onnx \
  test_name_1 test_name_2
```

Extension 개발할 때 Test는?

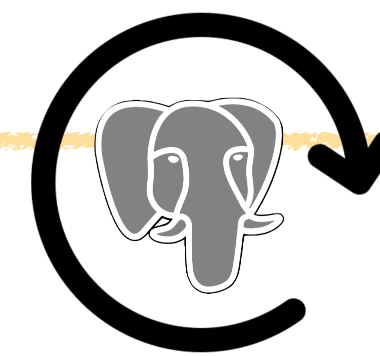


코딩

컴파일



배포/설치

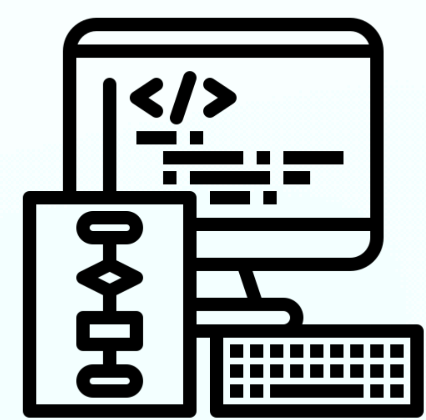


PostgreSQL
재실행

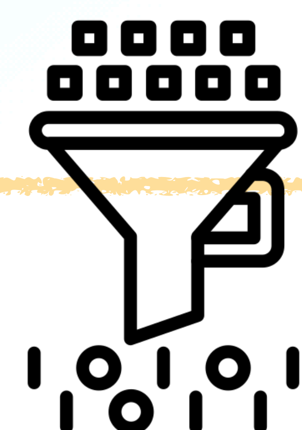
SQL 실행
결과 확인

Extension 개발할 때 Test는?

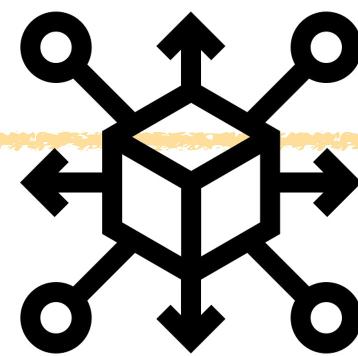
pg_regress 🤗



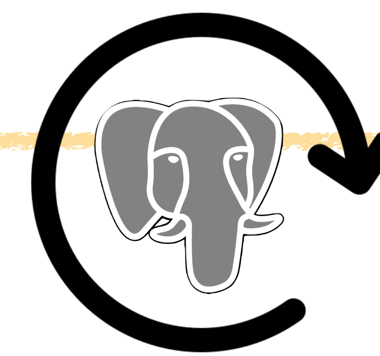
코딩



컴파일



배포/설치



pg_regress

SQL 실행
결과 확인

참고자료

PostgreSQL

- Server Programming Interface
- Background Worker Processes
- pg_regress
- PostgreSQL extension 개발 템플릿(CMake + pg_regress)

감사합니다

pg_onnx에 ⭐ 좀...