

## DATABÁZOVÝ DESIGN

Podpora výuky databázových systémů na SOŠ, založené na technologiích společnosti ORACLE.

Publikace vznikla v rámci projektu CZ.1.07/1.1.07/02.007,  
Podpora výuky databázových systémů na středních odborných  
školách, založené na technologiích společnosti ORACLE.

© 2011 Vydala Střední průmyslová škola elektrotechniky  
informatiky a řemesel, příspěvková organizace, Křižíkova 1258,  
Frenštát p. R., [www.spsfren.cz](http://www.spsfren.cz)

Studijní kapitoly jsou synchronizovány s mezinárodním vzdělávacím  
programem Oracle Academy. Více informací na [academy.oracle.com](http://academy.oracle.com)  
nebo na portálu [ucimedatabase.cz](http://ucimedatabase.cz).

**Manager projektu:** Mgr. Richard Štěpán

**Překlad:** Oracle Czech, Bc. Tomáš Romanovský,  
Mgr. Dana Mikesková, Mgr. Markéta Kytková

**Metodik:** Bc. Tomáš Romanovský

**Jazyková korektura:** Mgr. Pavlína Chovancová

**Sazba:** Bc. Tomáš Romanovský

**Obálka:** Bc. Tomáš Romanovský

**Tisk:** Reprografické studio LWR GRAPHIC

Žádná část této publikace nesmí být publikována a šířena žádným způsobem a v žádné podobě  
bez výslovného souhlasu vydavatele

*Zvláštní poděkování patří společnosti Oracle Czech za  
dlouholetou podporu vzdělávání v oblasti databázových  
technologií a za spolupráci při vytváření této publikace.*

*Autoři projektu*

## Obsah

1.oddíl.....	3
Úvod k databázovým technologiím.....	3
Data versus informace.....	4
Historie databází.....	5
2.oddíl.....	8
Konceptuální & fyzický model.....	8
Entita, instance, atributy a identifikátory.....	10
Modelování entit a vztahů (ERD).....	12
3.oddíl.....	14
Identifikace vztahů.....	14
Konvence ER Diagramu.....	17
Vyjádření ERDish a kreslení vztahů v diagramu .....	18
4.oddíl.....	20
Supertypy a podtypy.....	20
Dokumentování „podnikových“ pravidel.....	23
5.oddíl.....	25
Typy vztahů.....	25
Řešení M:N vztahu.....	27
6.oddíl.....	29
Umělé, složené a sekundární UID (unikátní identifikátory).....	30
Normalizace databáze.....	33
7.oddíl.....	36
Hierarchické a rekurzivní vztahy.....	36
Modelování historických dat.....	38
8.oddíl.....	41
Konvence pro tvorbu diagramů pro lepší čitelnost.....	41
9.oddíl.....	43
Úvod do relačních databází.....	43
Základní mapování:	
Proces transformace ERD do RMD.....	46
Mapování vztahů:.....	51

# 1. ODDÍL

## Obsah oddílu

- Úvod k databázovým technologiím
- Data versus informace
- Historie databází

## Úvod k databázovým technologiím

### Lekce 01

S příchodem mikropočítačů a jejich vstupem do života každého z nás se zdá, že vzrůstá zájem o počítačově zpracovávaná data, a to ne pouze ve smyslu něco si vypočítat, ale především ve smyslu něco se dozvědět. Vysvětlení je zcela pochopitelné, uvážíme-li, že i ty nejdokonalejší počítačové hry časem omrzí a že programovat v Basicu výpočty typu řešení soustavy dvou rovnic o dvou neznámých nepřináší až tak velké uspokojení.

Člověk by si rád uspořádal a vyhledával některé informace, které často používá a které se v průběhu používání mění. Za předpokladu vhodných prostředků pro ukládání takových informací ve formě dat by byl ochoten prosedět několik večerů u klávesnice a příslušná data si sám vyrobit, koupit nebo vyměnit s jiným podobným nadšencem. Příkladem může být knihovna, úřad, zpracovávání letenek, městské muzeum, nemocnice, podnik apod.

Hotový objekt popsaného typu můžeme nazvat **informačním systémem (IS)**. IS je tedy soubor prvků ve vzájemných informačních a procesních vztazích (informační procesy), které zpracovávají data a zabezpečují komunikaci informací mezi prvky. Z uživatelského hlediska by měl mít IS takové vlastnosti, aby manipulace s ním byla co nejjednodušší (vstup dat, formulace dotazů, použití aplikačních programů), a současně, aby funkce, které poskytuje, byly dostatečně silné a rychlé. Předložit uživateli pružný „prázdný“ IS lze dnes poměrně rychle, avšak s dostatečnou mírou znalostí.

Naším cílem bude ukázat si tzv. databázovou technologii zpracování dat. Databázová technologie je soubor pojmů, prostředků a technik sloužící pro vytváření informačních systémů (IS). Na nejzákladnější úrovni si můžeme představit architekturu IS s databází takto: data jsou organizována v **databázi (DB)**, jsou řízena balíkem programů, který se nazývá **systém řízení bází dat (SŘBD)**. Databáze a SŘBD tvoří tzv. **databázový systém (DBS)**. V naší terminologii můžeme jednoduše psát **DBS = DB + SŘBD**. IS využívá data z DBS buď přímo, nebo je zpracovává aplikačními programy.

# Data versus informace

Lekce 02

dd\_S01\_02

Co se v této lekci naučíte:

- rozlišovat mezi daty a informacemi a uvést příklady dat a informací
- popsat příklad toho, jak se z dat stává informace

Proč se to učit?

- Všechny druhy informací (záznamy školy, záznamy volání z mobilních telefonů, nákupy v obchodě s potravinami) jsou uloženy v databázích. Jsme ve styku s databází každý den, a to vědomě, či nevědomě. Je důležité pochopit, co jsou data uložená v databázi, a co z nich lze získat.

Slova Data, Informace se často používají jako synonyma. Nicméně, mají různé významy.

**Data:** materiál, z kterého vyvozujeme závěry. Fakta, z nichž dedukujeme nová fakta.

**Informace:** znalosti, inteligence, příslušná (určitá) data speciálního významu nebo funkce. Informace je často výsledkem spojování, porovnávání a počítání s daty.

Kdykoliv student, učitel, administrátor (nebo kterákoliv osoba používající počítač) pracuje na webu, sbírají se data. GUI může být unikátní pro danou školu nebo společnost, ale co se děje „za oponou“?

Představte si výsledky testu. Jestliže každý student dostane číselný výsledek, pomocí něhož můžete vypočítat průměr třídy. Pak pomocí průměrů tříd můžete určit průměr školy. Oracle databáze převede nahraná / uložená data a statistické údaje na použitelnou informaci.

**Data:** Výsledek testu každého studenta jsou data.

**Informace:** Průměrný výsledek třídy nebo školy.

## Co je databáze?

**Databáze** je centralizovaná a strukturovaná množina dat uložená v počítači. Poskytuje vybavení na získávání, přidávání, změnu a mazání dat dle potřeby. Taktéž poskytuje vybavení pro transformaci dat na užitečnou informaci.

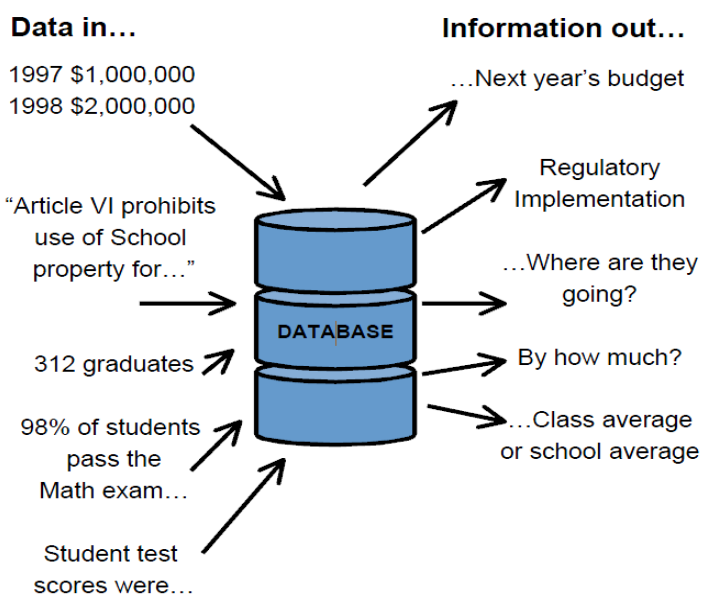
Databáze je obvykle spravována **databázovým administrátorem (DBA)**.

## Dokumenty, obrázky, video a zvuk

U nejmodernějších databází můžete získávat a ukládat širokou škálu dat a dokumentů. Uvnitř databáze se data ukládají v „surové“ podobě. Dotázaná nebo získaná surová data jsou převedena na užitečnější výstupy nebo informace.

**Otázka:** Co má databáze co do činění s mým každodenním životem?

**Odpověď:** Víc, než si myslíte ... Spousta webů, které navštívíte, jsou řízeny databází.



# Historie databází

Lekce 03

dd\_S01\_03

## Co se v této lekci naučíte:

- popsat vývoj databáze a uvést příklad jeho role v podnikatelském světě
- důležité historické inovace ve vývoji a konstrukci databáze
- popsat proces rozvoje databáze

## Proč se to učit?

- Historie poskytuje pohled na to, kde jsme dnes v oblasti informačních technologií. Až budete příště používat počítač, herní systém nebo PDA, uvědomíte si, jak daleko jsme v ICT došli a co všechno předcházelo tomu, abychom se dostali do tohoto bodu.
- Datové modelování je prvním krokem v rozvoji databází. Tato lekce zahrnuje přehled toho, o čem pojednává zbytek studia databází.

## Historie databáze na časové ose

**1960:** Počítače se staly nákladově efektivní pro soukromé společnosti spolu se zvýšením kapacity uložených dat v letech 1970 — 1972: EF Codd navrhuje relační model databáze, který umožňuje oddělení logického uspořádání od fyzického uložení dat.

**1976:** P. Chen představuje konceptuální model (ERM) pro návrh databáze.

**Začátek 80. let:** První komerčně dostupný relační databázový systém se objevil na začátku roku 1980 – Oracle verze 2.

**Polovina 80. let:** SQL (Structured Query Language) je standardizován.

**Polovina 90. let:** Objevuje se použitelný Internet / World Wide Web (WWW). To vede k obrovským aktivitám umožňující vzdálený přístup k počítačovému systému se staršími daty.

**Konec 90. let:** velké investice do internetových firem pomáhají vytvářet nástroje pro tržní propojení Web / Internet / DB.

**Počátek 21. století:** Pokračuje souvislý nárůst používání DB.

**Příklady:** Komerční internetové stránky (yahoo.com, amazon.com), vládní systémy (Úřad státního občanství a imigrační služby, Úřadu pro sčítání lidu), umění, muzea, nemocnice, školy atd.

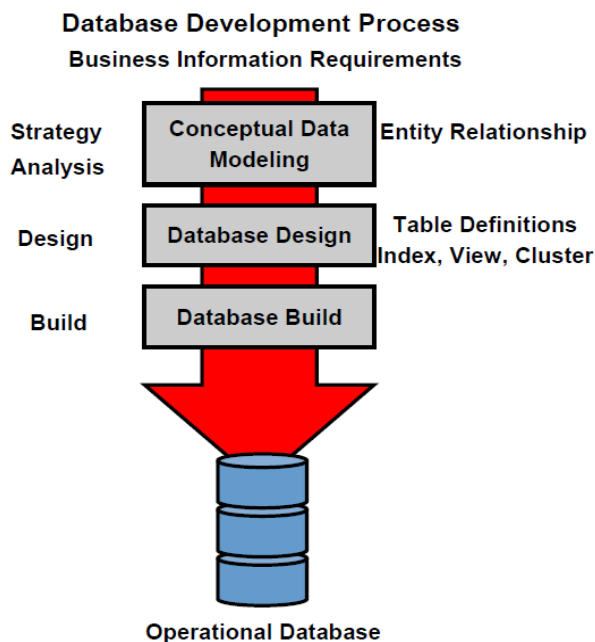
**Otázka:** Co má společného databáze s modelováním dat? Datové modelování je první část procesu vývoje databáze.

## Proces vývoje databáze začíná požadavky na podnikové informace

### PŘÍKLAD

Zde je soubor požadavků na informace:

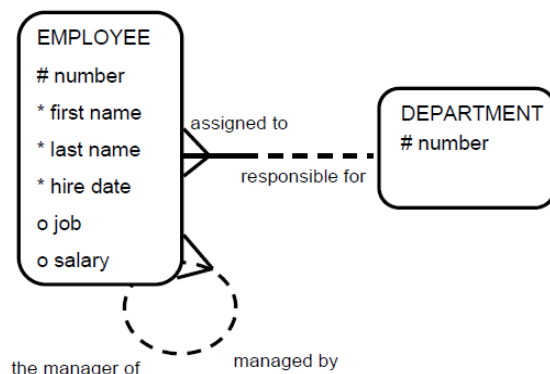
„Řídím oddělení lidských zdrojů velké společnosti. Potřebujeme uchovávat informace o každém z našich zaměstnanců. U každého zaměstnance musíme sledovat jméno, příjmení, pracovní zařazení nebo funkci, datum nástupu a plat. U všech zaměstnanců musíme také sledovat jejich potenciální provizi. Každý zaměstnanec má přiřazeno unikátní číslo zaměstnance. Naše společnost je rozdělena na oddělení. Každý zaměstnanec je přiřazen oddělení – například účetnictví, prodej nebo vývoj. Potřebujeme vědět, který zaměstnanec je zodpovědný za dané oddělení a také umístění oddělení. Každé oddělení má jedinečné číslo. Někteří zaměstnanci jsou manažeři. Potřebujeme vědět, kdo je vedoucím každého zaměstnance.“



## Proces vývoje databáze

ERD („entitně relační model“) model by měl přesně modelovat informace, které potřebuje organizace, a podporovat funkčnost podniku.

### PŘÍKLAD



Následující konceptuální model (ERD) představuje požadavky na informace v oblasti lidských zdrojů.

V návrhu databáze se požadavky na informace projeví v modelu jako entity a vztahy a jsou převedeny do relační databáze použitím tabulky instancí.

### Tabulka má tyto části:

- Název tabulky.
- Názvy sloupců.
- Klíče: primární klíč (PK) je jedinečný identifikátor pro každý řádek dat, cizí klíč (FK) spojuje data jedné tabulky s daty jiné tabulky.
- Null: uveďte, zda sloupce musí mít hodnotu (povinná hodnota).
- Unikátnost: označujeme, pokud je hodnota ve sloupci unikátní v rámci tabulky.
- Datový typ: týká se formátu a definice dat v každém sloupci.

K vytvoření fyzické databáze jsou používány SQL příkazy (DDL — „Data Definition Language“)

### Ukázka tvorby databáze:

```
SQL>CREATE TABLE DEPARTMENT
DEPTNO NUMBER(5) NOT NULL PRIMARY KEY, NAME VARCHAR2(25) NOT NULL,
LOC VARCHAR2(30) NOT NULL);

SQL>CREATE TABLE EMPLOYEES
(EMPNO NUMBER(9) NOT NULL PRIMARY KEY, FNAME VARCHAR2(15) NOT NULL,
LNAME VARCHAR2(20) NOT NULL,
JOB VARCHAR2(15),
HIREDT DATE NOT NULL,
SAL NUMBER(9,2),
COMM NUMBER(9,2),
MGR NUMBER(2) REFERENCES EMPLOYEES DEPTNO NUMBER(5) REFERENCES
DEPARTMENT);
```

## 2. ODDÍL

### Obsah oddílu

- Konceptuální & fyzický model
- Entita, instance, atributy a identifikátory
- Modelování entit a vztahů (ERD)

## Konceptuální & fyzický model

LEKCE 01

dd\_S02\_L01

### V této lekci se naučíte:

- popsat důležitost požadovaných informací
- rozlišovat mezi konceptuálním modelem a fyzickou implementací
- seznam 5 důvodů pro výstavbu konceptuálního datového modelu
- seznam příkladů konceptuálních a fyzických modelů

### Proč se to učit?

- Když budete schopni rozlišit a analyzovat informace, lépe pochopíte, jak věci fungují a potenciálně je vylepšíte.
- Např.:  
jak zrychlit řadu u prodejního pultíku,  
jak v obchodě něco úspěšně vyměnit,  
jak zorganizovat a sledovat rostoucí sbírku CD nosičů.
- Rozpoznání a analýza informací pomáhá zamezit chybám a nedorozuměním. To je důležité v podniku, jelikož se šetří čas a peníze.

### Co je to Konceptuální model?

Konceptuální model je model, který zakresluje funkční a informační potřeby podniku.

Na základě aktuálních potřeb může odrážet budoucí potřeby. Zabývá se pouze potřebami podniku a nezabývá se problémem jejich implementace.

Nazývá se Model entit a vztahů („Entity Relationship Model”). Zobrazuje se jako Diagram entit a vztahů (“Entity Relationship Diagram”).

Firmy využívají data k růstu nebo ke snížení svých nákladů. V procesu vytváření datového modelu podniku představuje právě konceptuální model data podniku.



## Účelem konceptuálního modelu je:

- Přesně popsat potřeby podniku na informace.
- Usnadnit diskuzi.
- Předejít chybám a nedorozuměním.
- Zformovat důležité „ideální systémovou“ dokumentaci vytvořit základ pro fyzický návrh databáze.
- Dokumentovat procesy podniků (to je také známo jako „business rules“).
- Vzít v úvahu předpisy a zákony, které upravují dané výrobní odvětví.

Zhruba od poloviny 70. let se začalo prosazovat tzv. **konceptuální schéma (KS)**, které formalizovaně, ale přitom dostatečně srozumitelně, popisuje reálný svět a ne pouze data v počítači. Konceptuálním schématem popisujeme jednak strukturu světa uživatelské aplikace, jednak jistá fakta, která se v čase nemění, např. Materiál je jednoznačně určen číslem materiálu z číselníku. Tato tvrzení představují **integritní omezení (IO)** na konceptuální úrovni.

Objekty, vyhovující konceptuálnímu schématu, tvoří **informační bázi**. (Je to velmi abstraktní pojem – není „nikde vidět“.)

Implementací informační báze je **databáze**.

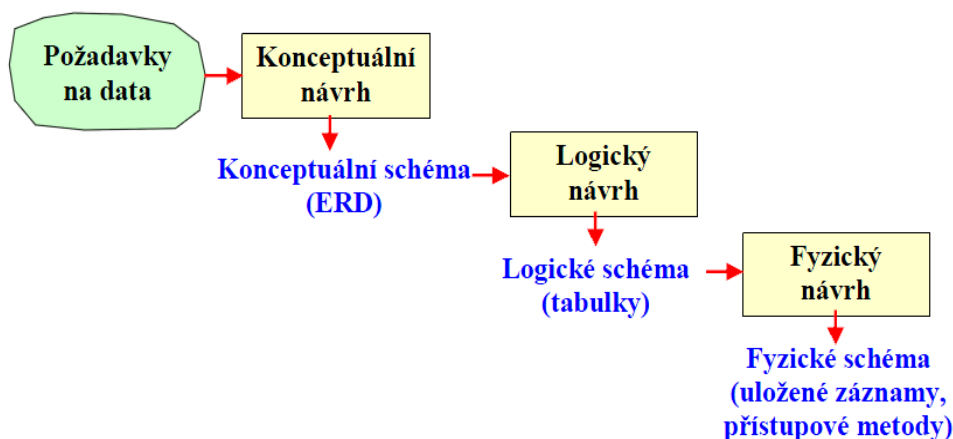
KS je implementačně nezávislé a slouží jako společný základ pro: pochopení objektů aplikace uživateli, projektanty atd. (společný pohled na věc různými skupinami zúčastněných), integraci uživatelských pohledů a návrh implementace (vnášení kompromisu do různých pohledů na tutéž realitu. Odstraňují se zde např. synonyma či homonyma).

**KS je tedy výsledkem analýzy a návrhu IS.** Jde o popis obsahující pro IS všechny důležité objekty reálného světa a tvoří dokumentaci budoucího IS.

Prostředky, kterými se KS vyjadřuje, se nazývají konceptuální či sémantické modely.

Chování aplikace se analyzuje funkční analýzou (většinou málo integrovanou s datovou analýzou).

## Fáze návrhu databáze



## Shrnutí:

**Konceptuální modely** jsou nástrojem pro vytvoření popisu dat v databázi, tj. **konceptuálního schématu**, nezávisle na logické úrovni i fyzickém uložení databáze. Tento popis by měl co nejvěrněji vystihnout konceptuální pohled člověka na danou část reálného světa.

Konceptuální modely slouží obvykle k vytvoření schémat s následnou transformací na databázové schéma (fyzické schéma).

Konceptuální a fyzikální modely jsou tedy uměním plánování, vývoje a komunikace, které produkuje požadovaný výsledek.

Datové modelování zahrnuje získávání požadavků, jejichž popis je zachycen pomocí diagramu. Tento diagram se stane plánem pro navrhování aktuálních požadavků.

Klientovo zadání (konceptuální model) se stane souborem fyzických objektů (fyzický model).

## Entita, instance, atributy a identifikátory

LEKCE 02

dd\_S02\_L02

### Co se naučíte:

- definovat a poskytnout příklad entit
- rozlišovat mezi entitou a instancí entity
- pojmenovat a popsat atributy určených entit
- rozlišit mezi atributem a jeho hodnotou
- rozlišovat mezi povinnými a volitelnými atributy a mezi nestálými a stálými
- dokážete vybrat a obhájit jedinečné identifikátory entit

### Proč se to učit?

- Znalost, jak organizovat a klasifikovat data, umožňuje vyvozovat po-  
užitelné závěry o zdánlivě náhodných faktech. Náš svět, bohatý na tech-  
nologie, vytváří obrovské množství faktů, které potřebují organizovat a  
řadit.
- Je důležité učit se o entitách, protože o nich ukládáme data.  
Příklad: Škola potřebuje uložit data (minimálně) o studentech, učitelích,  
hodinách, třídách a známkách. Je důležité učit se o attributech, protože  
informují o entitách a atributy pomáhají specifikovat, která data po-  
třebujeme sledovat.

### Např.:

- V restauraci potřebujete seznam různých položek seřazených tak  
abyste věděli, kolik požadovat financí.
- Když pro oddělení vytváříte různé zprávy, musíte "vypíchnout" pří-  
slušné zprávy ze seznamu zpráv.

### A co jedinečné identifikátory?

- Je důležité rozumět unikátním identifikátorům, protože rozlišují jednu  
instanci od druhé.

**Např.:**

- Ve třídě potřebuješ odlišit jednu osobu od druhé.
- Při klasifikaci sbírky CD potřebuješ najít konkrétní CD.
- Ve finančním výpisu seznamu transakcí potřebujete rozlišit mezi násobnými transakcemi z jednoho dne.

Projektant se musí podrobně seznámit s modelovanou realitou podniku, specifikovat požadavky na IS a v rámci datové analýzy:

- Identifikuje **entity** (entity types) jako množiny objektů stejného typu (STUDENT, UČITEL, KNIHA, ABONENT, ZAMĚSTNANEC). V objektové terminologii bude tento pojem většinou splývat s pojmem třída.
- Identifikuje typy **vztahů** (relationship types), do kterých mohou entity vstupovat (UČÍ, MÁ\_PŮJČEN,...).
- Na základě přiměřené úrovně abstrakce přiřadí entitám **popisné atributy** (PŘÍJMENÍ, DATUM\_VÝPŮJČKY).
- Formuluje různá **integritní omezení** – pravidla vyjadřující (s větší nebo menší přesností) soulad schématu s modelovanou realitou.

**Proč termín „omezení“?** Protože většinou jde skutečně o omezení v reálném světě – modelované databázi. IO nám tedy také mimo jiné zjednodušují návrh (např. máme-li IO formulované: „Každý klient má v naší bance nejvýše jednu půjčku“ – jistě povede na jednodušší řešení, než kdybychom museli řešit situaci, že klient může mít libovolný počet půjček); a integritní proto, že integrita je obecně soulad dat s realitou.

## Konstrukty E-R modelu:

**Entita** ... objekt reálného světa, který je:

- schopen nezávislé existence,
- je jednoznačně odlišitelný od jiných objektů.
- Entity mají instance. Instance je jeden výskyt entity. Např. (student NOVÁK JOSEF, r.č. 771201005).

**Vztah** ...vazba mezi dvěma nebo více entitami, např. student NOVÁK JOSEF může být ve vztahu „studuje“ k entitě předmět INFORMAČNÍ A KOMUNIKAČNÍ TECHNOLOGIE.

**Atribut** ... údaje přiřazující entitám či vztahům hodnotu (popisného typu), určující některou podstatnou vlastnost entity, např. datum výpůjčky dané knihy daným abonentem. Hodnota atributu může být číslo, řetězec, datum, obraz, zvuk atd. Jedná se o tzv. „datové typy“ nebo „formáty“. Každý atribut má určitý typ dat.

Atributy jsou atomické, tzn. že každý atribut může mít pouze jednu hodnotu (v každém okamžiku) pro každou instanci subjektu.

Vymezení těchto pojmů je volné. Klasifikace dat jako ENTITA nebo VZTAH závisí na pohledu analytika.

**Pravidla pro pojmenování:** entity převážně vyjadřovány podstatnými jmény; vztahy převážně vyjadřovány slovesy; nepoužíváme diakritiku a mezeru v názvu.

# Modelování entit a vztahů (ERD)

LEKCE 03

dd\_S02\_L03

## Co se naučíte:

- definovat význam volná implementace (implementation-free) tak, jak se vztahuje k implementaci datových modelů a návrhu databáze
- seznam 4 cílů modelování entit a vztahů
- identifikovat ERD

## Proč se to učit?

- Diagram entit a vztahů (ERD) je konzistentní nástroj používaný k reprezentaci datových požadavků bez ohledu na použitý typ databáze, dokonce bez samotného použití databáze.

## Bez-implementační modely (volné)

- Dobrý konceptuální datový model se nemění při změně typu databáze, na níž je nakonec systém vystavěn. A proto říkáme, že model je bez implementace.
- Datový model by měl zůstat stejný, i když není použita žádná databáze, např. Data jsou uložena jako kartičky v kartotéce.

## Co je ERD?

Seznam všech entit a atributů, rovněž seznam vztahů mezi důležitými entitami. Poskytuje podklady jako popis entit, datové typy a omezení dat. Pozn. Model nemusí obsahovat diagram, ale typicky je velmi užitečný.

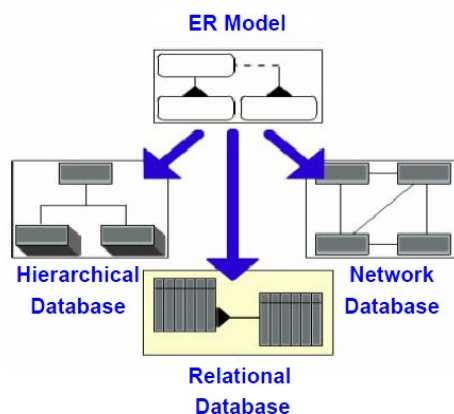
## Cíle modelování ERD:

Máme celkem 4 cíle modelování ERD:

- podchytit všechny potřebné informace,
- zajistit, aby se každá informace objevila pouze jedenkrát,
- vymodelovat informaci, která by se dala odvodit z jiné modelované informace,
- umístit informaci na očekávaném logickém místě.

Představte si školní záznamy. Od prvního dne se o vás sbírají data. Pravděpodobně jsou o vás zaznamenány informace o absenci, chování, rozvrhu a známkách.

## Database Types



## SCÉNÁŘ DJ ON DEMAND

Přečtěte si celý obchodní scénář firmy DJ on Demand, níže uvedený, a poté ověřte vytvořený ERD.

Obchod jsme rozjeli jako skupina přátel, kteří organizovali večírky, kde jsme používali vlastní hudbu. Pak jsme si řekli, že založíme firmu, abychom sledovali své zájmy a vydělali více peněz. Nazvali jsme se DJ on Demand („Žádání DJ“).

Všichni zde pracující jsme partneři. Každý máme určitou zodpovědnost. Projektový manager domlouvá první schůzku s klientem, kde se projednává typ akce. Jde o narozeniny? Svatbu, výročí, promoce? Kdy akce proběhne?

Po dohodě následuje plánovač akce, který s klientem probere konkrétní místa, občerstvení, výzdobu a další podrobnosti.

DJ s klientem probere požadovanou hudbu. Projektový manager dohlíží na DJ a plánovače. Taktéž podepisuje výdaje spojené s projektem. Máme velkou sbírku CD. Každé CD obsahuje několik písní a stejná píseň se může objevit na několika CD. Rozděluje písně podle typu (hip-hop, salsa, rock, pop, classic ...).

Můžeme nabídnout vzorový seznam písní, upravených podle typu akce. Samozřejmě i klient si může vyžádat určité písně. Seznam klientů nám roste. Naši klienti se k nám vracejí, protože mají rádi naši práci, a připravujeme pro ně další akce. Máme hodně aktivní zákazníky, pro které pořádáme i několik akcí najednou.

Akce dělíme tematicky. Např. tropická svatba, karnevalový večírek, výročí ve stylu 60. let. To nám pomáhá vybrat místo a vytvořit si představu, jak by měl být DJ a další hudebníci oblečení. Někteří partneři mají specializace a kvalifikace – takže nám téma pomůže odhadnout správnou osobu pro danou akci.

Akce konáme ve veřejných budovách i v soukromých domech. Manažer akce navštíví veřejné místo nebo soukromý dům. Manažer akce domlouvá podmínky, jak s nájemcem veřejné budovy, tak i s vlastníkem domu. Jelikož několik partnerů pracuje na akci a několik partnerů akci řídí, míváme zaznamenáno, kdo pracuje na jaké akci, jakou práci vykonal a kdy.



# 3. ODDÍL

## Obsah oddílu

- Identifikace vztahů
- Konvence ER Diagramu
- Vyjádření ERDish a kreslení vztahů v diagramu

## Identifikace vztahů

LEKCE 01

dd\_S03\_L01

### V této lekci se naučíte:

- interpretovat a popisovat volitelnost vztahu (členství ve vztahu)
- interpretovat a popisovat poměr ve vztahu
- nastavit vztah mezi entitami aplikováním pravidla poměru a volitelnosti vztahu

### Proč se to učit?

- Tím, že jste schopni porozumět identifikaci vztahů mezi entitami, lépe chápete vztahy mezi různými daty.
- Díky této schopnosti rozeznáte, jak se navzájem ovlivňují odlišné části systému. Např.: Entity STUDENT a KURZ mají mezi sebou určitý vzájemný vztah.
- Abychom správně vymodelovali zadání, vztahy jsou stejně důležité jako entity.

### Vztahy v datových modelech:

Vztahy:

- reprezentují něco důležitého v zadání,
- ukazují, v jakém vztahu jsou entity,
- vždy existují mezi dvěma entitami,
- vždy mají dvě strany,
- jsou pojmenovány na obou koncích,
- mají volitelnost (členství ve vztahu),
- mají stupeň poměru ve vztahu.

## Co je volitelnost ve vztahu?

Vztahy jsou buď povinné, nebo nepovinné. Vycházíme-li z toho, co známe o instancích entit, můžeme určit volitelnost, když odpovíme na 4 otázky: (příklad „firma“)

- Musí mít každý zaměstnanec práci? Jinými slovy: je toto povinné, nebo nepovinné členství ve vztahu zaměstnanec k práci?
- Mohou mít zaměstnanci více než jednu práci?
- Musí být každá práce prováděna nějakým zaměstnancem? Jinými slovy, má práce povinný nebo nepovinný vztah k zaměstnanci?
- Může být práce vykonávána více než jedním zaměstnancem?

## Co je poměr ve vztahu?

Poměr ve vztahu zahrnuje stupeň ve vztahu. Odpovídá na otázku „kolik“.

**Příklad:** Kolik má zaměstnanec zaměstnání? Jedno nebo více? Pouze jedno? Kolik zaměstnanců má dané zaměstnání? Jeden nebo více?

## Volitelnost a poměr ve vztahu

### ■ PŘÍKLAD

Každý zaměstnanec musí mít pouze jedno zaměstnání.

Každé zaměstnání může být obsazeno jedním nebo více zaměstnanci.

Každý produkt musí být klasifikován pouze jedním typem produktu.

Každý typ produktu může klasifikovat jeden nebo více produktů.

## Vztahy

- Každé sedadlo může být prodáno jednomu nebo více cestujícím.
- Každý pasažér si může koupit jedno sedadlo.
- Sedadlo je prodáno cestujícímu (nebo cestujícím, proto dojde ke změně rezervace).
- Pasažér si koupí nebo rezervuje jedno sedadlo.

### ■ OBCHODNÍ SCÉNÁŘ 1:

Jaké jsou vztahy v následujícím scénáři?

„Rádi rozdělujeme veškerou hudbu podle typu – každou písničku, každý soundtrack. Různé typy jsou jazz, country, pop .... Podle potřeby můžeme přidat nové typy, vlastně jsme teď přidali nový typ rapová hudba a také víme, že píseň může být zařazena do různých typů, ale pro naše potřeby jsme si vybrali pro každou písničku jeden typ.“

**volitelnost = „musí“ nebo „může“?**

**poměr = kolik?**



## Píseň má typ: volitelnost a poměr

Každé **písni** je přiřazen jeden (pouze jeden) **typ**.

Každý **typ** může klasifikovat jednu nebo více **písni**.

... co když neexistuje pro píseň typ?

Uvádějí obchodní (podniková) pravidla, že každá píseň má typ? Pokud ano, pak je potřeba přidat typ navíc.

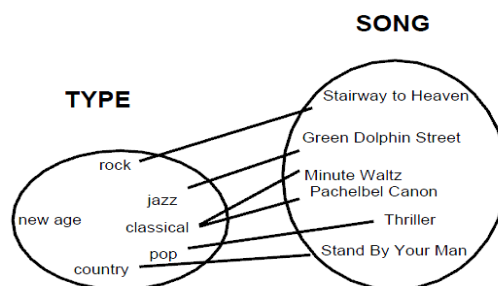
Může existovat typ bez písni?

Proč bys měl mít typ bez písni?

Pod kolik typů může píseň patřit?

Pravidla určují poměr. Ve vztahu.

Pokud písnička může patřit pod více než jeden typ, pak by měl být poměr určen jako: Každou píseň musí klasifikovat jeden nebo více typů.



## OBCHODNÍ SCÉNÁŘ 2:

Jaké jsou vztahy v následujícím scénáři?

„U nás v restauraci přijde host k baru a objedná si. Zákazník si může objednat jen pro sebe nebo pro celou skupinu lidí. Např.: Jestliže matka objedná pro sebe a děti, pak bereme matku za zákazníka, který je vlastníkem objednávky a je zodpovědný za platbu. Zákazník samozřejmě může učinit tolik objednávek, kolik chce.“

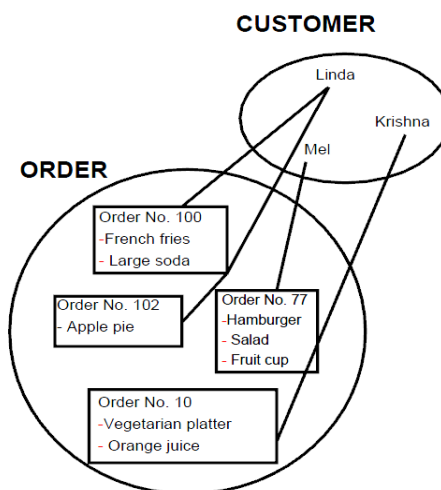
### Zákazník má objednávky: volitelnost a poměr

volitelnost = musí nebo může?

Poměr ve vztahu = kolik?

Každá objednávka musí být obdržena jedním (pouze jedním) zákazníkem.

Každý zákazník musí dát jednu nebo více objednávek.



## OBCHODNÍ SCÉNÁŘ 3:

Vztahy v rámci stejné entity.

Vyzkoušejte si následující scénář:

„Potřebujeme vést záznam o zaměstnancích a jejich managerech. Každý zaměstnanec má jednoho manažera, včetně generálního ředitele, který řídí sám sebe. Každý manažer může řídit více zaměstnanců. Jelikož jsou manažeři také zaměstnanci, existuje zde jen jedna entita – zaměstnanec“

### Vztahy:

Zaměstnanec řídí zaměstnance.

Zaměstnanec je řízen jedním zaměstnancem.



# Konvence ER Diagramu

LEKCE 02

dd\_S03\_L02

V této lekci se naučíte:

- vytvářet komponenty (konstrukty) ER Diagramu, které představují entity a atributy, podle konvencí diagramu

Proč se to učit?

- Lidé po celém světě hovoří různými jazyky, ale všichni rozumí dopravním značkám.
- Je efektivní zpracovávat informaci způsobem, kterému rozumí mnoho lidí. ERD fungují v tomto významu. Můžete popisovat nebo napsat věci odlišně, protože mluvíte jinak (přízvuk je jiný), ale každý maluje diagramy podle stejných principů.

## DJ on Demand: Klienti, akce a typy

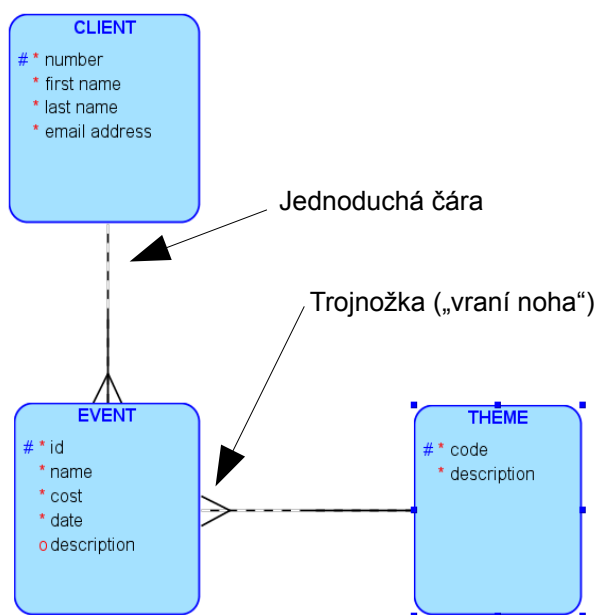
Popis reálného světa – viz „Scénář DJ on DEMAND“ v lekci Modelování entit a vztahů.

### Konvence kreslení ERD

- Entity jsou reprezentovány rámečky.
- Jména entit zapisujeme do rámečků.
- Názvy entit jsou vždy v jednotném čísle a velkými písmeny.

### Principy kreslení:

- Seznam Atributů umísťujeme pod jméno entity.
- Povinné atributy označujeme hvězdičkou: „\*“.
- Nepovinné atributy označujeme kroužkem: „○“.
- Jedinečné identifikátory označujeme mřížkou: „#“.
- Vztahy jsou čáry, které propojí entity.
- Tyto čáry jsou buď plné, nebo čárkované. Čáry končí jednoduchou čarou nebo trojnožkou (vraní noha).



Více se o významu vztahových čar dozvíte dál.

# Vyjádření ERDish a kreslení vztahů v diagramu

LEKCE 03

dd\_S03\_L03

V této lekci se naučíte:

- přesně popsat (slovy) vztahy mezi entitami (schéma ERDish), správně kreslit a označit vztahy v ERD

Proč se to učit?

- Většina činností má unikátní terminologii – slova, která mají speciální význam v rámci dané činnosti – kterou lidé používají, aby předali a zpracovali informaci.
- Modelování dat má také jedinečnou terminologii, kterou nazýváme ERDish. Když se naučíte, jak vytvořit diagramy a jak hovořit ERDish, získáte běžnou terminologii pro komunikaci s klienty a administrátory, kteří implementují váš návrh.

**ERDish** je jazyk, kterým vyjadřujeme vztahy mezi entitami.

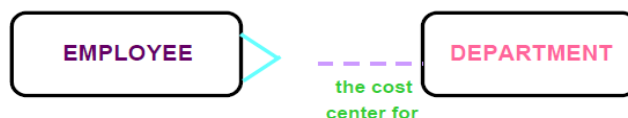
Už jsme tímto jazykem hovořili a psali, když jsme určovali vztahy a specifikovali volitelnost a poměr ve vztahu. Prostě jednoduše rozložíme každou ERDish větu na jednotlivé části.

## Části ERDish:

1. EACH („každá“)
2. ENTITA A
3. OPTIONALITY („musí“/ „může“)
4. RELATIONSHIP NAME (jméno vztahu)
5. CARDINALITY („jedna a pouze jedna“/ „jedna nebo více“)
6. ENTITA B



1. EACH
2. **EMPLOYEE** (entity A)
3. **MUST BE** (optionality, solid line)
4. **WORKING IN** (relationship name)
5. **ONE** (cardinality, single toe)
6. **DEPARTMENT** (entity B)

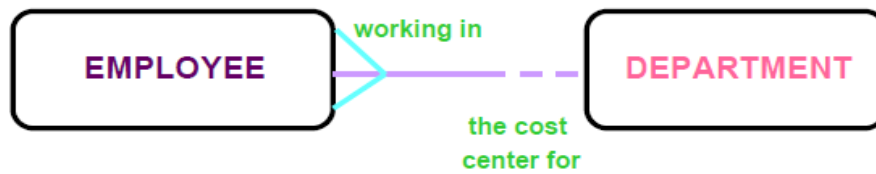


1. EACH
2. **DEPARTMENT** (entity B)
3. **MAY BE** (optionality, dotted line)
4. **THE COST CENTER FOR** (relationship name)
5. **ONE OR MORE** (cardinality, crow's foot)
6. **EMPLOYEE** (entity A)

Jelikož má vztah dvě strany, čteme nejprve jednu stranu – zleva doprava.

Potom čteme vztah zprava doleva.

Nyní spojíme obě vyjádření dohromady:



1. EACH
2. **EMPLOYEE** (entity A)
3. **MUST BE** (optionality, solid line)
4. **WORKING IN** (relationship name)
5. **ONE AND ONLY ONE** (cardinality, single toe)
6. **DEPARTMENT** (entity B)

1. EACH
2. **DEPARTMENT** (entity B)
3. **MAY BE** (optionality, dotted line)
4. **THE COST CENTER FOR** (relationship name)
5. **ONE OR MORE** (cardinality, crow's foot)
6. **EMPLOYEE** (entity A)

# 4. ODDÍL

## Obsah oddílu

- Supertypy a podtypy
- Dokumentování „podnikových“ pravidel

## Supertypy a podtypy

LEKCE 01

dd\_S04\_L01

### V této lekci se naučíte:

- definovat a uvést příklad podtypu
- definovat a uvést příklad supertypu
- uvést pravidla týkající se entit a podtypů, uvést jejich příklady
- použít pravidla pro supertypy a podtypy vyhodnocením přesnosti ER diagramů, které je reprezentují
- použít pravidla pro supertypy a podtypy a zahrnout je vhodně do diagramu

### Proč se to učit?

- Supertypy a podtypy se často vyskytují v reálném světě – objednávky jídla (konzumace na místě, s sebou), nákupní tašky (papírové, plastové), typy platby (šek, hotovost, úvěr).
- Chápání příkladů z reálného světa nám pomáhá pochopit, jak a kdy je modelovat.

Některé instance entity mají často atributy a/nebo vztahy, které se v jiných instancích nevyskytují. Představte si podnik, který potřebuje sledovat platby od zákazníků. Zákazníci mohou platit v hotovosti, šekem nebo kreditní kartou.

Všechny platby mají některé společné atributy: datum platby, výše platby atd.. Ale pouze kreditní karty budou mít atribut "číslo karty".

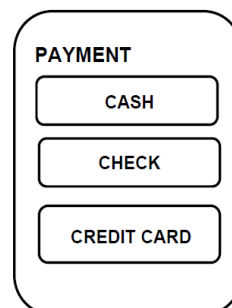
U kreditních karet a plateb šekem budeme chtít vědět, který ZÁKAZNÍK platbu provedl, přičemž u plateb v hotovosti to není potřeba.

Měli bychom vytvořit jedinou platební entitu nebo tři samostatné entity HOTOVOST, ŠEK, KREDITNÍ KARTA? A co se stane, pokud v budoucnu zavedeme čtvrtou metodu platby?

Někdy má smysl rozdělit entitu do podtypů. Např. v případě, kdy má skupina instancí zvláštní vlastnosti, jako jsou atributy nebo vztahy, které existují pouze pro tuto skupinu. V tomto případě se entita nazývá "supertyp" a každá skupina se nazývá podtyp.

### Podtyp:

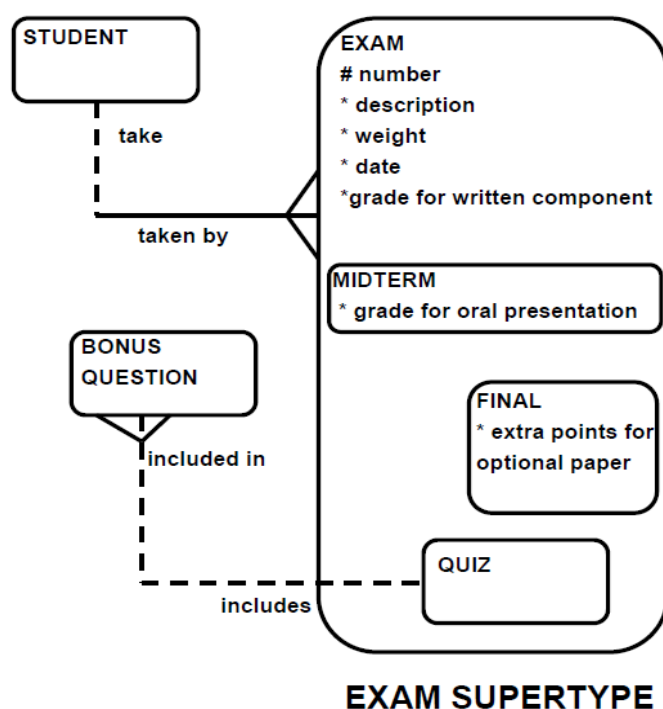
- přebírá všechny atributy supertypu
- přebírá všechny vztahy mezi supertypy
- má obvykle své vlastní atributy nebo vztahy
- je vybrán v rámci supertypu
- nikdy neexistuje sám o sobě
- může mít vlastní podtypy
- také se mu říká "podentita"



### ■ PŘÍKLAD :

ZKOUŠKA je supertyp zahrnující KVÍZ, POLOLETNÍ ZKOUŠKA a ZÁVĚREČNÁ ZKOUŠKA.

Podtypy mají několik společných atributů. Tyto společné atributy jsou uvedeny na úrovni supertypu. Totéž platí pro vztahy. Podtypy přebírají všechny atributy a vztahy entity typu supertyp.



## Vždy více než jeden podtyp

Když je ER model kompletní, nikdy nemá samostatné podtypy. Jinak řečeno, pokud má entita podtyp, měl by vždy existovat alespoň jeden další podtyp. Dává to smysl. K čemu by bylo rozlišovat mezi entitou a jediným podtypem?

Z této myšlenky plynou dvě pravidla pro podtypy:

- **Úplné:** Každá instance supertypu je zároveň instancí jednoho z podtypů.
- **Vzájemně se vylučující:** Každá instance supertypu je instancí pouze jednoho podtypu.

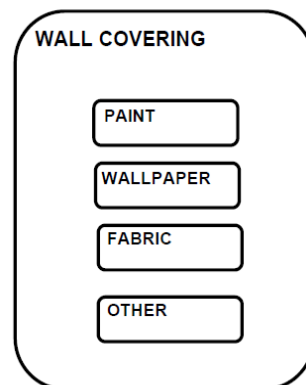
### POVRCHOVÁ ÚPRAVA STĚN

Ve fázi koncepčního modelování je dobrým zvykem zařadit podtypu OSTATNÍ, aby bylo jisté, že je váš seznam podtypů vyčerpávající – že jste ošetřili každou instanci supertypu.

## Podtypy vždy existují

Každé entitě lze vždy přiřadit podtypy. Vždy můžete vytvořit pravidlo, jak rozdělit instance do skupin.

Ale to není to hlavní. Důvodem pro tvorbu podtypů by vždy měla být potřeba ukazovat zároveň podobnosti a rozdíly.



## Správná identifikace podtypů

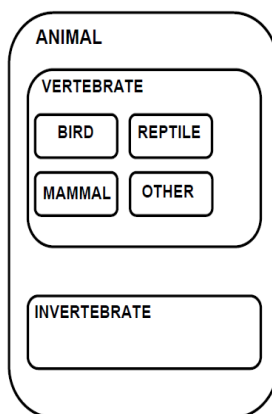
Při zvažování supertypů a podtypů si můžete položit tři otázky pro ověření, zda jste podtyp správně identifikovali:

- (1) Je tento podtyp druhu supertyp?
- (2) Ošetřil jsem všechny možné případy? (úplný)
- (3) Je možné daný příklad zařadit pouze pod jeden podtyp? (vzájemně se vylučující)

## Vnořené podtypy

Podtypy můžete vnořit. Pro snadné čtení -- "čitelnost" – obvykle ukazujete podtypy pouze do dvou úrovní, ale neexistuje žádné pravidlo, které by vám bránilo jít i do dalších úrovní.

### VNOŘENÝ SUPERTYP ŽIVOČICH



# Dokumentování „podnikových“ pravidel

LEKCE 02

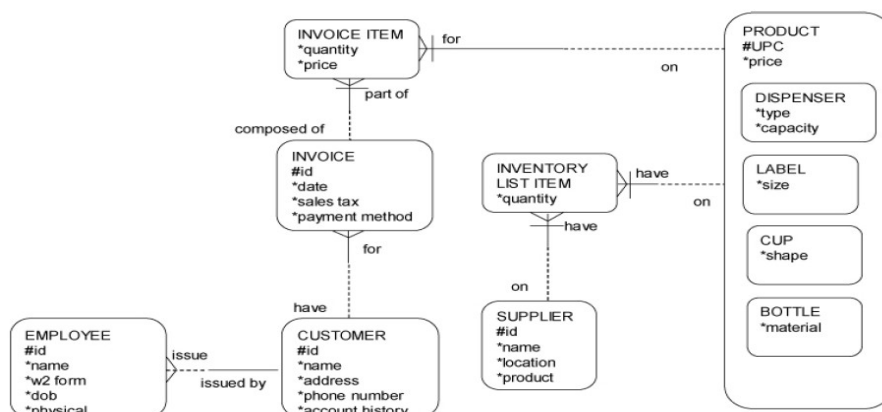
DD\_S04\_L02

V této lekci se naučíte:

- definovat a tvořit strukturální obchodní pravidla
- definovat a tvořit procesní obchodní pravidla
- rozpoznat, že některá obchodní pravidla vyžadují programování
- vytvořit diagram obchodních pravidel, je-li možné je vyjádřit ER modelem

Proč se to učit?

- Dokážete identifikovat tuto firmu?
- Jedním z hlavních cílů datového modelování je zajistit, že všechny informace, které jsou potřeba pro běh podniku, jsou identifikovány.
- Identifikace a dokumentování obchodních pravidel jsou klíčová pro ověření přesnosti a úplnosti datového modelu.
- Je důležité si uvědomit, že ne všechna obchodní pravidla lze vyjádřit ve formě ER diagramu. Některá obchodní pravidla je nutné naprogramovat.



## Dva typy obchodních pravidel: strukturální a procesní

Strukturální obchodní pravidla uvádějí typy informací, které se mají ukládat, a jak jsou elementy těchto informací vzájemně propojeny.

Procesní pravidla souvisejí s workflow nebo obchodním procesem.

Strukturální obchodní pravidla lze téměř vždy vyjádřit ER diagramem. Některá procesní obchodní pravidla nelze vyjádřit diagramem, ale i tak je nutné je zdokumentovat.

Mnoho procesních pravidel souvisí s časem: událost A musí stát dříve než událost B.

Strukturální obchodní pravidla uvádějí typy informací, které se mají ukládat, a jak jsou elementy těchto informací vzájemně propojeny.

## Příklady strukturálních obchodních pravidel:

Všechny objednávky v restauraci musí zpracovávat zaměstnanec (konkrétně, pracovník přijímající objednávky).

Neexistuje žádný samoobslužný systém objednávek.

Všichni učitelé na naší škole musí vlastnit platnou akreditaci.

## DISKUSE:

Jaké druhy pravidel u vašeho zaměstnavatele se týkají vás?

- Každá směna, kterou odpracuji, se musí zadokumentovat na záznamové kartě zaměstnance.
- Každá směna musí probíhat pod dohledem nadřízeného.

Naše škola má mnoho obchodních pravidel:

- Je rozumné/efektivní, aby třída neměla přiděleného třídního učitele?
- Je rozumné/efektivní, aby dva studenti měli stejné studentské ID číslo nebo vůbec žádné ID číslo?
- Je rozumné naplánovat učitelů výuku ve třídě, pokud do ní není zapsán žádný student?
- Je rozumné dovolit studentům chodit do školy, pokud nejsou zapsáni do žádné (jedné nebo více) třídy?

Procesní pravidla souvisí s workflow nebo obchodním procesem.

## Příklady procesních obchodních pravidel:

První kontakt s klientem DJs on Demand musí provést projektový manažer.

Schválení cestovní žádosti na konkrétní událost musí podepsat projektový manažer pro tuto událost.

## DISKUSE

- Studenti musí absolvovat algebru a geometrii, aby se mohli zapsat na trigonometrii. Dokážete to znázornit ER diagramem?
- Jak byste to naprogramovali?
- Pokud student absolvoval dané předměty, napadá vás další obchodní pravidlo, které by škola v tomto scénáři mohla chtít?

Při vytváření koncepčního datového modelu nelze vždy namodelovat všechna obchodní pravidla. Některá pravidla, jako ta níže uvedená, je nutné naprogramovat jako procesy, které interagují s daty:

**každý zaměstnanec, který má více než 10 přesčasových hodin týdně, musí dostat 1,5 násobek hodinové mzdy.**

Nebo:

**zákazníkům, jejichž závazky na účtu jsou 90 dní po splatnosti, nebude umožněno dávat další objednávky.**



# 5. ODDÍL

## Obsah oddílu

- Typy vztahů
- Řešení M:N vztahů

## Typy vztahů

LEKCE 02

dd\_S05\_L02

V této lekci se naučíte:

- rozlišovat a vytvářet příklady vztahu jedna ku jedné (1:1)
- rozlišovat a vytvářet příklady vztahu jedna ku více (1:N)
- rozlišovat a vytvářet příklady vztahu více ku více (M:N)
- rozpoznávat redundantní (vícenásobné) vztahy a odstranit je z ERD

Proč se to učit?

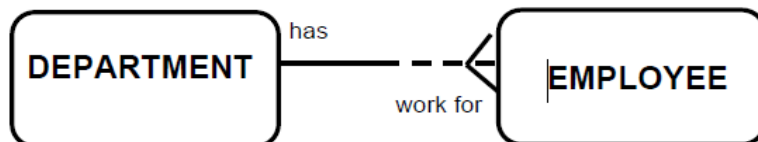
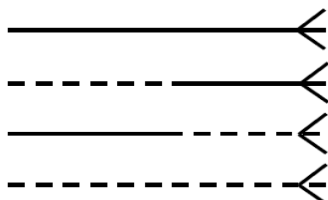
- Může jedna osoba vlastnit více CD nebo pouze jedno? Může jedno CD vlastnit více osob?
- Tak jak doladujeme svůj model, chceme si být jisti, že vztahy mezi entitami řádně modelují pravidla naší činnosti. **Nezapomeňte!** Čím dříve budete promýšlet návrh ERD do detailů, tím lépe se vyhnete drahým chybám v budoucnosti.

### Jedna ku více (1:N)

V ER modelu jsou nejčastější typy vztahů jedna ku více. Už jste viděli několik příkladů.

#### Relationship Types

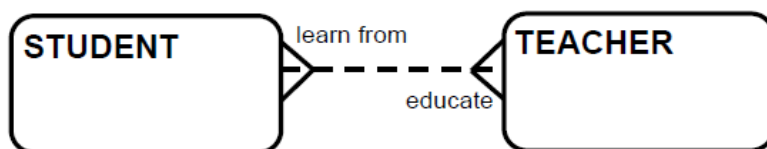
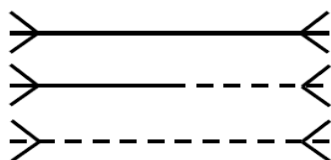
1:M



## Více ku více (M:M), (M:N)

Různé typy M:N vztahu jsou běžné zvláště v prvních verzích ERD. V dalším stádiu procesu modelování se většina vztahů M:N (takřka všechny) rozloží.

### Relationship Types M:M

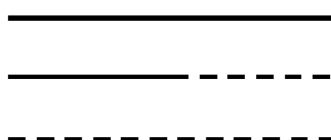


## Jedna ku jedné 1:1

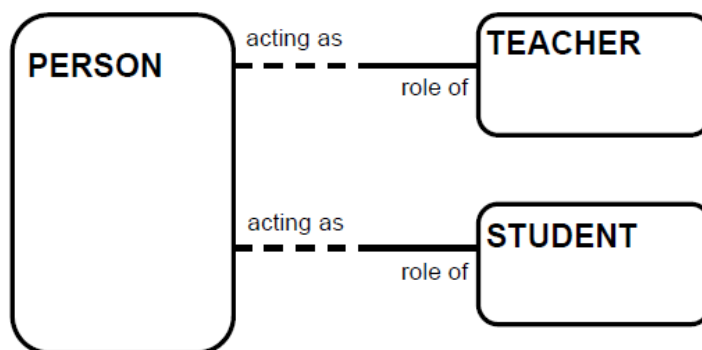
V každém ERD obvykle najdete pouze pár typů vztahu jedna ku jedné.

Obvykle se objevují na jednom konci vztahu.

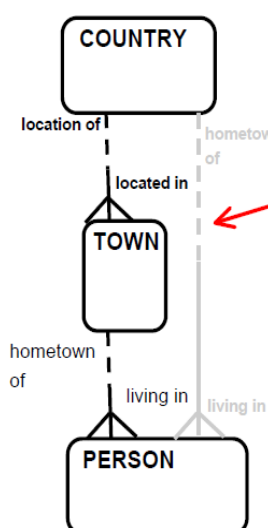
### Relationship Types 1:1



1:1 vztah (všechny 3 varianty) se také objevují, když některé z entit reprezentují různá stadia procesu

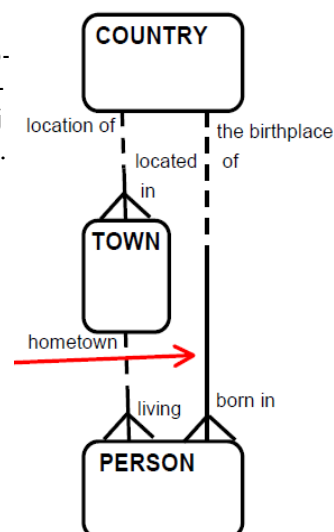


## Redundantní vztah



Redundantní vztah můžeme odvodit z jiného vztahu v modelu. Příklad nalevo – můžete odvodit vztah osoby k zemi a tento vztah můžete odvodit ze dvou ostatních vztahů a mohli byste jej odstranit z modelu, tak jak je znázorněno vlevo.

Avšak, pozor na vyvozování, že vztah je redundantní pouze na struktuře. Pro kontrolu čtete vztahy. ERD napravo neodráží redundantní vztah.



# Řešení M:N vztahu

LEKCE 02

dd\_S05\_L03

V této lekci se naučíte:

- identifikovat atributy, které patří do M:N vztahu
- uvést kroky vedoucí k vyřešení M:N vztahu s použitím průnikové entity
- určit UID průnikové entity a aplikovat jej v entitě relačního diagramu

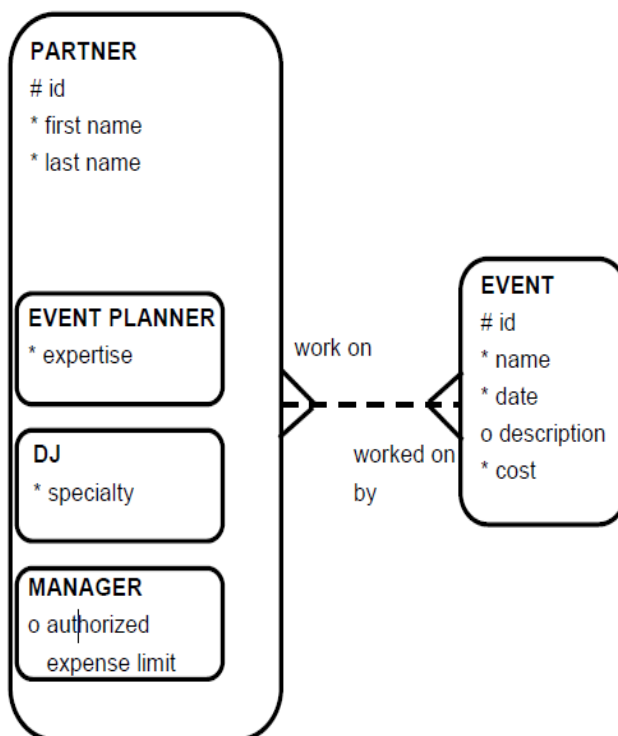
Proč se to učit?

- Tato lekce vám pomůže dokončit model – možná budete potřebovat vytvořit nové entity nebo nové vztahy založené na podnikovém zadání.
- Taktéž pomůže určit rozsah datového modelu – modelujete pouze důležité ze zadání.

## Vztah skrývací atribut – skrytý atribut

Podle podnikového zadání DJ on Demand může být každému partnerovi určena práce na jedné nebo více akcích. Každá akce může být prací pro jednoho nebo více partnerů. Když EVENT\_PLANNER nebo PROJEKT\_MANAGER pracují na jedné akci, chceme, aby zaznamenávali STAV své práce.

Ke které entitě by patřil atribut STAV?

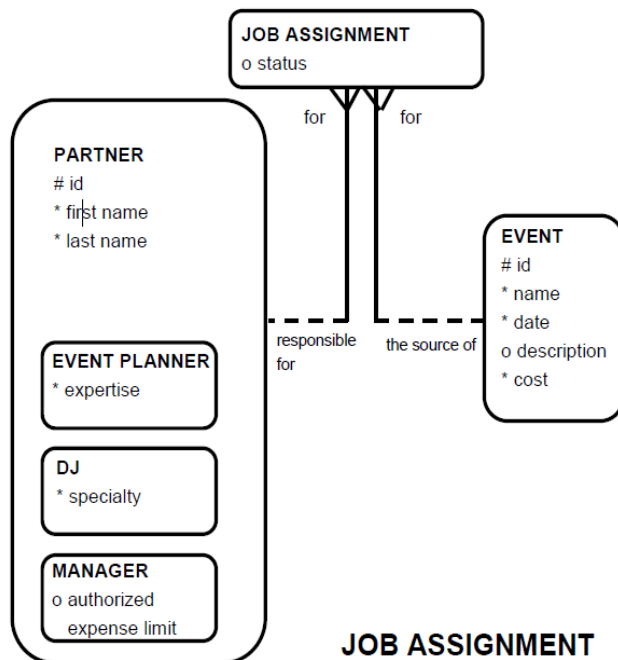


## Řešení vztahu M:N

Pro vyřešení vztahu M:N potřebujeme třetí entitu. **Ta se nazývá „průniková“ entita.**

Průniková entita – JOB ASSIGNMENT (PRACOVNÍ ZARÁZENÍ) – byla přidána včetně atributu stav. Z původního M:N vztahu vznikly dva 1:N vztahy.

Který atribut by mohl být UID pro průnikovou entitu?



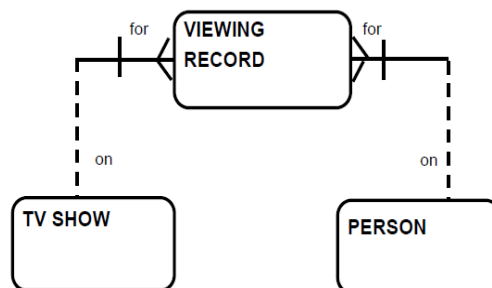
Jedinečný identifikátor (UID) často pochází z původních vztahů a je znázorněn mřížkou. V tomto případě se vztahy průnikové entity, vzniklé z původních entit, nazývají blokové.

## ■ PŘÍKLAD ŘEŠENÍ M:M: POŘADY V TELEVIZI

Každý pořad může být sledován jednou nebo více osobami. Každá osoba může sledovat jeden nebo více pořadů.

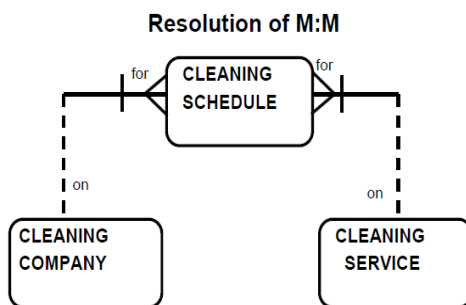
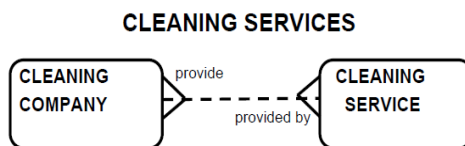


### Resolution of M:M



## ■ PŘÍKLAD ŘEŠENÍ M:N: ÚKLIDOVÁ SLUŽBA

Každá společnost může poskytnout jednu nebo více úklidových služeb. Každá úklidová služba může být poskytnuta jednou nebo více společnostmi.



# 6. ODDÍL

## Obsah oddílu

- Umělé, složené a sekundární UID (unikátní identifikátory)
- Normalizace databáze
- Normální formy (NF)

# Umělé, složené a sekundární UID (unikátní identifikátory)

LEKCE 01

DD\_S06\_L01

V této lekci se naučíte:

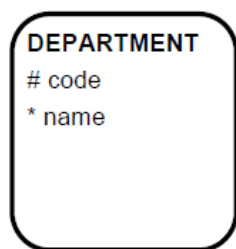
- definovat různé typy unikátních identifikátorů (UID)
- definovat kandidátní UID a vysvětlit, proč může mít entita někdy více než jeden kandidátní UID
- analyzovat obchodní pravidla a zvolit z kandidátů to nejvhodnější primární UID
- rozpoznat a diskutovat o otázkách identifikace v reálném světě

Proč se to učit?

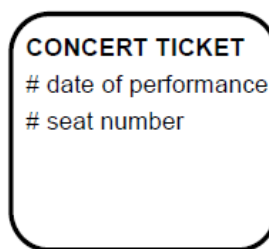
- Unikátní identifikátor (UID) je v relačních databázích velmi důležitý. Je to hodnota nebo kombinace hodnot, které umožní uživateli nalézt tuto unikátní položku mezi všemi ostatními. Identifikace toho pravého atributu, kombinace atributů a/nebo vztahů je dovednost, kterou si musí osvojit každý tvůrce databází. Unikátní identifikátor umožňuje najít svůj záznam v souboru, určitou kartu v balíčku karet, váš balíček ve skladu a konkrétní část dat v databázi.

## Jednoduchý UID versus složený UID

UID, který má formu jediného atributu, označujeme jako jednoduchý UID. Někdy však jeden atribut nestačí k unikátní identifikaci instance entity. Pokud je UID kombinací atributů, říkáme mu složený UID.



Simple Unique Identifier



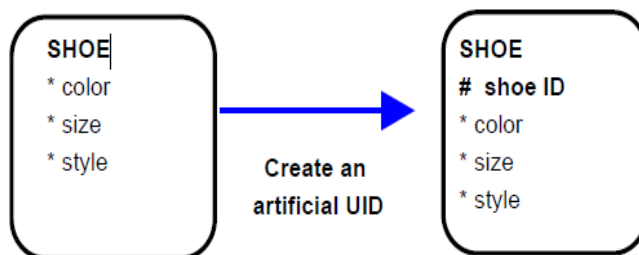
Composite Unique Identifier

## Umělý UID

Umělé UID jsou ty, které se nevyskytují v přirozeném světě, ale jsou vytvořeny pro účely identifikace v systému.

Lidé se nerodí s "čísly", ale mnoho systémů přiřazuje unikátní čísla pro identifikaci osob: ID studenta, ID zákazníka atd.

## PŘÍKLAD

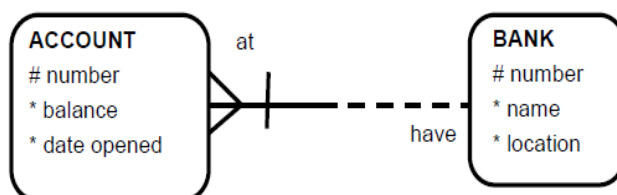


Bota má svou barvu, velikost, styl, ale žádné opravdu popisné "číslo". Obchod s obuví ale přiřadí unikátní čísla každému páru bot, takže je mohou jednoznačně identifikovat.

### UID z blokových vztahů

Někdy je UID kombinace atributu a vztahu. Jaký je UID ÚČTU? Je umělý? Je složený?

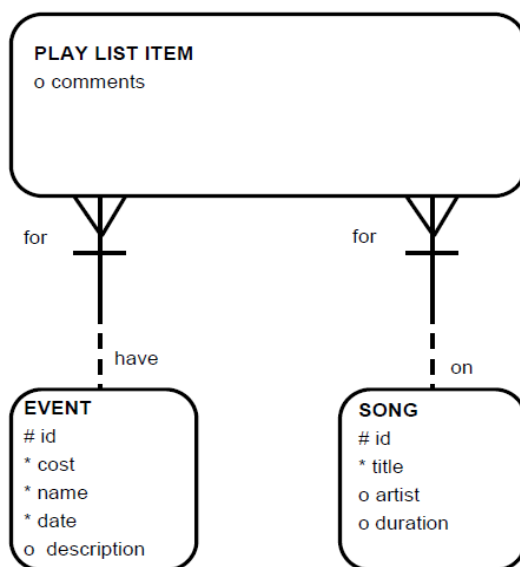
Dva lidé mohou mít stejné číslo bankovního účtu, ale u různých bank. Mezibankovní převody vždy vyžadují kód banky kromě čísla bankovního účtu.



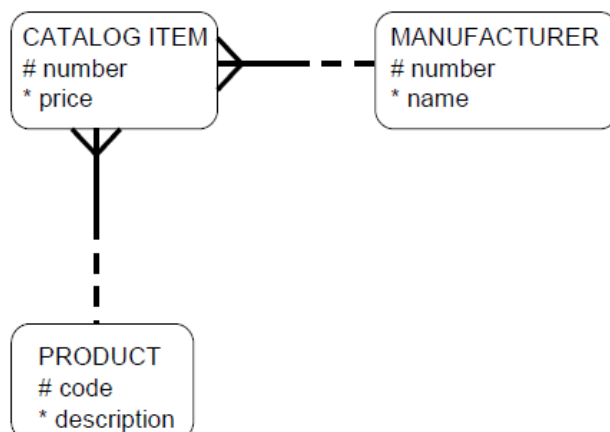
### UID z blokových vztahů: průnikové entity

Jak jsme již viděli dříve, řešení M:M vztahu často vede k blokovým vztahům od průsečíkové entity k těm původním.

V tomto příkladě vychází UID PLAY LIST ITEM z EVENT a SONG. Poznáte to podle bloků u vztahů.



Je možné, aby průsečíková entita použila umělý atribut jako UID místo blokových vztahů k původním entitám.



Každý VÝROBCE může vyrábět jeden nebo více PRODUKTŮ (boty, košile, džíny atd.). Každý PRODUKT může být vyroben jedním nebo více VÝROBCI (boty Nike, boty Adidas, džíny Levi's atd.).

KATALOGOVÁ POLOŽKA řeší tento vztah typu více ku více. Položka v katalogu se dá jednoznačně identifikovat podle čísla výrobce a kódu produktu. Vztahy nejsou blokovány, protože se místo toho vytvořilo umělé UID – katalogové číslo

## Kandidátské UID

Někdy je možný více než jeden UID.

Například když si objednáte produkt z komerční webové stránky, obvykle vám přidělí unikátní kód zákazníka a také jste vyzváni k zadání vaší emailové adresy.

Každá z těchto položek vás jednoznačně identifikuje a dá se použít jako UID. Obě jsou tedy kandidátské UID.

Pouze jeden z kandidátských UID se vybere jako skutečné UID. Tomu se říká primární UID. Ostatní kandidáti se nazývají sekundární UID.

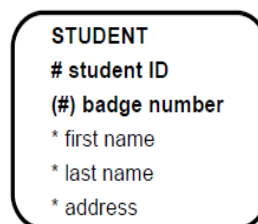
ID studenta se zvolilo jako primární UID u obou těchto entit STUDENT.

První entita má jeden sekundární UID, druhá má dva sekundární UID (z nichž jeden je složený).

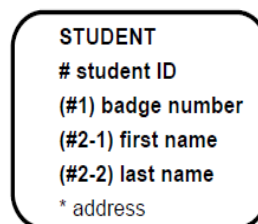
## Identifikace: databáze versus reálný svět

Unikátní identifikátory nám umožňují odlišit jednu instanci entity od jiné. Jak uvidíte později, stávají se primárními klíči v databázi. Primární klíč umožňuje přístup ke konkrétnímu záznamu v databázi.

V reálném světě ale není vždy tak snadné odlišit jednu věc od jiné.



One Primary UID  
One Secondary UID



One Primary UID  
Two Secondary UIDs



# Normalizace databáze

LEKCE 02 – 04

dd\_S06\_L02-04

**Normalizace** je proces organizace dat v databázi. Zahrnuje vytvoření tabulek, definici jejich struktur a nastavení vazeb mezi tabulkami podle pravidel ochrany dat a maximální flexibility databáze. Tato pravidla se snaží vyloučit dva faktory:

## 1. redundance (vícenásobné uložení stejných dat)

Několikanásobně uložené údaje jednak plýtvají prostorem na disku, jednak komplikují údržbu. Např. adresa firmy – lépe centrálně umístit v tabulce ADRESAR\_FIREM než pokaždé znovu v evidenci objednávek, faktur, dodacích listů ... ,

## 2. nekonzistentní závislost dat

Nekonzistentní závislosti dat rozumíme např. zaznamenávání platu zaměstnance v adresáři firem u firmy, ve které je zaměstnán. Plat zaměstnance je závislý na zaměstnanci a patří do tabulky zaměstnanců a ne do adresáře firem. Nekonzistentní závislost ztěžuje dostupnost k údajům.

Pro odstranění těchto faktorů bylo definováno několik pravidel. Každé pravidlo je nazýváno „**forma normalizace**“. O datech, která byla upravena daným pravidlem (formou normalizace), říkáme, že jsou v dané „**normální formě**“. Jsou-li dodrženy první tři formy normalizace, říkáme, že databáze je normalizována třetí formou normalizace nebo že databáze je ve třetí normální formě (3NF).

## První forma normalizace (1NF)

Cíl: Neopakovat skupiny v tabulkách.

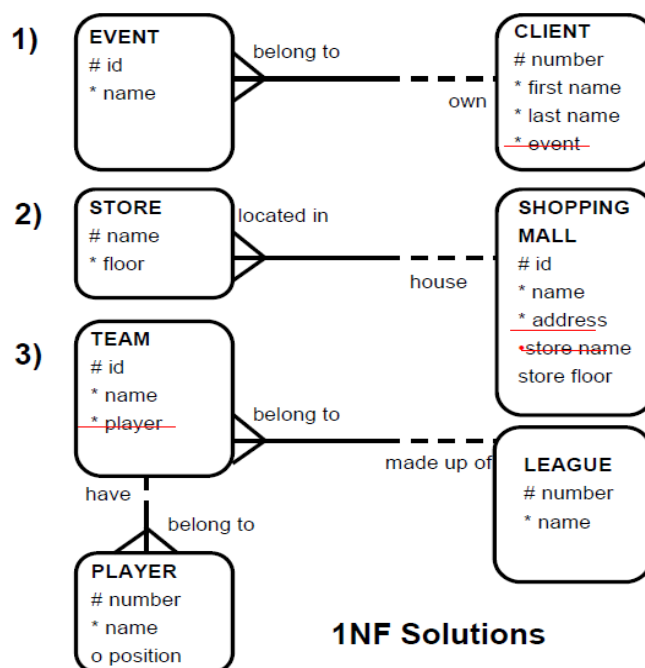
Jak?

- zrušit opakující se skupiny dat v individuálních tabulkách
- vytvořit samostatnou tabulku pro každou množinu svázaných údajů
- pro každou samostatnou množinu definovat primární klíč
- nepoužívat více položek pro uložení stejných dat

## ■ PŘÍKLAD :

Odstranění vícehodnotového atributu *DETI(jmeno, RC):multi* u entitního typu ZAMESTANEC.

## PŘÍKLAD



Pokud jsou všechny atributy v entitě jednoduché, pak říkáme, že entita je v 1NF.

## Druhá forma normalizace (2NF)

Cíl: Vyřadit redundantní údaje.

Jak?

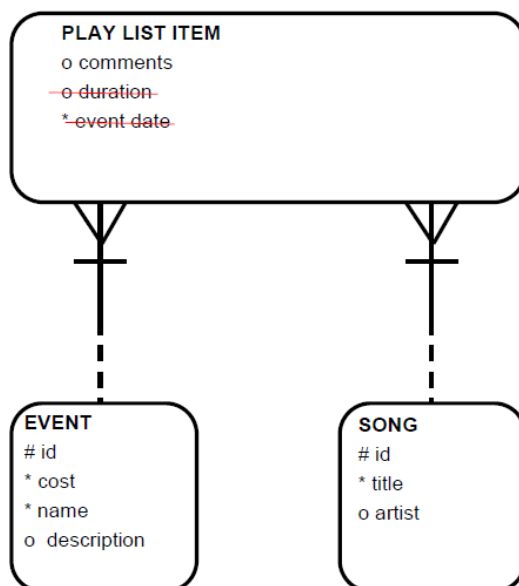
- vytvořit samostatnou tabulku pro množiny hodnot, které jsou použity ve více záznamech
- spojit tabulky pomocí klíčů

## PŘÍKLAD:

Adresa firmy – nebude opakovaně uvedena v mnoha evidencích (objednávka, faktura ...), ale bude uvedena v jediné tabulce "adresář firem" a připojena k jiné tabulce pomocí klíče.

## PŘÍKLAD:

Pojďme se podívat na mírně upravený ERD podnikání DJ. Co je špatného na tomto diagramu?  
Odpověď: atributy doba trvání (duration) a datum události (event date) jsou chybně umístěné.  
Délka závisí pouze na SONG (píseň) a datum události závisí pouze na EVENT (událost).



## Třetí forma normalizace (3NF)

Cíl: Vyřadit data nezávislá na klíči.

Jak?

- vyřadit položky, které nezávisí na primárním klíči

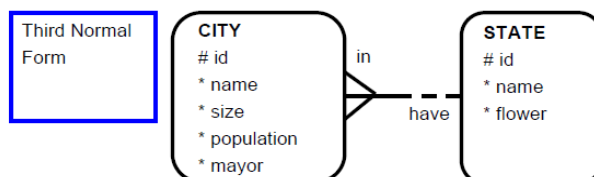
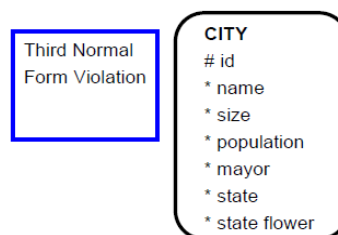
## PŘÍKLAD:

V evidenci zaměstnanců (tab. ZAMESTNANCI) potřebuji evidovat informace o univerzitě, na které získal zaměstnanec vzdělání. Nejenom název univerzity, ale i adresu. Adresa univerzity je již závislá na univerzitě a ne na pracovníkovi, proto vytvořím samostatnou tabulku UNIVERZITY a spojím se ZAMESTNANCI pomocí klíče.

## PŘÍKLAD

Přemýšlejte o systému, který sleduje informace o městech – velikost, počet obyvatel, starosta atd. První model ukazuje entitu CITY, která obsahuje informace o zemi (state). Ačkoli je ZEMĚ atributem města, státní symbol je opravdu atributem země.

Druhý model, s novou entitou STATE (země), je ve třetí normální formě.



# 7. ODDÍL

## Obsah oddílu

- Hierarchické a rekurzivní vztahy
- Modelování historických dat

## Hierarchické a rekurzivní vztahy

LEKCE 02

dd\_S07\_L02

### V této lekci se naučíte:

- definovat a uvést příklad hierarchického vztahu
- identifikovat UID v hierarchickém modelu
- definovat a uvést příklad rekurzivního vztahu
- znázornit rekurzivní vztah v ERD podle podnikového scénáře
- sestavit model s využitím rekurzí a hierarchií s cílem vyjádřit stejný konceptuální význam

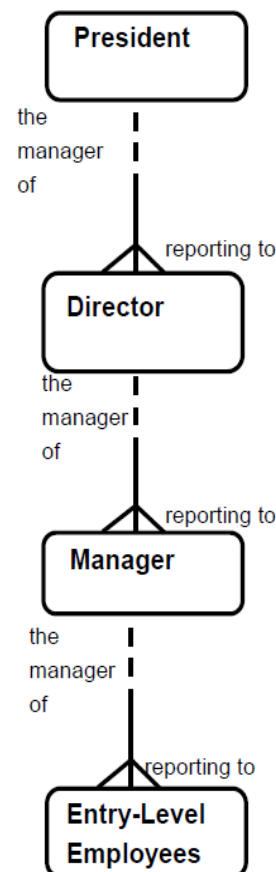
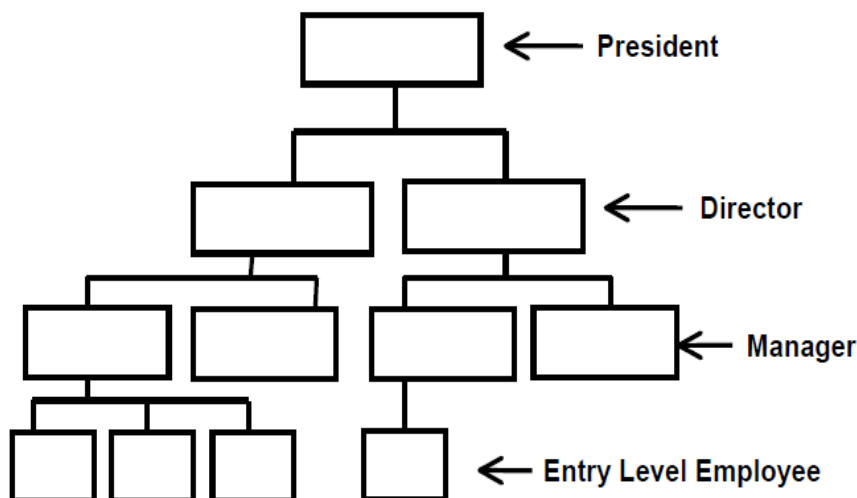
### Proč se to učit?

Role jsou často organizovány podle hierarchie – v práci (manažer, vedoucí skupiny, úředník, úředník, pracovník, který připravuje pokrmy) nebo ve škole (ředitel, zástupce ředitele, učitelé, další zaměstnanci). Hierarchická data jsou velmi častá. Jejich pochopení vám pomůže lépe modelovat:

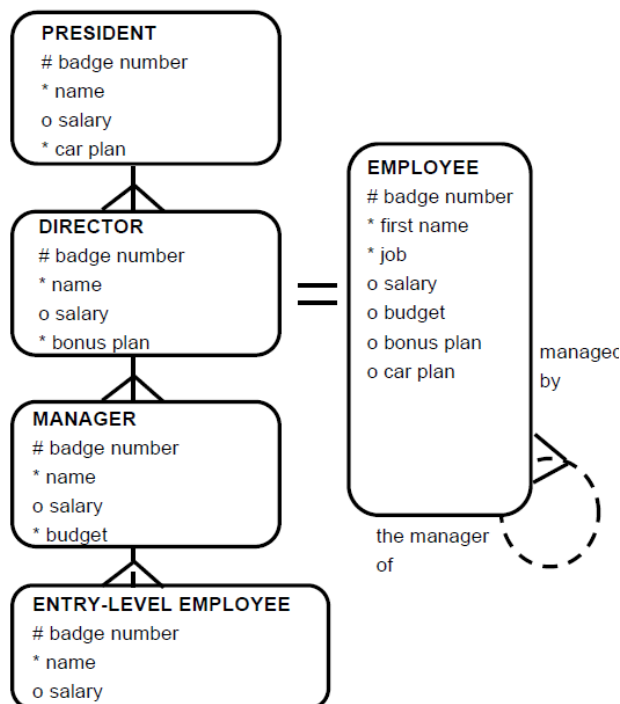
- organizační schémata firmy,
- stavební konstrukce,
- rodinné stromy,
- ... a mnoho dalších hierarchií, které nalezneme v reálném světě.

## Vztahy v organizačním schématu

Organizační schéma můžeme vyjádřit tímto datovým modelem. Jaké UID jsou zde pro každou entitu?



## Hierarchie versus rekurzivní vztah



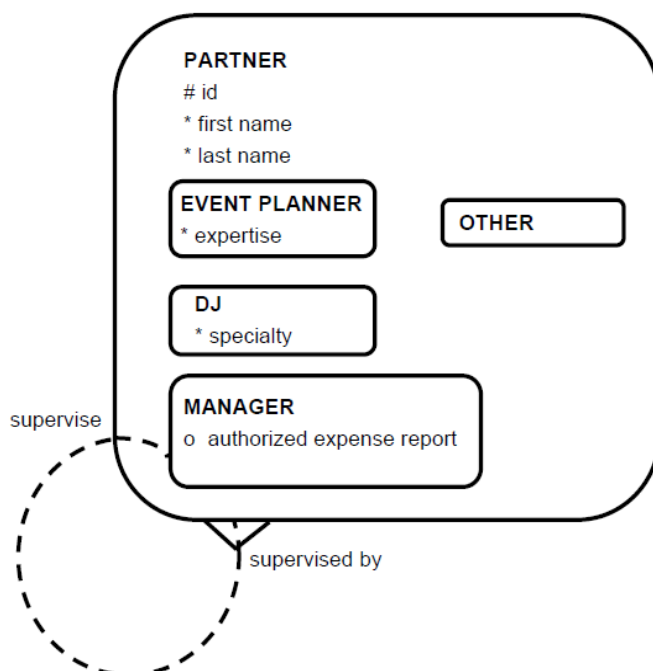
Oba tyto modely reprezentují všechny zaměstnance.

Ten vlevo představuje **hierarchickou strukturu**. Ten vpravo pracuje s **rekurzivním vztahem**.

Který z nich je podle vás lepší?

## ■ PŘÍKLAD: DJS ON DEMAND (PODNIKOVÝ SCÉNÁŘ)

V modelu DJS má manažer projektu celkovou odpovědnost za událost a řídí ostatní zaměstnance (plánovač událostí, DJ) pracující na dané události. Rozhodli jsme se zobrazit tuto hierarchii pomocí rekurzivního vztahu.



## ■ PŘÍKLAD: OBCHODNÍ SCÉNÁŘ PRO AUTOMOBILOVOU VÝROBU

U výrobce automobilů považujeme všechny základní díly, montážní podčásti a části, sestavy a produkty za instance entity nazvané KOMPONENT. Model můžeme vytvořit jako jednoduchý rekurzivní vztah.

Namodelujte data z "kusovníku" jako rekurzivní vztah typu více ku více:

- každý komponent může být součástí jednoho nebo více KOMPONENTŮ,
- každý komponent se může skládat z jednoho nebo více KOMPONENTŮ.

## Modelování historických dat

LEKCE 02

dd\_S07\_L03

V této lekci se naučíte:

- identifikovat potřebu sledovat data, která se mění v čase
- sestavit modely ERD, které obsahují elementy "dat v čase"
- identifikovat UID entity, která ukládá historická data, a vysvětlit a zdůvodnit výběr UID
- sestavit konceptuální model založený na daném obchodním scénáři
- aplikovat pravidla pro tvorbu E-R diagramů (entita-vztah) a vytvořit tak ERD, který odráží obchodní pravidla

- prezentovat a interpretovat datový model posluchačům
- vytvořit písemnou dokumentaci doprovázející ústní prezentaci a ERD

### Proč se to učit?

- Kolik jsi měřil/a v 5ti letech? Kolik jsi měřil/a v 10ti letech?
- Kolik měříš nyní? Pokud vaši rodiče tyto míry během vašeho dětství zapisovali, zaznamenávali tím historická data. Většina podniků potřebuje sledovat některá historická data. Pomáhá jim to identifikovat trendy a vzorce, které mohou být základem pro obchodní inovace nebo zlepšení procesů.
- Historie výpůjček filmů je užitečná pro prodejnu filmů. Vedoucí prodejny může zjistit, které filmy jsou populární a které by měly být přesunuty do zadních regálů.
- Napadá vás použití pro farmaceutickou společnost, potraviny nebo pekárnu nebo továrnu na zpracování mořských plodů?
- Jaká historická data budou chtít sledovat a proč?

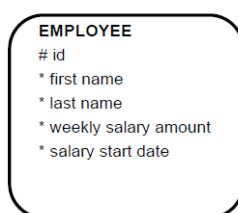
Kdy je to nutné modelovat data v průběhu času?

Zeptejte se svého klienta:

- Je nutný záznam auditu?
- Mohou se hodnoty atributů měnit v čase?
- Mohou se vztahy měnit v čase?
- Potřebujete generovat sestavy ze starších dat?
- Potřebujete uchovávat předchozí verze dat? Pokud ano, po jakou dobu?

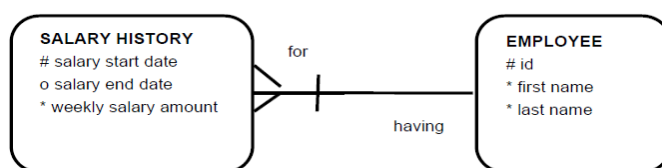
Organizace musí uchovávat údaje o platech zaměstnanců. Všichni zaměstnanci jsou vypláceni týdně.

Původně byla namodelována následující entita ZAMĚSTNANEC.



Dodatečné požadavky upřesňují, že organizace potřebuje uchovávat historická data o tom, jak a kdy se změnily platy zaměstnanců během jejich pracovního poměru.

Chcete-li modelovat změny platu v průběhu času, přidejte entitu SALARY HISTORY (HISTORIE PLATU).



UID entity HISTORIE PLATU je související ID ZAMĚSTNANCE a datum zahájení výplaty mzdy.

### PŘÍKLAD

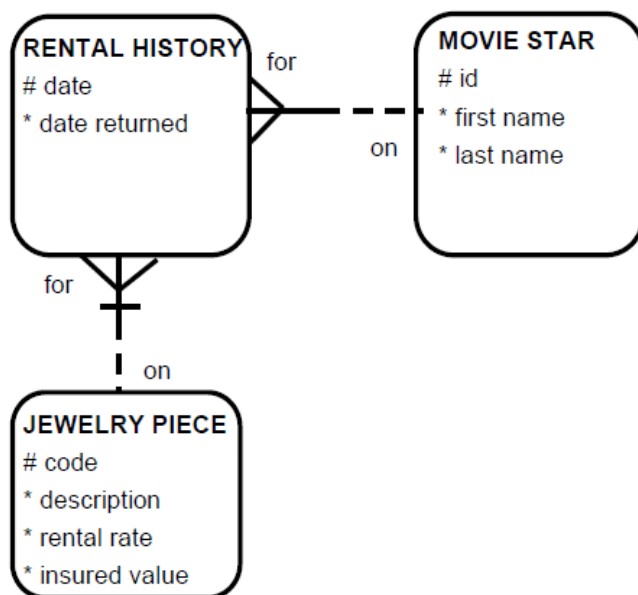
Klenotnictví půjčuje filmovým hvězdám šperky (náhrdelníky, náramky atd.) na zvláštní příležitosti, jako je udílení cen nebo premiéry filmů. Rádi by sledovali historii pronájmu jednotlivých šperků. Následující ER model sleduje pouze aktuálního nájemce daného šperku.

Jak byste vztah upravili, aby se sledovala historie?

Vztah mezi ŠPERK a FILMOVÁ HVĚZDA by se měl změnit na vztah M:M, který pak řeší průsečíková entita HISTORIE PRONÁJMU.

Jaký UID má HISTORIE PRONÁJMU?

UID HISTORIE PRONÁJMU je datum pronájmu a UID ŠPERKu (znázorněno blokováním vztahem).





# 8. ODDÍL

## Konvence pro tvorbu diagramů pro lepší čitelnost

LEKCE 01

dd\_S10\_L01

V této lekci se naučíte:

- aplikovat pravidla Oracle pro tvorbu diagramů datových modelů
- identifikovat velkoobjemové entity v diagramu datového modelu a vysvětlit jejich významu pro firmu
- Překreslit daný diagram datového modelu a zvýšit tak jeho přehlednost a čitelnost
- Rozpoznat užitečnost dělení složitého ERD do více funkčních subdiagramů

Proč se to učit?

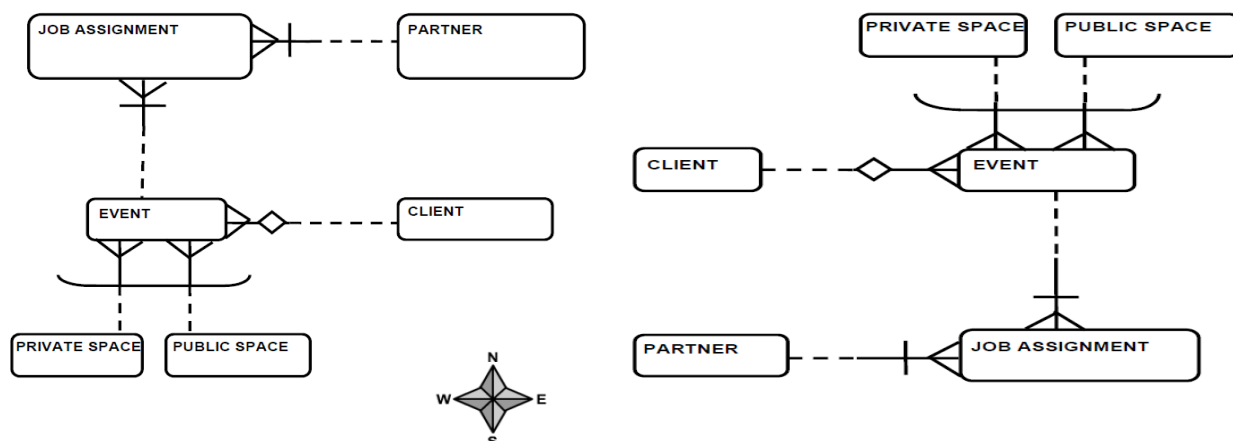
- Co kdyby si všichni výrobci obuvi zavedli své vlastní velikosti?
- Co kdyby každý architekt používal jiný systém k zákresu plánu budovy?
- Dodržování stejných konvencí usnadňuje práci v týmu. Jak se ERD zvětšuje a komplikuje, je stále těžší vyjádřit jej v jasném a čitelném formátu.

Dodržováním konvence, že "vrány létají k jihu a na východ" se velkoobjemové entity dostanou do levé horní části ERD.

Velkoobjemové entity jsou ty, které mají největší počet instancí ve srovnání s jinými entitami.

Dodržováním konvence, že "vrány létají na sever a na západ" se velkoobjemové entity dostanou do spodní pravé části ERD.

Velkoobjemové entity jsou často "centrální" nebo významnější entity v ERD. Budou mít nejvyšší počet vztahů k jiným entitám a většina obchodních funkcí bude mít vliv na data uložená v této entitě.



Často budete mít mix podle velikosti prostoru a vlastních preferencí.

Jasnost a čitelnost jsou hlavní kritéria.

Velkoobjemové entity nejsou vždy ty nejdůležitější.

Která z těchto entit bude mít nejvyšší objem?

Která entita je nejdůležitější?

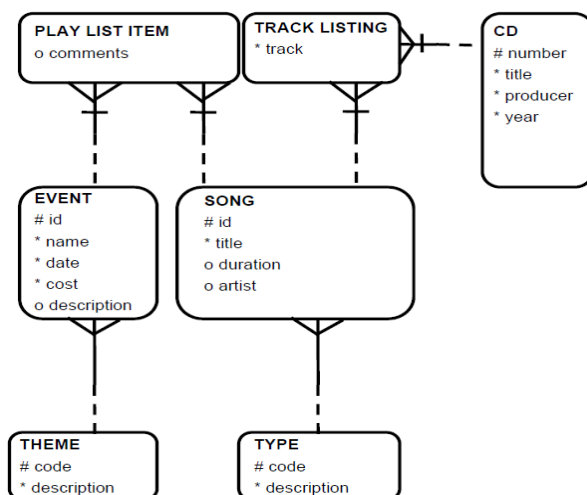
Opět platí, že jasnost a čitelnost jsou hlavní kritéria.

Čitelnost zabírá prostor a podléhá vkusu. Použití bílého prostoru pomáhá vyjasnit ERD.

Když máte velmi velký diagram, může pomoci rozdělit jej na menší diagramy funkčně souvisejících entit. Můžete použít menší subdiagramy pro prezentaci pro různé skupiny v rámci organizace zákazníka.

Často se stává, že více vývojářů vytváří aplikace, které používají stejnou databázi. Každý aplikační vývojář by mohl použít menší diagram, který obsahuje ty entity, pro které bude vytvářet obrazy, formuláře a sestavy.

Toto jsou entity, které budou nejvíce zajímat DJs nebo někoho, kdo pro ně vytváří aplikaci.



# 9. ODDÍL (OA S11)

## Obsah oddílu

- Úvod do relačních databází
- Základní mapování: Proces transformace ERD do RMD
- Mapování vztahů

## Úvod do relačních databází

LEKCE 01

DD\_S11\_01

V této lekci se naučíte:

- definovat primární klíč
- definovat cizí klíč
- definovat pravidlo sloupčové integrity
- definovat řádek, sloupec, primární klíč, unikátní klíč a cizí klíč podle diagramu tabulky obsahující tyto elementy
- identifikovat porušení pravidel integrity dat


Proč se to učit?

- Konceptuální datový model bude transformován do relační databáze. To znamená, že naše entity, atributy, vztahy a unikátní identifikátory budou převedeny do objektů v relační databázi.
- Proto je nutné pochopit strukturu těchto objektů.

**Relační databáze** je databáze, kterou uživatel vnímá jako soubor dvourozměrných tabulek. Níže uvedená tabulka obsahuje údaje zaměstnanců.

**EMPLOYEES (table name)**

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
100	Steven	King	90
101	Neena	Kochhar	90
102	Lex	De Haan	90
200	Jennifer	Whalen	10
205	Shelley	Higgins	110



Strukturovaný dotazovací jazyk nám umožňuje, abychom se efektivním způsobem dostali k datům v relačních databázích. Místo abychom procházeli každý řádek a hledali záznam o zaměstnanci 200, použijeme následující příkaz

```
SELECT last_name, department_id
FROM employees
WHERE employee_id = 200;
```

Výsledek příkazu:

EMPLOYEES (table name)

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
100	Steven	King	90
101	Neena	Kochhar	90
102	Lex	De Haan	90
200	Jennifer	Whalen	10
205	Shelley	Higgins	110

```
SELECT last_name, department_id
FROM employees
WHERE employee_id = 200;
```

LAST_NAME	DEPARTMENT_ID
Whalen	10

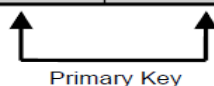
## Primární klíč

Primární klíč (dále PK) je buď sloupec, nebo množina sloupců, které jednoznačně identifikují každý řádek v tabulce.

Každá tabulka by měla mít primární klíč a ten musí být jedinečný. Žádná část primárního klíče nesmí být prázdná (null).

ACCOUNTS

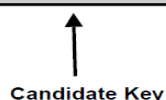
BANK_NO	ACCT_NO	BALANCE	DATE_OPENED
104	75760	12,0050.00	21-OCT-89
104	77956	100.10	
105	89570	55,775.00	15-JAN-85
103	55890	15,001.85	10-MAR-91
105	75760	5.00	22-SEP-03



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	...	DEPARTMENT_ID
100	Steven	King	...	90

MEMBERS

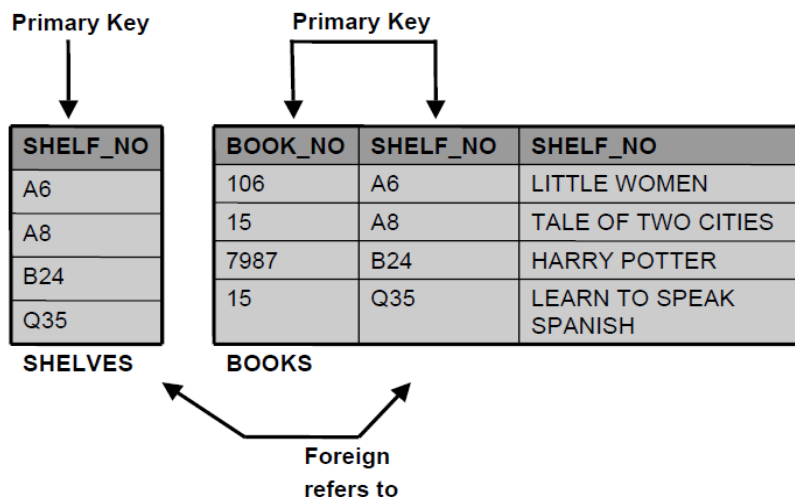
MEMBER_ID	LAST_NAME	FIRST_NAME	PAYROLL_ID
100	SMITH	DANA	21215
310	ADAMS	TYLER	59877
210	CHEN	LAWRENCE	1101
405	GOMEZ	CARLOS	52
378	LOUNGANI	NEIL	90386



Může mít více než jeden sloupec nebo kombinaci sloupců, které by mohly být použity jako primární klíč tabulky. Každý z nich se nazývá „kandidátní klíč“.

## Cizí klíč

Cizí klíč je buď sloupec, nebo množina sloupců v jedné tabulce, které se vztahují k primárnímu klíči téže nebo jiné tabulky.



Pokud je cizí klíč součástí primárního klíče, nesmí být prázdný.

## Sloupcová integrita

Sloupec musí obsahovat pouze konzistentní hodnoty s definovaným formátem sloupce

**ACCOUNTS**

BANK_NO	ACCT_NO	BALANCE	DATE_OPENED
104	75760	12,0050.00	21-OCT-89
104	77956	100.10	
105	89570	55,775.00	15-JAN-85
103	55890	15,001.85	10-MAR-91
105	75760	5.00	22-SEP-03

**ACCOUNTS Table Definition**

Column Name	Data Type	Optionality
BANK_NO	Number (5)	Not null
ACCT_NO	Number (8)	Not null
BALANCE	Number (12,2)	Not null
DATE_OPENED	Date	

## Shrnutí pravidel datové integrity:

Pravidla datové integrity, taktéž známá jako omezení (constraints), definují relačně korektní stav databáze.

Pravidla datové integrity zajišťují, že uživatel může provádět pouze ty operace, které zachovávají databázi v konzistentním stavu.

Typ omezení	Vysvětlení	Příklad
<b>Integrita entit</b>	Primární klíč musí být jedinečný a žádná část primárního klíče nemůže být prázdná (null)	V tabulce EMPLOYEES nemůže být sloupec emp_no prázdný.
<b>Referenční integrita</b>	Cizí klíč musí patřit k hodnotě existujícího primárního klíče (nebo musí být prázdný).	Hodnota ve sloupci dept_no tabulky EMPLOYEES musí odpovídat k hodnotě sloupce dept_no v tabulce DEPARTMENTS.
<b>Sloupcová integrita</b>	Sloupec musí obsahovat pouze hodnoty v souladu s definovaným formátem dat ve sloupci.	Hodnota ve sloupci zůstatek tabulky ACCOUNT musí být číslo.
<b>Uživatелеm definovaná integrita</b>	Údaje uložené v databázi musí být v souladu s pravidly zadání.	Pokud je hodnota ve sloupci zůstatek tabulky ACCOUNT nižší než 1,00, musíme poslat dopis vlastníkovu účtu (bude potřeba pravidlo zvlášť naprogramovat).

## Základní mapování: Proces transformace ERD do RMD

LEKCE 02

DD\_S11\_L02

V této lekci se naučíte:

- rozlišovat modely typu E-R (entita-vztah) od databázových modelů
- popsat mapování terminologie mezi konceptuálním modelem a modelem relační databáze
- pochopit a aplikovat pojmenovávací konvence Oracle pro tabulky a sloupce použitých v relačních modelech
- transformovat entitu do tabulkového diagramu

### Proč se to učit?

- Když navrhujete dům, rádi byste jej nakonec viděli postavený. Dokonce i když vy sám nestavíte, budete muset pochopit terminologii používanou stavaři, abyste jim mohli pomoci převést váš návrh do reality.
- Původní návrh databáze lze použít pro další diskusi mezi designéry, správci databází a vývojáři aplikací.

### Opakování relačních tabulek:

Table: **EMPLOYEES**

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	DEPARTMENT_ID	PAYROLL_ID	NICKNAME
100	SMITH	DANA	10	21215	Dana
310	ADAMS	TYLER	15	59877	Ty
210	CHEN	LAWRENCE	10	1101	Larry
405	GOMEZ	CARLOS	10	52	Chaz
378	LOUNGANI	NEIL	22	90386	Neil

↑ Primary Key Column (PK)

↑ Foreign Key Column (FK)

↑ Unique Key Column (UK)

Tabulka je jednoduchá struktura, ve které jsou organizována a ukládána data. Ve výše uvedeném příkladu se tabulka ZAMĚTNANCI (EMPLOYEES) používá pro ukládání dat o zaměstnancích.

Tabulky mají sloupce a řádky. V příkladu každý řádek popisuje zaměstnance. Každý sloupec slouží k ukládání specifických hodnot, jako je číslo zaměstnance, příjmení, jméno.

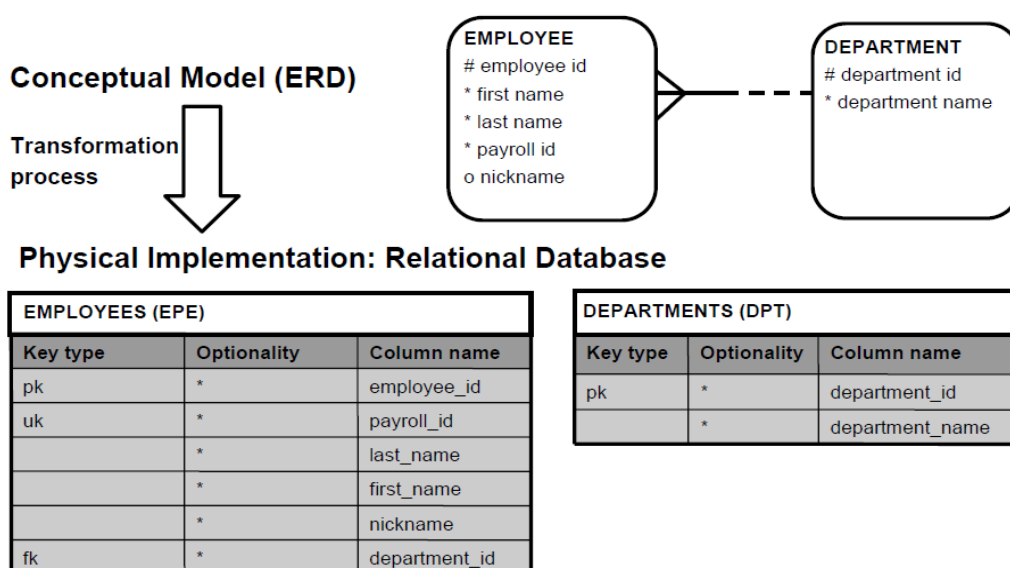
Sloupec employee\_id je primární klíč. Každý zaměstnanec má v této tabulce jednoznačné identifikační číslo. Hodnota ve sloupci primárního klíče od sebe odlišuje jednotlivé řádky.

Payroll\_id je jednoznačný (unikátní) klíč. To znamená, že systém nedovoluje dvěma řádkům mít stejné payroll\_id.

Sloupec cizího klíče odkazuje na řádek v jiné tabulce. V příkladu číslo oddělení odkazuje na řádek v tabulce oddělení.

V tomto případě víme, že Dana Smith pracuje v oddělení 10. Kdybychom chtěli zjistit více o oddělení Dany Smith, podívali bychom se v tabulce oddělení na řádek, který má číslo oddělení 10.

Konceptuální model ERD je transformován do fyzického modelu. Fyzická implementace bude reálná databáze.

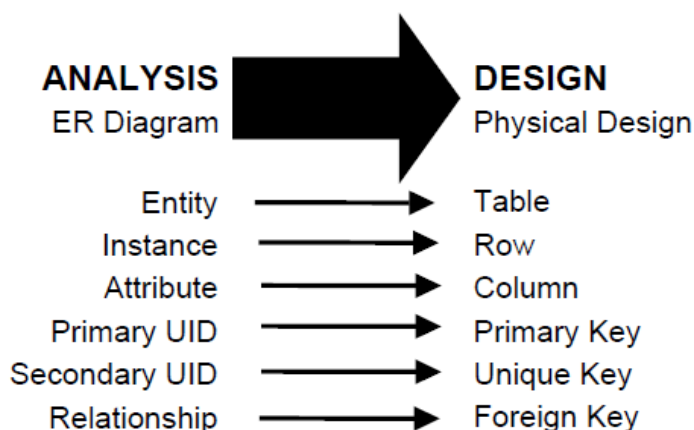


## Terminologie mapování

Přechod od analýzy (konceptuální model) k návrhu (fyzická realizace) také znamená změnu terminologie:

- entita se stává tabulkou,
- instance se stává řádkem,
- atribut se stává sloupcem,
- primární jedinečný identifikátor se stává primárním klíčem,
- sekundární jedinečný identifikátor se stává unikátním klíčem,
- vztah se transformuje do sloupce s cizím klíčem a omezení cizího klíče

## TERMINOLOGY MAPPING



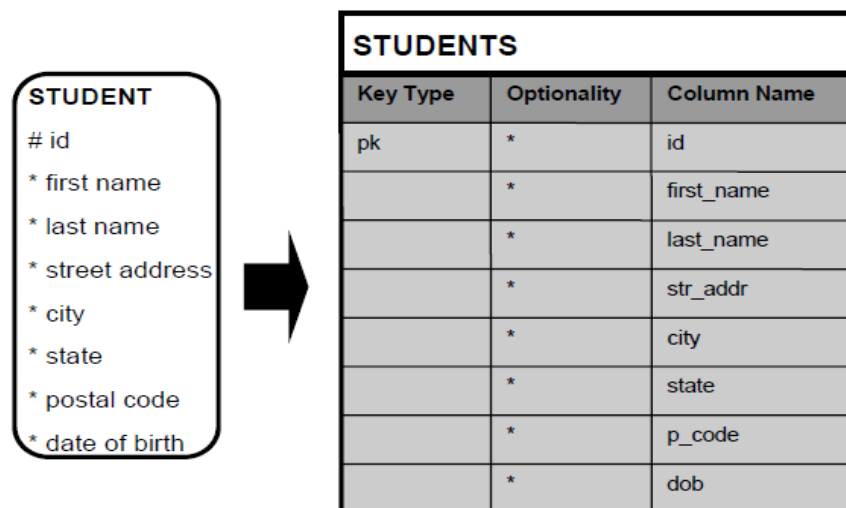


## Notace tabulky

Sloupec "typ klíče" by měl obsahovat hodnoty "pk" pro primární klíč, "uk" pro unikátní klíč a "fk" pro cizí klíč. Pokud tento sloupec není součástí žádného klíče, bude prázdný.

Volitelnost musí obsahovat „\*“, pokud jsou sloupce povinné, a „o“, pokud jsou nepovinné. Je to stejné jako v diagramu ERD.

Třetí sloupec je určen pro jméno sloupce.



## Pravidla pro jména tabulek a sloupců

Název tabulky je množné číslo entity; příklad: Student se stává Studenti. Názvy sloupců jsou shodné s názvy atributů, kromě toho, že speciální znaky a mezery jsou nahrazeny podtržítka. Názvy sloupců bývají častěji zkratky než názvy atributů.

### ■ PŘÍKLAD:

First name (křestní jméno) se stává first\_name nebo fname.

### Konvence pro jména: krátká jména:

Unikátní krátké jméno pro každou tabulku je užitečné pro pojmenovávání sloupců s cizím klíčem.

## Zkratky tvoříme podle následujících pravidel:

U názvů entity, skládající se z více než jednoho slova, vezměte:

- první znak prvního slova,
- první znak druhého slova,
- poslední znak posledního slova.

### ■ PŘÍKLAD

Pracovní pozice (JOB ASSIGNMENT) má zkratku JAT.

U názvů entity, která se skládá z víceslabičného slova, vezměte:

- první znak první slabiky,
- první znak druhé slabiky,
- poslední znak poslední slabiky.

### ■ PŘÍKLAD

Zaměstnanec (EMPLOYEE) – EPM.

U jednoslabičných názvů entit s více než jedním znakem použijte :

- první znak,
- druhý znak,
- poslední znak.

## Omezení v pojmenovávání v ORACLE

Názvy tabulek a sloupců:

- musí začínat písmenem,
- mohou obsahovat až 30 alfanumerických znaků,
- nemohou obsahovat mezery nebo speciální znaky jako „!“, ale \$, # a „\_“ jsou dovoleny
- názvy tabulek musí být unikátní v rámci jednoho uživatelského účtu v databázi ORACLE
- názvy sloupců musí být unikátní v rámci tabulky.

V ORACLE databázi a v jazyce SQL mají některá slova speciální význam. Říká se jim „rezervovaná“ (klíčová). Nepoužívejte tato slova pro názvy tabulek a sloupců. Zde jsou uvedeny některé příklady rezervovaných slov ORACLE:

- NUMBER,
- SEQUENCE,
- ORDER,
- VALUES,
- LEVEL,
- TYPE.

Kompletní seznam najdete na Technet. ([otn.oracle.com](http://otn.oracle.com))

# Mapování vztahů:

LEKCE 03

dd\_S11\_L03

V této lekci se naučíte:

- aplikovat pravidlo mapování vztahů na správnou transformaci 1:M a „blokovaných („barred“) vztahů
- aplikovat pravidlo mapování vztahů na správnou transformaci M:M
- transformovat vztahy 1:1

Proč se to učit?

- Co kdybyste někomu stavěli dům? Koupíte veškerý materiál – dřevo, barvu, dveře, hřebíky ... . Ale nevíte, jak poskládat jednotlivé části. Nevíte, kolik tam má být pokojů, kde by měla být okna, jak mají být orientovány dveře a jakou barvou vymalovat. Nějak byste dům postavili, ale pokud nemáte plán, který popisuje, jak jednotlivé části složit dohromady, finální produkt nemusí být tím domem, který měl zákazník na mysli.
- Vztahy jsou mapovány do cizích klíčů, které tabulkám umožňují se vzájemně provázat. Cizí klíče umožňují uživateli přístup k potřebným informacím v jiných tabulkách. Pokud nebudeme mapovat vztahy, budeme mít spoustu samostatných tabulek s informacemi, které nemůže využívat zbytek databáze.
- Stejně jako kopie slouží jako návrh domu. Mapování relací do struktur relační databáze je částí tvorby „first-cut“ návrhu databáze, který bude sloužit jako základ pro následnou diskuzi mezi analytiky, vývojáři a databázovými administrátory.

## Pravidla vztahů:

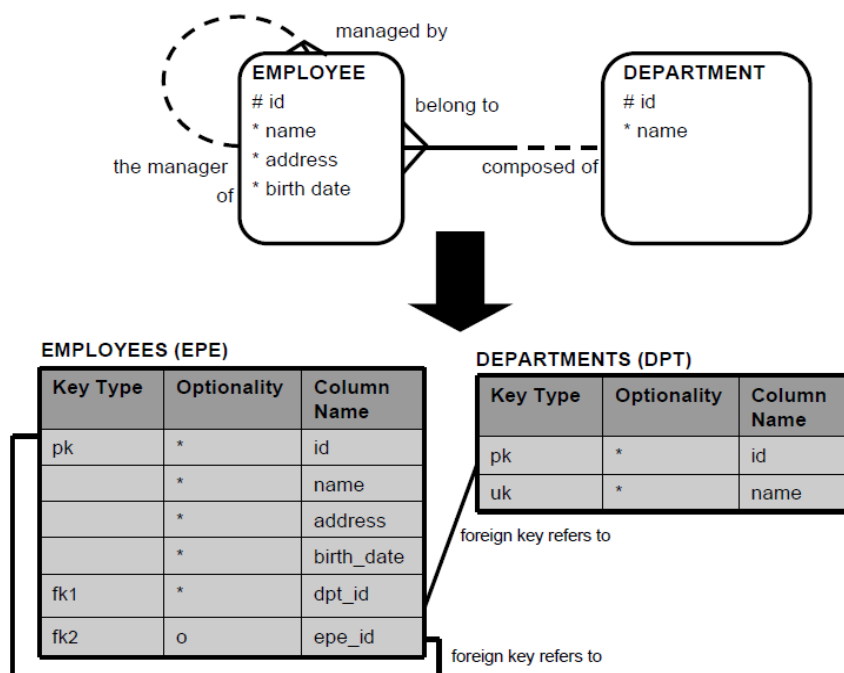
**Vztahy vytvářejí jeden nebo více cizích klíčů v tabulce.**

Do názvu cizího klíče používáme zkratku tabulky. Například sloupec cizího klíče v tabulce zaměstnanci je `dpt_id` pro vztah s oddělením a `epe_id` pro rekurzivní vazbu.

Cizí klíč je povinný nebo nepovinný podle toho, zda je vztah povinný nebo nepovinný.

## PŘÍKLAD

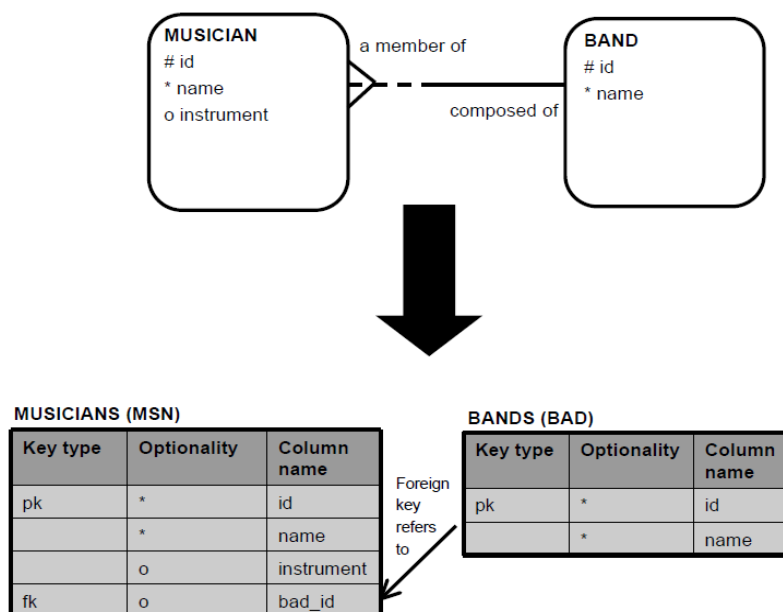
Vztahy mezi entitami ZAMĚSTNANEC a ODDĚLENÍ (v příkladu je dpt\_id povinný a epe\_id volitelný).



## Mapování povinného vztahu na jedné straně

Vztahy, které jsou povinné na jedné straně nebo na obou stranách, jsou mapovány úplně stejně jako vztah, který je volitelný na jedné straně. ERD je dostatečně bohatý na to, aby zachytil členství ve vztahu na obou koncích vztahu. Nicméně relační model je omezen v tom, že omezení cizího klíče vynucuje povinný vztah pouze pro konec „many“.

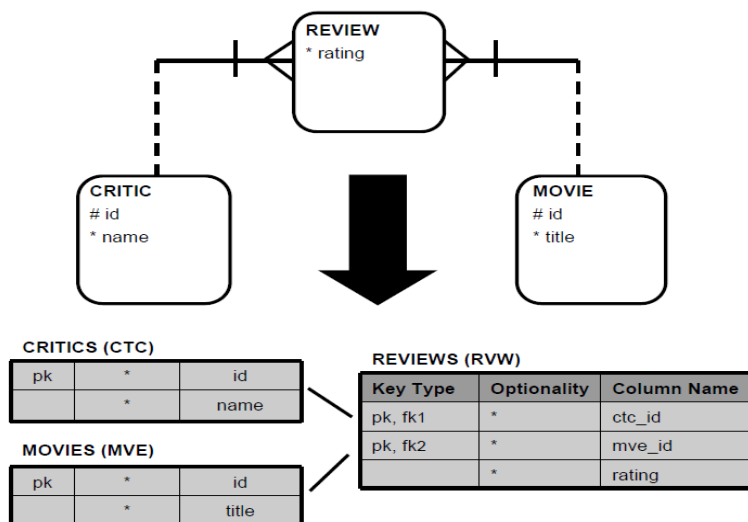
V následujícím příkladu si relační model nemůže vynutit, aby BAND (Skupina) musela obsahovat nejméně jednoho hudebníka. Volitelnost na jednom konci musí být implementována pomocí dalšího programu.



## Mapování M:N vztahu

M:N vztah se řeší pomocí průnikové entity, která se mapuje do průnikové tabulky. Tato tabulka bude obsahovat sloupce cizí klíče, které se budou odkazovat na původní tabulky.

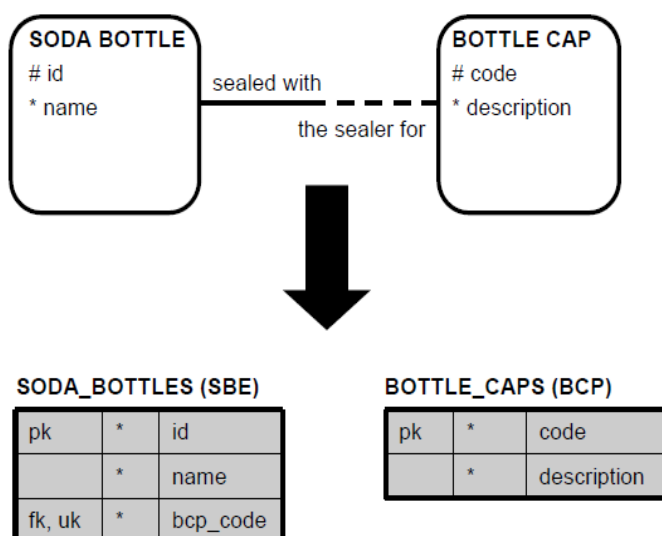
V následujícím příkladu REVIEWS obsahují všechny kombinace, které existují mezi kritikou a filmem.



## Mapování 1:1 vztahu

Při transformaci vztahu 1:1 tvoříte cizí a unikátní klíč. Všechny sloupce cizího klíče jsou také součástí unikátního klíče.

Pokud je vztah povinný z jedné strany, cizí klíč se vytvoří v odpovídající tabulce. Např. bcp\_code je sloupec cizího klíče SODA\_BOTTLES (nealkoholické nápoje), který odkazuje na primární klíč tabulky BOTTLE\_CAPS (víčka na nápoje). Bcp\_code by byl také unikátním klíčem v rámci tabulky SODA\_BOTTLES.



## Mapování vztahu 1:1 – nepovinný z obou stran

Jestliže je vztah nepovinný z obou stran, můžete si vybrat, která tabulka získá cizí klíč. Zde neexistují žádná absolutní pravidla, ale nabízíme následující návod:

- implementujte cizí klíč do tabulky s méně řádky, abyste ušetřili prostor,
- implementujte cizí klíč, kde je to v rámci zadání nejlogičtější.

V příkladu: Autopůjčovna by se více zajímala o auta než o parkovací místo, takže je logičtější vložit cizí klíč do tabulky CARS. Nicméně u firmy, provozující parkoviště, je hlavním objektem parkovací místo. Proto by bylo smysluplnější dát cizí klíč do tabulky SPACES.

## Mapování 1:1 – povinné na obou stranách

Jestliže je vztah povinný na obou koncích, existují v databázi stejné pravidla, jako vztah M:1, který je povinný z jedné strany. A proto budeme potřebovat další kód pro realizaci tohoto vztahu.

## PROGRAMOVÁNÍ V SQL

Podpora výuky databázových systémů na SOŠ, založené na technologiích společnosti ORACLE.

Publikace vznikla v rámci projektu CZ.1.07/1.1.07/02.007,  
Podpora výuky databázových systémů na středních odborných  
školách, založené na technologiích společnosti ORACLE.

© 2011 Vyдала Střední průmyslová škola elektrotechniky  
informatiky a řemesel, příspěvková organizace, Křižíkova 1258,  
Frenštát p. R., [www.spsfren.cz](http://www.spsfren.cz)

Studijní kapitoly jsou synchronizovány s mezinárodním vzdělávacím  
programem Oracle Academy. Více informací na [academy.oracle.com](http://academy.oracle.com)  
nebo na portálu [ucimedatabase.cz](http://ucimedatabase.cz).

**Manager projektu:** Mgr. Richard Štěpán

**Překlad:** Oracle Czech,

Bc. Tomáš Romanovský, Mgr. Markéta Kytková

**Metodik:** Bc. Tomáš Romanovský

**Jazyková korektura:** Mgr. Pavlína Chovancová

**Sazba:** Bc. Tomáš Romanovský

**Obálka:** Bc. Tomáš Romanovský

**Tisk:** Reprografické studio LWR GRAPHIC

Žádná část této publikace nesmí být publikována a šířena žádným způsobem a v žádné podobě  
bez výslovného souhlasu vydavatele

*Zvláštní poděkování patří společnosti Oracle Czech za  
dlouholetou podporu vzdělávání v oblasti databázových  
technologií a za spolupráci při vytváření této publikace.*

*Autoři projektu*

## Obsah

1. ODDÍL.....	3
SQL (Structured Query Language).....	3
2. ODDÍL.....	5
Vytvoření tabulek.....	5
Integritní omezení.....	9
Modifikace tabulky.....	12
3. ODDÍL.....	13
Anatomie příkazu SQL .....	13
Práce se sloupci, znaky, a záznamy (řádky) .....	17
Omezení výběru záznamů (Selection).....	19
Třídění řádků dotazu.....	22
4. ODDÍL.....	23
Manipulace se znaky.....	23
Číselné funkce.....	27
Datumové funkce.....	29
Konverzní funkce.....	32
Funkce NULL.....	36
Podmíněné výrazy.....	38
5. ODDÍL.....	40
Křížové a přirozené spojení.....	40
Klauzule JOIN.....	42
Vnitřní versus vnější spojení (inner join - outer join).....	44
6. ODDÍL.....	46
Skupinové funkce (agregační).....	46
Použití klauzulí GROUP BY a HAVING.....	48
7. ODDÍL.....	51
Základy vnořených dotazů (poddotazů).....	51
8. ODDÍL.....	54
Příkazy DML.....	54
Ostatní databázové objekty.....	56



# 1. ODDÍL

## Obsah oddílu

- Úvod k SQL
- Základní rozdělení příkazů
- Anotace syntaxe jazyka

## SQL (Structured Query Language)

### Úvod

Jazyk SQL (Structured Query Language - strukturovaný dotazovací jazyk) je v případě SQL databázových serverů rozhraním, které slouží ke zpřístupnění dat. Jazyk SQL byl vyvinut firmou IBM na počátku 70. Let jako dotazovací jazyk pro práci s velkými databázemi na počítačích střediskového typu. Cílem tvůrců SQL bylo vyvinout takový nástroj pro koncové uživatele, který by jim umožnil vybírat data z databáze přesně podle jejich individuálních požadavků a byl přitom co nej-jednodušší. První část záměru se podařilo realizovat celkem úspěšně. SQL je mimořádně silný dotazovací jazyk. Kromě dotazování můžeme s jeho pomocí definovat data, provádět aktualizace atd. Dnes už je zřejmé, že původní záměr autorů SQL nebyl reálný. Ukázalo se, že SQL je pro koncové uživatele-neprogramátory příliš složitý. Přesto se SQL prosadil a dnes představuje jeden z pevných standardů. Jeho výhodou je neprocedurálnost - programátor popisným způsobem definuje, co se má s jakými daty provést bez toho, že by musel specifikovat algoritmus vedoucí ke zpřístupnění dat, případně pro vlastní manipulaci s daty.

Jazyk SQL prošel mnoha změnami a úpravami a v současné době je součástí mnoha významných databázových systémů - ORACLE, INFORMIX, dBASE, INGRES, ACCESS atd. Jeho význam stále roste.

**S jazykem SQL můžeme pracovat dvěma způsoby:**

- interaktivní zadávání příkazů z terminálu
- komunikace aplikačního programu s databází

## Základní rozdělení příkazů SQL

### 1. DDL (Data Definition Language):

- vytváření (CREATE) databázových objektů
- změnu definice (ALTER) databázových objektů
- mazání (DROP) databázových objektů

### 2. DML (Data Manipulation Language):

- výběr záznamů (SELECT)
- vkládání záznamů (INSERT)
- mazání záznamů (DELETE)
- modifikace záznamů (UPDATE)

### 3. Ostatní příkazy a funkce

V následujících kapitolách probereme základní vlastnosti jazyka SQL postupně podle potřeby a obtížnosti. Pro vysvětlení příkazů budou uvedeny příklady, týkající se zpracování dat v databázi obchodní firmy, společnosti pro zajištění hudby na různých akcích a knihovně.

## Základní anotace SQL jazyka

Každý příkaz je ukončen středníkem (;).

### Použitá symbolika (obecná syntaxe):

- [ ] ... označení volitelnosti možností,
- | ... buď a nebo,
- { } ... označení povinnosti vybrat jednu z uvedených možností,
- ... ... libovolný počet opakování,

### ■ PŘÍKLAD SYNTAXE

```
SELECT*|{[DISTINCT] column | expression alias}..  
FROM table  
[WHERE condition(s)];
```

## 2. ODDÍL

### Obsah oddílu

- Vytvoření a zrušení tabulek
- Integritní omezení tabulek
- Modifikace tabulek

## Vytvoření tabulek

Lekce 01

dp\_S08\_I01

### Co se v této lekci naučíte?

- vyjmenovat a určit kategorii hlavních databázových objektů
- prozkoumat strukturu tabulky
- popsat schema objektů jak je použito v Oracle databázi
- vyjmenovat a poskytnout příklad každého datového typu - čísla, znaku a data
- vytvořit tabulku s použitím vhodného typu dat pro každý sloupec
- vložit do tabulky řádek
- zrušit tabulku

### Proč se to učit?

- V této lekci se seznámíte s nejčastěji používanými databázovými objekty, jak se orientovat ve struktuře tabulky a jak vytvořit nové tabulky. Vaše tabulky budou malé ve srovnání s tabulkami, které obsahují miliony záznamů (řádků) a stovky sloupců, ale vytvoření malé tabulky představuje stejný SQL příkaz a syntaxi, jako vytvoření velmi rozsáhlé tabulky.

## Objekty databáze

Oracle databáze může obsahovat mnoho různých typů objektů. V této části budou představeny nejčastěji používané objekty DB, a také to, jak Oracle server používá informace o těchto objektech, uložené v datovém slovníku, při řešení různých problémů v jazyce SQL.

**Hlavní typy databázových objektů jsou:**

- Tabulka
- Index
- Omezení - Constraint
- Pohled
- Sekvence
- Synonymum

Některé z těchto typů objektů mohou existovat nezávisle a jiné nemohou.

Některé z objektových typů obsazují prostor v databázi, jemuž říkáme Sklad a jiné objekty ne. Databázové objekty zabírající Sklad jsou známe jako Segmenty. Tabulky a indexy jsou příklady Segmentů - záznamy uložené v tabulce a hodnoty sloupců zabírají fyzický prostor na disku v databázi.

Pohledy, omezení, Sekvence a synonyma jsou jiné objekty. Jediný prostor, který zabírají v databázi, je definice těchto objektů; žádné z těchto objektů nemají vázány datové záznamy.

Databáze ukládá definice všech databázových objektů v Datovém Slovníku, a tyto definice jsou přístupné všem uživatelům databáze stejně jako databáze sama.

## Vytvoření tabulky

Všechna data v relační databázi jsou uložena v tabulkách (viz. RMD ve skriptech Databázový návrh). Tabulka je základním stavebním kamenem každé relační databáze. Sloupce v tabulce se nazývají **pole (fields)**, řádky se nazývají **záznamy (records)**.

The diagram illustrates a table structure. On the left, a box labeled 'řádek  
věta  
záznam  
record' has an arrow pointing to the first column of a table. Below the table, a box labeled 'sloupec  
pole  
field' has an arrow pointing to the first row of the table. The table itself has columns labeled PRUKAZKA, JMENO, PRIJMENI, ROD\_CIS, and an ellipsis. The rows are numbered 1, 2, and an ellipsis. A callout box points to the cell containing '810315/4563' with the text 'Přizpůsobit řádek tabulky'.

PRUKAZKA	JMENO	PRIJMENI	ROD_CIS	...
1	Jan	Adam	821016/7657	...
2	Petr	Barta	810315/4563	...
...	...	...	...	...

*Tab.1 - příklad tabulky (CTENARI)*

## ZÁSADY PRO JMÉNO

Při vytváření nové tabulky používejte následující pravidla pro jméno tabulky i jména sloupců:

- musí začínat písmenem
- musí být dlouhé 1 až 30 znaků
- musí obsahovat jen A - Z, a - z, 0 - 9, \_ (podtržítka), \$, a #
- nesmí být kopií jména dalšího objektu vlastněného stejným uživatelem
- nesmí být klíčové (vyhrazené) slovo používané Oracle serverem

Pro jméno tabulky je nejlépe použít (a jiné databázové objekty) popisné jméno. Jestliže jsou například v tabulce uloženy informace o studentech, pak by se tabulka měla jmenovat STUDENTI (STUDENTS) a nikoliv LIDE nebo DETI.

Jména nejsou citlivá na velikost písma. Například STUDENTS je stejné jako STuDents nebo studenti.

Vytváření tabulky je součástí SQL jazyka - příkazů pro definici dat (DDL). Jiné DDL příkazy, které nastavují, mění a odstraňují datové struktury tabulky jsou ALTER, DROP, RENAME, a TRUNCATE.

## PRO VYTVOŘENÍ NOVÉ TABULKY JE NUTNÉ:

- Systémové právo CREATE TABLE a přidělený pracovní prostor pro ukládání (tablespace). Správce databáze používá příkazy DCL k tomu, aby udělila toto privilegium uživatelům a přiřadit skladovací prostor.
- Tabulky, patřící jiným uživatelům, nejsou v našem schématu. Jestli chcete vytvořit tabulku, která nebude ve vašem schématu, použijte vlastnické jméno (uživatelské jméno) jako předponu ke jménu tabulky:  
**mary.students;**

## PRO KAŽDÉ POLE (FIELD) MUSÍME URČIT:

- **Jméno pole** - musí vyjadřovat obsah jednotlivých položek. Je doporučeno nepoužívat české znaky a v názvu se nesmí použít mezery; viz. pravidla pro jména výše.
- **Datový typ (Doména)** - určuje hodnoty, které jsou pro daný sloupec přípustné. Všechny typy, které můžete v Oracle použít jsou uvedeny v následující tabulce.
- **Integritní omezení (IO)** – pravidla, která musí splňovat data jednotlivých relací (tabulek). Tato pravidla jsou podrobně rozebrána v následující kapitole.

Datový typ	Parametry	Popis
<b>CHAR(d)</b>	d=1 až 2 000	Řetězec znaků s pevnou délkou. Implicitní délka je 1 znak. Při definici sloupce určete maximální délku (d)
<b>VARCHAR(d)</b>	d=1 až 4 000	Řetězec znaků proměnné délky. Při definici sloupce musíte určit maximální délku parametrem (d). Toto je zastaralý datový typ poskytovaný pouze pro podporu starších databází Oracle.
<b>VARCHAR2(d)</b> <sup>1</sup>	d=1 až 4 000	Řetězec znaků proměnné délky. Při definici sloupce musíte určit maximální délku parametrem (d).
<b>LONG</b>	nejsou	Řetězec znaků s proměnnou délkou. Maximální délka je 2 GB => LONG je vhodný pro velké množství dat.
<b>DATE</b>	nejsou	Datum v rozsahu 1.1.4712 př.n.l. až 31.12.4712 n.l. Systém Oracle 8 ukládá tento datový typ do sedmibajtového čísla, které také obsahuje čas v hodinách, minutách a sekundách.
<b>NUMBER(p,d)</b>	p=1 až 38, d= -84 až 127	Číslo. Přesnost je udána parametrem (p) v počtech číslic. Parametr (d) udává počet desetinných míst.
<b>FLOAT(p)</b>	p=1 až 126	Reálné číslo. Parametr (p) určuje přesnost počtem číslic.
<b>RAW(p)</b>	p=1 až 2 000	Binární data proměnné délky. Maximální délku musíte určit parametrem (p)
<b>LONG RAW</b>	nejsou	Binární data s proměnnou délkou. Maximální délka je 2 GB.
<b>BFILE</b>	nejsou	Rozsáhlý binární objekt (LOB) uložený mimo databázi s maximální velikostí 4 GB.
<b>BLOB</b> <b>CLOB</b> <b>NCLOB</b>	nejsou	LOB s maximální velikostí 4 GB.

1 Při ukládání dat typu VARCHAR2 ukládá Oracle pouze znaky. Na rozdíl od toho data typu CHAR jsou při ukládání zarovnávána na maximální možnou délku pomocí mezer a uložena i s těmito přebytnými mezery. U řetězců proměnné délky je tedy efektivnější použít datový typ VARCHAR2.

## VYTVOŘENÍ TABULKY (PŘÍKAZ DDL)

```
CREATE TABLE <jméno tabulky>
(<jméno sloupce>          <datový typ>, [NOT NULL] [UNIQUE]
[<jméno sloupce>          <datový typ>, [NOT NULL] [UNIQUE] ...]);
```

Vysvětlivky: NOT NULL ... sloupec nesmí obsahovat hodnotu NULL (prázdný)  
 UNIQUE ... sloupec je unikátní (hodnota se ve sloupci nesmí opakovat)

### PŘÍKLAD:

Vytvořte tabulku CTENARI, která bude obsahovat sloupce PRUKAZKA, JMENO, PRIJMENI, ROD\_CIS, DAT\_NAR, MESTO, ULICE.

```
CREATE TABLE CTENARI
(PRUKAZKA NUMBER NOT NULL,
JMENO VARCHAR2(20),
PRIJMENI VARCHAR2(35),
ROD_CIS CHAR(11) NOT NULL,
DAT_NAR DATE,
MESTO VARCHAR2(40),
ULICE VARCHAR2(30));
```

## VKLÁDÁNÍ ŘÁDKŮ DO TABULKY (PŘÍKAZ DML)

```
INSERT INTO <jméno tabulky> [seznam sloupců] VALUES (<seznam hodnot>);
```

Seznam hodnot musí být vytvořen tak, aby jeho hodnoty pořadím odpovídaly prvkům tabulky. Hodnoty se oddělují čárkou.

### PŘÍKLAD:

Vložte do tabulky CTENARI jeden celý záznam.

```
INSERT INTO CTENARI
VALUES (1, 'Jan', 'Novák', '540713/3422', '13.07.1954', 'Nový Jičín',
'K nemocnici 23');
```

## Odstranění tabulky:

```
DROP TABLE <jméno tabulky>
```

### PŘÍKLAD:

Odstraňte tabulku CTENARI

```
DROP TABLE CTENARI;
```

# Integritní omezení

## Integrita domén

Integrita domén znamená, že každá hodnota sloupce je prvkem domény sloupce. Zajišťuje se pomocí datových typů.

Doménu povolených hodnot sloupce lze zúžit pomocí integritního omezení **NOT NULL** (nepovolí prázdnou hodnotu). Dále je možné definovat tuto doménu pomocí výčtu povolených hodnot pomocí klíčového slova **CHECK**.

## Integrita entit

Integrita entit znamená, že každý řádek musí být jednoznačný. Zajistí se označením sloupce nebo množiny sloupců jako primární klíč- klauzule **PRIMARY KEY**. Každá hodnota primárního klíče musí být jednoznačná. Zamezení duplicitě i v jiných sloupcích se provede pomocí klauzule **UNIQUE**.

## REFERENČNÍ INTEGRITA

Referenční integrita definuje vztahy mezi různými sloupci různých tabulek relační databáze. Splnění podmínek referenční integrity lze zajistit pomocí definice cizího klíče, který se deklaruje pomocí klíčových slov **FOREIGN KEY** a **REFERENCES**. Každá hodnota cizího klíče musí odpovídat hodnotě rodičovského klíče. Pokud se rodičovský i cizí klíč nachází ve stejné tabulce, jedná se o tzv. sebe odkazující se integritu.

### Referenční akce při práci s cizím klíčem:

- Referenční akce kaskádovité rušení  
**FOREIGN KEY ON DELETE CASCADE**  
při rušení záznamu v rodičovské tabulce provádí i rušení všech závislých synovských záznamů
- Referenční akce omezení  
**FOREIGN KEY**  
zabrání všem modifikacím rodičovského klíče, který má závislé synovské záznamy (nedovolí aktualizaci nebo zrušení tohoto záznamu)

## PŘÍKLAD NA VYTVOŘENÍ TABULEK PRO SYSTÉM KNIHOVNY

```
CREATE TABLE kniha
(
    isbn VARCHAR2(16) PRIMARY KEY,
    nazev VARCHAR2(32) UNIQUE,
    autor VARCHAR2(24) NOT NULL,
    cena NUMBER(6,0),
    zeme_vydani CHAR(2) DEFAULT 'CZ');

CREATE TABLE exemplar
(
    id NUMBER(6,0) PRIMARY KEY,
    isbn VARCHAR2(16) REFERENCES kniha,
    dat_nakupu DATE DEFAULT SYSDATE,
    vypujceno CHAR(1) CHECK (vypujceno IN ('A', 'N')));
```

```

CREATE TABLE ctenar
(
    id NUMBER(6,0) PRIMARY KEY,
    jmeno VARCHAR2(10) NOT NULL,
    prijmeni VARCHAR2(15) NOT NULL,
    ulice VARCHAR2(20),
    mesto VARCHAR2(15),
    psc CHAR(5),
    rod_cis VARCHAR2(10),
    telefon VARCHAR2(15),
    vzdelani CHAR(1) CHECK (vzdelani IN ('Z','S','V'));
);

CREATE TABLE vypujcka
(
    exemplarid NUMBER(6,0) REFERENCES exemplar,
    ctenarid NUMBER(6,0) REFERENCES ctenar,
    pujceno DATE DEFAULT SYSDATE,
    vraceno DATE);

```

## Specifikace integritních omezení

Integritní omezení je možné vytvářet:

- uvnitř (column constraints)
- vně specifikace sloupce (table constraints)

Ve většině případů je možné použít kteroukoli možnost. Existují dvě výjimky NOT NULL je nutno definovat jako "column constraints" a pokud IO obsahuje více než jeden sloupec definuje se jako "table constraints". Ve výše uvedeném příkladu byly při vytváření tabulky EXEMPLAR integritní omezení definované jako "column constraints" tj. uvnitř specifikace sloupce. Pokud by se stejná integritní omezení definovaly formou "table constraints" příkaz CREATE TABLE by vypadal následovně:

### TABLE CONSTRAINTS

```

CREATE TABLE exemplar
(
    id NUMBER(6,0),
    isbn VARCHAR2(16),
    dat_nakupu DATE DEFAULT SYSDATE,
    vypujceno CHAR(1),
    PRIMARY KEY (id),
    FOREIGN KEY(isbn) REFERENCES kniha,
    CHECK (vypujceno IN ('A','N')));

```

Chceme-li definovat unikátní klíč např. na spojení sloupců ID a ISBN, musíme toto integritní omezení definovat také jako "table constraints" protože zahrnuje dva sloupce. (Definici nelze přímo přiřadit ke sloupci).

...

UNIQUE (isbn,id)

...



## Pojmenování integritních omezení

Ke zvýšení přehlednosti integritních omezení je možno využít možnosti integritní omezení pojmenovat. Příkaz pro vytvoření tabulky EXEMPLAR by pak mohl vypadat následovně:

```
CREATE TABLE exemplar
(
  id NUMBER(6,0) CONSTRAINT exemplar_klic PRIMARY KEY,
  isbn VARCHAR2(16),
  dat_nakupu DATE DEFAULT SYSDATE,
  vypujceno CHAR(1),
  CONSTRAINT exemplar_ref_kniha FOREIGN KEY(isbn) REFERENCES kniha,
  CONSTRAINT exemplar_vycet_vypujceno CHECK (vypujceno IN ('A','N')));
```

## Zjištění názvů integritních omezení

Názvy různých integritních omezení můžeme zjistit pomocí dotazu na pohled USER\_CONSTRAINTS systémového katalogu. Struktura tohoto pohledu je:

```
SQL> DESCRIBE user_constraints
```

Name	Null?	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
CONSTRAINT_TYPE		VARCHAR2(1)
TABLE_NAME	NOT NULL	VARCHAR2(30)
SEARCH_CONDITION		LONG
R_OWNER		VARCHAR2(30)
R_CONSTRAINT_NAME		VARCHAR2(30)
DELETE_RULE		VARCHAR2(9)
STATUS		VARCHAR2(8)

Vysvětlivky ke sloupci CONSTRAINT\_TYPE (v záhlaví zobrazen jako "C"). Rozlišení zda se jedná o CHECK nebo NOT NULL je ve sloupci SEARCH\_CONDITION):

- P PRIMARY KEY
- U UNIQUE
- C CHECK
- NOT NULL

# Modifikace tabulky

Lekce 03

dp\_s08\_i03

## ALTER TABLE

Příkaz ALTER TABLE je používán pro:

- přidání nového sloupce
- změnu existujícího sloupce
- definovat výchozí (DEFAULT) hodnotu pro sloupec
- zrušení sloupce

V tabulce můžeme přidat nebo změnit sloupec. ale nikdy nemůžeme specifikovat, kde se sloupec objeví. Nově přidáný sloupec se vždy stane posledním sloupcem tabulky.

Také v případě, že tabulka již obsahuje řádky dat a přidáte nový sloupec do tabulky, obsahuje nový sloupec zpočátku null hodnoty pro všechny řádky.

### ALTER TABLE: PŘIDAT SLOUPEC

Pro přidání sloupce používáme následující SQL syntaxi:

```
ALTER TABLE tablename  
ADD (column name datatype [DEFAULT expression],  
column name datatype [DEFAULT expression], ...
```

### PŘÍKLADY:

```
ALTER TABLE copy_f_staffs  
ADD (hire_date DATE DEFAULT SYSDATE);  
  
ALTER TABLE copy_f_staffs  
ADD (e_mail_address VARCHAR2(80));
```

### ALTER TABLE: ZMĚNA SLOUPCE

Modifikace sloupce může zahrnovat změnu datového typu, velikosti a výchozí hodnoty sloupce. Pravidla pro modifikaci sloupce - viz dokumentace:

### PŘÍKLADY:

```
ALTER TABLE mod_emp MODIFY (last_name VARCHAR2(30));  
ALTER TABLE mod_emp MODIFY (last_name VARCHAR2(10));  
ALTER TABLE mod_emp MODIFY (salary NUMBER(10,2));  
ALTER TABLE mod_emp MODIFY (salary NUMBER(8,2) DEFAULT 50);
```

# 3. ODDÍL

## Obsah oddílu

- Anatomie příkazu SQL
- Práce se sloupci, znaky a záznamy
- Omezení výběru záznamů (Selection)
- Třídění záznamů

## Anatomie příkazu SQL

Lekce 01

dd\_s15\_I01

### Co se v této lekci naučíte:

- spárovat projekci, selekci a spojování s jejich správnými funkcemi/schopnostmi
- vytvořit základní příkaz SELECT
- použít správnou syntaxi k zobrazení všech řádků v tabulce
- použít správnou syntaxi k výběru specifických sloupců v tabulce, změnit způsob zobrazení dat a/nebo provádět výpočty pomocí aritmetických výrazů a operátorů
- formulovat dotazy pomocí správné precedence operátoru k zobrazení požadovaných výsledků
- definovat prázdnou (NULL) hodnotu
- ukázat dopad prázdné hodnoty v aritmetických výrazech
- sestavit dotaz pomocí sloupcového aliasu

### Proč se to učit?

- SELECT je jedním z nejdůležitějších, ne-li nejdůležitější klíčové slovo v SQL. SELECT používáme k výběru informací z databáze. Když se naučíte používat SELECT, otevřete si dveře k databázím.
- Představte si databázi obsahující informace o filmech, jako je název, žánr, studio, producent, datum uvedení, série, země, jazyk, hodnocení, délka atd. Co když chcete pouze názvy filmů vyrobených v Indii? Příkaz SELECT vám umožní vyhledávat konkrétní data.

## Příkaz SELECT

Příkaz SELECT načítá informace z databáze.

### SYNTAXE PŘÍKAZU SELECT (ZÁKLADNÍ):

```
SELECT <column_name(s)>
FROM <table_name>;
```

Ve své nejjednodušší formě musí příkaz SELECT obsahovat následující:

- klauzuli SELECT, která určuje sloupce, které mají být zobrazeny
- klauzuli FROM, která určuje tabulku obsahující sloupce uvedené v klauzuli SELECT

### KONVENCE

V průběhu tohoto kurzu budeme používat tyto konvence:

- Klíčové slovo odkazuje na individuální příkaz SQL. Například, **SELECT** a **FROM** jsou klíčová slova.
- Klauzule je součástí SQL příkazu. Například, **SELECT title** je klauzule.
- Příkaz je kombinací dvou klauzulí. Například **SELECT title FROM d\_songs** je příkaz.

## Schopnosti příkazu SELECT

**Projekce:** používá se k výběru sloupců v tabulce.

**Selekce (Výběr):** Používá se k výběru řádků v tabulce.

Projekce

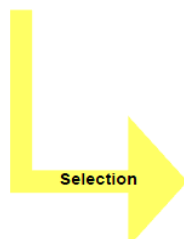
S1	S2	S3	S4	S5

Selekce

S1	S2	S3	S4	S5

### PROJEKCE A SELEKCE

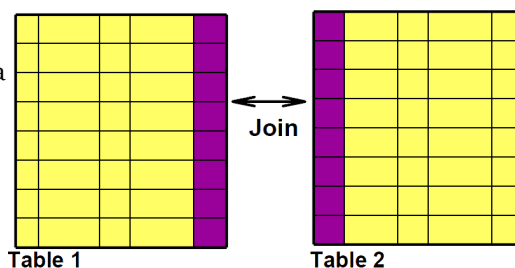
```
SELECT salary
FROM employees
WHERE last_name like 'Smith';
```



ID	FIRST_NAME	LAST_NAME	SALARY
10	John	Doe	4000
20	Jane	Jones	3000
30	Sylvia	Smith	5000
40	Hai	Nguyen	6000

## ■ SPOJOVÁNÍ TABULEK

**Spojení:** Používá se ke sloučení dat, která jsou uložena v různých tabulkách, vytvořením odkazu mezi nimi. O spojování se budete učit později během kurzu.



## ■ VÝBĚR VŠECH SLOUPCŮ

Můžete zobrazit všechny sloupce dat v tabulce pomocí hvězdičky (\*) místo názvu sloupce v klauzuli SELECT.

V následujícím příkladě vybereme všechny sloupce v tabulce d\_songs.

```
SELECT *
FROM d_songs;
```

Můžete také zobrazit všechny sloupce v tabulce tak, že je vyjmenujete jednotlivě.

```
SELECT id, title, duration, artist, type_code
FROM d_songs;
```

ID	TITLE	DURATION	ARTIST	TYPE_CODE
45	Its Finally Over	5 min	The Hobbits	12
46	I'm Going to Miss My Teacher	2 min	Jane Pop	12
47	Hurrah for Today	3 min	The Jubilant Trio	77
48	Meet Me at the Altar	6 min	Bobby West	1
49	Lets Celebrate	8 min	The Celebrants	77
50	All These Years	10 min	Diana Crooner	88

## ■ PROJEKTOVÁNÍ SPECIFICKÝCH SLOUPCŮ

Pokud chcete vybrat (projektovat) pouze určité sloupce z tabulky, které se mají zobrazit, jednoduše je vyjmenujte a jména sloupců oddělte čárkou v klauzuli SELECT.

```
SELECT id, title, artist
FROM d_songs;
```

ID	TITLE	ARTIST
45	Its Finally Over	The Hobbits
46	I'm Going to Miss My Teacher	Jane Pop
47	Hurrah for Today	The Jubilant Trio
48	Meet Me at the Altar	Bobby West
49	Lets Celebrate	The Celebrants
50	All These Years	Diana Crooner

## Použití aritmetických operátorů

Pomocí několika jednoduchých pravidel a pokynů můžete sestavit SQL příkazy, které budou snadno čitelné a snadno změnitelné. Budete-li znát tato pravidla, bude pro vás učení SQL jednoduché.

Možná budete muset změnit způsob zobrazování dat, provést výpočty, nebo vyhodnotit scénáře "co-kdyby". Například: "Co kdyby se každému zaměstnanci zvýšil plat o 5%? Jak by to ovlivnilo naše roční zisky?"

Tyto typy výpočtů jsou všechny možné pomocí aritmetických výrazů. Již znáte aritmetické výrazy v matematice:

**sčítat (+), odčítat (-), násobit (\*) a dělit (/).**

Všimněte si, že použití těchto operátorů nevytváří nové sloupce v tabulkách ani nemění aktuální hodnoty dat. Výsledky výpočtů se objeví pouze ve výstupu.

LAST_NAME	SALARY	SALARY + 300
King	24000	24300
Kochhar	17000	17300
De Haan	17000	17300

Uvedený příklad používá operátor sčítání pro výpočet zvýšení mezd o 300 pro všechny zaměstnance a zobrazí nový sloupec SALARY + 300 ve výstupu.

```
SELECT last_name, salary, salary + 300
FROM employees;
```

Prázdné mezery před a za aritmetickým operátorem nijak neovlivní výstup.

## PRECEDENCE V ARITMETICKÝCH OPERÁTORECH

Precedence je pořadí, ve kterém Oracle vyhodnocuje různé operátory ve stejném výrazu. Při hodnocení výrazu obsahujícího více operátorů Oracle nejprve hodnotí operátory s vyšší precedencí a poté ty s nižší. Operátory se stejnou precedencí v rámci jednoho výrazu hodnotí Oracle v pořadí zleva doprava.

Aritmetické operátory vykonávají matematické operace násobení, dělení, sčítání a odčítání. Pokud se tyto operátory objeví společně ve výrazu, provede se nejprve násobení a dělení. Takže pořadí je: \* / + -.

Pokud mají operátory ve výrazu stejnou prioritu, provádí se vyhodnocení zleva doprava. Vždy můžete použít závorky a tím si vynutit vyhodnocení výrazu v závorce jako první.

## NULL hodnoty

V jazyce SQL je NULL zajímavé slovo. Abychom jej pochopili, musíme vědět, co je a co není NULL. NULL je hodnota, která není k dispozici, není přiřazena, není známá nebo není použitelná.

NULL není totéž jako nula nebo mezera. V SQL je nula číslo, a mezera je znak.

Někdy neznáte hodnotu sloupce. V databázi můžete ukládat i neznámé hodnoty. Relační databáze používají zástupce, tzv. NULL nebo null, místo těchto neznámých hodnot.

Pokud je nějaká hodnota sloupce v aritmetickém výrazu null, je výsledek null nebo neznámý. Pokusíte-li se dělit hodnotou null, bude výsledek null nebo neznámý. Pokud se ale pokusíte dělit nulou, dostanete chybu!

## Alias

Alias je způsob, jak přejmenovat záhlaví sloupce ve výstupu.

Pokud zobrazujeme výsledek SQL příkazu bez aliasu, zobrazí se stejné názvy sloupců jako názvy v tabulce nebo název ukazující aritmetickou operaci, např. 12\*(SALARY + 100).

Nejspíš budete chtít, aby váš výstup pro zobrazení ukazoval jméno, které je snáze pochopitelné, více "přátelské". Alias sloupců vám dovolí přejmenovat sloupce ve výstupu.

Při použití aliasů k formátování výstupu platí několik pravidel.

### Alias sloupce:

- přejmenuje záhlaví sloupce
- je vhodný pro výpočty
- následuje ihned po názvu sloupce
- může mít nepovinné AS klíčové slovo mezi názvem sloupce a aliasem
- vyžaduje dvojité uvozovky, pokud alias obsahuje mezery, speciální znaky nebo rozlišuje velká a malá písmena

## POUŽITÍ ALIASŮ SLOUPCŮ

Syntaxe pro aliasy je:

```
SELECT * |column|expr [ AS alias], .....FROM table;
```

### PŘÍKLADY:

```
SELECT last_name AS name, commission_pct AS comm
FROM employees;
```

```
SELECT last_name "Name", salary*12 AS "Annual Salary"
FROM employees;
```

NAME	COMM
King	
Kochhar	
De Haan	

Name	Annual Salary
King	288000
Kochhar	204000
De Haan	204000

## Práce se sloupci, znaky, a záznamy (řádky)

Lekce 02

dd\_S16\_I01

Co se v této lekci naučíte:

- použití operátoru "zřetězení" k tomu, aby spojily sloupce v jeden, aritmetické výrazy, znakové výrazy, použití aliasů ve výrazech
- definovat a použít DISTINCT k odstranění zdvojených řádků

Proč se to učit?

- Kdybys psal článek o Olympiádě, možná že bys chtěl vědět kolik tam bylo různých zemí a kolik atletů z každé země závodilo. Musel bys procházet seznamy a prezenční listiny jmen, a to by mohlo být velmi nudné
- Naštěstí s použitím SQL by vaše práce mohla zabrat méně než minutu. Navíc byste mohli formátovat váš výstup tak, aby se četl jako věta. Poznáte tyto velmi užitečné vlastnosti SQL.

### Popis struktury tabulky

Používej příkaz DESCRIBE (DESC) pro zobrazení struktury tabulky.

```
DESCRIBE <jméno_tabulky>;
```

DESC vrací jméno tabulky, schéma, tablespace, indexy, spouštěče (triggery), omezení, a komentáře, stejně jako datové typy, primární a cizí klíče, a které sloupce mohou být NULL.

### PŘÍKLAD:

```
DESC department;
```

Table	Name	Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comments
DEPARTMENTS	DEPARTMENT_ID	Number		4	0	1			Primary key column of departments table.
	DEPARTMENT_NAME	Varchar2	30						A not null column that shows name of a department...
	MANAGER_ID	Number		6	0		√		Manager_id of a department. Foreign key to employee_id column of employees table...
	LOCATION_ID	Number		4	0		√		Location id where a department is located. Foreign key to location_id column of locations table.

Toto je důležitá informace při vkládání nových řádků do tabulky protože musíte znát typ dat každého sloupce a zda sloupec může být zanechán prázdný.

## Operátor zřetězení

Zřetězení je spojení řetězců dohromady. Symbol pro zřetězení je 2 x "svislý prut" někdy označovaný jako "roura" – "||". Sloupce po obou stranách operátoru || jsou zkombinované tak, aby vytvořily jeden výstupní sloupec.

Syntaxe je:

**string1 || string2 || string\_n**

Jestliže jsou hodnoty slučitelné dohromady, výsledná hodnota je znakový řetězec.

V SQL může operátor zřetězení spojit sloupec tabulky s dalšími sloupci, aritmetickými výrazy i konstantami, a vytvořit tak znakový výraz. Operátor zřetězení je užíván k tomu, aby vytvořil čitelný textový výstup.

V následující příkladu je spojen sloupec department\_id a department\_name a mezi nimi je vložena mezera - znak umístěný do apostrofů. Pro daný výraz je použit alias "Department Info":

Department Info
10 Administration
20 Marketing
50 Shipping
60 IT
...

```
SELECT department_id || ' ' || department_name AS "Department Info"
FROM departments;
```

Použitím zřetězení a doslovných hodnot můžete vytvořit výstup tak, že vypadá téměř jako věta.

Doslovné hodnoty mohou být zahrnuté ve výběrovém seznamu s operátorem zřetězení. Znaky a data musí být umístěny mezi jednoduché uvozovky - '... '.

Můžete také zahrnout čísla jako doslovné hodnoty. V následující příkladu je číslo 1 spojeno s řetězcem ' má ' a ' roční plat ' a výraz s výpočtem platu a ' dolarů '.

```
SELECT last_name || ' has a ' || 1 || ' year salary of ' ||
salary*12 || ' dollars.' AS Pay
FROM employees;
```

## Použití DISTINCT k odstranění zdvojených řádků

Mnohokrát chcete vědět kolik jedinečných příkladů něčeho existuje. Například chcete-li seznam všech oddělení, která někde jsou pro zaměstnance?



**PŘÍKLAD**

Můžete psát dotaz k tomu, aby vybral čísla oddělení z tabulky zaměstnanců:

```
SELECT department_id
FROM employees;
```

Všimněte si všech zdvojených řádků. Jak můžete modifikovat příkaz, aby se odstranil duplikát řádků?

```
SELECT DISTINCT department_id
FROM employees;
```

DEPARTMENT_ID
90
90
90
10
110
110
80
80
80
...

Používejte klíčové slovo DISTINCT k tomu, abyste odstranili zdvojené záznamy dotazu.

DISTINCT ovlivňuje veškeré uvedené sloupce a vrací každou zřetelnou kombinaci sloupců ve frázi výběru. Klíčové slovo DISTINCT musí být použito ihned po klíčovém slově SELECT.

## Omezení výběru záznamů (Selection)

Lekce 03

dd\_s16\_102

### Co se v této lekci naučíte?

- použít SQL syntaxi k tomu, abyste omezily výběr řádků vrácených jako výsledek dotazu a demonstrovat aplikaci fráze WHERE v syntaxi příkazu
- vysvětlit, proč je důležité z obchodní hlediska snadno omezit data získaná z tabulky
- vytvořit SQL dotaz, jehož výstupem budou znakové řetězce a datum

### Proč se to učit?

- Už jste někdy byli "přetíženi informacemi"? Běží televize, vaše máma se vás ptá, jak dnes bylo ve škole, zvoní telefon, a pes hlasitě štěká. Nebylo by hezké, kdybychom dokázali omezit množství informací, které musíme zpracovat najednou? V SQL je právě toto práce fráze WHERE.
- Je důležité vybrat si informaci, kterou potřebujete vidět z tabulky. Tabulky mohou mít miliony řádků dat, a je plýtvání zdroji hledat a vracet data, která nepotřebujete či nechcete.

### Příkaz SELECT - příkaz výběru

Pro vyhledání informace z databáze používáte SELECT. A příkaz výběru musí minimálně obsahovat frázi SELECT a frázi FROM. Fráze WHERE je volitelná<sup>2</sup>.

```
SELECT*|{[DISTINCT] column | expression alias}...
FROM table
[WHERE condition(s)];
```

<sup>2</sup> v obecné syntaxi je volitelnost fráze znázorněna hranatými závorkami

## Fráze WHERE

Při výběru dat z databáze můžete potřebovat omezit výběr zobrazených záznamů. To provedete toto použitím fráze WHERE. Klauzule WHERE obsahuje podmínku, která musí být splněna, a fráze WHERE přímo následuje za frází FROM v příkazu SQL.

### SYNTAXE PRO FRÁZI WHERE:

```
WHERE column_name comparison_condition {column_names
| constants | list of values}
```

Poznámka: Alias nemůže být použit ve frází WHERE!

### PŘÍKLAD

Následující příkazy SQL vybírají data z databáze "DJs on Demand":

```
SELECT id, first_name, last_name
FROM d_partners;
```

ID	FIRST_NAME	LAST_NAME
11	Jennifer	cho
22	Jason	Tsang
33	Allison	Plumb

Všimněte si jak se přidáním fráze WHERE omezily řádky výběru jen na ty, kde je hodnota ID rovna 22.

```
SELECT first_name, last_name, expertise
FROM d_partners
WHERE id=22;
```

ID	FIRST_NAME	LAST_NAME
22	Jason	Tsang

## Operátory pro podmínku klauzuli WHERE

Jak jste viděli na předchozím příkladu, operátor = může být použitý ve frází WHERE. Pro sestavení podmínky výběru lze použít následující operátory:

	Význam	Příklad
<b>Relační operátory</b>		
=	rovná se	TITUL = 'Ing.'
!= , < >	nerovná se	TITUL != 'Ing.'
>	větší než	NAROZEN > '23.5.1973'
<	menší než	NAROZEN < '12.12.1960'
>=	větší nebo rovno	POKUTA >= 50
<=	menší nebo rovno	POKUTA <= 100
BETWEEN <dolní mez intervalu> AND <horní mez intervalu>	leží v intervalu hodnot	NAROZEN BETWEEN '1.1.1970' AND '1.1.1980'
IN(seznam prvků množiny)	patří do množiny hodnot	OKRES IN('NJ', 'OV', 'FM')
LIKE 'vzor'	test podobnosti řetězce % ... zastupuje skupinu znaků _ ... zastupuje jeden znak	MĚSTO LIKE 'PRAHA%'
<b>Logické operátory</b>		
AND	logický součin („a zároveň“)	
OR	logický součet („nebo“)	
NOT	negace	
<b>Množinové operace</b>		
INTERSECT, UNION, MINUS	průnik sjednocení rozdíl	

Znakové řetězce a data musí být v podmínce fráze WHERE uzavřeny v apostrofu - '...'.  
Čísla se neuzavírají v apostrofech.

## PŘÍKLAD

```
WHERE event_date = '01-JAN-04'
WHERE rental_fee >=2000
WHERE cd_title = 'White Rose'
```

## PŘÍKLAD

Co si myslíte, že se stane při napsání následující fráze WHERE ?

```
WHERE prijmeni = 'novák';
```

Veškerá znaková hledání jsou case-sensitive, tzn. rozlišují se velká a malá písmena. Protože tabulka D\_CLIENTS ukládá všechna příjmení velkými písmeny, žádné řádky nejsou vráceny.

**Toto je důležitý bod**, který si zapamatujte. V další lekci se budete učit používat jiná SQL klíčová slova – UPPER, LOWER a INITCAP to pomůže vyhnout se chybě s velkými a malými písmeny.

## PŘÍKLAD

V následujícím příkladu z DJs on Demands: které řádky budou vybrány? Budou v sadě výsledků zahrnuté platy 3000?

```
SELECT last_name, salary
FROM employees
WHERE salary <= 3000;
```

## PŘÍKLAD

Podívejme se na následující příkaz SELECT. V jakém pořadí jsou výrazy vyhodnocovány a počítány?

```
SELECT last_name||' '||salary*1.05
As "Employee Raise"
FROM employees
WHERE department_id IN(50,80)
AND first_name LIKE 'C%'
OR last_name LIKE '%s%';
```

Naštěstí, když jsou věci komplikované, SQL má několik základních pravidel, které lze snadno sledovat.

## PRÁVIDLA PŘEDNOSTI A CO SE STANE PRVNÍ?

ORDER	OPERATORS
1	Arithmetic + - * /
2	Concatenation
3	Comparison <, <=, >, >=, <>
4	IS (NOT) NULL, LIKE, (NOT) IN
5	(NOT) BETWEEN
6	NOT
7	AND
8	OR

# Třídění řádků dotazu

Lekce 04

dd\_s17\_l02

## Co se v této lekci naučíte?

- konstrukci dotazu k tomu, aby třídil výsledky záznamů vzestupně nebo sestupně
- použít alias jako klíč třídění
- použít jednoduchý a složený sloupec jako klíč třídění

## Proč se to učit?

- Od přírody, většina z nás potřebuje ve svém životě řád (pořadí). Představte si, že před každým obědem byste museli prohlédnout každou kuchyňskou zásuvku či skříňku, abyste našli nůž a vidličku. Pořadí, seřazení a třídění pomáhá snadnějšímu hledání věcí.
- Biologové třídí zvířata podle druhů, astronomové poznají velikost hvězdy podle jasů, a programátoři organizují Java kód v třídách. Pro návrh databázi, obchodní funkce je důležité pořadí entit a atributů; SQL používá pro třídění klauzuli ORDER BY.

## Fráze ORDER BY

Informace tříděné ve vzestupném pořadí jsou důvěrně známé většině z nás. To je to, co dělá snadnější vyhledávání čísla v telefonním seznamu, nebo nalezení slova ve slovníku.

**SQL používá pro třídění řádků frázi ORDER BY napsanou za frází FROM.**

Následující příklad používá frázi ORDER BY k tomu, aby seřadila léta vzestupně. Všimněte si: ORDER BY být poslední klauzule v příkazu SQL dotazu.

### PŘÍKLAD

```
SELECT title, year
FROM d_cds
ORDER BY year;
```

TITLE	YEAR
The Celebrants Live in Concert	1997
Graduation Songbook	1998
Songs from My Childhood	1999
Carpe Diem	2000
Party Music for All Occasions	2000
Here Comes the Bride	2001
Back to the Shire	2002
Whirled Peas	2004

### TŘÍDĚNÍ - SESTUPNÉ

Standardní pořadí v ORDER BY můžete otočit na sestupné pořadí specifikováním klíčového slova DESC, zapsaném po jménu sloupce.

```
SELECT title, year
FROM d_cds
ORDER BY year DESC;
```

### POUŽÍVÁNÍ SLOUPCOVÝCH DRUHÝCH JMEN

Můžete seřadit data podle aliasu sloupce. Alias je používán jako každý jiný sloupec ve frází ORDER BY

# 4. ODDÍL

## Obsah oddílu

- Funkce pro manipulaci se znaky
- Číselné funkce
- Datumové funkce
- Konverzní funkce
- Funkce pracující s hodnotou NULL
- Podmíněné výrazy

## Manipulace se znaky

Lekce 01

dp\_S01\_I01

### Co se v této lekci naučíte?

- vybrat a aplikovat jednořádkové funkce, které provedou převod a/nebo manipulaci se znaky
- vybrat a aplikovat funkce pro manipulaci se znaky LOWER, UPPER, a INITCAP v SQL dotazu
- vybrat a použít funkce pro manipulaci se znaky CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD, TRIM a REPLACE v SQL dotazu
- napsat pružné dotazy pomocí substituce proměnných

### Proč se to učit?

- Přemýšleli jste někdy o různých způsobech, jimiž se prezentujeme? Máme šaty pro slavnostní příležitosti, pro hry, oblékáme dresy na sportovní akce a koncerty kapel. Být schopen změnit způsob, jakým se díváme na různé situace, je důležité. Jak by jste se chtěli prezentovat na přijímacím pohovoru?
- Být schopen změnit způsob, jakým jsou prezentována data je důležité při nakládání s údaji z databáze. Většinu času v SQL potřebujeme měnit způsoby (cesty) tak, aby se data zobrazovala v závislosti na požadavcích daného úkolu, kterého se snažíme dosáhnout.
- V této sekci se dozvíte několik možností, jak transformovat data tak, aby odpovídaly konkrétní situaci.

## Tabulka DUAL

Tabulka DUAL má jeden řádek s názvem "X" a jeden sloupec nazvaný "DUMMY". Tabulka DUAL se používá k vytváření příkazů SELECT a provedení příkazů, které přímo nesouvisí s konkrétní databázovou tabulkou. Dotazy, které používají tabulku DUAL, vrátí ve výsledku jeden řádek. Tabulka DUAL může být užitečná k provedení výpočtů, jako v následujícím příkladu a také k vyhodnocení výrazů, které nejsou získané z tabulky.

Tabulka DUAL bude použita k výuce mnoha jednořádkových funkcí.

### PŘÍKLAD

```
SELECT (319/29) + 12  
FROM DUAL;
```

## Jednořádkové znakové funkce

Jednořádkové znakové funkce jsou rozděleny do dvou kategorií:

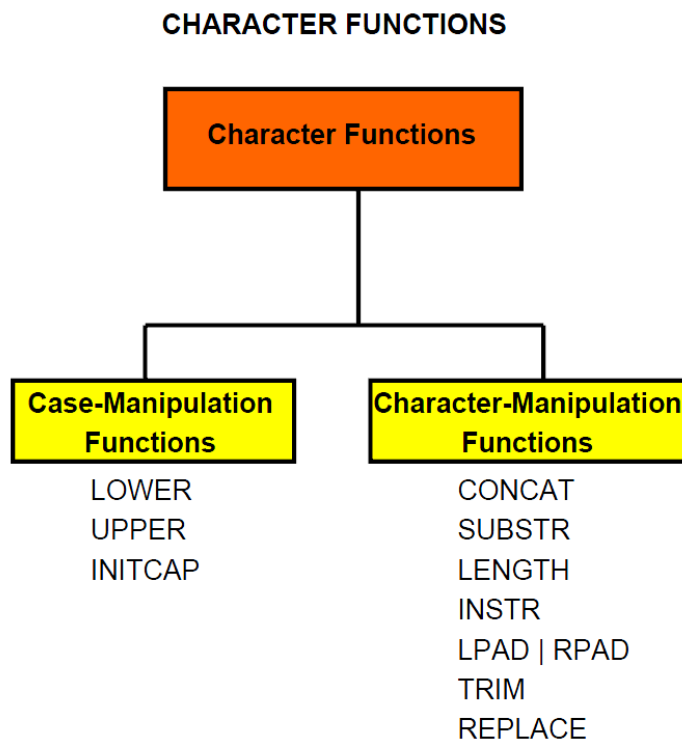
- Funkce, které převádějí znakové řetězce.
- Funkce, které mohou spojit, získat, ukázat, najít, doplnit a ořezat řetězce znaků.

Jednořádkové funkce mohou být použity v klauzuli SELECT, WHERE a ORDER BY.

Jednořádkové znakové funkce (pokračování)

Funkce pro manipulaci se znaky jsou důležité, protože nemusíte vždy vědět, jaká písmena (velká, malá nebo oboje) dat jsou uložena v databázi. Manipulace se znaky vám umožňuje dočasně převést databázová data na znaky dle vašeho výběru. Vyhneme se nesrovnalostem mezi uloženými databázovými znaky a znaky dotazu.

## Funkce pro manipulaci se znaky



Funkce pro manipulaci se znaky slouží k převodu z malých na velká písmena, nebo kombinovaná. Tyto převody mohou být použity pro formátování výstupu a mohou být také použity pro vyhledávání konkrétních řetězců.

Funkce pro manipulaci se znaky mohou být použity ve většině částí příkazu SQL.

Funkce pro manipulaci se znaky jsou často užitečné, když hledáte údaje a nevíte, zda údaje, které hledáte obsahují velká nebo malá písmena. Z pohledu databáze 'V' a 'v' nejsou stejné znaky, a proto je nutné hledat pomocí správného znaku.

### **LOWER(SLOUPEC | VÝRAZ)**

Převede alfa znaky na malá písmena.

```
SELECT title
FROM d_cds
WHERE LOWER(title) = 'carpe diem';
```

### **UPPER(SLOUPEC | VÝRAZ)**

Převede alfa znaky na velká písmena.

```
SELECT title
FROM d_cds
WHERE UPPER(title) = 'CARPE DIEM';
```

### **INITCAP(SLOUPEC | VÝRAZ)**

Převede první znak každého slova na velké písmeno.

```
SELECT title
FROM d_cds
WHERE INITCAP(title) = 'Carpe Diem';
```

## **Funkce pro manipulaci se znaky**

Funkce pro manipulaci se znaky se používají k získání, změně, formátování nebo úpravě řetězce znaků.

Jeden nebo více znaků nebo slov je předáno funkci, která vykoná svou úlohu na vstupním řetězci znaků a vrátí změněnou, získanou, spočítanou, nebo upravenou hodnotu.

- **CONCAT**: spojí dvě hodnoty dohromady
- **SUBSTR**: získá řetězec stanovené délky
- **LENGTH**: zobrazí délku řetězce jako číselnou hodnotu
- **INSTR**: najde číselnou pozici pojmenovaného znaku
- **LPAD**: doplní na levé straně řetězce znakem na požadovanou délku
- **RPAD**: doplní na pravé straně řetězce znakem na požadovanou délku
- **TRIM**: odstraní všechny uvedené znaky buď na začátku nebo na konci řetězce.
- **REPLACE**: nahradí posloupnost znaků v řetězci jinou sadou znaků.

**SYNTAXE FUNKCE TRIM:**

```
TRIM( [ leading | trailing | both
[character(s) to be removed ] ] string to trim)
```

**SYNTAXE FUNKCE REPLACE:**

```
REPLACE (string1, string_to_replace, [replacement_string] )
```

string1 je řetězec, ve kterém budou znaky nahrazeny, string\_to\_replace je řetězec, který bude vyhledán a vyjmut z řetězce1, [replacement\_string] je nový řetězec, který má být vložen do řetězce1.

**PŘÍKLAD REPLACE:**

```
SELECT REPLACE('JACK and JUE', 'J', 'BL') "Changes"
FROM DUAL;
```

**Použití aliasů (přezdívek) sloupců s funkcemi**

Všechny funkce pracují na hodnotách, které jsou v závorkách a každý název funkce označuje její účel, což je dobré mít na paměti při vytváření dotazů. Také si všimněte použití přezdívek pro sloupce s funkcemi.

Ve výchozím nastavení se zobrazí název sloupce jako záhlaví sloupce. V tomto dotazu ovšem není žádný sloupec v tabulce pro zobrazení výsledku, takže je místo toho použita syntaxe dotazu, jak je vidět v druhém příkladu.

**PŘÍKLAD 1:**

```
SELECT LOWER (last_name) || LOWER(SUBSTR(first_name,1,1))
AS "User Name"
FROM f_staffs;
```

**PŘÍKLAD 2**

```
SELECT LOWER (last_name) || LOWER(SUBSTR(first_name,1,1))
FROM f_staffs;
```

**Substituce proměnných**

Občas potřebujete spustit stejný dotaz s mnoha různými hodnotami. Bez použití substituce proměnných by to znamenalo, že budete muset neustále upravovat stejnou část klauzule WHERE. Naštěstí pro nás, Oracle APEX podporuje nahrazení proměnných. K jejich použití, vše, co musíte udělat, je vyměnit pevně zakódovanou hodnotu v prohlášení: named\_variable (pojmenovanou proměnnou). Oracle Application Express se potom ptá na hodnotu při spuštění vašeho příkazu.

**PŮVODNÍ DOTAZ:**

```
SELECT first_name, last_name, salary, department_id
FROM employees
WHERE department_id = 10; (and then 20, 30, 40...)
```



## ■ MŮŽE BÝT PŘEPSÁN NA NOVÝ:

```
SELECT first_name, last_name, salary, department_id
FROM employees
WHERE department_id = :dept_id
```

Všimněte si použití **:před dept\_id**. Tato dvojtečka je trochu kouzelná a umožňuje Oracle Application Express přijmout hodnoty proměnné. Proměnné jsou považovány v Oracle Application Express za řetězce znaků, což znamená, že při předávání znakových dat nebo dat ve formátu datum nemusíte použít jednoduché uvozovky, které se běžně používají pro uzavření řetězce.

Klauzule WHERE bude vypadat takto:

```
SELECT *
FROM employees
WHERE last_name = :l_name;
```

Po klepnutí na tlačítko Spustit (Run) se v Oracle Application Express zobrazí pop\_up menu (vyskakovací okno):



## Číselné funkce

Lekce 02

dp\_S01\_I02

### Co se v této lekci naučíte?

- vybrat a použít jednořádkové číselné funkce ROUND, TRUNC a MOD v SQL dotazu
- rozlišit výsledky získané aplikací TRUNC na číselné hodnoty a aplikací ROUND na číselné hodnoty
- uvést důsledky v účetnictví podniku při aplikaci TRUNC a ROUND na číselné hodnoty

### Proč se to učit?

- Jeden z důvodů, proč dát své peníze bance je jejich zúročení během této doby. Banky nastavují úrokové sazby podle různých ekonomických ukazatelů, jako je inflace a akciový trh. Obvykle se úrokové sazby vyjadřují v procentech, např. 3,45%.
- Co kdyby se banka rozhodla zaokrouhlit procentní sazbu na 3,5%? Bylo by to ve váš prospěch? Co kdyby se rozhodli zrušit desetinné hodnoty a vypočítat úrok na 3%, byli byste potom spokojeni?
- Zaokrouhlování a ořezávání čísel hraje důležitou roli v podnikovém účetnictví potažmo v podnikové databázi, která slouží k ukládání a zpracování číselných dat.

## Základní číselné funkce

Mezi 3 základní číselné funkce patří:

- ROUND
- TRUNC
- MOD

### ROUND

ROUND může být použit jak s čísly, tak i s hodnotami typu datum. Používá se především pro zaokrouhlování čísel na zadaný počet desetinných míst, ale může být také použit na zaokrouhlení číslíc vlevo od desetinné čárky.

#### Syntaxe

```
ROUND (sloupec|výraz, desetinná místa)
```

Všimněte si, že pokud není uveden počet desetinných míst, nebo je 0, číslo bude zaokrouhleno na celá čísla (bez desetinných míst).

### PŘÍKLADY ROUND

```
ROUND (45.926)           Výsledek: 46
ROUND (45.926, 0)        Výsledek: 46
```

Pokud je počet desetinných míst kladné číslo, je číslo zaokrouhleno na tento počet desetinných míst.

```
ROUND (45.926, 2)        Výsledek: 45.93
```

Pokud je počet desetinných míst záporné číslo, jsou zaokrouhlené číslice vlevo od desetinné čárky.

```
ROUND (45.926, -1)       Výsledek: 50
```

### TRUNC

Funkce TRUNC může být použita jak s čísly, tak i s hodnotami typu datum. Používá se především k ořezání sloupce, výrazu nebo hodnoty na zadaný počet desetinných míst. Pokud při použití funkce TRUNC není počet desetinných míst definován, je výchozí hodnota 0.

#### Syntaxe

```
TRUNC (sloupec|výraz, desetinná místa)
```

### PŘÍKLADY TRUNC

```
TRUNC (45.926, 2)        Výsledek: 45.92
```

Stejně jako u ROUND, pokud u funkce TRUNC není definován počet desetinných míst, nebo je 0, je číslo ořezáno na celá čísla (bez desetinných míst).

```
TRUNC (45.926, 0)        Výsledek: 45
TRUNC (45.926)           Výsledek: 45
```

Pamatujte si, že TRUNC není zaokrouhlení čísla. Je to prostě ukončení čísla v daném bodě..

## MOD

Funkce MOD zjistí zbytek po vydělení jedné hodnoty jinou hodnotou.

Například MOD z 5 děleno 2 = 1.

MOD může být použit k zjištění, zda hodnota je sudá, nebo lichá. Jestliže dělíte hodnotu dvěma a výsledek je beze zbytku, musí být číslo sudé.

## PŘÍKLADY MOD

```
SELECT MOD(1600,500) FROM DUAL;
```

Výsledek: 100 remainder

```
SELECT last_name, salary, MOD(salary, 2) As "Mod Demo"
```

```
FROM f_staffs
```

```
WHERE staff_type IN('Order Taker', 'Cook', 'Manager');
```

Sloupec "Mod Demo" ukáže, zda je plat sudé nebo liché číslo.

# Datumové funkce

Lekce 03

dp\_S01\_I03

### Co se v této lekci naučíte?

- vybrat a použít jednořádkové funkce MONTHS\_BETWEEN, ADD\_MONTHS, NEXT\_DAY, LAST\_DAY, ROUND a TRUNC, které pracují s datumovými daty
- vysvětlit, jak datumové funkce převádějí Oracle data na hodnoty typu datum nebo číslo
- předvést správné užití aritmetických operátorů s daty
- ukázat použití SYSDATE a datumových funkcí
- uvést důsledky pro mezinárodní společnosti v oblasti manipulace s daty ve formátu typu datum

### Proč se to naučit?

- Přemýšleli jste někdy, kolik dní má školní rok, nebo kolik týdnů zbývá do ukončení vašeho studia? Vzhledem k tomu, že Oracle databáze ukládá datum jako čísla, je snadné provést sčítání a odčítání hodnot typu datum.
- Podniky jsou závislé na schopnosti používání datumových funkcí pro plánování mezd a plateb, sledování hodnocení výkonu, odpracovaných let zaměstnanců a sledování objednávek a dodávek. Všechny tyto činnosti musí být snadno zvládnutelné pomocí jednoduchých SQL datumových funkcí.

## Zobrazení data

Výchozí formát zobrazení data je DD-MON-RR – to je, 02-DEC-99. (anglický formát)

Nicméně Oracle databáze ukládá data interně v číselném formátu, což představuje století, rok, měsíc, den, hodiny, minuty a sekundy.

Výchozí zobrazení a vstupní formát pro každé datum je DD-MON-RR. Oracle data jsou platná od 1. ledna 4712 př. n. l. Do 31. prosince 9999, což představuje rozsah dat, které si můžete úspěšně uložit v Oracle databázi.

## ■ SYSDATE

Když je záznam s datovým sloupcem vložen do tabulky, je informace o století převzata z funkce SYSDATE. SYSDATE je datumová funkce, která vrátí aktuální datum a čas nastavený na databázovém serveru (nebo klientu; podle nastavení).

Pro zobrazení aktuálního data použijeme tabulku DUAL.

## ■ PŘÍKLAD SYSDATE

```
SELECT SYSDATE  
FROM DUAL;
```

## Datový typ DATUM

Datumový datový typ vždy ukládá interně informace o roku jako čtyřmístné číslo: 2 číslice pro století a 2 číslice pro rok. Například Oracle databáze ukládá rok jako 1996 a 2004, ne jen jako 96 a 04.

Přestože vnitřní paměť udržuje informace o úplném datu, pokud sloupec s datem je zobrazen na obrazovce, století není zobrazeno ve výchozím formátu.

## ■ PRÁCE S DATY

```
SELECT last_name, hire_date + 60  
FROM employees;  
  
SELECT last_name, (SYSDATE - hire_date)/7  
FROM employees;  
  
SELECT order_no, amt_due, purch_date + 30 "Due Date"  
FROM dual;
```

## Základní datumové funkce

Datumové funkce uvedené v tabulce pracují s Oracle daty. Všechny datumové funkce vrací hodnotu datového typu datum (DATE), s výjimkou funkce MONTHS\_BETWEEN, která vrací číselný datový typ.

Function	Description
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

### ■ PŘÍKLADY POUŽITÍ DATUMOVÝCH FUNKCÍ.

```
SELECT employee_id, hire_date,
ROUND (MONTHS_BETWEEN (SYSDATE, hire_date)) AS TENURE,
ADD_MONTHS (hire_date, 6) AS REVIEW, NEXT_DAY (hire_date, 'FRIDAY'),
LAST_DAY (hire_date) FROM employees
WHERE MONTHS_BETWEEN (SYSDATE, hire_date) > 36;
```

Další příklad dotazu, který používá několik datumových funkcí.

```
SELECT employee_id, hire_date,
ROUND (MONTHS_BETWEEN (SYSDATE, hire_date)) AS TENURE,
ADD_MONTHS (hire_date, 6) AS REVIEW,
NEXT_DAY (hire_date, 'FRIDAY'),
LAST_DAY (hire_date)
FROM employees
WHERE MONTHS_BETWEEN (SYSDATE, hire_date) > 36;
```

Výsledek tohoto dotazu zahrnuje 20 řádků. Níže je ukázka jednoho z nich.

EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY	LAST_DAY
101	21-SEP-89	181	21-MAR-90	22-SEP-89	30-SEP-89

## Konverzní funkce

Lekce 04

dp\_S02\_I01

### Co se naučíte v této lekci?

- uvést příklad explicitní a implicitní konverze datových typů
- vysvětlit, proč je z obchodního hlediska důležitá schopnost jazyka převádět formáty dat
- sestavit SQL dotaz, který správně použije jednořádkové funkce TO\_CHAR, TO\_NUMBER a TO\_DATE pro dosažení požadovaného výsledku
- použít vhodné datum a/nebo znakový formát pro vytvoření požadovaného výstupu
- vysvětlit a aplikovat užití YYYY a RRRR pro získání správného roku tak, jak je uložen v databázi

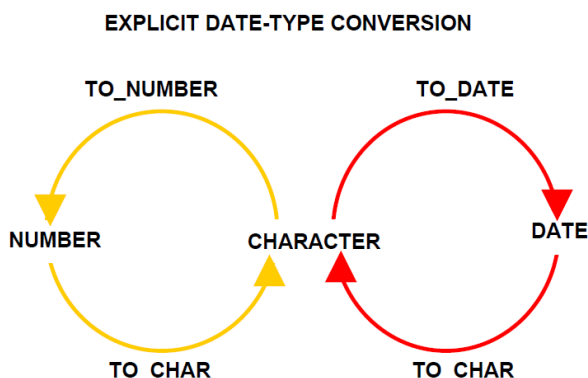
### Proč se to učit?

- Představte si, že čtete z učebnic, které jsou uloženy ve formě textových souborů bez odstavců a velkých písmen. Bylo by to obtížné čtení. Naštěstí existují programy, které mají k dispozici nastavení barvy textu, podtržení, tučné písmo, vycentrování, přidání grafiky. Pro formátování a zobrazení změn se v databázi používají konverzní funkce. Tyto funkce jsou schopny zobrazit čísla jako místní měnu, datum v různých formátech, zobrazit čas v sekundách, zjistit století.

Když je v databázi vytvořena tabulka, musí programátor SQL definovat jaká data budou uložena v každém poli tabulky. V SQL je několik různých typů dat. Tyto datové typy definují množiny (oblasti) hodnot, které každý sloupec může obsahovat. V této lekci budete používat:

- VARCHAR2      znaková data proměnné délky
- CHAR            text a znaková data pevné délky
- NUMBER        číselná data proměnné délky
- DATE            hodnoty data a času

Oracle server může implicitně převést data datových typů VARCHAR2 a CHAR na data datových typů NUMBER a DATE. I když je to užitečná funkce, je vždy lepší převést data na jiný datový typ explicitně pro zajištění spolehlivosti SQL příkazů.



Toto jsou čtyři konverzní funkce,  
které se naučíte:

- Převod datového typu datum na znakový datový typ
- Převod číselného datového typu na znakový datový typ
- Převod znakového datového typu na číselný datový typ
- Převod znakového datového typu na datový typ datum

## Převod dat typu datum na znaková data

Často potřebujete převést data uložená v databázi ve výchozím formátu DD-MON-YY do jiného, vámi zvoleného formátu.

### FUNKCE, KTERÁ SPLNÍ TENTO ÚKOL:

**TO\_CHAR** (date column name, 'format model you specify')

- Tento 'model formátu' musí být uzavřen v jednoduchých uvozovkách a rozlišuje velká a malá písmena.
- Hodnota data je od 'format model' oddělena čárkou.
- Může být vložen jakýkoliv platný formát data.
- Použijte FM element pro zrušení mezer nebo koncových nul z výstupu.
- Použijte SP pro slovní vyjádření čísla.
- Použijte TH pro zobrazení čísla jako čísla ordinálního (1.,2.,3., a tak dále)
- Použijte uvozovky pro přidání znakového řetězce do modelu formátu

YYYY	Full year in numbers
YEAR	Year spelled out
MM	Two-digit value for month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month

HH24:MI:SS AM	15:45:32 PM
DD "of" MONTH	12 of October

DDspth	FOURTEENTH
Ddspth	Fourteenth
ddspth	fourteenth
DDD or DD or D	Day of year, month or week

Tabulky ukazují různé modely formátu, které lze použít. Při zadávání času uvažujeme, že prvky formátu mohou být hodiny (HH), minuty (MI), sekundy (SS) a AM nebo PM.

Například následující dotaz vrátí May 14,2004. Kdyby datum události (event\_date) bylo 04-MAY-04, potom by model formátu pomocí fm vrátil May 4,2004 s potlačením nuly na začátku.

### PŘÍKLAD:

```
SELECT TO_CHAR(event_date, 'fmMonth dd, RRRR')
FROM d_events;
```

Jaký bude výstup následujícího dotazu?

```
SELECT id, TO_CHAR(event_date, 'MONTH DD, YYYY')
FROM d_events;
```

## Modely formátu data a času

Následující tabulky ukazují varianty modelů formátů typu datum a čas. Můžete určit modely formátu použité k zobrazení data dnešního dne jako následující výstup?

- August 6th, 2007
- August 06, 2007
- AUG 6, 2007
- August 6th, Friday, Two Thousand Seven

YYYY	Full year in numbers
YEAR	Year spelled out
MM	Two-digit value for month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month

## Převod čísla na znaková data (VARCHAR2)

Čísla uložená v databázi nejsou formátována. To znamená, že se neuchovávají žádné znaky měny, symboly, čárky desetinných míst nebo jiný typ formátování.

Chcete-li přidat formátování, musíte nejprve převést čísla na znakový formát. Tato konverze je zvláště užitečná při zřetězení.

HH24:MI:SS AM	15:45:32 PM
DD "of" MONTH	12 of October

DDspth	FOURTEENTH
Ddspth	Fourteenth
ddspth	fourteenth
DDD or DD or D	Day of year, month or week

## SQL FUNKCE, KTERÉ POUŽÍVÁTE K PŘEVODU SLOUPCŮ ČÍSEL DO POŽADOVANÉHO FORMÁTU:

**TO\_CHAR(number, 'format model')**

Tabulka ukazuje některé prvky formátu dostupné pro použití s funkcí TO\_CHAR.

```
SELECT TO_CHAR(cost, '$99,999')
COST
FROM d_events;
```

Můžete určit modely formátu použité k zobrazení následujících výstupů?

- \$3000.00
- 4,500
- 9,000.00
- 0004422

ELEMENT	DESCRIPTION	EXAMPLE	RESULT
9	Numeric position (# of 9's determine width)	999999	1234
0	Display leading zeros	099999	001234
\$	Floating dollar sign	\$999999	\$1234
L	Floating local currency symbol	L999999	FF1234
.	Decimal point in position specified	999999.99	1234.00
,	Comma in position specified	999,999	1,234
MI	Minus signs to right (negative values)	999999MI	1234-
PR	Parenthesize negative numbers	999999PR	<1234>
EEEE	Scientific notation ( must have four EEEE)	99.999EEEE	1,23E+03
V	Multiply by 10 n times (n= number of 9's after V)	9999V99	9999V99
B	Display zero values as blank, not 0	B9999.99	1234.00

## Konverze znaků na číslo

Často potřebujete převést řetězec na číslo.

## FUNKCE PRO TUTO KONVERZI JE :

**TO\_NUMBER(character string, 'format model')**

Převádí nečíselné hodnoty jako je „450“ na číslo, bez apostrofů (jednoduchých uvozovek). Jednoduché uvozovky jsou znaky. „450“ byla uložena v databázi jako znaková data a následující dotaz jej převede na číslo, se kterým lze provádět aritmetické operace. Nemůžete provádět výpočty se znakovými daty.



```
SELECT TO_NUMBER('450') AS "Number Change"
FROM DUAL;
```

```
SELECT TO_NUMBER('450', '9999') + 10 AS "Number Change" FROM DUAL;
```

SQL\*Plus zobrazí řetězec znaků – mřížky (#) na místě celého čísla, jehož počet číslic je větší než počet číslic předepsaných ve formátovacím modelu a zaokrouhlí čísla na takový počet desetinných míst, který je uveden ve formátovacím modelu.

Oracle Application Express vrátí předdefinovanou chybu Oracle - neplatné číslo, neshoduje-li se počet číslic ve formátovacím modelu se skutečným počtem vrácených číslic z databáze.

## Konverze znaků na datum

Chcete-li převést řetězec na datový formát datum, použijte:

```
TO_DATE('character string', 'format model')
```

Tato konverze převede řetězec znaků, jako je "November 3, 2001", na datum. Formátovací model říká serveru, jak znakový řetězec „vypadá“.

```
TO_DATE('November 3, 2001', 'Month dd, RRRR')      vrátí 03-NOV-01
```

Při převodu znaku na datum modifikátor fx určuje přesnou shodu pro znakový argument a formát datového modelu.

V následujícím příkladu si všimněte, že "May10" nemá žádnou mezeru mezi "May" a "10." Formátovací model FX odpovídá znakovému argumentu, zatímco také neuvádíme mezeru mezi "Mon" a "DD."

```
SELECT TO_DATE('May10,1989', 'fxMonDD,RRRR') AS "Convert"
FROM DUAL;
```

## RR a YY formát data

### NĚKOLIK JEDNODUCHÝCH PRAVIDEL:

Pokud je formát data zadán s YY nebo YYYY, bude hodnota vrácena v současném století. Takže, je-li rok 1995 a vy použijete formát YY nebo YYYY, je všechno v pořádku a data budou v 1900 století. Nicméně, pokud je rok 2004 a vy použijete YY nebo YYYY formát data pro rok 1989, získáte 2089! A to jste možná nezamýšleli.

Pokud je formát data zadán s RR nebo RRRR, pro vrácenou hodnotu jsou dvě možnosti.

Pokud se současný rok pohybuje mezi 00 až 49:

- Data 0-49: datum bude v současném století
- Data 50-99: datum bude v minulém století

Pokud je současný rok mezi 50 až 99:

- Data 0-49: datum bude v příštím století
- Data 50-99: datum bude v současném století

# Funkce NULL

Lekce 05

dp\_S02\_I02

## Co se naučíte v této lekci?

- ukázat a vysvětlit zhodnocení vnořené funkce
- seznam nejméně čtyř základních funkcí, které pracují se všemi datovými typy a řeší nullové hodnoty
- vysvětlit použití COALESCE a funkce NVL
- vysvětlit použití základních funkcí pro řešení nullové hodnoty v datech
- sestavit a spustit SQL dotaz, který správně aplikuje jednořádkové funkce NVL, NVL2, NULLIF a COALESCE

## Proč se to učit?

- Kromě funkcí, které určují, jak jsou data formátována, nebo převedena na jiný datový typ, SQL používá skupinu základních funkcí, které se speciálně zabývají nullovými hodnotami. Možná se divíte, jak si hodnota, která je k dispozici a je nepřirazená, neznámá, nebo nepoužitelná může zasloužit tolik pozornosti. Null může být „nic“, ale může ovlivnit, jak jsou výrazy vyhodnoceny, jak jsou spočítány průměry a kde se hodnota zobrazí v seřazeném seznamu. Tato lekce je celá o manipulaci s nullovými hodnotami.

## Jak jsou funkce vyhodnoceny

Až dosud jste použili jednořádkové funkce v jednoduchých příkazech. Je ovšem možné funkce vnořit do libovolné hloubky. Je ale důležité vědět, jak jsou vnořené funkce vyhodnocovány. V následujícím příkladu je vnořená funkce. Proces hodnocení začíná od nejvnitřnější úrovně k té nejvzdálenější.

```
SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date, 6), 'FRIDAY'), 'fmDay,
Month DDth, YYYY') AS "Next Evaluation"
FROM employees
WHERE employee_id=100;
```

Výsledky jsou:

```
Friday, December 18th, 1987
```

## Funkce týkající se hodnoty NULL

Na začátku samozřejmě termín „null“ představíme. Pokud si pamatujete, je to hodnota, která je k dispozici a je nepřirazená, neznámá nebo ji nelze uplatnit. My v podstatě nemůžeme testovat, zda je stejná jako jiné hodnoty, protože nevíme, jakou hodnotu má. Nerovná se to ničemu, dokonce ani ne nule. Ale jen proto, že to opravdu není nic, neznamená to, že to není důležité. Představte si tuto otázku: Je pravda, že  $X=Y$ ? Aby bylo možné odpovědět, musíme znát hodnoty  $X$  a  $Y$ . Oracle má čtyři základní funkce, které pracují s nullovými hodnotami.

## ČTYŘI FUNKCE S NULL HODNOTOU:

- NVL
- NVL2
- NULLIF
- COALESCE

## FUNKCE NVL

Můžete použít funkci NVL pro převedení hodnot sloupce, obsahujícího hodnoty null, na číslo před provedením výpočtu. Je-li aritmetický výpočet proveden s hodnotou null, výsledek je null. NVL funkce může převést hodnotu null na číslo, než jsou aritmetické výpočty provedeny, aby se zabránilo výsledku null.

## PŘÍKLAD NVL:

V tomto příkladu sloupec `auth_expense_amt` tabulky `D_PARTNERS` obsahuje hodnoty null. Funkce NVL je použita pro změnu hodnot null na nulu dříve, než jsou provedeny aritmetické výpočty.

```
SELECT first_name,  
last_name, NVL(auth_expense_amt, 0) * 1.05 AS Expenses  
FROM D_Partners;
```

## FUNKCE NVL2

Funkce NVL2 vyhodnotí výraz se třemi hodnotami. Pokud první hodnota není null, pak NVL2 vrací druhý výraz. Pokud je první hodnota null, pak je vrácen třetí výraz. Hodnota ve výrazu 1 může mít jakýkoliv datový typ. Výraz 2 a výraz 3 mohou mít jakýkoliv datový typ kromě datového typu LONG. Datový typ vrácené hodnoty je vždy stejný, jako datový typ výrazu 2, pokud výraz 2 obsahuje znaková data, jsou navraceny hodnoty typu VARCHAR2

### Syntaxe

```
NVL2 (výraz_1_hodnota, která může obsahovat null, výraz_2_hodnota, která  
je navracena, jestliže výraz 1 není hodnota null, výraz_3_hodnota, která  
je navracena, pokud hodnota výrazu 1 je null)
```

Snadný způsob, jak si funkci NVL2 zapamatovat je říct si: „Jestliže výraz 1 má hodnotu, nahradíme ji výrazem 2; pokud je výraz 1 null, nahradíme ho výrazem 3“. Uvedená NVL2 funkce má číselná data ve výrazu 1 a znaková data ve výrazech 2 a 3.

## PŘÍKLAD NVL2:

```
SELECT last_name, salary,  
NVL2(commission_pct, salary+(salary * commission_pct), salary) AS income  
FROM employees;
```

## FUNKCE NULLIF

Funkce NULLIF porovnává dva výrazy. Pokud se rovnají, funkce vrací hodnotu null. Jestliže se nerovnají, vrací funkce první výraz.

Syntaxe NULLIF je:

```
NULLIF(expression 1, expression 2)
```

#### PŘÍKLAD NULLIF

```
SELECT first_name, LENGTH(first_name) "Length FN",  
last_name, LENGTH(last_name) "Length LN",  
NULLIF(LENGTH(first_name),  
LENGTH(last_name)) AS "Compare Them"  
FROM D_PARTNERS;
```

#### FUNKCE COALESCE

Funkce COALESCE je rozšíření funkce NVL, akorát, že COALESCE může mít více hodnot. Slovo COALESCE doslovně znamená „sejít dohromady“ a to je to, co se děje. Je-li první výraz null, funkce pokračuje řádek po řádku, dokud nenajde výraz, který nemá hodnotu null. Samozřejmě, má-li první výraz hodnotu, funkce vrátí první výraz a funkce se zastaví.

Syntaxe COALESCE:

```
COALESCE (expression 1, expression 2, ...expression n)
```

#### PŘÍKLAD COALESCE

Prozkoumejte příkaz SELECT z tabulky zaměstnanci. Kterí zaměstnanci nemají obdržet provizi? Jak to můžete říct? Je tu někdo, kdo neobdrží žádnou provizi ani plat?

```
SELECT last_name, COALESCE(commission_pct, salary, 10) comm  
FROM employees  
ORDER BY commission_pct;
```

## Podmíněné výrazy

Lekce 06

dp\_s02\_l03

Co se v této lekci naučíte?

- porovnat funkce DECODE a CASE
- sestavit a spustit SQL dotaz, který správně používá funkce DECODE a CASE
- sestavit a spustit dvěma způsoby provedení IF-THEN-ELSE podmíněné logiky (jako výraz)

Proč se to učit?

- Analytici se rozhodují, které obchodní funkce je třeba modelovat a které ne. Proces datového modelování vyžaduje od návrhářů analýzu informací k identifikaci osob, řešení vztahů a výběr vlastností. Typickým rozhodnutím by mohlo být, jestliže (IF) podnik potřebuje sledovat data v

průběhu času, potom (THEN) čas může být entitou nebo (ELSE) atributem.

- Tento rozhodovací proces se příliš neliší od těch, které děláme v každodenním životě. Zamyslete se natím, kdy jste v poslední době dělali if-then-else rozhodnutí. Pokud jsem si udělal domácí úkol před 21:00, potom se mohu dívat na televizi, jinak se na televizi dívat nemůžu.
- V SQL tyto druhy rozhodování zahrnují metody podmíněného zpracování. Vědět, jak lze použít podmíněné zpracování, umožňuje rozhodováním získat snadněji data, která potřebujete.

## Podmíněné výrazy

CASE a DECODE jsou dva podmíněné výrazy. Studovali jste již funkci NULLIF, která je logicky ekvivalentní výrazu CASE, který v tomto případě porovnává dva výrazy. Pokud jsou si výrazy rovny, vrátí se hodnota null, když nejsou stejné, vrátí se první výraz.

### VÝRAZ CASE

Výraz CASE v podstatě dělá práci rozhodování IF-THEN-ELSE. Datové typy CASE, WHEN a ELSE musí být stejné.

#### CASE syntax

```
CASE expr
  WHEN comparison_expr1 THEN return_expr1
  WHEN comparison_expr2 THEN return_expr2
  WHEN comparison_exprn THEN return_exprn
  ELSE else_expr]
END
```

ID	LOC_TYPE	RENTAL_FEE	REVISED_FEES
100	Private Home	0	No increase
105	Private Home	0	No increase
101	Private Home	0	No increase
95	School Hall	75/hour	75/hour
99	National Park	400/flat fee	400/flat fee
220	Hotel	300/per person	Increase 5%

#### PŘÍKLAD:

```
SELECT id, loc_type, rental_fee,
CASE loc_type
  WHEN 'Private Home' THEN 'No Increase'
  WHEN 'Hotel' THEN 'Increase 5%'
  ELSE rental_fee
END AS "REVISED_FEES"
FROM d_venues;
```

### VÝRAZ DECODE

Funkce DECODE vyhodnotí výraz podobnou cestou jako IF-THEN-ELSE logika. DECODE porovnává výraz s každou z hledaných hodnot. Syntaxe DECODE je:

```
DECODE(column1|expression, search1, result1
[, search2, result2,...,]
[, default])
```

Jestliže chybí výchozí hodnota, je vrácena hodnota NULL, pokud vyhledávané hodnotě neodpovídá žádná z hodnot.

# 5. ODDÍL

## Obsah oddílu

- Křížové a přirozené spojení
- Klauzule JOIN
- Vnitřní versus vnější spojení (inner join - outer join)

## Křížové a přirozené spojení

Lekce 01

dp\_s03\_l01

### Co se v této lekci naučíte?

- vytvořit a provést přirozené spojení (natural join) pomocí syntaxe ANSI-99 SQL
- vytvořit křížové spojení (cross join) pomocí syntaxe ANSI-99 SQL
- definovat vztah mezi křížovým spojením a karteziánským součinem
- definovat vztah mezi přirozeným spojením a spojením equijoin

### Proč je třeba se to učit?

- Vaše současné zkušenosti s používáním SQL se zatím omezovaly na dotazování a získávání informací z jedné databázové tabulky najednou.
- To by nebyl problém, pokud by všechna data v databázi byla uložena pouze v jedné tabulce. Z datového modelování ale víte, že jádrem relačních databází je možnost oddělovat data do jednotlivých tabulek a vzájemně tyto tabulky propojovat. SQL naštěstí nabízí spojovací podmínky, které umožňují dotazovat informace z různých tabulek a kombinovat je do jedné sestavy.

## Příkazy pro spojení tabulek

Existují dvě skupiny příkazů či syntaxe, které se mohou použít k propojení tabulek v databázi:

- Oracle proprietární spojení (joins)
- Standardní spojení kompatibilní s ANSI/ISO SQL 99

V tomto kurzu se naučíte používat obě skupiny spojovacích příkazů.

## ANSI

ANSI je zkratka pro American National Standards Institute. ANSI byla založena v roce 1918 a jde o soukromou, neziskovou organizaci, která spravuje a koordinuje systém USA pro dobrovolnou standardizaci a posuzování shody.

Posláním Institutu je zvyšovat globální konkurenceschopnost podniků v USA a kvalitu života v USA díky podpoře dobrovolných standardů a systémů pro posuzování shody a ochrana jejich integrity.

## SQL (HISTORIE)

Strukturovaný dotazovací jazyk (SQL) je jazyk pro zpracování informací, který je standardem pro řídicí systémy relačních databází (RDBMS).

Jazyk původně vytvořila společnost IBM v polovině 70.let, v 80.letech se velmi rozšířil a v roce 1986 se stal odvětvovým standardem, když jej přijal ANSI.

ANSI dosud udělal tři standardizace SQL, každá z nich v návaznosti na předchozí. Jsou pojmenovány podle roku, ve kterém byly poprvé navrženy, a jsou známy pod svými krátkými názvy: ANSI-86, ANSI-92 a ANSI-99.

## Přirozené spojení (NATURAL JOIN)

Oracle proprietární equijoin vrací všechny řádky, jejichž hodnoty odpovídají v obou tabulkách.

ANSI/ISO SQL: 1999 join, který dosahuje stejné výsledky, se nazývá **přirozený join (natural join)**.

Přirozený join je založen na všech sloupcích v obou tabulkách, které mají stejný název, a vybere řádky z obou tabulek, které mají stejné hodnoty ve všech spárovaných sloupcích.

**Equijoin = ANSI/ISO SQL: 1999**

## NATURAL JOIN

Jak je uvedeno v ukázkovém kódu, při použití přirozeného joinu je možné propojit tabulky, aniž byste museli výslovně specifikovat sloupce v odpovídající tabulce. Názvy a datové typy ale musejí být stejné v obou sloupcích.

```
SELECT event_id, song_id, cd_number
FROM d_play_list_items NATURAL JOIN d_track_listings
WHERE event_id = 105;
```

Klauzule WHERE byla přidána kvůli dalšímu omezení pro jednu ze dvou tabulek, aby se omezily řádky ve výstupu.

## PŘÍKLAD:

Který sloupec nebo sloupce se použijí k přirozenému spojení (join) těchto dvou tabulek?

Všimněte si, že sloupec pro přirozené spojení se nemusí objevit ve výstupu.

```
SELECT first_name, last_name, event_date, description
FROM d_clients NATURAL JOIN d_events;
```

## Křížové spojení (CROSS JOIN)

Oracle proprietární kartézský produkt spojí každý řádek v jedné tabulce s každým řádkem v tabulce druhé. Ekvivalentem kartézského produktu v ANSI/ISO SQL: 1999 SQL je křížové spojení.

Výsledky z obou typů spojení jsou stejné. Výsledkový soubor reprezentuje všechny možné kombinace sloupců z obou tabulek. Těch může být potenciálně velmi mnoho!

### ■ PŘÍKLAD KŘÍŽOVÉHO SPOJENÍ:

```
SELECT name, event_date, loc_type, rental_fee
FROM d_events CROSS JOIN d_venues;
```

## Klauzule JOIN

Lekce 02

dp\_s03\_l02

### Co se v této lekci naučíte?

- vytvořit a provést příkaz join pomocí klauzulí ANSI-99 USING a ON
- vytvořit a provést dotaz ANSI-99, který propojí tři tabulky.

### Proč se to učit?

- S dalšími příkazy, které se naučíte, budete stále lépe schopni sestavovat dotazy, které vrátí požadované výsledky. Cílem propojení je spojit dohromady data, přes tabulky, aniž by se všechna data musela opakovat v každé tabulce.

### Fráze USING

V přirozeném propojení, pokud mají tabulky sloupce se stejnými názvy, ale různými typy dat, způsobí spojení chybu.

Aby se takové situaci předešlo, můžeme klauzuli join změnit pomocí klauzule USING. Klauzule USING určuje sloupce, které se mají použít pro equijoin.

### ■ FRÁZE USING

Zobrazený dotaz je příkladem klauzule USING. Sloupce uvedené v klauzuli USING by nikde v příkazu SQL neměly mít kvalifikátor (název tabulky nebo alias).

```
SELECT client_number, first_name, last_name, event_date
FROM d_clients JOIN d_events
USING (client_number);
```

Klauzule USING nám umožňuje použít WHERE k omezení řádků z jedné nebo obou tabulek:

```
SELECT client_number, first_name, last_name, event_date
FROM d_clients JOIN d_events
USING (client_number)
WHERE last_name = 'Peters';
```



## Fráze ON

Co když sloupce, které se mají propojit, mají různé názvy nebo spojení používá srovnávací operátory jako <, > nebo BETWEEN?

Nemůžeme použít USING, tak místo toho použijeme klauzuli ON. To umožňuje specifikovat větší paletu podmínek pro spojení. Klauzule ON nám také umožňuje použít WHERE k omezení řádků z jedné nebo obou tabulek.

V tomto příkladě je klauzule ON použita v self-join, kde má tatáž tabulka dva různé odkazy. V tabulce zaměstnanců jsou někteří zaměstnanci také vedoucími. Self-join se použije pro výběr zaměstnanců, kteří jsou zároveň vedoucími.

EMP	MGR
Hartstein	King
Zlotkey	King
Mourgos	King
De Haan	King
Kochhar	King
Higgins	Kochhar
.....	.....

```
SELECT e.last_name as "EMP", m.last_name as "MGR"
FROM employees e JOIN employees m
ON (e.manager_id = m.employee_id);
```

Zde je stejný dotaz s klauzulí WHERE, kterou se omezuje výběr řádků.

```
SELECT e.last_name as "EMP", m.last_name as "MGR"
FROM employees e JOIN employees m
ON (e.manager_id = m.employee_id)
WHERE e.last_name like 'H%';
```

## Spojování 3 tabulek

Jak USING tak i ON se dá použít ke spojení tří i více tabulek.

Předpokládejme, že potřebujeme report o našich klientech, jejich událostech a tématech pro tyto události? Musíme spojit tři tabulky: d\_clients, d\_events a d\_themes.

```
SELECT last_name, event_date, t.description
FROM d_clients c JOIN d_events e
USING (client_number)
JOIN d_themes t
ON (e.theme_code = t.code);
```

## SROVNÁNÍ ORACLE PROPRIETÁRNÍHO SPOJENÍ S ANSI/ISO SQL:1999 SPOJENÍM

Oracle Proprietary Join	ANSI/ISO SQL: 1999 Equivalent
Cartesian Product	Cross Join
Equijoin	Natural Join (if the join columns have the same name and data type)  USING clause (if the columns have the same name but different data types)  ON clause (if the columns have different names)
Non-equijoin	ON clause

## Vnitřní versus vnější spojení (inner join - outer join)

Lekce 03

dp\_s03\_l03

Co se v této lekci naučíte?

- srovnávat a popsat rozdíly mezi vnitřním a vnějším spojováním
- vytvořit a provést dotaz k použití levého vnějšího spojení
- vytvořit a provést dotaz k použití pravého vnějšího spojení
- vytvořit a provést dotaz k použití úplného vnějšího spojení

Proč se to učit?

- Až doposud všechna spojení vracela data, která splňovala podmínku spojení. Někdy však chceme vybrat nejenom data, která splňují podmínku spojení, ale také data, která ji nesplňují. To by mělo znít povědomě! Vnější spojení v ANSI-99 SQL tuto funkčnost umožní.

### Vnitřní a vnější spojení

V ANSI SQL-99 se spojení dvou nebo více tabulek, které vracejí pouze odpovídající řádky, nazývá vnitřní spojení.

Když spojení vrátí neodpovídající i odpovídající řádky, uzavřeno řádky, říkáme tomu vnější spojení.

Syntaxe vnějšího spojení používá pojmy "levý, úplný a pravý." Tato jména souvisí s **pořadím názvů tabulek** v klauzuli FROM v příkazu SELECT.

### LEVÉ A PRAVÉ VNĚJŠÍ SPOJENÍ (LEFT OUTER JOIN, RIGHT OUTER JOIN)

V tomto příkladě levého vnějšího spojení si všimněte, že název tabulky nalevo od slov "levé vnější spojení" uvádí "levá tabulka". Tento dotaz vrátí všechny odpovídající řádky a všechna příjmení zaměstnanců, i když nejsou přiřazeni do oddělení.

```
SELECT e.last_name, d.department_id, d.department_name
FROM employees e
LEFT OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

LAST_NAME	DEPT_ID	DEPT_NAME
King	90	Executive
Kochhar	90	Executive
...		
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Higgins	110	Accounting
Gietz	110	Accounting
Grant		

Pravé vnější spojení by vrátilo všechna ID a názvy oddělení, i kdyby v nich nebyli přiděleni žádní zaměstnanci.

```
SELECT e.last_name, d.department_id, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

LAST_NAME	DEPT_ID	DEPT_NAME
King	90	Executive
Kochhar	90	Executive
...		
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Higgins	110	Accounting
Gietz	110	Accounting
	190	Contracting

## ■ ÚPLNÉ VNĚJŠÍ SPOJENÍ (FULL OUTER JOIN)

Je možné vytvořit podmínku spojení tak, aby se načetly všechny odpovídající řádky a všechny ne-odpovídající řádky z obou tabulek ve spojení. Tento problém vyřeší úplné vnější spojení. Výsledky úplného vnějšího spojení zahrnují všechny řádky v obou tabulkách, i když neexistuje žádná shoda v druhé tabulce.

Uvedený příklad je úplné vnější spojení.

```
SELECT e.last_name, d.department_id, d.department_name
FROM employees e
FULL OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

LAST_NAME	DEPT_ID	DEPT_NAME
Whalen	10	Administration
Fay	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		
	190	Contracting

## ■ PŘÍKLAD

Sestavte spojení ke zobrazení seznamu zákazníků Global Fast Foods a jejich objednávek. Zahrňte všechny zákazníky, ať už měli zadanou objednávku nebo ne.

# 6. ODDÍL

## Obsah oddílu

- Skupinové (agregační funkce)
- Použití klauzulí GROUP BY a HAVING

## Skupinové funkce (agregační)

Lekce 01

dp\_s04\_I02

### Co se v této lekci naučíte

- definovat a uvést příklad sedmi skupinových funkcí: SUM, AVG, COUNT, MIN, MAX, STDDEV, VARA
- vytvořit a provádět SQL dotaz pomocí skupinových funkcí
- vytvořit a provádět skupinové funkce, které pracují pouze s numerickými datovými typy

### Proč se to učit?

- Co když budete psát článek do školních novin a k nějakému tvrzení budete chtít znát průměrný věk studentů na Vaší škole? Co byste museli udělat pro pořízení těchto informací? Můžete požádat všechny studenty, aby vám uvedli svůj věk v letech, měsících, a dnech a součet poté vydělít počtem studentů ve vaší škole. To je jeden způsob - velmi pomalý a obtížný - jak tyto informace získat. Co když tuto informaci potřebujete hned, protože máte termín do 15:00? Pak asi budete mít problém!
- Co když jsou všechna data narození studentů ve školní databázi v tabulce STUDENT? Pak by to bylo tak snadné! V této lekci se dozvíte o síle skupinových funkcí v SQL.

## Skupinové funkce

Následující skupinové funkce v SQL mohou pracovat s celou tabulkou nebo jen se specifickou skupinou řádků. Každá funkce vrací jeden výsledek.

- |         |          |
|---------|----------|
| • AVG   | • SUM    |
| • COUNT | • VARA   |
| • MIN   | • STDDEV |
| • MAX   |          |

**MIN:** Používá se na sloupce, které ukládají libovolný typ dat, a funkce vrátí minimální hodnotu.

**MAX:** Používá se na sloupce, které ukládají libovolný typ dat, a funkce vrátí maximální hodnotu.

**SUM:** Používá se na sloupce, které ukládají numerická data, a funkce vrací celkovou hodnotu či součet.

**AVG:** Používá se na sloupce, které ukládají numerická data, a funkce vrací průměrnou hodnotu.

**COUNT:** Vrací počet řádků

**VARIANCE:** Používá se na sloupce, které ukládají numerická data, a funkce počítá rozptyl dat kolem střední hodnoty. Pokud je např. průměrná známka v testu ve třídě 82 % a výsledky studentů jsou v rozmezí od 40 % do 100 %, rozptyl výsledků by byl větší než v případě, kdy jsou výsledky v rozmezí 78 % až 88 %.

**STDDEV:** Podobně jako variance hodnotí standardní odchylka rozptyl dat. Vezmeme-li dvě skupiny dat s přibližně stejnou střední hodnotou, platí, že čím větší je rozptyl, tím větší je standardní odchylka.

**Skupinové funkce se píšou v klauzuli SELECT. Skupinové funkce pracují se soubory řádků a vracejí jeden výsledek za celou skupinu.**

DEPT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
...	...
60	11000
60	8600
70	7000
10	4400

**MAX (SALARY)**  
24000

## PŘÍKLAD

Maximální plat v tabulce EMPLOYEES

```
SELECT MAX(salary)
FROM employees;
```

## NĚKOLIK DŮLEŽITÝCH VĚCÍ O SKUPINOVÝCH FUNKCÍCH:

- Skupinové funkce nelze použít v klauzuli WHERE
- Skupinové funkce ignorují hodnoty NULL. V níže uvedeném příkladu se hodnoty NULL nepoužily k nalezení průměrné míry přesčasů.

```
SELECT AVG(overtime_rate) FROM f_staffs;
```

- V klauzuli SELECT můžete mít více než jednu skupinovou funkci, na stejný nebo různé sloupce.
- Můžete také skupinovou funkci omezit na podmnožinu tabulky pomocí klauzule WHERE.
- Dvě skupinové funkce, MIN a MAX, se mohou použít s jakýmkoliv datovým typem.
- Pomocí těchto funkcí je možné najít jméno poslední osoby na seznamu, nejnižší plat nebo nejbližší datum nábory pracovníka. Například, je snadné najít osobu, jejíž jméno je první v abecedním seznamu zaměstnanců.

## PRAVIDLA PRO SKUPINOVÉ FUNKCE

- Skupinové funkce ignorují hodnoty Null.
- Skupinové funkce nelze použít v klauzuli WHERE.
- MIN a MAX lze použít s jakýmkoli datovým typem, SUM, AVG, STDDEV a VARIANCE lze použít pouze s numerickými datovými typy.

## Použití klauzulí GROUP BY a HAVING

Lekce 02

dp\_s05\_01

### Co se v této lekci naučíte?

- vytvořit a provést SQL dotaz pomocí GROUP BY
- vytvořit a provést SQL dotaz pomocí GROUP BY ... HAVING
- vytvořit a provést GROUP BY na více než jednom sloupci
- vnořit skupinové funkce

### Proč je třeba se to učit?

- Co když budete chtít vědět průměrnou výšku všech studentů? Můžete zapsat dotaz, který vypadá takto: `SELECT AVG(height) FROM students;`
- Co když jste ale chtěli vědět průměrnou výšku studentů podle ročníků? Zatím byste museli napsat několik různých SQL příkazů, abyste dostali výsledek:

```
SELECT AVG(height) FROM students WHERE year_in_school = 10;
```

```
SELECT AVG(height) FROM students WHERE year_in_school = 11;
```

```
SELECT AVG(height) FROM students WHERE year_in_school = 12;
```

A tak dále! Pro zjednodušení takovýchto problémů stačí použít jen jeden příkaz a klauzule GROUP BY a HAVING.

## GROUP BY

Pomocí klauzule GROUP BY rozdělíme řádky v tabulce do menších skupin. Poté můžete použít skupinové funkce a získat souhrnné informace za každou skupinu.

V uvedeném příkaze SELECT se řádky seskupují podle department\_id. Na každou skupinu poté automaticky použijeme funkci AVG.

```
SELECT MAX(salary)
FROM employees
GROUP BY department_id;
```

DEPT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
...	...

MAX (SALARY)
24000
9000
5800
...

Co kdybychom chtěli zjistit maximální plat zaměstnanců v každém oddělení? Použijeme klauzuli GROUP BY, kde uvedeme, podle jakého sloupce se mají řádky seskupit.

Ale jak můžeme zjistit, který maximální plat patří do kterého oddělení? Obvykle chceme sloupec GROUP BY zahrnout do seznamu SELECT.

DEPT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
...	...

DEPT_ID	MAX (SALARY)
90	24000
60	9000
...	...

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id;
```

Skupinové funkce vyžadují, aby každý sloupec, který je uvedený v klauzuli SELECT ale není součástí skupinové funkce, byl uvedený v klauzuli GROUP BY.

Co je špatně v tomto příkladě?

```
SELECT job_id, last_name, AVG(salary)
FROM employees
GROUP BY job_id;
```

## COUNT



```
SELECT count(first_name), shirt_color
FROM students
GROUP BY shirt_color
```

Tento příklad ukazuje, kolik studentů nosí košile jaké barvy.

Pamatujte si, že skupinové funkce ignorují hodnoty null, takže pokud nějaký student nemá křestní jméno, nebude zahrnut do COUNT. Samozřejmě je to nepravděpodobné, ale při konstrukci SQL příkazů musíme myslet na všechny možnosti.

Bylo by lepší začít takto: SELECT COUNT(\*), shirt\_color

Můžeme také použít klauzuli WHERE a vyřadit řádky ještě před rozčleněním zbývajících řádků do skupin.

```
SELECT department_id, MAX(salary)
FROM employees
WHERE last_name <> 'King'
GROUP BY department_id;
```

## DŮLEŽITÉ POKYNY PRO POUŽÍVÁNÍ GROUP BY KLAUZULE:

- Pokud zahrnete skupinovou funkci (AVG, SUM, COUNT, MAX, MIN, STDDEV, VARIANCE) do klauzule SELECT a jakékoliv jiné jednotlivé sloupce, musí se každý jednotlivý sloupec uvést také v klauzuli GROUP BY.

- V klauzuli GROUP BY nelze použít aliasy sloupců.
- Klauzule WHERE vylučuje řádky ještě předtím, než jsou rozděleny do skupin.

## Skupiny uvnitř skupin (podskupiny)

Někdy je třeba rozdělit skupiny do menších skupin. Například potřebujete rozdělit všechny zaměstnance podle oddělení a pak v rámci každého oddělení ještě podle pracovní funkce.

### PŘÍKLAD

Tento příklad ukazuje, kolik zaměstnanců dělá jakou práci v rámci jednotlivých oddělení.

```
SELECT department_id, job_id, count(*)
FROM employees
WHERE department_id > 40
GROUP BY department_id, job_id;
```

## Vnoření skupinových funkcí

Když používáme GROUP BY, můžeme skupinové funkce vnořit do hloubky dvou.

### PŘÍKLAD:

Kolik hodnot vrátí tento dotaz? Odpověď je: jednu - dotaz najde průměrný plat pro každé oddělení a pak z tohoto seznamu vybere jednu největší hodnotu.

```
SELECT max(avg(salary))
FROM employees
GROUP BY department_id;
```

## HAVING

Předpokládejme, že chceme najít maximální plat v každém oddělení, ale pouze u těch oddělení, která mají více než jednoho zaměstnance? Co je špatně na tomto příkladě?

```
SELECT department_id, MAX(salary)
FROM employees
WHERE COUNT(*) > 1
GROUP BY department_id;
```

**Chyba: ORA-00934: group function is not allowed here**

Tak jako jste použili klauzuli WHERE k omezení vybraných řádků můžete použít klauzuli HAVING k omezení skupin.

V dotazu, který obsahuje klauzule GROUP BY a HAVING, se nejprve seskupí řádky, poté se použijí skupinové funkce a poté se zobrazí pouze ty skupiny, které odpovídají klauzuli HAVING.

Klauzule WHERE slouží k omezení řádků, HAVING slouží k omezení skupin, které vrací klauzule GROUP BY.

Ačkoli klauzule HAVING může v příkazu SELECT předcházet klauzuli GROUP BY, doporučuje se použít tyto klauzule v uvedeném pořadí. Klauzule ORDER BY (pokud se použije) je Vždy poslední!



# 7. ODDÍL

## Obsah oddílu

- Základy vnořených dotazů (poddotaz, vnořený dotazies)

## Základy vnořených dotazů (poddotazů)

Lekce 01

dp\_s06\_01

### Co se v této lekci naučíte?

- definovat a vysvětlit účel vnořených dotazů pro získávání dat
- vytvořit a vykonat jednořádkový vnořený dotaz ve frázi WHERE
- rozlišit jednořádkový a víceřádkový vnořený dotaz
- rozlišit párový a nepárový vnořený dotaz
- použít EXIST a NOT EXISTS operátory v dotazu

### Proč se to učit?

- Známy se vás ptá, zda můžete jít do kina, ale předtím, než byste mohl odpovědět "ano" či "ne", musíte se poradit s vašimi rodiči? Někdo vás prosí o odpověď na příklad z matematiky, ale předtím, než ji můžete dát, musíte sám příklad vyřešit?
- Dotazující se rodiče, či řešení příkladu z matematiky, jsou příklady vnořených dotazů. V SQL umožňují vnořené dotazy najít informaci, kterou potřebujeme.

## Vnořené dotazy - celkový pohled

V průběhu studia SQL jste se naučili psát dotazy k tomu, aby získaly data z databáze.

Co když chcete psát dotaz jen proto, abyste zjistili všechny informace, které ještě nemáte pro sestavení nějakého jiného dotazu? Můžete řešit tento problém kombinací dvou dotazů, umístěním jednoho dotazu uvnitř jiného dotazu. Vnitřní dotaz je nazvaný "vnořený dotaz". vnořený dotaz hledá informaci, kterou neznáte. Vnější dotaz používá tuto informaci ke zjištění toho, co potřebujete dále vědět.

Schopnost spojit dva dotazy do jednoho může být velmi užitečné, když potřebujete vybrat řádky z tabulky na základě podmínky, která závisí na datech stejné tabulky.

## VNOŘENÝ DOTAZ - PŘÍKLAD

Vnořený dotaz je příkaz SELECT, který je vložen do zápisu dalšího příkazu SELECT.

Vnořený dotaz se vykoná jednou před provedením hlavního dotazu. Výsledek vnořeného dotazu je použit hlavním či vnějším dotazem. Vnořené dotazy mohou být umístěny v řadě SQL frází, včetně frází WHERE, HAVING a FROM.

## VNOŘENÝ DOTAZ SYNTAXE JE:

```
SELECT vybraný_seznam_hodnot
FROM tabulka
WHERE výraz operátor
      (SELECT vybraný_seznam
       FROM tabulka);
```

Příkaz dotazu závorkách (rámečku) je vnitřní dotaz nebo také 'vnořený dotaz'.

## PRAVIDLA PRO POUŽITÍ VNOŘENÝCH DOTAZŮ:

- vnořený dotaz je uzavřený v závorkách
- vnořený dotaz je umístěný na pravé straně porovnávací podmínky
- vnější a vnitřní dotazy mohou dostat data z rozdílných tabulek
- pro příkaz výběru (dotaz) může být použita jen jedna fráze ORDER BY; jestliže je použita, musí být ve vnějším dotazu jako poslední fráze; vnořený dotaz nemůže mít svou vlastní frázi ORDER BY
- jediný limit pro počet vnořených dotazů je velikost vyrovnávací paměti používanou dotazem

## Dva druhy vnořených dotazů:

- Jednořádkový vnořený dotaz - ten používá jednořádkové operátory (>, =, >=, < <>, <=) a vrátí jen jeden záznam z vnitřního dotazu.
- Víceřádkový vnořený dotaz - ten používá víceřádkové operátory (IN, ANY, ALL) a může vrátit víc než jeden záznam z vnitřního dotazu.

## PŘÍKLAD - VNOŘENÝ DOTAZ

Co kdyby jste chtěli najít jména členů personálu společnosti s rychlým občerstvením, kteří se narodili později, než Monique Tuttle? Co potřebujeme zjistit jako první? Kdy se narodila Monique? Jakmile znáte její datum narození, pak můžete vybrat ty členy personálu, jejichž data narození jsou větší, než její.

```
SELECT staff_id, first_name, last_name, birth_date
FROM f_staffs
WHERE birth_date >= (SELECT birth_date
                     FROM f_staffs
                     WHERE last_name = 'Tuttle');
```

## Vícesloupcový vnořený dotaz

Vnořené dotazy mohou používat (vybírat) jeden nebo více sloupců. Jestli používají víc než jeden sloupec, nazývají se vícesloupcové vnořené dotazy. Vícesloupcový vnořený dotaz může být buď s párovým porovnáním nebo nepárovým porovnáním.

## PŘÍKLAD:

Příklad ukazuje vícesloupcový párový vnořený dotaz - zvýrazněný červeně

```
SELECT employee_id,manager_id, department_id
FROM employees
WHERE (manager_id,department_id) IN      (SELECT manager_id,department_id
                                           FROM employees
                                           WHERE employee_id IN (149,174))
AND employee_id NOT IN (149,174)
```

Dotaz zobrazuje seznam zaměstnanců, jejichž manažer a oddělení jsou stejné, jako manažer a oddělení zaměstnanců s identifikačním číslem 149 nebo 174.

Nepárový vícesloupcový vnořený dotaz také používá (vybírání) víc než jeden sloupec, ale srovnává je jeden po druhém, takže srovnání se provede různými vnořenými dotazy. Budete tak potřebovat psát jeden vnořený dotaz pro každý sloupec, který chcete porovnávat při vykonávání nepárového vícesloupcového poddotazu.

### PŘÍKLAD:

Příklad ukazuje vícesloupcový nepárový vnořený dotaz se vnořeným dotazem, který je zvýrazněný červeně.

```
SELECT employee_id,manager_id,department_id
FROM employees
WHERE manager_id IN      (SELECT manager_id
                           FROM employees
                           WHERE employee_id IN (174,199))
AND department_id IN    (SELECT department_id
                           FROM employees
                           WHERE employee_id IN(174,199))
AND employee_id NOT IN (174,199) ;
```

Výsledek dotazu je seznam zaměstnanců, kteří mají manager\_id a department\_id stejné se zaměstnanci s čísly 174 nebo 199.

## EXIST & NOT EXIST ve vnořených dotazech

Fráze EXIST a jeho opačná fráze NOT EXIST jsou další dvě klauzule, které mohou být použity při testování odpovídajících vnořených dotazů. EXISTS testuje na hodnotu TRUE, nebo odpovídající výsledek vnořeného dotazu.

Jestliže chcete vidět kolik zaměstnanců bylo jen zaměstnanci a nebyli zároveň manažeři, můžete použít NOT EXIST:

```
SELECT count(*)
FROM employees t1
WHERE NOT EXISTS (SELECT NULL
                  FROM employees t2
                  WHERE t2.manager_id = t1.employee_id ) ;
```

V tomto příkladu vnořený dotaz vybírá NULL hodnotu proto, abychom zajistili test výskytu známů vnořeného dotazu, který má vrátit něco jiného, než TRUE nebo FALSE.

Jestliže stejný dotaz je vykonaný s NOT IN namísto NOT EXISTS, výsledek bude velmi odlišný. Výsledek tohoto dotazu ukazuje, že tam nejsou žádní zaměstnanci, kteří nejsou manažeři, takže

všichni zaměstnanci jsou zároveň manažeři. Ale my již víme, že to není pravda. Co způsobilo tento výsledek?

```
SELECT count(*)  
FROM employees t1  
WHERE t1.employee_id NOT IN (SELECT t2.manager_id  
                             FROM employees t2 );
```

Příčina špatného výsledku je kvůli NULL hodnotě vrácené vnořeným dotazem. Jeden ze záznamů v tabulce zaměstnanců nemá manažera, a to dělá celý výsledek špatný. Vnořený dotaz může vrátit tři hodnoty: PRAVDA, NEPRAVDA a NEZNÁMOU hodnotu. NULL ve výsledku vnořeného dotazu bude vracet UNKNOWN hodnotu, kterou Oracle nemůže vyhodnotit, takže to nejde.

## 8. ODDÍL

### Obsah oddílu:

- Příkazy DML
- Ostatní objekty databáze

## Příkazy DML

Lekce 01

### Co se v této lekci naučíte:

- Uvést příklady, proč je důležité mít možnost měnit data v databázi
- Sestavit a spustit příkazy INSERT, UPDATE, DELETE

### Proč se to naučit?

- V podnikání jsou databáze dynamické. Jsou neustále v procesu vkládání, aktualizace a odstraňování dat. Zamyslete se, kolikrát se mění školní databáze studentů ze dne na den a rok co rok. Pokud by nedošlo ke změnám, databáze by rychle ztratila svou užitečnost.
- DML příkazy umožňují uživatelům provádět změny v databázi. Spuštění jednoho DML příkazu je považováno za transakci.

## INSERT

INSERT slouží k přidání nových řádků do tabulky. Příkaz vyžaduje tři hodnoty.

### ■ INSERT (SYNTAXE)

Syntaxe ukazuje použití příkazu INSERT pro přidání nového zákazníka do tabulky Global Fast Foods. Tento příkaz jednoznačně uvádí každý sloupec tak, jak je zobrazen v tabulce. Hodnoty pro každý sloupec jsou uvedeny ve stejném pořadí. Všimněte si, že číselné hodnoty nejsou uzavřeny v jednoduchých uvozovkách.

```
INSERT INTO copy_f_customers
(id, first_name, last_name, address, city, state, zip, phone_number)
VALUES (145, 'Katie', 'Hernandez', '92 Chico Way', 'Los Angeles', 'CA',
98008, 8586667641);
```

## UPDATE

Příkaz UPDATE se používá k úpravě stávajících řádků tabulky. Vyžaduje čtyři hodnoty.

### ■ UPDATE (SYNTAXE)

Uvedený příklad ukazuje použití příkazu UPDATE, který změní telefonní číslo jednoho zákazníka v databázi Global Fast Foods. Všimněte si, že v této transakci je použita tabulka copy\_f\_customers.

```
UPDATE copy_f_customers
SET phone_number='4475582344'
WHERE id=123;
```

## DELETE

Příkaz DELETE se používá k odstranění existujících řádků v tabulce. Příkaz vyžaduje dvě hodnoty

### ■ DELETE (SYNTAXE)

Uvedený příklad používá databázi Global Fast Foods k vymazání jednoho řádku, zákazníka, jehož ID je 123.

```
DELETE FROM copy_f_customers
WHERE ID= 123;
```

## Ostatní databázové objekty

### Lekce 02

#### Pohled (VIEW)

Pohled je databázový objekt, který zobrazuje data jako tabulku. Nicméně, pohledy nejsou "skutečné" tabulky. Jsou to logické reprezentace existující tabulky nebo jiného pohledu. Pohledy neobsahují žádné vlastní údaje. Fungují jako okno, jímž lze data z tabulek vidět nebo změnit

#### JAK VYTVOŘIT POHLED

```
CREATE VIEW view_employees
AS SELECT first_name, last_name, email
FROM employees
WHERE employee_id BETWEEN 100 and 124;
```

#### SEQUENCE

SEQUENCE je sdílený objekt, který slouží k automatickému generování unikátních čísel.

Protože se jedná o sdílený objekt, přístup k němu může mít více uživatelů. Obvykle sekvence slouží k vytvoření primárního klíče.

Sekvence čísel je uložena a generována nezávisle na tabulkách. Proto může být stejná sekvence použita pro více tabulek.

#### JAK VYTVOŘIT SEKVENCI:

```
CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

#### PŘÍKLAD POUŽITÍ SEQUENCE

Předpokládám, že teď chcete najmout zaměstnance pro nové oddělení. Příkaz INSERT, který vloží všechny nové zaměstnance, může obsahovat následující kód:

```
INSERT INTO employees (employee_id, ...)
VALUES (employees_seq.NEXTVAL, ...);
```

## CVIČNÉ PŘÍKLADY PRO MODULY DATABÁZOVÝ NÁVRH PROGRAMOVÁNÍ SQL

Podpora výuky databázových systémů na SOŠ, založené na  
technologiích společnosti ORACLE.

Publikace vznikla v rámci projektu CZ.1.07/1.1.07/02.007,  
Podpora výuky databázových systémů na středních odborných  
školách, založené na technologiích společnosti ORACLE.

© 2011 Vydała Střední průmyslová škola elektrotechniky  
informatiky a řemesel, příspěvková organizace, Křižíkova 1258,  
Frenštát p. R., [www.spsfren.cz](http://www.spsfren.cz)

Cvičné příklady jsou synchronizovány s mezinárodním vzdělávacím  
programem Oracle Academy. Více informací na [academy.oracle.com](http://academy.oracle.com)  
nebo na portálu [ucimedatabase.cz](http://ucimedatabase.cz).

**Manager projektu:** Mgr. Richard Štěpán

**Překlad:** Oracle Czech, Bc. Tomáš Romanovský,  
Mgr. Dana Mikesková, Mgr. Markéta Kytková

**Metodik:** Bc. Tomáš Romanovský

**Jazyková korektura:** Mgr. Pavlína Chovancová

**Sazba:** Bc. Tomáš Romanovský

**Obálka:** Bc. Tomáš Romanovský

**Tisk:** Reprografické studio LWR GRAPHIC

Žádná část této publikace nesmí být publikována a šířena žádným způsobem a v žádné podobě  
bez výslovného souhlasu vydavatele.

*Zvláštní poděkování patří společnosti Oracle Czech za  
dlouholetou podporu vzdělávání v oblasti databázových  
technologií a za spolupráci při vytváření této publikace.*

*Autoři projektu*

## Obsah

CVIČENÍ 1.....	4
NÁVRH DATABÁZE ODDÍL 1 - 3.....	4
CVIČENÍ 2.....	7
NÁVRH DATABÁZE ODDÍL 5, PŘENOSITELNOST VZTAHU.....	7
NÁVRH DATABÁZE ODDÍL 5, ŘEŠENÍ VZTAHU M:M.....	7
NÁVRH DATABÁZE ODDÍL 6, NORMALIZACE DB.....	7
CVIČENÍ 3.....	10
NÁVRH DATABÁZE ODDÍL 6, NORMALIZACE DATABÁZE.....	10
NÁVRH DATABÁZE ODDÍL 7, OBLOUKY.....	10
NÁVRH DATABÁZE ODDÍL 7, MODELOVÁNÍ HISTORICKÝCH DAT.....	10
CVIČENÍ 4.....	14
NÁVRH DATABÁZE ODDÍL 15, ANATOMIE SQL.....	14
NÁVRH DATABÁZE ODDÍL 16, PRÁCE SE SLOUPCI A ŘÁDKY.....	14
NÁVRH DATABÁZE ODDÍL 16, RELAČNÍ OPERÁTORY.....	14
NÁVRH DATABÁZE ODDÍL 17, LOGICKÉ OPERÁTORY, PŘEDNOSTI.....	14
NÁVRH DATABÁZE ODDÍL 17, TRÍDĚNÍ ŘÁDKŮ.....	14
CVIČENÍ 5.....	20
PROGRAMOVÁNÍ SQL ODDÍL 1, MANIPULACE SE ZNAKY.....	20
PROGRAMOVÁNÍ SQL ODDÍL 1, ČÍSELNÉ FUNKCE.....	20
PROGRAMOVÁNÍ SQL ODDÍL 1, DATUMOVÉ FUNKCE.....	20
PROGRAMOVÁNÍ SQL ODDÍL 2, KONVERZNÍ FUNKCE.....	20
PROGRAMOVÁNÍ SQL ODDÍL 2, NULL FUNKCE.....	20
PROGRAMOVÁNÍ SQL ODDÍL 1, PODMÍNĚNÉ VÝRAZY.....	20
CVIČENÍ 6.....	26
PROGRAMOVÁNÍ SQL ODDÍL 3, CROSS A NATURAL JOIN.....	26
PROGRAMOVÁNÍ SQL ODDÍL 3, KLAUZULE JOIN.....	26
PROGRAMOVÁNÍ SQL ODDÍL 3, INNER & OUTER JOIN.....	26
PROGRAMOVÁNÍ SQL ODDÍL 4, GROUP FUNCTION.....	26
PROGRAMOVÁNÍ SQL ODDÍL 4, COUNT, DISTINCT, NVL.....	26
CVIČENÍ 7.....	29
PROGRAMOVÁNÍ SQL ODDÍL 6, GROUP BY & HAVING.....	29
PROGRAMOVÁNÍ SQL ODDÍL 6, PODDOTAZY.....	29
PROGRAMOVÁNÍ SQL ODDÍL 6, JEDNOŘÁDKOVÉ PODDOTAZY.....	29
PROGRAMOVÁNÍ SQL ODDÍL 6, VÍCEŘÁDKOVÉ PODDOTAZY.....	29
CVIČENÍ 8.....	32
PROGRAMOVÁNÍ SQL ODDÍL 7, PŘÍKAZ INSERT.....	32
PROGRAMOVÁNÍ SQL ODDÍL 7, PŘÍKAZ UPDATE, DELETE.....	32
PROGRAMOVÁNÍ SQL ODDÍL 7, DEFAULT, MERGE, MULTI i.....	32
PROGRAMOVÁNÍ SQL ODDÍL 8, TVORBA TABULEK.....	32
PROGRAMOVÁNÍ SQL ODDÍL 10, MODIFIKACE TABULEK.....	32



## CVIČENÍ 1

## NÁVRH DATABÁZE

## ODDÍL 1 – 3

## ÚKOL #1

Napište věty ERDish pro každý vztah popsany v modelu ERD společnosti DJ on Demand (ERD viz příloha)

*Write ERDish sentences for each relationship documented in the DJs on Demand ERD. Refer to Section 0, Database Design Instructor Course Resources.*

Each CLIENT may be owner of one or more EVENTs.  
Each EVENT must be owned by exactly one CLIENT.

Each EVENT must be hold at PRIVATE HOME or PUBLIC SPACE.  
Each PRIVATE HOME may be venue for one or more EVENTs.  
Each PUBLIC SPACE may be venue for one or more EVENTs.

Each THEME may be catogorize one or more EVENTs.  
Each EVENT may be categorized by exactly one THEME.

Each EVENT may be the source of one or more JOB ASSIGNMENTs.  
Each JOB ASSIGNMENT must be for of exactly one EVENT.

Each PARTNER may be responsible for one or more JOB ASSIGNMENTs.  
Each JOB ASSIGNMENT must be for one PARTNER.

Each MANAGER may be supervise one or more PARTNERs.  
Each PARTNER may be supervised by exactly one MANAGER.

Each PLAY LIST ITEM must be for one SONG.  
Each SONG may be an item on one or more PLAY LIST ITEMs.

Each EVENT may be require one or more PLAY LIST ITEMs.  
Each PLAY LIST ITEM must be for one EVENT.

Each TYPE may be classify one or more SONGs.  
Each SONG may be classified by exactly one TYPE.

Each SONG may be located on one or more TRACK LISTINGs.  
Each TRACK LISTING must be locate one SONG.

Each EVENT must be classified by exactly one TYPE.  
Each TYPE may be classify one or more EVENTs.

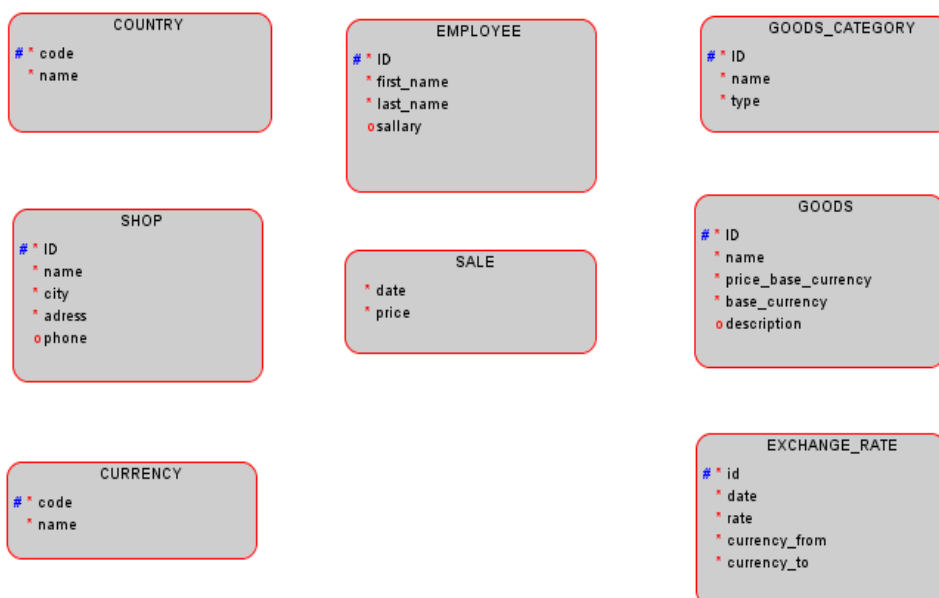
## ÚKOL #2

Na základě následujícího scénáře určete entity a atributy. Nakreslete rámečky s entitami a seznamem atributů. Označte jedinečné atributy pomocí hash značky (#) a pokuste se určit volitelnost každého atributu („NULL“).

Given the following scenario, choose the entities and attributes. Draw the soft boxes with the entities and list the attributes underneath. Mark UNIQUE attributes with a hash mark (#) and try to determine optionality of each attribute.

**Scénář:** **Moonlight Coffees** is a fast growing chain of high quality coffee shops with currently over 500 shops in 12 countries of the world. Shops are located at first-class locations such as major shopping, entertainment and business areas, airports, railway stations and museums. **Moonlight Coffees** has some 9,000 employees.

All shops serve coffees, teas, soft drinks and various kinds of pastries. Most shops sell nonfoods, like postcards and sometimes even theater tickets. Shop management reports sales figures on a daily basis to Headquarters, in local currency. **Moonlight** uses an internal exchange rates list that is changed monthly. Since January 1, 1999, the European Community countries must report in Euros.



## ÚKOL #3

Zkopírujte a vložte scénář **Summit Sporting** (viz níže) do textového dokumentu a zvýrazněte všechna podstatná jména (entity a vztahy).

*Copy and paste the Summit Sporting Goods scenario (below) into a text document and underline all nouns before the chat session. Have this ready to use during that chat.*

Underlined – entity; marked - atribut

"I'm a manager of a sporting-goods wholesale company that operates worldwide to fill orders from retail sporting-goods stores. The stores are our customers (some of our people prefer to call them our clients). Right now we have 15 customers worldwide, but we're trying to expand our customer base by about 10% each year starting this year. Our two biggest customers are Big John's Sports Emporium in San Francisco and Womansports in Seattle. For each customer, we must track an ID and a name.

We may track an address (including the city, state, zip code, and country) and phone number. We maintain warehouses in different regions to best fill the order of our customers. For each order, we must track an ID. We may track the date ordered, date shipped, and payment type when the information is available."

"Right now we have the world divided into five regions: North America, South America, Africa/Middle East, Asia, and Europe. That's all we track; just the ID and name. We try to assign each customer to a region so we'll generally know the best location from which to fill each order. Each warehouse must have an ID. We may track an address (including the city, state, zip code, and country) and phone number. We currently have only one warehouse per region, but we're hoping to have more soon.

I manage the order-entry functions for our wholesale sporting-goods business. My department is responsible for placing and tracking the orders when our customers call. For each department, we must track the ID and name. Sometimes, our customers just mail us the orders when they are not in a rush, but most often they call us or fax us an order. We are hoping to expand our business by providing immediate turnaround of order information to our clients. Do you think we can put this application on the Web?"

"We can promise to ship by the next day as long as the goods are in stock (or inventory) at one of our warehouse locations. When the information is available, we track the amount in stock, the reorder point, maximum stock, a reason as to why we are out of stock, and the date we restocked the item. When the goods are shipped, we fax the shipping information automatically through our shipping system. No, I don't manage that area. My department just ensures that our customers have the correct billing information and verifies that their account is in good credit standing. We may also record general comments about a customer.

We do make sure that all the items they have requested are in stock. For each item we track an ID. We may also track the item price, quantity, and quantity shipped if the information is available. If they are in stock, we want to process the order and tell our clients what the order ID is and how much their order total is. If the goods are not in stock, the customer tells us whether we should hold the order for a full shipment or process the partial order."

"The accounting department is responsible for maintaining the customer information, especially for assigning new customer IDs. My department is allowed to update the customer information only when an order is placed and the billing or ship-to address has changed. No, we are not responsible for collections. That's all handled by accounts receivable. I also think that the sales reps get involved because their commission depends on customers who pay! For each sales rep, or employee, we must know the ID and last name. Occasionally we need to know the first name, user ID, start date, title, and salary. We may also track the employee's commission percent and any comments about the individual.

Our order-entry personnel are well versed in our product line. We hold frequent meetings with marketing so they can inform us of new products. This results in greater customer satisfaction because our order-entry operators can answer a lot of questions. This is possible because we deal with a few select customers and maintain a specialty product line. For each product, we must know the ID and name. Occasionally we must also know the description, suggested price, and unit of sale. We would also like the ability to track very long descriptions of our products and pictures of our products, when it is necessary."

## CVIČENÍ 2

NÁVRH DATABÁZE

ODDÍL 5, PŘENOSITELNOST VZTAHU

NÁVRH DATABÁZE

ODDÍL 5, ŘEŠENÍ VZTAHU M:M

NÁVRH DATABÁZE

ODDÍL 6, NORMALIZACE DB

## ÚKOL #1

Pro níže uvedené příklady nakreslete rámečky. Nakreslete vztahové linky a správně popište vztah v obou směrech. Kde je nutné vyznačte nepřenositelnost vztahu.

b. Do každého pokoje se může ubytovat jeden nebo více hostů. Každý host může být ubytován pouze v jednom pokoji.

d. V každém hotelu se může ubytovat jeden nebo více hostů. Každý host může být ubytován v jednom nebo více hotelích.

f. Každé oblečení musí mít pouze jednu cenu. Každá cena může být pro jedno nebo více oblečení.

h. Každý automobil musí používat pouze jeden rozměr pneumatik. Jedno nebo více aut může použít stejný rozměr pneumatik.

j. Každá osoba musí mít pouze jednu krevní skupinu. Každou krevní skupinu může mít jedna nebo více osob.

l. Každého studenta může učit jeden nebo více učitelů. Každý učitel může učit jednoho nebo více studentů.

n. Každý otisk prstu musí patřit pouze jedné osobě. Každá osoba musí mít pouze jeden otisk prstu.

Draw softboxes for each of the following. Draw relationship lines and correctly label each relationship in both directions. Indicate non-transferability when appropriate.

b. Each room may house one or more guests. Each guest may stay in one and only one room.

d. Each hotel may be the host of one or more guests. Each guest may be hosted in one or more hotels.

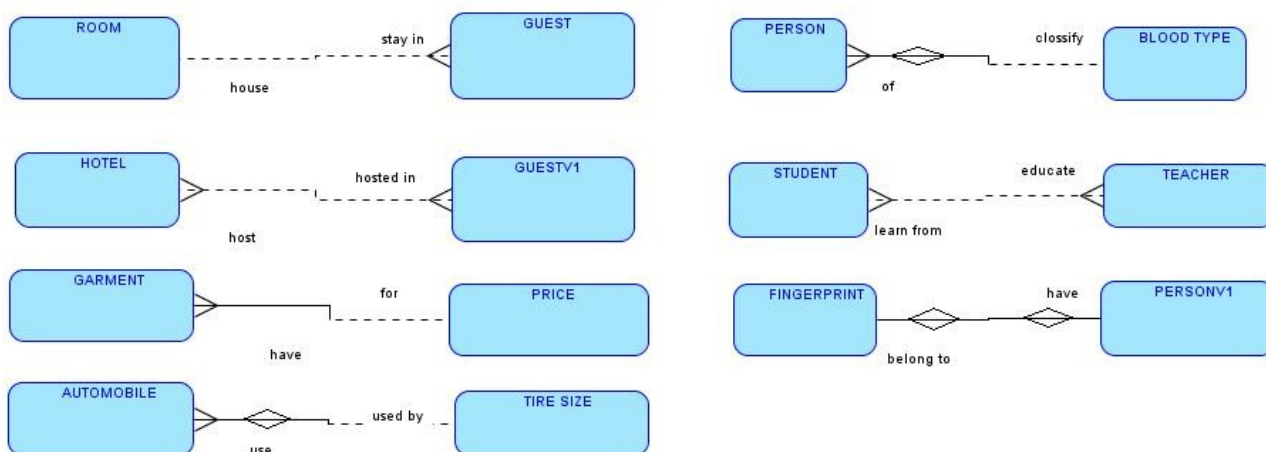
f. Each garment must have one and only one price. Each price may be for one or more garments.

h. Each automobile must use one and only one tire size. Each tire size may be used by one or more automobiles.

j. Each person must be of one and only one blood type. Each blood type may classify one or more.

l. Each student may learn from one or more teachers. Each teacher may educate one or more students.

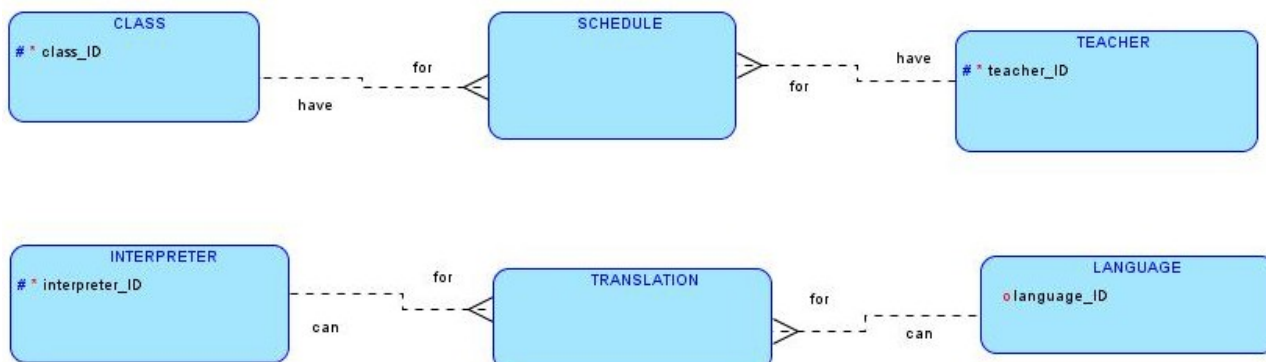
n. Each fingerprint must belong to one and only one person. Each person must have one and only one fingerprint.



## ÚKOL #2

Vyřešte vztah M:M mezi učitelem a třídou, stejně jako mezi překladatelem a jazykem. Pro každou průnikovou entitu vymyslete další atributy jako UID.

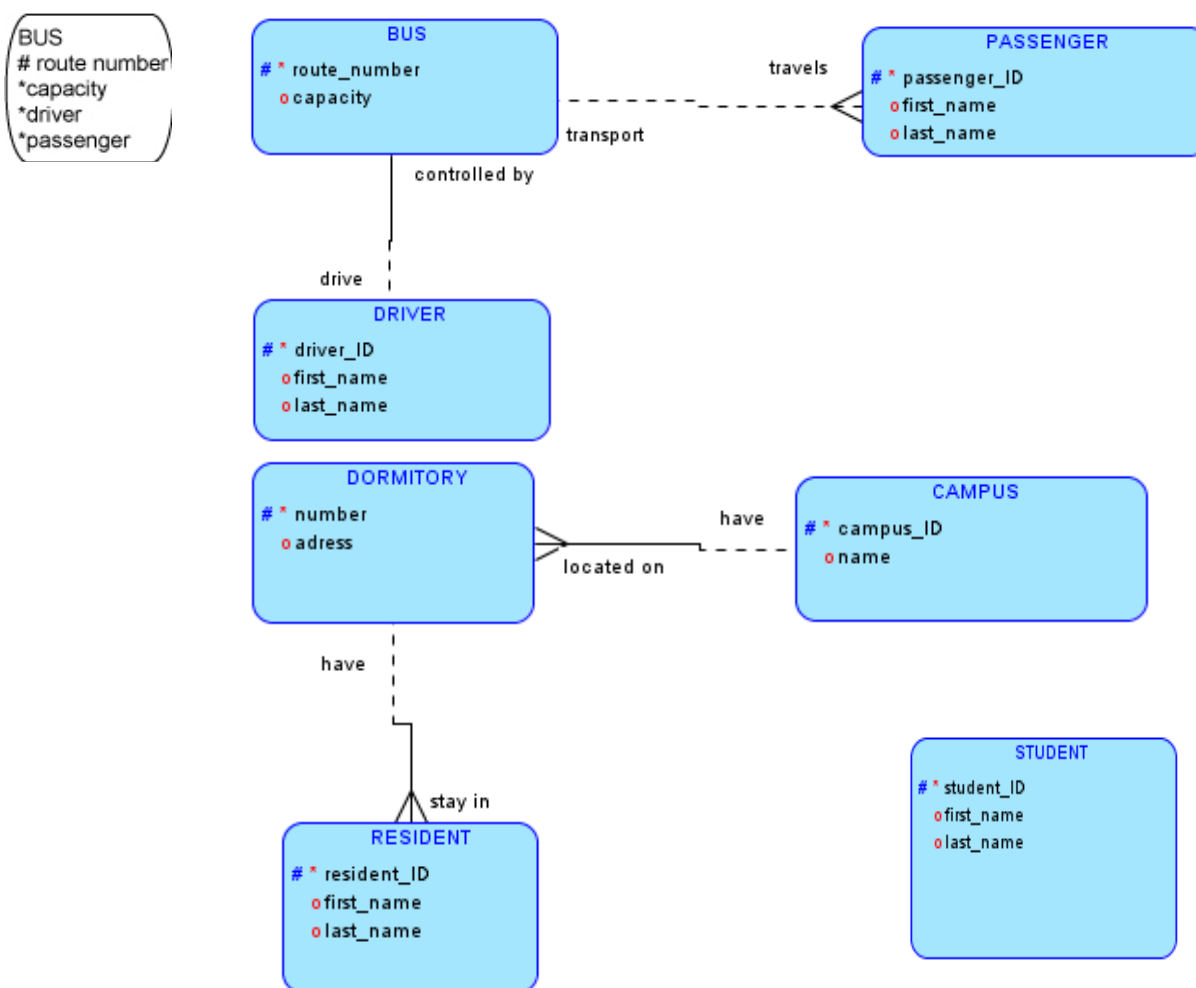
Resolve the M:M between TEACHER and CLASS as well as INTERPRETER and LANGUAGE. For each intersection entity, think of additional attributes like a UID.



## ÚKOL #3

Zkontrolujte, zda je každý ERD model v 1NF. Pokud tomu tak není, proveďte potřebné změny.

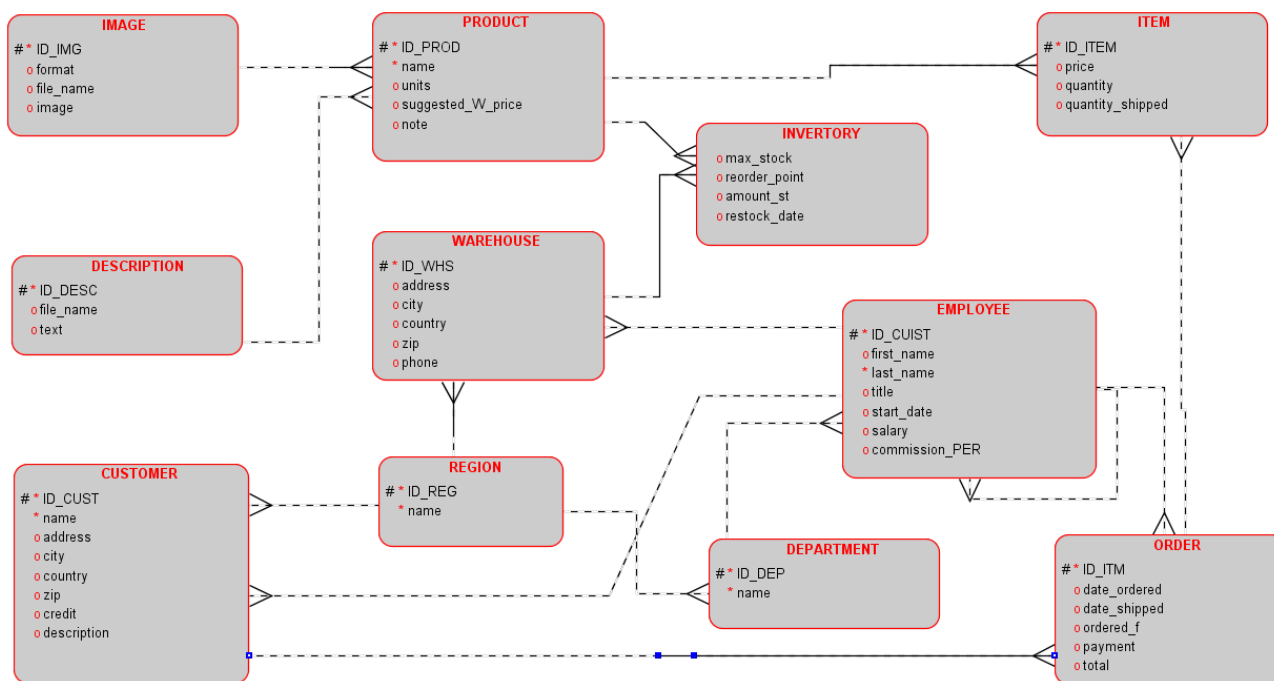
Check to see if each ERD is in 1NF. If not, make the necessary changes to correct it.



## ÚKOL #4

Pomocí scénáře “Summit Sporting Goods”, který je popsán výše (Cvičení 1, úkol #3), nakreslete ERD s využitím entit a atributů projednaných na posledním chatu. Pokuste se vyřešit všechny M:N vztahy pomocí průnikových entit.

Using the “Summit Sporting Goods” scenario described above, draw an ERD using the entities and attributes discussed on your last chat. Try to resolve any many-to-many relationships with intersection entities.



## CVIČENÍ 3

NÁVRH DATABÁZE

ODDÍL 6, NORMALIZACE DATABÁZE

NÁVRH DATABÁZE

ODDÍL 7, OBLOUKY

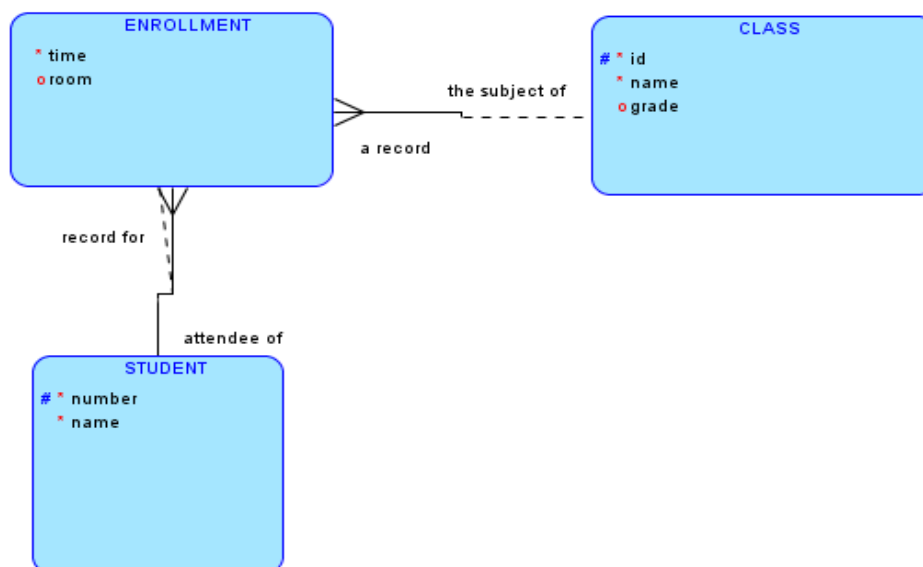
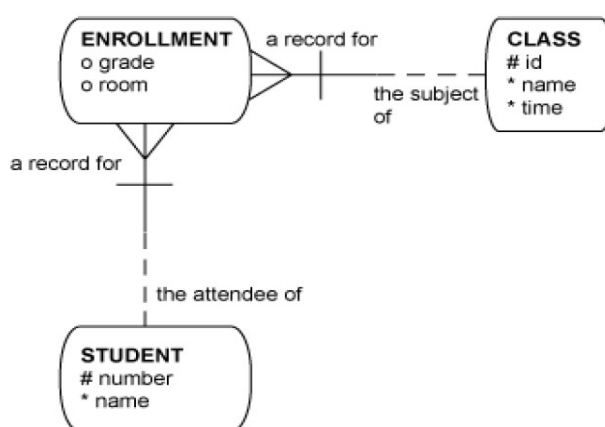
NÁVRH DATABÁZE

ODDÍL 7, MODELOVÁNÍ HISTORICKÝCH DAT

## ÚKOL #1A

„Zápis do třídy“ je průnikovou entitou, která řeší vztah M:N mezi entitami STUDENT a CLASS. Jsou v následujícím ERD uplatněna pravidla 2NF? Pokud ne, opravte je.

Class Enrollment is the intersection entity that resolves the M:M between STUDENT and CLASS. Does the ERD follow the rules of Second Normal Form? If you spot a violation, correct it.



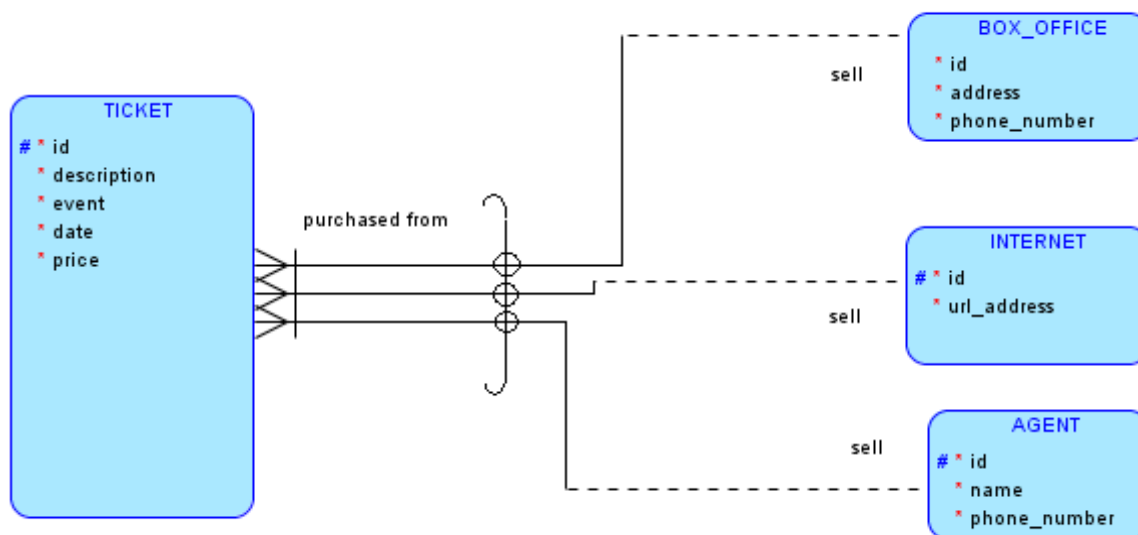
## ÚKOL #1B

Lístek může být zakoupený od prodejce, v pokladně nebo na Internetu. Na lístku je zobrazen popis, událost, datum a cena. Prodejce má jméno a telefonní číslo. Pokladna má adresu a telefonní číslo. Internet má URL adresu.

Nakreslete entity a vyznačte exkluzivitu vztahu.

A show ticket is purchased from an agent, the box office, or the Internet. A ticket has a description, an event, a date and a price. An agent has a name and a phone number. The box office has an address and a phone number. The Internet has a URL address.

Draw the entities and represent the exclusive relationship.



## ÚKOL #1C

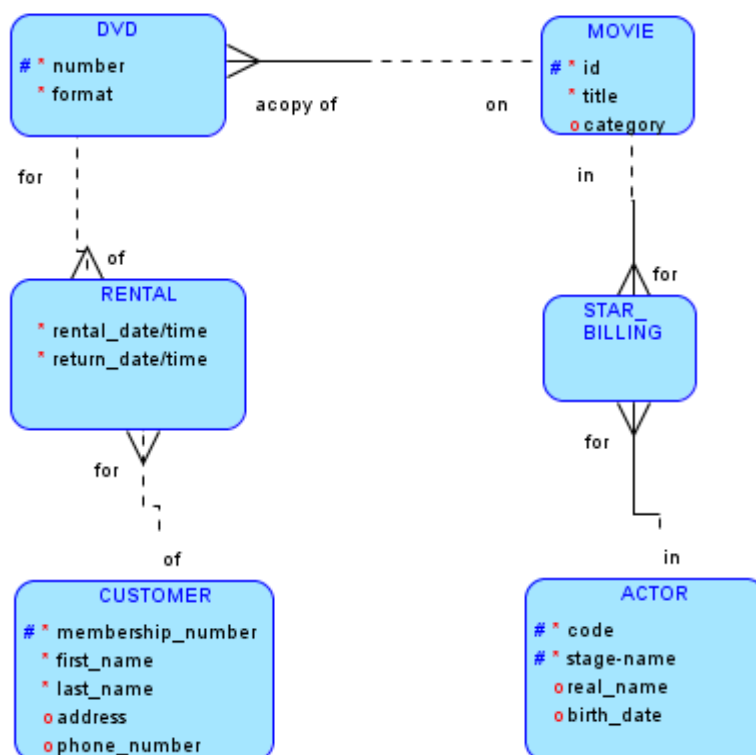
Změňte ER model Video Store a doplňte následujícími požadavky:

Jak víte, potřebujeme zachovat historii všech našich výpůjček. Pokaždé, když si zákazník vypůjčí DVD, chceme uchovat datum a čas vypůjčení a datum a čas vrácení. Všechny naše DVD jsou splatné druhý den, takže nepotřebujeme uchovat datum splatnosti. Vedení historie výpůjček nám umožní analyzovat strukturu výpůjček. Budeme schopni určit, kolik DVD si každý zákazník vypůjčil a kolikrát vrátil DVD pozdě. Budeme vědět, kolikrát bylo DVD vypůjčeno, a tím pádem budeme vědět, kdy je nutné DVD vyřadit. Budeme vědět, které filmy preferují naši zákazníci.

Modify the Video Store ER model below to accommodate the following additional requirements: "You know, we really need to keep a history of all our rentals. Each time a customer rents a DVD, we would like to keep the rental date/time and the return date/time. All our DVDs are due back the next day, so we don't need to keep a due date. Keeping this rental history will allow us to analyze the pattern of our rentals. We will be able to determine how many DVDs each customer rents and how many times a customer has returned a DVD late. We will also know how many times a particular DVD has been used and will then know when to retire each DVD. We will also be able to analyze our customers' movie preferences."







## ÚKOL #2

Vytvořte hierarchický a rekurzivní model následující společnosti. Nakreslete pro každý z nich ERD.

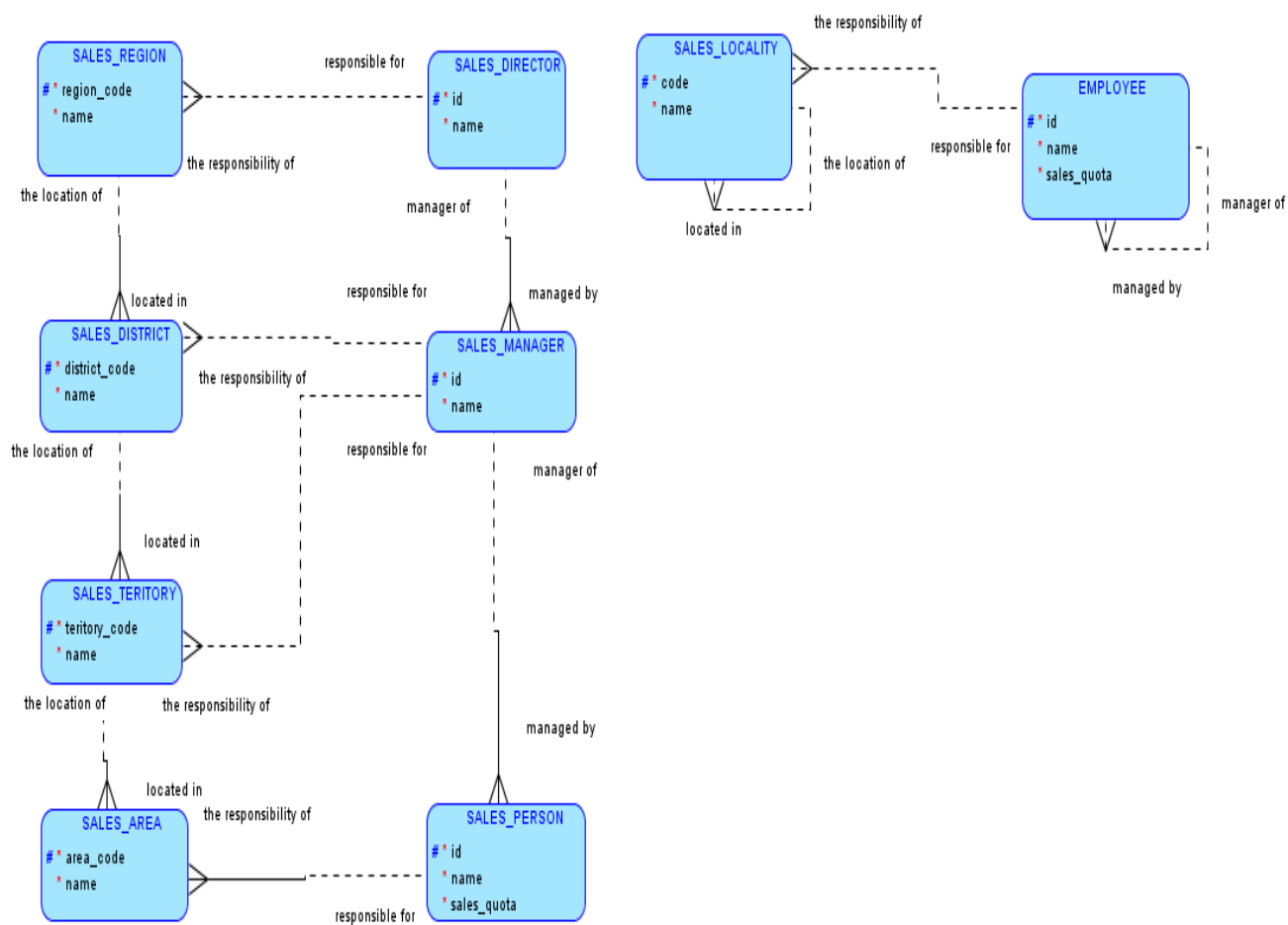
Develop a hierarchical model and a recursive model for the following company scenario. Draw the ERD for each

Our company sells products throughout the World. So we've divided our company into four major sales regions: Region 1, Region 2, Region 3, and Region 4. Each sales region has a unique region code. Each sales region is then divided into sales districts. For example, Region 1 is divided into the U.S, Canadian, and Southern districts. Each district has a unique district code. Each district is made up of sales territories. The Southern District is composed of three territories: Mexico, South America, and U.S. Territories. The U.S. District is made up of three territories: the West, Middle, and East. The Canadian District is composed of two territories East and West. Each territory has a unique territory code.

Each sales territory is then broken down into sales areas. For example, South America is made up of two sales areas: Brazil, and the Coastal sales areas. Each sales area has a unique sales area code. The Brazil area includes Uruguay, Paraguay, and Ecuador.

Each salesperson is responsible for one or more sales areas, and has a specific sales quota. We also have sales managers who are responsible for one or more sales districts and sales directors who are responsible for one or more sales regions. Each sales manager is responsible for the territories with his districts. We don't overlap our employees' responsibilities. Sales area is always the responsibility of a single salesperson, and our managers and director's responsibilities don't overlap. Sometimes our salespersons, manager, and directors will be on leave or special assignments and will not have sales turf responsibilities. We identify all our sales personnel by their employee ids.





## CVIČENÍ 4

NÁVRH DATABÁZE

ODDÍL 15, ANATOMIE SQL

NÁVRH DATABÁZE

ODDÍL 16, PRÁCE SE SLOUPCI A ŘÁDKY

NÁVRH DATABÁZE

ODDÍL 16, RELAČNÍ OPERÁTORY

NÁVRH DATABÁZE

ODDÍL 17, LOGICKÉ OPERÁTORY, PŘEDNOSTI

NÁVRH DATABÁZE

ODDÍL 17, TŘÍDĚNÍ ŘÁDKŮ

Spusťte následující příkazy v Oracle Application Express. Vložte kopii každého dotazu do dokumentu Wordu nebo do Poznámkového bloku.

## ÚKOL #1

Napište dotaz, který zobrazí příjmení a emailové adresy všech lidí z tabulky d\_client databáze DJ on Demand. Hlavička sloupce by měla být "Client" nebo "Email Address."

Write a query that displays the last\_name and email addresses for all the people in the DJ on Demand d\_client table. The column headings should appear as "Client" and "Email Address."

```
select last_name "Client", email "Email Address"
from d_clients;
```

## ÚKOL #2

Ředitel Global Fast Foods se rozhodl dát všem zaměstnancům 5% navýšení hodinové mzdy a bonus \$.50 za hodinu. Nicméně když se díval na výsledky, nemohl pochopit, proč nové navýšení není takové, jaké předpokládal. Paní Doe by měla mít nový plat \$7.59, pan Miller by měl mít \$11.00 a Monique Tuttle by měla mít \$63.50. Použil následující dotaz. Co měl udělat?

The manager of Global Fast Foods decided to give all employees at 5%/hour raise + a \$.50 bonus/hour. However, when he looked at the results, he couldn't figure out why the new raises were not as he predicted. Ms. Doe should have a new salary of \$7.59, Mr. Miller's salary should be \$11.00, and Monique Tuttle should be \$63.50. He used the following query. What should he have done?

```
SELECT last_name, salary *1.05 + 0.50
FROM f_staffs;
```

## ÚKOL #3

Ředitel Global Fast Foods by rád poslal kupóny pro následující prodej. Chce poslat jeden kupón do každé domácnosti. Vytvořte SELECT příkaz, který vrátí zákazníkovo příjmení a emailovou adresu.

The manager of Global Fast Foods would like to send out coupons for the upcoming sale. He wants to send one coupon to each household. Create the SELECT statement that returns the customer last name and a mailing address.

```
select last_name, address, city, state, zip
from f_customers;
```

## ÚKOL #4

Sue, Bob a Monique byli zaměstnanci měsíce. Užitím tabulky `f_staffs` vytvořte `SELECT` příkaz, který zobrazí výsledky znázorněné v tabulce Super Star.

Sue, Bob, and Monique were the employees of the month. Using the `f_staffs` table, create a `SELECT` statement to display the results as shown in the Super Star chart.

```
select '*** ' || first_name || ' *** ' || first_name || ' ***' "Super
Star"
from f_staffs;
```

## ÚKOL #5

*Global Fast Foods se rozhodl navýšit plat všem zaměstnancům o 5%. Vypracujte zprávu, která zobrazí výstup uvedený v tabulce.*

Global Fast Foods has decided to give all staff members a 5% raise. Prepare a report that presents the output as shown in the chart.

```
select last_name "EMPLOYEE LAST NAME",
salary "CURRENT SALARY", salary*1.05 "SALARY WITH 5% RAISE"
from f_staffs;
```

## ÚKOL #6

Majitelé DJs on Demand by rádi získali zprávu o všech položkách v tabulce `D_CDs` s následujícími hlavičkami sloupců: Inventory Item, CD Title, Music Producer a Year Purchased. Připravte tuto zprávu.

The owners of DJs on Demand would like a report of all items in their `D_CDs` table with the following column headings: Inventory Item, CD Title, Music Producer, and Year Purchased. Prepare this report.

```
select CD_NUMBER "Inventory Item", TITLE "CD Title",
PRODUCER "Music Producer", YEAR "Year Purchased"
from d_cds;
```

## ÚKOL #7

Užitím databáze Global Fast Foods získejte jméno, příjmení a adresu zákazníka, jehož ID je 456.

Using the Global Fast Foods database, retrieve the customer's first name, last name, and address for the customer who uses ID 456.

```
select first_name, last_name, address
from f_customers
where ID = 456;
```

## ÚKOL #8

Zobrazte název, datum začátku a konce propagační akce Global Fast Foods, kde se rozdávaly dárky "ballpen and highlighter".

Show the name, start date, and end date for Global Fast Foods' promotional item "ballpen and highlighter" giveaway.

---

```
select name, start_date, end_date
from f_promotional_menus
where give_away = 'ballpen and highlighter';
```

---

## ÚKOL #9

Manažér DJ on Demand požaduje seznam všech CD titulů a roků výroby těch CD, které byly vyrobeny před rokem 2000.

The manager of DJ on Demand would like a report of all the CD titles and years of CDs that were produced before 2000.

---

```
select title, year
from d_cds
where year < 2000;
```

---

## ÚKOL #10

Napište SQL příkaz, který zobrazí číslo studenta (studentno), jehož hlavní předmět v tabulce students je tělocvik. Označte sloupec studentno titulkem Student Number.

Write a SQL statement that will display the student number (studentno) of any student who has a PE major in the table named students. Title the studentno column Student Number.

---

```
select studentno "Student Number"
from students
where major is not null;
```

---

## ÚKOL #11

Napište příkaz SQL, který vypíše zaměstnance Global Fast Foods narozené před rokem 1980.

Write a SQL statement that lists the Global Fast Foods employees who were born before 1980.

---

```
select first_name, last_name
from F_staffs
where birthdate < '1.1.1980';
```

---

## ÚKOL #12

Zobrazte jméno, příjmení a plat všech zaměstnanců Global Fast Foods, jejichž plat se pohybuje v rozmezí \$5.00 až \$10.00 za hodinu.

Display the first name, last name, and salary of all Global Fast Foods staff whose salary is between \$5.00 and \$10.00 per hour.

---

```
select first_name, last_name, salary
from F_staffs
where salary between 5 and 10;
```

---

## ÚKOL #13

Pouze užitím operátorů menší než, rovná se, větší než přepište následující dotaz:

```
SELECT first_name, last_name
FROM f_staffs
WHERE salary BETWEEN 20.00 and 60.00;
```

Using only the less than, equal, or greater than operators, rewrite the following query:

```
SELECT first_name, last_name
FROM f_staffs
WHERE salary BETWEEN 20.00 and 60.00;
```

```
SELECT first_name, last_name
FROM f_staffs
WHERE salary > 20 and salary <= 60;
```

## ÚKOL #14

Vyberte všechny zaměstnance Oracle databáze, jejichž příjmení končí „s“. Změňte záhlaví sloupce na Possible Candidates.

Select all the Oracle database employees whose last names end with “s” Change the heading of the column to read Possible Candidates.

```
select last_name "Possible Candidates"
from employees
where last_name like '%s';
```

## ÚKOL #15

Napište SQL příkaz, který vypíše seznam skladeb z inventáře DJs on Demand, jejichž typ kódu je 77, 12 nebo 1.

Write a SQL statement that lists the songs in the DJs on Demand inventory that are type code 77, 12 or 1

```
select title, type_code
from d_songs
where type_code IN (77, 12, 1);
```

## ÚKOL #16

Zobrazte příjmení všech zaměstnanců Global Fast Foods, kteří mají „e“ a „i“ ve svém příjmení.

Display the last names of all Global Fast Foods employees who have “e” and “i” in their last names.

```
select last_name
from f_staffs
where last_name like '%e%' and last_name like '%i%';
```

## ÚKOL #17

Pomocí tabulky zaměstnanci napište dotaz, který zobrazí všechny zaměstnance, jejichž příjmení začíná na „D“ a mají „a“ a „e“ kdekoli v příjmení.

Using the employees table, write a query to display all employees whose last names start with “D” and have “a” and “e” anywhere in their last name.



---

```
select last_name
from employees
where last_name like 'D%' and last_name like '%e%' and
last_name like '%a%';
```

---

### ÚKOL #18

Kde jinde než v soukromých domech pořádali DJs on Demand akce? In which venues did DJs on Demand have events that were not in private homes?

---

```
select loc_type
from d_venues
where loc_type != 'Private Home';
```

---

### ÚKOL #19

Kdo jsem?

Byl jsem přijat Oraclem po květnu 1998, ale před červnem 1999. Můj plat je menší než \$8,000 za rok a mám „en“ v příjmení.

Who am I?  
I was hired by Oracle after May 1998 but before June of 1999. My salary is less than \$8000 a year and I have an “en” in my last name.

---

```
select first_name, last_name
from employees
where hire_date between '31.5.1998' and '1.6.1999'
and salary*12 < 8000 and last_name like '%en%';
```

---

### ÚKOL #20

V níže uvedeném příkladu přejmenujte sloupec employee\_id pomocí alias na „Number“. Dokončete příkaz SQL tak, aby seřadil výsledky podle sloupce s aliasem.

In the example below, assign the employee\_id column the alias of “Number.” Complete the SQL statement to order the results set by the column alias.

---

```
SELECT employee_id as Numbers, first_name, last_name
FROM employees
order by Numbers;
```

---

### ÚKOL #21

Seřadte písně databáze DJ on Demand sestupně podle názvu. Použijte alias „Our Collection“ pro název skladby.

Order the DJ on Demand songs by descending title. Use the alias “Our Collection” for the song title.

---

```
select title "Our Collection"
from d_songs
order by title desc;
```

---

## ÚKOL #22

Napište SQL příkaz pomocí klauzule *ORDER BY*, který získá potřebné informace. Dotaz nespouštějte.

Vytvořte seznam studentů, kteří jsou v prvním ročníku školy. Zahrňte jméno, příjmení, ID studenta a číslo parkovacího místa. Seřadte výsledky abecedně podle příjmení studentů a následně podle jména. Jestliže více než jeden student má stejné příjmení, seřadte jejich jména od Z do A. Všechny ostatní výsledky by měly být v abecedním pořadí (od A do Z).

Write a SQL statement using the *ORDER BY* clause that could retrieve the information needed. Do not run the query.

Create a list of students who are in their first year of school. Include the first name, last name, student ID number, and parking place number. Sort the results alphabetically by student last name and then by first name. If more than one student has the same last name, sort each first name in Z to A order. All other results should be in alphabetical order (A to Z).

```
select first name, last name, student_ID, parking_place  
from students  
order by last_name, first_name desc;
```

## CVIČENÍ 5

PROGRAMOVÁNÍ SQL	ODDÍL 1, MANIPULACE SE ZNAKY
PROGRAMOVÁNÍ SQL	ODDÍL 1, ČÍSELNÉ FUNKCE
PROGRAMOVÁNÍ SQL	ODDÍL 1, DATUMOVÉ FUNKCE
PROGRAMOVÁNÍ SQL	ODDÍL 2, KONVERZNÍ FUNKCE
PROGRAMOVÁNÍ SQL	ODDÍL 2, NULL FUNKCE
PROGRAMOVÁNÍ SQL	ODDÍL 1, PODMÍNĚNÉ VÝRAZY

## ÚKOL #1

Tři samostatná slova “Oracle,” “Internet,” a “Academy,” použijte v příkazu, který zobrazí následující výstup:

The Best Class

Oracle Internet Academy

Using the three separate words “Oracle,” “Internet,” and “Academy,” use one command to produce the following output:  
The Best Class  
Oracle Internet Academy .

```
select CONCAT('Oracle', CONCAT(' Internet', ' Academy'))
"The Best Class" from dual;
```

## ÚKOL #2

Jaké je pozice znaku “I” v řetězci “Oracle Internet Academy”?

What’s the position of “I” in “Oracle Internet Academy”?

```
select INSTR('Oracle Internet Academy', 'I') as position
from dual;
```

## ÚKOL #3

Řetězec “Oracle Internet Academy” doplňte na následující výstup:

Oracle\$\$\$Internet\$\$\$Academy

Starting with the string “Oracle Internet Academy”, pad the string to produce:  
Oracle\$\$\$Internet\$\$\$Academy

```
select REPLACE('Oracle Internet Academy', ' ', '$$$') as
replace_space from dual;
```

## ÚKOL #4

Použitím parametru (substituční proměnné) pro jméno oddělení napište dotaz, který vypíše ID oddělení, jméno oddělení, ID umístění pro oddělení zadané v proměnné the\_department\_of\_your\_choice. Použijte tabulku DEPARTMENTS. Poznámka: všechny substituční proměnné jsou brány jako znakové řetězce, tudíž apostrofy ( ' ) nejsou nutné.

Using a substitution variable for the department name, write a query listing department id, department name and location id for departments located in the\_department\_of\_your\_choice. Use the DEPARTMENTS table. Note: All substitution variables in OAE are treated as character strings, so no quotes ( ' ) are needed.

---

```
SELECT department_id, department_name, location_id
FROM departments
WHERE department_name = :the_department_of_your_choice;
```

---

### ÚKOL #5

Zobrazte Oracle databázi zaměstnanců – příjmení a plat těch, jejichž employee\_id je mezi 100 a 102. Připojte třetí sloupec, který vydělí každý plat hodnotou 1,55 a zaokrouhlí výsledek na dvě desetinná místa.

Display Oracle database employee last\_name and salary for employee\_ids between 100 and 102. Include a third column that divides each salary by 1.55 and rounds the result to two decimal places.

---

```
select last_name, salary, round(salary/1.55 ,2) as th_col
from employees
where employee_id between 100 and 102
```

---

### ÚKOL #6

Zobrazte příjmení a plat těch zaměstnanců, kteří pracují v oddělení 80. Zvyšte každému z nich plat o 5,33 % a výsledek ořežte na dvě desetinná místa.

Display employee last\_name and salary for those employees who work in department 80. Give each of them a raise of 5.33% and truncate the result to two decimal places.

---

```
select last_name, salary, trunc(salary*1.0533 ,2) as
new_salary
from employees
where department_id = 80;
```

---

### ÚKOL #7

Vydělte plat každého zaměstnance třemi. Zobrazte příjmení a platy pouze těch zaměstnanců, jejichž plat je násobkem 3.

Divide each employee's salary by 3. Display only those employees' last names and salaries who earn a salary that is a multiple of 3.

---

```
select last_name, salary
from employees
where mod(salary, 3) = 0;
```

---

### ÚKOL #8

Zobrazte počet dní mezi začátkem minulých letních prázdnin a začátkem letošního školního roku. Předpokládejme, že měsíc má 30,5 dne. Nazvěte výstup „Days“.

Display the days between the start of last summer's school vacation break and the day school started this year. Assume 30.5 days per month. Name the output "Days."

---

```
select months_between('1.9.2009', '1.7.2009') * 30.5 "Days"
from dual;
```

---

## ÚKOL #9

Pomocí jednoho výrazu zaokrouhlete dnešní datum s přesností na měsíc a rok a také jej ořežte s přesností na měsíc a rok. Použijte alias pro každý sloupec.

Using one statement, round today's date to the nearest month and nearest year and truncate it to the nearest month and nearest year. Use an alias for each column.

```
select round(sysdate, 'MM') as today_round_MM,
round(sysdate, 'YYYY') as today_round_YYYY,
trunc(sysdate, 'MM') as today_trunc_MM,
trunc(sysdate, 'YYYY') as today_trunc_YYYY
from dual;
```

## ÚKOL #10

Zobrazte počet let mezi dnem narození zaměstnance Boba Millera a dnešním dnem. Zaokrouhlete na nejbližší rok.

Display the number of years between the Global Fast Foods employee Bob Miller's birthday and today. Round to the nearest year.

```
select round((sysdate - birthdate)/365.25) as Age
from f_staffs
where first_name = 'Bob' and last_name = 'Miller';
```

## ÚKOL #11

Učitel řekl, že termín odevzdání projektu je poslední den tohoto měsíce. Který den to bude? Nazvěte výstup "Deadline."

The teacher said you have until the last day of this month to turn in your research paper. What day will this be? Name the output, "Deadline."

```
select last_day(sysdate) as Deadline
from dual;
```

## ÚKOL #12

Vypište příjmení a datum narození zaměstnanců Global Fast Food. Převed'te datum narození na znaková data ve formátu Month DD, YYYY. Potlačte výpis úvodních nul.

List the last names and birthdays of Global Fast Food Employees. Convert the birth dates to character data in the Month DD, YYYY format. Suppress any leading zeros.

```
select last_name,
to_char(birthdate, 'Month DD, YYYY') as birthdate
from f_staffs;
```

## ÚKOL #13

Zformátujte dotaz z tabulky f\_promotional\_menus databáze Global Fast Foods pro tisk data zahájení propagace s kódem 110 ve tvaru: Propagace začala desátého února 2004.

Format a query from the Global Fast Foods f\_promotional\_menus table to print out the start\_date of promotional code 110 as: The promotion began on the tenth of February 2004.

```

select 'The promotion began on the ' || to_char(start_date,
'ddspth') || ' of ' || to_char(start_date, 'Month YYYY') ||
'.' as start_date
from f_promotional_menus
where code = 110;

```

## ÚKOL #14

Ellen Abel je zaměstnankyní, které byl zvýšen plat o \$2,000. Zobrazte její jméno a příjmení, současný plat a nový plat. Zobrazte oba platy se znakem \$ a dvěma desetinnými místy. Nazvěte sloupec nového platu jako New Salary.

Ellen Abel is an employee who has received a \$2,000 raise. Display her first name and last name, her current salary, and her new salary. Display both salaries with a \$ and two decimal places. Label her new salary column AS New Salary.

```

select first_name, last_name, to_char(salary,
'$9999,999.00') as Salary, to_char(salary + 2000,
'$9999,999.00') as "New Salary"
from Employees
where first_name = 'Ellen' and last_name = 'Abel';

```

## ÚKOL #15

Vytvořte dotaz, který zformátuje v tabulce d\_packages sloupce low-range a high-range, kde náklady za balíček budou ve formátu \$2,500.00.

Create a query that will format the DJ on Demand d\_packages columns, low-range and high-range package costs, in the format \$2500.00.

```

select to_char(low_range, '$9999,999.00') as "low range",
to_char(high_range, '$9999,999.00') as "high range"
from d_packages;

```

## ÚKOL # 16

Vytvořte zprávu, která zobrazí v databázi Global Fast Foods název propagace, datum zahájení a ukončení z tabulky f\_promotional\_menus. Pokud je uvedeno datum ukončení, dočasně ho nahraďte výrazem "end in two weeks." Jestliže není uvedeno žádné datum ukončení, nahraďte jej dnešním datem.

Create a report that shows the Global Fast Foods promotional name, start date and end date from the f\_promotional\_menus table. If there is an end date, temporarily replace it with "end in two weeks." If there is no end date, replace it with today's date.

```

select name, NVL2(end_date, 'end in two weeks',
to_char(sysdate, 'DD.MM.YY'))
from F_PROMOTIONAL_MENUS;

```

## ÚKOL #17

Manažér Global Fast Foods se rozhodl dát všem zaměstnancům, kteří nevydělávají přesčas, přesčasový příplatek \$5.00. Vytvořte dotaz, který zobrazí příjmení a přesčasový příplatek \$5.00.

The manager of Global Fast Foods has decided to give all staff that currently does not earn overtime an overtime rate of \$5.00. Construct a query that displays last names and overtime rate shown as \$5.00.

---

```
select last_name,
       NVL(to_char(overtime_rate), '$5.00') as "Over time"
from f_staffs;
```

---

## ÚKOL # 18

Všechny null hodnoty ve sloupci specialty z tabulky d\_partners databáze DJs on Demand nahradte výrazem “No Specialty”. Zobrazte jen jméno a sloupec specialty.

For all null values in the specialty column in the DJs on Demand d\_partners table, substitute “No Specialty.”. Show the first name and specialty columns only.

---

```
select first_name, NVL(specialty, 'No Specialty') as
       Specialty
from d_partners;
```

---

## ÚKOL #19

Vytvořte dotaz z tabulky d\_songs databáze DJ on Demand, který nahradí 2-minutové písně výrazem “shortest” a 10-minutové písně výrazem “longest.” Označte sloupec s výstupem “Play Times.”

From the DJ on Demand d\_songs table, create a query that replaces the 2-minute songs with “shortest” and the 10-minute songs with “longest.” Label the output column “Play Times.”

---

```
select title,
       DECODE(duration,
               '2 min', 'shortest',
               '10 min', 'longest') as "Play Times"
from d_songs;
```

---

## ÚKOL #20

Použijte tabulku employees Oracle databáze a CASE výraz pro dekódování ID oddělení. Zobrazte ID oddělení, příjmení, plat a sloupec nazvaný “New Salary”, jehož hodnoty jsou založeny na následujících podmínkách:

Jestliže ID oddělení je 10, potom plat vynásobte 1,25.

Jestliže ID oddělení je 90, potom plat vynásobte 1,5.

Jestliže ID oddělení je 130, potom plat vynásobte 1,75.

Jinak zobrazte původní plat.

Use the Oracle database employees table and CASE expression to decode the department id. Display the department id, last name, salary and a column called “New Salary” whose value is based on the following conditions:

If the department id is 10 then 1.25 \* salary.

If the department id is 90 then 1.5 \* salary.

If the department id is 130 then 1.75 \* salary.

Otherwise, display the old salary.

---

```
select department_id, last_name, salary,
       CASE department_id
         WHEN 10 then 1.25 * salary
         WHEN 90 then 1.5 * salary
         WHEN 130 then 1.75 * salary
```

---

---

```
ELSE salary
END AS "New Salary"
from employees;
```

---

## ÚKOL #21

Zobrazte jméno, příjmení, ID manažera a provizi v procentech všech zaměstnanců v odděleních 80 a 90. Zobrazte ID manažera v dalším sloupci nazvaném "Review." Jestliže nemají manažera, zobrazte procenta provize. Jestliže nemají provizi, zobrazte 99999.

*Display the first name, last name, manager ID, and commission percentage of all employees in departments 80 and 90. Display the manager ID in an additional column called "Review." If they don't have a manager, display the commission percentage. If they don't have a commission, display 99999.*

---

```
select first_name, last_name, manager_ID, commission_pct,
CASE WHEN manager_id IS NULL THEN commission_pct
      ELSE COALESCE(commission_pct, 99999)
END as "Review"
from employees
where department_id in(80, 90);
```

---



## CVIČENÍ 6

PROGRAMOVÁNÍ SQL	ODDÍL 3, CROSS A NATURAL JOIN
PROGRAMOVÁNÍ SQL	ODDÍL 3, KLAUZULE JOIN
PROGRAMOVÁNÍ SQL	ODDÍL 3, INNER & OUTER JOIN
PROGRAMOVÁNÍ SQL	ODDÍL 4, GROUP FUNCTION
PROGRAMOVÁNÍ SQL	ODDÍL 4, COUNT, DISTINCT, NVL

## ÚKOL #1

Vytvořte cross-join (křížové spojení), které zobrazí příjmení a název oddělení z tabulek employees a departments.

Create a cross-join that displays the last name and department name from the employees and departments tables.

```
select last_name, department_name
from departments cross join employees;
```

## ÚKOL #2

Vytvořte dotaz, kde použijete natural join pro spojení tabulek departments a locations pomocí sloupce location\_id. Zobraz id oddělení, název oddělení, id umístění a město.

Create a query that uses a natural join to join the departments table and the locations table by the location\_id column. Display the department id and name, location id and city.

```
select department_id, department_name, location_id, city
from locations natural join departments;
```

## ÚKOL #3

Vytvořte dotaz, kde použijete natural join pro spojení tabulky departments pomocí sloupce location\_id. Omezte výstup pro id oddělení od 20 do 50. Zobrazte id oddělení, název oddělení, id umístění a město.

Create a query that uses a natural join to join the departments table by the location\_id column. Restrict the output to only department IDs of 20 and 50. Display the department id and name, location id and city.

```
select department_id, department_name, location_id, city
from locations natural join departments
where department_id IN(20, 50);
```

## ÚKOL #4

Spojte v Oracle databázi tabulky locations a departments na sloupci location\_id. Omezte výstup pouze pro umístění s číslem 1400.

Join the Oracle database locations and departments table using the location\_id column. Limit the results to location 1400 only.

```
select department_name, location_id
from locations join departments
```

---

```
using (location_id)
where location_id = 1400;
```

---

### ÚKOL #5

Zobrazte název země, id oblasti a název oblasti pro americký kontinent. Vyberte region\_ID, region\_name, country\_name.

Display country name, region ID and region name for Americas. Select region\_ID, region\_name, country\_name.

---

```
select region_ID, region_name, country_name
from regions join countries
using (region_id)
where region_name = 'Americas';
```

---

### ÚKOL #6

Napište příkaz, který zobrazí ID zaměstnance, jméno, příjmení zaměstnance, ID manažera, jméno a příjmení manažera každého zaměstnance z tabulky employees. Tip: Je to self-join.

Write a statement that displays the employee ID, first name, last name, manager ID, manager first name, and manager last name for every employee in the employees table. Hint: this is a self-join.

---

```
select B.employee_ID,
B.first_name,
B.last_name,
A.employee_ID as "manager ID",
A.first_name as "manager first name",
A.last_name as "manager last name"
from employees A join employees B
ON (A.employee_id = B.manager_id);
```

---

### ÚKOL #7

Zobraz ID manažera, ID oddělení, jméno oddělení, jméno a příjmení všech zaměstnanců v oddělení 80, 90, 110 a 190.

Query and display manager ID, department ID, department name, first name, and last name for all employees in departments 80, 90, 110, and 190.

---

```
select manager_ID, department_ID, department_name,
first_name, last_name
from departments natural join employees
where department_id IN (80, 90, 110, 190);
```

---

Display the employee's last name and employee number along with the manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively.

Modify problem 4 to display all employees, including those who have no manager. Order the results by the employee number

Upravte úkol 7 tak, aby jste zobrazili všechny zaměstnance včetně těch, kteří nemají žádného manažera. Seřadte výsledky podle počtu zaměstnanců.

---

```

select B.employee_ID "Emp#",
B.last_name "Employee",
A.employee_ID as "Mgr#",
A.last_name as "Manager"
from employees A full outer join employees B
ON (A.employee_id = B.manager_id)
order by b.employee_id;

```

---

## ÚKOL #8

Vytvořte dotaz, který zobrazí průměrnou cenu události databáze DJ on Demand. Zaokrouhlete na dvě desetinná místa.

Create a query that will show the average cost of the DJ on Demand events. Round to two decimal places.

---

```

select AVG(cost) "Average Cost"
from d_events;

```

---

## ÚKOL #9

Najděte průměrný plat zaměstnanců Global Fast Foods, jejichž manažera je 19.

Find the average salary for Global Fast Foods staff members whose manager ID is 19.

---

```

select AVG(salary) "Average Salary"
from f_staffs
where manager_id = 19;

```

---

## ÚKOL #10

Kolik písní je uvedeno v tabulce D\_SONGS databáze DJs on Demand?

How many songs are listed in the DJs on Demand D\_SONGS table?

---

```

select COUNT(*) "Count of songs"
from d_songs;

```

---

Na kolika různých místech měli DJs on Demand akce?

In how many different location types has DJs on Demand had venues?

---

```

select COUNT(distinct loc_type) "Count of type locations"
from d_venues;

```

---

# CVIČENÍ 7

**PROGRAMOVÁNÍ SQL**

ODDÍL 6, GROUP BY &amp; HAVING

**PROGRAMOVÁNÍ SQL**

ODDÍL 6, PODDOTAZY

**PROGRAMOVÁNÍ SQL**

ODDÍL 6, JEDNOŘÁDKOVÉ PODDOTAZY

**PROGRAMOVÁNÍ SQL**

ODDÍL 6, VÍCEŘÁDKOVÉ PODDOTAZY

## ÚKOL #1

Každý z těchto SQL dotazů obsahuje chybu. Najděte ji a opravte.  
K ověření správnosti použijte Oracle Application Express.

Each of the following SQL queries has an error. Find the error and correct it. Use Oracle Application Express to verify that your corrections produce the desired results.

*a:*

```
SELECT manager_id, AVG(salary)
FROM employees
WHERE salary <16000
GROUP BY manager_id;
```

*b:*

```
SELECT cd_number, COUNT(title)
FROM d_cds
WHERE cd_number < 93;
```

*c:*

```
SELECT ID, MAX(ID), artist AS Artist
FROM d_songs
WHERE duration IN('3 min', '6 min', '10 min')
HAVING MAX(ID) < 50
GROUP by ID, artist;
```

*d:*

```
SELECT loc_type, rental_fee AS Fee
FROM d_venues
WHERE id <100
ORDER BY Fee;
```

## ÚKOL #2

Které záznamy DJs on Demand z tabulky d\_play\_list\_items mají stejné event\_id jako ty, kde song\_id je 45.

What DJs on Demand d\_play\_list\_items song\_id's have the same event\_id as song\_id 45?

```
select *
from d_play_list_items
where event_id = (select event_id
                  from d_play_list_items
                  where song_id = 45);
```

Které události v databázi DJs on Demand stojí více než ty, jejichž event\_id = 100?

Which events in the DJs on Demand database cost more than event\_id = 100?

```
select id, name
from d_events
where cost > (select cost
              from d_events
              where id = 100);
```

## ÚKOL #3

Která pracovní pozice v Global Fast Foods má nižší plat než kterýkoliv kuchař?

What is the staff type for those Global Fast Foods jobs that have a salary less than those of any Cook staff-type jobs?

```
select staff_type
from f_staffs
where salary = (select MIN(Salary)
               from f_staffs);
```

## ÚKOL #4

Najděte příjmení všech zaměstnanců, jejichž plat je stejný jako nejmenší mzda z každého oddělení.

Find the last names of all employees whose salaries are the same as the minimum salary for any department.

```
select last_name, salary
from employees
where salary = ANY (select min(salary)
                   from employees
                   group by department_id);
```

## ÚKOL #5

Který zaměstnanec z Global Fast Foods má nejmenší plat? Tip: Můžete použít jak jednořádkový, tak víceřádkový poddotaz.

Which Global Fast Foods employee earns the lowest salary? Hint: You can use either a single-row or a multiple-row subquery.

*var. A: Single-Row*

---

```
select last_name, salary "Min Salary"
from employees
where salary = (select MIN(salary)
                from employees);
```

---

*var. B: Multi-Row*

---

```
select last_name, salary "Min Salary"
from employees
where salary <= ALL (select salary
                     from employees);
```

---

## CVIČENÍ 8

PROGRAMOVÁNÍ SQL	ODDÍL 7, PŘÍKAZ INSERT
PROGRAMOVÁNÍ SQL	ODDÍL 7, PŘÍKAZ UPDATE, DELETE
PROGRAMOVÁNÍ SQL	ODDÍL 7, DEFAULT, MERGE, MULTI i.
PROGRAMOVÁNÍ SQL	ODDÍL 8, TVORBA TABULEK
PROGRAMOVÁNÍ SQL	ODDÍL 10, MODIFIKACE TABULEK

## ÚKOL #1

V databázi DJs on Demand byly právě koupeny čtyři CD. Použitím příkazu INSERT přidejte každé CD do tabulky copy\_d\_cds. Po dokončení vkladu proveďte příkaz SELECT \* pro ověření vaší práce.

*DJs on Demand just purchased four new CDs. Use an explicit INSERT statement to add each CD to the copy\_d\_cds table. After completing the entries, execute a SELECT \* statement to verify your work.*

```
CREATE TABLE copy_d_cds
AS (select * from d_cds);
```

```
INSERT INTO copy_d_cds(cd_number, title, producer, year)
VALUES
(97, 'Celebrate the Day', 'R&B Inc.', 2003);
```

```
INSERT INTO copy_d_cds(cd_number, title, producer, year)
VALUES
(98, 'Holiday Tunes for All Ages', 'Tunes are Us', 2004);
```

```
INSERT INTO copy_d_cds(cd_number, title, producer, year)
VALUES
(99, 'Party Music', 'Old Town Records', 2004);
```

```
INSERT INTO copy_d_cds(cd_number, title, producer, year)
VALUES
(100, 'Best of Rock and Roll', 'Old Town Records', 2004);
```

```
SELECT * FROM copy_d_cds;
```

## ÚKOL #2

V databázi DJs on Demand jsou očekávány dvě nové události. Jedna je podzimní fotbalová párty a druhá párty ve stylu šedesátých let. Na tyto akce klienti vyžadují písně uvedené v tabulce. Přidejte tyto písně do tabulky copy\_d\_songs užitím příkazu INSERT.

*DJs on Demand has two new events coming up. One event is a fall football party and the other event is a sixties theme party. The DJs on Demand clients requested the songs shown in the table for their events. Add these songs to the copy\_d\_songs table using an implicit INSERT statement.*

```
INSERT INTO copy_d_cds(cd_number, title)
VALUES(52, 'Sufing Summer');
```

```
INSERT INTO copy_d_cds(cd_number, title)
VALUES(53, 'Victory Victory');
```

```
SELECT * FROM copy_d_cds;
```

## ÚKOL #3

Monique Tuttle, manažérka Global Fast Foods, poslala memorandum požadující okamžitou změnu cen. Cena jahodového koktejlu se zvýší z 3,59 dolarů na 3,75 dolarů a cena za hranolky se zvýší na 1,20 dolarů. Tyto změny proveďte v tabulce copy\_f\_food\_items.

*Monique Tuttle, the manager of Global Fast Foods, sent a memo requesting an immediate change in prices. The price for a strawberry shake will be raised from \$3.59 to \$3.75, and the price for fries will increase to \$1.20. Make these changes to the copy\_f\_food\_items table.*

```
CREATE TABLE copy_f_food_items
AS SELECT * FROM f_food_items;
```

```
UPDATE copy_f_food_items
SET price = 3.75
WHERE description = 'Strawberry Shake';
UPDATE copy_f_food_items
SET price = 1.2
WHERE description = 'Fries';
```

## ÚKOL #4

Přidejte tyto nové zákazníky do tabulky copy\_f\_customers. Možná jste už přidali Katie Hernandez. Podaří se vám přidat všechny tyto záznamy?

*Add the new customers shown below to the copy\_f\_customers table. You may already have added Katie Hernandez. Will you be able to add all these records successfully?*

```
INSERT INTO copy_f_customers
```



```
VALUES (145, 'Katie', 'Hernandez', '92 Chico Way', 'Los
Angeles', 'CA', 98008, '8586667641');

INSERT INTO copy_f_customers
VALUES (225, 'Daniel', 'Spode', '1923 Silverado', 'Denver',
'CO', 80219, '7193343523');

INSERT INTO copy_f_customers
VALUES (230, 'Adam', 'Zurn', '5 Admiral Way', 'Seattle',
'WA', NULL, '4258879009');
```

We cannot put NULL for a ZIP code because this column is defined as NOT NULL.

## ÚKOL #5

Sue Doe, vynikající zaměstnankyni, byl zvýšen plat. Nyní bude placena stejně jako Bob Miller. Aktualizujte její záznam v tabulce copy\_f\_staffs.

Sue Doe has been an outstanding Global Foods staff member and has been given a salary raise. She will now be paid the same as Bob Miller. Update her record in copy\_f\_staffs.

```
UPDATE copy_f_staffs
SET salary = (SELECT salary
              FROM copy_f_Staffs
              WHERE CONCAT(first_name,last_name) = 'BobMiller'
WHERE CONCAT(first_name,last_name) = 'SueDoe');
```

## ÚKOL #6

Kdy chcete nastavit výchozí (DEFAULT) hodnotu?

*When would you want a DEFAULT value?*

This option prevents null values from entering the columns if a row is inserted without a specified value for the column.

Using default values also allows you to control where and when the default value should be applied.

## ÚKOL #7

Doplňte popis tabulky GRADUATE CANDIDATE. Sloupec credits je cizím klíčem odkazujícím na požadovanou tabulku.

*Complete the GRADUATE CANDIDATE table instance chart. Credits is a foreign-key column referencing the requirements table.*

Column Name	student_id	last_name	first_name	credits	graduation_date
Key Type	Yes				
Nulls/Unique		No / No	No / No	No / No	Yes / No
FK Column				Yes	
Datatype	NUMBER	VARCHAR2	VARCHAR2	NUMBER	DATE
Length	6	30	20	3	

## ÚKOL #8

Napište příkaz pro vytvoření tabulky grad\_candidates.

*Write the syntax to create the grad\_candidates table.*

```
CREATE TABLE grad_candidates
(student_id NUMBER(6) PRIMARY KEY,
last_name VARCHAR2(30) NOT NULL,
first_name VARCHAR2(20) NOT NULL,
credits NUMBER(3) FOREIGN KEY REFERENCES
parent_table(credit_id),
graduation_date DATE);
```

## ÚKOL #9

Ověřte vytvoření tabulky pomocí příkazu DESCRIBE.

*Confirm creation of the table using DESCRIBE.*

```
DESC grad_candidates;
```

## ÚKOL #10

Vložte do tabulky o\_employees nový sloupec s názvem "Termination.". Datový typ pro nový sloupec by měl být VARCHAR2. Nastavte výchozí hodnotu pro tento sloupec jako SYSDATE ve formátu: February 20th, 2003.

*In your o\_employees table, enter a new column called "Termination.". The datatype for the new column should be VARCHAR2. Set the DEFAULT for this column as SYSDATE to appear as character data in the format: February 20th, 2003.*

```
CREATE TABLE o_employees
AS SELECT * from employees;
```

```
CREATE TABLE o_jobs
AS SELECT * from jobs;
```

```
DESC o_employees;
```

```
ALTER TABLE o_employees
ADD (termination VARCHAR2(20) DEFAULT TO_CHAR(SYSDATE,
'Month DDTH YYYY'));
```

## ÚKOL #11

Vytvořte nový sloupec v tabulce o\_employees s názvem start\_date. Použijte datový typ TIMESTAMP WITH LOCAL TIME ZONE.

*Create a new column in the o\_employees table called start\_date. Use the TIMESTAMP WITH LOCAL TIME ZONE as the datatype.*

---

```
ALTER TABLE o_employees  
ADD (start_date TIMESTAMP WITH LOCAL TIME ZONE);
```

---

## ÚKOL #12

---

Užitím příkazu CREATE or REPLACE vytvořte pohled s názvem view\_copy\_d\_songs, který zobrazí všechny sloupce z tabulky copy\_d\_songs.

Use the CREATE or REPLACE option to create a view of all the columns in the copy\_d\_songs table called view\_copy\_d\_songs.

---

```
CREATE TABLE copy_d_songs  
AS SELECT * from d_songs;
```

---

---

```
CREATE OR REPLACE VIEW view_copy_d_songs  
AS  
SELECT * FROM copy_d_songs;
```

---

Příklady použité v této publikaci jsou založeny na vzorových tabulkách vzdělávacího programu ORACLE Academy.

Další cvičení je možné nalézt na portálu [www.ucimedatabase.cz](http://www.ucimedatabase.cz)