# IW4
# Microsoft SQL Server 2008

Pro vývojáře databází

David Gešvindr
MCT | MSP | MCITP


Osnova je orientační pro FIT, u FEKTu se dá předpokládat, že budou zohledněny předchozí znalosti studentů, kde většina s databází nikdy přímo nepracovala.

# Organizační záležitosti

- Rozdělení kurzu na 3 tématické bloky
- Lektoři: David Ešvindr, Martin Poisel

- SQL: 5 cvičení

- Možnost získat 40 bodů za cvičení
- Zkouška na 60 bodů

- http://www.fit.vutbr.cz/study/courses/IW1/public

# Základní pojmy

„Bez kterých se neobejdete“

# Základní pojmy

- *Databáze = strukturovaná perzistentní data využívaná aplikačními systémy*
- Další vlastnosti:
  - *Sdílená*
  - *Bezpečná*
  - *Integrita dat*

# Základní pojmy

- Základní úrovně abstrakce dat:
  - Fyzická úroveň
  - Logická úroveň
  - Úroveň pohledů
- *Relační model = na logické úrovni jsou data strukturována do tabulek*
- *Schéma databáze = metainformace popisující data v databázi*

# Základní pojmy

- DDL = Data Definition Language
- DML = Data Manipulation Language
- DCL = Data Control Language

- T-SQL = Transact SQL

# Základní pojmy

- Kandidátní klíč
- Primární klíč
- Cizí klíč
- Relace mezi tabulkami

# Návrh databáze

# Návrh databáze

- Pomocí ER-diagramu
- Pomocí normalizace

# ER-diagram

- Diagram popisující vztahy mezi entitami a jednotlivé entity
- Entita = reální objekt abstrahovaný dle našich potřeb
- Relace = vztah mezi entitami
  - 1:1
  - 1:N
  - M:N – nelze realizovat přímo, 2x 1:N

# Úkol 1: ER-diagram Projekty

# *Zadání*

**PROJEKTY**

Uvažujte, že máte zakázku na vytvoření databázové aplikace (příp.modulu většího systému), která bude spravovat informace o projektech, řešených u zadavatele. Dosud zadavatel vedl příslušnou agendu v papírové podobě na třech typech formulářů, které vám dal k dispozici pro datovou analýzu. Všechny tři formuláře mají stejnou hlavičku, která obsahuje tyto údaje: číslo projektu, název projektu, jméno, osobní číslo a název oddělení zodpovědného řešitele. Zodpovědný řešitel je jeden z řešitelů, jeden řešitel může řešit i zodpovídat za více projektů. **Každý formulář obsahuje kromě hlavičky tabulku**, která má tvar:

**Formulář „Základní údaje o projektu"- udává zdroje financování projektu.**

Ve sloupci 'Finanční zdroj', je udedena jednoznačná zkratka zdroje, např.MŠMT (musí existovat číselník možných zdrojů). Poslední řádek udává celkovou částku, která představuje množství peněz, které má projekt k dispozici (tj. rozpočet).

| Poř.číslo | Finanční zdroj | Částka |
|---|---|---|
| ..... | ..... | ..... |
| Rozpočet | | součet ve sloupci |

**Formulář „Řešitelé projektu" představuje seznam řešitelů.**

'Doba řešení' udává dobu, kterou odpracoval daný řešitel na daném projektu a 'Mzdové náklady' mzdu, kterou za práci na projektu dostal.

| Poř.číslo | Os.číslo | Jméno | Oddělení | Doba řešení | Mzdové náklady |
|---|---|---|---|---|---|
| ..... | ..... | ..... | ..... | ..... | ..... |

**Formulář „Investice projektu"**

Investicí se rozumí nakoupený produkt, jehož cena přesahuje určitou výši.

Ve sloupci převzal je podpis některého z řešitelů.

| Poř.číslo | Faktura č. | Datum | Částka | Převzal |
|---|---|---|---|---|
| ..... | ..... | ..... | ..... | ..... |

Poznámka: Zápis „....." v řádku tabulek značí, že v tabulce může být několik takových řádků.

Nakreslete ER diagram, který bude reprezentovat výše uvedené požadavky.

Autor zadání je doc. Ing. Jaroslav Zendulka, materiály k předmětu IDS, FIT


Autorem řešení je David Gešvindr a Pavla Kůrková

# Rekapitulace zadání

- Uvažujte, že máte zakázku na vytvoření databázové aplikace, která bude spravovat informace o projektech, řešených u zadavatele. Dosud zadavatel vedl příslušnou agendu v papírové podobě na třech typech formulářů.
- Všechny formuláře mají stejnou hlavičku, která obsahuje tyto údaje: číslo projektu, název projektu, jméno, osobní číslo a název oddělení zodpovědného řešitele. Zodpovědný řešitel je jeden z řešitelů, jeden řešitel může řešit i zodpovídat za více projektů.

# „Základní údaje o projektu“

- Udává zdroje financování projektu
- Ve sloupci 'Finanční zdroj' je jednoznačná zkratka zdroje (musí existovat číselník možných zdrojů)
- Poslední řádek udává celkovou částku, kterou má projekt k dispozici (tj. rozpočet)

| Poř. číslo | Finanční zdroj | Částka |
|---|---|---|
| ..... | ..... | ..... |
| Rozpočet | | Součet ve sloupci |

# „Řešitelé projektu“

- Představuje seznam řešitelů
- 'Doba řešení' udává dobu, kterou odpracoval daný řešitel na daném projektu
- 'Mzdové náklady' udávají mzdu, kterou řešitel za práci na projektu dostal

| Poř. číslo | Os. číslo | Jméno | Oddělení | Doba řešení | Mzdové náklady |
|---|---|---|---|---|---|
| ..... | ..... | ..... | ..... | ..... | ..... |

# „Investice projektu“

- Investicí se rozumí nakoupený produkt, jehož cena přesahuje určitou výši
- Ve sloupci „Převzal“ je podpis některého z řešitelů

| Poř. číslo | Faktura č. | Datum | Částka | Převzal |
|---|---|---|---|---|
| ..... | ..... | ..... | ..... | ..... |

# Entity ER diagramu

## „Projekt“

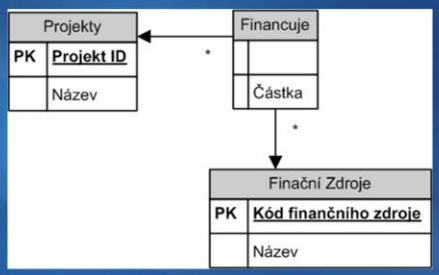
- Vychází z hlavičky formulářů

| Projekty | |
|---|---|
| PK | **Projekt ID** |
| | Název |

# Entity ER diagramu

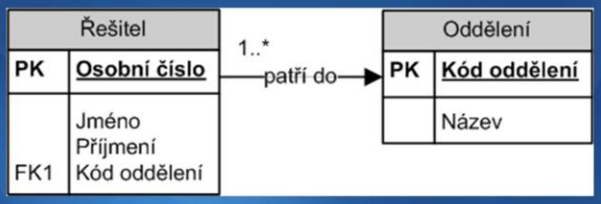
## „Finanční zdroje“

- Dle prvního formuláře
- „ Musí existovat číselník možných zdrojů“



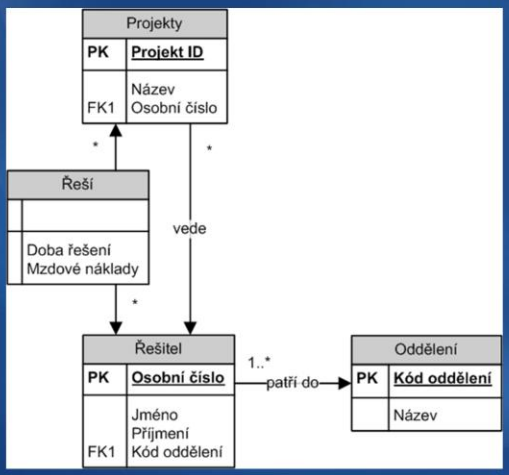
# Entity ER diagramu

## „Řešitel“



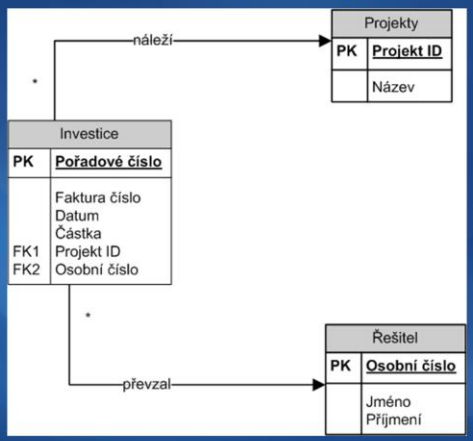
# Entity ER diagramu

„Řešitel“

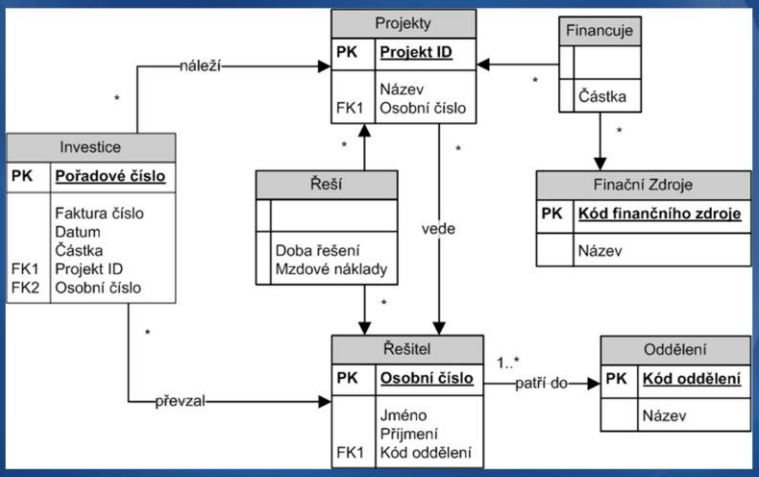


# Entity ER diagramu

## „Investice“



# Výsledný ER diagram



# Tvorba databáze


Databázi je možné kdykoliv přejmenovat, stejně jako je možné změnit vlastníka databáze. Občas dochází k problémům při přenosu databáze mezi servery a je nutné vlatníka databáze resetovat.

Znaková sada určuje kódování textových řetězců u datových typů char, varchar. Pro využití unicode je třeba využít datové typy nchar a nvarchar. Dále pak znaková sada určuje pořadí symbolů při řazení dat.

Recovery model určuje způsob práce s transakčním logem a možnosti jeho zálohování.

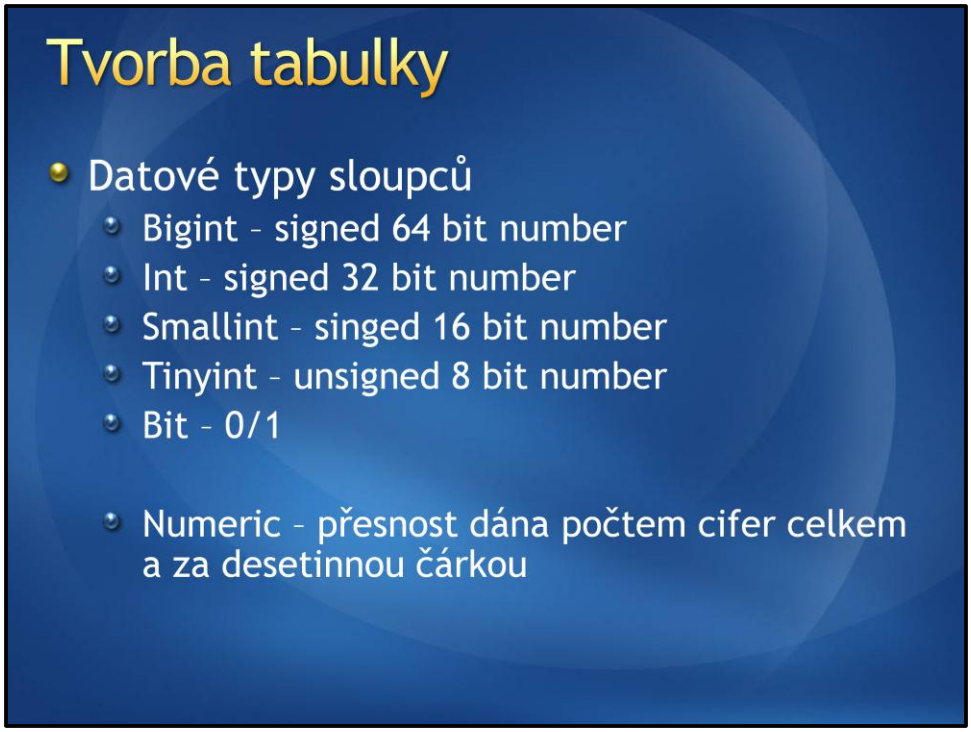
Compatibility level je důležitý pokud uvažujeme, že vytvářená databáze by měla fungovat i na předchozích verzích SQL serveru. Compatibility level je možné povýšit, ale ponížit jej lze jen v případě že není při tvorbě DB využito nové funkcionality, kterou předchozí verze SQL serveru nedisponují.

# Další operace s databází

- Detach
- Attach

- Back Up
- Recover

- Delete




Numeric – až 38 cifer (17 bajtů)

# Tvorba tabulky

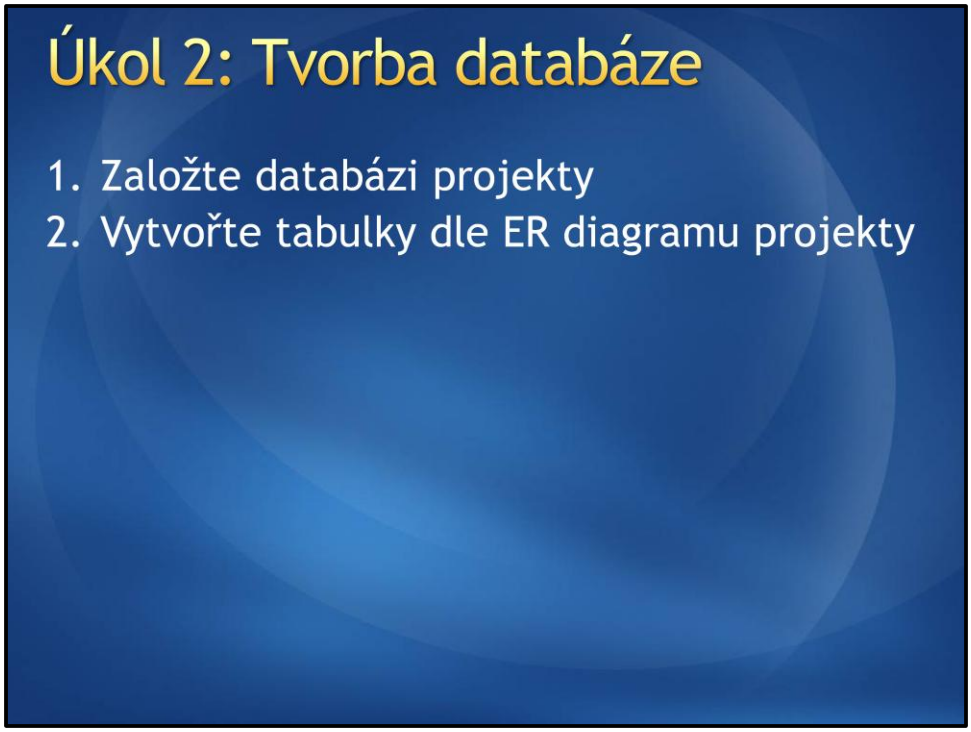
- Datový typy sloupců
  - Float, Real – aproximovaná desetinná čísla
  - Date, DateTime, DateTimeOffset
  - Char, Varchar – respektují nastavení znakové sady
  - Nchar, Nvarchar – unicode řetězce
  - Nvarchar(MAX) – unicode, délka 2^31-1 bajtů

# Tvorba tabulky

- Datové typy sloupců
  - Xml
  - Geography
  - Geometry
  - Uniqueidentifier
  - Timestamp

  - Binary
  - Varbinary


Úkol 2: Tvorba databáze
Zadání




Tabulky je možné tvořit 2 způsoby, pravým tlačítkem kliknete na Tables a vyberete New Table...
Nebo je možné využít Diagramy.

# Jazyk T-SQL: SELECT


K čemu je to dobré?
Zamyšlení

# Základní dotazy

**Získání všech řádků a sloupců z tabulky**

- `SELECT * FROM Person.Contact`

# Základní dotazy

**Získání všech řádků a vybraných sloupců z tabulky**

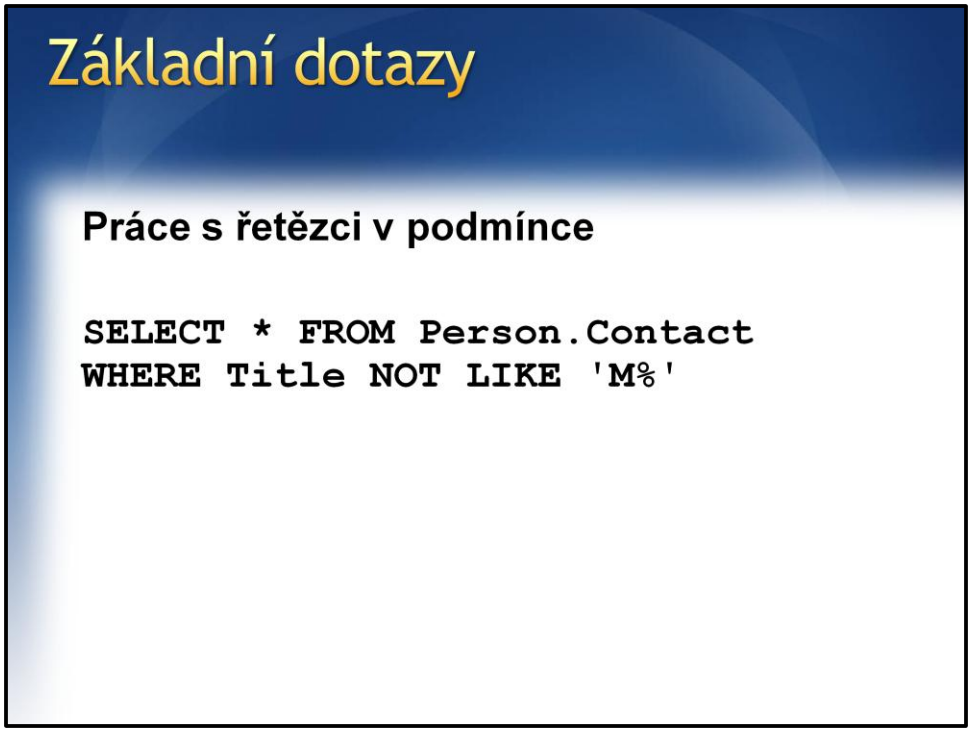
- ```
  SELECT FirstName, LastName
  FROM Person.Contact
  ```
  ```

# Základní dotazy

## Aplikace filtru na vracené řádky

- `SELECT * FROM Person.Contact WHERE Title = 'Mr.,`
- `SELECT * FROM Person.Contact WHERE Title = 'Mr.' AND Suffix = 'Jr.`
- `SELECT * FROM Person.Contact WHERE ContactID < 10`

V rámci WHERE je jeden logický výraz, který může vzniknout spojením dalších výrazů pomocé logických operátorů.




Výraz:
SELECT * FROM Person.Contact
WHERE Title LIKE 'M%,

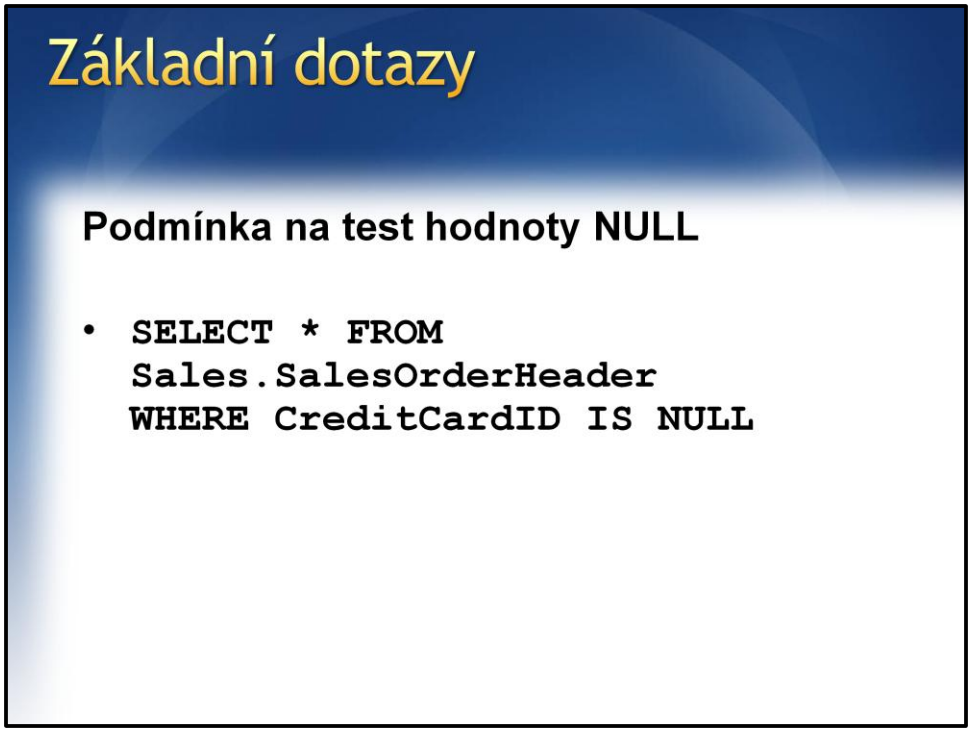
a

SELECT * FROM Person.Contact
WHERE Title = 'M%'

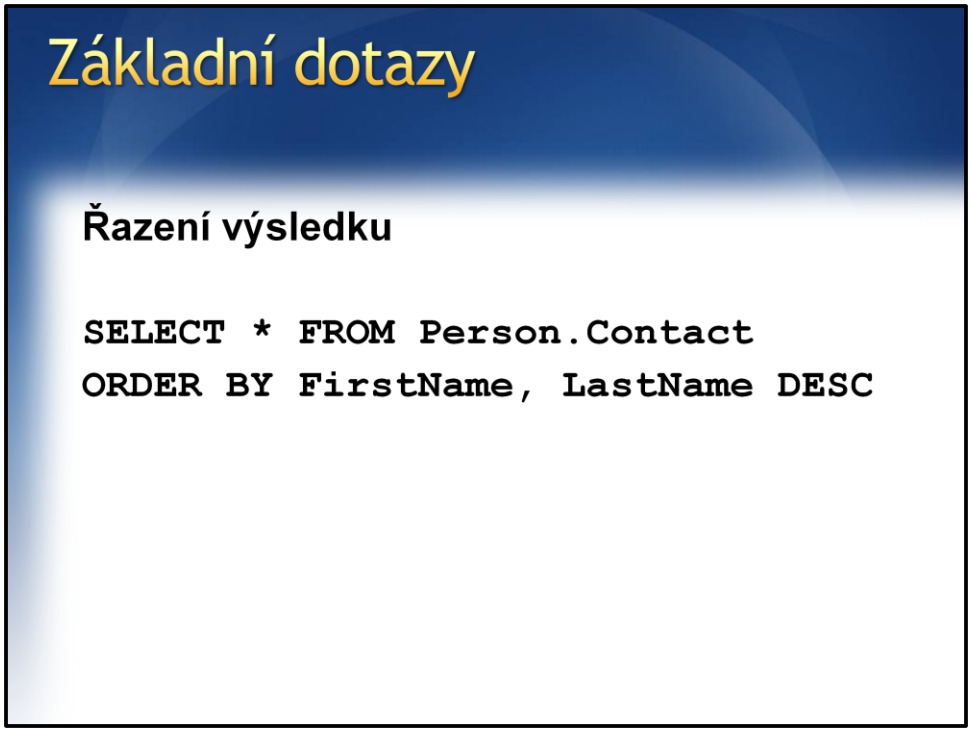
Nedělají to samé. 1. hledá lidi s titulem začínající na M, 2. hledá lidi s titulem v přesném znění: 'M%'

# Operátor LIKE

- *% - 0..n libovolných symbolů*
- *_ - 1 libovolný symbol*
- *[abc] - množina symbolů (aplikován 1x)*
- *[^abc] -všechny symboly, kromě určených (aplikován 1x)*




**Nefunguje:**
SELECT * FROM Sales.SalesOrderHeader
WHERE CreditCardID = NULL




Záznamy se vrací seřazeny vzestupně dle FirstName a druhotně seřazeny sestupně dle LastName

# TOP

```
SELECT TOP 5 *
FROM Person.Contact
WHERE Suffix IS NOT NULL
ORDER BY LastName DESC, FirstName

SELECT TOP 5 PERCENT *
FROM Person.Contact
WHERE Suffix IS NOT NULL
ORDER BY LastName DESC, FirstName
```

1: Vrací jen prvních 5 záznamů z výsledku
2: Vrací jen prvních 5% záznamů z výsledku

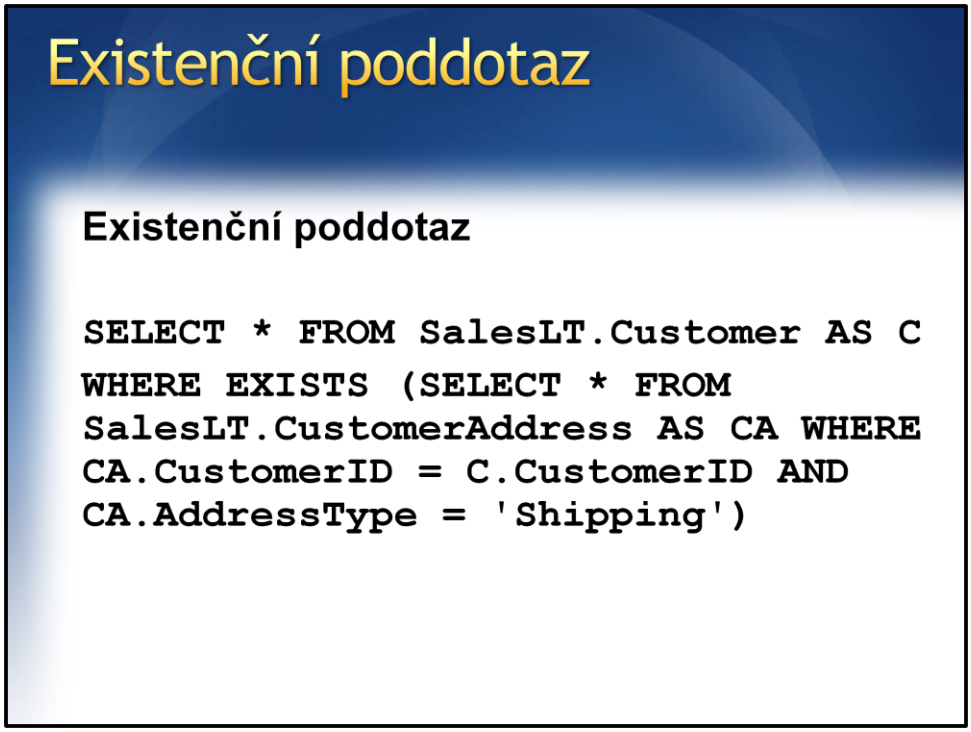
# Dotazy nad více tabulkami

- Spojování řádků na základě stejné hodnoty v určeném sloupci

  JOIN

- Dotazování se na hodnoty na základě výsledku vnějšího dotazu

  PODDOTAZ




Pro každý záznam vnějšího dotazu je zpracován poddotaz. Exists vrací true, pokud poddotaz vrací alespoň jeden záznam.

# Typy příkazu JOIN

INNER

- INNER JOIN

OUTER

- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

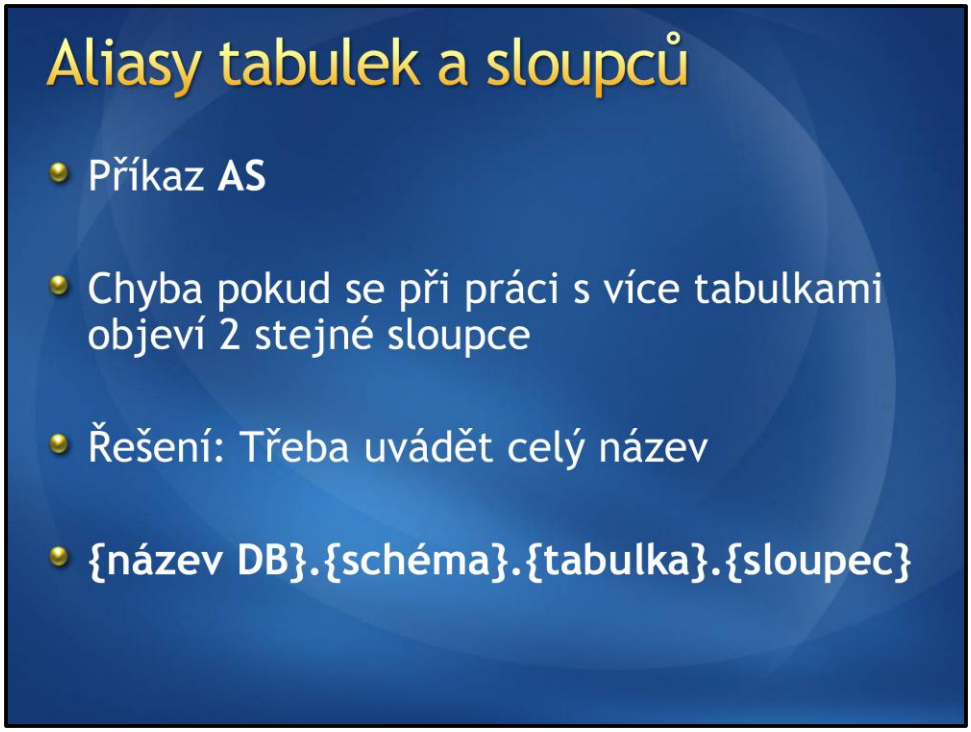
- CROSS JOIN

# INNER JOIN

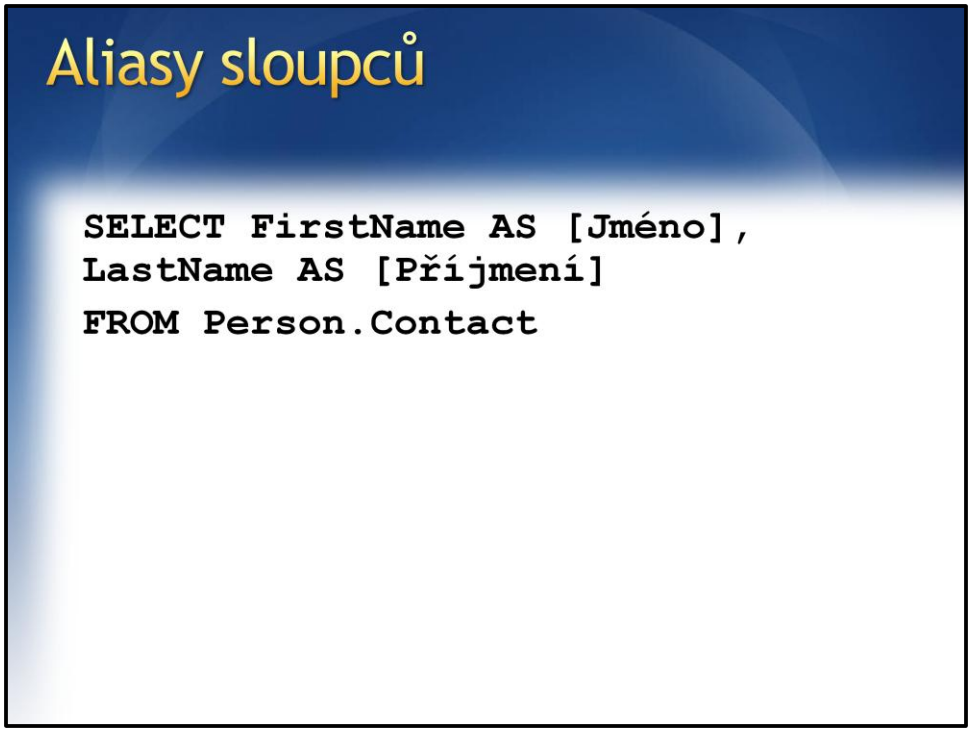
```
SELECT SOH.SalesOrderNumber,
CC.CardType
FROM Sales.SalesOrderHeader AS SOH
INNER JOIN Sales.CreditCard AS CC
ON SOH.CreditCardID =
CC.CreditCardID
WHERE CC.CardType = 'Vista'
```

# LEFT JOIN

```
SELECT SOH.SalesOrderNumber,
CC.CardType
FROM Sales.SalesOrderHeader AS SOH
LEFT JOIN Sales.CreditCard AS CC ON
SOH.CreditCardID = CC.CreditCardID
```



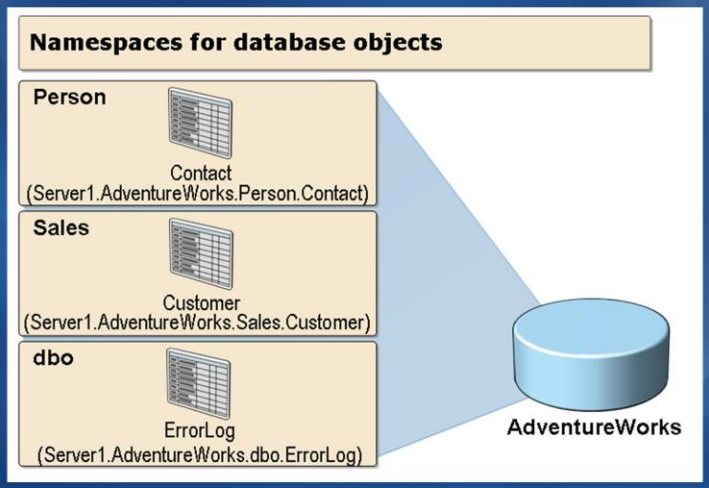

Pokud se v rámci dotazu, kdy spojujeme více tabulek vyskytne 2x sloupec se stejným názvem, což může nastat pokud se primární klíč jmenuje stejně jako cizí klíč, tak dotaz nepůjde přeložit, protože je zde nejednoznačně identifikovaný sloupec. Toto se dá využít doplněním celého názvu sloupce a nebo s využitím alias tabulky pro zkrácení.




Pokud chcete vytvořit alias s mezerou a nebo klíčovým slovem, použijte hranatých závorek.

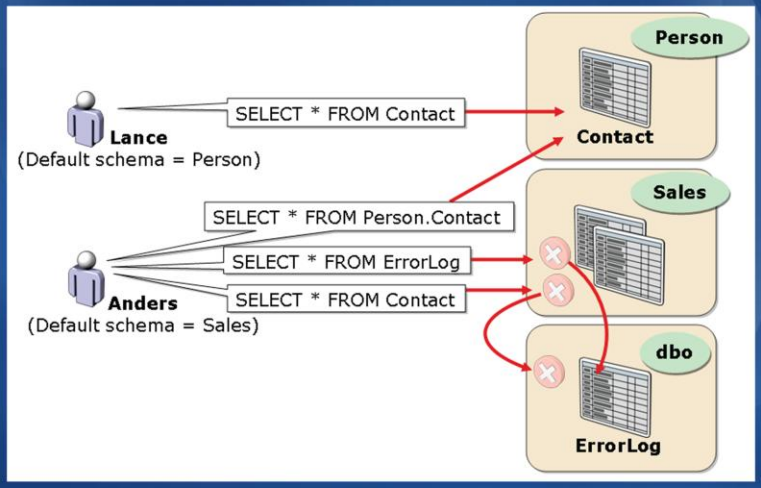
# Schéma

- Co je schéma?

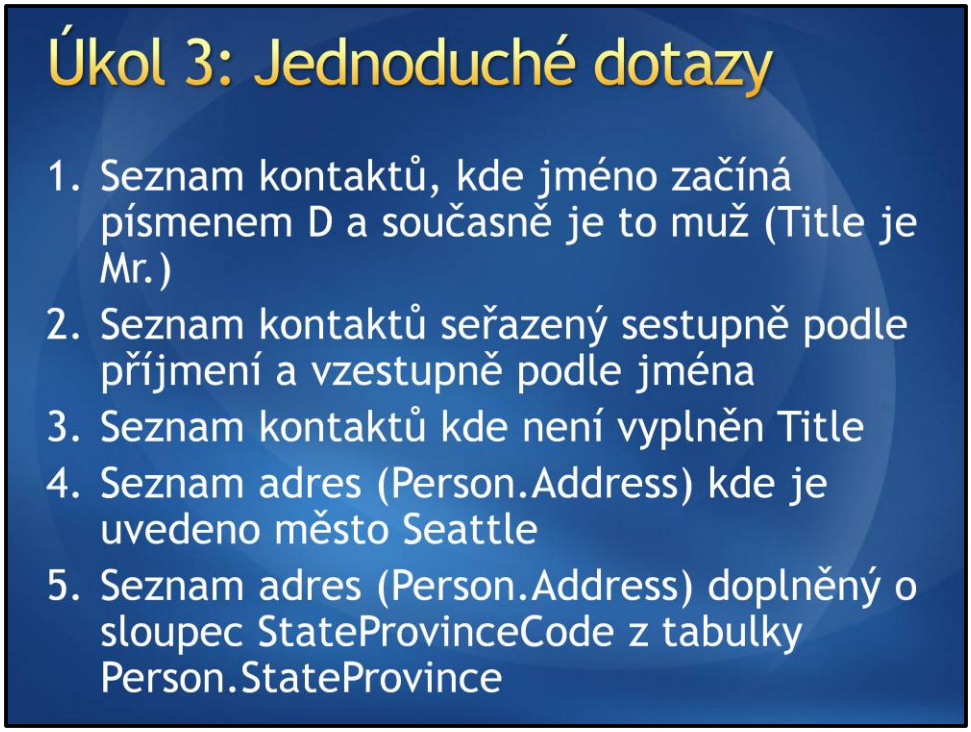


# Schéma

- Názvy objektů ve schématech:




Úkol 3:
Jednoduché dotazy
Zadání




**AdventureWorks**
*Řešení*

1:
```
SELECT * FROM Person.Contact
WHERE Title = 'Mr.' AND FirstName LIKE 'D%,
```

2:
```
SELECT * FROM Person.Contact
ORDER BY LastName DESC, FirstName
```

3:
```
SELECT * FROM Person.Contact
WHERE Title IS NULL
```

4:
```
SELECT * FROM Person.Address
WHERE City = 'Seattle’
```

5:
```
SELECT a.*, sp.StateProvinceCode FROM Person.Address AS a
INNER JOIN Person.StateProvince AS sp ON a.StateProvinceID = sp.StateProvinceID
```

# Seskupování

- Seskupení dat do skupin podle hodnoty daného sloupce
- Nad skupinami možnost provádět agregační funkce

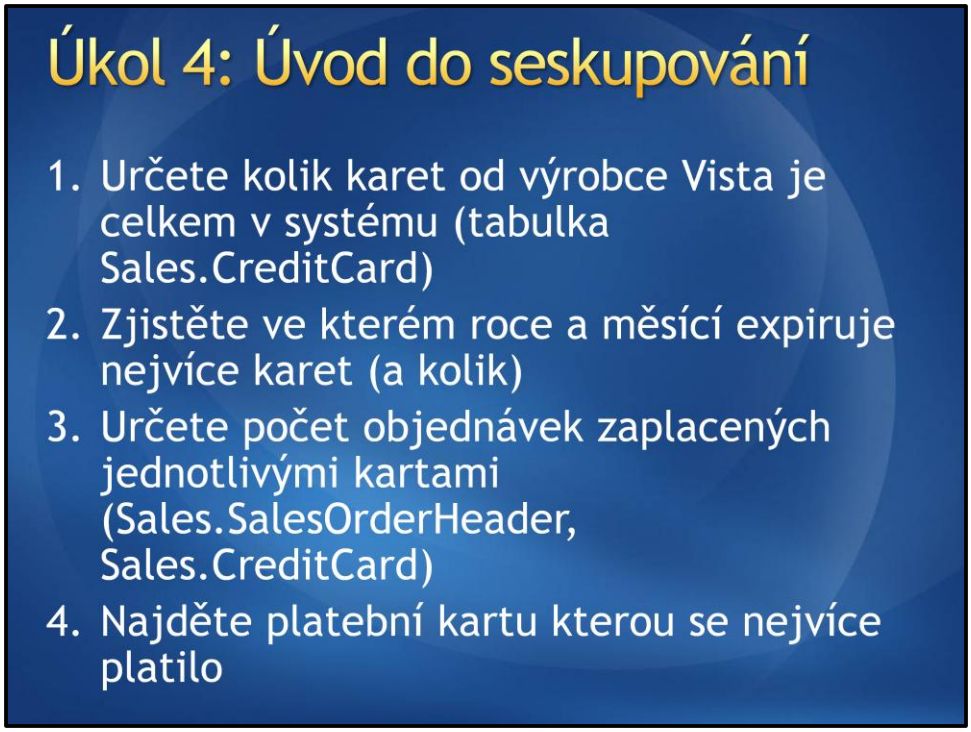
# GROUP BY

```
SELECT CardType, COUNT(*) AS CardCount
FROM Sales.CreditCard
GROUP BY CardType

SELECT CardNumber, CardType, COUNT(*)
FROM Sales.CreditCard AS cc
     INNER JOIN Sales.SalesOrderHeader
     AS soh
     ON cc.CreditCardID = soh.CreditCardID
GROUP BY CardNumber, CardType
ORDER BY COUNT(*) DESC
```

1: Vrací seznam typů platebních karet a počet jejich výskytů.
2: Vrací počet užití kreditní karty v rámci objednávek.


Úkol 4:
Úvod do seskupování
Zadání




**AdventureWorks**
*Řešení*

1:
```
SELECT COUNT(*)
FROM Sales.CreditCard
WHERE CardType = 'Vista'
```
*nebo*
```
SELECT CardType, COUNT(*)
FROM Sales.CreditCard
GROUP BY CardType
```
*Pozn.: Zobrazí i počty karet jiných výrobců*

2:
```
SELECT ExpYear, ExpMonth, COUNT(*)
FROM Sales.CreditCard
GROUP BY ExpYear, ExpMonth*
ORDER BY COUNT(*) DESC
```

3:
```
SELECT cc.CardNumber, COUNT(*)
FROM Sales.CreditCard AS cc
```

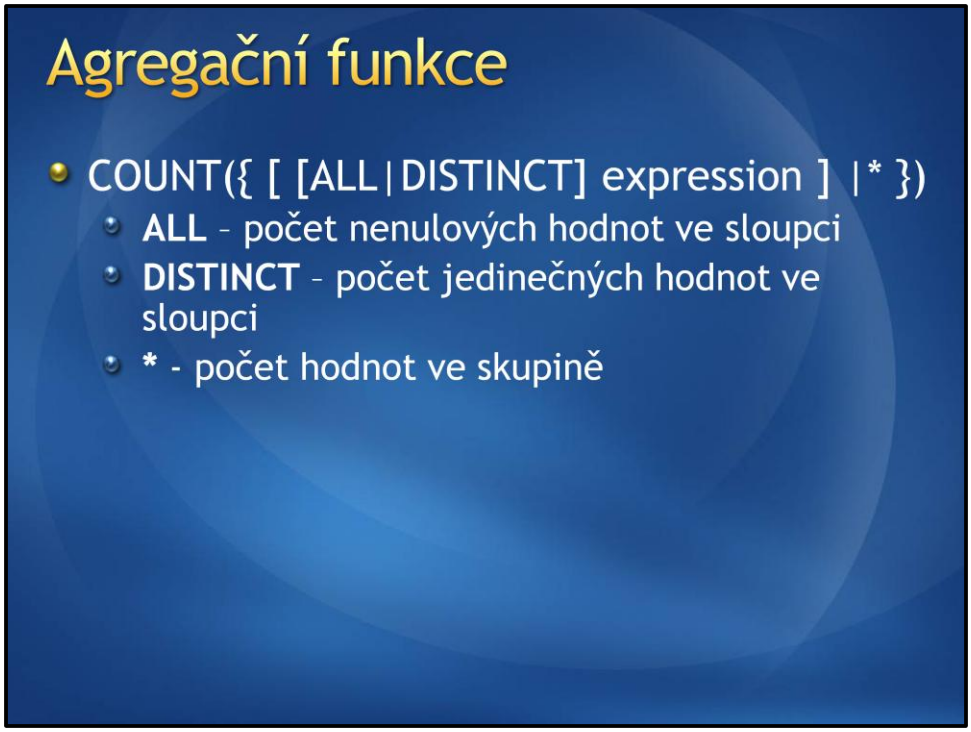
```
		INNER JOIN Sales.SalesOrderHeader AS soh ON cc.CreditCardID =
soh.CreditCardID
GROUP BY cc.CardNumber
```

4:

```
SELECT cc.CardNumber, COUNT(*)
FROM Sales.CreditCard AS cc
		INNER JOIN Sales.SalesOrderHeader AS soh ON cc.CreditCardID =
soh.CreditCardID
GROUP BY cc.CardNumber
ORDER BY COUNT(*) DESC
```


*Ukázka možnosti současného použití více agregačních funkcí nad skupinou záznamů:*

```
SELECT CardType, SUM(TotalDue) AS Total,
                 AVG(TotalDue) AS Average,
                 MIN(TotalDue) AS Minimum,
                 MAX(TotalDue) AS Maximum
FROM Sales.SalesOrderHeader AS soh
                 INNER JOIN Sales.CreditCard AS cc
                 ON soh.CreditCardID = cc.CreditCardID
GROUP BY cc.CardType
```




*1: Vrací seznam objednávek = počet řádků ve výsledku. Není použito GROUP BY! Pokud je ve výsedku jen agregační funkce. Výsledkem bude jeden řádek.*

```
SELECT COUNT(*)
FROM Sales.SalesOrderHeader
```

*2: Vrací počet objednávek placených kartou (řádky kde CreditCardID je NULL nejsou zahrnuty)*

```
SELECT COUNT(ALL CreditCardID)
FROM Sales.SalesOrderHeader
```

*3: Vrací počet použitých kreditních karet*

```
SELECT COUNT(DISTINCT CreditCardID)
FROM Sales.SalesOrderHeader
```

# Agregační funkce

```
SELECT cc.CardNumber, SUM(soh.TotalDue) AS
Total
FROM Sales.CreditCard AS cc
     INNER JOIN Sales.SalesOrderHeader AS
soh ON cc.CreditCardID = soh.CreditCardID
GROUP BY cc.CardNumber
```

Vrací seznam platebních karet a částku kolik bylo danou kartou utraceno peněz.

# Filtrování seskupených záznamů

- Pro filtrování seskupených záznamů slouží klíčové slovo **HAVING**
- Rozdíl mezi HAVING a WHERE:
  - **WHERE** se aplikuje na řádky před seskupováním
  - **HAVING** se aplikuje na seskupené záznamy

# Filtrování seskupených záznamů

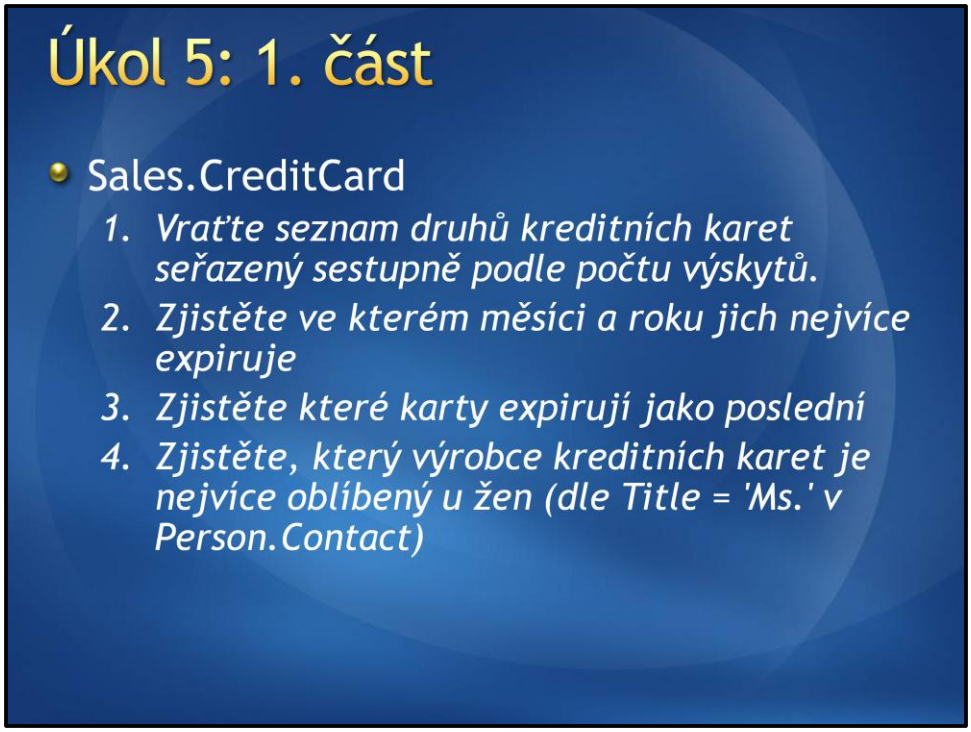
```
SELECT CardType, COUNT(*) AS CardCount
FROM Sales.CreditCard
GROUP BY CardType
HAVING COUNT(*) > 4800

SELECT CardType, COUNT(*) AS CardCount
FROM Sales.CreditCard
WHERE ExpYear = 2006
GROUP BY CardType
HAVING COUNT(*) > 1200
```

1: Vrací jen ty typy karet, kterých je víc jak 4800.
2: Vrací jen ty typy karet, kterých v roce 2006 expiruje více jak 1200.

# Úkol 5:
# Procvičení SELECTů

## *Zadání*




**AdventureWorks**
*Řešení*

1:
```
SELECT CardType, COUNT(*)
FROM Sales.CreditCard AS cc
GROUP BY CardType
ORDER BY COUNT(*) DESC
```

2:
```
SELECT ExpYear, ExpMonth, COUNT(*)
FROM Sales.CreditCard AS cc
GROUP BY ExpYear, ExpMonth
ORDER BY COUNT(*) DESC
```
*Pozn.: Expiruje jich nejvíce hned ve 2 měsících, použití TOP 1 tedy není vhodné.*

3:
```
DECLARE @ExpYear int
DECLARE @ExpMonth int

SELECT TOP 1 @ExpYear = ExpYear, @ExpMonth = ExpMonth
FROM Sales.CreditCard
```

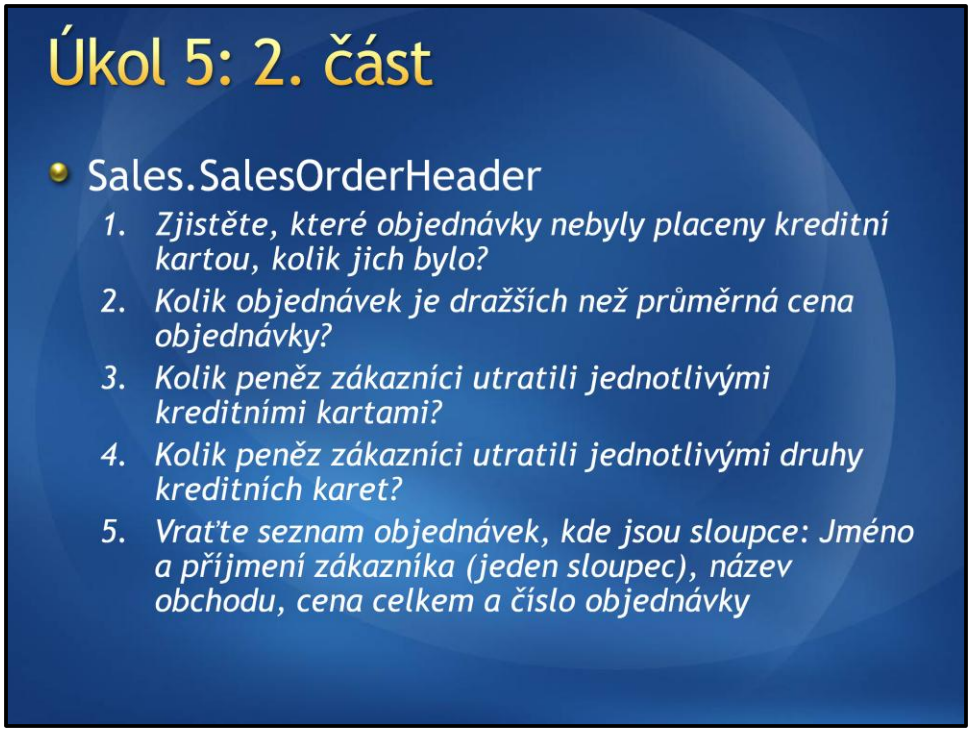
```
ORDER BY ExpYear DESC, ExpMonth DESC

SELECT * FROM Sales.CreditCard
WHERE ExpYear = @ExpYear AND ExpMonth = @ExpMonth
```

*Pozn.: Při řešení je využito proměnných, kdy proměnná se před použitím deklaruje pomocí DECLARE, v prvním selectu si do nich uložíme údaje o posledním roku a měsíci, kdy expiruje nějaká karta (pozor na TOP 1) a následně tyto proměnné použijeme v rámci posledního selectu, který vrací seznam karet, které v tom období expirují.*

4:

```
SELECT CardType, COUNT(*)
FROM Sales.CreditCard AS cc
                INNER JOIN Sales.ContactCreditCard AS ccc
                                ON cc.CreditCardID = ccc.CreditCardID
                INNER JOIN Person.Contact AS c ON ccc.ContactID = c.ContactID
WHERE c.Title = 'Ms.'
GROUP BY CardType
ORDER BY COUNT(*) DESC
```




**AdventureWorks**
*Řešení*

1:
```
SELECT * FROM Sales.SalesOrderHeader
WHERE CreditCardID IS NULL

SELECT COUNT(*) FROM Sales.SalesOrderHeader
WHERE CreditCardID IS NULL
```

2:
```
SELECT COUNT(*) FROM Sales.SalesOrderHeader
WHERE TotalDue > (SELECT AVG(TotalDue) FROM Sales.SalesOrderHeader)
```

*Pozn.: Jak vidíte, je možné z dotazu v závorce vrátit jednu hodnotu a použít ji jako skalární hodnotu v podmínce dotazu jiného.*

3:
```
SELECT cc.CardNumber, SUM(TotalDue)
FROM Sales.CreditCard AS cc
                INNER JOIN Sales.SalesOrderHeader AS soh
                ON cc.CreditCardID = soh.CreditCardID
```

```
GROUP BY cc.CardNumber
```

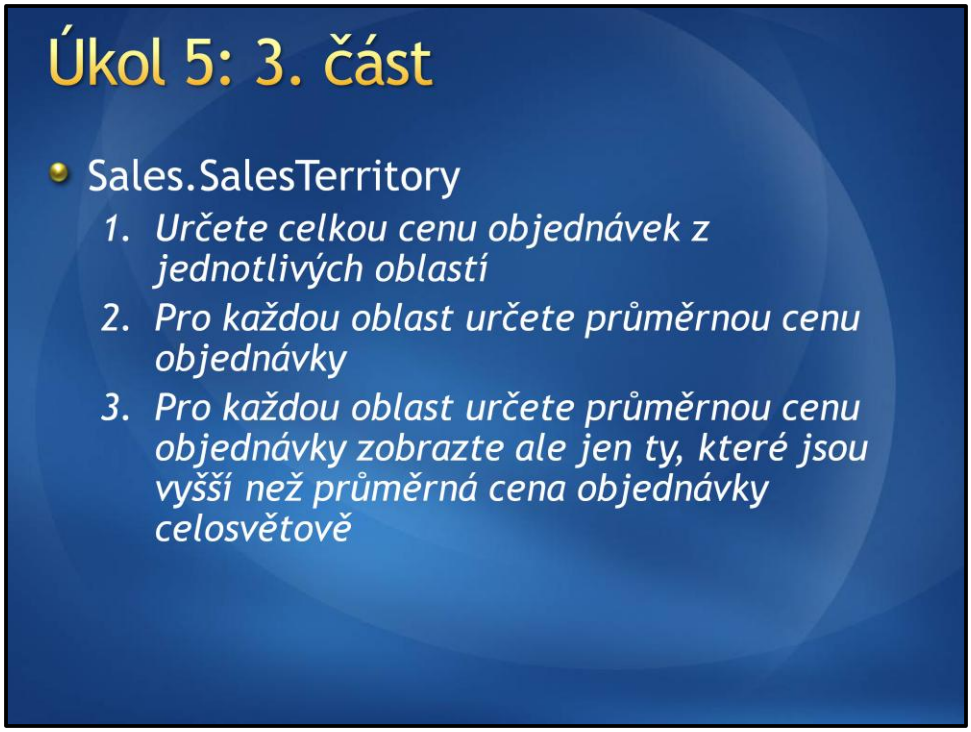
4:

```
SELECT CardType, SUM(TotalDue)
FROM Sales.CreditCard AS cc
        INNER JOIN Sales.SalesOrderHeader AS soh
        ON cc.CreditCardID = soh.CreditCardID
GROUP BY CardType
```

5:

```
SELECT (c.FirstName + ' ' + c.LastName) AS CustomerName,
        s.Name, soh.TotalDue, soh.SalesOrderNumber
FROM Sales.SalesOrderHeader AS soh
        INNER JOIN Sales.Store AS s ON soh.CustomerID = s.CustomerID
        INNER JOIN Person.Contact AS c ON soh.ContactID = c.ContactID
```

*Pozn.: Sloupeček CustomerName vzniká výpočtem, kdy dojde ke spojení řetězce ze sloupce FirstName s mezerou a hodnotou ze sloupce LastName. Toto je jen ukázka, je samozřejmě možné nové sloupce zakládat i na základě matematických operací. Např. součet 2 sloupců....*



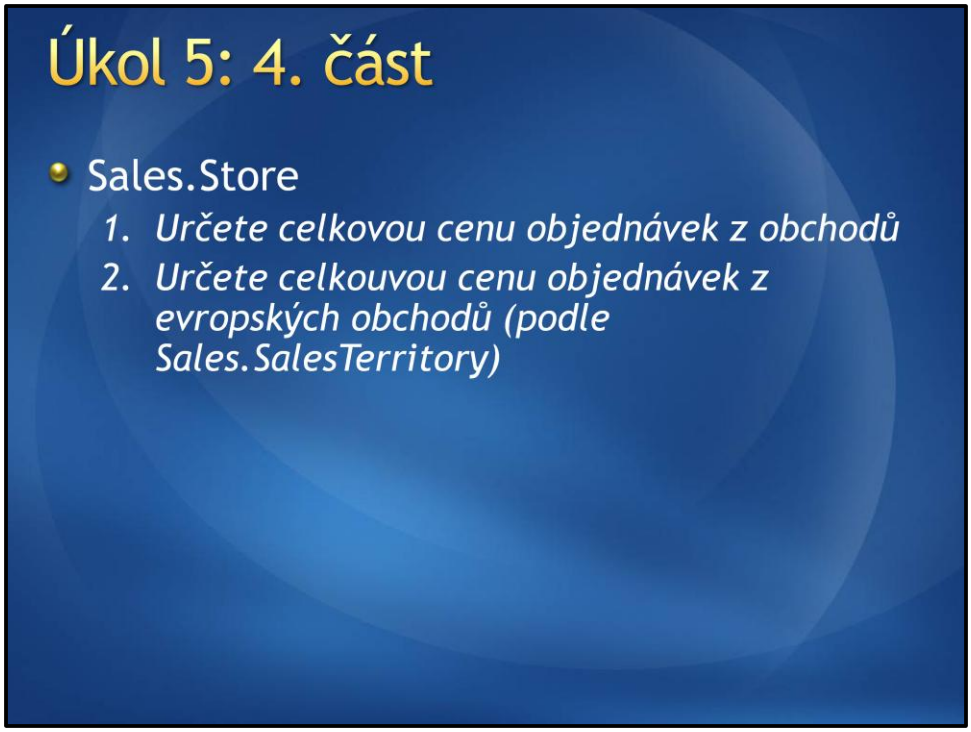

**AdventureWorks**
*Řešení*

1:
```
SELECT st.TerritoryID,st.CountryRegionCode, st.Name,
                SUM(soh.TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader AS soh
                INNER JOIN Sales.SalesTerritory AS st
                ON soh.TerritoryID = st.TerritoryID
GROUP BY st.TerritoryID, Name, CountryRegionCode
```

2:
```
SELECT st.TerritoryID,st.CountryRegionCode, st.Name,
                AVG(soh.TotalDue) AS AverageTotalDue
FROM Sales.SalesOrderHeader AS soh
                INNER JOIN Sales.SalesTerritory AS st
                ON soh.TerritoryID = st.TerritoryID
GROUP BY st.TerritoryID, Name, CountryRegionCode
```

3:
```
SELECT st.TerritoryID,st.CountryRegionCode, st.Name,
                AVG(soh.TotalDue) AS AverageTotalDue
```

```
FROM Sales.SalesOrderHeader AS soh
                INNER JOIN Sales.SalesTerritory AS st
                ON soh.TerritoryID = st.TerritoryID
GROUP BY st.TerritoryID, Name, CountryRegionCode
HAVING AVG(soh.TotalDue) > (SELECT AVG(TotalDue) FROM Sales.SalesOrderHeader)
```




**AdventureWorks**
*Řešení*

1:
```
SELECT SUM(TotalDue) FROM Sales.SalesOrderHeader AS soh
                INNER JOIN Sales.Store AS s ON soh.CustomerID = s.CustomerID
```

*Pozn.: Inner join zajistí, že objednávka je z obchodu (existuje ovchod s CustomerID které je v objednávce).*
*Celkem je přes 31000 objednávek, ale jen přes 3800 je z obchodů.*

*2:*
```
SELECT SUM(TotalDue) FROM Sales.SalesOrderHeader AS soh
                INNER JOIN Sales.Store AS s ON soh.CustomerID = s.CustomerID
                INNER JOIN Sales.SalesTerritory AS st ON soh.TerritoryID =
st.TerritoryID
WHERE st.[Group] = 'Europe,
```

*Pozn.: Group je klíčové slovo, proto je v hranatých závorkách.*

# Jazyk T-SQL: INSERT

# INSERT

- Příkaz sloužící k vložení dat do tabulky
- Rozdělujeme 2 hlavní varianty:
  - Vložení nových dat zadaných uživatelem
  - Vložení dat z jiné tabulky

# Vložení dat zadaných uživatelem

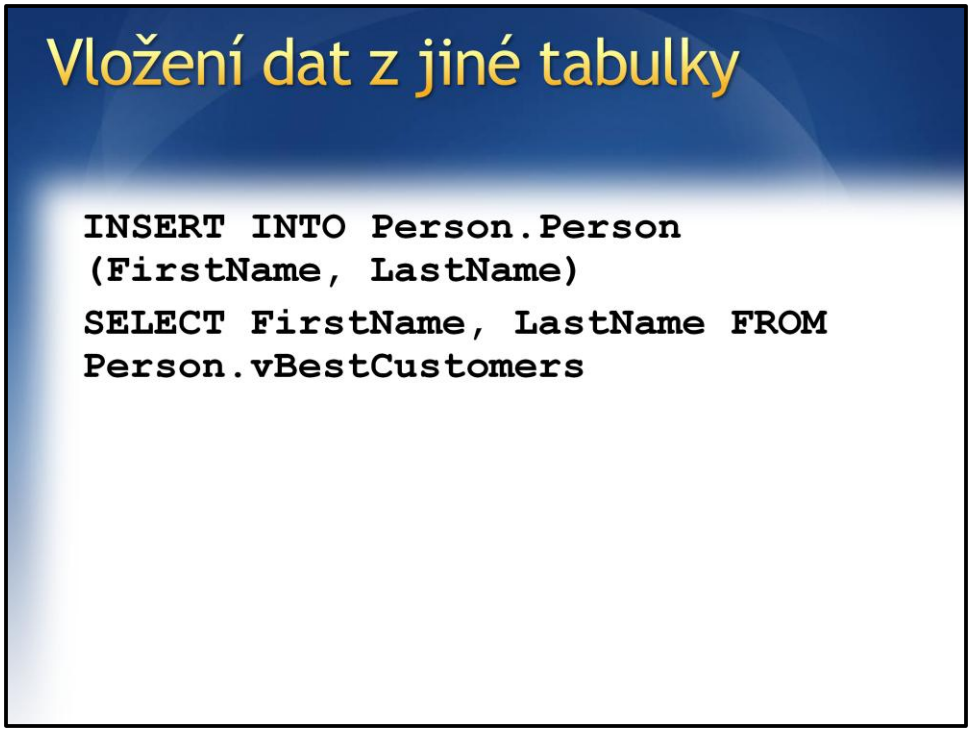
```
INSERT INTO Person.Person
VALUES (vkládané hodnoty)

INSERT INTO Person.Person
(FirstName, LastName)
VALUES (vkládané hodnoty)
```

Vloží do tabulky Person.Person nový řádek, který bude tvořen hodnotami uvedenými v závorce za klíčovým slovem VALUES.
**Důležité je, že hodnoty se zadávají ve stejném pořadí jako jsou definovány sloupce v tabulce!**

**Vždy platí:**
1. Sloupce které mají vyžadovanou hodnotu musí být vyplněny, jinak nemůže být řádek vložen
2. Automaticky v seznamu sloupců kam je třeba vložit hodnota nejsou sloupce které mají vypočítanou hodnotu (IS IDENTITY nebo COMPUTED COLUMN)

Potřebujete-li vložit jen některé sloupce, je možné za tabulkou specifikovat seznam sloupců, které se budou plnit. **I přesto platí 1. zde uvedené pravidlo.**

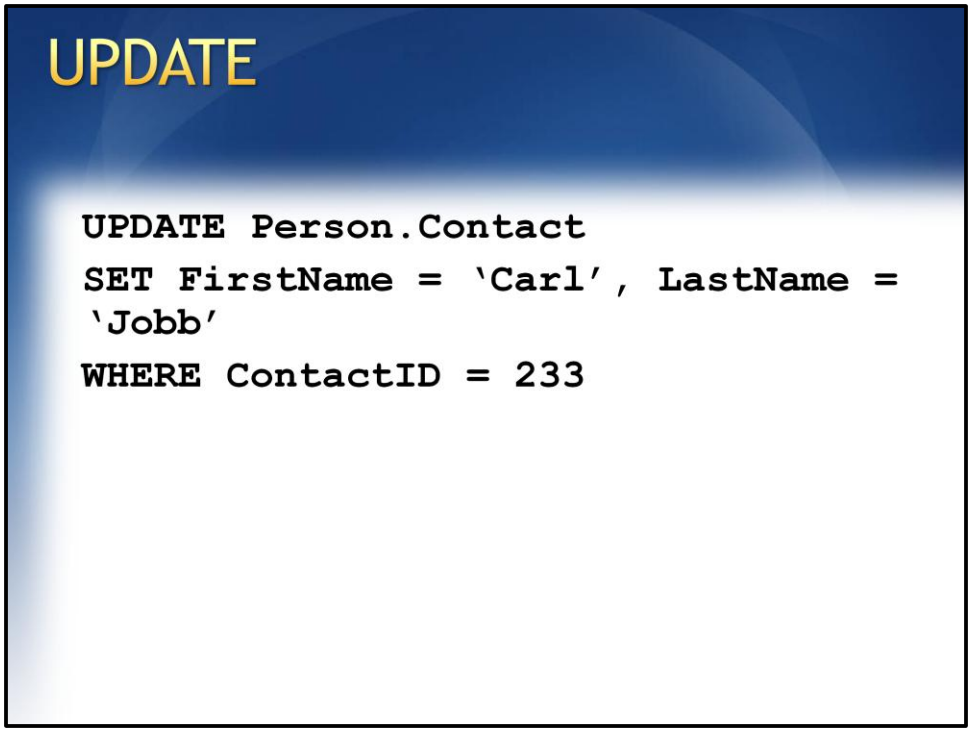



Tato varianta dotazu INSERT neobsahuje klíčové slovo VALUES, které je nahrazeno příkazem SELECT, který získá potřebná data, která budou vložena do 1. tabulky. Je třeba dát pozor na splnění podmíny ALLOW NULLS u jednotlivých řádků a je třeba, aby sedělo pořadí sloupců mezi vkládanými a načítanými daty stejně jako jejich datové typy.

# Jazyk T-SQL: UPDATE

# UPDATE

- Příkaz UPDATE slouží k aktualizaci již existujících dat v tabulce
- Je to příkaz nad tabulkou nikoliv nad záznamem!
- Absence WHERE má za následek aplikaci na všechny řádky!



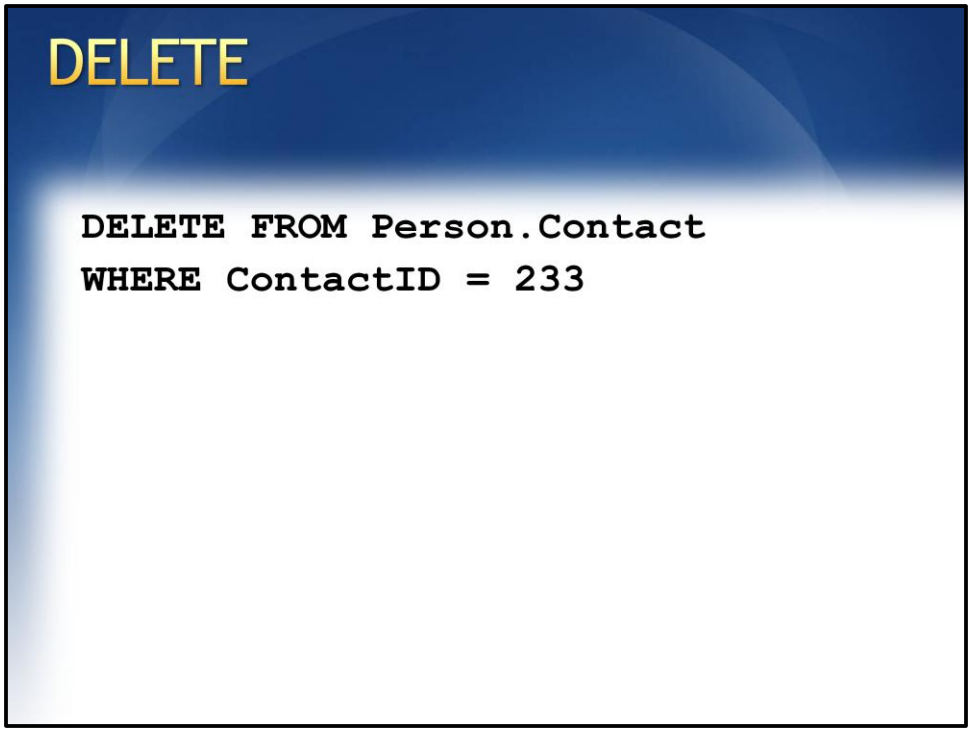

Aktualizuje řádek kde ContactID = 233 a nastaví hodnotu sloupce FirstName na „Carl“ a sloupce LastName na „Jobb“.

Je samozřejmě možné aktualizovat víc řádků, pokud jsou zahrnuty v podmínce. Např.: Změnit Title u všech lidí co mají FirstName John...

# Jazyk T-SQL: DELETE

# DELETE

- Příkaz DELETE slouží k odstranění řádků z tabulky, které splňují podmínku.
- Je to příkaz nad tabulkou nikoliv nad záznamem!
- Absence WHERE má za následek aplikaci na všechny řádky!




Odstraní řádek, kde ContactID = 233.

Je samozřejmě možné odstranit víc řádků, pokud jsou zahrnuty v podmínce.

# Pohledy

# Pohledy

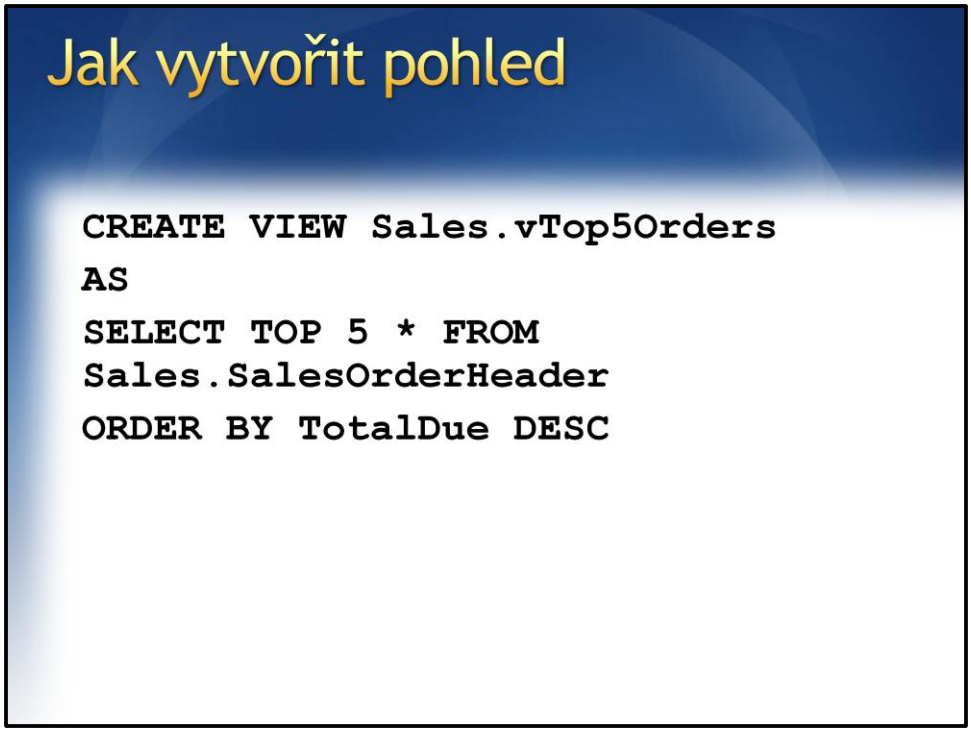
- Pohled je objekt v databázi který má v sobě uložený konkrétní dotaz SELECT
- Pohled se virtuálně tváří jako tabulka
- Je možné je použít v části FROM a JOIN dotazu SELECT
- Pohled může umožňovat za jistých podmínek i změny dat

# Pohledy

- Nedoporučuje se zakládat pohledy nad pohledy
- Toto zřetězení může v konečném výsledku vést k neefektivním a pomalým dotazům
- Vývojář zapomene co za výpočty stojí za pohledem – nízký výkon

# Pohledy – Kdy použít

- Poskytnout uživateli data v podobě v jaké je to vhodné a povolit jen operaci SELECT nad daným pohledem
- Je třeba, aby se zdroj dat (tabulka) jmenoval stále stejně, ale přitom se data z něj generovala dynamicky
  - Když nejde přepsat aplikaci
- Zapouzdření častých dotazů, které by se musely psát pořád znovu




Pohled se vytvoří pomocí CREATE VIEW, kde za klíčovým slovem AS následuje kód 1 SELECT dotazu.

Někdy je třeba, když se data načítají z více tabulek a pak se spojí za sebe nějakým operátorem. Podmínkou pohledu však je, že výsledkem musí být jedna tabulka.

Operátory pro spojení dat (množinové operace):
**UNION (sjednocení)**

*SELECT ProductModelID, Name*
*FROM Production.ProductModel WHERE ProductModelID NOT IN (3, 4)*
*UNION*
*SELECT ProductModelID, Name FROM dbo.Gloves ORDER BY Name;*

**UNION ALL (sjednocení které vrací jedinečné záznamy)**

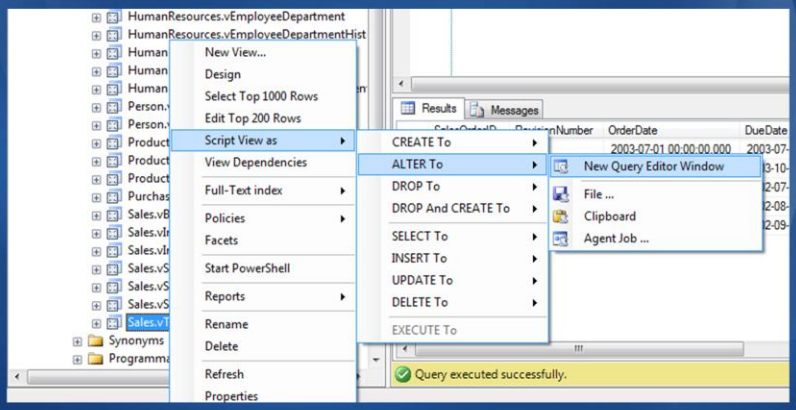
**EXCEPT**
Odebere z výsledku prvního dotazu záznamy které jsou vráceny 2. dotazem

**INTERSECT**
Vrací jen řádky v obou dotazech shodné

*Pozn.: Syntaxe použití je u o všech operátorů stejná*

# Další operace s pohledem

- Možné pomocí Management studia vygenerováním T-SQL příkazů

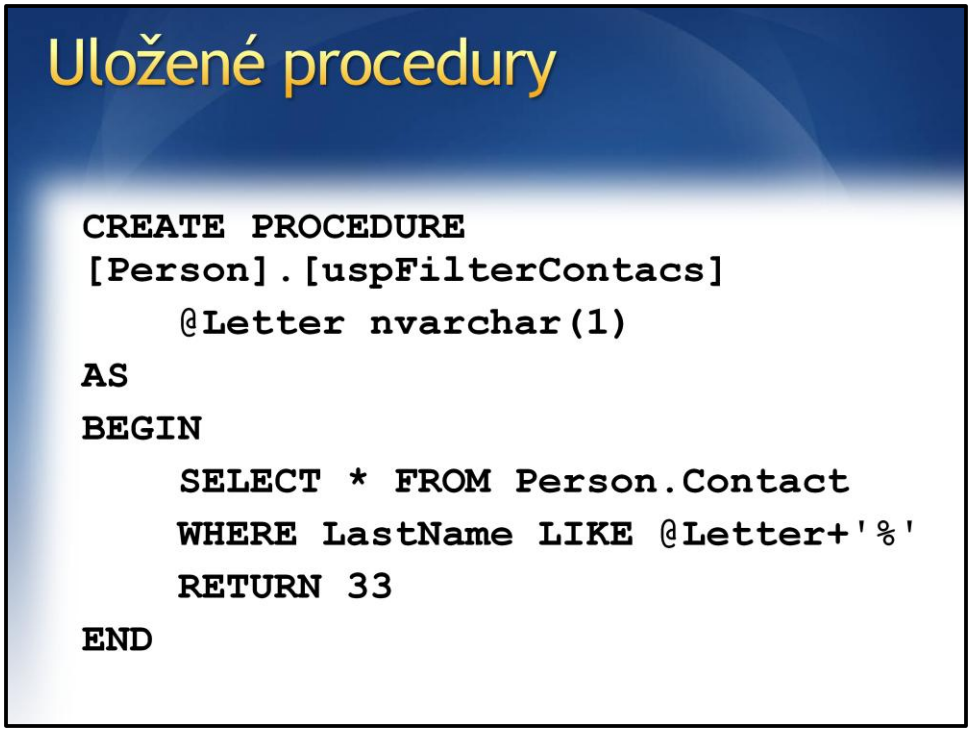



Každá operace kterou provádí SQL Server Management studio je na pozadí dotaz v jazyce T-SQL.

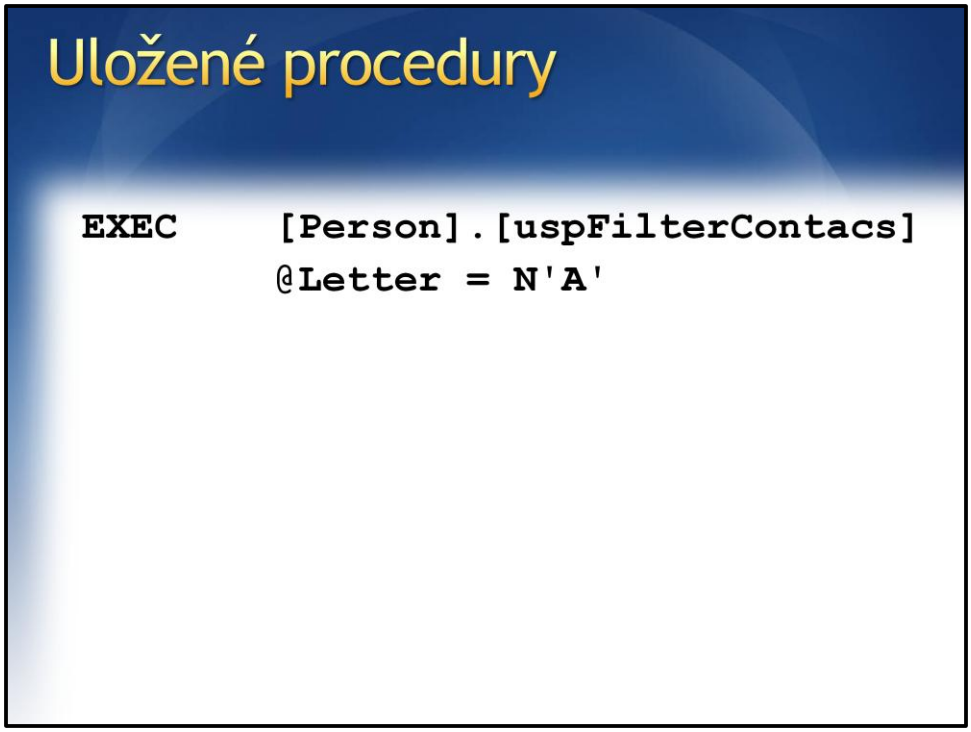
# Uložené procedury

# Uložené procedury

- Uložené procedury zapouzdřují blok T-SQL kódu do jednoho objektu uloženého na SQL Serveru
- Mají mnohem širší možnosti použití než pohledy
  - nejsou omezeny na 1 příkaz SELECT
  - mají možnost mít vstupní parametry




Vytvoří uloženou proceduru „uspFilterContacts“ ve schématu „Person“. Tato uložená procedura bude přijímat jeden parametr typu nvarchar o délce 1 znak.
Parametr je pak v těle použit v rámci dotazu, který vrací zákazníky jejichž příjmení začíná na dané písmeno.




Kód který dříve vytvořenou proceduru spouští. Jako vstupní parametr předává písmeno „A“

# Funkce v SQL serveru

# Rozdělení funkcí

- Built-in
- User defined

- Scalar-valued function
- Table-valued function

# Built-in funkce

- Agregační funkce
  - AVG
  - COUNT
  - MIN
  - MAX
  - SUM
  - STDEV

# Built-in funkce

- Matematické funkce
  - ABS (absolutní hodnota)
  - POWER (mocnina)
  - ROUND (zaokrouhlení)
  - PI
  - RAND
  - …


LEFT a RIGHT – vrací řetězec o délce zadaného počtu znaků (zleva nebo zprava)

# Built-in funkce

- Pro práci s NULL
  - ISNULL
    - Ověří jestli je NULL, jinak nahradí hodnotou v parametru
  - COALESCE
    - Vrací 1. nenullový parametr
  - NULLIF
    - Vrátí NULL, když mají oba zadané parametry stejnou hodnotu. Jinak vrací 1. výraz

# Built-in funkce

- Funkce pro práci s datem a časem
  - GETDATE
    - Vrací aktuální datum a čas
  - DATEADD
  - DATEDIFF

# Vlastní skalární funkce

- Vrací skalární hodnotu pro zadané parametry
- Nevolat příliš často pokud funkce pracuje s moc daty – snížení výkonu

# Vlastní skalární funkce

```
CREATE FUNCTION Soucet
(
    @a int, @b int
)
RETURNS int
AS
BEGIN
    RETURN @a+@b
END
```

# Vlastní tabulkové funkce

- Vracejí tabulku dat
- Možnost využít jako zdroj dat pro dotaz
- Rozdíl pokud obsahují jeden SELECT nebo víc SELECTů
- Pozor na výkon
- Samostudium: Příkaz APPLY na MSDN

# Triggery

# Triggery

- Rozdělení:
  - DDL triggery
    - Reagují na změny schématu DB
  - DML triggery
    - Reagují na operace nad daty

# Triggery

- Každá modifikační operace otevírá transakci a trigger **je součástí** této transakce.
- Když trigger zavolá ROLLBACK, tak se tedy nic neprovede v rámci dané transakce
- Opatrně na výkon

# DDL Triggery

- Reagují na změny objektů v databázi
- Dají se použít k auditování nebo prevenci nechtěných změn

# DDL Trigery

```
CREATE TRIGGER safety ON DATABASE FOR
DROP_VIEW
AS
RAISERROR ('You must disable Trigger
"safety" to drop views!',10, 1)
ROLLBACK
GO
```

http://msdn.microsoft.com/en-us/library/ms189799.aspx

# DML Triggery

- Slouží jako reakce na operace
  - INSERT
  - UPDATE
  - DELETE
- Existují 2 typy
  - AFTER – po provedení operace
  - INSTEAD OF – provede trigger místo dané operace

# DML Triggery

```
CREATE TRIGGER reminder2
ON Sales.Customer
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    -- např. vložíme data do logu
END
```

# Transakce

# Transakce

- Základní vlastnosti:
  - **A**tomicity - buď jsou provedeny všechny operace v rámci transakce nebo žádná
  - **C**onsistency – běh jediné transakce zachovává konzistenci databáze
  - **I**solation – souběžně běžící transakce o sobě neví a neovlivňují se
  - **D**urablity – po dokončení transakce jsou změny uchovány tak, že přežijí i případný výpadek systému

# Concurrency Control

- Pessimistic Concurrency
  - Systém uzamyká modifikovaná data a dalším operací je umožní modifikovat až po odemčení.
- Optimistic Concurrency
  - Transakce neuzamče data, když je načte, ale před aktualizací zkontroluje, jestli nebyla změněna. Při změně podkladových dat dojde k chybě.

# Isolation Level

- Definuje úroveň zámků nad daty při pokusu o jejich čtení, když jsou data jinou transakcí uzamčena pro zápis.
- http://msdn.microsoft.com/en-us/library/ms189122.aspx

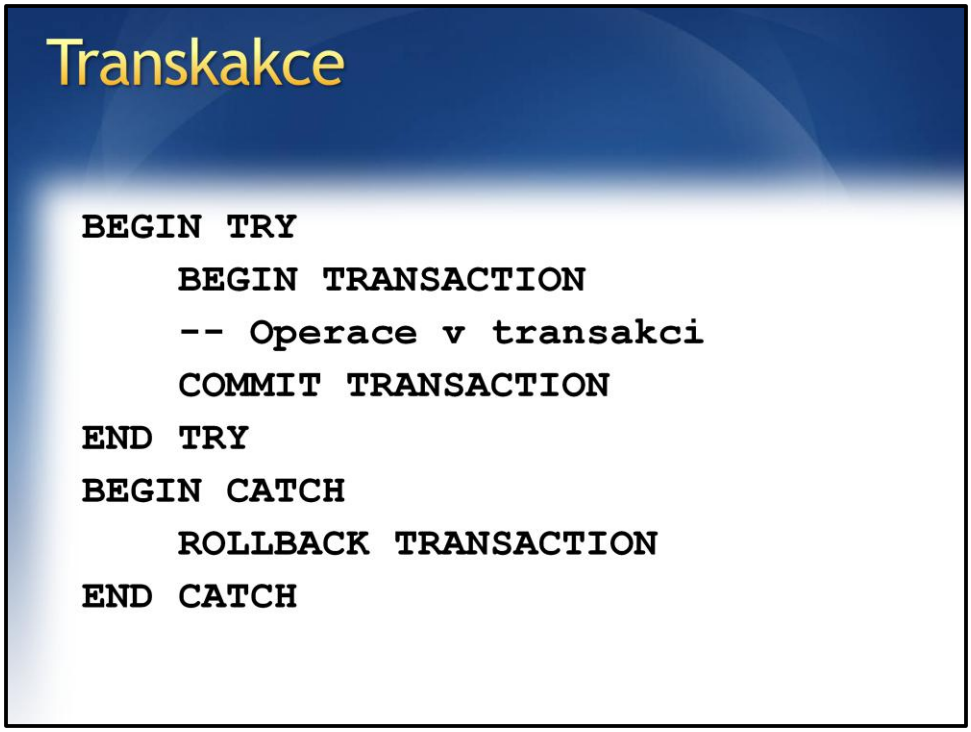
# Následky souběžného zpracování

- Včetně ukázek popsáno na: http://msdn.microsoft.com/en-us/library/ms190805.aspx

# Práce s transakcí

- Transakci je třeba zahájit pomocí **BEGIN TRANSACTION**
- Při chybě je třeba zavolat **ROLLBACK TRANSACTION**
- Pro potvrzení transakce **COMMIT TRANSACTION**

- Vhodné použití TRY/CATCH




Pokud se v bloku try vyskytne chyba, je zavoláno ROLLBACK v bloku catch a tím je transakce zrušena. Pokud se při zpracování bez chyby dostaneme až k příkazu COMMIT, je transake dokončena.