SQL Server Express LocalDB를 이용한 유저 개인 서버 구현

유영천

https://megayuchi.com

tw:@dgtman

미리 공지

- DB에 대해 잘 모릅니다. DB그 자체에 대한 질문하지 마세요.
- ODBC 사용해본 적 없으니 ODBC에 대한 질문하지 마세요.
- .NET 잘 모르니 .NET관련 질문하지 마세요.

퍼블릭 서버 (1인)운영의 어려움

- 서버 크래시가 발생하면 덤프 저장하고 이메일 보냄
- 365일 24시간 불안함
- 서버 비용의 압박 ㅠㅠ
- 스트레스에 비해 돈도 안된다.

유저들이 직접 서버를 운영하도록 하자

- 가장 큰 목적은 책임 회피
  - 온라인 게임인데 퍼블릭 서버를 닫으면 문제가 된다.
  - 어떤 방법으로든 지속적으로 게임을 플레이 가능하도록 해줘야 한다.

- 유저 서버 네트워크 형성 유도
  - ~~망한 게임에선 어차피 아무도 시도하지 않는다~~
- 의지가 있는 이들은 게임을 뜯어고쳐 보셈.
  - ~~망한 게임에선 어차피 아무도 시도하지 않는다~~

- 이하 Private서버라 부름.

Private 서버 배포의 장애물

- 서버 어플리케이션은 충분히 작다.
- 딱 한가지 종속성이 문제된다 - DBMS
- 게임 서버는 MS SQL Server를 사용한다.
  - 유저들에게 직접 MS SQL Server Express를 설치하라고 할까?
  - 퍽이나 잘 돌아가겠다…-,.-

SQL Server Express LocalDB

- SQL Server와 기능적으로는 거의 호환된다.
  - SSMS접속 가능.
  - 대부분의 쿼리 호환
- 법적으로 배포 가능하다.
- 기술적으로 배포 가능하다.
  - 사이즈가 작다.
  - 설치가 간편하다(DirectX Runtime수준).
- 용량/성능 제한이 있지만 문제되지 않는다.
  - 게임서버는 DB에 의존적이지 않다.
  - 몇 백 개의 계정을 유지하는데 1GB면 충분하다.

DB Access Library

- 네트워크 라이브러리처럼 간편하게 DB기능을 사용할 수 있는 라이브러리가 필요하다.
- 비동기 처리 가능
- 여러 개의 DB Connection을 만들어서 동시 다발적인 억세스 가능
- DLL기반 라이브러리(유지보수 용이)
- 게임서버를 비롯한 각종 툴에서 재사용
- 하부 DB 드라이버로 OLEDB 사용
- 수 년간 잘 사용해왔다.

OLEDB를 사용하게 된 이유

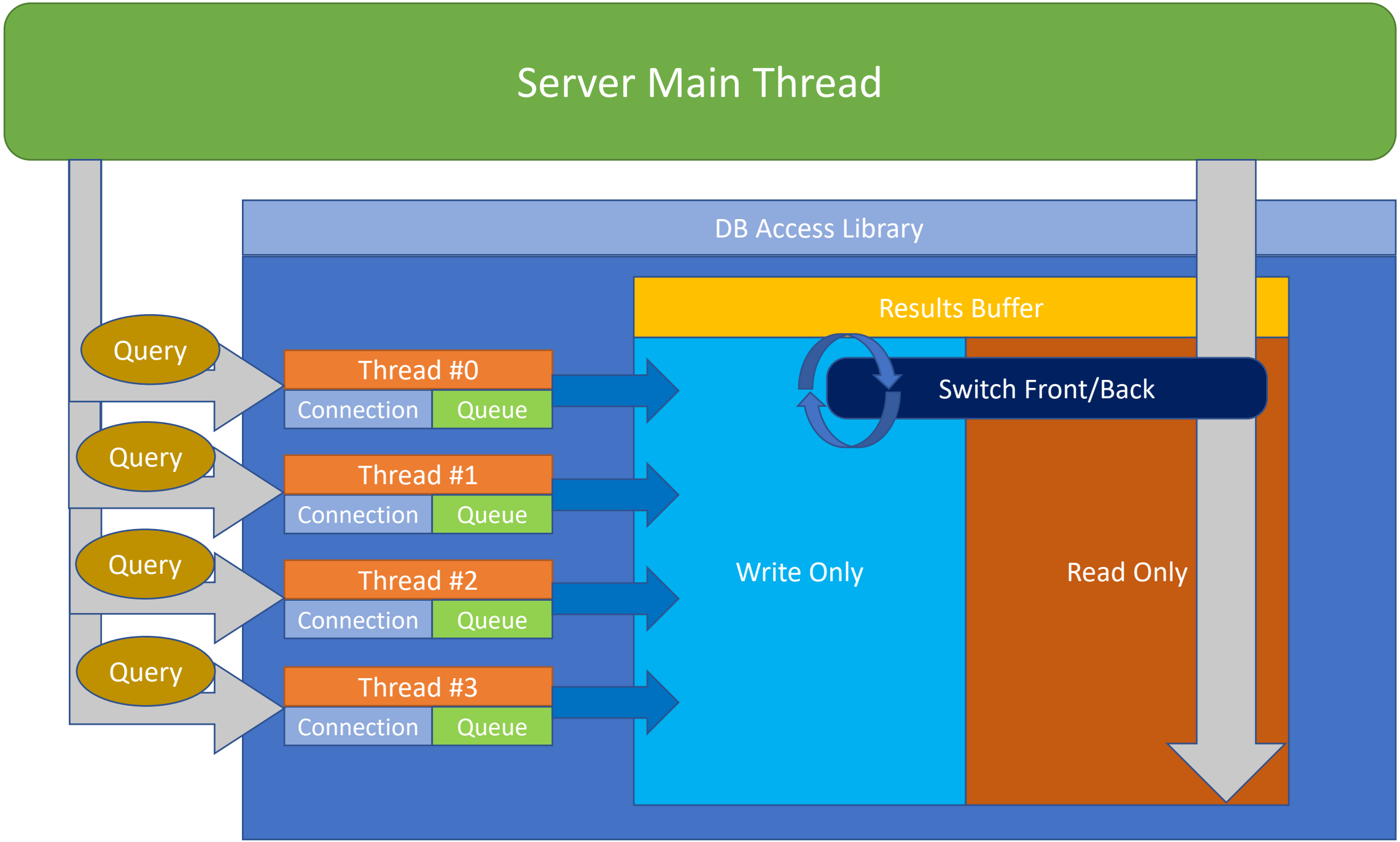
- 그 시절 트렌드였다.
- ODBC가 오래된 API다. OLEDB는 최신이었다. (최신은 좋고 오래된건 나쁘다는 뜻이 결코 아니다)
- ODBC는 시스템 설정을 따로 잡아줘야 한다. OLEDB에선 필요없다.
- OLEDB는 COM기반인데 DirectX로 단련(!)된 이 몸에게는 거부감이 없었다.
- 자료가 부족하여 애먹었지만 후회하고 있을 때는 이미 늦었다.

interface

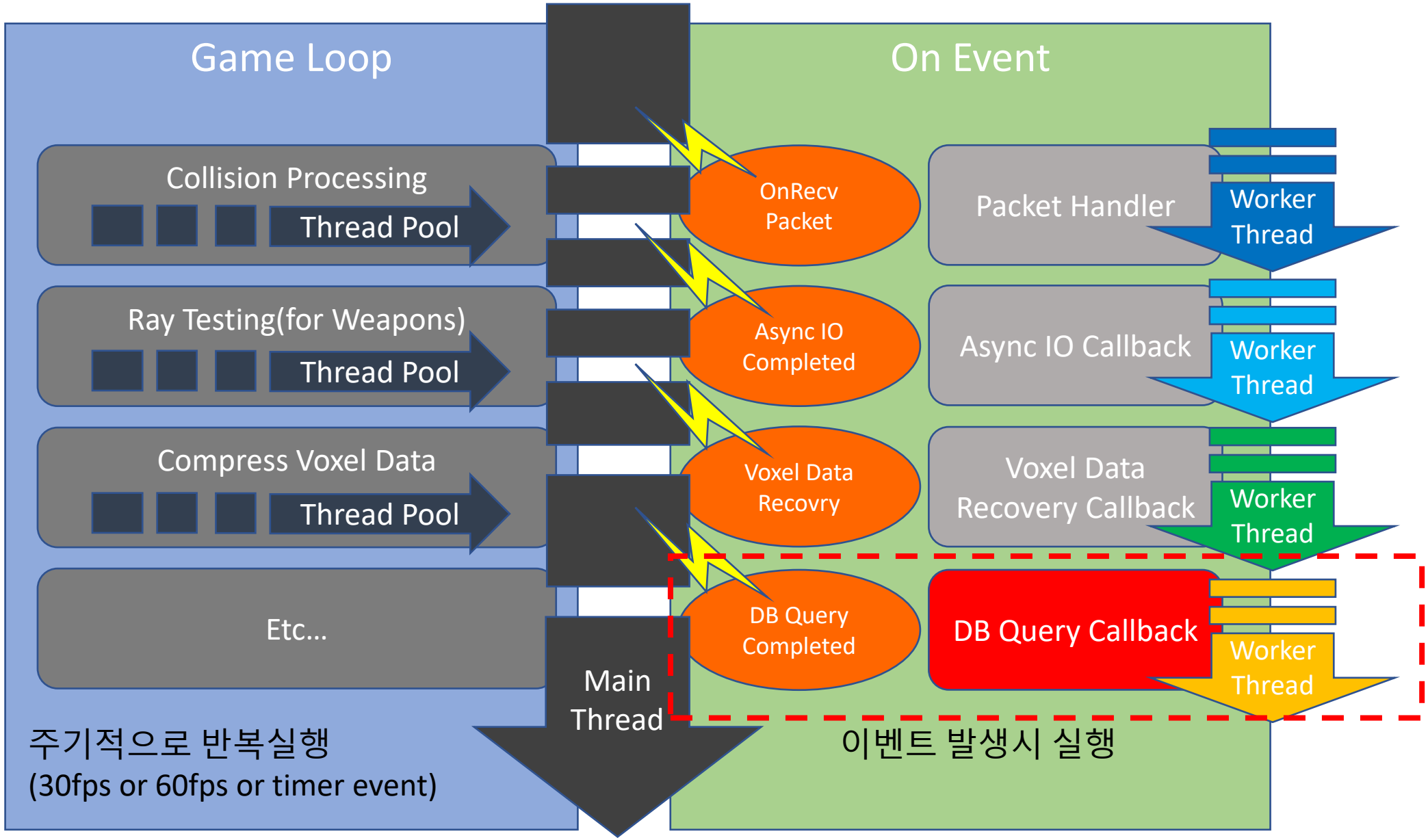
```
interface IMegayuchiDBAccess
{
    virtual BOOL     __stdcall          Initialize(const WCHAR* wchServerName, const WCHAR* wchDBName, const WCHAR* wchUserName, const WCHAR* wchPassword, DWORD dwMaxDBConnectionNum, DWORD
    virtual BOOL     __stdcall          Query(DB_CONNECTION_HANDLE hDBConnectionHandle, WCHAR* wchQuerySQL, int iQueryLen, char* pExtData, int iExtDataSize, OnQueryResultFunc pFunc, UINT64
    virtual BOOL     __stdcall          CallSP(DB_CONNECTION_HANDLE hDBConnectionHandle, SP_PARAM_BUFFER* pParamBuffer, int iParamBufferSize, char* pExtData, int iExtDataSize, OnQueryResul
    virtual BOOL     __stdcall          InsertRow(DB_CONNECTION_HANDLE hDBConnectionHandle, WCHAR* wchTableName, int iTableNameLen, char* pRowData, int iRowDataSize, int* piSkipColumnIndex
    virtual BOOL     __stdcall          UpdateRow(DB_CONNECTION_HANDLE hDBConnectionHandle, WCHAR* wchSelectQuery, int iQueryLen, char* pRowData, int iRowDataSize, int* piUpdateColumnIndex
    virtual BOOL     __stdcall          DeleteRow(DB_CONNECTION_HANDLE hDBConnectionHandle, WCHAR* wchSelectQuery, int iQueryLen, char* pExtData, int iExtDataSize, OnQueryResultFunc pFunc,

    virtual BOOL     __stdcall          Query_BlockMode(char* pOutResultBuffer, int iResultBufferSize, int* piSizePerRow, int* piRowNum, WCHAR* wchQuerySQL, BOOL* pbInsufficient) = 0;
    virtual BOOL     __stdcall          CallSP_BlockMode(char* pOutResultBuffer, int iMaxBufferSize, int* piSizePerRow, int* piResultRowNum, SP_PARAM_BUFFER* pParamBuffer, BOOL* pbInsuffic
    virtual BOOL     __stdcall          InsertRow_BlockMode(WCHAR* wchTableName, char* pOutNewRow, int* piOutRowSize, char* pRow, int iRowSize, int* piSkipColumnIndexList, int iSkipColumnN
    virtual BOOL     __stdcall          UpdateRow_BlockMode(int* piResultRowNum, char* pRow, int iRowSize, int* piUpdateColumnIndexList, int iUpdateColumnNum, WCHAR* wchSelectQuery) = 0;
    virtual BOOL     __stdcall          DeleteRow_BlockMode(int* piResultRowNum, WCHAR* wchSelectQuery, int iQueryLen) = 0;

    virtual DWORD    __stdcall          ProcessQueryResult() = 0;
    virtual void     __stdcall          Release() = 0;
    virtual DB_CONNECTION_HANDLE     __stdcall              GetDBConnectionRot() = 0;
    virtual void      __stdcall         StopProcess() = 0;
};
```

기본 구현

- CPU core개수만큼의 스레드와 DB Connection 준비
- 유저 접속-인증이 완료되면 유저 connection에 대해 N개의 DB Connection중 하나를 배정
- DB Connection 당 스레드 하나
- 이후 이 유저의 요청에 관련된 쿼리는 이 connection만을 사용한다-순서보장됨.
- 쿼리가 완료되면 결과 버퍼에 써넣고 게임서버(DB Library입장에선 클라이언트)의 메인스레드에 통지.
- 게임서버의 메인스레드는 DB Access Library의 결과버퍼의 front/back을 스위칭하고 결과버퍼를 읽는다


Server Main Thread
DB Access Library
Results Buffer
Switch Front/Back
Query
Query
Query
Query
Thread #0
Connection
Queue
Thread #1
Connection
Queue
Thread #2
Connection
Queue
Thread #3
Connection
Queue
Write Only
Read Only

서버의 작업 스케쥴링


Game Loop
On Event
Collision Processing
Thread Pool
Ray Testing(for Weapons)
Thread Pool
Compress Voxel Data
Thread Pool
Etc...
Main Thread
OnRecv Packet
Async IO Completed
Voxel Data Recovry
DB Query Completed
Packet Handler
Async IO Callback
Voxel Data Recovery Callback
DB Query Callback
Worker Thread
Worker Thread
Worker Thread
Worker Thread
주기적으로 반복실행
(30fps or 60fps or timer event)
이벤트 발생시 실행

DB Access Library for LocalDB

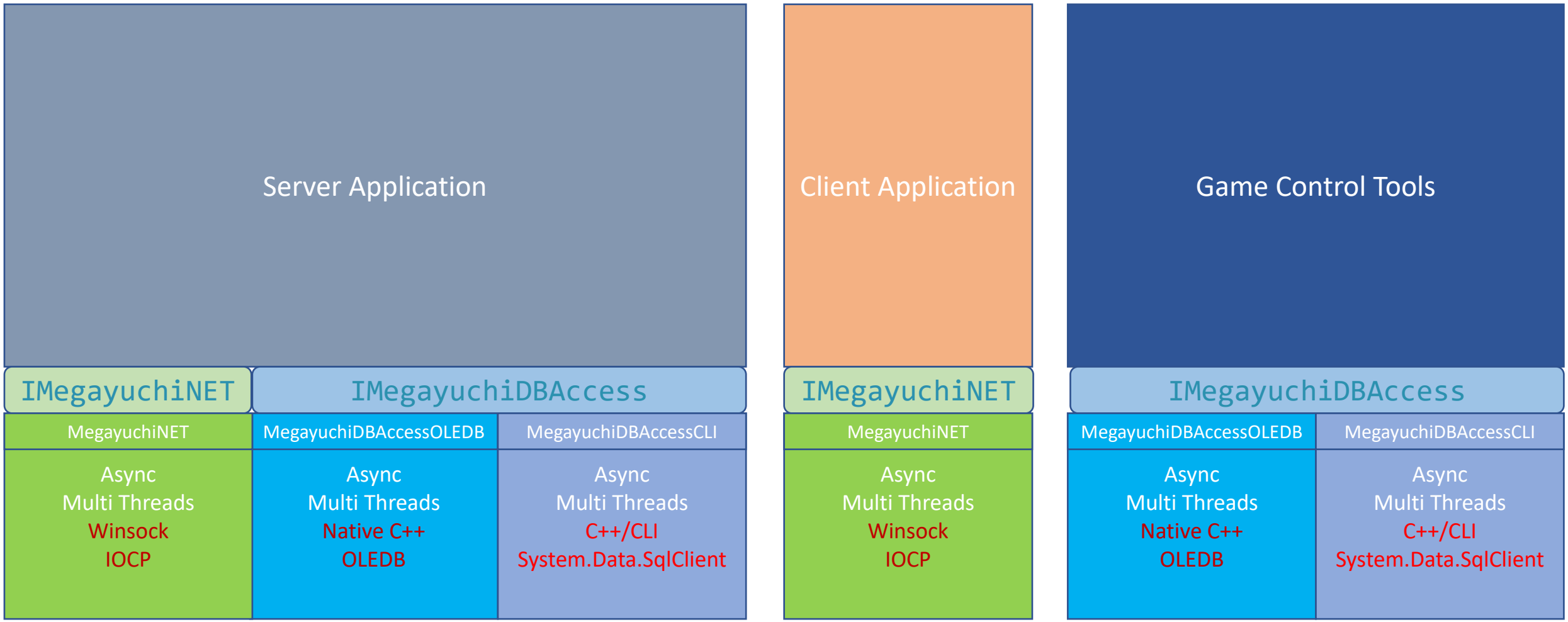
- OLEDB/C++ 코드로 LocalDB연결이 안된다.
- .NET OLEDB는 된다고 하는데 Native C++로는 어떻게 해도 안된다.
- 내가 뭘 잘못한건지도 모르겠지만 일단 자료가 없으니 확인할 길이 없음. OLEDB/C++로 LocalDB 연결하는 예제 한번도 못 봄.
- OLEDB/C++로 LocalDB연결에 성공하신 분들은 연락 바랍니다_._
- 시간은 흘러만 가네. 그래서 기존 DB Access Library와 똑.같.은 인터페이스를 노출하는 DLL 라이브러리를 추가로 만들기로 했다.

C++/CLI

- 과거 Managed C++이란 이름이었다.
- C# 대신 C++로 .NET을 사용할 수 있다.
- Mixed 모드 디버깅으로 manage와 unmanaged 코드를 넘나들 수 있으나 뒤지게 느리다(현재는 상당히 개선된 것으로 보임).
- C++프로그래머에게 있어 Native C++과 .NET의 interop의 가장 좋은 선택이다.
- C++/CX과 묘하게 비슷하다(C++/CX만들때 C++/CLI를 많이 참고한듯).
- C++/CLI를 사용하기로 했다면 DLL로 격리하기를 강력하게 권장한다.(디버깅/빌드 이슈를 최대한 피할 수 있다)

C++/CLI를 이용한 .NET의 System.Data.SqlClient 사용

- SqlConnection 클래스 (System.Data.SqlClient) | Microsoft Docs


Server Application
Client Application
Game Control Tools
IMegayuchiNET
IMegayuchiDBAccess
IMegayuchiNET
IMegayuchiDBAccess
MegayuchiNET
MegayuchiDBAccessOLEDB
MegayuchiDBAccessCLI
MegayuchiNET
MegayuchiDBAccessOLEDB
MegayuchiDBAccessCLI
Async
Multi Threads
Winsock
IOCP
Async
Multi Threads
Native C++
OLEDB
Async
Multi Threads
C++/CLI
System.Data.SqlClient
Async
Multi Threads
Winsock
IOCP
Async
Multi Threads
Native C++
OLEDB
Async
Multi Threads
C++/CLI
System.Data.SqlClient

Insert

Update

Delete

Stored Procedure

배포

- Steam의 경우 launch메뉴를 여러 개 설정할 수 있다.
- 2번째 메뉴는 Private Server 시작
- 서버 시작시
  - LocalDB인스턴스가 존재하는가?
  - 없으면 Local DB설치
  - 함께 배포한 DB Backup파일로부터 DB(VOXEL_HORIZON) 복구

DB 설정 테이블 변경시 업데이트

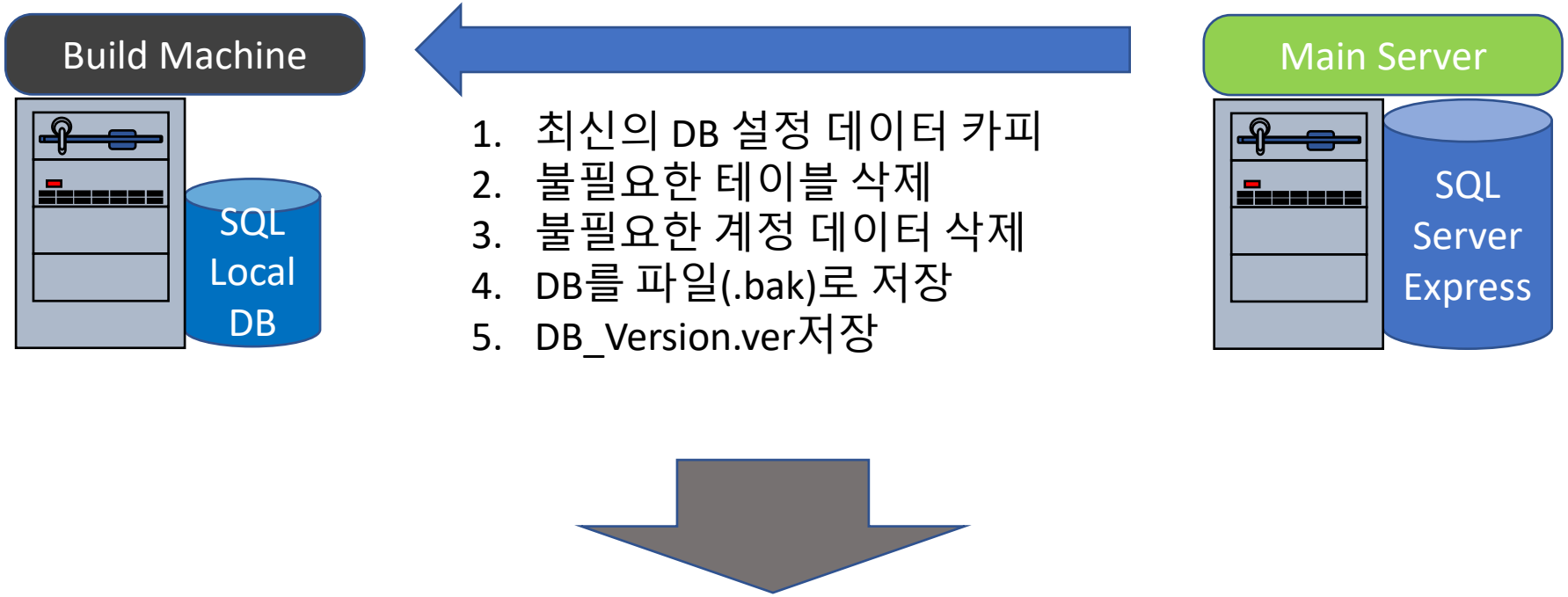
- 메인 DB의 버전을 업데이트

- 빌드머신(localdb설치된)에서
- 메인 DB로부터 테이블 디자인과 DB버전을 갱신
- 배포판에 적합하지 않은 테이블과 계정 삭제
- DB를Backup파일로 저장
- 메인 DB로부터 얻은 버전정보를 DB_Version.ver로 저장
- 배포판 빌드
  - 배포판에 backup된 DB파일과 DB_Version.ver파일을 포함
- 배포

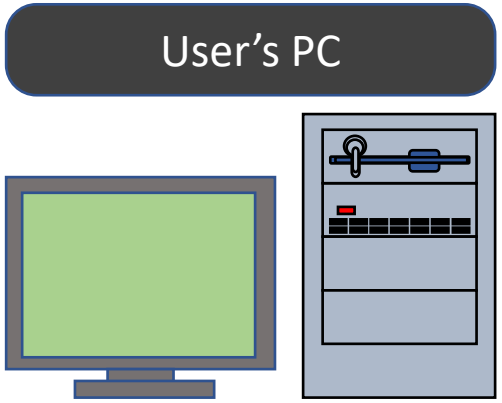
유저 PC에서 업데이트 체크

- Local DB인스턴스가 존재하는가?
  - 없으면 Install SQL Local DB
  - Backup본으로부터 기본 DB복구
- 서버 실행시 DB version파일과 DB로부터 얻은 version정보를 비교.
- Version정보가 다르면
  - Bakcup파일로부터 VOXEL_HORIZON_UPDATE로 복구
  - VOXEL_HORIZON_UPDATE -> 기존 VOXEL_HORIZON으로 게임 설정 테이블들을 덮어씀.

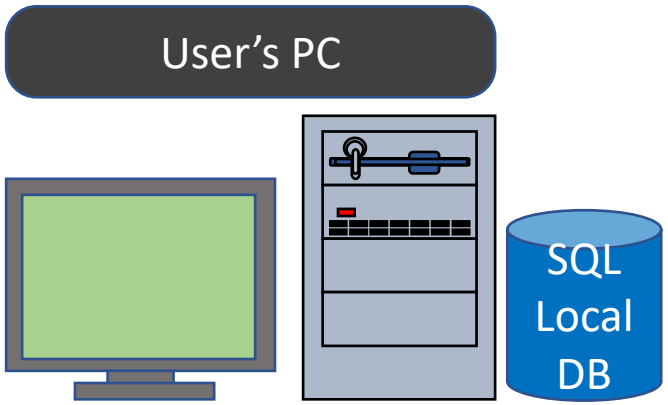
개발실에서


Build Machine
SQL
Local
DB
1. 최신의 DB 설정 데이터 카피
2. 불필요한 테이블 삭제
3. 불필요한 계정 데이터 삭제
4. DB를 파일(.bak)로 저장
5. DB_Version.ver저장
Main Server
SQL
Server
Express
Build Machine
SQL
Local
DB
1. 프로젝트 전체를 빌드
2. Steamworks 폴더로 카피
3. Stemaworks 업로드

유저 장비에서



1. 스팀에서 게임 다운로드
2. Private Server 실행
3. Local DB인스턴스가 존재하는가? - 처음엔 당연히 존재하지 않는다.
    1. Local DB 설치
    2. DB Backup파일로부터 DB복구
4. 서버 실행 단계로~



1. 패키지의 DB version파일과 DB로부터 얻은 version정보를 비교.
2. Version정보가 다르면
    1. 패키지의 DB Backup파일로부터 VOXEL_HORIZON_UPDATE이름으로 DB를 복구
    2. VOXEL_HORIZON_UPDATE(DB) -> 기존 VOXEL_HORIZON(DB)으로 게임 설정 테이블들을 덮어쓴다(DB Version정보 포함).
3. 서버 시작

결론

- 별 문제 없이 잘 도는 것 같은데 게임이 망해서 유저들이 잘 사용하고 있는지 확인 불가.
- OLEDB/C++로 SQL Local DB에 연결 성공하신 분 계시면 연락 바랍니다.
- OLEDB/C++로 SQL Local DB에 연결하는 코드 샘플을 보신 분 계시면 연락 바랍니다.