UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Databázová vrstva Oracle pro komunikaci s MSSQL

Ondřej Švec

Diplomová práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Ondřej Švec**
Osobní číslo:	**I10424**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Databázová vrstva Oracle pro komunikaci s MSSQL**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Teoretická část bude obsahovat:
Rešerši možností migrace mezi databázovými systémy Oracle a MSSQL.
Popis nejzásadnějších rozdílů v Oracle a MSSQL (zejména rozdíl v pojetí uživatelů a schémat).
Analýzu navrhovaného řešení.
Dokumentaci k navrženému řešení.
Cílem práce bude vytvořit databázovou vrstvu formou balíčků PL/SQL pro databázový systém Oracle 11g, která umožní navázání spojení do systému MSSQL. Navržená vrstva bude umožňovat:
Migraci uživatelských účtů z Oracle na MSSQL.
Migraci základních databázových objektů (tabulky, pohledy, indexy, omezení).
Spouštění základních dotazů na serveru MSSQL a poskytnutí výsledků formou tabulkové funkce (umožní získání dat z obou databázových serverů současně).

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

LACKO, Luboslav. Oracle: správa, programování a použití databázového systému. 1. vyd. Praha: Computer Press, 2002, 464 s. ISBN 80-722-6699-3.
LONEY, Kevin. Mistrovství v Oracle Database 10g. 1. vyd. Brno: Computer Press, 2006, 700 s. ISBN 80-251-1277-2.
KYTE, Thomas. Expert Oracle database architecture: Oracle database 9i, 10g, and 11g programming techniques and solutions. 2nd ed. Berkeley, Calif.: Apress, 2010. ISBN 978-143-0229-469.
LACKO, L'uboslav. Jak vyzrát na Microsoft SQL Server 2008: správa, konfigurace, programování. Vyd. 1. Brno: Computer Press, 2009, 469 s. ISBN 978-80-251-2101-6.
www.oracle.com
msdn.microsoft.com

Vedoucí diplomové práce: **Ing. Jiří Zechmeister**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2012

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 15. 05. 2013

Ondřej Švec

Poděkování

Tímto bych rád poděkoval panu Ing. Jiřímu Zechmeisterovi za to, že mi umožnil pracovat na zadaném tématu, za vedení a rady při tvorbě této diplomové práce.

Své poděkování bych chtěl vyjádřit i svým rodičům za podporu, trpělivost a vstřícnost nejen během tvorby této práce, ale i za dobu celého studia.

Anotace

Práce se zabývá migrací mezi databázovými systémy Oracle a Microsoft SQL Server. Nejprve se věnuje zjištěním základních rozdílů mezi oběma databázemi a to jak z pohledu uživatelských účtů, tak z pohledu databázových objektů (rozdíly v tabulkách, datových typech, integritních omezení, indexů a pohledů). Dále se zabývá možnostmi migrace jednotlivých databázových objektů z Oracle database do Microsoft SQL Serveru. Na závěr je analýza a dokumentace vlastního řešení a vytvoření PL/SQL balíčku, pomocí kterého bude možné migraci provést.

Klíčová slova

Oracle, Microsoft SQL Server, databáze, migrace, SQL, PL/SQL

Title

Oracle Database layer for communication with MSSQL

Annotation

This diploma thesis deals with the system migration between two platforms: Oracle and Microsoft SQL Server. The theoretical part describes differences between those platforms from user perspective as well as from the technical part (differences in tables, data types, integrity constraints, indexes and views). It also deals with the possibility to migrate database objects from Oracle database to Microsoft SQL Server. Finally, analysis and documentation of custom solutions and create a PL/SQL package, with which it will be possible to perform the migration.

Keywords

Oracle, Microsoft SQL Server, database, migration, SQL, PL/SQL

Obsah

Seznam zkratek

MSSQL	Microsoft SQL Server
SEQUEL	Structured English Query Language
SQL	Structured Query Language
SQUARE	Specifying Queries As Relational Expressions
PL/SQL	Procedural Language/Structured Query Language
T-SQL	Transact-SQL
SŘBD	Systém Řízení Báze Dat (systém pro správu databází)
RDBMS	Relational Database Management System
ANSI	American National Standards Institute
ISO	International Organization for Standardization
GCS	Geographic Coordinate System
PCS	Projected Coordinate System
IOT	Index-Organized Tables
XML	Extensible Markup Language
LOB	Large OBjects
ODBC	Open DataBase Connectivity
JDBC	Java DataBase Connectivity

Seznam obrázků

Seznam tabulek

Úvod

Databázové systémy jsou jedny z nejdůležitějších systémů velké části společností. Během svého rozvoje provozuje mnoho společností několik různých databázových systémů (ať už z důvodu odkoupení jiných společností, různých potřeb nezávislých pracovišť či z jiných důvodů), čímž vzniká heterogenní přístup k datům.

Vzniklý heterogenní přístup může být velký problém, data se duplikují, změny jsou třeba řešit na několika místech nezávisle na sobě, vzniká nekonzistentnost. Není jistota, že veškeré nové informace se promítnou správně do všech systémů.

Proto je vhodné provést sloučení databázových systémů do jediného nebo aspoň omezit jejich počet a sjednotit databázové platformy. To společnostem umožní jednodušší, efektivnější, levnější a bezpečnější správu podnikových dat.

K takovému úkonu je nejprve nutné zjistit rozdíly mezi jednotlivými databázovými systémy. Dále je důležité nalezení vhodného způsobu propojení mezi databázovými platformami pro samotnou migraci dat. Do doby, než se provede úplný přechod do jedné konsolidované databáze, je vhodné, aby různé heterogenní databázové systémy spolupracovaly. Díky tomu se mohou aplikace pomalu připravovat na konsolidaci obou systémů, kdy již bude část dat přesunuta a sloučena. K tomu slouží takzvané tunelování jednoho databázového systému skrz druhý, kdy se SQL dotazy posílají pouze do jedné databáze, která příkazy vyhodnotí, případně potřeby dat z jiných databází tyto data získá, transformuje na vhodné datové typy a data pošle zpět uživateli (aplikaci) jako by byly uložené na dané databázi.

Pro tuto práci byly vybrány dva z nejrozšířenějších databázových systémů: Oracle a Microsoft SQL Server (dále označován již jen jako MSSQL). Konkrétně půjde o migraci z Oracle database verze 11g do MSSQL verze 2008R2.

Práce se lehce dotkne historie a standardizace relačních databázových systémů, což by mělo dát odpověď na otázku, proč jsou databázové systémy dnes natolik podobné, ale přesto výrazně jiné. Dále se bude zabývat rozdíly mezi jednotlivými databázovými systémy z pohledu migrace Oracle do MSSQL, čímž vytyčí hranice možností migrace jednotlivých objektů.

V další části práce se hovoří o jednotlivých možnostech komunikace zvolených databázových systémů mezi sebou. Tato komunikace by měla být základem pro migraci dat.

V posední části práce je popsána analýza vhodného řešení migrace dat z Oracle databáze do MSSQL za pomocí PL/SQL balíčku a dokumentace k navrženému řešení.

1 Vznik jazyka SQL a počátky relačních databází

Ačkoliv dnes existuje standardizovaný SQL jazyk, který by měl zaručovat jednotnost všech relačních SQL databází, ve skutečnosti tomu tak není. Jazyk SQL již od svého vzniku existuje v mnoha dialektech (dalo by se říct, kolik je SQL databází, tolik je i dialektů SQL jazyka). Každá databázová firma se snaží přidat vlastní nápady pro co nejlepší uchovávání strukturovaných dat, následnou manipulaci, získávání informací a konečné zobrazení za použití co nejmenších nároků na paměť, výkon a čas. Tím postupně vzniká větší či menší nekompatibilita mezi jednotlivými systémy. S rostoucími možnostmi databázových systémů se jednotlivé rozdíly stále zvětšují (a to i přes snahu o standardizaci a sjednocení dialektů do jednoho univerzálního jazyka). Otázkou může být jak moc se výrobci opravdu snaží jednotlivé standardy dodržovat a implementovat je do svých databází a jestli jistá míra nekompatibility není vytvářena záměrně, aby nebylo příliš lehké migrovat na konkurenční platformy.

Relační databáze byla poprvé představena v práci v roce 1969 doktorem Edgarem F. Coddym, který pracoval jako výzkumný pracovník u IBM hledající nové způsoby zpracování velkého množství dat. Relační model pohlížel na data jako na tabulky a byl postaven na dvou základních matematických odvětvích: teorii množin a predikátové logice. Název *relační* vznikl z pojmu teorie množin - *relace* neboli vztah. Každá konkrétní tabulka realizuje podmnožinu kartézského součinu množin přípustných hodnot všech sloupců – relaci (relace je libovolný vztah mezi skupinou prvků jedné nebo více množin).

Firma IBM spustila výzkumný projekt na potvrzení životaschopnosti relačního modelu zvaný System R. V roce 1974 se podařilo vytvořit první minimální prototyp relační databáze. Po představení modelu bylo potřeba vyvinout jazyk pro dotazování, modifikaci dat a řízení databáze, který by jako základ využíval databázový systém podporující relační model. V roce 1974 dr. Donald D. Chamberlain a jeho tým představili jazyk Structured English Query Language (SEQUEL)[1]. Jazyk SEQUEL umožnoval uživatelům pomocí přesně definovaných vět v anglickém jazyku vytvářet dotazy do relační databáze. Následně byl implementován do prototypu relační databáze SEQUEL-XRM. V letech 1976 – 1977 byl jazyk přepracován a vydán jako SEQUEL/2, který byl následně z právních důvodů přejmenován na SQL (Structured Query Language). Firma IBM projekt System R ukončila v roce 1979 se závěrem, že relační model je použitelná databázová technologie s velkým komerčním potenciálem [2].

Implementace relačního modelu a jazyka SQL

Práce IBM na projektu vývoje relační databáze byly s velkým zájmem sledovány technickými médii a výrobci postupně začali pracovat na vlastních produktech založených na tomto modelu dřív, než obsadí firma IBM celý trh.

[1] Jazyk SEQUEL nebyl vyvinut úplně od základů, ale opíral se o jazyk SQUARE (Specifying Queries As Relational Expressions), který byl určen k implementaci relační algebry s větami ve stylu anglického jazyka.

V roce 1977 skupina techniků v Kalifornii založila společnost SDL (Software Development Laboratory) se záměrem vytvořit vlastní databázový podkut založený na relačním modelu a jazyku SQL nazvaný Oracle[2]. Firma se během krátké doby přejmenovala na Relational Software Inc. a následně na Oracle Corporation. Svůj první komerčně dostupný systém (taktéž první komerční produkt na světě) správy relačních databází (RDBMS) uvedla v roce 1979 jako Oracle 2.0 (první verze nebyla veřejnosti nikdy představena) [11].

Postupně vznikaly i další společnosti zabývající se relačním databázovým modelem např. Relational Technology s produktem INGRES představený v roce 1981 (předchůdce PostgreSQL). IBM vydala svůj komerční RDBMS až v roce 1982 [2].

Microsoft se v boji o vytvoření relačního databázového systému dlouho neangažoval a těžil z partnerství s firmou Sybase (společnost založena v roce 1984 a svoji první relační databázi uvedla v roce 1988). První verze, na které se Microsoft podílel svojí částí kódu, byla SQL Server 4.2, vydána v roce 1992. Následně Microsoft chce plně využít potencionálu nových Windows NT (kdežto Sybase chce být nadále multiplatformní) a partnerství je v roce 1994 rozpuštěno. První verze, která byla vyvinuta čistě firmou Microsoft bez cizí pomoci, ovšem stále na základě zdrojových kódů od Sysbase, byla Microsoft SQL Server 6.0 z roku 1995 [3][4][13].

V dnešní době jsou databázové systémy spíše objektově-relační. Relační model jako takový je většinou implementován pouze částečně. Existují však databázové systémy ve formě výzkumu, které 100% implementují relační model.

Standardizace SQL

Každý výrobce vylepšoval a vyvíjel SQL jazyk vlastním potřebám, čímž vznikaly postupně různé dialekty SQL jazyka. V roce 1982 zareagovala organizace ANSI (American National Standards Institute) na rostoucí potřebu oficiálního jazyka relačních databází. Pověřila technickou komisi databází X3H2 organizace X3, která byla složena z expertů databázového odvětví a z významných zástupců tvůrců databází využívajících SQL k vytvoření jednotného standardu. Komise vycházela především z dialektu systému DB2 společnosti IBM. Organizace ANSI ratifikovala vytvořený standart skupiny X3H2 v roce 1986 jako dokument „*ANSI X3.135-1986 Database Language SQL*“. Společnost ISO (International Organization for Standardization) schválila vlastní dokument, který přesně odpovídal dokumentu ANSI SQL/86 jako mezinárodní standard „*ISO 9075-1987 Database Language SQL*” v roce 1987. Oba standardy se obecně označují jako SQL/86 [2][13].

SQL/86 definuje minimální sadu požadavků, kterou by měli všichni výrobci splňovat. V zásadě se standardizovaly jen ty elementy, které si byly v různých dialektech SQL značně podobné a které již výrobci implementovali. Nový standard položil jasné základy, na kterých bude možné jazyk dále vyvíjet. Ve specifikaci však chyběly základní příkazy jak pro definici dat, tak i pro manipulaci s daty. Například nebylo standardizováno jak data vkládat (příkaz `INSERT`). Definice obsahovala příkazy pro vytvoření tabulky (příkaz `CREATE TABLE`), ale již neobsahovala příkazy na odstranění nebo změnu tabulky (příkazy `DROP`, `ALTER`). Proto

[2] Původně na zakázku pro americkou ústřední zpravodajskou službu CIA.

vznikly pozdější specifikace SQL/89 (*ANSI X3.168-1989 Database Language Embedded SQL*) a hlavně SQL/92 definovaný v dokumentech *X3.153-1992 Database Language SQL* a *ISO/IEC 9075:1992 Database Language SQL* v roce 1992, které jazyk SQL dále rozšiřovaly. Další revize jazyka SQL byla standardizována v roce 1999 v dokumentech *ISO/IEC 9075:1999* nazývaná jako SQL:1999 nebo SQL3. Jedním z hlavních cílů této revize bylo zahnutí objektově relačního modelu. V dalších letech postupně přicházejí další revize jazyka SQL: SQL:2003 (*ISO/IEC 9075:2003*), SQL2006 (*ISO/IEC 9075:2006*), SQL:2008 (*ISO/IEC 9075:2008*) a zatím nejnovější SQL:2011 (*ISO/IEC 9075:2011*) [2][12][13].

Vzhledem k tomu, že standardizace jazyka začala vznikat až v dobách, kdy už na trhu byly komerčně úspěšné databázové systémy, které se neustále vyvíjejí, není standardizace SQL jazyka a jeho implementace v rovnováze. Navíc standardizace je jen ve formě doporučení a záleží na výrobcích databázových systémů, jestli vydaná doporučení implementují, případně v jakém rozsahu. Standardizace tak většinou zaostává za implementací výrobců. Původně vybírala vhodné minimum z již existujících funkcí a řešení použitých jednotlivými výrobci. V dalších letech k tomu začala přidávat i vlastní návrhy na zlepšení.

Jednotliví výrobci nečekají na nové revize standardu, neustále přidávají nové vlastnosti a vylepšení do svých databázových systémů. K tomu se snaží (aspoň částečně) implementovat existující standardy SQL jazyka na již vytvořená vlastní řešení. Díky standardizaci SQL jazyka jsou jednotlivé systémy relačních databází aspoň do jisté míry kompatibilní a programátorům se zjednodušuje celkový pohled na SQL jazyk. Neustálý vývoj a konkurenční boj o zákazníky (pro který je jistá míra nekompatibility výhodná) jednotlivé databázové systémy od sebe opět oddaluje. Databázové systémy nejsou plně kompatibilní ani přes existenci standardizačního SQL jazyka a pravděpodobně ani nikdy nebudou.

2 Rozdíly databázových systémů Oracle a MSSQL

V následující kapitole jsou podrobněji rozebrány základní rozdíly mezi databázovými systémy Oracle a MSSQL. Jednotlivé rozdíly jsou brány především z pohledu přípravy na následné řešení problému migrace ze systému Oracle na MSSQL.

2.1 Rozdíly v pojetí uživatelů a schémat

2.1.1 Oracle database

Uživatelský účet

Přístup k databázi je zajištěn pomocí uživatelského účtu, většinou nazývaný jako *uživatel*. Každý uživatel v Oracle databázi je definovaný uživatelským jménem a uživatelskými atributy. Základními atributy jsou: uživatelské heslo, uživatelské role a oprávnění, základní tabulkový prostor (default tablespace) pro vlastní databázové objekty, dočasný tabulkový prostor (temporary tablespace), uživatelský profil a kvóty.

Uživatel může existovat v databázi i bez toho, aby vlastnil databázové objekty uložené v jeho schématu, a může používat pouze objekty jiných uživatelů.

Nového uživatele vytvoříme pomocí příkazu `CREATE USER`.

Tabulkový prostor

Data jsou v Oracle uložena v tabulkových prostorech. Jedná se o logické úložiště dat v Oracle databázi, které se skládá z jednoho či více fyzických datových souborů. Každý uživatel má přidělený defaultní (výchozí) tabulkový prostor, kam se standardně ukládají jednotlivé uživatelské objekty. Uživatelé mají přiřazené ke každému tabulkovému prostoru omezující kvóty, které určují, kolik mají v daném prostoru vyhrazeného místa. Po vytvoření nového uživatele nemá právo zápisu do žádného tabulkového prostoru a to ani defaultního (změnu kvót k jednotlivým tabulkovým prostorům je nutné udělit přímo danému uživateli, příkazem `ALTER USER`). Pokud má uživatel přiděleny volné kvóty k více tabulkovým prostorům, může uživatel ukládat svá data do libovolných tabulkových prostorů při použití klauzule `TABLESPACE`. Pro čtení jsou tabulkové prostory transparentní.

Defaultní tabulkový prostor uživatele lze změnit pomocí příkazu `ALTER USER` (projeví se pouze pro objekty vytvořené po tomto příkazu).

Nový tabulkový prostor lze vytvořit příkazem `CREATE TABLESPACE`.

Schéma

Pojem schéma v Oracle představuje sadu všech tabulek, pohledů, sekvencí, procedur, balíčků, … tedy všech databázových objektů jednoho uživatele. Schéma obsahuje všechny databázové objekty, které vlastní jediný uživatel. Schéma se vytváří automaticky příkazem `CREATE USER`[3] spolu s uživatelem a sdílí s ním stejný název. Uživatel schématu má vždy

[3] Existuje lehce matoucí příkaz `CREATE SCHEMA`, který však nevytváří nové schéma jak by se mohlo zdát, ale umožňuje současně provést skupinu příkazů `CREATE TABLE`, `CREATE VIEW`, `GRANT` a vytvořit několik objektů jako jednu transakční událost. Tím je zaručeno, že proběhne „všechno nebo nic“.

plnou kontrolu nad objekty ve svém vlastním schématu. Jednotlivé objekty vidí pouze jejich vlastník, případně správce databáze. Je ovšem možné přidělit přístup ke svým objektům ostatním uživatelům pomocí příkazu `GRANT` (přidělit přístup k celému schématu není možné, musí se pro veškeré objekty přidělit zvlášť). Pokud má uživatel přidělená potřebná oprávnění, může přistupovat k objektům do jiných schémat. Na tabulku v jiném schématu se odkáže zápisem `schéma.tabulka`, případně vytvořeným synonymem.

Z důvodu podobnosti termínů uživatel a schéma a hlavně z důvodu navázání právě jednoho uživatele na právě jedno schéma se shodným jménem, dochází často k záměně pojmů, nebo spíše k zjednodušování a používání pouze jednoho termínu pro oba pojmy.

Pokud je potřeba oddělit data od uživatelů, obvykle se vytvoří uživatel (schéma), pod kterým jsou vytvořené tabulky a další objekty aplikace. Vedle toho jsou vytvořené další účty jednotlivých skutečných uživatelů, kteří následně dostanou příslušná práva pro přístup k tabulkám aplikace. Uživatelé tak můžou mít práva vidět data, případně je editovat, ale již nebudou mít možnost měnit strukturu tabulek. Pokud je nutné skrýt data jiných uživatelů přistupujících do jedné tabulky, případně skrýt pouze některé sloupce, je možné toho docílit například pomocí techniky skrývání: do každé tabulky do samostatného sloupce se vloží jméno uživatele (a automaticky jej lze plnit například pomocí triggeru) a následně vyfiltrovat důležitá data pomocí vhodně vytvořeného pohledu, který vrací pouze vlastní data. Uživatel nebude mít k originální tabulce vůbec přístup. Další možností je využití Virtual Private Database (VPD), které umožní vytvořit zásady zabezpečení pro řízení přístupu k databázi na úrovni řádků a sloupců.

Role a oprávnění

Role je v Oracle sadou oprávnění a zároveň i typem přístupu do databáze. Rolím se přidělují konkrétní oprávnění (případně další role) a následně se jednotlivé role přidělují příslušným uživatelům[4], tím uživatelé získají veškerá oprávnění plynoucí z dané role. (Jedna role může být přidělena více uživatelům a jednomu uživateli může být přiděleno více rolí, vztah M:N.) Případně uživatel může udělovat oprávnění (má-li na to práva) jiným uživatelům přímo, bez využití rolí.

Oprávnění dělíme na dvě skupiny: systémová a objektová. Systémová oprávnění umožňuje uživateli provádět specifické sady příkazů[5]. Jako příklad systémových oprávnění lze uvést dvě oprávnění: `CREATE SESSION`, které je nutné pro samotné přihlášení uživatele do databáze[6] a `CREATE TABLE` pro oprávnění vytvářet tabulky ve svém schématu. Systémová oprávnění může udělit pouze správce databáze nebo uživatel, který dané oprávnění dostane

[4] Pouhým přidělením role (pomocí příkazu GRANT) se uživateli role implicitně neaktivuje, roli musí aktivovat příkazem SET ROLE, případně pomocí příkazu ALTER USER uložit roli jako defaultní, čímž se zaručí její automatická aktivace role při přihlášení.

[5] Pro příkazy SELECT, INSERT, UPDATE, DELETE spouštěných nad vlastními objekty není nutné žádné speciální oprávnění.

[6] Každý uživatelský účet, pod kterým se chceme přihlásit, musí mít minimálně oprávnění CREATE SESSION. Bez tohoto oprávnění bude přihlášení do databáze zamítnuto i při zadání správného uživatelského jména a hesla. Samotný účet ovšem zůstane nadále plně funkční a přístup k jeho databázovým objektům bude přes jiné uživatele (mající příslušná práva) nadále umožněn, na rozdíl od uzamčeného účtu.

s klauzulí `WITH GRANT OPTION`. Objektové oprávnění určuje, jaké objekty bude mít uživatel přístupné a jaké operace bude moci na nich vykonat. Tento typ oprávnění uděluje uživatel daných objektů rolím či ostatním uživatelům, (nebo předá udělené oprávnění, které získal s klauzulí `WITH GRANT OPTION`).

Role se vytváří pomocí příkazu `CRETE ROLE`. Oprávnění (nebo role) se přidělují konkrétním uživatelům (nebo jiným rolím) pomocí příkazu `GRAND` a naopak příkazem `REVOKE` se přidělená oprávnění a role odebírají.

Profil

Databáze musí spravovat a alokovat prostředky, pro všechny databázové uživatele, které nejsou neomezené. Omezení těchto prostředků se dá vynutit pomocí profilů. Profil je tedy skupina vlastností přidělená jednomu, či více uživatelům. Ve výchozím nastavení je každému uživateli přidělen profil `DEFAULT`.

Profily pomáhají lépe vyladit víceuživatelské prostředí, případně slouží jako další bezpečnostní politika. Umožňují nastavovat politiku hesel nebo jiné omezující kroky vůči uživateli, jako je například čas využití CPU, počet souběžných relací, čas připojení, aj.[7]

Nový profil se vytvoří příkazem `CREATE PROFILE` a následně se může přiřadit uživatelům pomocí příkazu `ALTER USER`.

Shrnutí uživatelských údajů na Oracle database:

- Uživatelský účet (uživatel) – umožnuje přihlášení k databázi, je mu automaticky přiděleno právě jedno schéma, vlastní oprávnění (přímo nebo pomocí rolí), určuje prostor a kvóty na ukládání vlastních dat.
- Schéma – kontejner objektů vlastněn právě jedním uživatelem, vlastnictví je nepřenositelné.
- Role – sdružuje systémová a objektová oprávnění.
- Profil – spravuje systémové prostředky databáze.

2.1.2 MSSQL

MS SQL Server je na rozdíl od většiny konkurenčních produktů, které fungují na různých operačních systémech jednoplatformový, úzce svázaný s operačním systémem Microsoft Windows. Pro autentifikaci a autorizaci může využívat dvě metody: klasické přihlašování pomocí jména a hesla, nebo integrované zabezpečení systému Windows, které umožnuje MSSQL sdílet uživatelská jména a hesla používaná ve Windows.

Většina objektů týkajících se zabezpečení (uživatelské účty a role) se na MSSQL vyskytují ve dvou úrovních: jednak pro databázový server a jednak samostatně pro jednotlivé databáze. Vzhledem k tomu, že jednotlivé databáze jsou ještě pod správou serveru, některé objekty dědí zabezpečení od serveru.

[7] Omezení velikosti přiděleného úložného prostoru se neuvádí v profilu, ale uděluje se přímo uživatelskému účtu pomocí nastavení příslušných kvót u jednotlivých tabulkových prostorů.

Uživatelský účet (Login)

Uživatelský účet slouží pouze pro přihlášení (povolení přístupu) do MSSQL a obsahuje některé základní atributy. Nový uživatelský účet na úrovni MSSQL serveru se vytvoří příkazem `CREATE LOGIN` se způsobem autentifikace a základními atributy jako je jméno uživatelského účtu a heslo, název defaultní databáze (ta bude pro uživatele výchozí), serverové role a případně další parametry.

Uživatel (User)

K uživateli se přiřadí oprávnění daného uživatelského účtu k určité databázi. Pro vytvoření uživatele na úrovni databáze slouží příkaz `CREATE USER`. Uživatel k jednotlivým databázím může být vytvořen pouze tehdy, existuje-li uživatelský účet pro MSSQL se shodným uživatelským jménem. Uživatel uchovává také další důležité informace, například výchozí schéma a databázové role.

Role

Role jsou důležitou součástí přístupových práv a umožnují sdružovat uživatele do skupin. Mezi rolemi a uživateli je vztah typu M:N. Role se na SQL Serveru dělí na dvě skupiny.

Serverové role se týkají operací na úrovni serveru. Jsou předdefinované a nemohou být uživatelsky upravovány (např.: sysadmin, securityadmin, dbcreator, …). Role se přidělují uživatelskému účtu a obsahují práva, která se vztahují na celý databázový systém. Tyto role se přidělují jen těm uživatelům, kteří mají vykonávat nějakou správu nad celým serverem, běžnému uživateli nemusí být žádná serverová role přiřazena.

Databázové role (existují jak předdefinované, tak uživatelsky definované) jsou specifické pro danou databázi a umožnují přístup k dané databázi v rozsahu, který je určený danou rolí. Databázové role se dále větví na „klasické“ databázové role a aplikační. Aplikační role se nepřiřazují uživatelům přímo, ale může ji dočasně aktivovat jakýkoliv uživatel, který zná povinné přístupové heslo. Aktivováním aplikační role se uživateli ztratí jeho standardní databázová práva a využívá pouze nová práva určená aplikační rolí.

Nové databázové role se vytváří příkazem `CREATE [APPLICATION] ROLE` a následně se pomocí příkazu `GRANT` přidělí roli jednotlivá oprávnění.

Schéma

Schémata slouží jako oddělující vrstva mezi uživatelem databáze a objekty v rámci databáze. Existují nezávisle na uživatelích a uživatelé nezávisle na schématech. Uživatelé databáze vlastní schémata (každý uživatel má přednastavené jedno defaultní schéma, ale může přistupovat ke všem schématům, ke kterým má přístupová práva), která vlastní jednotlivé databázové objekty. Pro použití jiného schématu (kromě defaultního) se musí použít dvojdílný název `schema.objekt`. Název schématu tedy nahrazuje jméno uživatele. Uživatel může vlastnit několik schémat, ovšem jedno schéma může být vlastněno v jednom okamžiku pouze jediným uživatelem. Pokud by uživatelé vlastnili objekty přímo (podobně jako je tomu v Oracle) není možné zrušit uživatele bez předchozího odstranění či převedení všech jeho objektů jinému vlastníkovi. Převedením objektů jinému vlastníkovy by se zároveň změnil i název daného objektu. Díky tomu, že je mezi uživateli a objekty umístěno

schéma, lze uživatele z databáze odstranit, aniž by to mělo vliv na název objektu nebo na funkčnost aplikací.

Nové schéma lze vytvořit pomocí příkazu `CREATE SCHEMA`. Přiřazení schéma uživateli se následně provádí pomocí příkazu `ALTER AUTHORIZATION`.

Shrnutí zabezpečení údajů na MS SQL serveru:

- Login - přihlášení k MS SQL Serveru.
- User - přístup k jednotlivým databázím.
- Role - oprávnění používat jednotlivé příkazy.
- Schéma - kontejner objektů, vlastní user, vlastnictví je přenositelné.

2.2 Základní datové typy

2.2.1 Oracle database

Základní datové typy pro Oracle jsou uvedeny v tabulce (*Tabulka 1*) [6][10].

Tabulka 1 – Základní datové typy Oracle

Datový typ	Popis
NUMBER(p, s), DECIMAL(p, s)	Sloupec číselných hodnot o přesnosti *p* a rozsahu *s*, *p* (1, 38), *s* (-84, 127), uloženo jako NUMBER(p, s).
FLOAT(p)	Reálné číslo (floating point) s počtem platných binárních číslic *p*, max. 126 (38 desítkově).
DOUBLE PRECISION	Reálné číslo, uloženo jako FLOAT(126).
REAL(p)	Reálné číslo, uloženo jako FLOAT(63).
INTEGER, INT, SMALLINT	Celé číslo +–2 147 483 648, uloženo jako NUMBER(38).
BINARY_FLOAT	32bitové číslo s plovoucí desetinou čárkou.
BINARY_DOUBLE	64bitové číslo s plovoucí desetinou čárkou.
CHAR(size [CHAR])	Znaková data o pevné délce *size* bajtů nebo znaků, (max. 2000 B).
NCHAR(size)	CHAR umožňující uchování znaků národních znakových sad, (max. 2000 B).
VARCHAR2(size [CHAR])	Řetězec o proměnné délce s max. délkou *size* bajtů nebo znaků, (max. 4000 B).
NVARCHAR2(size [CHAR])	Řetězec o proměnné délce s max. délkou size bajtů (znaků), (max. 4000 bytes). V závislosti na volbě národní znakové sady.
DATE	Vyjádření kalendářního datu a času s přesností na sekundy.
TIMESTAMP(precision)	Kalendářní datum a čas. *Precision* označuje počet číslic reprezentující zlomky sekund (0-9).
TIMESTAMP WITH TIME ZONE	Datový typ TIMESTAMP s údajem o časovém posunu.
TIMESTAMP WITH LOCAL TIME	Datový typ TIMESTAMP normalizovaný podle lokální datové zóny.
INTERVAL YEAR(precision) TO MONTH	Časový interval v rocích a měsících, *precision* označuje počet číslic v poli YEAR (0-9).
INTERVAL DAY (day_precision) TO SECOND (second_precision)	Časový interval ve dnech, hodinách, minutách a sekundách. *Day_precision* určuje maximální počet číslic reprezentující dny

	(0-9), *seconds_precision* určuje počet číslic reprezentující zlomky sekund (0-9).
LONG	Zastaralý typ, znaková data o proměnné délce max. 2GB.
LONG RAW	Zastaralý typ, binární data o proměnné délce max. 2GB.
BLOB	Binární objekty, 8 až 128 TB. Maximální hodnota určená vztahem: `((4 GB - 1)*(DB_BLOCK_SIZE))`.
CLOB	Znakový objekt, 8 až 128 TB. Maximální hodnota určená vztahem: `((4 GB - 1)*(DB_BLOCK_SIZE))`.
NCLOB	CLOB obsahující znaky národních abeced, 8 až 128 GB.
BFILE	Ukazatel na externí binární soubor, pouze pro čtení, délku omezuje operační systém.
XMLType	Sloupec obsahující XML data, max. 4 GB.
ROWID	Hexadecimální řetězec reprezentující jedinečnou adresu řádku v databázi (10 B).
UROWID	Univerzální ROWID - Hexadecimální řetězec pro logickou či fyzickou adresu řádku (max. 4000 B).

Vlastní datové typy

Díky objektové podpoře umožnuje Oracle vytvářet vlastní (objektové) datové typy tvořené příkazem `CREATE TYPE`. Vlastní datové typy umožnují zabalit více základních typů do jednoho logického celku a případně k nim připojit další vlastnosti známé z objektového programování, jako jsou konstruktory, členské metody, statické metody, map a order metody, podpora dědičnosti, ukazatele na objekty… Objektové typy je následně možné používat v tabulkách obdobně jako základní datové typy.

Oracle umožňuje vytváření kolekce (seznamy) jako rozšíření vlastních datových typů. V Oracle existují 3 typy kolekcí:

- Pole o proměnné délce – umožňuje ukládat konečný počet prvků stejného datového typu, je nutné dopředu znát maximální počet prvků v poli, přístup pomocí celočíselného indexu.
- Vnořené tabulky – obdoba pole s variabilním počtem hodnot, jedná se o takzvanou tabulku v tabulce, přístup pomocí celočíselného indexu.
- Asociativní pole – pole s indexem (indexem může být číslo nebo řetězec), u kterého není třeba znát počet prvků, přístup pomocí zadaného indexu. Tento druh kolekce je pouze v PL/SQL.

```
-- Základní schéma pro vytvoření jednoduchého vlastního datového
typu v Oracle a jeho:

CREATE [OR REPLACE] TYPE [schema.]object_type_name AS OBJECT (
  <instanceVariables> {sql_or_plsql_datatype}
  [,…]
);
```

```
-- Použití vlastního datového typu v tabulce:

CREATE TABLE tab (
  id NUMBER(4),
  typ OBJECT_TYPE_NAME
  [,…]
);
```

Oracle Spatial

Oracle Spatial je integrovaná množina funkcí a procedur v databázi, které usnadňují ukládání, získávání, aktualizace a vyhledávání prostorových prvků v databázi Oracle[8]. Oracle Spatial umožňuje ukládat geometrická data (jednotlivé body, úsečky, n-bodové polygony, kružnice a jejich části, optimalizované obdélníky, případně jejich kombinace) uložená v různých souřadnicových systémech: kartézský, geografický (GCS), předpokládaný (PCS), nebo místní.

2.2.2 MSSQL

Základní datové typy pro MS SQL Server jsou uvedeny v tabulce (*Tabulka 2*) [3][7].

Tabulka 2 – Základní datové typy MSSQL

Datový typ	Rozsah	Velikost
TINYINT	0 až 255	1 B
SMALLINT	-2^15 (-32 768) až 2^15-1 (32 767)	2 B
INT	-2^31 (-2 147 483 648) až 2^31-1 (2 147 483 647)	4 B
BIGINT	-2^63 až 2^63-1	8 B
DECIMAL(p, s), NUMERIC(p, s)	Kde *p* = <1; 38>, 0 <= *s* <=*p*	5-17 B
MONEY	-2^63 do 2^63-1, přesností na desetitisíciny	8 B
SMALLMONEY	-2^31 (- 214,748.3648) až -2^31-1 (214,748.3647)	4 B
FLOAT(n)	Kde *n* je počet bitů mantisy (1-24 značí přesnost 7 číslic a velikost 4 B, 25-53 značí přesnost 15 číslic a velikost 8 B)	4-8 B
REAL	Definovaný jako float(24)	4 B
CHAR(n)	Znakový řetězec pevné délky a max. délce 8000 znaků.	max. 8000 B
VARCHAR(n)	Znakový řetězec proměnlivé délky a max. délce 8000 znaků.	
NCHAR(n)	Znakový Unicode řetězec pevné délky a max. délce 4000 znaků.	
NVARCHAR(n)	Znakový Unicode řetězec proměnlivé délky a max. délce 4000 znaků.	
VARCHAR(max), NVARCHAR(max)	Řetězce znaků do max. velikosti 2GB.	max. 2 GB
Znakové datové typy využívající znakovou sadu Unicode začínají písmenem n, ostatní jsou ve zvolené znakové sadě ANSI.		

[8] Oracle Spatial je součástí Enterprise Edition Oracle Database 11g. Pro nižší verze existuje zjednodušená podskupina pro práci s prostorovými daty, nazývaná Oracle Locator. Podrobnější informace o Oracle Spatial lze nalézt v dokumentaci: http://docs.oracle.com/cd/E11882_01/appdev.112/e11830/toc.htm.

SMALLDATETIME	1. 1. 1900 až 6. 6. 2079, přesnost 1 minuta	4 B
DATETIME	1. 1. 1753 až 12. 31. 9999, přesnost 0,00333 sekundy	8B
DATETIME2	1. 1. 0001 až 12. 31. 9999, přesnost 100 nanosekund	6-8 B
DATETIMEOFFSET	1. 1. 0001 až 12. 31. 9999, přesnost 100 nanosekund	8-10 B
DATE	1. 1. 0001 až 12. 31. 9999, přesnost 1 den	3B
TIME	0 až 23:59:59.9999999, přesnost 100 nanosekund	3-5 B
BIT	Pouze hodnoty `1`, `0` a `null`	1 b
BINARY(n)	Binární data s pevnou šířkou o délce *n* bajtů	až 8000 B
VARBINARY(n)	Binární data s poměrnou šířkou o maximální délce *n* bajtů	až 8000 B
VARBINARY(max)	Binární data s poměrnou šířkou o maximální délce až 2 GB	až 2 GB

Data typu XML

Datový typ `XML` umožňuje nativně ukládat XML dokumenty do velikosti 2 GB dat. Jeden dokument může obsahovat nejvýše 128 úrovní.

Data typu filestream

Slouží k ukládání nestrukturovaných dat označovaných jako `BLOB` (Binary Large Object). `FILESTREAM` provádí kontrolu a správu dat typu `BLOB`, ale data se nacházejí v souboru operačního systému.

Do databáze se neuloží celý obsah dokumentu, ale jenom odkaz na něj. Samotný dokument se uloží do administrátorem definovaného a databázovým serverem spravovaného prostoru v souborovém systému.

Pro využití datového typu `FILESTREAM` je nutné, aby server disponoval souborovým systémem NTFS. Maximální velikost uloženého dokumentu je omezena pouze limity souborového systému a volnou kapacitou úložiště. Integrovaná správa zabezpečuje, že úložiště je součástí systému a při zálohování budou zálohovány (případně obnoveny) i dokumenty v souborech [6][10].

Prostorový datový typ

`GEOMETRY` – založen na euklidovské geometrii, slouží k ukládání bodů, čar, křivek a polygonů.

`GEOGRAPHY` – struktura dat vycházející z elipsoidu, umožňuje ukládat hodnoty typu zeměpisné šířky a délky.

Datový typ HierarchyID

Datový typ `HIERARCHYID` umožňuje strukturovat hierarchická data.

Uniqueidentifier

`UNIQUEIDENTIFIER` je datový typ pro uložení GUID (Globally Unique IDentifier). Jedná se o identifikátor, který je jednoznačný v místě a čase. Obsahuje 16 b binární hodnotu

v hexadecimálním tvaru `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`, novou hodnotu lze získat pomocí funkce `NEWID()`.

Vlastní datový typ

MSSQL nemá přímou podporu objektových typů, umožňuje sice vytvářet vlastní datové typy, ale jejich použití není natolik univerzální jako v Oracle. Nejčastěji se používají jako vstupní parametr pro funkce a uložené procedury. Tento typ nelze uložit jako objekt do tabulky, nemůže obsahovat členské metody a ani nemá jiné objektové vlastnosti.

2.3 Pravidla pro pojmenování objektů

Pravidla pro pojmenování objektů v Oracle:

- Název musí mít od 1 do 30 bajtů (znaků).
- Musí začínat písmenem a obsahovat pouze alfanumerické znaky a znaky _, $, #.
- Nesmí se jednat o vyhrazená slova.
- V jednom jmenném prostoru se nesmí opakovat stejný název.
- Nezáleží na velikosti písma (case-insensitive), v databázi budou uloženy s velkými písmeny (pokud název není uveden v uvozovkách, nedoporučuje se).
- Názvy, které nesplňují pravidla na identifikátory, je možné použít, avšak takové případy se nedoporučují a je nutné takové názvy opatřit uvozovkami.

Pravidla pro pojmenování objektů v MS SQL:

- Mohou obsahovat nejvýše 128 znaků.
- Prvním znakem musí být písmeno.
- Nesmí se jednat o vyhrazené slovo jazyka T-SQL.
- Nesmí obsahovat mezery nebo speciální znaky.
- Znaky, které nesplňují pravidla na identifikátory, je nutné opatřit uvozovkami nebo hranatými závorkami.

2.4 Rozdíly v tabulkách

V relačních databázích je základním prvkem tabulka. Její základní implementace je díky standardizaci SQL (až na datové typy) jednotná, implementace pokročilejších voleb jsou již u většiny databázových systémů odlišné. Často jednotlivé databáze vůbec nemají proti sobě shodný (nebo aspoň funkčně podobný) protějšek. Viz dokumentace k příkazu `CREATE TABLE` pro systém Oracle[9] a MS SQL Server[10].

Databáze většinou kromě jednoduchých základních tabulek podporují další, pokročilejší, typy, které mají lepší vlastnosti pro specifické úkony. Platforma Oracle umožnuje vytvářet tyto druhy tabulek:

- „Základní“ relační tabulky.

[9] http://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_7002.htm

[10] http://msdn.microsoft.com/en-us/library/ms174979.aspx

- Temporary tables (dočasné tabulky) – tabulky s dočasnými daty.
- Index-organized tables (indexově orientované tabulky) – tabulky ukládající svá data přímo do indexové struktury.
- Object tables (objektové tabulky) – pro uchovávání objektových datových typů, každý řádek je řádkovým objektem.
- XMLType tables (tabulky datového typu `XMLType`) – XML tabulka s jedním sloupcem typu XML.
- External tables (externí tabulky) – umožní přistupovat ke zdroji dat, kterým je textový soubor uložený na disku, jako k tabulce z databáze.
- Cluster tables (seskupené tabulky) – cluster je skupina tabulek fyzicky svázaných k sobě za použití speciálního (nejčastěji primárního) klíče a uložená do stejné diskové oblasti.
    - Hash clusters tables (hašované seskupené tabulky) – speciální typ seskupené tabulky. Pomocí hodnot sloupce cluster zjišťuje fyzické umístění, ve kterém se řádek nachází.
    - Sorted hash clusters tables (řazené hašované seskupené tabulky) – umožnují, aby byly tabulky navíc seřazeny.
    - Single-table hash clusters (hašované clustery s jedinou tabulkou).
- Partition tables (rozdělené tabulky) – fyzické rozdělení tabulky na menší oddíly.
- Nested tables (vnořené tabulky) – tabulka vnořená do jiné tabulky.

V následujících kapitolách se budu zabývat jednotlivými druhy tabulek pro platformu Oracle. Ke každému druhu bude následně uvedena informace, zda daný druh podporuje i MSSQL, případně zda alespoň existuje druh podobný.

2.4.1 Heap-organized table (haldově uspořádané tabulky)

Jedná se o základní relační tabulky, které jsou haldově uspořádané. Obecně se nazývají pouze jako „tabulky“ bez další specifikace (aby v tomto přehledu byly jednotlivé druhy tabulek odlišené, bude se o těchto tabulkách psát jako o „základních tabulkách“). Tento typ tabulek je základním stavebním prvkem všech relačních databází.

Oracle umožňuje použití v tabulkách různých druhů sloupců:

- sloupce se základními datovými typy,
- virtuální sloupce,
- sloupce s objektovými typy (sloupcové objekty),
- ukazatelé na objekt (REF),
- kolekce (pole a vnořené tabulky).

MSSQL vzhledem k tomu, že na rozdíl od Oracle není plně objektově-relační databází, nepodporuje vlastní objektové typy a neumožňuje použití sloupcových objektů, ukazatelů na objekty ani kolekcí.

Základní tabulky se vytvářejí na databázových systémech Oracle i v MSSQL stejně, pomocí příkazu `CREATE TABLE`.

```
-- Obecná syntaxe pro tvorbu tabulek:

CREATE TABLE <tableName> (
  <columnName> <tataType> [<inlineConstraint>] [DEFAULT <value>]
  [, …]
  [<outOfLineConstraint> [, …]]
);

<column_constraint> = [CONSTRAINT <name>] {
  [NOT] NULL |
  [NOT NULL] UNIQUE | [NOT NULL] PRIMARY KEY |
  REFERENCES <tableName> [(<columnName>)] | CHECK (<condition>)
}

<table_constraint> = [CONSTRAINT <name> ] {
  UNIQUE (<seznam_sloupců>) |
  PRIMARY KEY (<seznam_sloupců>) |
  CHECK (<condition>) |
  FOREIGN KEY (<columns>) REFERENCES <tabulka> [(<columns>)]
}
```

2.4.2 Temporary tables (dočasné tabulky)

Oracle považuje temporary tables jako trvalé skladiště dočasných dat. Dočasné je v tomto případě jen uložení samotných dat a nikoliv samotná tabulka, která zůstává trvale. Po vypršení platnosti dat se odstraní veškerá uložená data, ale tabulka je stále připravena k dalšímu použití.

U temporary tables každý uživatel vidí a manipuluje pouze s daty, které sám uloží. Data ostatních uživatelů jsou pro něj neviditelná. U tabulky je možné specifikovat dvě události, po kterých budou data z tabulky automaticky vymazána – po ukončení transakce nebo relace.

```
-- Temporary table pro Oracle:

CREATE GLOBAL TEMPORARY TABLE <tempTableName>(…)
  ON COMMIT {PRESERVE | DELETE} ROWS;
```

V MSSQL jsou dočasná jak data v tabulkách, tak i samotné tabulky. S koncem platnosti je tedy odstraněna celá tabulka a pro další použití je nutné tabulku znovu vytvořit.

MSSQL rozděluje dočasné tabulky na lokální a globální, čímž definuje platnost dat a vlastně celých tabulek. Lokální se definuje jednoduše pouze prefixem „#“ u jména tabulky, zbytek definice je shodný se základními tabulkami. Tento typ tabulky je viditelný pouze v aktuální relaci a trvá pouze po dobu aktuální procedury nebo dávky. Globální se definuje prefixem „##“ u jména tabulky. Tabulka je viditelná pro všechny uživatele a její platnost končí při ukončení relace.

```
-- Temporary table pro MSSQL:

CREATE TABLE #<localTempTable> (…); -- lokální dočasná tabulka
CREATE TABLE ##<globalTempTable> (…); --globální dočasná tabulka
```

2.4.3 Index-organized tables (IOT, indexově orientované tabulky)

Oracle ukládá data IOT jako index v B*-stromové struktuře v pořadí hodnot primárního klíče. To znamená, že kromě primárního klíče jsou zde uloženy i hodnoty ostatních sloupců. Každý záznam je dvojice (primární_klíč; ostatní_data).

Indexově orientovaná tabulka musí mít specifikovaný primární klíč, který jednoznačně určuje každý řádek a podle kterého se data do tabulky ukládají a rovnají. Pro vytvoření tabulky stačí na konec příkazu `CREATE TABLE` přidat klauzuli `ORGANIZATION INDEX`.

```
-- Syntaxe pro vytvoření index-organized tables:

CREATE TABLE <iot_name> (
  <column1> NUMBER
  [, <column2> NUMBER] [, …]
  PRIMARY KEY(<column> [, …])
) ORGANIZATION INDEX;
```

MSSQL index-organized tables nenabízí.

2.4.4 Object tables (objektové tabulky)

Oracle umožňuje používání objektových typů. Objekty můžeme ukládat v objektové tabulce, která umožňuje uložit pouze jediný typ předem deklarovaného objektu a každý řádek je tzv. řádkovým objektem. Hlavní výhoda tohoto řešení spočívá v tom, že každý řádek má vlastní OID (objekt ID) a díky tomu se lze na řádky odkazovat prostřednictvím jiných objektů v databázi.

Nejprve je nutné vytvořit objektový typ, který bude sloužit jako podkladový a následně na jeho základě můžeme vytvořit objektovou tabulku.

```
-- Tvorba objektového typu a objektové tabulky pro Oracle:

CREATE TYPE <objectName> AS OBJECT
  (<attributes and methods object type>);

CREATE TABLE <objectTableName> OF <objectName>;
```

MSSQL neumožňuje používání objektů, tedy nezná ani objektové tabulky.

2.4.5 XMLType tables (tabulky datového typu XMLType)

XMLType table obsahuje jediný sloupec, který je typu `XMLType` a umožňuje uchovávat libovolný XML dokument. Tento dokument může být dále volitelně určen XML schématem. Pomocí XMLSchema se při vkládání validují jednotlivé řádky a nevyhovující řádky se do tabulky nepodaří vložit. XMLSchema je třeba nejprve registrovat pomocí procedury `DBMS_XMLSCHEMA.registerSchema()`, až poté může být použito v tabulce.

```
-- Vytvoření XMLType table s XML schématem pro Oracle:

EXECUTE DBMS_XMLSCHEMA.registerSchema(
  SCHEMAURL => 'documentation/dok.xsd', -- identifikace schéma
  SCHEMADOC => bfilename('<directories>','<documentName.xsd>'),
  LOCAL => TRUE,
  CSID => nls_charset_id('AL32UTF8')
);

CREATE TABLE <XMLTypeTableName> OF XMLTYPE
  XMLSCHEMA "<documentation/dok.xsd>" ELEMENT "<rootElement>";
```

V MSSQL není přímo tabulka typu XML. Použití jednoduché jednosloupcové tabulky se sloupcem typu XML vykazuje téměř shodné výsledky. Dovoluje také validaci pomocí XML schéma. (Vytvoření schéma zajistí příkaz `CREATE XML SCHEMA COLLECTION`.)

```
-- Vytvoření XML tabulky s XML schématem pro MSSQL:

CREATE TABLE <XMLTableName> (<XmlData> XML [(<XmlSchema>)]);
```

2.4.6 External tables (externí tabulky)

External tables umožní přistupovat k externímu zdroji dat, kterým může být textový nebo binární soubor uložený na disku, jako by se jednalo o tabulku uvnitř databáze. Taková tabulka má některá omezení, jako je například nemožnost řádky tabulky aktualizovat, odstraňovat nebo indexovat.

Soubory musí být uloženy v adresáři, který se následně zpřístupní pro databázi. V Oracle se adresář na disku mapuje pomocí příkazu `CREATE DIRECTORY`. Poté je třeba přidělit práva pro čtení (případně i pro zápis) daného adresáře.

Následně je již možné vytvořit externí tabulku pomocí příkazu `CREATE TABLE (…) ORGANIZATION EXTERNAL (…)` s parametry daného souboru. Nejzajímavější parametry jsou uvnitř klauzule `ACCESS PARAMETERS`, které specifikují formát souboru, především jak je specifikován ukončující znak jednotlivých polí, řádků a v případě pevné šířky sloupců počet znaků jednotlivých sloupců.

```
-- Vytvoření directory, přidělení práv a vytvoření external
tables z textového souboru s proměnnou délkou:

CREATE DIRECTORY <directoryName> AS '<pathToDirectory>';

GRANT READ [, WRITE] ON DIRECTORY <directoryName> TO <user>;

CREATE TABLE <externalTableName> (
  <column> <type>
  [, …]
)
ORGANIZATION EXTERNAL (
  DEFAULT DIRECTORY <directoryName>
```

```
  ACCESS PARAMETERS (
       records delimited BY <newline>
       fields terminated BY <";">
  )
  LOCATION ('<fileName>')
);
```

V MSSQL neexistuje pojem externí tabulka, ale k přístupu k externím datům se používají přímo příkazy `SELECT` nebo `INSERT`. Pro zobrazení dat z externího souboru se používá speciální typ příkazu `SELECT`.

```
-- Zobrazení dat v podobě tabulky v MSSQL z externího souboru:

SELECT * FROM OPENROWSET
  (BULK '<externalFile>' FORMATFILE = '<formatFiles>');
```

Kde `externi_soubor` je kompletní adresa k souboru a `format_files` je soubor s metadaty ve formě XML dokumentu (doporučeno), nebo textového non-XML souboru, popisující formu uložení dat v externím souboru. Metadata nelze vložit přímo do dotazu (jak to umí Oracle) a musíme k tomu použít speciálně určený soubor. Select můžeme následně vložit do pohledu a pracovat s ním obdobně jako s externí tabulkou v Oracle (pouze v režimu pro čtení).

Pokud bychom chtěli použít externí soubor jako zdroj dat pro naplnění předem připravené tabulky v databázi, je nejlepší využít příkazu `BULK INSERT`.

```
-- Obecný a konkrétní příklad importu dat z externího zdroje pro
MSSQL:

BULK INSERT <tableName> FROM '<Path\file>'
  WITH (FORMATFILE = '<formatFiles>');

BULK INSERT myTestCharData
   FROM 'C:\myTestCharData-c.Dat'
   WITH (
      DATAFILETYPE='char',
      FIELDTERMINATOR=','
   );
```

V tomto případě již nemusíme využít formátovacího souboru s metadaty, ale můžeme tato metadata vložit přímo do dotazu do kluzule WITH.

2.4.7 Cluster tables (seskupené tabulky)

Cluster tables jsou tabulky uložené do clusteru (klastru), kde jsou v jedné diskové oblasti spojené dohromady podle klíče clusteru. Klíč clusteru by měl být sloupcem nebo skupinou sloupců, podle kterého se tabulky obyčejně spojují.

Oracle zná čtyři typy clusterů:

- Výchozí seskupení je cluster tables, kde jsou tabulky spojeny podle zadaných hodnot klíče.
- Přidáním klauzule `HASHKEYS` vznikne seskupení hash clusters tables, kde jsou tabulky spojeny podle hash funkce.
- Přidáním klauzule `SINGLE TABLE HASHKEYS` dostaneme single-table hash cluster table pro jednu tabulku s rychlým přístupem k datům.
- Jako poslední se dá do výčtu zahrnout typ *sorted hash clusters tables*, který má u jednoho či více sloupců uvedenou klauzuli `SORT`.

Pro vytvoření seskupené tabulky je nejprve nutné vytvořit samotný cluster příkazem `CREATE CLUSER` s přesným počtem a typem sloupců, podle kterých se bude seskupovat. Nad clusterem je nutné následně vytvořit index, který bude identifikovat jednotlivé řádky v skupení (pokud je použit hash cluster, index se nevytváří. Indexem je samotná hash funkce, která řádky identifikuje přímo). Následně již můžeme vytvořit jednotlivé tabulky s klauzulí `CLUSTER` se jménem clusteru a názvy sloupců, podle kterých bude daná tabulka v clusteru spojena. Názvy sloupců musí mít sejný počet a stejné datové typy jako má cluster.

```
-- Vytvoření clusteru a cluster tables pro Oracle:

CREATE CLUSTER <clusterName> (
  <clusterColumn1> <type> [SORT]
  [, …]
) [<next parameters>];

CREATE INDEX <indexName> ON CLUSTER <clusterName>;

CREATE TABLE <tableName1> (
  <t1Column1> <type> [SORT],
  <t2Column2> <type>
  [, …]
) CLUSTER <clusterName> (<t1Column> [, …]);

CREATE TABLE <tableName2> (
  <Column1> <type>,
  <Column2> <type>
  [, …]
) CLUSTER <clusterName> (<t2Column> [, …]);
```

V MS SQL serveru obdobné seskupení tabulek není implementováno.

2.4.8 Partition tables (rozdělené tabulky)

Partition tables usnadňují lepší správu a výkon rozsáhlých datových tabulek díky fyzickému rozdělení tabulky na menší části (oddíly). Oddíly mohou být definované rozsahem, hash funkcí nebo seznamem. Jednotlivé oddíly rozdělené tabulky je vhodné uložit do různých tabulkových prostorů (kterým jsou přiděleny různé datové soubory), čímž se data skutečně fyzicky oddělí. Oracle umožňuje i tvorbu složených pododdílů (composite partitioning), kde každý oddíl je znova dělený na pododdíly. Následně můžou vznikat různé kombinace: seznam-seznam, seznam-rozsah, seznam-hash, rozsah-seznam, rozsah-rozsah, rozsah-hash.

Kromě vytvoření tabulkových prostorů se veškeré parametry určují při vytváření tabulky v příkazu `CREATE TABLE`.

```
-- Vytvoření partition table pro Oracle:

CREATE TABLE <partitionTableName> (
  <Column1> <type>
  [, …]
)
PARTITION BY {RANGE | LIST | HASH} (<column>) (
  PARTITION <partitionName1> VALUES LESS THAN
      <valuesClause> TABLESPACE <part1_ts>,
  PARTITION <partitionName2> VALUES LESS THAN
      <valuesClause> TABLESPACE <part2_ts>,
  PARTITION <partitionNameN> VALUES LESS THAN
      (MAXVALUE) TABLESPACE <partN_ts>;
);
```

MS SQL Server podporuje rozdělené tabulky též. Data se ukládají do různých filegroups (diskových skupin), kterým jsou přiděleny konkrétní datové soubory. Nejprve je nutné vytvořit dělící funkci, podle které budou data rozdělena. Poté následuje vytvoření dělícího schématu, které mapuje tabulku na příslušné diskové skupiny a nakonec následuje vytvoření samotné tabulky, která je rozdělena podle dané dělící funkce. Dělící funkce umožňuje dělit pouze pomocí rozsahu vložených hodnot.

```
-- Příklad vytvoření partition table pro MSSQL:

CREATE PARTITION FUNCTION <pf> (INT) AS
  RANGE RIGHT FOR VALUES (1,2,3,4);

-- Dělící schéma, všechny oddíly budou uloženy do jedné skupiny:
CREATE PARTITION SCHEME <ps> AS
  PARTITION <pf> ALL TO ([PRIMARY]);

CREATE TABLE <tableName> (
  column_a INT,
  column_b INT
)
  ON <ps> (<column_a>);
```

2.4.9 Nested tables (vnořené tabulky)

Oracle umožňuje vložit tabulku do jiné tabulky. Využívá k tomu objektový typ typu kolekce. Nejprve se vytvoří objekt, který obsahuje jednotlivé řádky vnořené tabulky. Tento typ se následně uloží jako objekt typu tabulka (kolekce s neomezeným počtem prvků). Následně se při tvorbě tabulky vloží sloupec s datovým typem „tabulkového objektu“ a za definicí tabulky připíše klauzule `NESTED TABLE`.

```
-- Příklad vytvoření vnořené tabulky pro Oracle:

CREATE OR REPLACE TYPE <nestedItem> AS OBJECT (
  id NUMBER(4),
```

```
  popis VARCHAR2(10)
);

CREATE OR REPLACE TYPE <nestedItems> AS TABLE OF <nestedItem>;

CREATE TABLE <tableName> (
  bag_id NUMBER(7) PRIMARY KEY,
  nested_items nestedItem
)
NESTED TABLE <nestedItems> STORE AS items_nt;
```

MSSQL použití vnořených tabulek neumožnuje.

2.5 Rozdíly v integritních omezení (constraints)

Integritní omezení jsou sadou pravidel, která umožnují zachovávat správnost a úplnost dat pomocí omezujících pravidel nad jedním či více sloupci tabulky. Klauzule omezení se uvádí přímo v příkazu `CREATE TABLE`, nebo dodatečně pomocí příkazu `ALTER TABLE`.

Mezi základní typy omezení patří:

- Not NULL – zabraňuje použít hodnot `NULL` u každého takto ošetřeného sloupce.
- Unique – zajišťuje unikátnost všech hodnot v daném sloupci (v celé množině sloupců).
- Check – umožňuje vložit podmínku, kterou musí daný sloupec či sloupce splnit.
- Primary key – množina sloupců určující primární klíč. Každý sloupec klíče je automaticky `NOT NULL`, celý klíč má omezení `UNIQUE` a zároveň je vytvořen nad touto množinou index.
- Foreign Key – odkazuje na primární klíč rodičovské tabulky. Veškeré hodnoty musí odpovídat primárnímu klíči rodičovské tabulky nebo mít hodnotu `NULL`.

Integritní omezení jsou podporována u obou databázových systémů podobně a v základním pojetí není mezi nimi žádný větší rozdíl.

```
-- Ukázka integritních omezení platná pro Oracle i MSSQL:

CREATE TABLE tableConstraint (
  id NUMBER(6),
  email VARCHAR2(25) NOT NULL,
  hireDate DATE DEFAULT SYSDATE CONSTRAINT hireDateNN NOT NULL,
  jobId VARCHAR2(10) CONSTRAINT empJobNN NOT NULL,
  salary NUMBER(8,2) CONSTRAINT empSalaryNN NOT NULL,
  CONSTRAINT tabPK  PRIMARY KEY (id),
  CONSTRAINT tabSalary CHECK (salary > 0),
  CONSTRAINT tabJob CHECK (jobId IN (1,2,5,12)),
  CONSTRAINT jobFK FOREIGN KEY (jobId) REFERENCES tabJob(id)
  CONSTRAINT tabEmail UNIQUE (email)
);

-- Zneplatnění omezení
ALTER TABLE tableConstraint NOCHECK CONSTRAINT tabSalary;
```

2.6 Rozdíly v indexech

Indexy jsou volitelnou strukturou asociovanou (nejčastěji) s tabulkou. Index lze zjednodušeně řečeno považovat za seznam klíčových slov uložený do vhodné datové struktury doprovázený informací o umístění příslušných dat. Urychluje tak přístup k řádkům tabulky vyhovující dané podmínce oproti prostému sekvenčnímu čtení.

Indexy se dělí na několik základních typů, které jsou níže podrobněji rozepsány. Popis se vždy nejprve zabývá typem indexu pro Oracle a následně je uvedena informace, zda stejný typ indexu je možné použít v MSSQL, případně zda existuje jeho přímá obdoba.

2.6.1 B-strom indexy (b-tree indexes, normal indexes)

B-strom indexy jsou označovány také jako základní, v Oracle dokumentaci často jako normal (normální), nebo pouze jako index bez žádného přívlastku. Index je vystavěn na datové struktuře B^+-stromu. Tento index obsahuje uspořádaná data v požadovaném pořadí spolu s odpovídajícími RowID (jedinečný identifikátor řádku v rámci databáze) – každá indexovaná hodnota ukazuje přímo na řádek tabulky s uloženými daty.

Index může být jednoznačný (unique), nebo víceznačný (nonunique). Jednoznačný index zaručuje, že neexistují dva řádky v tabulce se shodnými hodnotami v indexovaných sloupcích. Dále můžeme mít indexy složené - index je vystaven nad více sloupců tabulky. V tomto případě záleží na zvoleném pořadí sloupců.

```
-- Vytvoření indexu se provádí pomocí příkazu:

CREATE [UNIQUE] INDEX <indexName>
  ON <tableName>(column1 [,…]);
```

Takto postavený příkaz pro vytvoření indexu je nejtypičtější použití indexů a je shodný pro obě databáze.

2.6.2 Bitmapové indexy (bitmap indexes)

Bitmapový index vytvoří jedinou bitmapu pro všechny řádky a sloupce indexu. Index říká, zda je odpověď na daný dotaz pro aktuální řádek v databázi pravdivá či nepravdivá (Bit je nastaven, když odpovídající řádek obsahuje klíčovou hodnotu.). Je to výhodné pro řádky s nízkou variabilitou a dotazy na rovnost a selekci typu `AND`, `OR`, `NOT`.

Dotaz pro vytvoření je až na atribut `BITMAP` shodný s normálním indexem.

```
-- Vytvoření bitmapového indexu pro Oracle:

CREATE [UNIQUE] BITMAP INDEX <indexName>
  ON <tableName>(column1 [,…]);
```

Bitmapové indexy databáze MSSQL nepodporuje.

2.6.3 Reverzní indexy (reverse key indexes)

Reverzní indexy jsou indexy s převráceným klíčem – u indexu se otočí pořadí bytů u každého sloupce, nikoliv však pořadí jednotlivých sloupců. Hlavní výhoda takto upravených indexů je při modifikaci sloupců, kdy se modifikace rozprostřou po celém indexu a nejsou koncentrovány v malém množství listových uzlů. Při vkládání většího počtu hodnot sekvenčních dat se zlepší distribuci vstupně-výstupních operací díky rozprostření zátěže. Hlavní nevýhodou reverzního indexu je možnost provádět jen dotazy na přesnou shodu.

```
-- Příkaz pro vytvoření reverzního indexu pro Oracle:

CREATE [UNIQUE] INDEX <indexName>
  ON <tableName>(column1 [,…]) REVERSE;
```

Reverzní indexy databáze MSSQL nepodporuje.

2.6.4 Funkční indexy (function-based indexes)

Indexovanou hodnotou není přímo sloupec tabulky, ale hodnota výrazu. Jako výraz mohou být použity základní aritmetické operace, matematické, datumové a textové, PL/SQL, SQL nebo např. externí funkce napsaná v jazyce Java nebo C. Výraz nesmí obsahovat agregační funkci a musí být deterministický.

Dotaz se neliší od normálního indexu až na výběr indexovaného sloupce, kde místo jeho názvu je vložen výraz.

```
-- Příkaz pro vytvoření funkčního indexu:

CREATE INDEX <indexName>
  ON <tableName>(functions [,…]);
```

MSSQL pojem funkční index nezná, zato existuje jeho náhrada ve formě virtuálního sloupce, který může nést výraz a následně nad tímto sloupcem lze vytvořit základní index, který bude navenek splňovat stejné podmínky jako funkční index.

2.6.5 Index pro cluster (cluster index)

Cluster je skupina tabulek sdílející data pro společné sloupce. Pro každý cluster je nutné vytvoření indexu typu B-strom, ještě před samotným vložením dat. Viz *2.4.7 Cluster tables (seskupené tabulky)*.

```
-- Příkaz pro vytvoření indexu pro cluster:

CREATE INDEX <indexName> ON CLUSTER <clusterName>;
```

2.6.6 Indexově orientované tabulky (index-organized tables)

Indexově orientované tabulky jsou samy o sobě indexem. Celá tabulka je organizována v B*stromě, v listech stromu jsou místo RowID veškeré neklíčové hodnoty tabulky. Existuje

tedy pouze jedna struktura dat. Viz *2.4.3 Index-organized tables (IOT, indexově orientované tabulky)*.

2.6.7 Doménový index (domain indexes)

Oracle umožňuje indexovat komplexní datové typy jako jsou dokumenty, obrázky, videa, a jiné druhy dat. K tomu účelu právě slouží uživatelská struktura zvaná doménový index. Ta umožní vytvoření vlastního indexu nad tabulkou. K indexu je možné dále přiřadit statistiky a cenové funkce. Optimalizátor následně dokáže tyto indexy používat v optimalizacích stejně jako vestavěné indexy.

```
-- Příkaz pro vytvoření doménového indexu:

CREATE INDEX <indexName>
  ON <table>(<column>)
  INDEXTYPE IS <indexType>;
```

Pro vytvoření doménové indexu je nejprve nutné vytvořit speciální objekt s pravidly pomocí příkazu `CREATE INDEXTYPE`.

Alternativou doménového indexu u MSSQL může být fultextový index.

2.7 Rozdíly v pohledech

Pohled lze definovat jako předpis pro získání podmnožiny dat z jedné či více databázových tabulek. Pohled představuje pojmenovaný příkaz `SELECT`, který skrývá svoji logiku a navenek se tváří jako základní tabulka, která se ovšem před každým novým voláním musí znovu vytvořit podle svého předpisu (data zůstávají uložena pouze v původních tabulkách). Většinou se používá z důvodu bezpečnosti nebo pro zjednodušení složitých dotazů a následné jednoduché znovupoužitelnosti již jednou napsaného dotazu.

```
-- Obecný příkaz pro vytvoření pohledu:

CREATE [OR REPLACE] VIEW <viewName> AS <select statement>
  [WITH {READ ONLY | CHECK OPTION [ CONSTRAINT <constraint>};
```

Tvorba pohledů je na základní úrovni v Oracle i MSSQL databázích shodná. Hlavní rozdíl nastává v samotném příkazu `SELECT`, kde každá databáze používá vlastní příkazy a funkce (například pro vytvoření aktuálního času: `sysdate` proti `GetDate()`).

3 Možnosti migrace mezi databázovými systémy Oracle a MSSQL

Databázové systémy jsou aplikace pro uchovávání dat, které komunikují s uživatelskými aplikacemi, které využívají jejich služeb. Komunikace mezi programy je založena na modelu klient-server probíhající po síti. Klient odešle příkaz (v jazyce SQL) na server, server příkaz přijme a zpracuje. Výsledek je odeslán zpět klientovi spolu se všemi případnými chybovými hlášeními.

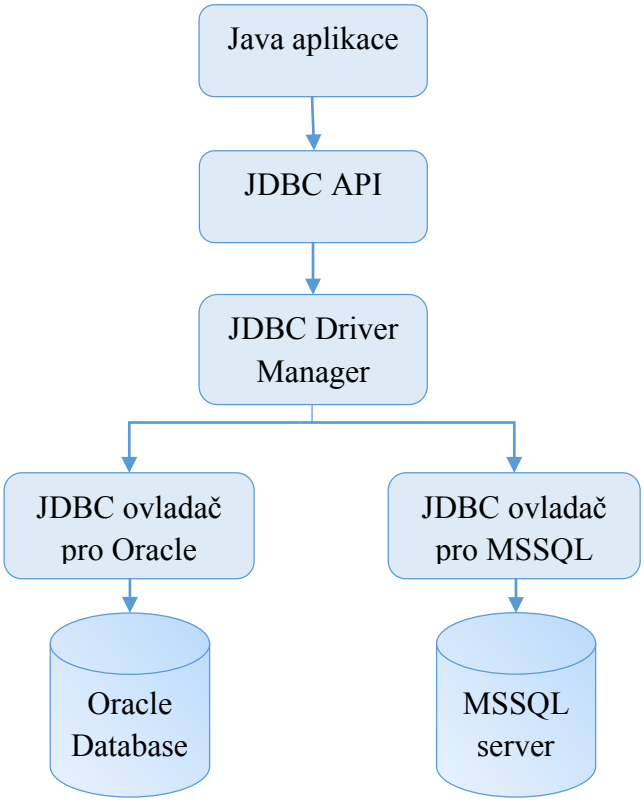
3.1 Vlastní externí aplikace

Využití externí aplikace (naprogramované ve vhodném programovacím jazyce) je nejjednodušší metodou jak získat data z jednoho databázového systému, zpracovat je a následně přeposlat do jiného databázového systému.

K jednotlivým databázovým systémům se přistupuje většinou přes jednotné API a ovladače pro příslušnou databázovou platformu, například JDBC, který funguje jako vrstva mezi aplikací a vlastní databází. JDBC ovladač pro zvolenou databázi je součástí aplikace a umožňuje snazší komunikaci s jednotlivými SQL databázemi. Nicméně je stále potřeba psát dotazy v dialektu SQL pro konkrétní databázovou platformu, viz ukázka v příloze A – *Připojení k databázím v jazyce Java.*

Po naprogramování připojení k jednotlivým databázím je možné přímo se tázat pomocí SQL jazyka systémového katalogu Oracle databáze (po zajištění příslušných práv), ze kterého lze vyčíst veškeré potřebné informace ohledně uložených objektů. Následně je možné v aplikaci informace vhodným způsobem zpracovat (případně vhodně využít interakce s uživatelem), vytvořit SQL příkazy pro vytvoření daných tabulek pro MSSQL a kompletní příkazy na MSSQL odeslat.

Velkou výhodou takto připravované aplikace je možnost využití veškerých možností zvoleného programovacího jazyka, možnost naprogramovat aplikaci přesně podle vlastních rozhodnutí a uvážení.



Obrázek 1 – Schéma připojení databází přes JDBC ovladače

3.2 Využití hotových migračních aplikací od výrobců třetích stran

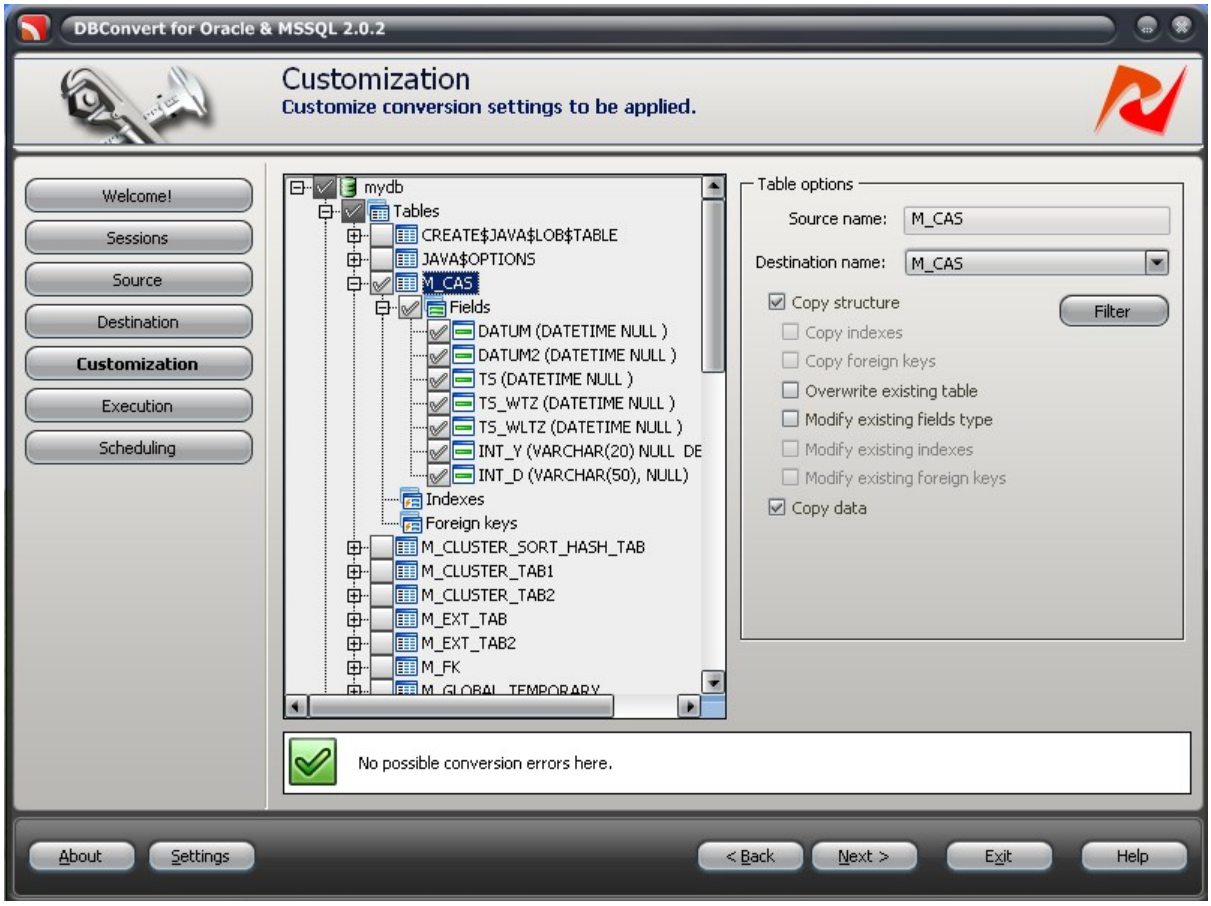
Další možností je využití aplikací od jiných výrobců, které umožňují migraci mezi oběma databázovými platformami. Na trhu jich je relativně velké množství, jako příklad se můžou uvést dvě aplikace, které byly v rámci projektu otestovány:

- *DBConvert for Oracle and MSSQL*[11] od DMSoft Technologies,
- *MS SQL Server to Oracle Converter*[12] od Interface Computers.

Jednotlivé aplikace mají podporu pro migraci různých objektů a s jednotlivými problémy migrace se vypořádávají po svém. Většina aplikací umožňuje se přihlásit pod libovolným uživatelským účtem z Oracle, vybrat jednotlivé tabulky daného schématu. U těchto tabulek lze většinou vybrat konkrétní sloupce, které budou migrovány, poupravit datový typ, případně migraci dat omezit vlastním filtrem Jako poslední je přihlášení pod uživatelským účtem MSSQL a samotná migrace tabulek i dat. Některé pokročilejší aplikace umožňují migraci default hodnot, virtuálních sloupců, některých omezení (většinou jen cizí a primární klíč), indexů a pohledů. Dokumentace jednotlivých produktů nebývá většinou příliš rozsáhlá, proto je vhodné aplikace před použitím v ostrém provozu důkladně otestovat.

[11] http://dbconvert.com/convert-oracle-to-mssql-pro.php
[12] http://www.convert-db.com/mssql-to-oracle.htm

Výhodou hotových řešení je jednoduchost použití, díky mnohaletému vývoji předpokládaná dobrá vyladěnost daného programu, rychlost a stabilita. Jako hlavní nevýhoda je nemožnost jakkoliv změnit výsledek procesu, pokud nevyhovuje navržené nastavení. Je nutné se podřídit možnostmi dané aplikace.

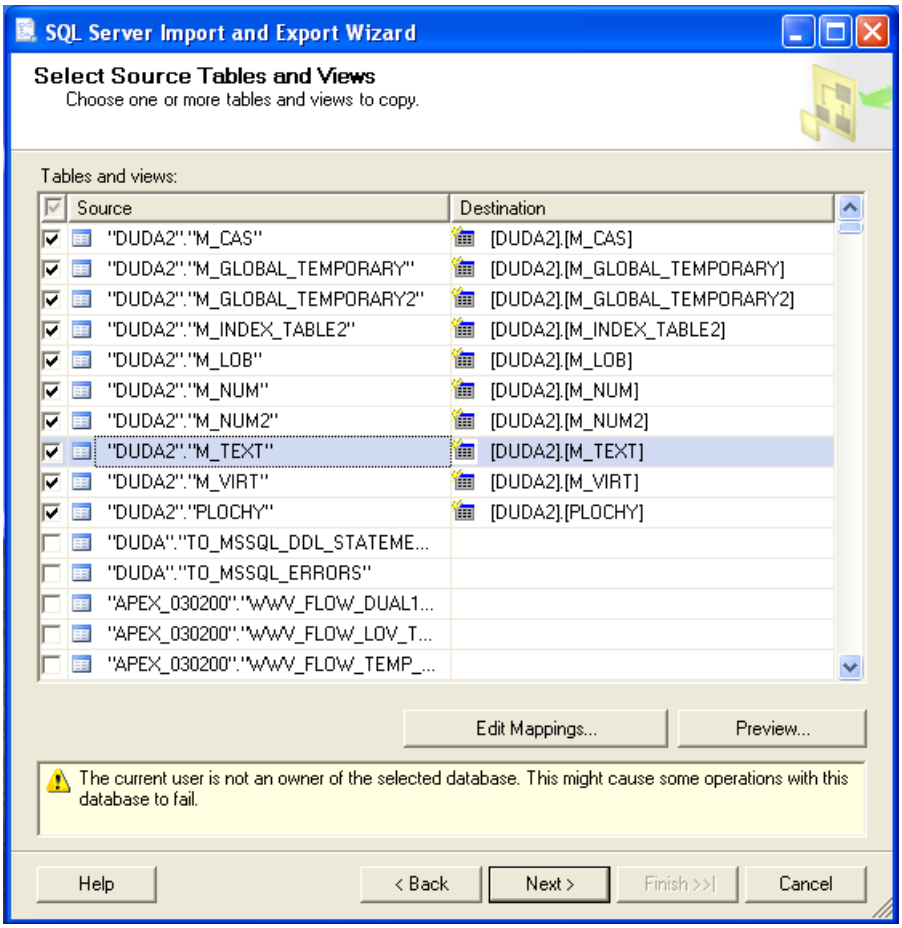


Obrázek 2 – Ukázka prostředí DBConvert for Oracle and MSSQL

3.3 MS SQL Server - Import and Export Data

Součástí MSSQL je aplikace Import and Export Data. Tato aplikace umožňuje importování dat do MSSQL z různých zdrojů, jako je jiný MSSQL, Microsoft Access, Microsoft Excel, textový soubor, nebo Oracle database.

Importování dat z Oracle do MSSQL pomocí toho nástroje je relativně intuitivní záležitost. Bohužel dokáže z Oracle přenést pouze tabulky a žádné jiné objekty. Nevýhody při použití tohoto nástroje jsou problémy s určením některých datových typů, nepodpora datových typů například LOB, interval, objektových datových typů aj., nepřenosnost virtuálních sloupců, defaultních hodnot u jednotlivých sloupců a omezení. Naopak za výhodu se dá považovat příjemné prostředí, kde se dají ještě před exportem upravit jednotlivé datové typy tabulek a mapovat je na jiné typy, případně problematický sloupec vynechat.



Obrázek 3 – Ukázka prostředí SQL Server Import and Export

3.4 Přístup z Oracle database za pomocí JDBC a Java procedury

Oracle database podporuje jazyk Java i na serverové straně a umožňuje nahrát jednotlivé Java třídy nebo celé knihovny, na které se lze následně v databázi odkazovat.

Zdrojové kódy (*.java) a přeložené třídy (*.class) jsou uloženy ve schématu uživatele v databázi (jsou viditelné v katalogu v pohledu `USER_OBJECTS`). Jejich spouštění má na starosti JServer, který běží jako součást Oracle database. K nahrávání tříd slouží utilita `loadJava`.

```
loadjava -u user/password@server:1521:DB_name soubor.java
```

Nahranou třídu je nutné nejprve zaregistrovat a následně je možné ji používat z libovolného prostředí pomocí příkazu `call`.

```
-- Registrace Java funkce:

CREATE FUNCTION javaFunction (text VARCHAR2) RETURN VARCHAR2 AS
  LANGUAGE JAVA
  NAME 'javaClass.function(java.lang.String)
  RETURN java.lang.String';
```

```
-- Volání Java funkce:

VARIABLE vystup VARCHAR2;
CALL javaFunction ('Vstupní parametr') INTO :vystup;
PRINT f;
```

Tímto způsobem se dají jednoduše volat funkce napsané v jazyce Java přímo z prostředí Oracle database.

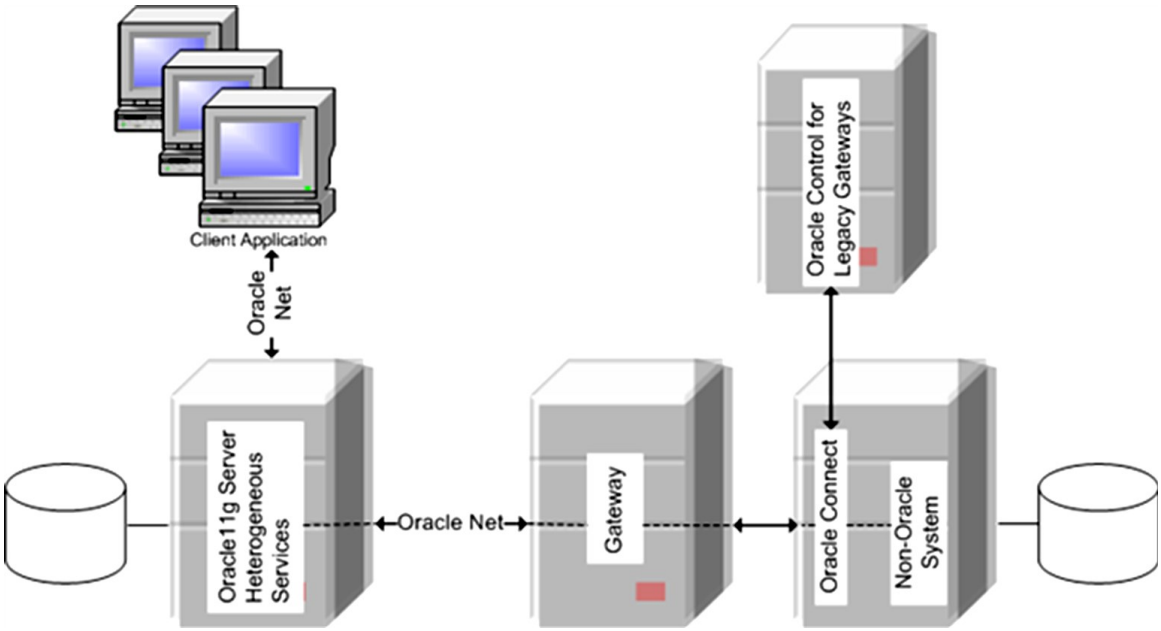
Vzhledem k informaci, že JDBC ovladač je napsaný plně v Javě, měla by existovat možnost naimportovat JDBC ovladač umožňující propojení s MSSQL do Oracle a následně pomocí jednoduchých Java funkcí komunikovat přímo s MSSQL.

3.5 Heterogenní databáze - Oracle Database Gateways

Různorodý přístup k datům je problém, který ovlivňuje hodně společností. Mnoho z nich běží na několika různých databázových systémech. Každý ze systémů vlastní svoje data a bylo by vhodné, kdyby se takto rozmístěná data mohla sjednotit skrz jedinou aplikaci, která by umožňovala komplexní pohled na data bez ohledu na databázový nebo operační systém. Oracle Database Gateways právě umožňuje sloučit tyto non-Oracle systémy s Oracle prostředím.

Oracle Database Gateways poskytují možnost transparentně přistupovat k datům umístěných v non-Oracle systému přímo z prostředí Oracle. Tato transparentnost eliminuje potřebu pro vývojáře aplikací přizpůsobit své aplikace pro přístup k datům z různých non-Oracle systémů, čímž se snižuje úsilí o rozvoj a zvyšování mobility žádosti. Oracle řešení se skládá ze dvou částí: první je obecná technologie pro připojení k non-Oracle systému, která je společná pro všechny non-Oracle systémů, nazývaná Heterogeneous Services (HS). Druhá část je specifická pro každý podporovaný non-Oracle systém, který brána připojuje. Nazývá se Agent a jako hlavní část obsahuje ODBC ovladač, přes který s ostatními systémy komunikuje [10].

Pomocí Oracle database Gateway se může Oracle připojit k databázím: DB2, Sybase, MSSQL, Teradata, Informix, IMS, VSAM, Adabas, a dalším databázím podporující obecné ODBC rozhraní jako jsou například MySQL, Foxpro, Access, dBase, Excel [10].



Obrázek 4 – Propojení Oracle Database Gateway s non-Oracle systémem

(Zdroj: http://docs.oracle.com/cd/B28359_01/gateways.111/b32527/gettingstarted.htm)

Oracle Database Gateway umožňuje pouze propojení mezi Oracle a non-Oracle systémy. Vlastní způsob migrace je nutné již doprogramovat přímo v Oracle za pomocí interního PL/SQL jazyka, uložených Java funkcí, případně externích aplikací, které budou komunikovat pouze s Oracle a vlastní přenos dat bude probíhat skrz Oracle Database Gateway.

Aplikace pro migraci postavená na Oracle Database Gateway má hlavní výhodu právě v onom přímém propojení databází. Je tedy možné využívat tohoto propojení a pomocí jednoho dotazu získat data ze dvou (případně i více) nezávislých databázových systémů a získání výsledků v jedné tabulce.

4 Analýza navrhovaného řešení

Záměr práce je najít vhodné propojení Oracle database s MSSQL Serverem, umožnit z Oracle database spouštění základních dotazů na MSSQL Serveru (DDL a DML operace, vzdálené spouštění uložených procedur…) a vytvoření PL/SQL balíčku, který by měl zvládnout migraci databázových objektů (tabulky, pohledy, indexy, omezení a případně celé uživatelské účty) z Oracle database do MSSQL pomocí navrhnutého propojení.

4.1 Způsob propojení systémů

Z důvodu pokusu přímého propojení mezi databázemi a hledání možností jak spouštět MSSQL příkazy přímo z Oracle byly v této práci vyzkoušené dvě cesty propojení: pomocí JDBC ovladače ovládaného přes uložené Java procedury a Oracle Database Gateway.

Uložené Java procedury

Jako nejvhodnější možnost pro komunikaci byla zvolena volba uložené Java procedury. Jejich hlavní výhodou měla být možnost jednoduchého importu procedur a ovladače do databáze bez nutnosti instalovat jakýkoliv jiný program a provést tak celé nastavení bez nutnosti hlubšího zásahu do běžící databáze.

Veškeré pokusy implementace této metody, která by sice teoreticky měla být možná, skončily neúspěchem. Ačkoliv vložení JDBC knihovny a komunikačních Java tříd pomocí příkazu `loadjava` bylo úspěšné, zprovoznit vzájemnou komunikaci se již nepodařilo.

```
-- Importování JDBC knihovny do Oracle:

loadjava -user <user>/<password>@<server>:<port>:<dbName>
  -v <C:/sqljdbc4.jar>
```

Nepodařilo se ani nalézt dostatek ověřených informací o úspěšném zprovoznění této metody od jiných autorů (až na projekt *orajdbclink*[13]).

Oracle Database Gateway

Druhou variantou je propojení obou databází pomocí *Oracle Database Gateway*. Základní práce s nainstalovanou bránou je popsána v následující kapitole *4.2 – Spouštění základních dotazů na serveru MSSQL*. Postup instalace a konfigurace je podrobně popsán v kapitole *5.1.1 – Instalace Oracle Database Gateway pro SQL Server*.

4.2 Spouštění základních dotazů na serveru MSSQL z Oracle

Pomocí Oracle Database Gateway je umožněn přímý přístup do MSSQL databáze z Oracle prostředí přímo pomocí základních SQL dotazů nebo pomocných funkcí.

Pokud je potřeba získat data přímo z tabulek MSSQL pomocí příkazu `SELECT` (aniž by bylo v dotazu použito funkcí na straně MSSQL), vypadá dotaz stejně jako na Oracle, jen název

[13] Projekt *orajdbclink* - http://orajdbclink.sourceforge.net/ by měl umožnit připojení MSSQL a jiných databází přímo z Oracle pomocí JDBC ovladačů. Bohužel v době psaní toho textu se nepodařilo pomocí tohoto projektu propojit testovací Oracle databázi s MSSQL a potvrdit funkčnost takto postaveného modelu.

tabulky bude ve tvaru `schema.tabulka@database_link`, kde `database_link` je odkaz na MSSQL přes Oracle Database Gateway. Stejným postupem lze řádky do tabulky přidávat, upravovat a odstraňovat. Není však možné podobným způsobem posílat DDL dotazy (`CREATE TABLE`, …), nebo využívat MSSQL funkcí. Pokud je v příkazu použita libovolná funkce, je vykonána ještě v prostředí Oracle a až poté je příkaz přenesen na MSSQL a vykonán, proto můžeme využívat vnitřních Oracle funkcí jako je `sysdate`, `to_date()`, `to_char()`, aj., případně vlastní uložené funkce.

Při používání názvů je nutné si uvědomit, že Oracle na rozdíl od MSSQL převádí automaticky veškeré názvy objektů na velká písmena. Pokud tedy chceme získat data z tabulky `MssqlSchem.MssqlTab`, je nutné název umístit do uvozovek `"MssqlSchem"."MssqlTab"`, jinak se bude hledat název tabulky `MSSQLSCHEM.MSSQLTAB`, který neexistuje a tabulka nebude nalezena.

Příklady přístupu do MSSQL přímo z Oracle prostředí pomocí Oracle Database Gateway (za předpokladu, že databázový odkaz `link_MSSQL` odkazuje do MSSQL přes Oracle Database Gateway).

```
-- Výběr dat ze vzdálené MSSQL databáze v Oracle:
SELECT column1, column2 FROM schema.table@link_MSSQL;

-- Výběr dat za použití Oracle funkce:
SELECT * FROM schema.table@link_MSSQL
  WHERE "kdy" = TO_DATE('1.1.2012 12:30', 'DD.MM.YYYY HH24:MI');

-- Výběr dat z MSSQL a Oracle databází pomocí jednoho dotazu:
SELECT *
  FROM ms_schema.ms_table@link_mssql mst
  LEFT JOIN oracle_table ort ON ort.id = ms.id
  WHERE ms.month = interval_to_num(INTERVAL '2' MONTH);

-- Vložení dat do MSSQL:
INSERT INTO table@link_MSSQL (id, date)
  VALUES (2, sysdate);

-- Aktualizace řádků na MSSQL:
UPDATE table@link_MSSQL SET "col" = moje_oracle_fce(2)
  WHERE "id" = 1;

-- Vymazání dat v MSSQL:
DELETE FROM table@link_MSSQL
  WHERE date = TRUNC(sysdate+1, 'day');

-- Volání vzdálené MSSQL procedury s parametrem:
EXECUTE schem.myFunction@link_MSSQL(-1);
```

Další možností přístupu do MSSQL z Oracle je využití balíčku `DBMS_HS_PASSTHROUGH` a příkazu `EXECUTE_IMMEDIATE`. Tento příkaz umožňuje okamžitě odeslat libovolný SQL dotaz (kromě `SELECT`) do MSSQL aniž by byl interpretován na serveru Oracle. Díky tomuto příkazu je možné na vzdáleném serveru vykonávat DDL operace, nebo využívat MSSQL

funkcí v příkazech. Použití tohoto příkazu neumožňuje vrácení hodnot. Příkaz vrací hodnotu značící počet ovlivněných řádků daným dotazem.

```
--Příklady využití příkazu
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE:

DECLARE
  num INTEGER;
  dotaz VARCHAR2(1000);
BEGIN
  dotaz := 'UPDATE "tab" SET "cloumn_A" = LEN("column_B")';
  num :=
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@link_MSSQL(dotaz);

  dotaz := 'CREATE TABLE tab2 (datum INT default GetDate())';
  num :=
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@link_MSSQL(dotaz);

  dotaz := 'EXEC schem.myFunction @param = 2013';
  num :=
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@link_MSSQL(dotaz);
END;
```

Pro vytváření dynamických příkazů s použitím vazebních (bind) proměnných (při kterých nebude prováděný `SELECT` nebo dotaz, který vrací hodnotu), lze využít funkce `EXECUTE_NON_QUERY`. Tato funkce vyžaduje ruční otevření kurzoru a analýzu dotazu, vložení vazebných proměnných, provedení příkazu a zavření kurzoru (předchozí příklad obsahuje ve skutečnosti stejné kroky, ale jsou zabalené do jednoho příkazu a volány automaticky), viz následující příklad:

```
DECLARE
  c INTEGER;
  nr INTEGER;
  dotaz VARCHAR2(1000);
BEGIN
  dotaz = 'UPDATE tab SET a=5 WHERE b=?';
  c := DBMS_HS_PASSTHROUGH.OPEN_CURSOR@link_MSSQL;
  DBMS_HS_PASSTHROUGH.PARSE@link_MSSQL(c, dotaz);
  DBMS_HS_PASSTHROUGH.BIND_VARIABLE@link_MSSQL(c, 1, 'tab');
  nr := DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY@link_MSSQL(c);
  DBMS_OUTPUT.PUT_LINE(nr||' rows updated');
  DBMS_HS_PASSTHROUGH.CLOSE_CURSOR@link_MSSQL(c);
END;
```

Dalším případem může být spuštění dynamického dotazu s možností získání výsledku (pomocí příkazu `SELECT`):

```
DECLARE
  val VARCHAR2(100);
  c INTEGER;
  nr INTEGER;
BEGIN
```

```
  c := DBMS_HS_PASSTHROUGH.OPEN_CURSOR@dg4;
  DBMS_HS_PASSTHROUGH.PARSE@dg4(c, 'SELECT col1 FROM tab');
  LOOP
    nr := DBMS_HS_PASSTHROUGH.FETCH_ROW@dg4(c);
    EXIT WHEN nr = 0;
    DBMS_HS_PASSTHROUGH.GET_VALUE@dg4(c, 1, val);
    DBMS_OUTPUT.PUT_LINE(val);
  END LOOP;
  DBMS_HS_PASSTHROUGH.CLOSE_CURSOR@dg4(c);
END;
```

4.3 Způsob přenosu databázových objektů a dat

Migrace databázových objektů a tabulkových dat bude pomocí navrhnutého způsobu postupně probíhat ve 4 fázích.

1. Tvorba jednotlivých SQL příkazů (tabulek, omezení, indexů a pohledů) pro MSSQL a jejich uložení do připravené pomocné tabulky. Pokud během převodu dojde k chybové události, k neexistenci dané volby na straně MSSQL serveru, nemožnosti zachovat všechny možnosti objektu, či jiné problémy během převodu, dojde k zapsání dané události do chybové pomocné tabulky. Z ní je následně možné vyčíst podrobné informace problému.
2. Kontrola SQL příkazů a manuální vyřešení případných chyb vzniklých během tvorby příkazů.
3. Přenesení a vykonání SQL příkazů na MSSQL pomocí připraveného propojení.
4. Přenesení samotných dat z tabulek.

4.4 Přenos tabulek

Jak už bylo zmíněno, základní tabulky se vyskytují v obou databázových systémech a jejich základní podoba se vytváří stejně a to pomocí příkazu `CREATE TABLE`. Problémem je transformování jednotlivých datových typů sloupců co možná do nejvhodnější podoby a přenos ostatních druhů tabulek do MSSQL. Veškeré přenesené tabulky musí mít zachovalou původní funkčnost, veškerá jména tabulek a sloupců musí zůstat zachována. Tabulka se nesmí rozpadnout na více tabulek nebo jí chybět jakýkoliv sloupec. Pokud by tabulka měla být přenesena jen z části, nebo není zaručena její plná funkčnost, je lepší tabulku automaticky nepřenést a nechat rozhodnutí o přenesení na uživateli.

Pomocí aplikace bude možné migrovat tabulky jen s některými možnostmi. Bohužel není zvládnutelné obsáhnout univerzální řešení pro migraci na MSSQL veškerých možností konfigurací Oracle databáze.

Tabulky, ke kterým má daný uživatel přístup (můžou být migrovány), najdeme v datovém slovníku v pohledu `all_object`, kde ve sloupci `object_type` je uvedena hodnota „`TABLE`“. Tabulky XMLType, object a nested table v tomto pohledu nenajdeme, tyto tabulky se dají vyčíst z pohledů `all_xml_tables` a `all_object_tables`.

Přenos samotných tabulek z jednoho profilu bude probíhat v několika fázích. Nejprve je nutné zjistit názvy tabulek, které budou k migraci použité. Následně bude každá tabulka podrobena analýze druhu tabulky. Jednotlivé druhy tabulek mají své specifikace a možnosti (viz kapitola *4.4.3 - Druhy tabulek v Oracle a jejich přenos*). Následně je potřeba zjistit názvy jednotlivých sloupců a jejich datové typy. Jednotlivé typy se převedou na typy MSSQL podle kapitoly *4.4.1- Přenos datových typů z Oracle do MSSQL*. Následně se zjistí, jestli není zakázána hodnota `NULL` u daného sloupce (ve výchozím stavu jsou v obou databázích povoleny), zda nemá sloupec nastavené defaultní hodnoty, případně jestli není virtuální (virtuálními sloupci se zabývá kapitola *4.4.2 - Virtuální sloupec (computed column)*). Pro převedení defaultních hodnot, stejně jako u virtuálních sloupců, bude nutné vytvořit transformační funkci, ve které se převedou základní funkce v Oracle na funkce, kterým rozumí MSSQL. Po analýze veškerých sloupců, převedení typů a zjištění, že tabulku je možné kompletně migrovat, se vytvoří celý příkaz `CREATE TABLE`, který bude uložen do předem vytvořené pomocné tabulky pro kontrolu a případné další ruční úpravy a připraven k migraci.

Pokud dojde během tohoto procesu k zjištění, že není možné vytvořit příkaz pro celou tabulku (najdou-li se v tabulce nepřenositelné datové typy či přenos daného typu tabulky není možný), dojde k vypsání hlášení o stavu a jeho uložení do pomocné tabulky. V takovém případě nebude tabulka připuštěna k automatické migraci. Pro případné zahrnutí odmítnuté tabulky k migraci jsou nutné dodatečné úpravy v pomocné tabulce.

4.4.1 Přenos datových typů z Oracle do MSSQL

Nejprve je nutné zanalyzovat datové typy, které nabízejí jednotlivé databázové systémy. Následně je potřeba navrhnout vhodné alternativy datových typů MSSQL, které jsou co nejblíže k datovým typům Oracle, aby nedocházelo pokud možno k žádným omezením. Bohužel ne pokaždé se tomu dá zabránit. Některé typy zůstanou nepřenositelné, jiné se musí zaokrouhlit nebo transformovat na jiný datový typ.

Většina databází umožňuje používat základní standardizované datové typy jazyka SQL, které by se měly jednoduše převést. Ovšem databáze používají i vlastní nestandardizované typy a v konečném vnitřním uspořádání často převádí standardní datové typy na své vlastní.

Datové typy

Vnitřní datové typy, které používá Oracle pro ukládání dat, se při převodu budou řídit tabulkou *Tabulka 3 – Návrh záměny datových typů*. Pokud je nutné datový typ upravit, zkrátit přesnost nebo délku, je dané omezení vždy uvedeno v poznámce.

Tabulka 3 – Návrh záměny datových typů

Oracle	MSSQL	Poznámka
CHAR(n)	CHAR(n)	
VARCHAR2(n)	VARCHAR(n)	
NCHAR(n)	NCHAR(n)	
NVARCHAR2(n)	NVARCHAR(n)	

Oracle	MSSQL	Poznámka
NUMBER(p, s)	NUMERIC(p, s)	0<= *s* <= *p*, *p* max. 38
FLOAT(p)	FLOAT(p)	
BINARY_FLOAT	FLOAT(24)	
BINARY_BOUBLE	FLOAT(53)	
DATE	SMALLDATETIME	
TIMESTAMP (n)	DATETIME2(n)	*n* max. 7 desetinných míst
TIMESTAMP(n) WITH LOCAL TIME ZONE	DATETIME2 (n)	*n* max. 7 desetinných míst
TIMESTAMP(n) WITH TIME ZONE	DATETIMEOFFSET (n)	n max. 7 desetinných míst
INTERVAL YEAR (p) TO MONTH	NUMERIC(14)	Data uložit jako číslo ve tvaru: ±YYYYYYYYYMMDD
INTERVAL DAY(p) TO SECOND (p)	NUMERIC(26)	Data uložit jako číslo ve tvaru: ±DDDDDDDDDHHMISSFFFFFFFFF
RAW	VARBINARY(MAX)	
LONG	VARCHAR(MAX)	
BLOB	VARBINARY(MAX)	Max. 2GB, větší jako FILESTREAM
CLOB	VARCHAR(MAX)	Max. 2GB, větší jako FILESTREAM
NCLOB	NVARCHAR(MAX)	Max. 2GB, větší jako FILESTREAM
BFILE	VARCHAR(530)	Uložit pouze cestu a název souboru jako textový řetězec. (Odpovídající datový typ BFILE je FILESTREAM)
XMLType	XML	Max. 2GB, větší jako FILESTREAM
ROWID, UROWID	-	Sloupec s tímto datovým typem bude vynechán.
Ostatní datové typy	-	Pokud bude tabulka obsahovat jiný datový typ, tabulka nebude přenesena.

Datový typ `INTERVAL` nemá svůj ekvivalent v MSSQL databázi a ani vhodný datový typ, kterým by šel nahradit. Je tedy nutné jej převést na jiný datový typ, který zajistí aspoň základní funkčnost a práci s tímto typem. Jako nejvhodnější řešení bylo nakonec zvoleno uložit interval ve formě číselného řetězce v přesném tvaru (viz *Tabulka 3 – Návrh záměny datových typů*), kde jednotlivé řády plní úlohu oddělovače. Díky tomu půjdou data bez problému číst a porovnávat podle velikosti. Operace jako je přičtení, odečtení datového úseku a další operace, které budou potřeba, lze vyřešit pomocí dodatečného naprogramování potřebných funkcí. Druhé možné řešení je převedení intervalu na počet sekund (případně dní) daného intervalu. Toto druhé řešení však naráží na problém nejednotné délky kalendářních měsíců, početní úlohy by stejně potřebovaly další úpravy. Proto bylo nakonec zvolené řešení popsané výše s předpokladem, že stejně bude muset dojít k následné úpravě tohoto datového typu.

Datové typy `*LOB` a `XMLType` větší než 2GB a typ `BFILE` se v MSSQL nedají uložit přímo do databáze, ale k tomuto účelu slouží datový typ `FILESTREAM`, který ukládá rozsáhlé dokumenty přímo do souborového systému a v MSSQL jsou uloženy jen příslušné atributy. Univerzální automatický přenos takových sloupců není možný, vlastní přenos je nutné učinit ručně. Datový typ `BFILE` je automaticky převeden jen jako název a cesta k danému souboru.

Datové typy `ROWID`, `UROWID` též nemají přímou náhradu. Datový typ `Uniqueidentifier` sice jednoznačně identifikuje libovolný řádek libovolné tabulky v databázi, ovšem už neplní roli adresy daného řádku a tedy nejrychlejšího možného přístupu. Navíc je nutné tento identifikátor nejprve uměle vygenerovat. Sloupec se nebude přenášet.

Datové typy pro jednotlivé sloupce všech tabulek lze najít v datovém slovníku v pohledu `all_tab_cols`.

Vlastní datové typy

MS SQL Server sice má funkci na vytváření vlastních datových typů `CRETE TYPE` jako Oracle, ale možnosti tohoto příkazu jsou výrazně slabší. Dá se použít jako alias pro známé datové typy nebo může nést hodnoty odpovídající řádkům tabulky, ale typ nelze použít jako datový typ sloupce při vytváření nové tabulky.

Veškeré tabulky, které používají objektové datové typy, je nutné před zahájením migrace rozložit na základní datové typy. To je jediná možnost jak tabulky s objektovými typy přenést. Automatický přenos tabulky s vlastním datovým typem není umožněn.

4.4.2 Virtuální sloupec (computed column)

Automatický přenos virtuálního sloupce je složitá operace a výsledek nemůže být plně zaručen z důvodu velké variability výrazu, který může sloupec obsahovat. Pokud bude obsahovat pouze aritmetické operace s hodnotami jiných sloupců téže tabulky, zvětšení, zkrácení či spojení textového řetězce, je možné sloupec bezpečně migrovat. Obecné vyřešení problému formátování časových a datumových funkcí a využívání vlastních funkcí uložených v databázi již nelze obecně vyřešit. Z toho vyplývají tři možnosti postupu.

- Pokusit se přenést virtuální sloupec tak jak je napsaný, případně výsledný text ještě upravit funkcí. Jedná se o změnu často se vyskytujících funkcí, které mají stejné parametry a chování, jen jiné názvy (v případě vytvoření pokročilého kompilátoru je možné přenést většinu vnitřních funkcí). Pokud se přenesení nepovede, zamezit přenesení celé tabulky.
- Přenesení tabulky bez virtuálního sloupce, který by se následně po kompletní migraci mohl přidat ručně (pomocí příkazu `ALTER TABLE`) po zvážení všech možností databáze MSSQL. Tím by se ovšem změnila struktura tabulky a znemožňovalo by to například následnou migraci pohledů nebo indexů, které jsou na dané tabulce závislé.
- Poslední možností je přenesení pouze názvu sloupce bez vnitřní funkce, čímž by byla tabulka na první pohled kompletní. MSSQL neumí modifikovat virtuální sloupce. Bylo by nutné pro editaci vytvořený nevirtuální sloupec nejprve smazat a následně vytvořit nový.

V aplikaci bude použito řešení první, tedy zkusit přenést celou tabulku a věřit, že budou použity jen operace, které jsou snadno přenositelné. V případě nalezení známých funkcí, které se dají jednoduše transformovat do syntaxe MSSQL, dojde k přepisu dané funkce. Pokročilé úpravy bude samozřejmě možné provést ještě před odesláním na MSSQL server a tím zvýšit pravděpodobnost takového přenosu.

4.4.3 Druhy tabulek v Oracle a jejich přenos

V této kapitole bude upřesněn způsob přenosu všech typů tabulek, které Oracle podporuje, konkrétně se jedná o typy:

- základní relační tabulky,
- temporary tables (dočasné tabulky),
- index-organized tables (indexově orientované tabulky),
- object tables (objektové tabulky),
- XMLType tables (tabulky datového typu XMLType),
- external tables (externí tabulky),
- cluster tables (seskupené tabulky),
- partition tables (rozdělené tabulky),
- nested tables (vnořené tabulky).

Základní relační tabulky

Základní relační tabulky jsou naprostým základem všech relačních databází. Vyskytují se v obou databázových systémech a jejich parametry jsou takřka totožné. Přenesení základních tabulek je bezproblémové.

Identifikace základních tabulek je prostá. Jsou to všechny tabulky z pohledu `all_tables`, které nepatří do žádné z ostatních typů tabulek.

Temporary tables

V Oracle je dočasné jen uložení samotných dat, nikoliv definice tabulky, která zůstává trvale uložena a i po ukončení platnosti dat je stále připravena k dalšímu použití. V MS SQL serveru jsou temporary table vymazány s koncem platnosti jak data, tak i samotná definice a doba platnosti temporary table je buď do konce procedury (lokální) nebo relace (globální). Z důvodu rozdílů samotného principu tabulek nemá smysl převádět do MSSQL tabulku jako dočasnou. Jediným vhodným řešením je převedení tabulky na základní tabulku a automatické vymazání dat doprogramovat například pomocí triggeru při vhodné události nebo data mazat manuálně po skončení jejich platnosti, například na konci procedury.

Dočasná tabulka se zjistí z pohledu `all_tables`, kde ve sloupci `temporary` je příznak „`Y`“, ve sloupci `duration` se ukazuje typ dočasnosti („`SYS$TRANSACTION`“ nebo „`SYS$SESSION`“). Během přenosu této tabulky dojde vždy k upozornění na její výskyt.

Index-organized tables (IOT)

Oracle ukládá data IOT v B-stromu v pořadí primárního klíče jako index. MSSQL index-organized tables nenabízí, ale nabízí částečnou alternativu ve formě základní tabulky, nad

kterou bude vytvořen clustered index (shlukovaný index). Cluster index organizuje data v datovém souboru tak, že záznamy se stejným nebo blízkým vyhledávacím klíčem jsou uloženy ve stejném nebo blízkém bloku.

Dočasná tabulka se zjistí z pohledu `all_tables`, kde hodnota sloupce `iot_type` není `NULL` ale `IOT`. Další podrobnosti ohledně indexu prozradí pohled `all_indexes`, který bude mít u indexově orientované tabulky sloupec `index_type` hodnotu `IOT - TOP`. Jednotlivé názvy indexovaných řádků poté získáme z pohledu `all_ind_columns`.

Object tables

Oracle nabízí možnost ukládat objekty v takzvané objektové tabulce, v níž je každý řádek řádkovým objektem. MSSQL není objektově relační databází, neumožňuje používání objektů. Atributy objektů jsou složeny ze základních datových typů (případně z dalších objektů a až ty jsou složeny ze základních datových typů). Je zde možnost object tables převést na základní relační tabulku, čímž data zůstanou zachovány, ale ztratí se tím veškerá objektová výhoda a případné objektové vazby mezi tabulkami vytvářené pomocí OID se rozpadnou.

Tento typ tabulek nebude možné automaticky převést. Pokud by se tabulky převedly na základní relační tabulky, poté převedení bude umožněno.

Veškeré přístupné object tables lze zjistit z pohledu `all_object_tables` (v pohledu `all_tables` se nenacházejí), kde je zajímavý především sloupec `table_type` s jménem objektového typu, na němž je tabulka postavena. Veškeré atributy a metody objektu se dají vyčíst z pohledu `all_source`, kde sloupec `type` má hodnotu „`TYPE`“.

XMLType tables

Tabulka datového typu `XMLType` obsahuje právě jeden sloupec XML dokumentu. (volitelně určen pomocí XML schéma). V MSSQL není přímo tabulka typu XMLType, lze jí však nahradit jednosloupcovou tabulkou s typem XML. Validaci pomocí XML schéma je podporováno také.

XMLType tabulky najdeme v pohledu `all_xml_tables`. Informace o použitém XML schéma v pohledu `all_xml_schemas`.

External tables

Externí tabulky umožní přistupovat k externímu zdroji dat (kterým je textový/binární soubor uložený na disku) jako by se jednalo o tabulku uvnitř databáze.

Hlavní využití external tables je použití externího souboru jako zdroj dat pro tabulky. Proto bylo rozhodnuto funkci externí tabulky neimplementovat. Bude zobrazena pouze zpráva o existenci tabulky v databázi Oracle i s popisem parametrů. Zda se uživatel nakonec rozhodne externí tabulku importovat a případně jakým způsobem, je ponecháno na jeho rozhodnutí.

Veškeré externí tabulky v systému Oracle jsou uloženy v pohledu `all_external_tables`, přístupovou cestu k těmto tabulkám najdeme v pohledu

all_external_locations, případně cesta k adresáři v pohledu all_directories. Externí tabulky se dají také nepřímo odvodit z pohledu all_tables, kde u sloupce tablespace_name je hodnota NULL. Tím dává najevo, že tabulka není uložena v databázi, musí tedy být uložena externě.

Cluster tables

Cluster tables je vylepšení implementované pouze v Oracle. V MSSQL cluster tables ani obdobná funkcionalita neexistuje. Migrovat se můžou pouze jednotlivé tabulky bez clusteru. Veškerá funkcionalita bude zachována, jen tabulky nebudou „předspojené“ a případné spojení bude vytvářeno před každým výběrem znovu, čímž zabere více strojového času.

Cluster tables jsou uloženy mezi ostatními tabulkami v pohledu all_tables, a poznají se podle toho, že hodnota sloupce cluster_name není NULL, ale název clusteru, v kterém se daná tabulka nachází. Další pomocné pohledy jsou all_clusters se základními informacemi o clusteru, dba_clu_columns s informacemi o názvu společných sloupců a pohled all_cluster_hash_expressions zobrazuje hash funkce pro všechny hash seskupení.

Partition tables

Rozdělené tabulky usnadňují lepší správu a výkon rozsáhlých datových tabulek díky fyzickému rozdělení tabulky na menší oddíly. Oracle definuje oddíly pomocí seznamu, rozsahu, hash funkcí. Případně má podporu i pro composite partitioning (složených pododdílů), kde můžeme kombinovat jednotlivé možnosti: rozsah-hash, rozsah-seznam, seznam-hash, seznam-seznam, seznam-rozsah a rozsah-rozsah.

MSSQL umožňuje dělit tabulky pomocí dělící funkce, která dělí jen pomocí předem definovaného rozsahu.

Díky rozdílům mezi oběma databázemi a neznámému uspořádání dat na MS SQL serveru, není možné rozdělené tabulky přenést v automatickém režimu. Lze uživateli sdělit jak byla tabulka v Oracle dělena a navrhnout základní příkaz CREATE TABLE s názvy a typy sloupců. Konečnou podobu příkazu CREATE TABLE s dělící funkcí si musí uživatel doprogramovat sám. Tabulka bude přenesena jako základní. Pokud by chtěl uživatel přenést tabulku dělenou na oddíly, musí nejprve ručně v MSSQL vytvořit dělící funkci a schéma. Následně přidat za již vytvořený příkaz CREATE TABLE (ještě před samotným přenesením příkazu) klauzuli ON <partition_schema> (<columns_name>).

Rozdělené tabulky najdeme mezi tabulkami v pohledu all_tables, kde sloupec partitioned nabývá hodnoty „YES“. Další informace lze vyčíst z pohledů: all_part_tables – základní informace o dělených tabulkách jako jsou informace o typu a podtypu rozdělení (RANGE, LIST, HASH), počtu oddílů a pododdílů. All_part_key_columns – informace o sloupci, podle kterého dochází k dělení. All_tab_partitions – informace o jednotlivých oddílech. Podrobnější informace o pododdílech (pokud jsou vytvořeny) najdeme v pohledech all_tab_subpartitions a all_subpart_key_columns.

Nested tables

Nested table je základní tabulka, která obsahuje jeden sloupec s objektovým datovým typem uloženým jako tabulka, čímž vytváří dojem tabulky v tabulce. MSSQL neumožňuje použití kolekcí ani vlastních datových typů jako datový základ pro tabulky. Tento typ nelze automaticky převést do MSSQL.

Definice vnořené tabulky není mezi ostatními tabulkami. Základní podkladová tabulka je základní tabulka, která má sloupec s vlastním datovým typem typu `COLLECTION`. Podrobnosti o uložené vnořené tabulce najdeme v datovém slovníku v pohledech `all_types` a `all_coll_types`.

4.5 Přenos omezení

Nejprve je nutné zjistit názvy jednotlivé omezení, které budou přenášeny. Jednotlivé omezení lze najít v pohledu datového slovníku `all_constraints`. Pokud potřebujeme znát názvy a pořadí jednotlivých sloupců, kterých se omezení týká, najdeme je v pohledu `all_cons_columns` (kromě omezení typu check, který má název sloupec uveden přímo v podmínce ve sloupci `search_condition`). Omezení jsou rozděleny do jednotlivých typů ve sloupci `constraint_type`:

- „`C`“ – check constraint on a table – přesný text omezení lze vyčíst ze sloupce `search_condition`. Přitom je nutné odfiltrovat podmínky `NOT NULL`, které jsou uloženy také jako typ check. Omezení `NOT NULL` je již ošetřeno v příkazu `CREATE TABLE` při vytváření tabulek.
- „`P`“ – primary key – k zjištění primárního klíče je potřeba využít pohledu `all_cons_columns`, který obsahuje názvy jednotlivých sloupců v klíči a jejich pořadí, pokud známe název omezení.
- „`U`“ – unique key – opět je nutné využít pohledu `all_cons_columns`, s názvy jednotlivých sloupců, které dohromady tvoří unikátní klíč.
- „`R`“ – referential integrity – název omezení cizího klíče lze vyčíst ze sloupce `r_constraint_name`, název primárního klíče je ve sloupci `constraint_name`. Následně z pohledu `all_cons_columns` lze vyčíst jednotlivá jména a pořadí sloupců primárního a cizího klíče.
- Další typy („`V`“ - with check option, on a view, „`O`“ - with read only, on a view, „`H`“ - hash expression, „`F`“ - constraint that involves a REF column, „`S`“ - supplemental logging) nejsou využívány.

Po zjištění typu omezení dojde k vytvoření příkazu `ALTER TABLE` s příslušným omezením nad danou tabulkou. Omezení se následně uloží do pomocné tabulky. Případné úpravy lze ovlivnit editací tabulky před migrací příkazů.

Z pohledu `all_constraints` lze také vyčíst, zda je dané omezení aktivní. Pokud je neaktivní, bude po přenesení zneplatněno. MSSQL podporuje zneplatnění omezení pouze u cizích klíčů a u omezení typu check (primární a unikátní klíč se musí přímo odstranit, volba zneplatnění u těchto typů neexistuje).

4.6 Přenos indexů

Pro přenos indexů je nejdůležitější pohled `all_indexes`, ve kterém jsou uloženy základní informace ohledně jednotlivých indexů dosažitelných daným uživatelem. Z této tabulky je nutné nejprve vyfiltrovat indexy, které se budou přenášet (indexy z konkrétního schéma, případně z konkrétní tabulky). Následně je nutné odstranit z výběru indexy, které jsou automaticky generovány, jako je automatický index nad primárním a cizím klíčem, nebo index pro zabezpečení jedinečnosti dat (unique). Informace o jednotlivých sloupcích, ze kterých se indexy skládají, najdeme v pohledu `all_ind_columns`.

Indexy lze aplikovat na různé objekty (table, cluster, view, synonym, sequence). V této práci je pozornost soustředěna jen na indexy nad tabulkami. Pokud se ve schématu vyskytují indexy nad jinými objekty, bude uživateli odesláno pouze hlášení o nalezení indexu pro daný objekt, samotný index migrován nebude.

Indexy mohou být ve dvou zneplatňujících stavech:

- UNUSABLE – nepoužitelný index. Index není aktualizovaný a nelze jej využít, dokud se ručně neobnoví. Index bude přenesen a po přenesení zneplatněn.
- INVISIBLE – neviditelný index. Index je platný a aktuální, ale optimalizátor jeho možností nevyužívá. Takový index je možné použít jen za pomocí hintů. Tento typ MSSQL nezná a bude přenesen jako nepoužitelný index. Jeho případná aktivace je na uživateli.

Jednotlivé indexy následně můžeme rozdělit podle následující hodnot sloupce `all_indexes.index_type` na jednotlivé typy:

- normal,
- normal/rev,
- bitmap,
- function-based normal,
- function-based normal/rev,
- function-based bitmap,
- cluster,
- iot – top,
- domain.

Normální indexy (normal)

Přenos normálního indexu (b-strom index) by měl být bez potíží. Indexy fungují v obou databázových systémech stejně a i jejich syntaxe je shodná. Přenášet se bude jméno indexu, sloupce ve správném pořadí, informace o vzestupném nebo sestupném třídění indexu u jednotlivých sloupců a zda se jedná o unikátní index.

Reverzní indexy (normal/rev)

Reverzní indexy nelze převést z důvodu neexistence takového indexu na databázi MSSQL.

Reverzní indexy je nejlepší nahradit normálními indexy. Teoreticky by se daly nahradit ještě virtuálním sloupcem, který by obsahoval funkci `REVERSE()`, která obrací řetězce (vrací jej pozpátku) a nad tímto sloupcem vystavět normální index. Toto řešení však platí pouze na řetězcové sloupce. Navíc zde jistě bude výkonnostní rozdíl a těžko posoudit bez většího testování, zda toto řešení není spíše kontraproduktivní. Jako nejlepší řešení se jeví použití normálního indexu a informovat uživatele o této volbě.

Bitmapové indexy (bitmap)

Bitmapové indexy v databázi MSSQL taktéž neexistují. Jako jejich možná náhrada se může použít filtrovaných indexů. Ty umožňují vytvořit filtr na indexu – klauzule `WHERE`, index poté obsahuje jen řádky splňující daná kritéria.

```
-- Příklad filtrovaného indexu na MSSQL:

CREATE NONCLUSTERED INDEX <indxName> ON <table>(sl1)
  WHERE sl1 IN ('2028', '2008', '2018');
```

Míst, kde se dá bitmapový index tímto způsobem nahradit, není mnoho. Další alternativou může být nahrazení normálním indexem vystavěným na B-stromové struktuře. Neexistuje jasné doporučení jakým způsobem bitmapový index nahradit. V této práci bylo zvoleno automatické vytvoření normálního indexu a podat informaci uživateli o tomto kroku.

Funkční indexy (function-based normal, function-based normal/rev, function-based bitmap)

MSSSQL pojem funkční index opět nezná, zato existuje jeho náhrada ve formě virtuálního sloupce, který může nést výraz a následně nad tímto sloupcem lze vytvořit index, který se bude navenek tvářit jako funkční index.

Funkční index se pozná podle hodnot: „`FUNCTION-BASED NORMAL`“ pro normální funkční index, „`FUNCTION-BASED NORMAL/REV`“ pro reverzní funkční index a „`FUNCTION-BASED BITMAP`“ pro bitmapový funkční index. Podrobnosti ohledně funkčního výrazu indexu najdeme v pohledu `all_ind_expressions`. V tomto pohledu nenajdeme název sloupce, pro který je funkce definována. Nezbývá nežli věřit, že veškeré sloupce `column_name` začínající systémovým prefixem `SYS_` (tento prefix používá Oracle pro názvosloví sloupců, tabulek, indexů, aj., které nemají vlastní uživatelský název) jsou funkční sloupce a podle jména a pozice indexu najdeme příslušný výraz. Další komplikací je zjištění, že sloupec `all_ind_expressions.column_expression` je typu `LONG`, se kterým se obtížně pracuje pro nepodporu většiny funkcí. Navíc patří mezi již zastaralé datové typy.

Automatická migrace funkčního indexu s přidáním nového sloupce do podkladové tabulky, nad kterým by byl vytvořen normální index, byla zavrhnuta z důvodu nutného zásahu do struktury tabulky a celkové změny práce s tímto indexem. Navíc zde není možné zajistit úplnou přenositelnost funkce pro ohromnou variabilitu možností. Přesto bude vygenerován příkaz pro přidání vypočítaného sloupce a nad ním vygenerovaný index. Tento příkaz bude ovšem zakomentovaný s doporučením na jeho důkladnou analýzu před použitím.

Index nad clusterem (cluster)

Index je nutný pro vytvoření clusteru. Jeho přenesení bez vytvoření clusteru nemá význam, proto tento index nebude přenesen.

Indexově orientované tabulky (iot – top)

Nejedná se o samotný index, ale o celou tabulku. V tomto případě tedy bude přenesena tabulka již s tímto indexem viz *4.4.3 – Druhy tabulek v Oracle a jejich přenos*. Index v tomto bodu nebude přenesen, neboť již existuje.

Doménový index (domain)

Doménový index je specifický pro Oracle a je na MSSQL nepřenosný. Tento druh indexu nebude přenesen.

4.7 Přenos pohledů

Přenos jednoduchých pohledů je snadný, neboť příkaz je takřka shodný. Problém však tkví v samotném příkazu `SELECT` uvnitř pohledu. Rozložení samotného příkazu `SELECT` by byla velice složitá operace. V příkazu `SELECT` se můžou vyskytovat další vnořené selecty, odkazy na jiné tabulky (i v jiných schématech a případně přejmenované pomocí synonym), vlastní funkce, vnitřní Oracle funkce, formátovací funkce výstupu jednotlivých sloupců. Úspěšné vyhodnocení takového příkazu by vyžadovalo naprogramování velmi pokročilého kompilátoru. Ale ani to by nestačilo pro úspěšné převedení pohledu, jen by se daly přesněji lokalizovat místa, která mohou mít za následek selhání přenosu dotazu.

Přenesení pohledu bude zjednodušeno. Samotný pohled bude transformován pouze funkcí, která zajistí náhradu jednoduchých Oracle funkcí za MSSQL funkce, které mají jiné názvy, ale jednotné rozhraní. Tím se aspoň částečně zvýší šance na úspěšný přenos.

Dalším problémem při přenosu pohledů je v odkazech na jednotlivé tabulky pokud budou migrovány mimo základní schéma v MSSQL. Vzhledem k tomu, že samotný pohled nebude analyzovaný, nebudou nalezeny názvy jednotlivých tabulek uvnitř pohledu a nedojde u nich ke změně schéma. Zdrojové tabulky v MSSQL nebudou nalezeny a k migraci takového pohledu nedojde.

Posledním problémem je neexistence klauzule `WITH READ ONLY` v MSSQL, bez které se musíme při přenosu obejít.

Přenos pohledu z mnoha důvodů nemůže být zaručen, proto je doporoučeno u každého pohledu před samotným přenosem provést důkladnou analýzu a případně potřebné ruční úpravy.

Je nutné zvážit veškerá omezení této automatické migrace pohledů a zvážit, zda je vůbec výhodné pohledy tímto způsobem migrovat.

Jednotlivé pohledy a jejich kód se dá vyčíst z pohledu `all_views`.

4.8 Přenos dat z tabulek

Přenos dat bude následovat po přesunu všech objektů do MSSQL. V případě zrychlení přenosu je možné nejprve migrovat pouze tabulky, následně data a na závěr omezení a indexy. Touto metodou může dojít k částečné úspoře procesorového času na MSSQL vzhledem k tomu, že nebude nutné každý vložený záznam kontrolovat na omezení a přestavovat indexy.

Pokud sloupce obsahují pouze základní textové datové typy `VARCHAR2`, `CHAR`, aj., k žádným problémům nedochází. U číselného sloupce `NUMBER` může dojít k omezení přesnosti, případně velikosti, neboť Oracle má daleko větší možnosti rozsahu. Další omezení plyne z datových typu `INTERVAL`, jeho převod na řetězec je bezproblémový, ale řešení nemusí být plně vyhovující.

Zásadním problémem je nepodpora přenosu LOB objektů pomocí Oracle Database Gateway mezi Oracle a MSSQL. Mezi některými jinými platformami je tento přenos bezproblémový, ale bohužel s MSSQL není doposud přenos umožněn.

5 Dokumentace k PL/SQL balíčku „to_mssql“

Celý program pro migraci databázových objektů z Oracle do MSSQL se skládá z jednoho PL/SQL balíčku nazvaného `to_mssql` a dvou pomocných tabulek s názvy `to_mssql_DDL_statements` a `to_mssql_errors`. Balíček je možné nainstalovat na Oracle databázi ve verzi 11g (měl by být plně funkční i pro verzi 10g, pro tuto verzi však nebyl plně otestovaný). Na straně importované databáze se nachází MS SQL Server ve verzích 2008 R2 a 2012 (migrace by měla být funkční i na verze 2005 a 2008, ale pro tyto verze nebyl balíček testován). Dále vyžaduje na straně Oracle database mít nainstalovanou a správně nakonfigurovanou bránu Oracle Database Gateway (viz kapitola *5.1.1 - Instalace Oracle Database Gateway pro SQL Server*), která slouží jako komunikační kanál mezi oběma databázemi. Oracle database musí být tato brána přístupná přes databázový link s názvem `link_to_mssql`.

Během přípravné fáze dochází k zpracování jednotlivých tabulek, omezení, indexů a pohledů určených k migraci a vytváření příslušných SQL příkazů, které se ukládají spolu s kódy a dalšími pomocnými informacemi do pomocné tabulky `to_mssql_DDL_statements`, kde se dají příkazy před odesláním na MSSQL prohlédnout, analyzovat, upravit, případně odstranit.

Díky přehlednému kódování každé skupiny dotazů (podrobný seznam všech kódů je uveden v kapitole: *5.4 Seznam použitých kódů pro pomocné tabulky*), lze skupiny následně jednoduše filtrovat a postupovat přesně podle vlastního uvážení.

Jelikož databáze mají mezi sebou relativně velké rozdíly, během vytváření příkazů vzniká velké množství chybových zpráv, které je vhodné řešit ručním zásahem, neboť automaticky některé případy vyřešit nejdou, nebo řešení nemusí být optimální. Tyto zprávy jsou zachytávány do chybové tabulky `to_mssql_errors` spolu s popisem a kódem chyby (seznam kódů chyb je uveden v kapitole: *5.4 Seznam použitých kódů pro pomocné tabulky*). Veškeré chyby uložené v této tabulce je doporučeno před migrací příkazů na MSSQL pročíst a vyvodit pro ně náležité závěry.

Zdrojové kódy vytvořeného programu jsou přiloženy v souborech *to_mssql_install.sql* a *to_mssql_package.sql*. Dále je přiložen soubor *to_mssql_test_table.sql*, ve kterém se nacházejí testovací tabulky. Na těchto tabulkách je možné otestovat veškeré možnosti daného programu.

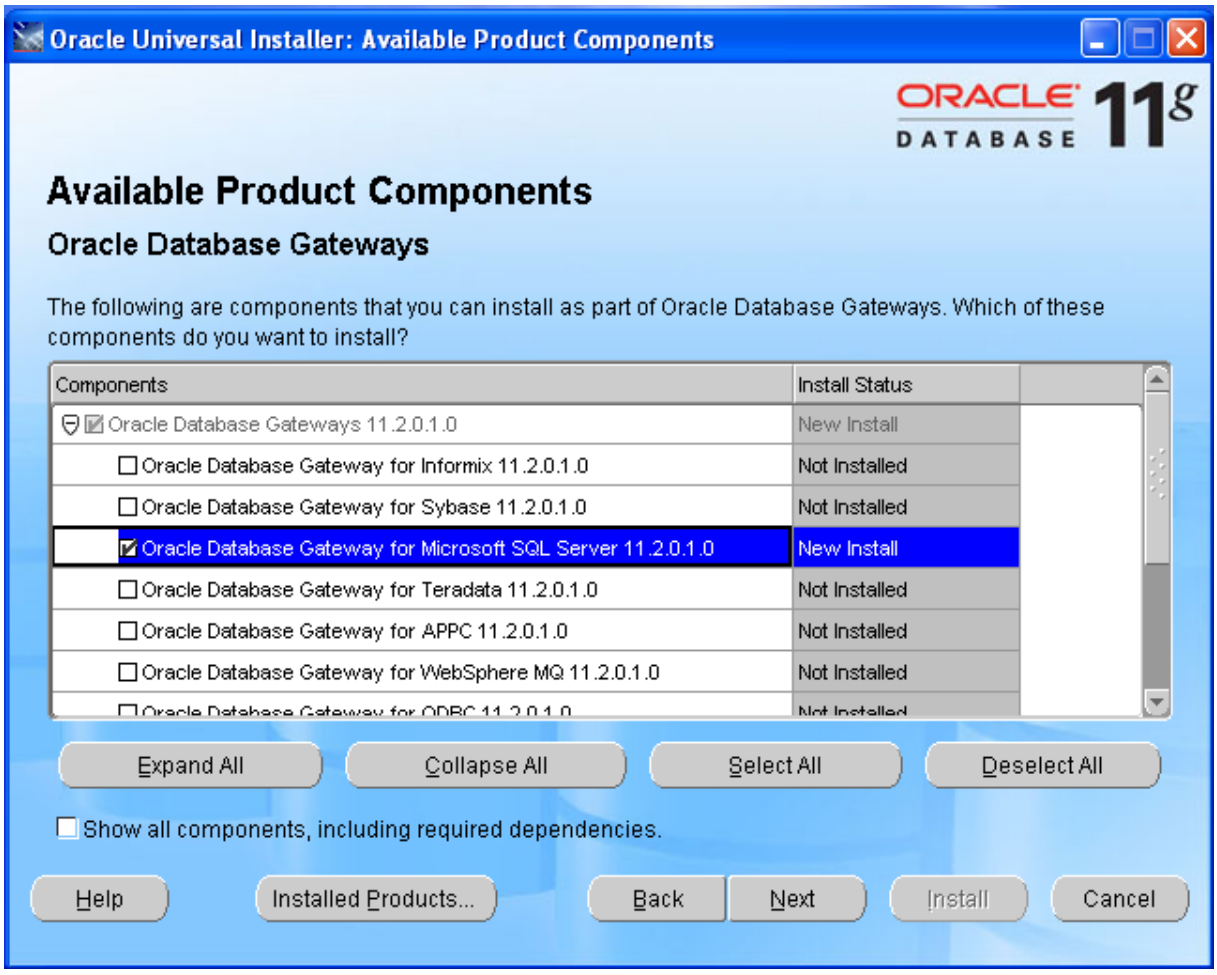
5.1 Instalace

Instalace celého řešení se stává ze dvou kroků:

- První částí instalace je instalace brány pro komunikaci mezi jednotlivými platformami Oracle Database Gateway.
- Druhou částí je instalace samotných databázových objektů pro migraci (balíčku `to_mssql` a pomocných tabulek).

5.1.1 Instalace Oracle Database Gateway pro SQL Server

Nejprve je nutné mít nainstalované oba databázové servery, poté je možné zahájit instalaci Oracle Database Gateways. Oracle Database Gateways[14] je možné stáhnout z internetových stránek společnosti Oracle pro danou verzi a operační systém, na kterém databázový server Oracle provozujeme[15]. Instalace brány se provádí pomocí přehledného grafického rozhraní Oracle Universal Installer známého z instalace Oracle. V prvním kroku je nutné při výběru komponent zaškrtnout položku *Oracle Database Gateway for Microsoft SQL Server*. V dalším kroku instalace vyžaduje vložení základních informací o cílovém MS SQL Serveru: název databázového serveru, název instance a název databáze. Následně dojde k instalaci brány.



Obrázek 5 – Grafické rozhraní Oracle Universal Installer

Konfigurace brány - inicializační parametry MSSQL

Po instalaci je třeba bránu nakonfigurovat. Inicializační soubor brány je vytvořen na adrese `ORACLE_HOME\dg4msql\admin\initdg4msql.ora`, kam se promítnou informace

[14] Oracle Database Gateway patří pod licenci RDBMS Oracle database. Není licencován zvlášť.
[15] http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html

z instalace. Pokud bude potřeba připojit se k jiné databázi nebo změnit adresu serveru či číslo portu, na kterém MS SQL Server naslouchá, je potřeba tento soubor náležitě poupravit.

V tomto souboru se také nastavuje v jakém jazyce a kódování bude Oracle s MSSQL komunikovat. Proto je nutné správně nastavit parametr `HS_LANGUAGE`. V ukázkovém příkladu je nastavení na `AMERICAN_AMERICA.EE8MSWIN1250`, neboť nejlépe vyhovovalo potřebám testovací databáze (kódování `Windows1250`, z důvodu použití testovací MSSQL databáze s kolekcí `Czech_CI_AS`).

Ukázka konfiguračního souboru `initdg4msql.ora`:

```
# This is a customized agent init file that contains
# the HS parameters that are needed for the Database Gateway
# for Microsoft SQL Server
#
# HS init parameters
#
# HS_FDS_CONNECT_INFO =[TEST_SERVER]/TEST_SERVER\MSSQLSERVER/DB
HS_FDS_CONNECT_INFO = TEST_SERVER,1433/myDB
HS_FDS_TRACE_LEVEL = OFF
HS_FDS_RECOVERY_ACCOUNT = RECOVER
HS_FDS_RECOVERY_PWD = RECOVER

HS_LANGUAGE = AMERICAN_AMERICA.EE8MSWIN1250
```

Konfigurace Oracle Net

Jako další krok je nutné nakonfigurovat pro bránu Oracle Net Listener. Databáze Oracle komunikuje s bránou pomocí rozhraní Oracle Net. Oracle Net Listener naslouchá požadavkům přicházející do databáze Oracle. Konfigurační soubor se nachází na adrese `ORACLE_HOME\network\admin\listener.ora`. Zde je potřeba přidat *SID_list* pro bránu, které bude využito v dalším kroku konfigurace a *listener*, který zajistí správné naslouchání pro příchozí požadavky z MSSQL.

Ukázka části konfiguračního souboru `listener.ora` pro využití Oracle Database Gateway:

```
SID_LIST_LISTENER =
    (SID_LIST =
       (SID_DESC =
          (SID_NAME = gateway_sid)
          (ORACLE_HOME = oracle_home_directory)
          (PROGRAM = dg4msql)
       )
    )

LISTENER=
        (ADDRESS =
          (PROTOCOL = TCP)
          (HOST = host_name)
          (PORT = port_number)
        )
```

Pro jednodušší konfiguraci je možné využít již předpřipraveného souboru `ORACLE_HOME\dg4msql\admin\listener.ora.sample`, který byl vytvořen při instalaci brány. Je však třeba dbát na pravidla, neboť jakékoliv nepřesné změny mohou vést k nefunkčnosti služby Oracle Net a tím přerušení veškeré komunikace databáze s vnějším prostředím. Je vhodné před změnou souboru *listener.ora* důkladně prostudovat dokumentaci *Oracle Database Net Services Reference*[16] a zálohovat původní soubor.

Následně je potřeba změny aktivovat restartováním služby Oracle Net Listener pomocí dvou příkazů v příkazovém řádku:

```
lsnrctl stop
lsnrctl start
```

Konfigurace databáze Oracle pro přístup k bráně

Před použitím brány pro přístup k MS SQL Serveru se musí nakonfigurovat databáze Oracle pomocí takzvaného popisovače spojení, aby komunikovala s bránou přes Oracle Net. Konfigurační soubor najdeme na adrese `ORACLE_HOME\network\admin\tnsnames.ora`, do kterého přidáme řádky:

```
connect_descriptor =
   (DESCRIPTION =
      (ADDRESS =
         (PROTOCOL = TCP)
         (HOST = host_name)
         (PORT = port_number)
      )
      (CONNECT_DATA =
         (SID = gateway_sid))
      (HS = OK))
```

Opět je možné využít již předpřipraveného souboru `ORACLE_HOME\dg4msql\admin\tnsnames.ora.sample`. Tímto úkonem je konfigurace brány dokončena.

Vytvoření databázového odkazu

Pro přístup k MSSQL je nutné v Oracle vytvořit databázový odkaz (link), pomocí kterého se jednotlivé SQL příkazy do MSSQL odkazují pomocí Oracle Database Gateway. Odkaz se vytvoří pomocí příkazu `CREATE DATABASE LINK` se jménem a heslem k uživatelskému účtu MSSQL a s klauzulí `USING`, kam vložíme jméno popisovače spojení vytvořeného v konfiguraci brány (*connect_descriptor*).

```
-- Příklad vytvoření databázového odkazu do MSSQL:

CREATE [PUBLIC] DATABASE LINK <dblink> CONNECT TO
  "<user>" IDENTIFIED BY "<password>" USING '<tns_name_entry>';
```

[16] http://docs.oracle.com/cd/E11882_01/network.112/e10835/toc.htm

Pro potřeby balíčku `to_mssql` bude používán link s názvem `link_to_MSSQL`.

Každý klient připojený k databázi Oracle (vlastnící příslušná práva pro čtení příslušného databázového linku) může získat přístup do MSSQL. Připojení k bráně se otvírá prostřednictvím prvního použití databázového linku v relaci a končí v okamžiku ukončení aktuální relace.

Po vytvoření databázového odkazu, můžete ověřit připojení k databázi MSSQL.

```
-- Otestování úspěšného propojení:

SELECT null FROM dual@link_to_MSSQL;

-- Informace o času, názvu serveru a jménu uživatele na MSSQL
přístupném pomocí linku link_to_MSSQL:

DECLARE
  v1 DATE;
  v2 VARCHAR2(200);
  v3 VARCHAR2 (200);
  c PLS_INTEGER;
BEGIN
  c := DBMS_HS_PASSTHROUGH.OPEN_CURSOR@link_to_MSSQL;
  DBMS_HS_PASSTHROUGH.PARSE@link_to_MSSQL (c,
      SELECT GETDATE(), HOST_NAME(), SUSER_NAME());
  WHILE DBMS_HS_PASSTHROUGH.FETCH_ROW@link_to_MSSQL (c) > 0
  LOOP
    DBMS_HS_PASSTHROUGH.GET_VALUE@link_to_MSSQL (c, 1, v1);
    DBMS_HS_PASSTHROUGH.GET_VALUE@link_to_MSSQL (c, 2, v2);
    DBMS_HS_PASSTHROUGH.GET_VALUE@link_to_MSSQL (c, 3, v3);
    DBMS_OUTPUT.PUT_LINE(v1||' '||v2||' '||v3);
  END LOOP;
  DBMS_HS_PASSTHROUGH.CLOSE_CURSOR@link_to_MSSQL (c);
END;
```

5.1.2 Instalace balíčku „to_mssql“

Před instalací je nejprve nutné ověřit, zda má uživatel, který bude migraci provádět práva pro vytváření tabulek, sekvencí a balíčků (`CREATE TABLE, CREATE SEQUENCE, CREATE PROCEDURE`). Dále musí mít uživatel přístupová práva ke všem migrovaným objektům (pokud chce uživatel migrovat celou databázi, nejjednodušší je přiřadit si právo `SELECT ANY TABLE`, které umožní prohlížet veškeré tabulky všech uživatelů, případně všem objektům, které je potřeba migrovat přiřadit právo `GRANT SELECT ON <table> TO <uživatel>;`).

Po instalaci Oracle Database Gateway je potřeba se přesvědčit, zda je skutečně uživateli přístupný název linku `link_to_MSSQL`, pomocí kterého dochází ke komunikaci mezi databázemi. Bez tohoto linku nebude instalace balíčku možná.

Pro instalaci stačí v Oracle (v daném schématu) spustit přiložený soubor *to_mssql_install.sql*, který vytvoří dvě tabulky (`to_mssql_DDL_statements`,

to_mssql_errors) a sekvenci (to_mssql_event_sq). Následně je třeba spustit soubor *to_mssql_install.sql*, ve kterém se nachází PL/SQL balíček (to_mssql). Tím je celá instalace dokončena a aplikace je připravena k použití.

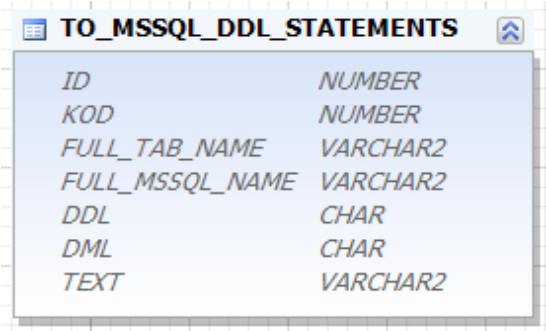
5.1.3 Příprava MSSQL k migraci

Na straně MSSQL musí existovat uživatel, který je uveden v databázovém linku link_to_MSSQL. Tento uživatel musí mít práva pro vytváření tabulek a pohledů (CREATE TABLE, CREATE VIEW) a nastavené potřebné paměťové limity. Pokud bude migrace prováděna do různých schémat na MSSQL, musí být jednotlivá schéma vytvořena před zahájením migrace.

Pokud bude chtít uživatel migrovat data do různých uživatelských účtů MSSQL, je nutné provést migraci na několik etap a mezi každou etapou změnit databázový odkaz link_to_MSSQL na vždy aktuálně potřebného uživatele.

5.2 Pomocné tabulky

TABLE to_mssql_DDL_statements



Obrázek 6 – ER diagram tabulky to_mssql_DDL_statements

Tabulka to_mssql_DDL_statements uchovává veškeré DDL dotazy, které budou migrovány na MSSQL server. Tabulka obsahuje:

- id - jednoznačné id řádku.
- kod - kód operace (jednotlivé kódy jsou popsány v kapitole *5.4 - Seznam použitých kódů pro pomocné tabulky*).
- full_table_name - plný název migrovaného objektu.
- full_mssql_name - plný název importovaného objektu (pro tabulky a pohledy) nebo název objektu, kterého se bude migrace na MSSQL týkat (pro omezení a indexy).
- ddl – informace o přenosu, kde jednotlivé kódy značí:
  - „1“ – příkaz připraven k přenosu.
  - „2“ – příkaz připraven k přenosu, ovšem může selhat z důvodu použití virtuálního sloupce, nebo pohledu – takový příkaz nejde plně automaticky odladit a je doporučeno takový příkaz před migrací zanalyzovat a případně ručně poupravit.

- „3“ – příkaz ALTER pro změnu tabulek a pohledů.
- „N“ – automaticky nepřenositelný příkaz, který je sice vytvořen, ale nemůže být převeden. Každý takový případ je nutné zanalyzovat a vyřešit ruční úpravou. Následně je možné změnit kód na odpovídající označení a tím umožnit vykonání daného příkazu.
- „0“ – přenos skončil úspěchem.
- „E“ – přenos příkazu skončil neúspěchem.
- „I“ – daný objekt již v databázi MSSQL existuje.

Kódy „1“, „2“, „3“, „N“ se týkají fáze vytváření jednotlivých příkazů a kódy „0“, „E“, „I“ informují o úspěšnosti přenesení daného příkazu.

Migrace příkazů do MSSQL bude provedena pomocí procedury migration_DDL pouze u těch příkazů, kde budou uvedeny hodnoty „1“, „2“ nebo „3“. V ostatních případech nebude migrace provedena.

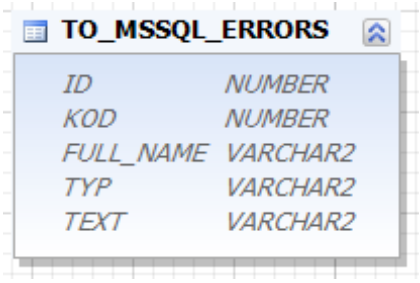
- dml – informace o přenosu dat tabulky. Tento sloupec je určen jen pro příkaz CREATE TABLE a podává informace ohledně přenosu dat v tabulkách. Jednotlivé kódy označují:
  - „1“ – tabulka je připravená pro přenos dat.
  - „N“ – přenos dat není možný (tento kód mají nastaven také všechny příkazy, které nevytvářejí tabulku).
  - „0“ – přenos dopadl úspěšně.
  - „E“ – chyba při přenosu dat.
  - „I“ – tabulka v MSSQL neexistuje, přenos dat není možné aplikovat.

  Kódy „1“ a „N“ se týkají fáze vytváření jednotlivých příkazů a kódy „0“, „E“, „I“ se týkají stavu po migraci dat.

  Migrace dat v tabulkách do MSSQL bude uskutečněna pomocí procedury pouze u těch případů, kde bude hodnota sloupce DDL rovna „0“ a hodnota sloupce DML rovna „1“.

- text – kompletní příkaz DDL operace ve tvaru, v jakém bude spuštěn na MSSQL. (Pokud by měl příkaz více nežli 4000 znaků, není možné celý příkaz vložit do tabulky. Bude uložena jen část příkazu a vygenerováno hlášení do chybové tabulky.)

Tabulka to_mssql_errors



Obrázek 7 – ER diagram tabulky to_mssql_errors

Tabulka `to_mssql_errors` zaznamenává veškeré chyby a možné problémy, které vznikají při tvorbě dotazů.

Její struktura je následující:

- `Id` – jednoznačná identifikace řádku.
- `Kód` – kód chyby (jednotlivé kódy jsou popsány v kapitole *5.4 - Seznam použitých kódů pro pomocné tabulky*).
- `Full_name` – celý název tabulky, které se problém týká.
- `Typ` – typ příkazu, kterého se chyba týká (create/alter table/constraint/index/view).
- `Text` – přesný popis chyby.

5.3 Popis balíčku

Základní rozhraní balíčku vypadá následovně:

```
CREATE PACKAGE to_mssql AS
  PROCEDURE clear_temp_table();

  PROCEDURE create_table();
  PROCEDURE add_constrains();
  PROCEDURE create_indexes();
  PROCEDURE create_wiews();

  PROCEDURE migration_setting();
  PROCEDURE migration_DDL();
  PROCEDURE migration_insert();

  FUNCTION interval_DS_to_char() RETURN NUMBER;
  FUNCTION interval_YM_to_char() RETURN NUMBER;
END to_mssql;
```

Průběh migrace z Oracle do MSSQL se skládá z několika kroků:

1. Postupné vytvoření příkazů pro tvorbu tabulek, omezení, indexů a pohledů (pomocí procedur `create_table`, `add_constrains`, `create_indexes`, `create_wiews`) Objekty můžou být migrovány do různých schémat (vícenásobným použitím jedné procedury s jinými parametry) budou ukládány pod jedním MSSQL uživatelským účtem uloženém v linku `link_to_MSSQL` (pro migraci pod více

uživatelskými účty je nutné tyto kroky pro každý účet podstoupit samostatně a na konci migrace vždy náležitě upravit daný link).

2. Kontrola vygenerovaných příkazů a analýza případných chybových hlášení a následně možná úprava jednotlivých příkazů v pomocné tabulce.
3. Volitelně je možné změnit natavení migrace pomocí procedury `migration_setting`.
4. Přenesení veškerých příkazů z tabulky `to_mssql_DDL_statements`, které mají příznak v sloupci `DDL` povolující přenesení.
5. Provedení kontroly úspěšného převedení veškerých příkazů (v pomocné tabulce bude ve sloupci `DDL` hodnota „`0`“, v případě chyby hodnota „`E`“). V případě existence neúspěšných přenosů je možné řádky opravit, vložit hodnotu „`1`“ do sloupce `DDL` a provést proceduru znova.
6. Přenesení dat uložených v tabulkách pomocí příkazu `migration_insert`. K přenesení dat dojde za předpokladu, že ve sloupci `DDL` je hodnota „`0`“ značící úspěšný přenos a ve sloupci `DML` hodnota „`1`“ značící připravenost k přenosu.
7. Kontrola úspěšného přenosu dat. V sloupci `DML` v případě úspěchu bude hodnota „`0`“. V případě existence chyby je opět možné přenesení dat opakovat (za předpokladu provedení opravy chyb a náležité změny sloupců `DDL` a `DML`.
8. Po úspěšné migraci je možné odstranění dat z pomocných tabulek použitím procedury `clear_temp_table`.

V následujících kapitolách se budou postupně rozebírat jednotlivé funkce tohoto balíčku.

5.3.1 Pomocné funkce a procedury

PROCEDURE clear_temp_table

Procedura `clear_temp_table()` slouží k vymazání dat z obou pomocných tabulek.

PROCEDURE migration_setting (
p_delete_old_data IN BOOLEAN DEFAULT false,
p_delete_old_table IN BOOLEAN DEFAULT false)

Procedura umožňuje nastavit automatické vymazání starých dat na MSSQL. Parametr `p_delete_old_data` určuje, zda budou vymazána veškerá data z tabulky před importem nových dat. Druhý parametr `p_delete_old_table` určuje, zda při nalezení tabulky stejného jména na MSSQL jako je importovaná, bude původní tabulka odstraněna a nahrazena novou, nebo zůstane původní tabulka zachována a k migraci nové tabulky nedojde.

Nastavení existuje především z důvodu jednoduššího testování balíčku.

Pomocné funkce interval_

Dále se v balíku nacházejí dvě veřejné pomocné funkce `interval_DS_to_char()` a `interval_YM_to_char()`. Funkce slouží pro vnitřní převod intervalových datových typů. Kvůli dynamickému volání těchto funkcí je nutné je mít jako veřejné.

5.3.2 Příprava tabulek

Přípravou tabulek na migraci se zabývá procedura `create_table()`.

PROCEDURE create_table (
p_schema VARCHAR2,
p_table VARCHAR2 DEFAULT NULL,
p_mssql_schema VARCHAR2 DEFAULT NULL)

Vstupní parametry jsou:

- `p_schma` – určuje schéma v Oracle databázi, ze kterého se budou tabulky vybírat. Hodnota musí být nastavena, hodnota `NULL` není přípustná.
- `p_table` – určuje konkrétní tabulku, která bude migrována. Pokud bude nastavena hodnota `NULL` (výchozí hodnota), budou migrovány veškeré tabulky, které se v zadaném profilu vyskytují.
- `p_mssql_schmema` – určuje schéma na straně MSSQL, do kterého se budou tabulky importovat. Pokud bude zadána hodnota NULL, budou tabulky odeslány do výchozího schéma uživatele uvedeného v linku `link_to_MSSQL`.

Procedura `create_table()` vytváří kompletní SQL příkaz `CREATE TABLE` pro každou nalezenou vstupní tabulku a ukládá kompletní příkazy do pomocné tabulky `to_mssql_ddl_statements`.

Procedura postupně vybírá tabulky z pohledu `all_object`. Zjistí jakého je tabulka typu a podle toho provádí převod (tabulky typu XML, objektové tabulky a případně vnořené tabulky se nacházejí v pohledech `all_xml_tables` repletivě `all_object_tables`). Způsob, jakým dochází k převodu, je důkladně zanalyzovaný v kapitole *4.4 – Přenos tabulek*. Datové typy se poté převádí podle tabulky *Tabulka 3 – Návrh záměny datových typů*.

Pokud není možné plně automaticky přenést tabulku, nebo pokud dochází při migraci k změně typu tabulky anebo se vyskytují jakékoliv jiné informace o chybném či neúplném průběhu převodu, jsou tyto informace zapisovány do pomocné tabulky `to_mssql_errors`.

Pokud tabulka nebude moci být přenesena celá, nebude připuštěna k automatické migraci a pro pokračování migrace dané tabulky bude potřeba záznam v tabulce `to_mssql_ddl_statements` editovat ručně.

5.3.3 Příprava omezení

Vytvářením SQL příkazů pro přesun jednotlivých omezení se zabývá procedura `add_constrains()`.

PROCEDURE add_constrains (
p_schema VARCHAR2,
p_table VARCHAR2 DEFAULT NULL,
p_mssql_schema VARCHAR2 DEFAULT NULL)

Vstupní parametry jsou shodné s parametry procedury `create_table`:

- `p_schma` – určuje schéma v Oracle databázi, ze kterého se budou tabulky vybírat.
- `p_table` – určuje konkrétní tabulku, ke které se mají daná omezení týkat. Pokud bude nastavena hodnota `NULL`, budou migrovány všechny omezení k tabulkám, které se v zadaném profilu vyskytují.
- `p_mssql_schmema` – schéma na straně MSSQL, do kterého se budou omezení importovat.

Procedura postupně prochází jednotlivé omezení z datového slovníku `all_constraints`, které se týkají zadaného schématu a tabulky. Procedura vynechává omezení typu `NOT NULL` a `DEFAULT` hodnot, které jsou řešeny při přesunu tabulek. V proceduře není řešena migrace omezení typu hash expression. Pro toto omezení vzniká chybové hlášení a může se případně doprogramovat ručně. Podrobněji je přenos jednotlivých omezení popsán v kapitole *4.5 – Přenos omezení*.

Pro omezení, která jsou v Oracle databázi zneplatněná, jsou vystaveny dotazy pro pozastavení jejich funkce.

5.3.4 Příprava indexů

Přípravou migrace indexů se zaobírá procedura `create_indexes()`.

PROCEDURE create_indexes (
p_schema VARCHAR2,
p_table VARCHAR2 DEFAULT NULL,
p_mssql_schema VARCHAR2 DEFAULT NULL)

Vstupní parametry jsou opět shodné s parametry procedury `create_table`:

- `P_schma` – určuje schéma v Oracle databázi, ze kterého se budou indexy vybírat.
- `P_table` – určuje konkrétní tabulku, pro kterou jsou indexy hledány. Pokud bude nastavena hodnota `NULL`, budou migrovány všechny indexy tabulek, které se v zadaném profilu vyskytují.
- `P_mssql_schmema` – schéma na straně MSSQL, do kterého se budou indexy importovat.

Procedura prochází jednotlivé indexy z datového slovníku `all_indexes`, které se týkají zadaného schématu a případně tabulky. Pro každý nalezený index se snaží vytvořit příkaz `CREATE INDEX`, který bude uložen mezi ostatními příkazy v pomocné tabulce příkazů. Vynechány jsou indexy, které se automaticky vytváří při tvorbě primárního klíče, cizího klíče a unikátních indexů. Tyto indexy vzniknou při zpracování předešlé procedury zabývající se omezeními.

Převod jednotlivých typů indexů je popsán v kapitole *4.6 – Přenos indexů*. Řešení pro funkční indexy pomocí virtuálního sloupce je naznačeno, ale pro velké množství možností funkčního výrazu není přesun ve výchozím nastavení povolen. Příkazy jsou sice vytvořeny, ale nebudou automaticky přeneseny a je doporučena jejich důkladná analýza před povolením jejich přenosu.

Veškeré chyby a upozornění při přenosu indexů nad tabulkami jsou protokolovány v pomocné tabulce chyb. Indexy použité nad jinou strukturou, nežli je tabulka, nebudou zaznamenány.

Příkazy pro zneviditelněné (`UNUSABLE`) a zneplatněné (`INVISIBLE`) indexy jsou vytvořeny a automaticky zneplatněny pomocí zvlášť vytvořeného příkazu. Příkazy k zneplatnění je možné odstranit ještě před samotným přenosem příkazů, případně následně v MSSQL příkazem `ALTER INDEX <indexName> ENABLE` zneplatněné indexy znovu aktivovat.

5.3.5 Příprava pohledů

Přípravou migrace pohledů se zabývá procedura `create_view()`.

PROCEDURE create_wiews (
p_schema VARCHAR2,
p_view VARCHAR2 DEFAULT NULL,
p_mssql_schema VARCHAR2 DEFAULT NULL)

Vstupní parametry jsou:

- `P_schma` – určuje schéma v Oracle databázi, ze kterého se budou pohledy vybírat.
- `P_view` – určuje konkrétní pohled, který by měl být přenesen. Pokud bude hodnota nastavena na `NULL`, budou migrovány všechny pohledy, které se v zadaném schématu vyskytují.
- `P_mssql_schmema` – schéma na straně MSSQL, do kterého se budou pohledy importovat.

Procedura prochází jednotlivé pohledy z datového slovníku `all_views`, které se týkají zadaného schématu a případně konkrétního pohledu. Pro každý nalezený pohled vytvoří nový příkaz `CREATE VIEW` a uloží mezi ostatní příkazy v pomocné tabulce příkazů.

Každý text nového pohledu je překládán pomocí funkce `translate_function_to_MSSQL()`. Tato funkce se snaží minimalizovat rozdíly mezi jednotlivými databázemi a nahrazuje základní funkce v Oracle database funkcí z MSSQL, které mají stejný význam, jen jiné jméno. Funkce vyhledává a překládá jen pár nejzákladnějších a nejpoužívanějších funkcí. Z důvodu velké variability pohledů, rozdílnosti jednotlivých databázových systémů a nepřipravenosti jazyka PL/SQL pro tyto úkony, není možné zachytit veškeré rozdíly.

Každý pohled ještě před jeho odesláním na MSSQL Server je nutné zkontrolovat a zajistit jeho plnou funkčnost. Úspěšná automatická migrace pohledů nemůže být zaručena.

5.3.6 Migrace připravených příkazů

Přenos příkazů z Oracle na MSSQL probíhá pomocí procedury `migration_DDL()`.

Procedura postupně prochází jednotlivé řádky pomocné tabulky `to_mssql_ddl_statements`, z které jsou vyfiltrované podle hodnoty kódu ve sloupci `ddl` s hodnotami „`1`“, „`2`“ nebo „`3`“. Řádky s jinými kódy nebudou přeneseny. Následně jsou záznamy seřazené podle hodnoty kódu, čímž se zajistí vykonání ve správném pořadí. Nejdříve budou přeneseny veškeré definice tabulek, následně omezení, indexů a nakonec pohledů.

Pokud na MSSQL již tabulka daného jména existuje, zachová se podle nastavení `migration_setting()`: buď původní tabulku odstraní a následně vytvoří novou podle definice nebo program a původní tabulku ponechá, vypíše chybu a do pole `DDL` zapíše příznak „`I`“ (výchozí nastavení).

Následně dojde k přenosu jednotlivých příkazů, v případě úspěšného přenosu příkazu, zapíše se do sloupce `DDL` příznak „`0`“, v případě chyby příznak „`E`“. Nakonec vypíše hlášení o úspěchu přenosu. Pokud dojde k chybě přenosu, ovlivní to vždy jen konkrétní příkaz a funkce pokračuje dál.

5.3.7 Migrace dat z tabulek

Migrace dat probíhá pomocí procedury `migration_insert()`.

Procedura postupně prochází jednotlivé řádky tabulky `to_mssql_ddl_statements`, vybírá pouze řádky s hodnotou kódu ve sloupci `ddl` „`0`“ – značící úspěšný přenos tabulky a hodnotou kódu ve sloupci `DML` „`1`“ – značí připravenost tabulky k přenosu dat (příkazy, které nevytvářejí tabulky nebo tabulky, které nejsou vhodné k migraci dat, mají ve sloupci `DML` hodnotu „`N`“). Z těchto řádků se zjistí názvy tabulek, u kterých dojde k migraci dat.

U každé tabulky určené k migraci je provedena kontrola na existenci tabulky na straně MSSQL. V případě neexistence tabulky dojde k zapsání příznaku „`I`“ do sloupce `DML` a přeskočení na další. Pokud je v `migration_setting()` nastaveno mazání starých dat, před migrací dojde k odstranění těchto dat. Jinak se data sloučí s původními.

Následuje samotná migrace dat. Na konci migrace každé tabulky bude vytvořeno hlášení o případném úspěchu nebo neúspěchu. V případě neúspěšného přenosu dojde k vygenerování chybového stavu, k vykonání příkazu `ROLLBACK` pro neúspěšnou tabulku a pokračuje se dalším záznamem.

Během programování balíčku bylo zjištěno omezení rozhraní Oracle Database Gateway, které nepodporuje přenášení LOB objektů mezi Oracle a MSSQL. Pokud se hodnoty převedou na řetězec `VARCHAR2`, lze tyto hodnoty přenést. Limity typu `VARCHAR2` (typ `VARCHAR2` je omezen na velikost 4000 B v SQL režimu a 32 767 B v PL/SQL) jsou pro LOB objekty nevyhovující. Je možné vytvořit řešení v PL/SQL, kdy se LOB rozloží na menší textové řetězce, které se budou postupně přenášet a v MSSQL se spojí do jednoho

objektu. Toto řešení je nutné programovat na míru konkrétním tabulkám, tedy předem znát strukturu tabulek a jejich primárního klíče. Efektivního řešení takového problému v PL/SQL nejde dosáhnout. Řešením může být naprogramování jednoduché aplikace v libovolném programovacím jazyce, která zvládne přenést LOB objekty. Jednoduchá ukázka takové aplikace naprogramovaná v jazyce Java je přiložena v příloze: *příloha A – Připojení k databázím v jazyce Java.* (Ukázka obsahuje jen nejnutnější funkční minimum, nikoliv doporučený a správně ošetřený kód). Druhou možností je exportování tabulek s LOB objekty do textového souboru a následné importování do MSSQL.

Pomocí vytvořeného balíčku *to_MSSQL* je možné přenášet LOB objekty pouze o maximální velikosti 4000 B, větší objekty nebudou přeneseny a musejí se přenést alternativními způsoby.

5.4 Seznam použitých kódů pro pomocné tabulky

Kódy v tabulce *Tabulka 4 – Kódy příkazů pro přenos objektů* jsou v rozmezí 0 – 100 a jsou vyhrazeny jednotlivým příkazům `CREATE` a `ALTER`, které budou přenášeny na MSSQL.

Kódy v tabulce *Tabulka 5 – Kódy chyb a chybových zpráv* jsou v rozmezí 100 – 200 pro chybové zprávy při vytváření příkazů, 500 – 600 pro chybové kódy při migraci příkazů a kód 999 pro případ výskytu přenášeného příkazu delšího než 4000 znaků.

Tabulka 4 – Kódy příkazů pro přenos objektů

Kód	Příkaz	Popis
11	Create table	„Jednoduchá“ tabulka.
12	Create table	Dočasná tabulka.
13	Create table	Indexově orientovaná tabulka.
14	Create table	Seskupená tabulka.
15	Create table	Rozdělená tabulka.
17	Create table	Tabulka typu XMLType.
21	Create constraint	Přidání omezení unikátního klíče.
22	Create constraint	Přidání primárního klíče.
23	Create constraint	Přidání cizího klíče.
29	Alter constraint	Zneplatnění omezení.
31	Create index	Index typu B-strom.
32	Create index	Reverzní index.
33	Create index	Bitmapový index.
34	Alter table a create index	Funkční index, 2 příkazy, zakomentované.
38	Alter index	Zneplatnění neviditelného indexu.
39	Alter index	Zneplatnění neplatného indexu.
40	Create view	Vytvoření pohledu.

Tabulka 5 – Kódy chyb a chybových zpráv

Kód	Příkaz	Popis chyby
100	-	Objekt nebyl nenalezen.

101	Create table	Neznámý datový typ.
102	Create table	Virtuální sloupec.
112	Create table	Dočasná tabulka bude přenesena jako normální, informace o dočasnosti.
114	Create table	Seskupená tabulka bude přenesena jako normální, informace o seskupení.
115	Create table	Rozdělená tabulka bude přenesena jako normální, typ a způsob rozdělení a subrozdělení.
116	Create table	Externí tabulka, nebude přenesena
117	Create table	Tabulka XMLType s definicí XML schéma, XML schéma nebude přeneseno
118	Create table	Objektová nebo vnořená tabulka, tabulka nebude přenesena.
121	Create constraint	Omezení typu check nemusí být kompatibilní s pravidly MSSQL.
122	Create constraint	Nepodporovaný typ omezení.
130	Create index	Index nad jiným typem objektu než je tabulka.
132	Create index	Reververzní index bude přenesen jako normální.
133	Create index	Bitmapový index bude přenesen jako normální.
134	Create index	Funkční (normální/reverzní/bitmapový) index. Vyžaduje analýzu, kontrolu funkčního výrazu.
135	Create index	Nepřenositelný index typu cluster, iot - top, domain.
138	Create index	Neviditelný index bude přenesen jako neplatný.
140	Create view	Přenesení pohledu není zaručeno.
501	Migration_DDL	Objekt v MSSQL databázi již existuje.
502	Migration_DDL	Migrace objektu selhala.
511	Migration_insert	Objekt v databázi neexistuje.
512	Migration_insert	Během příkazu INSERT INTO došlo k chybě.
999	Fce: Uloz_prikaz	Příkaz je příliš dlouhý, tento příkaz nebude možné převést.

Závěr

Cílem práce bylo porovnat dva databázové systémy: Oracle database a Microsoft SQL Server. Následně navrhnout a vytvořit PL/SQL balíček, který umožní migraci objektů z Oracle do MSSQL.

Již během porovnání obou databázových systémů se ukázalo, že systémy jsou značně odlišné a to i přes existenci standardizovaného SQL jazyka pro relační databáze, který měl být sjednocovacím prvkem. Systémy mají svá vlastní proprietární řešení a jednotlivé SQL standardy implementují pozvolna. Rozdíly jsou poznat již během porovnání datových typů, kde je vidět snaha o pokrytí základních datových typů a zároveň je rozšiřovat, případně přidávat další potřebné datové typy. Rozdíly jsou jasně viditelné i v jednotlivých typech tabulek a indexů. V integritních omezeních a pohledech nejsou rozdíly mezi systémy významné.

Při hledání vhodných způsobů řešení migrace a propojení mezi zvolenými databázovými systémy bylo analyzováno a popsáno pět možností, z nichž dvě vyhovovaly zadání práce. V praktické části jsou obě metody zanalyzovány a mělo být pro obě metody vytvořeno funkční řešení. Metodu spojení databází pomocí nahrání JDBC ovladače do Oracle se bohužel nepodařilo na testovacím serveru zprovoznit ani nalézt ověřené informace o úspěšné implementaci této metody. Druhá metoda, propojení databází pomocí Oracle Database Gateways, již byla úspěšná. Migrace mezi zvolenými databázovými systémy je provedena pomocí této metody.

Během praktické části bylo nutné se vypořádat s celou škálou rozdílů mezi jednotlivými databázovými systémy. Těchto rozdílů mezi systémy je velká řada, což sebou nese značná omezení a přenos jednotlivých objektů není vždy optimální, ať už se jedná o datové typy, jednotlivé druhy tabulek nebo druhy indexů. Veškeré změny datových typů je nutné provádět s opatrností a podrobně je sledovat, protože mohou mít negativní vliv na funkčnost aplikací postavených nad novým databázovým systémem. Proto během migrace se uživateli dostává hlášení pokud přenos objektů není možný nebo není optimální. Vzniklé situace je třeba řešit v průběhu samotné migrace.

V průběhu vývoje aplikace se vyskytlo mnoho různých problémů, které byly postupně eliminovány. Jako příklad by se dala uvést tvorba příkazu `INSERT INTO` s předem neznámým počtem a typem jednotlivých vkládaných sloupců. Vypořádat se nepodařilo s jediným problémem, nepodporou LOB objektů mezi Oracle a MSSQL v Oracle Database Gateway. Data těchto datových typů se proto musí přenášet alternativními způsoby.

Spouštění základních dotazů z prostředí Oracle na serveru MSSQL a poskytnutí výsledků formou tabulkové funkce je díky použití Oracle Database Gateway velice jednoduché a intuitivní.

Co se týká případného pokračování vývoje aplikace, je nutné důkladně zvážit, jestli má další vývoj v takto obecné rovině, v jaké byla aplikace psána, vůbec smysl (například z důvodu

již zmíněné nepodpory LOB objektů). Osobně si myslím, že nikoliv a případný další vývoj bych viděl spíše jen pro konkrétní potřeby jednotlivých migrací.

Na závěr je vhodné se zmínit, že i když je balíček plně funkční, jeho nasazení a uskutečnění migrace v produkčním prostředí bych doporučil „jen“ jako přesun dat, nikoliv jako finální řešení, pro které ani není stavěná. Každá databáze má svá specifika a jednotlivé databázové objekty by měly být vyladěny a optimalizovány přesně na daný systém. Obecné řešení nemůže být nikdy optimální.

Literatura

[1] BRYLA, Bob a Kevin LONEY. *Mistrovství v Oracle Database 11g*. Vyd. 1. Brno: Computer Press, 2009, 700 s. ISBN 978-80-251-2189-4.

[2] HERNANDEZ, J. *Myslíme v jazyku SQL: tvorba dotazů*. Vyd. 1. Praha: Grada, 2000, 378 s. ISBN 80-247-0899-X.

[3] HOTEK, Mike. *Microsoft SQL Server 2008: krok za krokem*. Vyd. 1. Brno: Computer Press, 2009, 488 s. ISBN 978-80-251-2466-6.

[4] LACKO, Ľuboslav. *Jak vyzrát na Microsoft SQL Server 2008: správa, konfigurace, programování*. Vyd. 1. Brno: Computer Press, 2009, 469 s. ISBN 978-80-251-2101-6.

[5] LACKO, Luboslav. *Oracle: správa, programování a použití databázového systému.* Vyd. 1. Brno: Computer Press, 2002, 464 s. ISBN 80-722-6699-3.

[6] LONEY, Kevin. *Oracle Database: kompletní průvodce*. Vyd. 1. Brno: Computer Press, 2010, 1368 s. ISBN 978-80-251-2489-5.

[7] MICROSOFT. *Microsoft SQL Server Library* [online]. © 2013 [cit. 2013-01-12]. Dostupné z: <http://msdn.microsoft.com/en-us/library/bb545450.aspx>.

[8] NIELSEN, Paul W. *Microsoft sql server 2008 bible*. Vyd. 1. Indianapolis, 2009, 1642 s. ISBN 04-702-5704-0.

[9] ORACLE CORPORATION. *Ask Tom* [online]. [2013] [cit. 2013-01-12]. Dostupné z: <http://asktom.oracle.com>.

[10] ORACLE CORPORATION. *Oracle Database Documentation Library: 11g Release 2 (11.2)* [online]. © 2013 [cit. 2013-01-12]. Dostupné z: <http://www.oracle.com/pls/db112/homepage>.

[11] ORACLE CORPORATION. *Oracle* [online]. Redwood Shores (CA), [2012] [cit. 2013-01-12]. Dostupné z: <http://www.oracle.com>.

[12] SQL. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-10]. Dostupné z: <http://en.wikipedia.org/wiki/SQL>.

[13] STEPHENS, Ryan K. *Naučte se SQL za 21 dní*. Vyd. 1. Brno: Computer Press, 2004, 581 s. ISBN 80-722-6870-8.

[14] URMAN, Scott, Ron HARDMAN a Michael MCLAUGHLIN. *Oracle: programování v PL/SQL*. Vyd. 1. Překlad Jiří Fadrný. Brno: Computer Press, 2007, 720 s. ISBN 978-80-251-1870-2.

[15] VIEIRA, Robert. *Professional Microsoft SQL server 2008 programming*. Indianapolis: Wiley Pub., 2009, 893 s. ISBN 04-702-5702-4.

Příloha A – Připojení k databázím v jazyce Java

Ukázka aplikace, která se připojí k databázovým systémům MSSQL a Oracle v jazyce Java za použití JDBC ovladačů. Následně přesune `CLOB` objekty z Oracle v tabulce `M_LOB` do téže tabulky v MSSQL za identického `id`.

Zároveň je v ukázce názorně vidět shodnost práce pro různé databázové systémy. Jediné rozdíly jsou ve vlastnostech připojení a následně v samotném SQL příkazu, který může být pro každou platformu různý.

```
package clob_oracle_to_MSSQL;

import java.sql.*;

public class Clob_oracle_to_MSSQL {

  public static void main(String[] args) throws Exception {
    Connection conn_ora = null;
    Connection conn_MS = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      conn_ora = DriverManager.getConnection (
        "jdbc:oracle:thin:@localhost:1521:Oracle_DB_name",
        "oracle_name", "oracle_password"
      );

      Class.forName(
        "com.microsoft.sqlserver.jdbc.SQLServerDriver");
      conn_MS = DriverManager.getConnection(
        "jdbc:sqlserver://localhost:1433;databaseName=mssql_DB",
        "mssql_name", "mssql_password"
      );

      String sql_ora = "SELECT id, cl FROM m_lob ";
      PreparedStatement ps_ora =
        conn_ora.prepareStatement(sql_ora);
      ResultSet rs_ora = ps_ora.executeQuery();

      String sql_ms = "UPDATE M_LOB SET cl = ? WHERE id = ?";
      PreparedStatement ps_ms =
        conn_MS.prepareStatement(sql_ms);

      Integer id;
      Clob body = null;
      while (rs_ora.next()) {
        id = rs_ora.getInt(1);
        body = rs_ora.getClob(2);

        String text =
          (body.length()>10) ? body.getSubString(1, 10):"NULL";
        System.out.println("ID = " + id + ", Length: " +
          (int) body.length() + ", Text: " + text
        );
```

```
        ps_ms.setClob(1, body);
        ps_ms.setInt(2, id);
        int count = ps_ms.executeUpdate();
      }
      System.out.println("=== DONE ===");
      ps_ms.close();
      ps_ora.close();
    }
    catch (SQLException e) {
      e.printStackTrace();
    }
    finally {
      if (conn_ora != null && !conn_ora.isClosed()) {
        conn_ora.close();
      }
      if (conn_MS != null && !conn_MS.isClosed()) {
        conn_MS.close();
      }
    }
  }
}
```