JAK VE VISUAL FOXPRO ZACHÁZET S TRANSAKCEMI SQL SERVERU

(IGOR VÍT NA TECHED/DEVCON 2010)

PŘÍPRAVA PROSTŘEDÍ K POUŽITÝM PŘÍKLADŮM

- **Visual FoxPro 9.0** používá **Cliserv.dbc** a z něho tabulku **Evidence.DBF** a vzdálený pohled **vwEvidence**
- **SQL Server 2008** má připojenou databázi **DbSk** (restore ze souboru DbSk.bak) a používají se **skripty .sql**

EXPLICITNÍ ZÁMEK VFP

Visual FoxPro spustit 2x (obě SET EXCL OFF):

- v obou instancích otevřít Evidenci do BROWSE LAST
- v první instanci editovat údaj, ale neopouštět políčko
- ve druhé instanci editovat políčko (objeví se hlášení pokus o uzamknutí)
- v první instanci přejít na jiný řádek – zámek věty se uvolní a 2. instance operaci dokončí

Závěr: druhý zápis v pořadí zvítězil.

OBVYKLÝ POSTUP V SQL PRO STEJNĚ ZADANOU ÚLOHU

Query txUpdate

V databázi DbSk na SQL Serveru máme stejnou tabulku jako byla v souboru DBF

```
BEGIN TRAN
UPDATE Evidence SET Firma = 'Změna X' WHERE CisZ = '14021'
```

druhý se otevře Query Update:

```
UPDATE Evidence SET Firma = 'Změna Y' WHERE CisZ = '14021'
```

zůstane čekat, dokud se v prvním Query nevyvolá:

```
COMMIT
```

ve třetím Query Select:

```
SELECT * FROM Evidence
```

Závěr: druhý zápis v pořadí zvítězil

JAK SLEDOVAT ZÁMKY NA SQL SERVERU

---

otevřít Query ListLocks:

sys.dm_tran_locks

Zavřít všechna okna ostatní Query (aby ubyly nepotřebné zámky z vypisovaného seznamu) a volat

```
sp_lock
```

Nějaká DB je otevřena jako výchozí – seznam bude mít zámek i na ní.

DB **1** je master a DB **5** je DbSk (DB **2** je tempdb)

Obě mají už teď zámek typu S: „sdílený zámek“ – znamená „s databází se pracuje, jsem ochoten přístup s někým sdílet, ale dávám zámkem **S** najevo, že libovolně s ní ostatní přístupy nakládat nesmí – třeba smazat ji“

otevřít Query txUpdate:

(samotné otevření dalšího spojení generuje pár nových zámků S)

```
BEGIN TRAN  (žádný zámek nepřibude)
UPDATE Evidence SET Firma = 'Změna W' WHERE CisZ = '14021'
```

výstup z

```
sp_lock
```

na *řádku* (RID) se objevil zámek **X** (exklusivní), ale zároveň zamýšlené zámky (IX) na nadřízených částech řádku – *stránce* a *tabulce* – tím se hlásí ostatním „nedovolím vám teď provádět úplně cokoliv s částmi, na kterých leží řádek“

otevřít Query Update a spustit

```
UPDATE Evidence SET Firma = 'Změna Q' WHERE CisZ = '14021'
```

Přibyly další zámky:

U – update (ten ale čeká – WAIT)

IX na stránce a tabulce (ty jsou povoleny, protože nejsou v konfliktu s ostatními IX – zatím je oba procesy mají jen v plánu)

otevřít Query Select

```
SELECT * FROM Evidence
```

a znovu spustit:

```
sp_lock
```

Přibyly zámky: pokus o sdílený zámek **S** na *řádku* – čeká. Ostatní jsou **I** a ty jsou povoleny.

V **Query txUpdate** dokončit transakci a ověřit zámky:

Zůstaly jen zámky **S** nad databází z dosud spuštěných spojení.

Systém zamykání = jde o tzv. kompatibilitu zámků

nejsou-li v kolizi, jsou slučitelné, kompatibilní

ROZDÍLY TRANSAKCÍ VFP A SQL

Aktualizaci přes BROWSE, ale tentokrát se spuštěnou transakcí:

- v obou instancích otevřený v Browse evidence.dbf
- v první instanci

```
BEGIN TRAN
```

- oprava údaje a přeskok jinam (sloupec/řádek)
- ve druhé instanci je vidět, že se neaktualizuje
- v první instanci přeskočit na jiný řádek
- ve druhé instanci se nadále neaktualizuje (na rozdíl od úvodní ukázky)
- v první instanci

```
END TRAN
```

- ve druhé instanci se projeví změna

Závěr: příkazy na ukončení transakce se v SQL a VFP liší

SROVNÁME VNOŘOVÁNÍ

Účinek a chování vnoření **VFP**:

```
BEGIN TRANSACTION
? TXNLEVEL()
```

- změnit obsah údaje firma

```
BEGIN TRANSACTION
? TXNLEVEL()
```

- změnit obsah údaje kontakt

```
ROLLBACK (vnitřní tx)
? TXNLEVEL() -> 1
```

- vrátí se kontakt

```
END TRANSACTION
? TXNLEVEL() -> 0
```

- operace byla potvrzena

(Je možné přepnout do **2. instance** na potvrzení, že ostatní uživatelé vidí **stejný výsledek**.)

(Pozn.: i opačné pořadí – nejprve END TRAN a pak ROLLBACK ukáže, že vnější tx vítězí – tedy „anuluje“ END TRAN z vnitřní; v tom se podle dostupných informací chová VFP jako Oracle http://www.oracle.com/technology/documentation/berkeley-db/db/gsg_txn/C/nestedtxn.html)

SQL Server dle Query txNested:

```
SELECT *  FROM Evidence
BEGIN TRAN
PRINT @@TRANCOUNT
UPDATE Evidence SET Firma = 'tx LVL 1' WHERE CisZ = '14021'
SELECT *  FROM Evidence

BEGIN TRAN
PRINT @@TRANCOUNT
UPDATE Evidence SET Kontakt = 'tx LVL 2' WHERE CisZ = '14021'
SELECT *  FROM Evidence
ROLLBACK
SELECT *  FROM Evidence -> anulují se operace obou transakcí
PRINT @@TRANCOUNT -> 0
```

SROVNÁME TECHNOLOGII ZÁPISU OPERACÍ V TRANSAKCI

ve Visual FoxPro:

- V *první* instanci: Zapsat do Firmy hodnotu **0**, přejít na jiný řádek; po krátkém čase i *druhá* instance novou hodnotu vidí
- V *první* instanci:

```
BEGIN TRAN
```

- Zapsat do Firmy „Zápis“, pocestovat ukazatelem na jiný řádek (bylo zapsáno, zámky nejsou)
- Ve *druhé* instanci BROWSE (nebo např. přes „? firma“) vidíme, že nově zadaná hodnota zatím není k dispozici (=>> dá se usuzovat, že není na disku)
- V *první* transakci

```
END TRAN
```

Ve *druhé* instanci se nová hodnota projeví.

Závěr:

Během transakce zapisuje VFP do paměti *dané instance*. Druhá instance si nemá jak hodnoty načíst: na disku ještě nejsou a instance nemají žádnou sdílenou datovou paměť (natož aby měly nějakou ve víceuživatelském prostředí, kdy běží na různých strojích).

v SQL Serveru podobná úloha:

Query Select

```
SELECT…
```

Ověřit aktuální obsah

Query txUpdate

```
BEGIN TRAN
UPDATE … - s nějakou jinou hodnotou, než zobrazuje SELECT
```

Query Select

```
SELECT…
```

Nedokončí se, čeká…

Chceme-li načíst novou hodnotu i za cenu rizika, že po ROLLBACKu tam nezůstane, lze:

použít:

```
SELECT * FROM Evidence (NOLOCK)
```

nebo `SELECT * FROM Evidence (READUNCOMMITTED)`

Závěry:

VFP během transakce zapisuje do operační paměti stanice, kde běží.

SQL Server zapisuje také do paměti (datové paměti) – ale běží na serveru a tím pádem mají možnost vidět obsah i ostatní.

ROZDÍLY V TRANSAKCÍCH – IZOLACE

Oba případy z ukázek – SELECT …NOLOCK a READUNCOMMITTED jsou podle nápovědy odsouzeny k postupnému zániku. Jaké jsou tedy ty správné postupy k řešení stejných úloh? Je třeba využít příslušné nastavení izolovanosti transakcí.

Query txIsolations

Aktuální nastavení ověřit voláním

```
DBCC USEROPTION
```

Změnit na:

```
…READ UNCOMMITTED
```

Po příkazu `UPDATE` …

Ověřit co vrací `SELECT`

Pak volat `ROLLBACK`

Ověřit co vrací `SELECT`

Celkem jsou nastavení 4 tradiční a v 2005 nové SNAPSHOT. Krom dosavadních 2 si ještě jedno ukážeme v jiné úloze později. Podrobněji a bohatě je vysvětleno v Books Online k SQL Serveru.

VFP S DATY SQL

V DBC je připraven **vzdálený pohled** na tabulku Evidence, která je na SQL Serveru.

```
USE cliserv!vwevidence CONNSTRING "Driver=SQL Server Native Client
10.0;Server=(local);Database=DbSk;Trusted_Connection=yes"
```

Akualizovat obsah přes BROWSE a ověřit náhledem na SQL Server, jak se promítla změna.

Vlivem tzv. transakce typu „**autocommit**“ se jednotlivý příkaz Update vykonal v rámci transakce /včetně příslušných optimistických zámků.

Další nastavení, které má vliv na chování je jeden specifický parametr *spojení*:

Ve **VFP**:

```
lnHandle = CursorGetProp("ConnectHandle")
SQLSETPROP(lnHandle,"Transactions",2)
```

Transactions –Auto = 1, Manual = 2

- Aktualizovat údaje a přeskočit na jiný řádek

Na **SQL Serveru**:

`SELECT * FROM Evidence` -> musí tam být změna vidět (pokud zůstalo READ UNCOMMITTED, nebo to načítá Exec query…)

Ve **VFP**:

```
prom = 0
?SQLEXEC(1,"SET ?@prom = @@TRANCOUNT")
? prom --> má vrátit 1
```

To ovšem znamená, že běží transakce, aniž bychom volali BEGIN TRAN!

Transakci tedy dokončíme:

```
SQLCOMMIT(1)
```

Abychom ukončili režim ručních transakcí, zavoláme:

```
?SQLSETPROP(1,"Transactions",1)
```

a ověříme, že transakce neběží

```
?SQLEXEC(1,"SET ?@prom = @@TRANCOUNT")
? prom
```

Ostatně, co jsme si ukázali ve VFP, má obdobu na SQL Serveru. Tam lze také dosáhnout toho, že **příkazy pro manipulaci s daty** automaticky startují transakce – i bez explicitně uváděného BEGIN TRAN:

Na **SQL Serveru** dle:

Query txImplicit

ÚLOHA – KOORDINACE VFP TX SE SQL TX

V minulé ukázce jsme u pohledu napojeného na vzdálená data prošli pouze cestou s potvrzením transakce – COMMIT. Zkusme teď i ROLLBACK.

Ve **VFP**:

ohlídat, aby bylo jasné, na kterém spojení pohled otevíráme

nejlépe **pozavírat vše** - USE, případně SQLDISC(0)

```
USE cliserv!vwevidence
BROWSE LAST
?SQLSETPROP(1,"Transactions",2)
```

Změnit údaj a **ukázat na SQL Serveru**:

`SELECT * FROM Evidence` -> čeká se… nebo se ukáže nová zůstal-li READ UNCOMMIT

Ve **VFP**:

```
SQLROLLBACK(1)
```

na **SQL Serveru**:

`SELECT * FROM Evidence` -> zobrazí „původní hodnotu“ před aktualizací

Ve VFP ale zůstává být vidět ta nová hodnota, podobně jako po úspěšném volání **TABLEUPDATE**, považuje VFP situaci za vyřešenou, uzavřenou.

Možná řešení ve **VFP**:

`REQUERY() – s nevýhodou nového načítání dat z SQL serveru`

lepší řešení:

angažovat transakce VFP! Tedy:

Ve **VFP**:

```
BEGIN TRANSACTION
```

editovat údaj, přeskočit

Na **SQL Serveru**:

ukázat UNCOMMITED READ – SELECT – zobrazí novou hodnotu

Ve **VFP**:

```
? SQLROLLBACK(1)
```

SQL Server:

verze COMMITTED - SELECT – ukáže „původní data“ před tx

Ve **VFP**:

```
ROLLBACK
```

Vrátí se původní hodnoty před transakcí

ÚLOHA „ÚČETNÍ UZÁVĚRKA“

---

v **Query txIsolations**:

Použít ISOLATION LEVEL REPEATABLE READ

```
BEGIN TRAN
SELECT * FROM Evidence
```

Ukázat zámky – těch je víc jak 500: na každém řádku!

(Pozn.: lze vyřešit použitím SELECT * FROM Evidence WITH (TABLOCK), pokud je předem jasné, že je třeba pracovat se všemi řádky)

`UPDATE` odjinud neprojde

`COMMIT` … a `UPDATE` doběhne