Databázové systémy I. – II. 2009/2010

Relační databáze, databázový server, tabulka

Relační databáze (systém řízení báze dat) - sada nástrojů které umožňují spolehlivě a efektivně ukládat data a manipulaci s daty. Jedná se o vrstvu mezi uživatelem a daty. Relační databáze jsou založeny na principu teorie množin. Umožňují tedy základní množinové operace jako:

- **sjednoceni** - nová tabulka obsahuje všechny záznamy z první tabulky a všechny záznamy z druhé tabulky,
- **průnik** - nová tabulka obsahuje pouze totožné řádky z obou tabulek,
- **množinový rozdíl** - nová tabulka bude mít všechny řádky z první tabulky krom těch které se vyskytuji ve druhé,
- **symetrické rozdíl** - vytvoří tabulku ve, které budou všechny záznamy z obou tabulek krom těch, které mají obě tabulky společné,
- **kartézský součin** - vytvoří tabulku, která bude mít počet sloupců roven součtu sloupců obou vstupních tabulek a počet záznamů bude roven součinu všech řádků první tabulky se všemi řádky druhé tabulky.

A dále je pak možné využít **speciální** množinové operace:

- **projekce** - výběr sloupců z tabulky,
- **restrikce** - výběr řádků z tabulky,
- **spojení** - výsledkem je zřetězení řádků ze spojovaných tabulek na základě nějaké shodné položky.

Databázový server - sada programových prostředků umožňujících organizaci a zpřístupnění dat pro klienty. Klient může data číst nebo zapisovat.

Tabulka - představuje základní stavební prvek databáze. Vše je uloženo ve formě tabulek. Mezi tabulkami je možné definovat vztahy:

- **1:N** - jeden záznam první tabulky odpovídá více záznamům v druhé tabulce,
- **1:1** - jeden záznam první tabulky odpovídá jednomu záznamu v druhé tabulce,
- **N:M** - více záznamů jedné tabulky odpovídá více záznamům ve druhé tabulce. Tento vztah se řeší pomocí třetí spojovali tabulky.

Vlastnosti tabulek:

- pořadí řádků je nevýznamné (obvykle jsou řazeny podle toho jak byly do tabulky vkládány),
- pořadí sloupců je nevýznamné,
- na sloupce se odkazujeme pomoci atributu (název sloupce v záhlaví) tabulky,
- neobsahuje duplicitní sloupce ani řádky.

Jazyk SQL a jeho vývoj, příkazy DDL, DML, DCL

Jazyk SQL vznikl na základě výzkumu relačních databází v laboratořích IBM. Původní označení bylo SEQUEL (Structured English Quary Language). Později jazyk přejmenován na dnes známý SQL (Structured Query Language). Došlo ke standardizaci jazyka:

- SQL86 - obsahoval nedostatky v oblasti integrity dat,
- SQL92 - nahrazoval nedostatky předchozí verze, též označován jako SQL2,
- SQL99 - reaguje na potřeby databází s objektově orientovanými prvky.

Úloha jazyka je poskytnout uživatelům přístup a umožnit manipulaci s daty pomoci následujících příkazů:

- **DDL** - **příkazy** pro definici dat (Data Definition Language) umožňují tvorbu, úpravu a mazání struktury databázových objektu (tabulka, index, pohled, …). Do této kategorie spadají příkazy `CREATE, ALTER, DROP`.
- **DML** - **příkazy** pro manipulaci s daty (Data Manipulation Language) umožňují vkládat, upravovat, mazat a vybírat data z tabulek. Jedné se o příkazy `SELECT, DELETE, INESRT, UPDATE`.
- **DCL** - **příkazy** pro řízeni dat (Data Control Language) slouží k nastavování, změnám a odebírání přístupových prav. Jde o příkazy `GRANT, REVOKE, CREATE USER, ALTER USER, DROP USER`. Tato skupina příkazů ještě obsahuje podskupinu pro řízení transakcí (Transaction Control Commands) `COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION`.
- Ostatní příkazy, do této kategorie patří příkazy pro správu databáze jako je např. nastavování systémových proměnných.

Datové typy SQL a datové typy Oracle

Datové typy definované ve standardu SQL:

- `DECIMAL(X,Y)` - desetinné číslo s celkovým počtem X číslic a s počtem Y za desetinou čárkou,
- `FLOAT(X)` - desetinné číslo s plovoucí desetinou čárkou,
- `REAL(X)` - desetinné číslo s plovoucí desetinou čárkou rozdíl od předchozího je v maximálním počtu číslic, který udává parametr X,
- `INTEGER` - celé čísla,
- `SMALLINT` - celé číslo s menším rozsahem než předchozí,
- `CHAR(X)` - řetězec znaků pevné délky,
- `VARCHAR(X)` - řetězec znaků proměnlivé délky,
- `DATE` - vyjádření data,
- `TIME` - vyjádření času,
- `TIMESTAMP` - počet sekund od 1.1.1970.

Platforma Oracle tyto základní typy ještě rozšiřuje o **datové typy Oracle**:

- `NUMERIC(X,Y)` - desetinné číslo,
- `CHAR(X)` - umožňuje delší řetězce jak standardní typ až 2000 znaků,
- `VARCHAR2(X)` - umožňuje delší proměnlivé řetězce jak standardní typ 4000 znaku,
- `LONG` - řetězec znaků max. 2GB,
- `DATE` - vyjádření data a času,
- `TIMESTAMP(X)` - X značí přesnost sekund,

- `INTERVAL YEAR` (rozsah roku) `TO MONTH` - vyjádření delších časových údajů,
- `INTERVAL DAY`(rozsah dni) `TO SECOND`(přesnost sekund) - k vyjádření kratšího časového rozsahu,
- `BLOB` - binární objekty do 4GB,
- `CLOB` - znakové objekty do 4GB,
- `NCLOB` - textové objekty v národních sadách 4GB.

Příkaz SELECT - základní syntaxe, projekce, restrikce, aliasy, setřídění výsledků, klauzule DISTINCT, DISTINCTROW

Příkaz SELECT se používá pro cílený výběr dat z tabulek. Příkazy SELECT je možné do sebe vnořovat. Lze jej použít i ke tvorbě kopie (`CREATE T AS SELECT …`) tabulky z vykonaného výběru.

- **projekce** výběr sloupců z požadované tabulky
- **restrikce** výběr řádků z požadované tabulky.

Řazení výsledků se provádí pomocí klauzule `ORDER BY` sloupec `ASC/DESC`

- `ASC` - vzestupně
- `DESC` - sestupně

Aliasy představují jiné pojmenování pro sloupec, tabulku. Na alias se lze odkazovat jako na sloupec, tabulku, který zastupuje. Bez aliasu se neobejdeme při spojeni tabulky sama se sebou.

- **DISTINCT** - ve výsledku nebudou obsaženy duplicitní řádky, které mají stejné hodnoty ve vypisovaných sloupcích.
- **DISTINCTROW** - ve výsledku nebudou duplicitní řádky, které mají stejné hodnoty ve všech sloupcích bez ohledu zda jsou vypisovány

Základní syntaxe příkazu SELECT je následující:

`SELECT` seznam_sloupců `FROM` seznam_tabulek `WHERE` možná_podmínka `ORDER BY` sloupec_podle_kterého_se_řadí `ASC/DESC`

Příkazy INSERT, UPDATE, DELETE

Jedná se o příkazy spadající do kategorie příkazů pro manipulaci s daty **DML**. Umožňují tedy vkládání, aktualizování a mazání dat z tabulek.

- **INSERT** - příkaz sloužící pro uložení jednoho řádku do tabulky, nebo více řádků pomocí vloženého dotazu. `INSERT INTO` (seznam_sloupců) `VALUES` (seznam_hodnot).
- **UPDATE** - příkaz pro aktualizaci hodnot řádků. Pomoci klauzule `WHERE` se uvádí podmínka pro které řádky má aktualizace proběhnout. V případě že podmínka není uvedena změna se týká všech řádku.
- **DELETE** - příkaz pro mazání řádků z tabulky. Pro mazané řádky se používá podmínka `WHERE` které vyhovuji mazané řádky. V případě že není uvedena dojde ke smazání všech řádků tabulky.

Hodnota NULL, operátory, výrazy, podmínky

Hodnota NULL představuje prázdnou hodnotu nikoliv nulovou. Se zavedením této hodnoty bylo nutné zavést do podmínek obsahujících logické výrazy TRUE, FALSE ještě jeden termín UNKNOWN čímž se vyhodnocování výrazů mění.

A	B	A and B	A or B	not A
True	True	*True*	*True*	*False*
True	False	*False*	*True*	*False*
True	Unknown	*Unknown*	*True*	*False*
False	True	*False*	*True*	*True*
False	False	*False*	*False*	*True*
False	Unknown	*False*	*Unknown*	*True*
Unknown	True	*Unknown*	*True*	*Unknown*
Unknown	False	*False*	*Unknown*	*Unknown*
Unknown	Unknown	*Unknown*	*Unknown*	*Unknown*

Operátory:

- logické operátory AND, OR, NOT,
- matematické operátory - umožňují na základě původních hodnot vypočítat hodnoty nové jedná se o +, -, *, /,
- operátor zřetězení - používá se ke spojení sloupců do jedné hodnoty ||.
- Operátory pro porovnávání - umožňují porovnávat hodnoty =,<>,>,<,>=,<=. Dále pak do této kategorie spadají operátory:
  - `BETWEEN` x `AND` y - větší nebo rovno x a menší nebo rovno y,
  - `IN` - patří do množiny,
  - `ANY`, `SOME` - porovnání hodnot s každou hodnotou v seznamu. Musí být doplněn jedním z operátorů =,>,<,<=,<=,
  - `ALL` - porovnání hodnot s každou hodnotou v seznamu musí být taktéž doplněn o operátory jako výše,
  - `EXISTS` - ve vnořeném dotazu je vracen alespoň jeden řádek
  - `IS NULL` - test na hodnoty NULL,
  - x `LIKE` y - porovnává řetězec s maskou. Je možné použít zástupné znaky_ právě jeden znak nebo **%** nula nebo více znaku.
- Množinové operátory - používají se ke spojení výsledků dvou dotazu. Obě spojované množiny musí mít stejný formát (počet sloupců a typ).
  - `UNION` - sjednocení množin - ve výsledku jsou všechny řádky obou dotazu krom duplicit.
  - `UNION ALL` - ve výsledku jsou všechny řádky včetně duplicit

- `INTERSECT` - průnik - ve výsledku jsou řádky které se vyskytují v obou dotazech.
- `MINUS` - rozdíl - ve výsledku jsou řádky které se vyskytli v prvním dotazu a nejsou ve výsledku druhého dotazu.

Výraz je skupina konstant, proměnných a funkcí spojených pomocí operátorů. Výsledkem je hodnota jejíž datový typ je odvozen z položek výrazu. Ve výrazu lze použít název sloupce, textové a číselné konstanty, výsledek funkce, hodnotu NULL nebo vnořené dotazy.

Podmínky se používají k omezení dotazu a značí se klíčovým slovem `WHERE` následovaným výrazem pro podmínku. Výrazy je možný řetězit pomocí logických operátorů.

Vnitřní a vnější spojení tabulek

Vnitřní spojení (inner join) - spojení v němž jsou záznamy dvou tabulek kombinovány pouze tehdy existuje li k záznamům první tabulky odpovídající položka v tabulce druhé. Sloupce přes které se realizuje spojeni se uvádí v podmínce `WHERE` nebo je možné využít klauzule `JOIN` tabulka `ON` podmínka_spojení.

Vnější spojení (outer join) - spojení kdy je každý záznam ze dvou tabulek kombinován do jednoho záznamu ve výsledném dotazu. Není li k tabulce poskytující všechny záznamy nalezena odpovídající hodnota v tabulce druhé je vypsáno **prázdné pole** v místě kde nebyla nalezena shoda. Spojeni se realizuje pomoci:

- `RIGHT JOIN` - ve výsledku budou všechny řádky z pravé (druhé) tabulky a nebyla li nalezena shoda v levé (první) tabulce budou pole vyznačeny jako `NULL`,
- `LEFT JOIN` - ve výsledku budou zahrnuty všechny řádky z levé tabulky a pro řádky pro které nebyla nalezena shoda v pravé tabulce bude prázdné pole,
- `FULL JOIN` - ve výsledku budou všechny řádky z pravé I z levé tabulky.

Tabulky je možné spojovat samu se sebou a nebo s ostatními tabulkami.

Další možnosti spojení:

- `NATURAL JOIN` - umožňuje spojit tabulky bez uvedení podmínky jejich spojeni, ta je vybrána automaticky dle shodných sloupců,
- `JOIN USING` (podmínka) - v případě že je shodných sloupců v více tak `NATURAL JOIN` nebude fungovat. Pomoci tohoto příkazu lze vyjmenovat sloupce přes které se spojení provede.

Souhrnné a skupinové dotazy, agregační funkce

Agregační funkce pracují nad množinou řádků a vrací jeden výsledek pro celou množinu. Sloupcové (agregační) funkce ignorují hodnotu `NULL`. Mezi agregační funkce patří:

- `AVG(sloupec)` - průměr,
- `COUNT(sloupec)` - počet výskytů,
- `MAX(sloupec)` - vrací maximum,
- `MIN(sloupec)` - vrací minimum,
- `SUM(sloupec)` - vrací součet hodnot.

Souhrnné dotazy pracuji s agregačními funkcemi a slouží k vypsání souhrnných informaci jako je například počet zaměstnanců firmy.

Skupinové dotazy umožňují mezi součty které se vztahují k jednotlivým skupinám. Pod pojmem skupina si lze představit pobočku, město, … Všechny sloupce které nejsou agregovány musí být uvedeny za klauzuli `GROUP BY`. Omezení výsledku skupinového dotazu lze pomoci klauzule `HAVING` podmínka_pro_agregované_veličiny. Omezení za klauzuli `HAVING` se aplikuji až po vytvoření skupin.

Vnořené dotazy, množinové operátory

Výsledkem **vnořeného dotazu** je virtuální tabulka a nebo jedna hodnota. Vnořené dotazy představují několik SELECTU vložených do sebe. Vnořený dotaz musí být zapsán v závorkách. Nejprve se vyhodnocuje vnořený dotaz a pak je nad jeho výsledky proveden hlavní dotaz. Vnořený dotaz lze pojmenovat aliasem a pak se na něj odkazovat jako na tabulku. Vnořený dotaz může být umístěn:

- za klauzuli FROM,
- za klauzuli WHERE,
- za klauzuli HAVING.

Vnořené dotazy mohou byt:

- jednořádkové a jedno sloupcové - vrací jednu hodnotu též označován za skalární pod dotaz,
- jedno sloupcové a víceřádkové - vrací sloupec (virtuální tabulka s jedním sloupcem),
- vícesloupcové a více řádkové - vrací virtuální tabulku

Množinové operátory - používají se ke spojení výsledků dvou dotazu. Obě spojované množiny musí mít stejný formát (počet sloupců a typ).

- UNION - sjednocení množin - ve výsledku jsou všechny řádky obou dotazů krom duplicit.
- UNION ALL - ve výsledku jsou všechny řádky včetně duplicit.
- INTERSECT - průnik - ve výsledku jsou řádky které se vyskytují v obou dotazech.
- MINUS - rozdíl - ve výsledku jsou řádky které se vyskytují v prvním dotazu a nejsou ve výsledku druhého dotazu.

Funkce jazyka SQL, práce s datovým typem DATE

Funkce představuje pojmenovaný blok příkazů který provádí požadované operace. Na tento blok se lze odvolávat. Rozlišujeme funkce a procedury kde procedura nemá návratovou hodnotu. Funkce lze rozdělit:

Jednořádkové:

- funkce pro práci se znaky
- konverzní- používají se pro převod mezi typy
    - bin_to_num
    - to_char
    - to_date
- matematické
- funkce pro práci s časem a datem

- ostatní

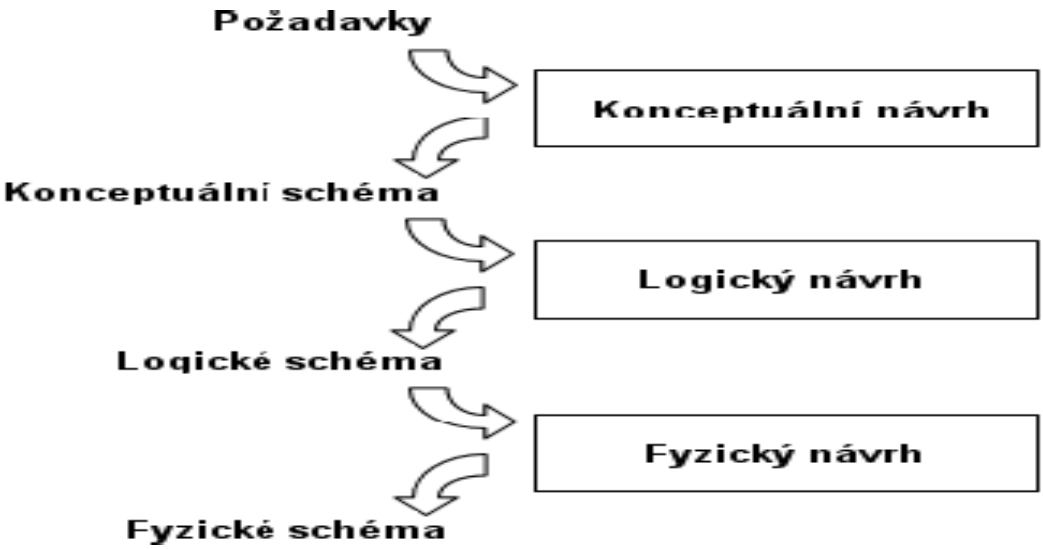
Agregační funkce:

- `AVG(sloupec)` - průměr,
- `COUNT(sloupec)` - počet výskytů,
- `MAX(sloupec)` - vrací maximum,
- `MIN(sloupec)` -vrací minimum,
- `SUM(sloupec)` - vrací součet hodnot.

Funkce pro práci s datovým typem date:

- `EXTRACT`(část `FROM` z čeho) - slouží pro získání části rok, den mesic, … z data,
- `NEXT_DAY`(datum, den_v_tydnu) - vrací nové datum,
- `LAST_DAY`(datum) - poslední den v měsici,
- `ADD_MONTHS`(datum,n-měsíců) - posun o n měsíců
- `MONTHS_BETWEEN`(datum, datum) - počet měsíců mezi dvěma daty,
- `SYSDATE` - vrací aktualní datum,
- `LOCALTIMESTAMP` - vrací počet sekund od 1.1.1970 v lokálním časovém pásmu,
- `CURRENT_TIMESTAMP` - vrací aktuální počet sekund,
- `ROUND`(datum,část) - zaokrouhlení data na část year, month, …
- `TRUNC`(datum, část) - ořezaní data

Fáze návrhu databáze, E-R diagramy



V první fázi se prostřednictvím konzultací se zadavatelem a uživateli formulují požadavky na to co má být v databázi uloženo. Definuje se účel (k čemu databáze slouží) a úkony které může uživatel s daty provádět Výsledkem první fáze je **konceptuální model** který popisuje data na abstraktní úrovni. Výstupem konceptuálního modelu je **schéma** (E-R model, který zavedl Peter Chen)

- Entita - objekt reálného světa.

- Vztah - vazba mezi dvěma nebo více entitami.
- Atribut - vlastnost entity.

V ER modelování se nejprve definují nezávislé entity, vztahy a závislé entity. V druhém kroku se pak formulují atributy a klíče. Závislá entita je závislá na jiných entitách (řádek faktury na faktuře a výrobku).

- **Identifikující relace** - primární klíč jedné entity je primárním klíčem v jiné entitě.
- **Neidentifikující relace** - primární klič jedné entity je neklíčovým atributem jiné entity.

Vztahy mezi tabulkami:

Kardinalita - vyjadřuje skutečnost kolik řádku jedné tabulky může vstoupit do vztahu s řádky jiné tabulky 1:1, 1;N, N:M.

Parcialita vztahu - značí povinnost či nepovinnost existence vztahu (0,1), (1,1),(1:N), (0,N)

Další fáze tvorba relačního modelu transformací ER diagramu.

Normální formy, normalizace tabulek, dekompozice

Normální formy se používají pro lepší návrhy databázových systémů a pro efektivní ukládání dat. Obecně platí že čím je tabulka ve vyšší normální formě tím je návrh kvalitnější.

Dekompozice - rozpad jedné tabulky do tabulek více se zachováním závislostí a bez ztrátovosti.

0. NF - tabulka je v nulté normální formě jestliže obsahuje pole které nemá atomickou hodnotu.
1. NF - tabulka je v první normální formě jestliže obsahuje pouze pole která mají atomické hodnoty. Atributy nelze rozdělit.
2. NF - tabulka je v druhé normální formě jestli splňuje podmínky první a navíc platí že všechny neklíčová pole jsou závislá na celém klíči nikoliv na jeho části. Tedy z toho plyne že toto má smysl řešit pouze pokud máme složený klíč.
3. NF - tabulka je v třetí normální formě pakliže splňuje podmínku druhé normální formy a zároveň neexistuje závislost mezi neklíčovými sloupci.
4. NF - tabulka je ve čtvrté normální formě jeli ve třetí a zároveň popisuje pouze jeden fakt.
5. NF - tabulka která je ve čtvrté a není možné do ní vložit nový sloupec tak aby se vlivem skrytých závislostí rozpadla na několik dílčích. Relaci již není možno bezztrátově rozložit

BCNF - tabulka je v BCNF jeli ve třetí a zároveň musí platit I pro hodnoty mezi klíči. Neboli atributy, které jsou součástí primárního klíče, musí být vzájemně nezávislé

Vytváření tabulek, integritní omezení, primární a cizí klíče

Tabulky je možno vytvářet, mazat nebo měnit pomoci DDL příkaz. jako `CREATE TABLE, ALTER TABLE` tabulka (`ADD` sloupec, `MODIFY` úprava sloupce, `DROP COLUMN` sloupec, `RENAME COLUMN` sloupec `TO` sloupec2), `DROP TABLE`. Při tvorbě tabulky je možné uvést omezující podmínky pro sloupce jede především o:

- `NOT NULL` – hodnoty ve sloupci nemohou byt prázdné, tedy musí být zadané,
- `UNIQUE` – sloupec nepovoluje duplicity,

- `PRIMARY KEY` - primární klíč zakázány `NULL` a duplicity,
- `FOREIGN KEY` - cizí klíč - odkazuje se na sloupec jiné tabulky - podpora pro udržení integrity dat,
- `CHECK` - podmínka kterou musí splňovat vkládaná hodnota,
- `DEFAULT` - výchozí hodnota sloupce.

Integrita dat - fakt že data věrně zobrazují reálný stav který popisují. V případě výskytu nekonzistence dochází ke znehodnocení dat. Důvody ke vzniku chyb mohou být:

- vložení špatné hodnoty - vložená data neodpovídají realitě,
- nedostatečná aktualizace.

Referenční integrita - při rušení řádků v jedné tabulce může nastat situace že se na tento řádek odkazují řádky z jiné tabulky.

Zajištění integrity tedy vede k:

- stanoveni typů sloupců a určitých délek,
- stanoveni pouze určitých hodnot pro sloupce (výčet),
- jedinečnost pomocí `UNIQUE`,
- kontrola zadání dat `NOT NULL`,
- kontrola vztahu s jinou tabulkou se řeší pomocí `FOREIGN KEY`.

Integritu dat lze zajistit jednak v databázi ale I v aplikaci což se považuje pouze za doplněk.

Primární klíč - používá se k adresaci řádku, při jeho výběru klademe důraz na paměťové nároky, primární klíč může být složený z více polí.

Cizí klíč - používá se k zajištění referenční integrity, musí splňovat pravidlo že je buď plně zadaný nebo plně nezadaný.

Pohledy jejich význam, sekvence

Pohled představuje pojmenovaný dotaz který lze použít jako tabulku. Generuje se dynamicky podle dat vypočítaných v okamžiku použití. Používají se zejména ke zjednodušení konstrukce složitějších dotazů nebo jako bezpečnostní opatření. Lze do něj za určitých okolností data vkládat. Pohledy rozdělujeme na:

Jednoduché:

- vytvořené z dat jedné tabulky,
- neobsahují žádné funkce,
- lze v nich provádět změny.

Komplexní:

- jsou vytvořeny z více tabulek,
- mohou obsahovat funkce nebo skupiny,
- změnu dat umožňují jen velmi omezeným způsobem.

Pohled se vytváří příkazem `CREATE WIEV` nazev_pohledu `AS SELECT` … Pohledy je možné uzamknout pouze pro čtení.

Sekvence umožňují generovat jednoznačné identifikátory nebo posloupnosti. Vytvářejí se příkazem `CREATE SEQUENCE` název a parametry. Při použití sekvencí se využívají dva pseudo sloupce:

`NEXTVAL` - generuje další hodnotu sekvence,

`CURRVAL` - vrací aktuální hodnotu sekvence.

Indexy - druhy indexu a jejich význam

Záznamy v tabulkách jsou neutříděné a často v pořadí v jakém byly zadány. V případě hledání v takovéto oblasti dat je nutné projít minimálně polovinu intervalu aby se dosáhlo požadované hodnoty. Tento problém lze vyřešit pomocí indexu. **Index** je databázový objekt které v sobě nese informace o tom kde se daný záznam nachází. V moderních databázích se obvykle používají B-stromy které výrazně snižují čas při hledání Protože průchod stromem je oproti sekvenčnímu prohledáváni pole rychlejší.

Vlastnosti:

- Přináší zrychlení pouze tehdy chceme li najít několik záznamu.
- Přináší zvýšení režie při vkládání a úpravách dat. Protože se musí upravit I indexový objekt.

Nedoporučuje se používat indexy na málo variabilním sloupci krom bitmapových. BITMAP INDEX - slouží k zakódování málo variabilních sloupců pomocí bitů. Index se vytváří pomocí `CREATE INDEX` název_indexu `ON` název_tabulky (sloupec)

Použití indexu je vhodné:

- nad sloupci podle kterých probíhá řazení,
- nad sloupci kde hledáme maximum, minimum.

Zabezpečení a ochrana dat - uživatelské účty, systémová a objektová oprávnění, role

Cíle k zavedeni bezpečnostní politiky:

- důvěrnost - informace by neměla být přístupná nekompetentním osobám,
- integrita - modifikovat data muže jen kompetentní uživatel.

Bezpečnostní mechanismy/ ochrana dat

Pohledy:

- umožňují omezit přístup k určitým sloupcům,
- možnost omezit přistup I na úrovni řádku,
- možnost skrýt skutečnou strukturu tabulky.

Přístup k databázi

Aby bylo možné do databáze přistupovat je nutné mít vytvořen uživatelský účet. Účet je možné vytvořit příkazem `CREATE USER` jméno `IDENTYFIED BY` heslo. Heslo si pak může uživatel změnit příkazem `ALTER USER` jméno `IDENTIFYED BY` nove_heslo. Práva pro vykonávání příkazu je pak možné uživateli přidělit příkazem `GRANT CREATE SESSION, CREATE TABLE TO` jméno. Odebrat práva pak pomocí `REVOKE CREATE TABLE FROM` jmeno.

Systémová přístupová práva - umožňuji provádět operace na úrovni schématu nebo databáze.

- `CREATE PROCEDURE`
- `CREATE SEQUENCE`
- `CREATE SESSION`
- `CREATE SYNONYMUM`
- `CREATE TABLE`
- `CREATE TRIGGER`
- `CREATE USER`
- `CREATE VIEW`

Objektová přístupová práva - umožňují provádět operace s objekty (tabulky, indexy, sekvence, procedury, ...) patřící jinému uživateli. Práva může udělovat sám vlastník objektu.

Tabulky, pohledy:

`ALTER, DELETE, INDEX, INSERT, UPDATE< DELETE, SELECT, EXECUTE`

Role je pojmenovaná skupina práv kterou lze přidělovat uživatelům Byly zavedeny kvůli zjednodušení správy uživatelských práv. Role je možné vytvořit příkazem `CREATE ROLE` název. Do takto vytvořené role je pak možné přidat příkazy které budou v roli obsaženy a to pomoci `GRANT SELECT, INSERT, UPDATE, DELETE ON` tabulka `TO` jméno_uživatele. Uživatelům pak lze přiřadit skupinu práv příkazem `GRANT` název_role `TO` uživatel1, uživatel2, …Příkazem `SET ROLE` název_role aktivujeme příslušnou skupinu práv.

Oracle nabízí ještě řadu dalších možností jak zvýšit bezpečnost:

- zajištění přístupu pouze z určité aplikace,
- doplnit dotaz o podmínku která omezí uživateli přistup k určitým datům. Dva uživatelé mohou dostat různé výsledky při stejném dotazu,
- možnost šifrování komunikace i tabulek samotných.

Transakce, návratové body, automatické zamykáni, konzistentní čtení

Transakce umožňuje seskupit jeden nebo více příkazů do jednoho bloku, který bude vykonán atomicky. Transakce proběhne tedy buď jako celek nebo se neprovede vůbec. Cílem transakce je zajistit konzistentní čtení. Změny provedené v transakci jsou pro ostatní uživatele viditelné až po potvrzení transakce. Při zahájení transakce je vytvořen prostor kde jsou uchovávány staré údaje, aby bylo možné transakci odvolat. Transakce jsou vytvářeny implicitně. Transakci je možné pojmenovat příkazem `SET TRANSACTION NAME` nzev_transakce.

Transakce je ukončena buď:

- po dokončení bloku příkazů a potvrzením `COMMIT` v takovém případě se změny provedené v transakcí stávají trvalé,
- návratem zpět na začátek pomoci příkazu `ROLLBACK`,
- dojde k výpadku spojení mezi uživatelem a databází. V takovém případě se transakce ODROLLUJE zpět.
- Explicitně když se provede příkaz DDL tedy jakýkoliv příkaz CREATE, `ALTER`, `DROP` způsobí ukončení aktivní transakce a zahájení nové.

Způsoby odvolání transakce/ návratové body

- Statemanet-levet rollback - pokus o vložení duplicit primárního klíče, narušení referenční integrity.
- Rollback to savepoint - odvoláni transakce do **bodu návratu**, který musí být předem vytvořen příkazem `SAVEPOINT` název_bodu_návratu.

Konzistentní čtení zajišťuje správné a odpovídající hodnoty.

Na úrovni SQL příkazu - Oracle řeší automaticky, vyvolaný dotaz vidí data ve stavu ve kterém se nacházela před jeho spuštěním. Ani potvrzeni `COMMIT` jiné transakce původní dotaz neovlivní. Dotaz nevidí změny které sám provedl.

Na úrovni transakcí - data se kterými transakce pracuje odrážejí stav v jakém se data nacházela při spuštění transakce krom změn provedených transakcí samotnou (platí pro mód `SERIALIZABLE`).

Automatické zamykání - řeší konkurenční přístup více klientů k datům. Vede to k tomu že uživatel který provedl změny vidi tyto změny sám a ostatní k nim nemají přístup pokud je nepotvrdí. Zamykání je možné na úrovní řádků ale I celých tabulek.

Zámky:

- exkluzivní - zamčený zdroj nemůže být sdílen,
- sdílený - sdílený zdroj může být sdílen pro čtení.

Přístup k databázi z vyššího programovacího jazyka, bezpečnost SQL injection

SQL injection - jedná se o metodu podsunutí škodlivého kódu do SQL dotazu diky čemuž se původní smysl dotazu změní což vede k rizikům jako jsou:

- získání přístupu k citlivým datum,
- možnost změnit data,
- likvidace tabulek.

Řešením je kontrolovat vstupní hodnoty na povolené typy a případně odstranit nežádoucí znaky např. pomoci regulárních výrazů.

Účet pod kterým je z webové aplikace přistupováno by měl mít jen ta práva která skutečně potřebuje. Ideální řešení je volat pouze uložené procedury a příslušný účet by měl práva pouze na jejich spouštění.

Aplikace by neměla zobrazovat chybové hlášky.

Databáze může sloužit jako podklad pro webové aplikace. K databázi je proto možné **přistupovat z řady programovacích jazyku** které nabízejí programové prostředky pro práci s databází např. PHP, Java. ODBC (mezivrstva) snaží se poskytnout standardní rozhraní pro všechny DB servery, ovladače pak řeší konkrétní realizace.

Jazyk PL/SQL, proměnné, syntaxe bloku

Jazyk PL/SQL – (Transaction Processing Language) procedurální jazyk, umožňuje používat klasické programátorské konstrukce jako jsou funkce, podmínky a cykly. Což v SQL nešlo tam se kód provádí sekvenčně.

PL/SQL umožňuje:

- deklarovat konstanty, proměnné, kurzory
- chyby ošetřit pomocí výjimek

Příkazy jsou uzavřeny v bloku, které je možné do sebe i vkládat. Blok kódu vypadá následovně (**syntaxe bloku**)

```
DECLARE
      deklarace proměnných
BEGIN
      vykonávaný kód
EXCEPTION
      ošetření výjimek
END;
```

Proměnné:

- je nutné před použitím vždy deklarovat,
- během deklarace je možné proměnou inicializovat, případně omezit že nesmí být `NULL`,
- je možně definovat jednu proměnnou na základě jiné proměnné,
    - v_var `VARCHAR2`(255);
    - v_var2 v_var%`TYPE`;
- je možné vytvořit proměnou a typ určit na základě sloupce
    - v_var tab.sloupec%`TYPE`
- zápis do proměnných v bloku PL/SQL lze provést pomocí klauzule `INTO. SELECT` jméno,příjmení `FROM` tabulka `INTO` v_jméno, v_příjmení.

Řízení toku programu - podmínky, cykly, kurzory, záznamy, ošetření chyb

Podmínky

```
IF vyraz THEN
      prikazy
ELSE
      prikazy
END IF;
```

Příkaz CASE slouží k vícenásobnému větvení programu.

```
CASE
        WHEN podminka1 THEN prikazy;
        WHEN podminka2 THEN prikazy;
        ...
        ELSE prikazy;
END CASE;
```

Cykly LOOP, FOR, WHILE

```
LOOP
      Příkazy;
      IF vyraz THEN
            EXIT;
      END IF;
END LOOP;

FOR x IN 1..100 LOOP
      prikazy
END LOOP;



WHILE vyraz LOOP
      prikazy
END LOOP;
```

Kurzory

Pro každý příkaz SQL vytvoří databázový server pracovní oblast v paměti. Kurzory mohou být **implicitní** jsou vytvářeny automaticky a nebo **explicitní** obsluhované programátorem. Kurzor v každém momentě identifikuje právě jeden řádek vráceného výsledku. Základní kroky pro práci s kurzorem:

- deklarace kurzoru `CURSOR` název `IS SELECT` …,
- otevření kurzoru `OPEN CURSOR`,
- výběr dat prostřednictvím kurzoru `FETCH` název_kurzoru `INTO` seznam_proměnných,
- uzavření kurzoru `CLOSE` název.

Stavy kurzoru:

- název_kurzoru%`ROWCOUNT` - zjištění aktuálního pořadového čísla, pokud nebyl vybrán žádný záznam je 0,
- název_kurzoru%`FOUND` - pokud příkaz `FETCH` načte nějaký záznam má hodnotu TRUE,
- název_kurzoru%`NOTFOUND` - používá se pro zajištění konce cyklu,
- název_kurzoru%`ISOPEN` - test zda je kurzor otevřen.

Záznamy

Struktura která obsahuje více položek s různými typy.

```
DELCARE
TYPE nazev_zaznamu IS RECORD(
      jmeno VARCHAR2(255);
      prijmeni VARCHAR2(255);
};
```

nebo

```
DECLARE
zaznam tabulka%ROWTYPE;
```

Pomocí záznamů lze pracovat s kurzory efektivněji.

```
DECLARE
      zanzam tabulka%ROWTYPE;
      CURSOR k1 IS SELECT …
BEGIN
      FOR zaznam IN k1
      LOOP
            prikazy
      END LOOP;
END;
```

Kurzory je také možné parametrizovat. Parametr bude dosazen až během otevření kurzoru.

```
DECLARE
      zanzam tabulka%ROWTYPE;
      CURSOR k1(_id INTEGER) IS SELECT …WHERE id>=_id;
BEGIN
      FOR zaznam IN k1(100)
      LOOP
            prikazy
      END LOOP;
END;
```

Ošetření chyb

Existují dva druhy chyb syntaktické a logické ty první se projeví již v době překladu. Chyby logické je možné ošetřit pomocí výjimek. Nejčastější výjimky:

- `NO_DATA_FOUND`,
- `TOO_MANY_ROWS`.

```
BEGIN
      příkazy
EXCEPTION
      WHEN název_výjimky THEN příkazy
END;
```

Výjimku je možné vyvolat příkazem `RAISE` název_výjimky. Oracle nabízí i definování vlastních výjimek. Aktuální kód chyby lze získat pomocí funkce `SQLCODE` a její textový popis funkci `SQLERRM`.

Procedury, funkce a balicky

Procedura je pojmenovaný blok příkazů na který se lze odvolávat. Procedura může mít parametry, které použije ve výpočtech.. Proceduru je možné vytvořit příkazem `CREATE PROCEDURE` název (parametry). Vyvolat jí pak lze pomoci `EXECUTE` název_procedury.

Funkce se liší od procedury tím že může vracet nějakou hodnotu. Vytváří se příkazem:

```
CREATE FUNCTION název (parametry)
      RETURN NUMBER AS návrat NUMBER
BEGIN
      RETURN navrat;
END;
```

Funkce je možné použít v příkazu SELECT.

Balíčky podporují zapouzdření to znamená že můžeme skrýt procedury které mimo balíček nemají využití. Podporují proměnné uchovávané po celou dobu relace. Umožňují seskupení souvisejících funkcí. Balíček se skládá ze dvou částí **specifikace balíčku** a **tělo balíčku**. Ve specifikaci jsou uvedeny deklarace proměnných, kurzorů, výjimek. V těle balíčku jsou pak obsaženy definice a případné privátní deklarace. Tvoří se pomocí příkazů `CREATE PACKAGE název AS` a `CREATE PACKAGE BODY název AS`.

Triggery pro DML, příkazy nad tabulkami, triggery pro databázové a klientské události

Trigger je uložená procedura, která se spouští za přesně stanovených událostí. A to před nebo po uskutečnění události.

Použití:

- kontrola zadaných hodnot,
- zajištění referenční integrity,
- zajištění logování.

Neměli by se používat pokud je místo nich možné použít integritní omezení jako například `NOT NULL, UNIQUE, CHECK`, … Je možné je vykonat buď jednorázově nebo pro každý ovlivněný řádek.Trigger je možné vytvořit příkazem:

```
CREATE TRIGGER název
      BEFORE/AFTER UPDATE OR DELETE tabulka
      FOR EACH ROW
BEGIN
      příkazy;
END;
```

V těle triggeru je možné používat **:old.sloupec** (značí staré hodnoty u delete, update) nebo **:new.sloupec**.(nové hodnoty u insert, update). INSTEAD OF trigger jsou určeny jen pro pohledy a umožňují provádět akce na tabulkách z kterých je pohled vytvořen. Příkazy `COMMIT, ROLLBACK` není možné použít v těle triggeru protože DDL příkazy implicitně volají `COMMIT`.

Triggery pro databázové události `STARTUP, SHUTDOWN, DB_ROLE_CHANGE, SERVERERROR`

Triggery pro klientské události `BEFORE/AFTER LOGON/LOGOFF, ALTER, DROP`

Export, import dat, práce s textovými soubory a XML dokumenty

Export a import používáme pro výměnu dat mezi jinými aplikacemi. Mezi často používané formáty patří:

- SQL - textový soubor obsahující příkazy SQL,

- XML soubory,
- soubory tabulkových procesorů.

Obvykle IDE (např. SQL developer) přímo nabízí import/export dat. Exportovat lze databáze, schémata nebo vybrané prvky. Pro export je nutné mít dostatečná oprávnění.

Migrace dat je proces převodu dat z jedné platformy do platformy druhé.

Práce s textovými soubory - k tomuto účelu se používá utilita SQL*Loader. Tabulka do níž importujeme musí existovat

XML (značkovací jazyk) stále více prosazován jako univerzální formát pro výměnu dat. **XML dokument** obsahuje kromě dat samotných i jejich strukturu. XML dokumenty jsou uloženy v repositářích které umožňují přístup k těmto dokumentům skrze FTP, HTTP. Oracle umožňuje vypsat tabulky, výsledky dotazu přímo v XML formátu pomoci funkce `SYS_XMLGEN`.

Datový typ `SYS.XMLType` – umožňuje data ukládat přímo ve formátu XML

Relační algebra, závislosti

Relační algebra

Pro práci s tabulka v relační databázi je nutné definovat alespoň základní aparát, který nám umožní zpracovávat data z tabulek. K těmto činnostem slouží prostředek nazývaný **relační** algebra. Relační algebra je nejzákladnějším prostředkem pro práci s tabulkami. Mezi základní operace patří **projekce, selekce a spojení. Tyto operace lze definovat takto:**

Projekce tabulky (relace) R s položkami **A** na množinu položek **B**, kde **B**$\subseteq$A:

- vytvoří tabulku s položkami **B** a záznamy, které vzniknou z původní tabulky **R** odstraněním položek **A-B**. Odstraněny jsou i eventuelně opakující se záznamy.
- -značení : R[B]

Selekce tabulky **R** s položkami **A** podle logické podmínky **Φ**:

- -vytvoří tabulku s týmiž položkami a ponechá ty záznamy z původní tabulky, které splňují logickou podmínku **Φ**. Podmínka **Φ** je logický výraz, jehož formule nad jednoduchými položkami tabulky mají tvar:

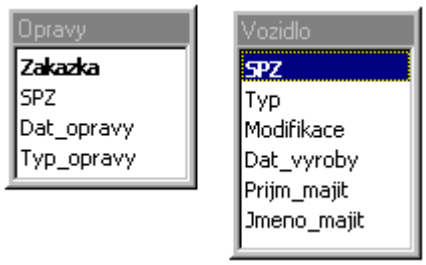
  $\Phi = t_1 \Theta t_2$
  kde:
  t_1, t_2 jména položek tabulky, konstanty nebo výrazy
  Θ logické operátory - $\Theta \in \{<, >, =, \leq, \geq\}$

- -značení: R(Θ)

Spojení tabulek **R** a **S** se položkami **A** a **B**:

- vytvoří minimalizovanou tabulku se schématem **A**$\cup$B a se záznamy, jejichž projekce na **A je z tabulky R a projekce na B je z tabulky S.**
- -značení: R * S

U spojení pracujeme s několika možnými případy. Nejčastější je **spojení přes rovnost** (**přirozené spojení).** Výsledná tabulka obsahuje jen ty záznamy z obou zdrojových tabulek, které se shodují v položkách, přes které se spojení provádí. Pokud některý záznam z jedné tabulky nemá odpovídající záznam v tabulce druhé, ve výsledné tabulce se neobjeví.
Takový postup se nám však ve všech případech nehodí, proto je nutno používat další typy spojení, **levé -polospojení a pravé -polospojení**, které z určené tabulky (stojící v operaci jako levý či pravý argument) připojí všechny záznamy, bez ohledu na to, zda mají odpovídající záznam v druhé zdrojové tabulce. Použití polospojení je důležité tam, kde není programově zajištěna plná **integrita dat**, případně pokud nemáme jistotu, že nemůže být porušena.
Příklad jednotlivých operací relační algebry si ukážeme na dvou tabulkách - OPRAVY a VOZIDLO. Tabulky OPRAVY a VOZIDLO jsou definované položkami podle obrázku Y.1. Tyto tabulky obsahují jednotlivé záznamy podle obrázku Obr. Y.2 a Obr. Y.3.



Obr. Y.1: Struktura tabulek příkladu.

	Zakazka	SPZ	Dat_opravy	Typ_opravy
▶	1011	OPM99-99	3.5.2000	běžná
	1023	OVX19-19	7.5.2000	reklamace
	1025	OVX19-19	7.5.2000	běžná
	1085	OPM99-99	10.7.2000	havárie

Obr. Y.2: Tabulka OPRAVY.

	SPZ	Typ	Mod	Dat_vyroby	Prijm_majit	Jmeno_majit
▶	FMI35-28	Škoda Š105	M	26.10.1985	Novák	Karel
	FMP10-20	Ford K		12.1.2000	Dvořák	Vladimír
	OPM99-99	Opel Vectra		30.6.1997	Novotný	Petr
	OVX19-19	Škoda Felicia	LX	15.2.1995	Novák	Josef

Obr. Y.3: Tabulka Vozidlo.

Na těchto tabulkách provedeme jednotlivé operace:
a) Projekce tabulky OPRAVY položkami Dat_opravy, Zakázka. Výsledek je na Obr. Y.4.

	Dat_opravy	Zakazka
	3.5.2000	1011
	7.5.2000	1023
	7.5.2000	1025
	10.7.2000	1085

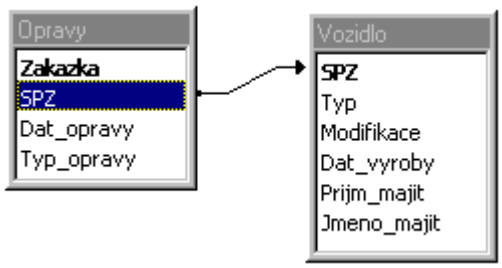
Obr. Y.4: Projekce OPRAVY[Dat_opravy,Zakázka].

b) Selekce tabulky OPRAVY.
Podmínka pro selekci je dána výrazem **Φ:** Dat_opravy = 7.5.2000
Výsledná tabulka je na obrázku Obr. Y.5.

Zakazka	SPZ	Dat_opravy	Typ_opravy
1023	OVX19-19	7.5.2000	reklamace
1025	OVX19-19	7.5.2000	běžná

Obr. Y.5: Selekce OPRAVY[Dat_opravy = 7.5.2000].

c) Spojení tabulek OPRAVY a VOZIDLA.
Spojení těchto tabulek se děje pomocí spojení přes rovnost položek OPRAVY.SPZ a VOZIDLA.SPZ (viz Obr. Y.6). Vznikne tabulka obsahující záznamy, jejichž struktura je dána souhrnem všech položek spojovaných tabulek. Takto vzniklá tabulky je ukázkou nejjednodušší formy spojení.



Obr. Y.6: Vazba položek při spojení.

Zakazka	Dat_opravy	SPZ	Typ	Mod
1011	3.5.2000	OPM99-99	Opel Vectra	
1023	7.5.2000	OVX19-19	Škoda Felicia	LX
1025	7.5.2000	OVX19-19	Škoda Felicia	LX
1085	10.7.2000	OPM99-99	Opel Vectra	

Obr. Y.7: Spojení OPRAVY*VOZIDLO přes OPRAVY.SPZ = VOZIDLO.SPZ (zobrazeny jen některé položky tabulky).

Z příkladu a předchozích definic je zřejmé, že operace **projekce** vybírá sloupce, operace **selekce** vybírá řádky (záznamy) a operace **spojení** provádí spojení dvou tabulek přes zvolenou položku nebo skupinu položek.

Zavislosti

*Nechť A, B jsou atributy relace R. Pak atribut B je **funkčně závislý** na atributu A, pokud v čase existuje ke každé hodnotě atributu B nejvýše jedna hodnota atributu A. Funkční závislost atributu B na atributu A budeme zapisovat* $A \rightarrow B$.

*Atribut B je **silně funkčně závislý** na atributu A, když je na něm funkčně závislý a zároveň neexistuje žádná podmnožina A´ složeného atributu A taková, že B funkčně závisí na A´.*

*Nechť A, B, C jsou atributy relace R. Pokud B je funkčně závislý na A a C je funkčně závislý na B a zároveň platí, že A funkčně nezávisí na B, pak C je **tranzitivně závislý** na A.*