

Haladó adattárház-technológiák

MSc

Kivonatos jegyzet (1. rész)

Készítette: Vassányi István, 2025

Kell-e az adattárház?

For years the Volvo Car Corporation collected a lot of data from the numerous digital sensors in each of their cars. Working alongside the devices that trigger signals like the Service Engine Soon light are hundreds of meters and data loggers. All these data points are collected from the car's central computer when you take it in for servicing and diagnostics.

Volvo identified that this data when analyzed alongside hardware, software and functional specifications, vehicle diagnostics, and warranty claims would allow them to reduce costs, improve the accuracy of warranty reimbursement, and improve the quality of their products.

To perform the intended analysis, Volvo first had to consolidate all their data sources into one location. Volvo teamed up with Teradata to implement a data warehouse.

As a result, the volume of actionable data at the disposal of Volvo analysts increased from 364 gigabytes to 1.7 terabytes. The data was made accessible to over 300 individuals across product design, manufacturing, quality assurance, and warranty departments. Volvo experienced immediate improvements in query performance, user access, targeted cost reduction, and an increase in accuracy through warranty claim analysis, as well as improvements in the quality of current and future production lines.

<http://assets.teradata.com/resourceCenter/downloads/CaseStudies/EB6286.pdf>

Adattárházak és OLTP rendszerek kapcsolata

Adattárház (AT, DW): Elemzési célra leválasztott és újrastrukturált adatok sok OLTP és egyéb forrásból
Line of business (LOB) alkalmazás = OLTP alkalmazás

„A DW is a centralized data silo for an enterprise that contains merged, cleansed, and historical data.”

Miért kell az adattárház?

ÖNÁLLÓ FELADAT #1: írjunk lekérdezést, amely a World Wide Importers (WWI) adatbázisban az áruházba eladott cikkek értékét adja vissza évenként és államonként.

Segítség (lásd az alábbi sémát):

- Áruházba történt az eladás, ha a Sales.CustomerCategories.
CustomerCategoryName='Supermarket'
- Egy rendelés értékének a megállapításához összegezni kell a rendeléshez (Sales.Orders tábla) tartozó tételek (Sales.OrderLines tábla rekordjai) értékét (Sales.OrderLines. Quantity és UnitPrice) mezők
- A rendelés évszáma a Sales.Orders tábla Orderdate mezőjéből vehető ki
- A rendelés vásárlója a Sales.Orders.CustomerID, mely mező a Sales.Customers táblára mutat

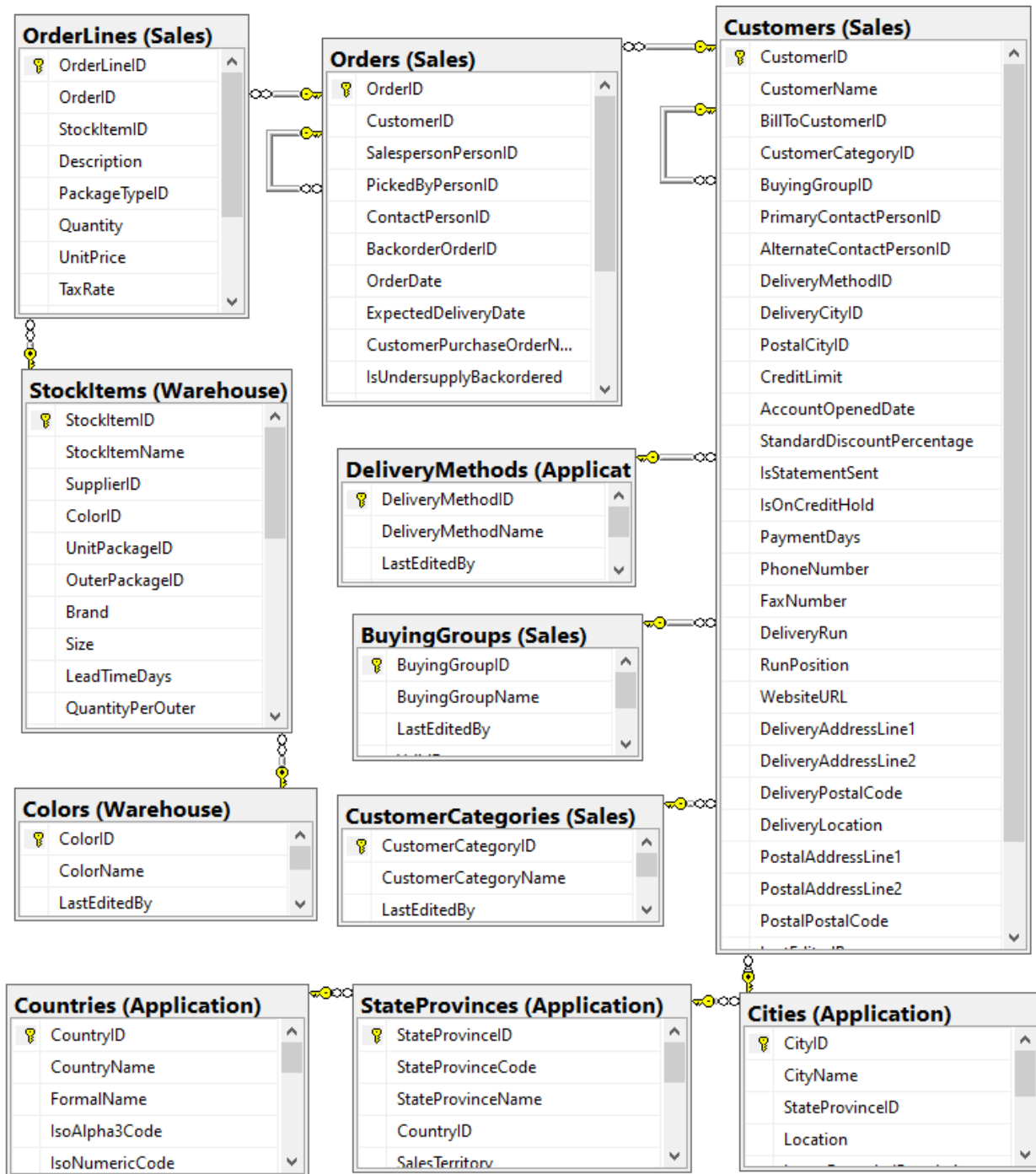
KÉRDÉS: 2013 és 2016 között melyik évben melyik állam hozta a legnagyobb forgalmat?

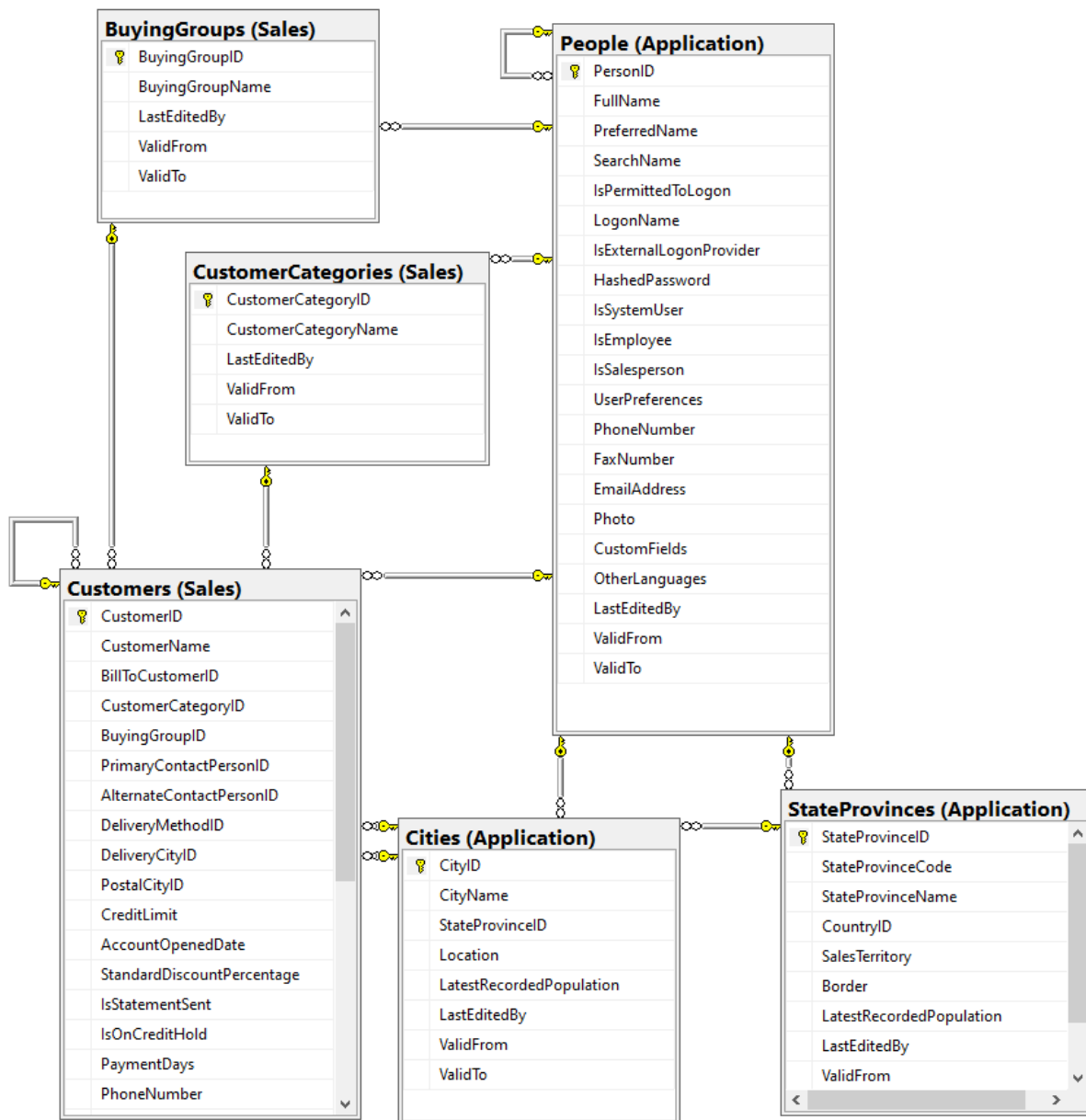
Ugye, hogy nem könnyű kideríteni...

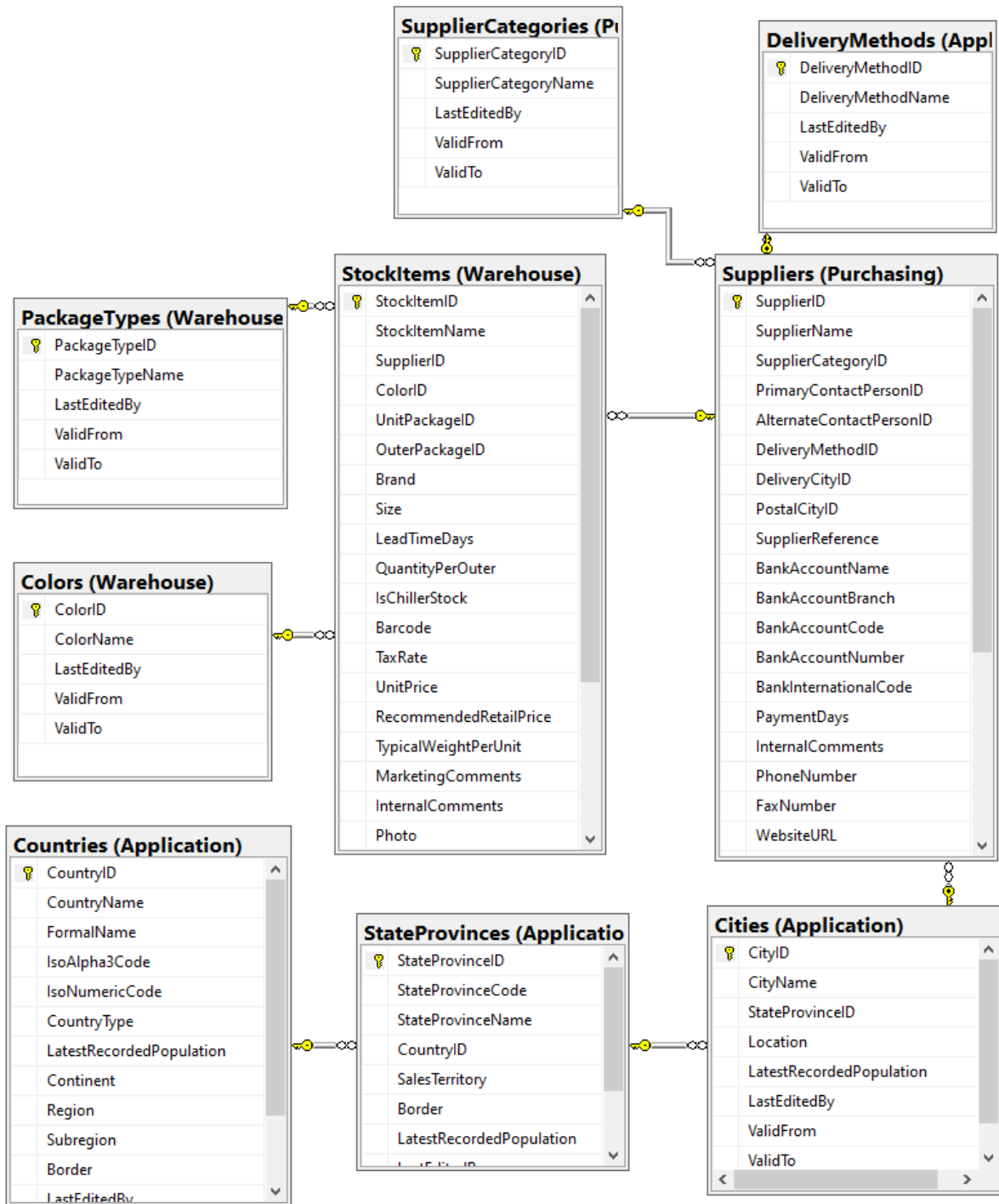
A WWI és az AdventureWorks demo adatbázisok letölthetők innen: <https://learn.microsoft.com/en-us/sql/samples/sql-samples-where-are?view=sql-server-ver16>

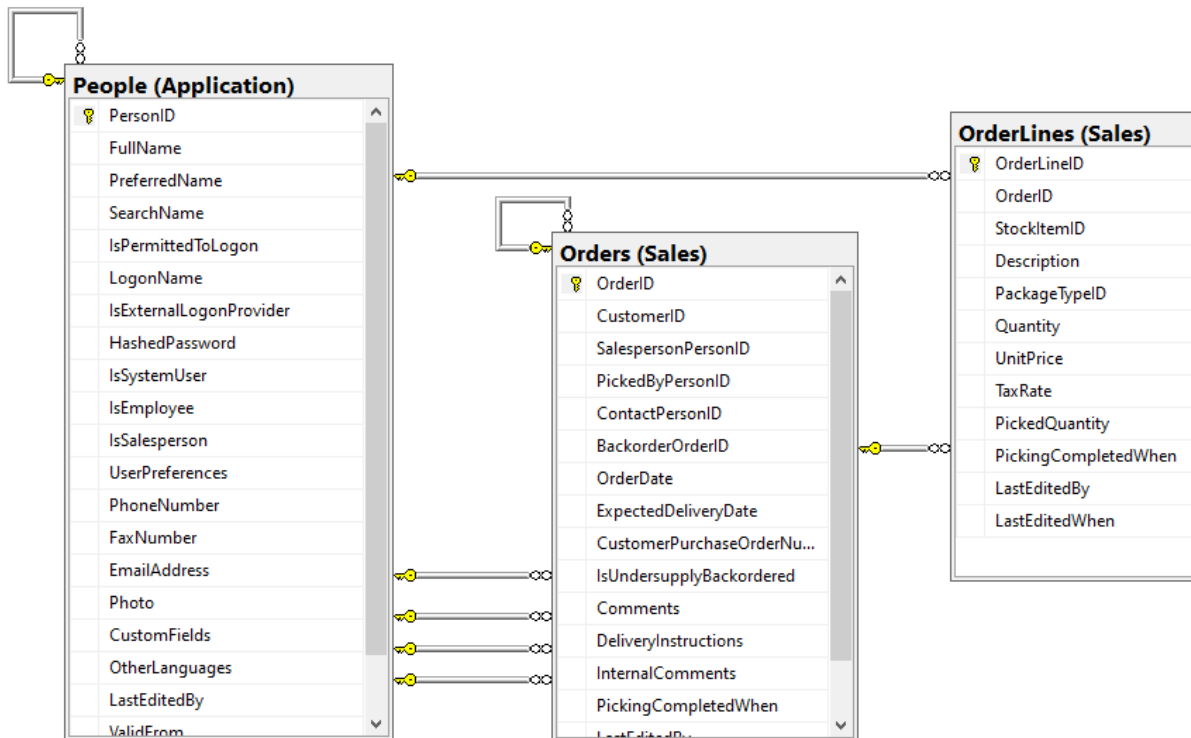
- <https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImporters-Full.bak>
- <https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImportersDW-Full.bak>

A WWI adatbázis számunkra érdekes táblái:

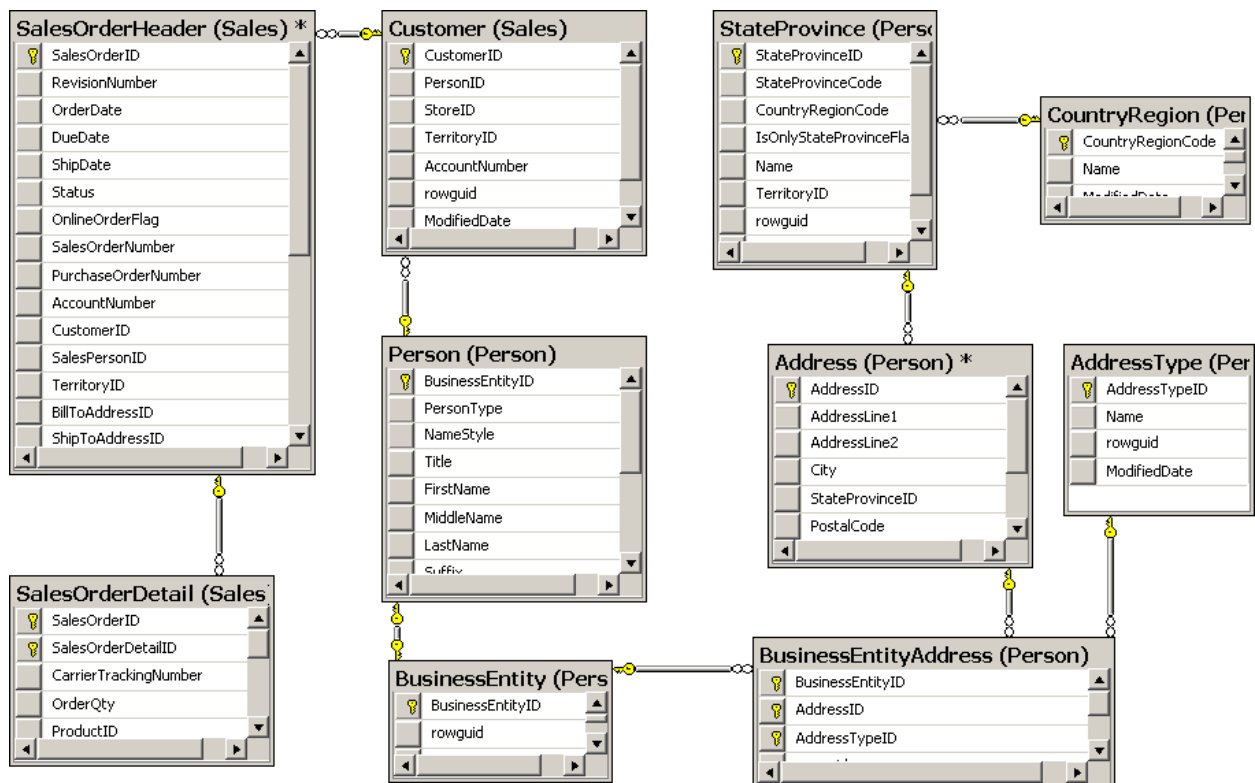








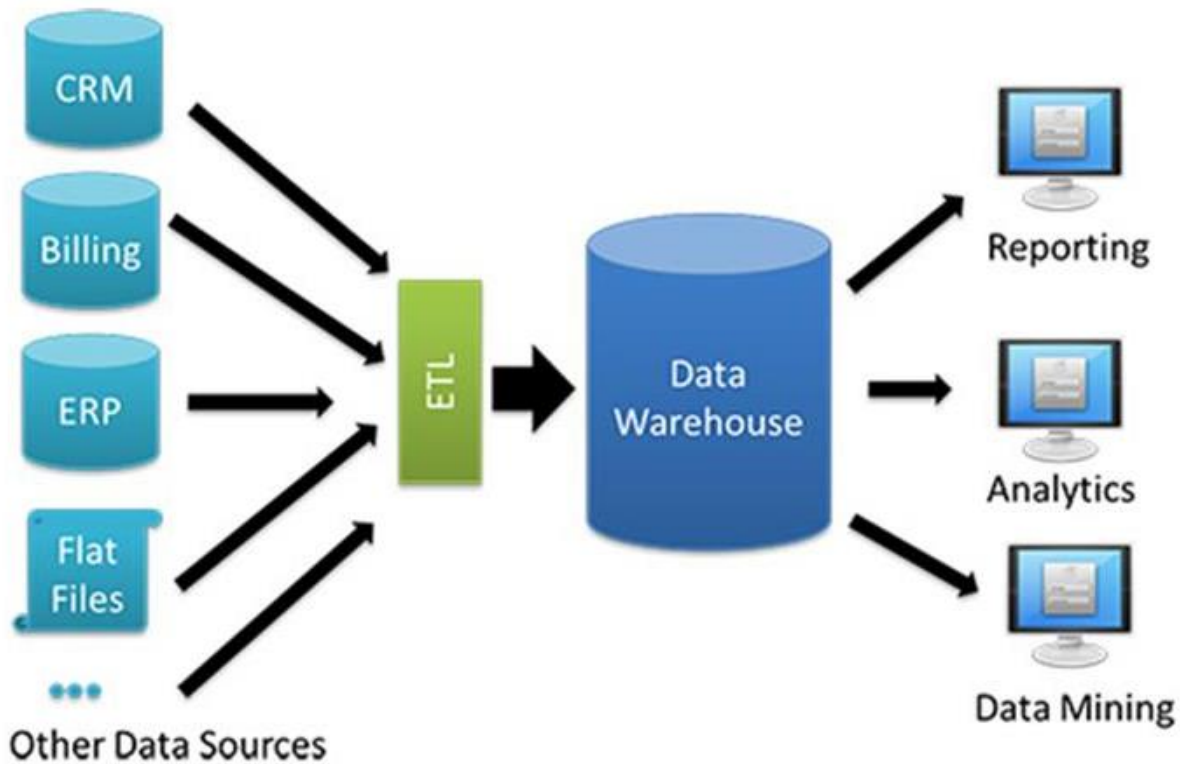
Az AdventureWorks2019 adatbázis lényeges táblái:



A tanulság az, hogy az OLTP adatbázison adatelemzést végezni nehéz. Ilyen problémák lépnek fel:

- Az OLTP adatbázisokban akár több ezer tábla is lehet, a hatékonyság és a flexibilitás céljából igen erősen normalizálva
- A mezőnevek nem mindig beszédesek. Segítség lehet a tranzakcionális középpont keresése.
- Sok nagy táblára JOIN -> lassú lekérdezés, lassítja az adatmódosító OLTP tranzakciók kiszolgálását
- Az OLTP adatbázis tipikusan csak a legfrissebb adatot tárolja, például a legutolsó címet. Ez hosszabb időszak alatt hibás kimutatást eredményezhet, például a fenti esetben akkor, ha más országba költözött a vevő.
- Több adatforrás is lehet, például más adatbázist használhat a könyvelés és a gyártás—ezek közt egyeztetni kell, hiányzó/ellentmondó adatok lehetnek...

A megoldás az adattárházba való átalakítás az ETL folyamat során (ETL=Extract-Transform-Load).



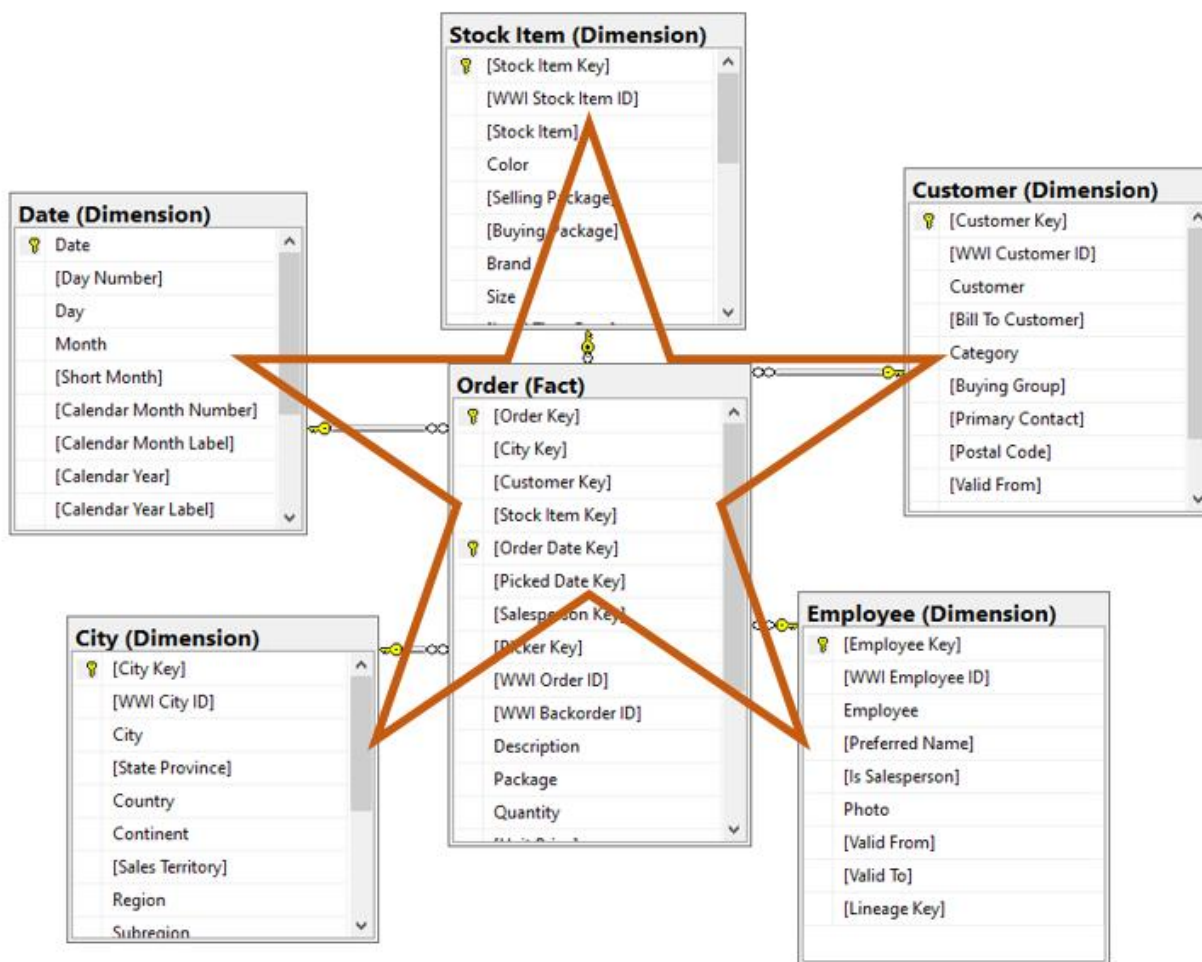
LOB/OLTP adatbázis	ADATTÁRHÁZ
CÉL: az üzleti folyamatok, tranzakciók támogatása -> előre megtervezett, alkalmazásba épített, optimalizált lekérdezések kiszolgálása	Döntéstámogatás -> az 'ad hoc' lekérdezések gyors kiszolgálása
FELHASZNÁLÓK: üzleti kliensek	Döntéshozók
IDŐABLAK: korlátozott (aktuális adatok, szűk időhorizonttal), méret korlátozás céljából	Archív, nagy időablakot átfogó adatok

STRUKTÚRA: 3 NF (normalizált)	A teljesítmény érdekében denormalizált (csillag vagy hópehely), lásd később
MÉRET: korlátozott méret a gyors működés érdekében	Nagy méret
ADATMÓDOSÍTÁS: folyamatosan a tranzakciók alapján	Nincs módosítás ¹ , csak rendszeresen ütemezett adatimport (például éjszakánként)
ADATÉRVÉNYESÉG: aktuális (létező legfrissebb) adatok	Az ETL ütemezésétől függően az adatok nem naprakészek, például éjszakai ETL esetén a tegnapi adatok a legfrissebbek

Adattárházak szerkezete

Szerkezet: a csillag és hópehely séma. Az előző lekérdezést a **csillag-séma** alapján egyszerűbben is meg lehetett volna írni:

¹ Az SCD1 és SCD3 esetet kivéve, lásd később



A csillag-séma tipikusan **denormalizált**. Emlékeztetőül: a 3NF-ben minden nem kulcs attribútum totálisan és közvetlenül (nem tranzitíven) függ az egész kulcstól. A denormalizálás tipikusan egy JOIN művelet. Például a fenti Customer dimenzió mezői a Customers és a CustomerCategories táblákból is származnak

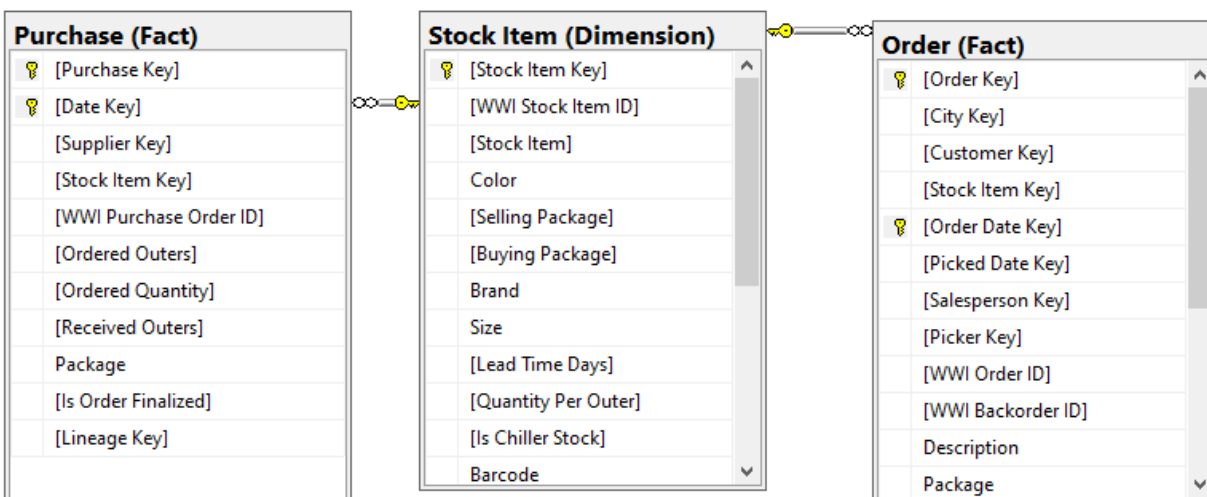
Egy csillag témája egy üzleti terület, például az eladás a fenti esetben. Egy adattárházban tipikusan több csillag is van.

A középen lévő tábla a **ténytábla**, ehhez külső kulcsokkal kapcsolódnak a **dimenzionális** táblák. Általában az adattárház is relációs adatbázis!

A **dimenziók** az esemény egy-egy **független jellemzőjét** írják le. Egy eseményt (itt: eladást) általában a külső kulcsok **együttvéve** azonosítanak. Ha nem, akkor a ténytáblának saját kulcsa van.

Egy csillag olyan, mint egy bővített mondat, melynek az állítmánya a ténytábla, a határozói és jelzői pedig a dimenziók. A dimenziók tehát **kontextust** adnak a ténynek.

Az adattárház csillagai a megosztott dimenziók révén vannak kapcsolatban egymással. Példa: az Order és a Purchase ténytáblák is használják a StockItem dimenziót:



Általában csillag-sémát használunk. Ha mégis valamelyik dimenzió legalább részben normalizált marad (tehát több táblát foglal el), akkor **hópehely-séma** keletkezik.

Megjegyzés: az OLTP adatbázisokra is a hólabda-szerkezet jellemző, de a törzstábláig vezető utak sokkal hosszabbak!! Nincs egyértelmű megfeleltetés az OLTP tranzakcionális középpontjai és az adattárházban lévő csillagok/hópehelyek között!

A **csillag-séma előnye** a hópehellyel szemben a gyorsabb lekérdezés és az átláthatóbb szerkezet, hátránya hogy nagyobb a tárigénye és több erőforrást igényel az elkészítése az OLTP adatbázisból. A gyakorlatban a hópehely-szerkezet *csak demo jellegű*, ún. Proof of concept (POC) projektekhez ajánlott.

Dimenziók tervezése

Először a dimenziókat tervezzük és valósítjuk meg, utána az ezeket használó ténytáblákat.

Mezőtípusok

1. **Kulcs-mező vagy kiegészítő kulcs-mező:** a dimenzió egy rekordját azonosítja, például a Customerkey a Dimension.Customer táblában. A tábla kulcsa, vagy hópehely-szerkezet esetén külső kulcs is lehet. Új **kiegészítő kulcsot** (surrogate key) vezetünk be, ha több forrásból van a dimenzió összefésülve, és a különböző forrásokból jövő OLTP kulcsok között ütközés lehet. Természetesen az eredeti OLTP kulcsokat is meg kell tartani a hivatkozás céljára. A ténytábla ilyenkor a kiegészítő kulcsra hivatkozik.
2. **Attribútum-mező:** a dimenzió olyan mezője, mely a tényre a vizsgált üzleti folyamat szempontjából jellemző. Például: Customer.Category. Mivel tipikusan pivot táblákban használják fel, ahol egy sor/oszlop van minden lehetséges értéknek, ezért a **folytonos** vagy túl nagy értékkészletű attribútumokat **diszkrétizálni** kell. Például ha az eladás ideje ezredmásodperc pontosságú, azért még nem akarunk egy új sort vagy oszlopot a kimutatásban minden ezredmásodpercnek, hanem napokra vagy hónapokra diszkrétizáljuk. Általános elv, hogy oszlopdiaگرامnak ne legyen 10-nél több oszlopa. Bár a diszkrétizálás automatizálható, a gyakorlatban (komoly alkalmazásra) csak szakértő által tervezett, feladatfüggő diszkrétizálást használunk.
3. **Név-mező:** az azonosított entitás kijelezhető neve, például Productname. A név lényegében egy speciális attribútum.

4. **Kiegészítő mező:** elemzési célra *nem használt* mező, mely az entitáshoz kapcsolódik, de szükséges lehet a kimutatáshoz, mint kiegészítő információ, például az alkalmazott telefonszáma. Ez a jelentésben címkéken jelenik meg.
5. A fentiek speciális esete a **lefordított** név-mező vagy kiegészítő mező
6. **Audit-mező:** A dimenzió- és ténytáblák **auditálása** (lineage) céljából speciális mezőkben tároljuk, honnan jött, hogyan keletkezett az adatrekord. A felhasználóknak soha nem mutatjuk meg.

Példa #1: A Stock Item dimenzió-tábla melyik mezője milyen típusú?

Megoldás:

Mezőnév	Mezőtípus
Stock Item Key	Kulcs (kiegészítő)
WWI Stock Item ID	Kulcs (OLTP)
Barcode	Kiegészítő mező
Photo	Kiegészítő mező
Stock Item	Név
Unit Price	attribútum
Color	attribútum
Brand	attribútum
Size	attribútum
Lead Time Days	attribútum
Tax Rate	attribútum
Is Chiller Stock	attribútum
Valid From	Audit
Valid To	Audit
Lineage Key	Audit

ÖNÁLLÓ FELADAT #3: A Customer dimenzió-tábla melyik mezője milyen típusú?

Hierarchiák

Az attribútumok lehetnek függetlenek egymástól, mint például a Stock Item táblában a Color, Size, Tax Rate stb. mezők. Ha viszont *funkcionálisan* (minden esetben és egyértelműen) függnek egymástól, akkor **természetes hierarchia** alakul ki. Például a Dimension.Date-ben: Date (egy teljes dátum), Month, Calendar Year. A hierarchiának általunk választott **szintjei** vannak: CalendarYear → Month → Date, de lehet, hogy adott célra elég a CalendarYear → Month is.

Az egyes szinteken a ténylegesen előforduló értékek a szint **tagjai** (members), például a 12 hónap a hónapos szinten.

ÖNÁLLÓ FELADAT #4: Lehet-e természetes hierarchia a Stock Item Color és a Buying Package mezői között (a tartalom vizsgálata alapján)? Ha igen, hogyan? Ha nem, miért nem?

Ténytáblák

A ténytáblák tárolják az információs rendszer eseményeit, melyek kontextusát a dimenziók adják. A ténytábla mezőtípusai:

1. A ténytábla alapjai a **mértékek** (measure): numerikus, aggregálható mezők, a folyamat szempontjából lényeges érték (figure of merit), például költség.
2. **Külső kulcsok** a dimenziókhoz illetve a ténytábla forrásául szolgáló OLTP táblához
3. **Kiegészítő kulcs mező(k)** (surrogate key) Identity típusú vagy sequence alapú mező. Mivel a ténytábla külső kulcsai általában (de nem mindig!) együttesen egyértelműen azonosítanak minden rekordot, ezért nem minden esetben szükséges a ténytáblába az adattárházban generált kiegészítő kulcsot tenni. Ellenpélda lehet az, ha például a dimenziók a dátum, a vevő és a termék, a dátumot csak napra pontosan tároljuk, és előfordulhat, hogy egy napon ugyanaz a vevő többször is megrendeli ugyanazt a terméket.
4. **Audit mezők**

Mértéktípusok

- **Extenzív** (összeadódó) jellegű, például bevétel, csapadékmennyiség, költség, felvett hitel. Ezekre általában SUM aggregációt számolunk.
- **Intenzív** (nem összeadódó) jellegű, például százalékok, árak, vízszint. Aggregáció: AVG, MIN/MAX vagy semmi.
- **Gyengén összeadódó** (semi-additive) jellegű, ha minden dimenzió mentén van értelme a SUM-nak, **kivéve az időt**. Példa: bankszámla-egyenleg. Ilyenre lehet speciális aggregációt is tervezni, például az idő dimenzióban az aggregáció eredménye az utolsó érvényes érték, a többi dimenzióban SUM.

Például a Fact.Sale ténytáblában extenzív a Quantity és a Tax Amount mérték. A Tax Rate és a Unit Price intenzív.

ÖNÁLLÓ FELADAT #6:

Keressünk extenzív és intenzív mértékeket a Fact Purchase ténytáblában!

Ténytáblák tervezése

Egy új adattárháznak először a dimenzió-tábláit kell megtervezni és feltölteni, mert a ténytáblák ezekre hivatkoznak.

Ezután, *egy-sok kapcsolat* esetén a ténytábla tervezése a ténytábla forrástáblájával kezdődik, ehhez adjuk hozzá a dimenzió-táblák külső kulcsait. Például egy olyan adatbázisban, amely a WWI-hez hasonlóan vevőkhöz, termékekhez tartozó rendeléseket és ezek tételeit tartalmazza, a tervezés bemenete:

Rendelési tételek: OrderLines tábla

- **OrderLineID**, StockItemID, Quantity, UnitPrice (félkövéren a kulcs)

Rendelések: Orders tábla

- **OrderID**, OrderDate, CustomerID

Tegyük fel, hogy a vásárló és a termék dimenzió azonosítására kiegészítő kulcsot vezetünk be (ProductKey, CustomerKey), és a dátumot is külön dimenzióban tároljuk a hatékonyság érdekében. Ekkor a ténytábla szerkezete:

- **OrderLineID**, ProductKey, CustomerKey, OrderDateKey, Quantity, UnitPrice (félkövér az elsődleges kulcs, aláhúzva a külső kulcsok)

Adattárházak megvalósítási kérdései

Hogyan kell megvalósítani a ténytáblákat és a dimenziókat az OLTP források alapján?

Az adatbázis beállításai

- Simple recovery (emlékeztető: a full recovery nem csonkítja a tranzakciós logot)
- Auto shrink -> off a fragmentáció elkerülése érdekében. Az Auto grow is ki lehet kapcsolva, ha helyesen meg tudjuk becsülni az adattárház várható méretét. Például, ha 10 évre tervezünk: $1.25 * (10 * 1 \text{ éves adatmennyiség betöltése utáni adatbázis fájl méret})$. Transaction log mérete: a legnagyobb lehetséges INSERT alapján becsülhető, ne feledjük, hogy minden SQL utasítás egy implicit tranzakció!
- A nagy ténytáblákat érdemes particionálni. Az egyes partíciókat külön fájlcsoporthoz lehet helyezni. Ennek részleteit itt nem tárgyaljuk.

Az átmeneti tároló (staging area)

Első lépésben célszerű az importált adatokat csak tárolni egy átmeneti tárolóban (**staging area**), amely lehet külön séma vagy akár külön adatbázis. Az **előnyök**:

- ha a források változnak, elég az Extract (adatimportálást végző) scripteket változtatni, a Transform scripteket nem kell
- az adattárházhoz általában sokan hozzáférnek, de ha a staging táblák általuk nem látható sémában vannak, akkor nem fognak nyers, tisztítatlan adatokból téves jelentéseket készíteni.

A staging több fokozatú is lehet:

- a nyers adatok tárolója: egyszerű másolat, vagy CDC (change data capture, lásd később)
- adattisztítás (Data Quality) után jön a következő tároló, a „tisztá” staging
- a szükséges átalakítások (Transform) után a betöltésre váró adatok tárolója
- betöltés után maga az adattárház
- az adattárházból ezután specializált célú kisebb adattárházak (**adatpiac, Data Mart**) készíthetők

Az adattárházat nem csak létrehozni és aktualizálni, hanem karbantartani is kell. Ha az adattárházra OLAP-kockák épülnek, akkor ezek processzálása (lásd később) szintén az adatbetöltés/aktualizálás részét képezi.

A dimenzió-táblák megvalósítása

Figyeljük meg az alábbi példában:

- Kiegészítő kulcsok tervezése
- Kategorizálás
- Számított mező (computed column) jól használható folytonosan változó attribútumokhoz például az életkor=rendszeridő - születési idő. A mező minden lekérdezéskor kiszámítódik, ezért mindig helyes lesz. A számított mezőt egyúttal lehet diszkretizálásra is használni (fiatal/középkorú/idős)

A ténytáblák megvalósítása

A ténytáblákat a dimenzió-táblák **után** kell megvalósítani a külső kulcs hivatkozások miatt.

PÉLDA ETL folyamatra:

A Sales.Customers és Warehouse.StockItems táblákból készítsünk staging táblákat, majd ezekből dimenzió táblákat. Az érdekes mezők: customerid, customername, deliverycity, customercategory. Adjunk a táblához kiegészítő kulcsot! Készítsük el a feltöltött dimenzió-táblák ténytábláját. A tábla neve legyen fact_sales, mértékei az egységár, mennyiség, és a tétel számított értéke. Három külső kulccsal biztosítsuk a vevő, a dátum és a termék dimenzió becsatolhatóságát! A ténytáblát csatoljuk a dimenzió-táblákhoz és töltsük fel adatokkal a WWI adatbázisból!

```
create database test
use test
go
if not exists (select * from sys.schemas where name='stg') exec ('create schema stg')
if not exists (select * from sys.schemas where name='test') exec ('create schema test')
go
--WWI: kiinduló staging táblák, csak a szükséges mezőkkel
use WideWorldImporters
EXEC sp_changedbowner 'sa';
ALTER AUTHORIZATION ON DATABASE::WideWorldImporters TO sa;

--INITIAL LOAD
go
--STOCKITEM dimenzió
--drop table test.stg.stockitems
create table test.stg.stockitems (
    stockitemid int,
    outerpackageid int,
    supplierid int,
    colorid int,
    stockitemname nvarchar(100),
    brand nvarchar(50),
    unitprice decimal(18,2),
    size nvarchar(20),
    validfrom datetime2 not null)
go
delete test.stg.stockitems
insert test.stg.stockitems (stockitemid, outerpackageid, supplierid, colorid,
stockitemname, brand, unitprice, size, validfrom)
    select StockItemID, OuterPackageID, SupplierID, ColorID, StockItemName, Brand,
UnitPrice, Size, ValidFrom
    from Warehouse.StockItems --227
go
create table test.stg.suppliers (
    supplierid int,
    deliverycityid int,
    suppliername nvarchar(100))
go
delete test.stg.suppliers
insert test.stg.suppliers (supplierid, deliverycityid, suppliername)
    select SupplierID, DeliveryCityID, SupplierName from Purchasing.Suppliers --13
go
create table test.stg.packagetypes (
    packagetypeid int,
    packagetypername nvarchar(50))
go
delete test.stg.packagetypes
insert test.stg.packagetypes (packagetypeid, packagetypername)
```



```

        select PackageTypeID, PackageTypeName from Warehouse.PackageTypes
go
create table test.stg.colors (
    colorid int,
    colorname nvarchar(20))
go
insert test.stg.colors (colorid, colorname)
        select ColorID, ColorName from Warehouse.Colors
go
--stockitem dim. tábla
--ehhez hasznoló bővebben: WideWorldImportersDW.Dimension.[Stock Item], staging tábla
feltöltése: WideWorldImporters.Integration.GetStockItemUpdates()
--drop table test..dim_stockitem
create table test..dim_stockitem (
    dw_stockitemid int identity(1000,1) primary key, --kieg. kulcs
    stockitemid int not null,
    itemname nvarchar(100) not null,
    color nvarchar(20) null,
    package nvarchar(50) not null, --SCD-2 mező
    brand nvarchar(50) null,
    unitprice_num decimal(18,2) not null,
    unitprice nvarchar(50) not null,
    size nvarchar(20) null,
    suppliername nvarchar(100) null,
    suppliercity nvarchar(50) null,
    validfrom datetime2 not null,
    validto datetime null )
go
--select UnitPrice from Warehouse.StockItems order by UnitPrice
--feltöltés: kategorizálás
delete test..dim_stockitem
insert test..dim_stockitem (stockitemid, itemname, color, package, brand, unitprice_num,
unitprice, size, suppliername, suppliercity, validfrom)
        select i.StockItemID, i.StockItemName, c.ColorName, pt.PackageTypeName, i.Brand,
i.UnitPrice,
        case when UnitPrice<=13 then 'low' when UnitPrice < 32 then 'middle' else 'high'
end,
        i.Size, s.SupplierName, ci.CityName, i.ValidFrom
        from test.stg.StockItems i join test.stg.PackageTypes pt on
i.OuterPackageID=pt.PackageTypeID
        join test.stg.Suppliers s on s.SupplierID=i.SupplierID
        join Application.Cities ci on ci.CityID=s.DeliveryCityID --ennek nem csináltunk
staging táblát
        LEFT join test.stg.Colors c on c.ColorID=i.ColorID--227
select * from test..dim_stockitem
go

--a fenti CASE így nézne ki egy SSIS Derived Column csomópontban:
-- unitprice <= 13 ? "low" : unitprice < 32 ? "middle" : "high"

--a csomagparaméteres DEMO kedvéért a WWI DW-ben lesz az 'éles' dim tábla:
create table WideWorldImportersDW..dim_stockitem (
    dw_stockitemid int identity(1000,1) primary key, --kieg. kulcs
    stockitemid int not null,
    itemname nvarchar(100) not null,
    color nvarchar(20) null,
    package nvarchar(50) not null, --SCD-2 mező
    brand nvarchar(50) null,

```

```

        unitprice_num decimal(18,2) not null,
        unitprice nvarchar(50) not null,
        size nvarchar(20) null,
        suppliername nvarchar(100) null,
        suppliercity nvarchar(50) null,
        validfrom datetime2 not null,
        validto datetime null )
go
create table test.test.dim_stockitem (
    dw_stockitemid int identity(1000,1) primary key, --kieg. kulcs
    stockitemid int not null,
    itemname nvarchar(100) not null,
    color nvarchar(20) null,
    package nvarchar(50) not null, --SCD-2 mező
    brand nvarchar(50) null,
    unitprice_num decimal(18,2) not null,
    unitprice nvarchar(50) not null,
    size nvarchar(20) null,
    suppliername nvarchar(100) null,
    suppliercity nvarchar(50) null,
    validfrom datetime2 not null,
    validto datetime null )
go
select * from test.test.dim_stockitem

--BUYER dimenzióknak nem készítünk staging táblákat
--ehhez hasonló bővebben: WideWorldImportersDW.Dimension.Customer
create table test..dim_buyer (
    dw_buyerid int identity(10000,1) primary key, --kieg. kulcs
    buyerid int not null, --OLTP kulcs
    buyername nvarchar(100) not null,
    paymentdays int not null,
    category nvarchar(50) not null,
    buyercity nvarchar(50) null,
    emailaddress nvarchar(50) null default 'N/A',
    birthyear int NULL, --azért csak az év, mert a kiegészítő személyi adatok között
csak ez elérhető
    age as year(getdate())-BirthYear, --számított mező
    maritalstatus nchar(5) null default 'N/A',
    gender nchar(5) NULL default 'N/A',
    education nvarchar(40) null default 'N/A',
    occupation nvarchar(100) null default 'N/A',
    validfrom datetime not null,
    validto datetime null )
go
delete test..dim_buyer
go
insert test..dim_buyer (buyerid, buyername, paymentdays, category, buyercity, validfrom)
select c.CustomerID, c.CustomerName, c.PaymentDays, cc.CustomerCategoryName, ci.CityName,
c.ValidFrom
    from Sales.Customers c join Sales.CustomerCategories cc on
c.CustomerCategoryID=cc.CustomerCategoryID
    join Application.Cities ci on ci.CityID=c.DeliveryCityID --663
select * from test..dim_buyer
go
create table test..dim_date (
    dateid int identity(10000,1) primary key,
    full_date date not null,

```

```

    year int not null,
    month int not null,
    day int not null)
go
insert test..dim_date (full_date, year, month, day)
select distinct OrderDate full_date, year(orderdate), month(orderdate), day(orderdate)
from sales.Orders --1069

--kiegészítő adatok egy szövegfájlból jönnek majd (példa a heterogén adatútra)
select * from test.stg.customerinfo order by personid --300..18783
select distinct 300+buyerid from test..dim_buyer where 300+buyerid not in
    (select personid from test.stg.customerinfo) --0: mind megvan

--KÖZÖSEN MEGOLDOTT FELADAT:
--készítsünk ténytáblát 3 dimenzióval: dátum, buyer, stockitem
--hasonló: WideWorldImportersDW.Fact.Sale, feltöltése:
WideWorldImporters.Integration.GetSaleUpdates()
--/*

```

ÖNÁLLÓ FELADAT: AW adatbázisban a Production.Product tábla alábbi 5 mezője alapján készítsünk egy staging táblát, majd ebből töltsük fel a dim_product táblát. Az érdekes mezők: ProductID, Name, Weight, Color, Style. Ezután készítsünk ténytáblát az elkészített product dimenzióhoz a Production.ProductInventory (raktárkészlet) tábla alapján! Erről először készítsünk másolatot a staging sémába, majd hozzuk létre a ténytáblát fact_inventory néven. A tábla csak a product dimenzióra hivatkozik, a mértéke a Quantity mező legyen. Végül töltsük fel a ténytáblát!

Változások kezelése

Az AT létrehozása és első feltöltése után gondoskodni kell az adatok rendszeres frissítéséről. Szokás ezt inkrementális adatbetöltésnek (incremental load) vagy aktualizálásnak is nevezni. Lényeges, hogy az aktualizálás minél kevésbé zavarja az OLTP rendszert, ezért tipikusan ütemezetten, alacsony terhelésű időszakokban történik az új adatok kinyerése (Extract). A változások követése lehet pull (az adattárház oldalán ütemezett job), vagy push jellegű (ez ritkább).

Az utolsó frissítés óta történt változások háromfélék lehetnek minden forrás-tábla esetén: új rekordok jelentek meg, rekordok módosultak, illetve rekordokat töröltek. A változás-kezelés stratégiája más a ténytáblák és a dimenzió-táblák esetén:

- A ténytáblák rekordjaira nem hivatkozik másik rekord. Ezért:
 - Az **új tényrekordokat** be kell szűrni (az általuk hivatkozott esetleges új dimenzió-rekordok beszúrása után)
 - A **törölt rekordok** érvényességét (státuszát) át kell állítani töröltre. Ezt logikai törlésnek is nevezik. A ténytáblára lehet például olyan nézetet (view-t) készíteni, amely csak a nem törölt rekordokat mutatja
 - A **módosított forrás-rekordok** esetén egyszerűen átírjuk a régi tartalmat az újra, mivel ez egy tényyszerű hiba javításának felel meg
- A dimenzió-táblák esetén:
 - Az **új rekordokat** be kell szűrni

- A **törölt rekordok** érvényességét (státuszát) át kell állítani töröltre. Ezzel együtt az összes rájuk hivatkozó tény-rekord státuszát is állítani lehet (ezt megítélés kérdése).
- A **módosítások** esetén a helyzet azért bonyolult, mert a rekordra már meglévő tény-rekordok hivatkozhatnak. Ha a változás olyan mezőben következett be, amely a probléma jellegéből adódóan soha nem változhat természetes módon (például a születési dátum), akkor a változás egy korábbi adathiba javítása, amit egyszerűen át kell írni a dimenzió-táblában. Hasonlóképpen, a kiegészítő mezők (telefonszám, email cím) átírhatók, mert ezek a jelentés érdemi részét nem befolyásolják. Egyéb esetben azonban tipikusan nem dobhatjuk el a régi értéket. Ezt a problémát a „lassan változó dimenziók” problémájaként ismerik.

FONTOS: Ha egy dimenzió-attribútum gyakran és „üzemszerűen” változik, akkor az már nem jellemzi a dimenzió-tagot, ezért le kell választani a dimenzióról, és önálló dimenzióként kell közvetlenül a ténytáblához csatolni. Ezzel megszűnik a változás kezeléséből adódó probléma. A döntést az adattárház tervezésekor kell meghozni!

Lassan változó dimenziók (Slowly changing dimensions, SCD)

Tegyük fel, hogy egy adattárházban a vásárló országa a vásárló dimenzió attribútuma, tehát nem önálló dimenzió, mivel a vásárló országa nem szokott változni. De ha valaki mégis elköltözik, akkor melyik címe legyen az adattárházban? Az alábbi példákban tegyük fel, hogy John Doe 2015. május 3-án Franciaországból Németországba költözött. A lehetséges megoldások:

SCD1: Csak a legutolsó érték tárolása, tehát ha az OLTP cím változik, akkor az adattárházban is UPDATE utasítást hajtunk végre. Ez a legegyszerűbb megoldás, azonban a múltra vonatkozó lekérdezések csálni fognak!

CustomerDwKey	CustomerKey	FullName	Country
1	100	John Doe	Germany
2	101	Tom Doe	Egypt

SCD2: Az idősor tárolása, tehát külön rekordokban tároljuk az összes értéket az érvényességével együtt. Új, erre a célra bevezetett mező(k)ben kell jelezni, hogy a több cím közül melyik az aktuális, vagy mikor volt használatban (például Current mező, vagy ValidAt/ValidFrom/ValidTo mező). **Az OLTP-kulcs sohasem változhat**, mert ez alapján azonosítjuk a megváltozott entitást (az alábbi ábrán az OLTP kulcs a CustomerKey). A ténytáblához való csatolás céljára kiegészítő kulcsot hozunk létre, az alábbi ábrán ez a CustomerDwKey **Ez a szakmai szempontból igényesebb, preferált megoldás**, amelynek megvalósítása ugyanakkor a másik két megoldásnál lényegesen nehezebb.

CustomerDwKey	CustomerKey	FullName	Country	CurrentFlag	ValidFrom	ValidTo
1	100	John Doe	France	False	2012-01-23	2015-05-02
2	100	John Doe	Germany	True	2015-05-03	NULL
3	101	Tom Doe	Egypt	True	2012-01-01	NULL

SCD3: Új mezők felvétele a dimenzió-táblába, és korlátozott méretű idősor tárolása, az alábbi példában egyetlen múltbeli értékre. Ez a I (nem ajánlott, csak a teljesség kedvéért említjük)

CustomerDwKey	CustomerKey	FullName	Country	ValidFrom	PreviousCountry
1	100	John Doe	Germany	2015-05-03	France
2	101	Tom Doe	Egypt	2012-01-01	NULL

ÖNÁLLÓ FELADAT #5: Demonstráljuk a lassan változó dimenziók támogatását egy egyetemi példán! A ténytáblában a kurzus-eredmények féléves időbeli felbontással, tehát szemeszterenként vannak. A kiinduló állapotban az adattárház hallgató dimenziójában csak az éppen aktuális értékek vannak tárolva.

Ritkán, de előfordul, hogy valaki szakot vált. Hogy a szakonkénti kimutatásaink ilyenkor se legyenek rosszak, vezessük be az SCD2 megoldást! Ehhez a hallgató-dimenzió táblájában alkalmazzunk identity típusú kiegészítő kulcsot! A teszteléshez tegyük fel, hogy Magas Dávid 2018 ősze után jött át a 'mérnök info.' szakra a 'prog. info.' szakról.

```
--kiinduló állapot:
use dw_test
go
--csillag-szerkezet
create table dim_hallgato (
    hallgato_id int primary key,
    neptun_kod char(6),
    hallgato_nev nvarchar(200),
    szak_nev varchar(100))
create table dim_kurzus (
    kurzus_id int primary key,
    kurzus_nev nvarchar(200))
create table dim_szemeszter (
    szemeszter_id int primary key,
    szemeszter_kod varchar(200))
create table tény_kurzus_eredmeny (
    hallgato_id int not null references dim_hallgato(hallgato_id),
    kurzus_id int not null references dim_kurzus(kurzus_id),
    szemeszter_id int not null references dim_szemeszter(szemeszter_id),
    eredmeny smallint,
    felvetelek_szama smallint,
    constraint p1 primary key (hallgato_id, kurzus_id, szemeszter_id))
go
--néhány demo rekord
--delete dim_szemeszter
insert dim_szemeszter (szemeszter_id, szemeszter_kod) values (1, '2018_1'), (2, '2018_2'),
(3, '2019_1') --a kód alapján időben sorba rendezhető
insert dim_kurzus (kurzus_id, kurzus_nev) values (100, 'Adattárházak'), (101, 'Zongora'),
(103, 'Filozófia')
insert dim_hallgato (hallgato_id, neptun_kod, hallgato_nev, szak_nev) values
(10, 'djkre4', 'Magas Dávid', 'mérnök info.'), (11, 'ds41e4', 'Szép Nóra', 'gazd.
info.')
--delete tény_kurzus_eredmeny
insert tény_kurzus_eredmeny (hallgato_id, kurzus_id, szemeszter_id, eredmeny,
felvetelek_szama) values
(10, 100, 1, 50, 1), (10, 101, 2, 70, 1), (10, 100, 3, 50, 2), (11, 100, 1, 70, 1),
(11, 101, 2, 80, 1)

select h.szak_nev, avg(t.eredmeny) eredm_avg
from tény_kurzus_eredmeny t inner join dim_hallgato h on t.hallgato_id=h.hallgato_id
group by h.szak_nev

szak_nev          eredm_avg
```

gazd. info.	75
mérnök info.	56

A fenti megvalósítás nem tudja követni a hallgató dimenzió lassú változását, tehát például azt, ha egy hallgató szakot vált. Ezért szak-váltás esetén a fenti lekérdezés hibás eredményt ad. Alakítsuk át ezt a dimenziót úgy, hogy külön mezőkben tárolja, hogy a szak melyik szemesztertől melyik szemeszterig érvényes. Ehhez hivatkozzon a dim_szemeszter táblára.

A megoldás lépései:

1. jelenlegi tény_kurzus_eredmeny, dim_hallgato táblák megszüntetése (DROP)
2. az új dim_hallgato tábla létrehozása a kiegészítő kulccsal és az új SCD2 mezőkkel
3. hallgató adatok feltöltése (INSERT)--ehhez tegyük fel, hogy Magas Dávid 2018 ősze után jött át a 'mérnök info.' szakra a 'prog. info.' szakról.
4. új ténytábla létrehozása, mely már a kiegészítő kulcsra hivatkozik
5. tény-adatok beírása (INSERT). Ezek nem változtak!
6. a szakonkénti eredményt helyesen visszaadó SELECT megírása

SCD 1-2 megvalósítása realisztikusabb példán

Valósítsuk meg egy dimenzió-táblánk (a dim_customers) változás-követését egy, a fenténél realisztikusabb példán!

SCD 1 esetén a teendő egy „UPSERT” (a jelenlegi tartalomtól függően vagy UPDATE vagy INSERT) jellegű művelet, tehát ha egy tag (vásárló) nem létezik, akkor INSERT, különben UPDATE. Az INSERT/UPDATE egy SQL MERGE utasítással is megoldható lenne.

```
--1. a még nem létező rekordokat beszúrjuk
insert dim_customers (CustomerKey, FullName, CountryRegion, StateProvince)
select p.BusinessEntityID as CustomerKey, p.FirstName+' '+p.LastName as fullname, t.CountryRegionCode as
CountryRegion,
t.Name as StateProvince
from (stg.person p inner join stg.Customer c on p.BusinessEntityID=c.PersonID)
left outer join AdventureWorks2016CTP3.Sales.SalesTerritory t on c.TerritoryID=t.TerritoryID
where p.BusinessEntityID not in (select CustomerKey from dim_customers)

--2. a változott rekordokat átírjuk
--tegyük fel, hogy a CountryRegion és a StateProvince mezőket szeretnénk követni
update dim_customers set CountryRegion=stg.CountryRegion, StateProvince=stg.StateProvince
from dim_customers d inner join (
select p.BusinessEntityID as CustomerKey, p.FirstName+p.LastName as fullname, t.CountryRegionCode
as CountryRegion,
t.Name as StateProvince
from (stg.person p inner join stg.Customer c on p.BusinessEntityID=c.PersonID)
left outer join AdventureWorks2016CTP3.Sales.SalesTerritory t on c.TerritoryID=t.TerritoryID
) as stg on d.CustomerKey=stg.CustomerKey --az OLTP kulcs nem változhat!!

--teszt
insert stg.person (BusinessEntityID, LastName, FirstName) values (100000, 'Joe', 'New')
insert stg.customer (TerritoryID, CustomerID, PersonID) values (6, 200000, 100000)
--1) futtatása után:
select * from dim_customers where CustomerKey=100000 --1 rekord, Kanadában

update stg.customer set TerritoryID=7 where PersonID=100000 --elköltözött
--2) futtatása után:
select * from dim_customers where CustomerKey=100000 --1 rekord, Franciaországban
```

SCD 2 esetén bonyolultabb a helyzet: a dim táblát először ki kell egészítenünk a követő mezőkkel, utána a változott tag-rekordokat meg kell jegyezni, és a változást dokumentáló új rekordokat ez alapján kell majd

beszúrni. A demo jól mutatja, **milyen számszerű hibát követünk el** az SCD 1 használatával, amit az SCD 2 megold. Mivel az OLTP rendszer nem tárolja a korábbi értékeket, ennek a problémának csak a rendszeres adattárház-frissítés lehet a megoldása. **Ha két frissítés között egy tag kétszer is változik, akkor az első változás elvész.** Ha az adattárházat naponta (például éjszakánként) frissítjük, ennek az esélye kizárható.

```
--dim. tábla kibővítése ValidFrom, ValidTo, Current mezőkkel
CREATE TABLE dbo.dim_customers_scd2 (
CustomerDwKey INT NOT NULL default next value for SeqCustomerDwKey,
CustomerKey INT NOT NULL,
FullName NVARCHAR(150) NULL,
StateProvince NVARCHAR(50) NULL,
CountryRegion NVARCHAR(50) NULL,
CurrentFlag BIT NOT NULL DEFAULT 1,
ValidFrom date,
ValidTo date
CONSTRAINT PK_dimCustomers_2 PRIMARY KEY (CustomerDwKey)
)
go
truncate table dim_customers
go
--a változást úgy vesszük észre, hogy a stg.customer táblában a ModifiedDate mező későbbi, mint a
--current mező ValidFrom dátuma
--lépések:
--1. az új rekordok beszúrása
--2. meglévő rekordok aktualizálása beszúrással

--1. az új rekordok beszúrása
insert dim_customers_scd2 (CustomerKey, FullName, CountryRegion, StateProvince, ValidFrom)
select p.BusinessEntityID as CustomerKey, p.FirstName+' '+p.LastName as fullname, t.CountryRegionCode as
CountryRegion,
t.Name as StateProvince, c.ModifiedDate --figyelem: itt az új mező
from (stg.person p inner join stg.Customer c on p.BusinessEntityID=c.PersonID)
left outer join AdventureWorks2016CTP3.Sales.SalesTerritory t on c.TerritoryID=t.TerritoryID
where p.BusinessEntityID not in (select CustomerKey from dim_customers_scd2)

--2. a meglévő rekordok aktualizálása, azaz új rekord beszúrás
--tegyük fel, hogy az adattárházunk még 2011-ben lett létrehozva:
use dw_test
update stg.customer set ModifiedDate='2010-09-10'
update dim_customers_scd2 set validfrom='2011-01-10'
--kiválasztunk egy vásárlót:
select top 1 * from dim_customers_scd2
--ő eredetileg US volt (mikor a dim-táblát feltöltöttük)
--customerkey, tehát customer.PersonID=291, customer.territoryid=1
--Tegyük fel, hogy 2012. febr. 10-én ő átköltözött Ausztráliába, territoryid = 9
--a 2012-es staging táblában már ez van:
update dw_test.stg.Customer set TerritoryID=9, ModifiedDate='2012-02-10' where PersonID=291
--SCD2 MEGOLDÁS:
--az új stg táblából észrevesszük, hogy változás történt
--és a változott vásárlókat megjegyezzük
USE dw_test
select d.CustomerKey
into #valtozott
from dim_customers_scd2 d inner join stg.customer c on d.CustomerKey=c.PersonID
where d.CurrentFlag=1 and d.ValidFrom<c.ModifiedDate --módosult vásárlónként csak egy rekord lehet a
CurrentFlag miatt
--a módosult korábbi rekordokat lezárjuk:
update dim_customers_scd2 set CurrentFlag=0, ValidTo=c.ModifiedDate
from dim_customers_scd2 d inner join stg.customer c on d.CustomerKey=c.PersonID
where d.CurrentFlag=1 and d.ValidFrom<c.ModifiedDate
--új rekordokba beszúrjuk az új értéket:
insert dim_customers_scd2 (CustomerKey, FullName, CountryRegion, StateProvince, ValidFrom)
select p.BusinessEntityID as CustomerKey, p.FirstName+' '+p.LastName as fullname, t.CountryRegionCode as
CountryRegion,
t.Name as StateProvince, c.ModifiedDate
from (stg.person p inner join stg.Customer c on p.BusinessEntityID=c.PersonID)
left outer join AdventureWorks2016CTP3.Sales.SalesTerritory t on c.TerritoryID=t.TerritoryID
```

```

where p.BusinessEntityID in (select CustomerKey from #valtozott)
--már 2 rekordja van a 290-nek:
use dw_test
select * from dim_customers_scd2 where CustomerKey=291

/*****
--demo: SCD2 nélkül rossz országhoz kerül egy korábbi bevétel
*****/
--ha csak a mostani állapot érdekes (SCD1), akkor ez a lekérdezés:
use AdventureWorks2016CTP3
SELECT year(soh.OrderDate) as Év, d.CountryRegion as Ország, cast(sum(sod.LineTotal) as money) as Összeg
FROM Sales.SalesOrderHeader AS soh
INNER JOIN Sales.SalesOrderDetail AS sod ON soh.SalesOrderID = sod.SalesOrderID
INNER JOIN dw_test.stg.Customer AS c ON soh.CustomerID = c.CustomerID
inner join dw_test.dbo.dim_customers d on d.CustomerKey=c.PersonID --itt a korábbi, SCD1-es dim táblát
használjuk
where c.PersonID=291
group by year(soh.OrderDate), d.CountryRegion
order by Év, Ország, Összeg --2011,12,13: mindhárom összeg AU, hamis az eredmény
--ha az SCD2-vel csak a legfrissebb értéket nézzük, ugyanezt kapjuk:
SELECT year(soh.OrderDate) as Év, d.CountryRegion as Ország, cast(sum(sod.LineTotal) as money) as Összeg
FROM Sales.SalesOrderHeader AS soh
INNER JOIN Sales.SalesOrderDetail AS sod ON soh.SalesOrderID = sod.SalesOrderID
INNER JOIN dw_test.stg.Customer AS c ON soh.CustomerID = c.CustomerID
inner join dw_test.dbo.dim_customers_scd2 d on d.CustomerKey=c.PersonID --itt már az SCD2-es dim táblát
használjuk
where c.PersonID=291 and d.CurrentFlag=1 --tehát nem a régi címét nézzük
group by year(soh.OrderDate), d.CountryRegion
order by Év, Ország, Összeg --hamis az eredmény, mert mind3 év AU
2011 AU 4049.988 --ez a sor hibás, mert akkor még az USA-ban lakott
2012 AU 65177.168 --ezt az évet kell majd megbontani, mert 2012-ben költözött
2013 AU 61875.8264 --ez a sor jó

--SCD2-vel, helyesen:
SELECT year(soh.OrderDate) as Év, d.CountryRegion as Ország, cast(sum(sod.LineTotal) as money) as Összeg
FROM Sales.SalesOrderHeader AS soh
INNER JOIN Sales.SalesOrderDetail AS sod ON soh.SalesOrderID = sod.SalesOrderID
INNER JOIN dw_test.stg.Customer AS c ON soh.CustomerID = c.CustomerID
inner join dw_test.dbo.dim_customers_scd2 d on d.CustomerKey=c.PersonID
where c.PersonID=291 and (
    (soh.OrderDate >= d.ValidFrom and soh.OrderDate < d.ValidTo)
    or
    --mert a From-To értékek nem lapolódnak át
    (d.CurrentFlag=1 and soh.OrderDate>d.ValidFrom)) --ez pedig az egyetlen aktuális rekordra
group by year(soh.OrderDate), d.CountryRegion
order by Év, Ország, Összeg
--az eredmény: sikerült megbontani a 2012-es évet, jó a lekérdezés:
2011 US 4049.988
2012 US 4079.988
2012 AU 61097.18
2013 AU 61875.8264
--Hurrá!

```

A változás-követést támogató technológiák

A fenti SCD2 megoldásban a változás-követés alapja a változott, új vagy törölt rekordok azonosítása volt, melyet a forrás táblában lévő állapot-mező (Customer.ModifiedDate) lekérdezésével oldottunk meg. Azonban erre nem mindig lehet építeni. Ezért, és a szükséges programozás csökkentése céljából több technológia is támogatja a forrás-táblák változásának a naplózását. A cél, hogy a naplók az adattárház frissítésekor automatizáltan feldolgozhatók legyenek. Ezeket a technológiákat minden komolyabb gyártó (Oracle, Postgres, SQL Server, stb.) támogatja.

- Change track (CT): Több változás esetén csak a legutolsó érték kerül a naplóba, ezért az ETL céljára nem ajánlott.

- Change Data Capture (CDC): A CDC jól alkalmazható adattárházak automatizált frissítésére, az Extract lépés végrehajtására. A megoldás előnye, hogy csak a változott adatot kell mozgatni. Hátrány, hogy bár van megoldás a CDC alatti séma-változások kezelésére, inkább kerülendő az aktívan logolt tábla sémájának változtatása.
 - A CDC bekapcsolása egy adatbázisra, illetve egy táblára. Ezáltal létrejönnek a CDC-t megvalósító rendszertáblák és jobok.
 - A megfigyelt tábla minden változása bekerül a tranzakciós log egy bejegyzésébe (a CDC nélkül is).
 - A logot figyeli egy folyamatosan (az SQL Server esetében 5 másodpercenként) futó job, amely azonosítja a releváns bejegyzést, és kimásolja a tartalmát speciális CDC-táblákba. Ez a job ideiglenesen, például nagy terhelésű időben, szüneteltethető, és ezalatt sem veszik el semmilyen változás. Az elv hasonlít a tranzakcionális replikációhoz, sőt az SQL Serveren a log figyelésére a mélyebb rétegben ugyanazt a modult is használják megosztva, azonban a replikációval szemben a CDC lehetővé teszi a legutóbbi szinkronizálás óta lezajlott események egyenkénti feldolgozását.
 - A régi, 3 naposnál idősebb eseményeket egy másik „takarító” job folyamatosan (naponta egyszer) törli a CDC-táblákból.
 - A CDC-táblák lekérdezhetők, segítségükkel a változások feldolgozhatók. Az induló LSN az a tranzakciós log bejegyzés szám (LSN), mikor a CDC logolás elkezdődött. Ezt az értéket a takarító job a törlések után aktualizálja. A záró LSN az az LSN, ameddig a naplózó job feldolgozta a tranzakciós logot. A CDC-logból táblaértékű függvényekkel lekérdezhetők két tetszőleges, az érvényességi időbe eső LSN közötti események. A kapott tábla tartalma a változás tranzakciójának kezdő LSN-je, a tranzakción belüli esemény-sorszám, a művelet kódja (1: DELETE, 2: INSERT, 4: UPDATE), a változott mezők bitmaszkja, és a mezők értéke. A lekérdezés tartalmazhatja az összes eseményt részletezve, vagy csak az összesített változás formájában. Például összesítve több egymás utáni UPDATE esetén csak az utolsó jelenik majd meg. Bővebben²
- Temporális táblák (Temporal tables, System versioned tables): A verziózott táblához létrejön egy speciális, csak rendszer által írható log-tábla, amely minden módosító és törölő utasítás hatására bővül (új rekord beszúrásakor nem). A tábla tartalmát egyszerű szintaxissal az aktuális, illetve tetszőleges múltbeli időpontra vonatkozóan le lehet kérni. A temporális táblák előnye a CDC-vel szemben, hogy nem LSN-szám, hanem időbélyeg alapján lehet keresni őket. Mivel tetszőlegesen hosszú idősort képesek tárolni, ezért nem csak az ETL támogatására, hanem trendelemzésre (például egy árfolyam napi alakulása) is lehet őket használni.

A szinkronizálás alapgondolata, hogy az egyes futtatások között elmentjük az időbélyeget vagy az LSN-t az adattárház egy alkalmas táblájába, és a következő alkalommal innen indítjuk a lekérdezés időablakát. Ezáltal a változások inkrementális betöltését robosztussá, újraindíthatóvá tudjuk tenni.

² <https://www.mssqltips.com/sqlservertip/5211/sql-server-temporal-tables-vs-change-data-capture-vs-change-tracking-part-1/>
<https://www.mssqltips.com/sqlservertip/5212/sql-server-temporal-tables-vs-change-data-capture-vs-change-tracking-part-2/>
<https://www.mssqltips.com/sqlservertip/5144/sql-server-temporal-tables-vs-change-data-capture-vs-change-tracking--part-3/>

PÉLDA: Adattárház létrehozása, feltöltése és szinkronizálása CDC-vel és temporális táblákkal

```
--WW: kiinduló staging táblák, csak a szükséges mezőkkel
use WideWorldImporters
EXEC sp_changedbowner 'sa';
ALTER AUTHORIZATION ON DATABASE::WideWorldImporters TO sa;

--INITIAL LOAD

--STOCKITEM dimenzió
create table test.stg.stockitems (
    stockitemid int,
    outerpackageid int,
    supplierid int,
    colorid int,
    stockitemname nvarchar(100),
    brand nvarchar(50),
    unitprice decimal(18,2),
    size nvarchar(20),
    validfrom datetime not null)
go
delete test.stg.stockitems
insert test.stg.stockitems (stockitemid, outerpackageid, supplierid, colorid,
stockitemname, brand, unitprice, size, validfrom)
    select StockItemID, OuterPackageID, SupplierID, ColorID, StockItemName, Brand,
UnitPrice, Size, ValidFrom
    from Warehouse.StockItems --227
go
create table test.stg.suppliers (
    supplierid int,
    deliverycityid int,
    suppliername nvarchar(100))
go
delete test.stg.suppliers
insert test.stg.suppliers (supplierid, deliverycityid, suppliername)
    select SupplierID, DeliveryCityID, SupplierName from Purchasing.Suppliers --13
go
create table test.stg.packagetypes (
    packagetypeid int,
    packagetypername nvarchar(50))
go
insert test.stg.packagetypes (packagetypeid, packagetypername)
    select PackageTypeID, PackageTypeName from Warehouse.PackageTypes
go
create table test.stg.colors (
    colorid int,
    colorname nvarchar(20))
go
insert test.stg.colors (colorid, colorname)
    select ColorID, ColorName from Warehouse.Colors
go
--stockitem dim. tábla
--drop table test..dim_stockitem
create table test..dim_stockitem (
    dw_stockitemid int identity(1000,1) primary key, --kieg. kulcs
    stockitemid int not null,
    itemname nvarchar(100) not null,
    color nvarchar(20) null,
```

```

package nvarchar(50) not null, --SCD-2 mező
brand nvarchar(50) null,
unitprice_num decimal(18,2) not null,
unitprice nvarchar(50) not null,
size nvarchar(20) null,
suppliername nvarchar(100),
suppliercity nvarchar(50) not null,
validfrom datetime not null,
validto datetime null )
go
--select UnitPrice from Warehouse.StockItems order by UnitPrice
--feltöltés: kategorizálás
delete test..dim_stockitem
insert test..dim_stockitem (stockitemid, itemname, color, package, brand, unitprice_num,
unitprice, size, suppliername, suppliercity, validfrom)
    select i.StockItemID, i.StockItemName, c.ColorName, pt.PackageTypeName, i.Brand,
i.UnitPrice,
    case when UnitPrice<=13 then 'low' when UnitPrice < 32 then 'middle' else 'high'
end,
    i.Size, s.SupplierName, ci.CityName, i.ValidFrom
from test.stg.StockItems i join test.stg.PackageTypes pt on
i.OuterPackageID=pt.PackageTypeID
join test.stg.Suppliers s on s.SupplierID=i.SupplierID
join Application.Cities ci on ci.CityID=s.DeliveryCityID --ennek nem csináltunk
staging táblát
LEFT join test.stg.Colors c on c.ColorID=i.ColorID--227
select * from test..dim_stockitem

--BUYER dimenzióknak nem készítünk staging táblákat
create table test..dim_buyer (
    dw_buyerid int identity(10000,1) primary key, --kieg. kulcs
    buyerid int not null, --OLTP kulcs
    buyername nvarchar(100) not null,
    paymentdays int not null,
    category nvarchar(50) not null,
    buyercity nvarchar(50) not null,
    validfrom datetime not null,
    validto datetime null )
go
insert test..dim_buyer (buyerid, buyername, paymentdays, category, buyercity, validfrom)
select c.CustomerID, c.CustomerName, c.PaymentDays, cc.CustomerCategoryName, ci.CityName,
c.ValidFrom
    from Sales.Customers c join Sales.CustomerCategories cc on
c.CustomerCategoryID=c.CustomerCategoryID
join Application.Cities ci on ci.CityID=c.DeliveryCityID --5304
select * from test..dim_buyer

--ÖNÁLLÓ FELADAT:
--ténytábla 3 dimenzióval: dátum, buyer, stockizem
/* segítség:
create table test..fact_sales()
select od.OrderLineID, ol.OrderID, ol.Quantity*ol.UnitPrice paid, ol.UnitPrice,
o.OrderDate
from Sales.OrderLines ol join Sales.Orders o on o.OrderID=ol.OrderID
*/
--demo céljára: drop table test..fact_sales

```

```

create table test..fact_sales(orderlineid int, dw_stockitemid int, paid money, datum
datetime)

--VÁLTOZÁS-KEZELÉS CDC-ALAPON (INCREMENTAL LOAD)
_*****
use WideWorldImporters
--SQL Server Agent fusson
--ab CDC engedélyezése
exec sys.sp_cdc_enable_db
--exec sys.sp_cdc_disable_db
go
--ELŐKÉSZÍTÉS
set nocount on
--Customer tábla CDC engedélyezése
--exec sys.sp_cdc_disable_table @source_schema = 'warehouse', @source_name =
'stockitems', @capture_instance = 'capture_stockitems'
exec sys.sp_cdc_enable_table @source_schema = 'warehouse', @source_name = 'stockitems',
@role_name = 'CDC_role'
, @capture_instance = 'capture_stockitems' --az új szerep létrejön: gating role,
amihez hozzá kell adni azt, akinek a CDC táblákat látni engedjük
, @supports_net_changes = 1, @captured_column_list = null --minden mező változását
figyeljük
--2 job elindult
--együtt a cdc sémában (System tables) létrejöttek a log táblák
go
--a staging területen tároljuk a legutóbbi szinkronizálás záró LSN-jét:
--drop table test.stg.stockitems_cdc_lsn
create table test.stg.stockitems_cdc_lsn (lsn binary(10) not null, synch datetime not
null default(getdate()), status int not null default(0))
--csak egy rekord státusza lehet 0 (amelyik periódust nem dolgoztuk még fel)
go
--beírjuk az induló LSN-t
--delete test.stg.stockitems_cdc_lsn
insert test.stg.stockitems_cdc_lsn (lsn) values (sys.fn_cdc_map_time_to_lsn('largest
less than or equal', getdate()))
--(sys.fn_cdc_get_min_lsn('capture_stockitems'))
select * from test.stg.stockitems_cdc_lsn

--ide vesszük át a legutóbbi szinkronizálás óta történt változásokat
--drop table test.stg.stockitems_modified
create table test.stg.stockitems_modified (item_id int, package_id int, valid_from
datetime)
--ELŐKÉSZÍTÉS vége

--FELHASZNÁLÁS
--mi történt a legutóbbi frissítés óta?
select * from cdc.fn_cdc_get_net_changes_capture_stockitems((select lsn from
test.stg.stockitems_cdc_lsn where status=0),
sys.fn_cdc_map_time_to_lsn('largest less than or equal', getdate()), 'all')
--minden változás:
select * from cdc.fn_cdc_get_all_changes_capture_stockitems((select lsn from
test.stg.stockitems_cdc_lsn where status=0),
sys.fn_cdc_map_time_to_lsn('largest less than or equal', getdate()), 'all')
--a cdc.lsn_time_mapping tábla csak a naplózott tábla tranzakcióit tartalmazza
--most még mindkét tábla üres
go
--változások

```

```

select OuterPackageID from Warehouse.stockitems where StockItemID=1
update Warehouse.stockitems set OuterPackageID=9 where StockItemID=1 --7 volt, 9 lett
(carton)
update Warehouse.stockitems set OuterPackageID=6 where StockItemID=1
insert Warehouse.stockitems (StockItemID, StockItemName, SupplierID, UnitPackageID,
OuterPackageID, LeadTimeDays, QuantityPerOuter,
IsChillerStock, TaxRate, UnitPrice, TypicalWeightPerUnit, LastEditedBy)
values (500, 'test item', 12, 7, 7, 14, 1, 0, 20, 40, 1, 1)
select * from Warehouse.stockitems where StockItemID in (1,500)
go
select * from cdc.fn_cdc_get_all_changes_capture_stockitems((select lsn from
test.stg.stockitems_cdc_lsn where status=0),
sys.fn_cdc_map_time_to_lsn('largest less than or equal', getdate()), 'all') --2
sor, __$operation = 2 -> insert, = 4 -> update
go
select sys.fn_cdc_get_max_lsn()

--INDUL A SZINKRONIZÁLÓ SCRIPT
delete test.stg.stockitems_modified --teendők tábla törlése
declare @end_lsn binary(10) = sys.fn_cdc_get_max_lsn(), --
sys.fn_cdc_map_time_to_lsn('largest less than or equal', getdate()),
@start_lsn binary(10) = (select lsn from test.stg.stockitems_cdc_lsn where
status=0)
--nem ez az lsn kell, mert ezt már feldolgoztuk az előző futás
idején, hanem az ez után következő:
set @start_lsn=sys.fn_cdc_increment_lsn(@start_lsn)
if @start_lsn <= @end_lsn --csak akkor módosítunk, ha történt valami a legutóbbi
szinkron óta
--a start azért lehet nagyobb, mert a
cdc.lsn_time_mapping tábla csak a naplózott tábla tranzakcióit tartalmazza
--a módosult értékek kigyűjtése, most csak a csomagolást figyeljük
insert test.stg.stockitems_modified (item_id , package_id, valid_from)
select stockitemid, OuterPackageID,
sys.fn_cdc_map_lsn_to_time(__$start_lsn)
from cdc.fn_cdc_get_all_changes_capture_stockitems(@start_lsn, @end_lsn,
'all')
where __$operation = 4 --4: UPDATE, a __$update_mask mutatja, melyik mezők
változtak, dekódolás -> sys.fn_cdc_has_column_changed()
-----> ezt a staging táblát kell majd SCD2-vel feldolgozni egy későbbi
lépésben
--a legutóbbi frissítés óta történt több egymás utáni UPDATE is feldolgozható
--a log ablak lezárása
update test.stg.stockitems_cdc_lsn set status=1 --feldolgozott
insert test.stg.stockitems_cdc_lsn (lsn) values (@end_lsn)
--SZINKRONIZÁLÓ SCRIPT VÉGE

go
select * from test.stg.stockitems_modified -- package id = 9

--újabb változás
update Warehouse.stockitems set OuterPackageID=6 where StockItemID=1 --9 volt, 6 lett
update Warehouse.stockitems set OuterPackageID=7 where StockItemID=1

--A SZINKRONIZÁLÓ SCRIPT KÖVETKEZŐ FUTÁSA
delete test.stg.stockitems_modified --teendők tábla törlése
declare @end_lsn binary(10) = sys.fn_cdc_get_max_lsn(),

```

```

    @start_lsn binary(10) = (select lsn from test.stg.stockitems_cdc_lsn where
status=0)
    set @start_lsn=sys.fn_cdc_increment_lsn(@start_lsn)
if @start_lsn <= @end_lsn --csak akkor módosítunk, ha történt valami a legutóbbi
szinkron óta
    insert test.stg.stockitems_modified (item_id , package_id, valid_from)
        select stockitemid, OuterPackageID,
sys.fn_cdc_map_lsn_to_time(__$start_lsn)
        from cdc.fn_cdc_get_all_changes_capture_stockitems(@start_lsn, @end_lsn,
'all')
        where __$operation = 4 --4: UPDATE
update test.stg.stockitems_cdc_lsn set status=1 --feldolgozott
insert test.stg.stockitems_cdc_lsn (lsn) values (@end_lsn)
--SZINKRONIZÁLÓ SCRIPT VÉGE -> nem dolgoztunk fel egy változást kétszer
go
select * from test.stg.stockitems_modified -- package id = 6

--TAKARÍTÁS
exec sys.sp_cdc_disable_table @source_schema = 'warehouse', @source_name = 'stockitems',
@capture_instance = 'capture_stockitems'
exec sys.sp_cdc_disable_db
drop table test.stg.stockitems_cdc_lsn
--drop table test.stg.stockitems_modified
delete Warehouse.stockitems where StockItemID=500
update Warehouse.stockitems set OuterPackageID=7 where StockItemID=1

--TEMPORÁLIS TÁBLÁKKAL
-- a forrás-táblák már eleve verziózva vannak:
--...WITH ( SYSTEM_VERSIONING = ON ( HISTORY_TABLE = [Warehouse].[StockItems_Archive] ))

--ELŐKÉSZÍTÉS
create table test.stg.stockitems_temporal (synch datetime not null, status int not null
default(0))
--vigyázat! az időbélyeg UTC, mégpedig a szerver óráján:
select GETDATE(), GETUTCDATE() --1 óra különbség!!
insert test.stg.stockitems_temporal (synch) values (getutcdate())

--változások
update Warehouse.stockitems set OuterPackageID=7 where StockItemID=1
update Warehouse.stockitems set OuterPackageID=9 where StockItemID=1
--az insert nem látszik a temp. tábla naplójában

--INDUL A SZINKRONIZÁLÓ SCRIPT
go
delete test.stg.stockitems_modified
insert test.stg.stockitems_modified (item_id , package_id, valid_from)
select StockItemID, OuterPackageID, ValidFrom from Warehouse.StockItems where StockItemID
in ( --volt-e változás?
    select StockItemID from Warehouse.StockItems_Archive where ValidTo > (select synch
from test.stg.stockitems_temporal where status=0))
    --ilyenkor a legutolsó új érték a táblában van
    --ilyen módon nem tudunk több változást feldolgozni, és nem lehet mezőre
szűrni sem
update test.stg.stockitems_temporal set status=1
insert test.stg.stockitems_temporal (synch) values (getutcdate())
select * from test.stg.stockitems_modified --az eredmény a legutolsó változás
--SZINKRONIZÁLÓ SCRIPT VÉGE

```

```

--ÖNÁLLÓ FELADAT: a fenti script átírása úgy, hogy több változást is tudjon kezelni

--TAKARÍTÁS
update Warehouse.stockitems set OuterPackageID=7 where StockItemID=1
drop table test.stg.stockitems_temporal

/*****
A MÓDOSÍTÁSOK FELDOLGOZÁSA, SCD2
*****/
--a módosítások
use test
--tegyük fel, hogy egy rekord csak egyszer változott!
select * from test.stg.stockitems_modified --1 6      2024-03-05 19:16:14.450

--utolsó rekord lezárása:
update d set validto=sm.valid_from --nincs státusz mező
from stg.stockitems_modified sm join dim_stockitem d on d.stockitemid=sm.item_id
where d.validto is null --csak egy lehet

--új SCD2 rekord beszúrása:
insert dim_stockitem (stockitemid, itemname, color, package, brand, unitprice_num,
unitprice, size, suppliername, suppliercity, validfrom)
select top 1 d.stockitemid, d.itemname, d.color, pt.packagetyponame, d.brand,
d.unitprice_num, d.unitprice, d.size, d.suppliername, d.suppliercity, sm.valid_from
from stg.stockitems_modified sm join dim_stockitem d on d.stockitemid=sm.item_id -
-nem kell sorbarakni, mert a régi mezők úgyis ugyanazok
join test.stg.PackageTypes pt on sm.package_id=pt.PackageTypeID

--ÖNÁLLÓ FELADAT: mi van, ha egynél több SCD2 változás történik? (a fenti dim_stockitem
insert bővítése)

select * from dim_stockitem where stockitemid=1
--ezután a tényrekordokat a megfelelő verzióra kell irányítani
--tegyük fel, hogy ez a stg tábla tartalma,
--csak az új tényrekordok, amiket be kell szűrni
create table stg.fact_demo (orderid int, stockitemid int, quantity int, unitprice
money, orderdate datetime)
go
delete stg.fact_demo
insert stg.fact_demo (orderid , stockitemid , quantity , unitprice , orderdate)
values (50000, 1, 13, 300, '2024-03-04'), --ez még a változás előtti
(50001, 1, 13, 300, '2024-03-06') --ez már azutáni

--így szűrjük be:
insert fact_sales(orderlineid , dw_stockitemid , paid , datum )
select f.orderlineid, ds.dw_stockitemid, f.quantity*f.unitprice, f.orderdate
from stg.fact_demo f join dim_stockitem ds on f.stockitemid=ds.stockitemid and
(ds.validto is null and ds.validfrom <= f.orderdate) or --vagy az utolsó rekord,
vagy pontosan egy korábbi
(f.orderdate >= ds.validfrom and f.orderdate < ds.validto) --between nem jó, mert
>= ... <= lenne

--takarítás
delete fact_sales

--ÖNÁLLÓ FELADAT: az új dim és fact rekordok átvétele stg táblákba

```

```

--további javítás: követjük a stockitems_modified tábla változásait -> temporális tábla
--így utólag megnézhető, mikor milyen értékeket írtunk be a dim táblába
drop table test.stg.stockitems_modified
go
create table test.stg.stockitems_modified --kell kulcs
(
    change_id int identity (1,1) primary key,
    item_id int, package_id int, valid_from datetime,
    version_from datetime2 generated always as row start not null, --ezek nélkül nem
    lehet temporális
    version_to datetime2 generated always as row end not null,
    period for system_time (version_from, version_to))
with (system_versioning = on (history_table = stg.stockitems_modified_history));

```

Egy robosztus SCD-2 ETL megoldás áttekintése (esettanulmány)

A WWI adatbázis és a ráépülő adattárház tisztán temporális táblákra épülő inkrementális adatbetöltést valósít meg. Előny, hogy nem csak a változásokat, hanem az újonnan beszűrt dimenzió-tagokat is egységesen tudja kezelni. A WWI OLTP és DW adatbázisokban megvalósított ETL megoldás elemei:

- A WWI-DW-ban szinkronizációkat („lineage”) táblánként tartja nyilván az Integration.Lineage tábla, melyet az Integration.GetLineageKey eljárás kezel. Egy másik táblában (Integration.[ETL Cutoff]) a dimenzió-táblák utolsó frissítési idői („cutoff time”) vannak tárolva.
- A WWI OLTP adatbázisban minden, dimenzió forrásául szolgáló törzs-tábla temporális (például: Application.TransactionTypes). A ténytáblák forrásai viszont nem temporálisak.
- Az OLTP adatbázis tartalmaz egy-egy tárolt eljárást minden ilyen tábla változásainak a kigyűjtéséhez, például: Integration.GetTransactionTypeUpdates. Az eljárás két paramétert kap, a tól-ig időbélyegeket.
 - Az eljárás a temporális tábla és a hozzá tartozó napló-tábla uniójából kigyűjti azon változásokat, melyek érvényességének *kezdet*e a tól-ig tartományba esik (egy rekordhoz több változás is lehet). A legutolsó érték nincs már benne a napló-táblában. Egy újonnan beszűrt, de még nem változott rekord szintén nem jelenik meg a naplóban. A változott rekordok kulcsát és érvényességének *kezdete*t egy kurzorba teszi.
 - Végig iterál a kurzoron, és meghatározza az összes változás paraméterét (mi volt a változás után az új mezőérték). Az eredményeket egy ideiglenes táblában tárolja.
 - Kijavítja az ideiglenes táblában az érvényesség végét (ami a következő érvényesség kezdete)
 - Visszaadja, majd törli az ideiglenes táblát
- A WWI-DW-ban a dimenzió-táblák (például Dimension.[Transaction Type]) nem temporálisak, de SCD-2 re fel vannak készítve a kiegészítő kulccsal és a ValidFrom/ValidTo mezőkkel. Ebben a megoldásban **a ValidTo mindig ki van töltve**, az aktuális rekordnál a max. dátummal ('9999-12-31'). Ezen kívül a dimenziótábla minden rekordja tartalmazza a szinkronizáció azonosítóját (Lineage Key) is az esetleges audit céljára.
- A tárolt eljárás kimenetét, mint adatforrást használja fel a WWI-DW-ban lévő migrációs tárolt eljárás, feltételezve, hogy a kimenet másolása a staging táblába megtörtént (például az Integration.MigrateStagedTransactionTypeData eljárás az Integration.TransactionType_Staging táblát olvassa).

- Az eljárás a le nem zárt időintervallumokat (ValidTo='9999-12-31') a dimenzió táblában lezárja, tehát beállítja a ValidTo értéket a tagot ért változások kezdetének a minimumára, a staging tábla alapján.
- Majd beszúrja a staging tábla rekordjait a dimenzió-táblába.
- A dimenziót használó ténytábla módosítása ezután következik, ezt is tárolt eljárások végzik (például Integration.GetTransactionUpdates illetve Integration.MigrateStagedTransactionData). A helyes kiegészítő dimenzió-kulcsokat a tényrekordok beillesztése előtt beírja a staging táblába a dátumok alapján.

ÖNÁLLÓ FELADAT: módosítsunk 1-2 rekordot az OLTP-forrástáblán, ellenőrizzük a napló-táblát és futtassuk helyesen beállított idő-paraméterekkel a változás-gyűjtő eljárást. Ellenőrizzük a kimenetet. Majd ezt egy staging táblába mentve futtassuk a migrációs tárolt eljárást!

Megjegyzés: A staging tábla feltöltését úgy végezzük, hogy a WWI-DW-ben (távolról) futtatjuk a tárolt eljárást, és az eredményt közvetlenül a staging táblába írjuk. Ugyanis a staging tábla memória-optimalizált, ezért egy másik adatbázisból nem érhető el a szokásos módon.

További érdekes elemek a WWI OLTP adatbázisban:

- Figyeljük meg a szekvenciák használatát az IDENTITY helyett az OLTP adatbázisban. Ezeket külső adatbetöltés esetén be lehet állítani úgy, hogy a jelenlegi maximális utáni következő értékről induljanak (WideWorldImporters.Sequences.ReseedSequenceBeyondTableValues(), ReseedAllSequences())
- Tábla típus, mint UDT használata paraméterátadásra³. Például a Website.InsertCustomerOrders eljárás használja a Website.OrderLineList tábla típust.
- A sémák következetes használata minden objektumtípus esetén (tábla, eljárás, adattípus...)

Az ETL automatizálása

Az ETL lépéseit tipikusan automatizált, alacsony terhelésű időben futó jobok végzik el, az ember feladata „csak” a jobok megtervezése és az eredmény ellenőrzése.

A csomagot nem csak a helyi fájlrendszerre lehet lementeni, hanem az SQL serverre is, így más gépről is el lehet majd érni. Az SQL serveren tárolt csomagot egy tárolt eljárás el tudja indítani, a tárolt eljárást pedig lehet ütemezetten indítani egy job belsejében. Ennek részleteit lásd később⁴.

- <https://www.mssqltips.com/sqlservertip/2992/how-to-execute-an-integration-services-ssis-package-from-a-sql-server-stored-procedure/>

Az SSIS csomagokat egymásba lehet ágyazni az Execute Package Task nevű vezérlő csomópont révén, ezáltal modulárisan megvalósítható egy komplex ETL feladat. A master csomag paramétereit adhat át az elindított csomagoknak.

³ <https://learn.microsoft.com/en-us/sql/relational-databases/tables/use-table-valued-parameters-database-engine>

⁴ Bővebben:

<https://learn.microsoft.com/en-us/sql/integration-services/ssis-quickstart-run-tsql-ssms>

<https://docs.microsoft.com/en-us/sql/integration-services/ssis-quickstart-run-tsql-ssms?view=sql-server-ver15>

A Data Tools (Integration Services) használata

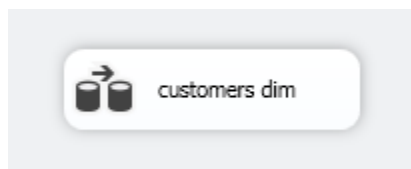
Komplex adatutak és transzformációk megvalósítására javasolt az SQL Server Data Tools (SSDT), mely adatfolyam-elvű grafikus programozást támogat⁵.

Minden projektben van legalább egy csomag (Package.dtsx). A csomag részei a kapcsolatmenedzserek (Connection managers) melyek lehetnek projekt- ill. csomag-szintűek, a **vezérlési út** (control flow) és az **adatút** (data flow).

PÉLDA: Visual Studio Data Tools indítása, SSIS csomag készítése a stg.person és stg.customer staging táblák feltöltésére, hibakezeléssel (a stg.customer tábla önálló munkában: **ÖNÁLLÓ FELADAT #11**). A kapcsolatmenedzsereket kézzel adjuk hozzá, projekt szinten. Mentsük el egy csomagba cust_dim_elokeszites.dtsx néven. Ez lesz a vezérlési út:

Az adatút egyszerű rekordmásolás egy forrásból egy célba:

PÉLDA: Építsük fel a dim_customer táblát most komplex adatútként az SSIS-ben, egy új csomagban, melynek neve cust_dim_dataflow.dtsx. Az eredmény egyetlen Data flow task:



Azonban a mögötte lévő adatfolyam már komplex:

Útmutatás:

- Az első Merge (inner join) a két staging táblát köti össze. Előtte azonos kulcs (a join kulcsa) szerint rendezni kell a forrásokat (merge join). A staging táblák forrásaiban minden mezőt kiválasztunk. A két „sort” csomópont beállítása (Person és Customer):

Az inner join:

A left joinban a territory táblából nem viszünk tovább minden mezőt:

Ahhoz, hogy a Fullname-et előállítsuk, be kell tenni egy új mezőt (Derived column):

⁵ Egy egyszerű SSIS ETL csomag létrehozása a FactCurrency ténytábla feltöltésére: <https://docs.microsoft.com/en-us/sql/integration-services/ssis-how-to-create-an-etl-package>

Control flow bevezetés <https://learn.microsoft.com/en-us/sql/integration-services/control-flow/control-flow>

Data flow bevezetés <https://www.red-gate.com/simple-talk/sql/ssis/ssis-basics-adding-data-flow-to-your-package/>

(A FirstName + " " + LastName is megfelelő)

Derived Column Name	Derived Column	Expression
Fullname	<add as new column>	(DT_STR,50,1252)FirstName + (DT_STR,50,1252)LastName

Végül a dim_customer-be írjuk az eredményt:

Az adatút lépései

Tipikus lépések:

- Adattisztítás (például hibás értékek eldobása)
- Normalizálás (a komplex adatszerkezetek, például egy XML struktúra lebontása)
- Adattípus-konverzió (például szövegből dátum típus)
- Átkódolás (például 1-> „Férfi”, 2-> „Nő”)
- Validálás (például negatív életkor javítása)
- Származtatott változók (**derived column**) számítása (például BMI, haszon, m/s->km/h stb.)
- Aggregálás
- Pivotálás

A több táblás lekérdezések, módosítások megvalósíthatók adatfolyam-transzformációs csomópontokkal is, például Lookup (törzstáblához, dim táblához csatolás), Merge Join (rendezett rekordhalmazokra). Adatút-kapcsoló transzformációk például a Conditional split, Multicast.

ÖNÁLLÓ FELADAT #12: Gyakorlásképpen készítsük elő a fact_internetsales tábla adatfolyam-alapú létrehozását egy új staging tábla készítésével, melyet a SalesOrderHeader és a SalesOrderDetail táblákból merge join illetve lookup művelettel hozunk létre!

Segítség az üres eredménytábla létrehozásához:

```
use AdventureWorks2016CTP3
select
soh.CustomerID, sod.ProductID as ProductKey, soh.OrderDate as DateKey,
sod.LineTotal as ItemPrice, sod.UnitPrice, sod.UnitPriceDiscount as DiscountAmount
into test.stg.internetsales
from Sales.SalesOrderHeader AS soh inner join Sales.SalesOrderDetail AS sod ON
soh.SalesOrderID = sod.SalesOrderID

use dw_test
select COUNT(*) from stg.internetsales
delete stg.internetsales
--ezzel elkészült az üres eredménytábla, kezdhethetjük az SSIS adatút-projekt megvalósítását
```

A fenti join elvégezhető a Lookup (lásd lejjebb) segítségével is.

Az adatfolyam **ütemezése** szempontjából egy transzformáció lehet:

- **Teljesen blokkoló**, például Sort, Group (ezt kell a legjobban meggondolni)
- **Részlegesen blokkoló** például Merge join, Pivot
- **Nem blokkoló** (egy-egy rekordot érintő átalakítás például Derived Column, Data conversion), SCD1-2, Lookup

Ha csak kiegészítő mezőket akarunk kivenni egy törzstáblából (vagy dim táblából) akkor a **Lookup** sokkal inkább megfelelő, mint a Merge Join, mert nem blokkoló, minden kulcshoz csak egy értéket (az elsőt) ad vissza. Az eredmény ugyanaz. **Azért fontos** a fentiek meggondolása, mert az adatfolyam-gráfon a végrehajtás sorrendjét egy hagyományos komplex SQL paranccsal ellentétben az ember tervezi!

A Lookup forrása lehet egy gyorsítótár (cache) is, mivel nem változik, és több csomag is használhatja. Ha a cache forrása egy Cache Connection Manager, akkor bármilyen forrása lehet, például akár szövegfájl is. A Lookup default esetben csak a sikeresen megtalált rekordokat engedi tovább, ezt felülírni a Specify How To Handle Rows With No Matching Entries drop-down list -> Ignore Failure beállítással lehet. Ezután a hiányzó értékek például egy Derived column csomópontban beállíthatók, egy ehhez hasonló kifejezéssel ISNULL(Gender) ? "N/A" : Gender. Megjegyzés: SQL-ben ez isnull(Gender, 'N/A') lett volna.

Természetesen a műveletek egy része elvégezhető a szerveren is SQL utasítások vagy tárolt eljárások formájában.

PÉLDA: építsük fel a dim_customer táblát most egyetlen Execute SQL Task felhasználásával, amelybe beírjuk a lekérdezést:

Futtassuk, az eredmény ugyanaz lesz.

Tanulság: Sok művelet, például a JOIN, megvalósítható adatfolyam-szinten és alacsonyabb (SQL-lekérdezés vagy script) szinten is. Az adatfolyam jól átlátható, paraméterezhető, és támogatja a **heterogén komponenseket** is (például beolvasás szövegfájlból). Azonban az adatfeldolgozás rekordonként történik, ami nagy táblák esetén nem megfelelő hatékonyságú lehet. Ilyen esetben célszerű a vezérlési útba ágyazott SQL utasítások alkalmazása.

PÉLDA: Készítsünk másolatot a csomagról, majd egészítsük ki a dim_customer táblát egy szövegfájlban kapott adatokkal. Olvassuk be a kiegészítő vásárlói információkat a CustomerInformation_01-03.txt fájlból egy Flat File Data Source elem segítségével, és csatoljuk hozzá a meglévő mezőkhöz!

A vezérlés tehát változik, bekerül az egész elé a stg.customerinfo tábla betöltése a szövegfájlból. Mielőtt elkezdenénk, létre kell hoznunk az adatbázisban a stg.customerinfo táblát, hogy a tervező tudja használni:

```
CREATE TABLE stg.customerinfo(
    [PersonID] [int] NULL,
    [EnglishEducation] [nvarchar](30) NULL,
    [EnglishOccupation] [nvarchar](50) NULL,
    [BirthYear] [int] NULL,
    [Gender] [nchar](5) NULL,
    [MaritalStatus] [nchar](5) NULL,
    [EmailAddress] [nvarchar](50) NULL
)
GO
delete stg.customerinfo
go
```

Töltsük be a korábban létrehozott dtsx csomagot, ha szükséges, pótoljuk a kapcsolatmenedzsereket és állítsuk be a forrás-csomópontokat, hogy ezeket használják. Az új vezérlési szerkezet:

Az Insert customer info nevű data flow task belső szerkezete:

A származtatott mező azért kell, mert ha belenézünk a txt fájlba, látható, hogy hiányoznak a sorvégek és a dátumok is rosszak, csak az évet lehet használni:

```
PersonID;EnglishEducation;EnglishOccupation;BirthDate;Gender;MaritalStatus;EmailAddress
300;Bachelors;Professional;1966408;M;M;jon24@adventure-works.com
301;Bachelors;Professional;1965514;M;S;eugene10@adventure-works.com
302;Bachelors;
303;Bachelors;Professional;1968215;F;S;christyl2@adventure-works.com
304;Bachelors;Professional;1968808;F;S;elizabeth5@adventure-works.com
305;Bachelors;Professional;
```

De ez nem baj, mert a Flat File forrás-menedzser fel van készítve az ún. tépett szélű (**ragged-right**) formátumú fájlok kezelésére. A „ragged-right” szerkezetben hiányozhatnak a nem szükséges pontosvesszők a CSV fájlból. Az új Flat File menedzserben válasszuk a Suggest types opciót az Advanced fülön, 20000 rekord alapján. Azért ilyen sok rekordot olvastunk be, hogy ne legyenek túl rövidek az automatikusan beállított maximális varchar értékek. Ellenőrizzük az automatikus adattípusokat és a tartalmat (Preview). A BirthDate 4 bájtos egészként lesz importálva.

A Derived column definíciójában a Birthdate egész számot át kell konvertálni stringgé, levágni az elejét, majd visszakonvertálni egészzé a (DT_UI2)SUBSTRING((DT_STR,10,1252)BirthDate,1,4) kifejezéssel:

Derived Column Name	Derived Column	Expression	Data Type
birth_year	<add as new column>	(DT_UI2)SUBSTRING((DT_STR,10,1252)BirthDate,1,4)	two-byte u

Az SSIS adattípusokról illetve konverziós függvényekről lásd bővebben⁶. Következik a plusz mezők beírása, melyhez használhatnánk Merge elemet, de ennél hatékonyabb a Lookup. Így fog kinézni az adatút:

A Lookup csomópont Connection fülén be kell állítani, hogy a stg.customerinfo táblában keressen, utána a Columns fülön a külső kulcsot:

A Lookup-nak 2 kimenete van, egyik irányba mennek azok a rekordok, melyeknél volt találat, a másikba azok, melyeknél nem. Mi most csak az elsőt használjuk (Lookup Match Output), ezt írjuk a kimenetre. Ez így egy INNER JOIN-nak (equijoin) felel meg. Ahhoz, hogy futás közben ne álljon le a csomag hibával az első hiányzó PersonID-nél, a Lookup General fülén a „Specify how to handle no matching entries” értékét „Redirect Rows To No Match Output”-ra állítsuk.

Ha meg akarnánk tartani azokat a vásárlókat is, akikhez nem találtunk illeszkedő rekordot a stg.customerinfo táblában (LEFT JOIN), akkor a fenti „Specify how to handle no matching entries” beállítás „Ignore failure” lenne.

⁶ <https://learn.microsoft.com/hu-hu/sql/integration-services/data-flow/integration-services-data-types>

Futtassuk le a csomagot és ellenőrizzük az eredményt! A dim táblában már csak 17120 rekord van, a többihez nem találtunk illeszkedő rekordot a stg.customerinfo táblában.

Haladó SSIS megoldások

Modularitás biztosítása konténerekkel és beágyazott csomagokkal: a vezérlési útban **szekvenciális konténerek** segítségével átláthatóbbá tehetjük a komplex csomagok szerkezetét. Példa: WWI ETL csomag. Egy csomagból egy másik csomagot is el lehet indítani az Execute Process Task segítségével. Az elindított csomagban elérhetőek a szülő csomag változói és paraméterei.

A felhasználó által definiált **változóknak** programozottan, futásidőben adunk értéket. Az érték futás közben változhat, általában skaláris, de lehet rekordhalmaz is, a hatókörük lehet a csomag, de egyetlen task-ra is korlátozott. A változók kívülről nem láthatóak, nem beállíthatóak. Példa: a rendszeridő tárolása a WWI ETL-csomagban a User::TargetETLCutoffTime változóban, a beolvasott fájlok száma, stb. Dinamikus SQL futtatható úgy, hogy egy Execute SQL Task SQLSourceType tulajdonságát 'Variable' értékre állítjuk, és az utasítást tartalmazó szöveges változót megadjuk a SourceVariable alatt.

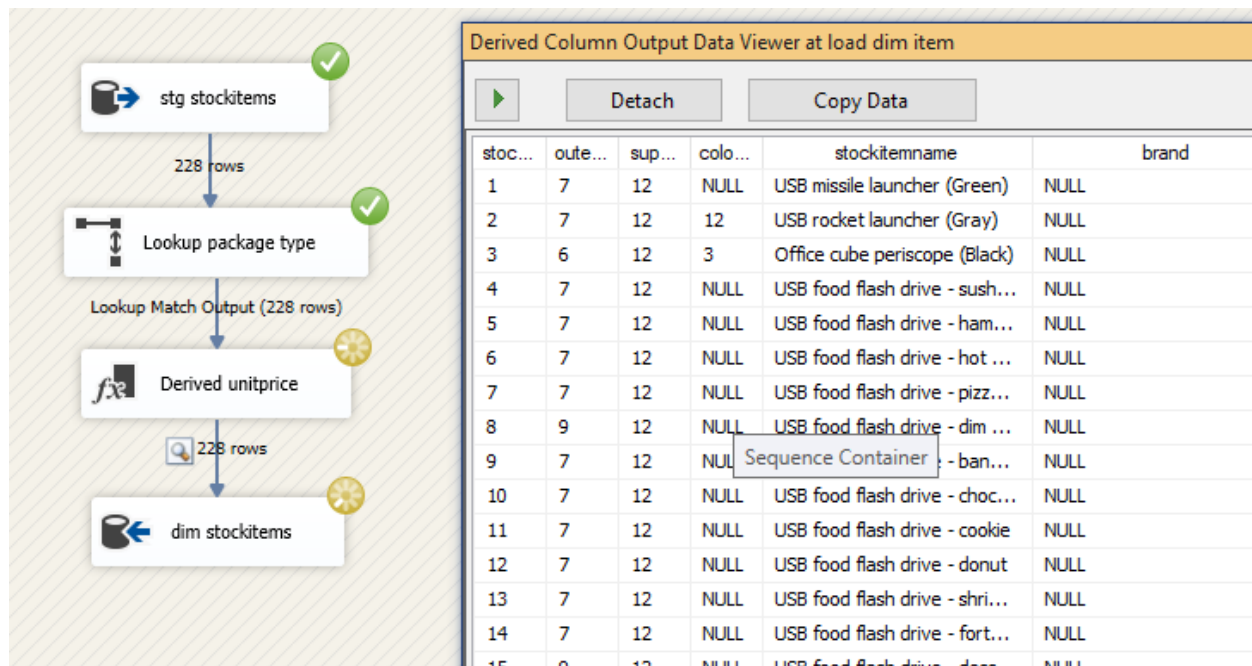
A változókat a vezérlési út szerkesztő felületén (alul) lehet deklarálni, futás közben pedig egy Expression Task segítségével lehet nekik értéket adni, pl. @[User::TableName] = "stg.Products". (vigyázat, SSIS-szintaxis).

A projekt- és csomag-paraméterek olyan speciális projekt- vagy csomag-szintű változók, melyeknek a futtatás **előtt** lehet értéket lehet adni, és ez a futás alatt nem változhat. A paraméterek értékét tipikusan nem lehet automatikusan meghatározni, kézzel kell beállítani. A paraméter lehet opcionális vagy kötelező. Példa: a csomag működési módját jelző szám. Másképp működhet a csomag fejlesztés közben, mint az éles környezetben.

A változókkal és paraméterekkel lehet szabályozni a vezérlési utat, ha két Task közötti vezérlési nyíl Evaluation operation tulajdonságát „Expression **and** Constraint”-ra állítjuk: ezután a vezérlés csak akkor halad a nyíl mentén, ha a modalitás (pl. Success) teljesül ÉS a kézzel megadott, változóra épülő feltétel is teljesül. Hasonlóképp lehetséges az „Expression **or** Constraint” is.

PÉLDA csomagparamétertől függő funkcionalításra: vagy a test sémában töröljük és töltjük fel a dimenzió táblát, vagy a dbo sémában

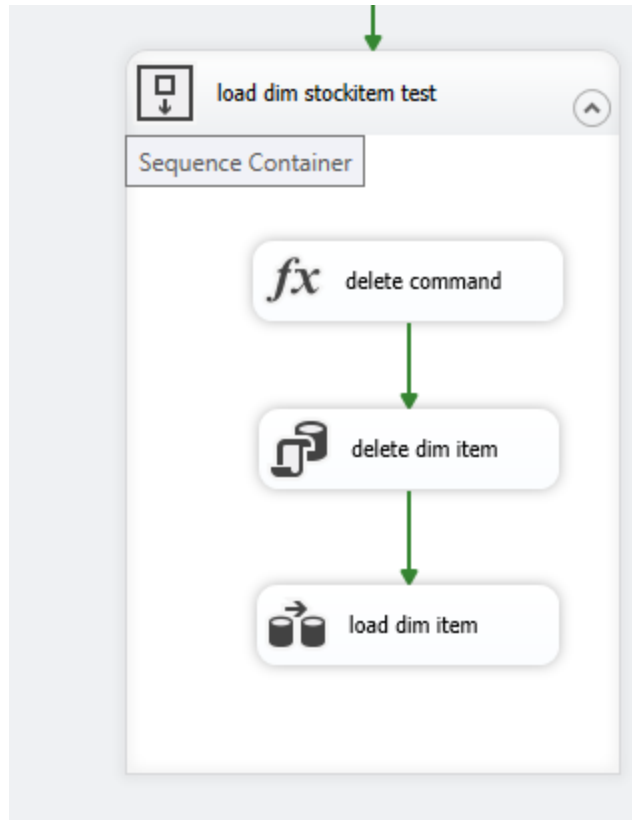
1. Hozzuk létre a tesztelési célú dimenzió táblát a test sémában.
2. Nyissuk meg a dim_stockitem dimenzió táblát létrehozó csomagot
3. Alkalmazzunk egy Data viewer-t az adatút ellenőrzésére (az adatútban a csomópontok közti kapcsolaton Enable data viewer)



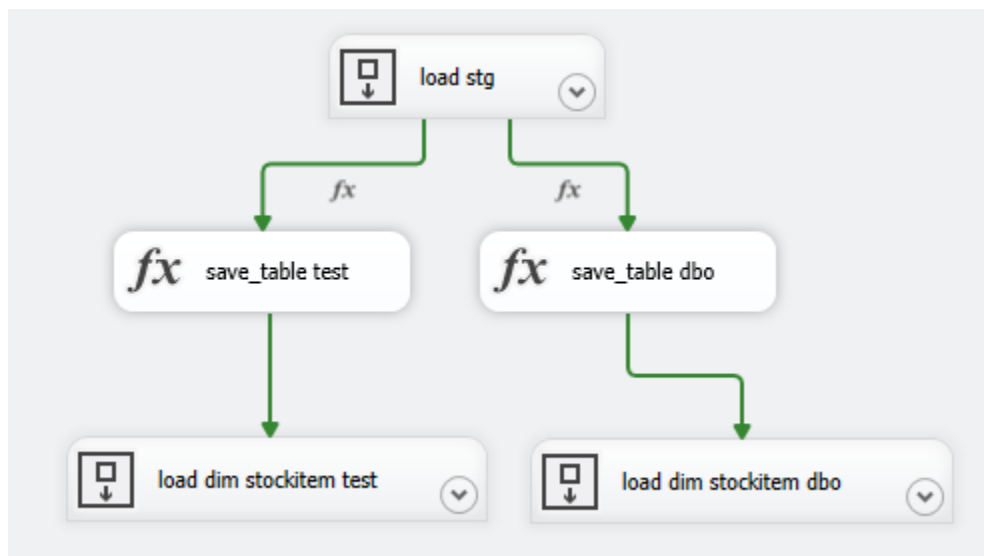
- Vezessünk egy kötelező csomag-paramétert: dev_mode. Ha ez 1, akkor a test sémában töröljük és töltjük fel a dimenzió táblát, ha 0, akkor a dbo sémában
- Vezessünk be string típusú egy csomag-változót save_table néven, kiinduló értéke: test.dim_stockitem, és egy delete_command nevű változót

Variables			
Name	Scope	Data type	Value
save_table	Package	String	test.dim_stockitem
delete_command	Package	String	

- Tegyük be egy szekvenciális konténerbe tesszük a staging táblák feltöltését (load stg), egy másikba a dimenzió tábla feltöltését (load dim stockitem)
- Az első konténer kimenete után tegyük be két párhuzamos Expression taszkot, melyek közül az egyikhez a @\$Package::dev_mode==1 feltétel teljesülése esetén, a másikhoz a @\$Package::dev_mode==0 esetén vezessen a vezérlés (a vezérlési nyíl megfelelő beállításával). Az egyikbe @[User::save_table] = "test.dim_stockitem", a másikba @[User::save_table] = "dbo.dim_stockitem" kerüljön.
- A második konténerben az Execute SQL (delete) taszk elé tegyük be egy Expression taszkot: @[User::delete_command] = "delete " + @[User::save_table], és a delete taszk SQLSourceType tulajdonságát állítsuk át „Variable” értékre, a delete_command változóra.



9. A konténerben a dim táblába mentést végző OLEDB destination csomópont Data access mode tulajdonságát állítsuk „Table name variable” értékre, és rendeljük hozzá a save_table változót.
10. Duplázzuk meg a konténert, és az egyiket kössük az egyik, a másikat a másik Expression kimenetére.



11. Teszteljük a csomagot.

A CDC támogatása

A CDC-vel követett táblák változásokövetését a CDC Control Task nevű SSIS taszk támogatja a vezérlési útban. Ez az AT erre a célra fenntartott táblájában tárolja az előző CDC-olvasás állapotát (mi volt az előző művelet, mettől-meddig LSN értékek), majd az adatút sikeres futtatása után ide írja vissza az új állapotot. Az alábbi ábrán ez a tábla a `dbo.cdc_states`.

CDC Control Task Editor

SQL Server CDC database ADO.NET connection manager:
LocalHost.AdventureWorks2019 New...

CDC control operation:
Get processing range
Mark initial load start
Mark initial load end
Mark CDC start
Get processing range
Mark processed range New...

current LSN:
[Empty text box]

☒ Automatically store state in a database table

Connection manager for the database where the state is stored:
LocalHost.AdventureWorks2019 New...

Table to use for storing state:
[dbo].[cdc_states] New...

State name:
CDC_State

OK Cancel Help

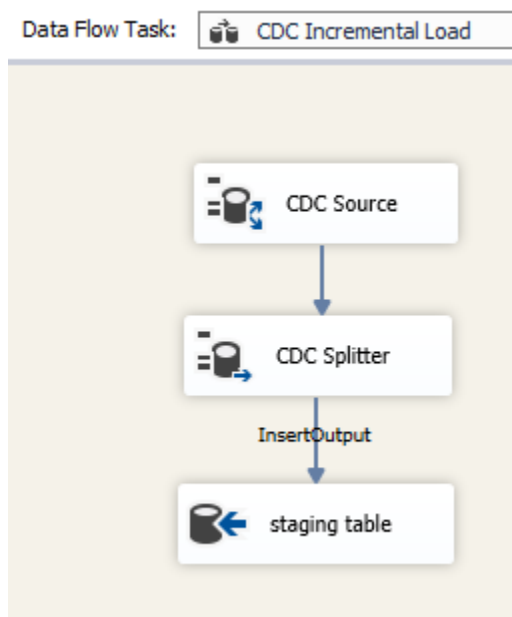
Az állapot paraméterei futás közben az adatút számára egy csomag-változóban érhetők el. A taszk tipikus felhasználása az első betöltésre:

1. Mark initial load start
2. staging tábla feltöltése az adatútban
3. Mark initial load end (=a maximális LSN tárolása az állapot-táblában)

Ezután több teljes feltöltésre (initial load) már nem lesz szükség, csak a változások követésére (incremental load):

1. Get processing range (=az előző olvasás végétől a mostani időpontig)
2. az adatútban külön feldolgozni a beszúrás/módosítás/törlés változásokat
3. Mark processed range (=a feldolgozott LSN-tartomány tárolása az állapot táblában).

Az adatútban egy speciális kapcsolatmenedzser, a CDC Source manager a csomagváltó alapján kiválasztott LSN-tartományban olvassa és dekódolja az OLTP adatbázis CDC rendszertábláit, és minden változott rekordot továbbad egy CDC splitter csomópontnak, amely a változás típusától függően bontja szét az adatfolyamot 3 részre (INSERT/UPDATE/DELETE). Ha csak a beszűrt rekordokkal szeretnénk foglalkozni, akkor így néz ki az adatút:



Az ETL csomag tárolása és futtatása

Az SSIS csomagokat, sőt egész IS projekteket verziókezeléssel együtt is lehet az SQL Serveren tárolni az Integration Services Catalogs adatbázisban bináris formában („deployment”)⁷. Lépések:

1. Integration Services Catalogs -> Create Catalog (Enable CLR integration kiválasztandó). Meg kell adni egy jelszót, mert a csomagokat titkosítják. Ekkor létrejön egy SSISDB nevű adatbázis. Ebben az adatbázisban mappákat lehet létrehozni az egyes projekteknek.
2. A kész projektet a VSDT-ben megnyitva a projekt Deploy menüjét kell választani, majd megadni, hogy az Integration Services Catalogs melyik mappájában szeretnénk tárolni ezt a projektet, az összes csomagjával együtt.
3. A csomagot kézzel el lehet indítani a grafikus felületen a csomag pop-up menüjében az Execute utasítással. Indítás előtt be lehet állítani a csomag paramétereit (ha vannak). A csomag végrehajtásáról taskokra lebontott jelentést lehet készíteni:


⁷ <https://www.sqlshack.com/deploying-packages-to-sql-server-integration-services-catalog-ssisdb/>

Execution Information

Operation ID	2
Package	komplex_adatut\2024_marc_11_komplex_adatut_teljes\dim_customer_init_load_teljes.dtsx
Environment	-
Status	Succeeded
Machine	DESKTOP-GT5C1GK

Duration (sec)	29.407
Start Time	3/11/2024 6:31:46 PM
End Time	3/11/2024 6:32:16 PM
Caller	DESKTOP-GT5C1GK\db

Execution Overview

 Filter Result: All: (3 more)

Result	Duration (sec)	Package Name	Task Name	Execution Path
Succeeded	13.985	dim_customer_init_load_teljes.dtsx	dim_customer_init_load_teljes	\dim_customer_init_load_teljes
Succeeded	1.438	dim_customer_init_load_teljes.dtsx	delete dim customer	\dim_customer_init_load_teljes\delete dim customer
Succeeded	1.937	dim_customer_init_load_teljes.dtsx	delete stg customer info	\dim_customer_init_load_teljes\delete stg customer info
Succeeded	7.75	dim_customer_init_load_teljes.dtsx	load dim customer	\dim_customer_init_load_teljes\load dim customer
Succeeded	2.235	dim_customer_init_load_teljes.dtsx	load stg customer info	\dim_customer_init_load_teljes\load stg customer info

Parameters Used

Name	Value	Data
CALLER_INFO		String
DUMP_EVENT_CODE	0	String
DUMP_ON_ERROR	False	Boolean
DUMP_ON_EVENT	False	Boolean
LOGGING_LEVEL	1	Int32
SYNCHRONIZED	False	Boolean

Property Overrides

Property Path	Property Value
---------------	----------------

- Az egyes csomag-végrehajtások összefoglaló adatait az SSIS irányítópultján lehet megnézni: SSISDB -> Reports -> Standard reports -> Intergration Services Dashboard
- Ha ütemezni szeretnénk a csomag automatikus végrehajtását, akkor egy jobot kell létrehozunk, ebben a lépés típusát SSIS Package-re kell állítani. A csomag paramétereit, kapcsolatmenedzsereit itt is be lehet állítani.
- A csomagot egy tárolt eljárás belsejéből is el lehet indítani⁸.

PÉLDA: mentsük az SSISDB-be a dim_stockitem csomagot, paraméterezzük, készítsünk rá percenként futó jobot, és teszteljük különböző paraméterekkel

Ha rendelkezésünkre áll egy bináris (*.ispac) formátumú SSIS-projekt, azt tervezésre a VSDT-ban az „Integration Services Import Project Wizard” projekt-típus kiválasztásával lehet megnyitni. Ugyanígy lehetséges egy projekt megnyitása deployment után az SSISDB-ből.

PÉLDA: nézzünk bele a WWI ETL csomag belsejébe (Wideworld_IIS_package_Daily.ETL.ispac)

Gyakorló feladatok

Készítsen egyetlen ténytablát tartalmazó adattárházat staging séma felhasználásával az alábbi adatokból!

- Tervezze meg a dimenziótblák és ténytblák szerkezetét, értelemszerűen a lényeges tényeket és attribútumokat felhasználva! Legalább egy dimenzió használjon kiegészítő kulcsot, és legalább egy esetben alkalmazzon adat-transzformációt új attribútum létrehozásához (kategorizálással vagy egyéb számítással)!
- Készítse el a staging táblákat, a dimenzió-táblákat és a ténytablát létrehozó SQL scriptet! Értelmes, magyar nyelvű tábla- és mezőneveket használjon!
- Készítse el és tesztelje a táblákat létrehozó SQL scriptet!
- Készítse el és tesztelje a ténytablát vagy az egyik dimenzió-táblát létrehozó SSIS-csomagot!

⁸ <https://learn.microsoft.com/en-us/sql/integration-services/ssis-quickstart-run-tsql-ssms>

Ténytábla forrása	Dimenziótáblák forrása
Purchasing.PurchaseOrderDetail	Purchasing.PurchaseOrderHeader, HumanResources.Employee, Purchasing.ShipMethod, Purchasing.Vendor
Production.WorkOrder	Production.ScrapReason, Production.Product
Production.ProductInventory	Production.Location, Production.Product
Production.BillOfMaterials	Production.Product, Production.UnitMeasure

MELLÉKLET: Megvalósítási technikák

Hagyományos (B-tree) indexek

Amikor az adattárházból lekérdeznek, a ténytáblát JOIN-nal kell összefűzni a dimenziókkal, amit jól segít az index. Néhány tény a **klaszterezett indexekről**:

- Mivel fizikai rekordsorrendet ír elő, ezért táblánként egy lehet
- A technológia B-tree
- Jó, ha az alapja egyedi, rövid (kis méretű index lesz) és folyamatosan növekvő
- A kiegészítő kulcsok ideális alapot szolgáltatnak
- A tábla kulcsához automatikusan készül index
- A lefedő index (covering index): olyan több mezős index, amely egy lekérdezéshez szükséges összes mezőt tartalmazza, ezért a táblát már nem is kell elővenni

A dimenzióhoz adott további indexek a lekérdezés szűrésénél lehetnek hasznosak—feltéve, hogy eléggé **szelektívek** (például cégnév vagy vásárlónév), és csak akkor, ha tényleg várható olyan lekérdezés, amely használja is őket. A betöltés gyorsítható, ha a nem klaszterezett indexeket a művelet előtt töröljük, majd utána újból létrehozuk: DROP/RECREATE. A kis méretű ténytábla külső kulcsaihoz általában nem kell nem klaszterezett indexet készíteni.

Nagy méretű ténytáblákhoz az oszloptár-index javasolt (lásd alább).

Az aggregált adattárház-lekérdezésekhez hasznos lehet az **indexelt view** (materialized view), ami nem igényel új táblát, viszont a Query Optimizer automatikusan fel tudja használni

Példa: Materialized view demo

```
use AdventureworksDW2016CTP3
select count(*) from FactInternetSales --60398
GO
SET STATISTICS IO ON;
GO
SELECT ProductKey,
SUM(SalesAmount) AS Sales,
COUNT_BIG(*) AS NumberOfRows
FROM dbo.FactInternetSales
GROUP BY ProductKey;
GO
--1246 logical reads

CREATE VIEW dbo.SalesByProduct
WITH SCHEMABINDING AS --különben nem indexelhető
--a hatása: az itt szereplő objektumok meta-adatai (sémája) nem változhatnak
--ezért a szervernek nem kell azt ellenőriznie, gyorsabban fog futni
SELECT ProductKey,
SUM(SalesAmount) AS Sales,
COUNT_BIG(*) AS NumberOfRows
FROM dbo.FactInternetSales
GROUP BY ProductKey;
GO
CREATE UNIQUE CLUSTERED INDEX CLU_SalesByProduct --indexed view
ON dbo.SalesByProduct (ProductKey);
GO
SELECT ProductKey,
SUM(SalesAmount) AS Sales,
COUNT_BIG(*) AS NumberOfRows
FROM dbo.FactInternetSales
GROUP BY ProductKey;
```

GO

--2 logical reads

Oszloptár (columnstore) indexek

A nagy tény táblák indexelésére ajánlott. Jellemzői:

- Több mezős lehet, de a mezőket külön-külön tárolja, ezért ha az egész rekord kell (nem lefedett a keresés), akkor az index keresése után a sort rekonstruálni kell
- Előny, hogy az egyes oszlopok indexei jól tömöríthetők, ezért kevés I/O-t igényel az index betöltése, és csak azt az indexet kell betölteni, melyre keresési feltétel van
- Akár az összes (lényeges) mezője benne lehet a tény táblának
- Nincs fizikai sorba rendezés
- Szegmensekben tárolják, minden szegmens fejlécében a minimális és maximális érték minden mezőre-> gyorsan kereshető
- Az index tömöríthető
- Táblánként csak egy oszloptár index lehet
- Particionált tábla esetén a partíciókhoz kell igazítani
- Az index elkészítése után a tábla csak az index újrakészítésével módosítható (read-only lesz)
- Továbbiak: <https://logicalread.com/sql-server-columnstore-index-w02>

Ha a hely kevés az adattárházban

Előfordulhat, hogy tömöríteni kell:

- Rekord-alapú tömörítés (row compression) A fix méretű adattípusú mezőket is változó méreten tárolja, minden rekord kisebb lesz. Általános célú, akár OLTP-ben is használható.
- Lap-tömörítés (page compression) Laponként egyszer tárolja a gyakori mezőérték-prefixeket (például egy tény táblában sokszor előfordul ugyanaz a külső kulcs)
- Unicode-tömörítés: Nvarchar, nchar esetén a 2 byte helyett csak egyet használ (például nagy méretű dim. táblák szöveges mezőire)

PÉLDA: a tény tábla tömörítése és oszloptár-indexelése. Az index lekérdezést gyorsító hatása a kisméretű tény tábla miatt nem demonstrálható.

```
EXEC sp_spaceused 'dbo.InternetSales', @updateusage = 'TRUE' --frissíti a statisztikákat
--dbo.InternetSales      60398
--reserved:              3208 KB
--data: 3064 KB          16 KB  128 KB
ALTER TABLE dbo.InternetSales REBUILD WITH (DATA_COMPRESSION = PAGE)
GO
EXEC sp_spaceused 'dbo.InternetSales', @updateusage = 'TRUE'
--dbo.InternetSales      60398
--reserved              1096 KB
--data: 1000 KB          16 KB   80 KB

--columnstore index demo
CREATE COLUMNSTORE INDEX CSI_InternetSales
ON dbo.InternetSales
(InternetSalesKey, CustomerDwKey, ProductKey, DateKey,
OrderQuantity, SalesAmount, UnitPrice, DiscountAmount) --az összes mező!
go
EXEC sp_spaceused N'dbo.InternetSales', @updateusage = 'TRUE';
--reserved: 1616 KB    data: 1000 KB    index: 424 KB  192 KB
```

Tehát az index nagyobb, mint korábban (424>16), de ez egy **teljesen lefedő index**!! A lekérdezéshez a táblára már nincs is szükség... (index mérete: 424 < tábla mérete: 1096).

Egy tipikus adattárház-lekérdezés:

```
SELECT C.CountryRegion, P.CategoryName, D.CalendarYear, SUM(I.SalesAmount) AS Sales
FROM dbo.InternetSales AS I INNER JOIN dbo.Customers AS C ON I.CustomerDwKey = C.CustomerDwKey
INNER JOIN dbo.Products AS P ON I.ProductKey = p.ProductKey
INNER JOIN dbo.Dates AS d ON I.DateKey = D.DateKey
GROUP BY C.CountryRegion, P.CategoryName, D.CalendarYear
ORDER BY C.CountryRegion, P.CategoryName, D.CalendarYear
--a lekérdezést kijelölve Ctl-L (estimated exec plan) Ctl-M (actual exec plan)
--ha a CSI index nélkül próbáljuk, figyelmeztet, hogy hiányzik egy index!
```

Összefoglalva, a lekérdezések gyorsítására 3-féle lehetőség van:

- A lekérdezésre indexelt view készítése
- A fő forrástáblára oszloptár-index készítése
- A hagyományos módszer: a lekérdezésben érintett minden külső kulcsra nem-klaszterezett B-fa index készítése

A konkrét lekérdezéstől és az alkalmazástól sok függ.