# Databáze

## Relační Databáze

* Relační databáze založená na relačním modelu
* Je založena na tabulkách:
  + - * jejichž řádky chápeme jako záznamy
      * Některé sloupce v nich (Cizí klíče) chápeme tak, že uchovávají informace o relacích mezi jednotlivými záznamy
* Pojem relační databáze definoval Edgar Frank Codd (1970)

### Terminologie

* Základním konstruktorem relačních databází jsou relace
* Sloupce se nazývají atributy
* Řádky se nazývají záznamy
* Každá konkrétní tabulka realizuje podmnožinu Kartézského so učinu

#### Kandidátní Klíč

* Atribut nebo skupina atributů, které jednoznačně identifikují záznam
* Může se stát primárním klíčem
* Ty, které se primárním klíčem nestanou, jsou označovány jako alternativní klíče

#### Primární Klíč

* Jednoznačný identifikátor záznamu, řádku tabulky
* Primárním klíčem může být jediný sloupec nebo kombinace více sloupců tak, aby byla zaručena jeho jednoznačnost
* Nesmí se vyskytovat nedefinovaná prázdná hodnota NULL

#### Cizí klíč

* Slouží pro vyjádření relací, mezi databázovými tabulkami
* Jedná se o pole či skupinu polí, která nám umožní identifikovat, které záznamy z různých tabulek spolu navzájem souvisí

### Integrita databáze

* Znamená. Že data jsou uložena konzistentně vůči definovaným pravidlům
* Lze zadávat jen data, která vyhovují předem definovaným kritériím
* K zajištění integrity slouží integritní omezení
* Jsou to nástroje, které zabrání vložení nesprávných dat či ztrátě nebo poškození stávajících záznamů v průběhu práce
* Typicky je možné zajistit mazání dat, která již ztratila svůj význam

#### Druhy integritních omezení

* **Entitní integritní omezení** – Jde o to, aby v jedné entitě nebylo možno do databáze vložit duplicitní záznamy, jedná se o zajištění unikátnosti
* **Doménová integritní omezení** – Zajištuje dodržování datových typů/domén definovaných sloupcům databázové tabulky
* **Aktivní referenční integrita** – Definuje činnosti, které databázový systém provede, pokud jsou porušena některá pravidla
* **Referenční integritní omezení** – Zabývají se vztahy dvou tabulek, kde jejich relace je určena vazbou primárního a cizího klíče

### Dodržování integritních omezení

* Tři způsoby:

1. Umístění jednoduchých mechanismů pro dodržování integritních omezení na straně databázového serveru. Jedná se o nejlepší způsob z hlediska ochrany dat. Obvykle přináší delší odezvu systému a nelze vždy zajistit jejich přenositelnost na jiný databázový systém
2. Umístění ochranných mechanismů na straně klienta. Pro komfort a nezávislost na databázovém systému je nejlepší volbou. Nutnost kontrolních mechanismů pro každou operaci může způsobit chyby u aplikací a v případě většího počtu aplikací je potřeba je opravit na více místech
3. Samostatné programové moduly na straně serveru. V moderních databázových systémech jsou pro tento účel implementovány tzv. triggery. Jedná se o samostatné procedury, které lze spouštět automatizovaně před a po operacích manipulujících s daty. Tento způsob umožňuje implementaci i složitých integritních omezení. Nevýhody opět přináší provádění na serveru, i velmi omezená možnost přenesení na jiný databázový systém

* Ideálním řešením je kombinace předchozích v závislosti na konkrétních podmínkách

### Vztahy mezi tabulkami

* Vztahy slouží k svázání dat, která spolu souvisejí a jsou umístěny v různých databázových tabulkách
* Každý vztah je charakterizován třemi základními vlastnostmi
  + - Stupněm
    - Kardinalitou
    - Volitelnosti účasti

#### Stupeň vztahu

1. **Unární vztah** – Relace je spojena sama se sebou
2. **Binární vztah** – Klasický vztah mezi dvěma relacemi
3. **Ternární vztah** – Vztah mezi třemi relacemi najednou
4. **N-ární vztah** – Vztah mezi n-relacemi zároveň

* Ternární a n-ární vztahy se nesnadno modelují a v praxi se objevují velice zřídka

#### Kardinalita vztahu

* Kardinalita vztahu je maximální počet vztahů instancí, kterých se může entita účastnit

1. Mezi daty v tabulkách není žádná spojitost, proto nedefinujeme žádný vztah.
2. 1:1 používáme, pokud záznamu odpovídá právě jeden záznam v jiné databázové tabulce a naopak. Takovýto vztah je používán pouze ojediněle, protože většinou není pádný důvod, proč takovéto záznamy neumístit do jedné databázové tabulky. Jedno z mála využití je zpřehlednění rozsáhlých tabulek. Jako ilustraci je možné použít vztah řidiče a automobilu – v jednu chvíli (diskrétní časový okamžik) jeden řidič řídí právě jeden automobil a zároveň jeden automobil je řízen právě jedním řidičem.
3. 1:N přiřazuje jednomu záznamu více záznamů z jiné tabulky. Jedná se o nejpoužívanější typ relace, jelikož odpovídá mnoha situacím v reálném životě. Jako reálný příklad může posloužit vztah autobusu a jím cestujícího pasažéra – v jednu chvíli pasažér jede právě jedním autobusem a v jednom autobuse může zároveň cestovat více pasažérů.
4. M:N umožňuje každému záznamu z jedné tabulky přiřadit libovolný počet záznamů z druhé tabulky, přičemž záznam v druhé tabulce přiřazením k záznamu k první tabulce svou možnost přiřazení „nespotřebuje“, takže jej lze přiřadit k libovolnému počtu záznamů první tabulky. V databázové praxi bývá tento vztah z praktických důvodů nejčastěji realizován kombinací dvou vztahů, a sice 1:N a 1:M, které ukazují do pomocné tabulky, složené z kombinace obou použitých klíčů (jde o třetí, resp. tzv. vazební tabulku). Příkladem z reálného života by mohl být vztah výrobku a jeho vlastnosti – výrobek může mít více vlastností a jednu vlastnost může mít více výrobků. Dalším příkladem je vztah herce a filmu – jeden herec může hrát ve více filmech a v jednom filmu může hrát více herců. M:N je v reálném životě obecně méně častým vztahem, při poněkud odlišném pohledu však existuje velké množství vztahů M:N; to mj. také proto, že často existuje praktická potřeba zachovávat i údaje o historii relačních vztahů (každý jeden řidič za minulost a přítomnost mohl řídit více rozličných automobilů a za volantem každého jednoho automobilu se mohlo vystřídat více řidičů).

### Normální formy

* Normalizace = Zjednodušování a optimalizace navržených struktur databázových tabulek
* Hlavním cílem je navrhnout databázové tabulky tak, aby vykazovaly minimum redundance
  + - **Nultá normální forma** – Tabulka obsahuje alespoň jeden sloupec, tento může obsahovat více druhů hodnot
    - **První normální forma** – Žádný sloupec tabulky nelze dále dělit na části nesoucí nějakou informaci, jinými slovy prvky musí být atomické (žádný sloupec neobsahuje složené hodnoty)
    - **Druhá normální forma** – Tabulka obsahuje pouze sloupce, které jsou závislé na celém klíči
    - **Třetí normální forma** – Mezi neklíčovými sloupci tabulky neexistují žádné závislosti (vztahy)
    - **Čtvrtá normální forma** – Každý sloupec tabulky popisuje pouze jeden fakt nebo jednu souvislost
    - **Pátá normální forma** – Tabulka je ve stavu, že přidáním nového sloupce by se rozpadla na více tabulek
* Nultou normální formu splňuje každá tabulka

### Příklady (relačních databází)

* **MS Acess**
* **MySQL**
* **MariaDB**
* **PostgreSQL**
* **Oracle**
* **Microsoft SQL Server**
* **SQLite**
* **Firebird**

## Datové modelování

* Proces, při němž se definují a analyzují požadavky na strukturu dat
* Výsledkem je datový model
* Je postaveno na přístupu – Princip tří architektur
* Cílem je zachytit a popsat tu část reality, o které chceme uchovávat informace
* Využívá se v mnoha oblastech např.:

Návrh databází a datových uložišť

Integrace informačních systémů

Správa dat

### Princip tří architektur

* Rozděluje proces tvorby databázového systému do tří mentálně zvládnutelných fází, které představují různé úrovně abstrakce
* Každá z vrstev se zaměřuje na jeden z hlavních aspektů vyvíjeného systému:

Obsah

Technologii

Implementační/Realizační specifika

#### Konceptuální úroveň

* První fáze datového modelování
* Je potřeba vymezit veškeré informace, které budeme uchovávat
* Následně vybereme informace, které jsou pro nás důležité
* Všechny tyto informace včetně vztahů mezi nimi musíme věrně popsat
* Ignorujeme technologická či implementační specifika
* Vzhledem k tomu, že z této vrstvy vycházejí zbylé dvě, je potřeba, aby bylo popsané vše, co je nebo bude podstatné pro kohokoli z členů realizačního týmu
* K tvorbě datového modelu na konceptuální úrovni se nejčastěji používá některá z modifikací ER diagramů

#### Technologická úroveň

* Představuje střední míru abstrakce
* V této fázi, obsahově vyplývající z konceptuálního schématu, je zohledněno technologické řešení problému
* Zde se určuje, jak budou uchovávaná data strukturovaná a jak bude toto pojetí v dané technologii realizováno
* Tato úroveň je stále odlehčena od konkrétních implementačních specifik řešení

#### Implementační úroveň

* Představuje nejnižší míru abstrakce
* Není zde prostor pro zjednodušení
* V této fázi probíhá realizace datové struktury, popsané v předchozí technologické úrovni
* Doposud nezávislý technologický model je transformován do modelu, který vyplývá z konkrétních specifik použité databázové platformy
* Musí zohledňovat všechny její dostupné prostředky a možnosti
* Vzniká fyzický model, který si lze představit jako model technologický, rozšířený o některé specifické informace pro konkrétní databázovou platformu

## Datové typy

### Exact numerics – Přesné číselně typy

* Typy jasně definující rozsah a přesnost, vhodné pro operace s čísly, kde je ztráta přesnosti nepřípustná a zároveň operace, kde postačí celočíselná hodnota
  + - **BIT** – 1 bit (0-1)
    - **TINYINT** – 1 byte (0-255)
    - **SMALLINT** – 2 byty (-32 768-32 767)
    - **INT** – 4 byty (-2ˇ31-2ˇ31-1)
    - **BIGINT** – 8 bytů (-2ˇ63-2ˇ63-1)
    - **DECIMAL(p,s)** – Číslo s desetinou čárkou (p=Maximum cifer, s=počet de. Mís.)
    - **NUMERIC** – Ekvivalent pro DECIMAL
    - **SMALLMONEY** – 4 byty (-214,748.3647-214,748.3647)
    - **MONEY** – 8 bytů desetinné číslo na desetitisíciny

### Approximate numerics – Přibližné číselné typy

* Mají obrovský rozsah na úkor přesnosti
* Matematické operace často a rády končí nepřesnostmi, proto využijeme typ právě pro operace s tolerancí k výpočtu
  + - **Float(n)** – Číslo s plovoucí čárkou n(1-53) určuje počet bitů, vždy zabírá 4 nebo 8 bytů
    - **REAL** – To samé jako FLOAT(24)

### Date and time – Datum a čas

* Datové typy pro uchování informace o času nebo datu.
  + - **SMALLDATETIME** – 4 byty (1900-01-01 do 2079-06-06)
    - **DATETTIME** – 8 bytů (1753-01-01 do 9999-12-31)
    - **DATE SQL 2008** – 3 byty (0001-01-01 do 9999-12-31)
    - **TIME(n)** – 3-5 bytů podle uvedené přesnosti, uchovává pouze čas
    - **DATETIME2(n) SQL 2008** – 6-8 bytů podle uvedené přesnosti, nastavitelný a přesnější nástupce DATETIME
    - **DATETIMEOFFSET(n) SQL 2008** – 8-10 bytů podle uvedené přesnosti. Rozšířen o uchování časového pásma

### Character strings – Běžné textové řetězce (1 byte na znak)

* Typy textových hodnot. Nedoporučuje se na kterékoliv texty, co mohou obsahovat nestandardní znaky
  + - **Char(n)** – Řetězec s pevnou délku n znaků (1-8000)
    - **VARCHAR(n)** – Řetězec s variabilní délkou, n udává max. délku (1-8000), použije-li se místo n klíčové slovo MAX – Max. délka je 2ˇ31-1 bytů

### Unicode character strings – Unicode textové řetězce

* Typy pro ukládání řetězců s kódováním UNICODE (1 znak = 2 byty) – Ideální pro všechen text, obsahující diakritiku, či jiné nestandardní znaky
  + - **NCHAR(n)** – To samé co char(n), jen zabírá 2x více místa v paměti a dokáže lépe zpracovat například diakritiku nebo obecně nestandardní znaky
    - **NVARCHAR(n)** – To samé co VARCHAR(n), zabírá 2x více místa v paměti a dokáže lépe zpracovat například diakritiku
    - **NTEXT** – Maximální délka textu, místo tohoto používat NVARCHAR(MAX)

### Binary strings – Binární formáty

* Typy pro ukládání binárních informací. Řešeni pro ukládání souborů, či serializovaných objektů
  + - **BINARY(n)** – Datový typ s pevnou délkou n bytů (1-8000) pro ukládání binárních dat
    - **VARBINARY(n)** – Datový typ s variabilní délkou, kde n udává max. kapacitu v bytech (1-8000) – stejně jako u VARCHAR lze použít slovo MAX
    - **IMAGE** – Místo tohoto se doporučuje používat VARBINARY(MAX)

### CLR Data Types – Datové typy importované z knihoven .NET

* Způsob jak využívat knihoven .NET přímo v SQL Serveru.
  + - **HIERARCHYID** – Uchovává hierarchickou adresu v binární podobě (např. /1/14/67.3/24/). Výhodou je možnost binárního třídění a optimalizací prohledávání

### Spatial Data Types – Prostorové Datové Typy

* Slouží pro uchování geometrických a geografických objektů
  + - **GEOMETRY SQL 2008** – Dokáže uchovávat pozice a geometrické obrazce na rovném souřadnicovém systému
    - **GEOGRAPHY SQL 2008** – Slouží pro uchování pozice a geometrických obrazců – tentokrát však na souřadnicovém systému země

### Other data types – Ostatní datové typy

* Typy, jejichž využití je tak specifické, že nejde kategorizovat
  + - **CURSOR** – Reprezentuje dotaz, jehož výsledky můžeme postupně číst přímo v T-SQL jazyce, nelze využít jako typ sloupce
    - **TABLE** – Slouží k dočasnému uložení datové tabulky, nelze použít jako typ sloupce
    - **SQL\_VARIANT** – Dokáže obsahovat různé datové typy (BIT, INT, DECIMAL, CHAR, BINARY, NCHAR …), CLR datové typy a uživatelské typy. SQL\_VARIANT nedoporučuji používat, pokud máte možnost znát typ dat
    - **TIMESTAMP** – Datum ani čas neukládá. Může být v tabulce pouze jeden takový sloupec, slouží pro verzovací účely
    - **UNIQUEIDENTIFIER** – Jedná se o 16ti bytový unikátní identifikátor, kde každý znak je hexadecimální znak – přes jeho název se může v rámci řádků tabulky opakovat, pokud unikátnost nevynutíte nějakým omezením
    - **XML** – Tento datový typ dokáže ukládat XML struktury a to buď volně nebo podle schématu – dále je možné uložit buď celý XML dokument i s hlavičkou nebo jen jeho část (větev)

### Nejdůležitější datové typy

|  |  |  |
| --- | --- | --- |
| **Datový typ** | **Délka** | **Popis** |
| TINYINT | 1 bajt | Rozsah od -128 do +127 (bez znaménka: od 0 do 255) |
| SMALLINT | 2 bajty | Rozsah od -32 768 do +32 767 (bez znaménka: od 0 do 65 535) |
| MEDIUMINT | 3 bajty | Rozsah od -8 388 608 do +8 388 607 (bez znaménka: od 0 do 16 777 215) |
| INT,INTEGER | 4 bajty | Rozsah od -2 147 483 648 do +2 147 483 647 (bez znaménka: od 0 do 4 294 967 295) |
| BIGINT | 8 bajty | Rozsah od -9 223 372 036 854 775 808 do +9 223 372 036 854 775 807 (bez znaménka: od 0 do 18 446 744 073 709 551 615) |
| BIT, BOOL | 1 bajt | synonymum pro typ TINYINT(1) |
| FLOAT |  | Rozsah od -3.402823466E+38 do +3.402823466E+38 |
| DOUBLE |  | Rozsah od -1.7976931348623157E+308 do +1.7976931348623157E+308 |
| DOUBLE PRECISION, REAL |  | synonymum pro typ DOUBLE |
| DECIMAL(m,d), DEC(m,d), NUMERIC(m,d |  | rozsah nastaví parametry "m" a "d", maximální rozsah je stejný jak pro typ DOUBLE |
| DATE | '0000-00-00' | datum ve formátu "rok-měsíc-den" respektive "RRRR-MM-DD". Rozsah od 1000-01-01 do 9999-12-31 |
| DATETIME | '0000-00-00 00:00:00' | datum a čas. Rozsah od 1000-01-01 00:00:00 do 9999-12-31 23:59:59 (format = "RRRR-MM-DD HH:MM:SS") |
| TIMESTAMP(n) | '0000-00-00 00:00:00' | - datum a čas v rozsahu od 1970-01-01 00:00:00 do 2037-01-01 00:00:00 (vždy se ukládá všech 14 čísel !)  - formát zobrazení (a pro dotazy) lze nastavit parametrem "m" s hodnotou 14 (nebo chybějící), 12, 10, 8, 6, 4, nebo 2  - "RRRRMMDDHHMMSS", "RRMMDDHHMMSS", "RRMMDDHHMM", "RRRRMMDD", "RRMMDD", "YYMM", "YY"  - pokud do buňky tohoto typu nic nezapíšeme, pak MySQL sám doplní aktuální čas změny v daném řádku |
| TIME | '00:00:00' | časový rozsah je od "-838:59:59" do "838:59:59" (formát "HH:MM:SS") |
| YEAR(n) | 0000 | YEAR(4) = rozsah od 1901 do 2155, (formát "RRRR"), YEAR(2) = rozsah od 1970 do 2069 |
| CHAR(m) |  | - délka řetězce "m" může být v rozsahu od 0 do 255  - pokud je vložený řetězec kratší než nastavíme, pak jsou chybějící znaky automaticky doplněny mezerami (má tedy "pevnou" délku)  - CHAR (bez "m") je považováno za CHAR(1) |
| VARCHAR(m) |  | - délka řetězce "m" může být v rozsahu od 0 do 255  - pokud je vložený řetězec kratší než nastavíme, pak se chybějící znaky nedoplňují (má tedy "plovoucí" velikost), ale navíc se ukládá informace o jeho délce |
| TINYBLOB, TINYTEXT |  | Délka řetězce je max. 255 znaků |
| BLOB, TEXT |  | Délka řetězce je max. 65 535 znaků |
| MEDIUMBLOB, MEDIUMTEXT |  | Délka řetězce je max. 16 777 215 znaků |
| LONGBLOB, LONGTEXT |  | Délka řetězce je max. 4 294 967 295 znaků |
| ENUM('item1','item2',...) |  | - pole předem definovaných řetězců (itemů) o max. počtu 65 535  - v buňce tabulky pak může být pouze jeden z itemů, které jsou předdefinované  - místo názvů 'item' můžeme používat i jejich pořadí, tedy: 1 (místo 'item1'), 2 (místo 'item2')... |
| SET('item1','item2',...) |  | - pole předem definovaných řetězců (itemů) o max. počtu 64  - v buňce tabulky pak může být i více z itemů, které jsou předdefinované |

## Základní tvar příkazu select (Select, Where, Order by, Group by, Having)

### Základní select

SELECT \*

FROM XXX

* Příklad:

SELECT ID, JMENO, PRIJMENI

FROM SEZNAM\_JMEN

### Základní select & Where

SELECT \*

FROM XXX

WHERE ‚POLE‘ = ‚PODMÍNKA‘

* Příklad:

SELECT ID, JMENO, PRIJMENI

FROM SEZNAM\_JMEN

WHERE ID IN (0, 1, 2)

### Základní select & Order By

SELECT \*

FROM XXX

WHERE ‚POLE‘ = ‚PODMÍNKA‘

ORDER BY ‚POLE1‘, ‚POLE2‘

* Příklad:

SELECT ID, JMENO, PRIJMENI

FROM SEZNAM\_JMEN

WHERE ID IN (0, 1, 2)

ORDER BY JMENO, PRIJMENI

### Základní select & Group By

SELECT \*

FROM XXX

WHERE ‚POLE‘ = ‚PODMÍNKA‘

GROUP BY ‚POLE1‘

* Příklad:

SELECT ID, JMENO, PRIJMENI

FROM SEZNAM\_JMEN

WHERE ID IN (0, 1, 2)

GROUP BY JMENO

### Základní select & Having

SELECT \*

FROM XXX

GROUP BY ‚POLE1‘

HAVING ‚PODMÍNKA‘

* Příklad:

SELECT ID, JMENO, PRIJMENI

FROM SEZNAM\_JMEN

GROUP BY JMENO

HAVING COUNT(JMENO) > 1

### Definice

* Pojmy:
  + **SELECT** – Příkaz používaný pro vybrání dat z databáze
  + **WHERE** – Příkaz používaný pro filtrování záznamů
  + **GROUP BY** – Příkaz používaný pro sjednocení řádků se stejnou hodnotou
  + **ORDER BY** – Příkaz používaný pro seřazení záznamů
  + **HAVING** – Omezuje výsledek GROUP BY dle podmínky

## Joiny

### INNER JOIN

* Nejjednoduší typ spojení tabulek
* Vrací záznamy z levé tabulky, které mají odpovídající záznam v pravé tabulce
* Výsledkem je průnik množin

### LEFT JOIN

* Nejpoužívanější typ spojení tabulek
* Vrací všechny záznamy z levé tabulky
* K nim připojí odpovídající záznamy z pravé tabulky

### RIGHT JOIN

* Opak spojení LEFT JOIN
* Vrací všechny záznamy z pravé tabulky
* K nim připojí odpovídající záznamy z levé tabulky

### OUTER JOIN

* Nazýváno také jako FULL OUTER JOIN, FULL JOIN
* Vrací všechny záznamy z obou tabulek
* Záznamy které odpovídají spojí

### LEFT Excluding JOIN

* Vrací ty záznamy z levé tabulky, které nemají odpovídající záznam v pravé tabulce

### RIGHT Excluding JOIN

* Vrací ty záznamy z pravé tabulky, které nemají odpovídající záznam v levé tabulce

### OUTER Excluding JOIN

* Není v MySQL
* Vrací záznamy z levé tabulky, které nemají odpovídající záznam v pravé tabulce a zároveň záznamy z pravé tabulky, které nemají odpovídající záznam v levé tabulce

### CROSS JOIN

* Slouží ke spojení 2 tabulek v relační databázi
* Jeho výsledkem je kartézský součin = kombinace levé a pravé tabulky

## Agregační funkce

* Statistické funkce, pomocí kterých systém řízení báze dat umožňuje seskupit vybrané řádky dotazů a spočítat nad nimi výsledek určité aritmetické nebo statistické funkce
* Agregační funkce se v SQL používají s konstrukcí GROUP BY

### Agregace

* Při výběru řádků z tabulky většina databázových strojů relačních databází dovoluje výsledné řádky seskupit podle zadaného sloupce nebo výrazu z nich složeného pomocí syntaktické konstrukce GROUP BY

### Nejčastější agregační funkce

* **AVG()** – Aritmetický průměr
* **SUM()** – Součet
* **COUNT()** – Počet
* **MIN()** – Minimum
* **MAX()** - Maximum

## Databázová transakce

* Skupina příkazů, které převedou databázi z jednoho konzistentního stavu do druhého

### Vlastnosti

* Databázové transakce musí splňovat tzv. vlastnosti ACID
  + - Atomicitu
    - Konzistenci
    - Izolovanost
    - Trvalost

#### Atomicita

* Databázová transakce je jako operace dále nedělitelná
* Provede se buď jako celek, nebo se neprovede vůbec

#### Konzistence

* Transakce převádí databázi z jednoho konzistentního stavu na druhý
* Její účel je zajistit určité dynamické integritní omezení
* Např. nelze ukončit převod peněz z účtu na účet bez toho, aby jak operace odečtení ze zdrojového účtu tak operace připočtení na cílový účet byly obě převedeny

#### Izolovanost

* Operace uvnitř transakce jsou skryty před vnějšími operacemi
* Vrácením transakce není zasažena jiná transakce, a když ano, i tato musí být vrácena
* V důsledku tohoto chování může dojít k tzv. řetězovému vrácení

#### Trvalost

* Změny, které se provedou jako výsledek úspěšných transakcí, jsou skutečně uloženy v databázi, a již nemohou být ztraceny

### Rozsah

* **Lokální transakce** – Probíhá pouze v jediném uzlu
* **Globální transakce** – Přesahuje rozsah jednoho uzlu

### Způsob zpracování

#### Pesimistický

* Zpracování se v jeho průběhu změny zaznamenávají do dočasných objektů a teprve po přesunu či změně dat se odznačí příznak dočasnosti a data se tak stanou platnými

#### Optimistický

* U zpracování se předpokládá, že při transakci nenastane chyba a provedené změny nebude třeba vrátit zpět. Měněné záznamy v tabulkách jsou při optimistickém zpracování transakce zapisovány „natvrdo“, současně s tím se však, coby seznam SQL příkazů, vytváří tzv. rollback log, na základě něhož je transakcí zastřešené změny možné vrátit zpět. V případě že při transakci dojde k nějaké nezotavitelné chybě, tento log se provede a transakce skončí ve výchozím stavu, načež se vyvolá chyba. V opačném případě, tedy když transakce proběhne bez výskytu nezotavitelné chyby, se na jejím konci rollback log stává bezpředmětným.

### Žurnály

* Záznamy, které uchovávají informace o průběhu transakcí a slouží k zotavení po vzniklé chybě. Žurnály musí být v každém uzlu a obsahují záznamy o historii transakce

#### Stavy

* **Aktivní** – Od počátku provádění transakce
* **Částečně potvrzený** – Po provedení poslední operace transakce
* **Chybný** – Nelze pokračovat v normálním průběhu transakce
* **Zrušený** – Po skončení operace ROLLBACK
* **Potvrzený** – Po úspěšném vykonání operace COMMIT

### Prováděné operace

* Pro práci s transakcemi je nutné zavést následující operace
  + - **BEGIN** – Začátek transakce
    - **COMMIT** – Ukončení transakce a uložení dosažených výsledků do databáze
    - **ROLLBACK** – Odvolání změn, není-li definován savepoint, návrat do stavu před započtením vykonávání transakce

### Transakční bezpečnost

* Charakteristický prvek transakce je to, buď proběhne úspěšně jako taková, nebo – v případě nějakých problémů a komplikací – se vrátí do původního stavu, tedy, nestane se, že by se proces „zadrhnul“ někde uprostřed. Vlastnost, která dokonání transakce v celku nebo bezpečný návrat do původního stavu dokáže zjistit, se nazývá transakční bezpečnost