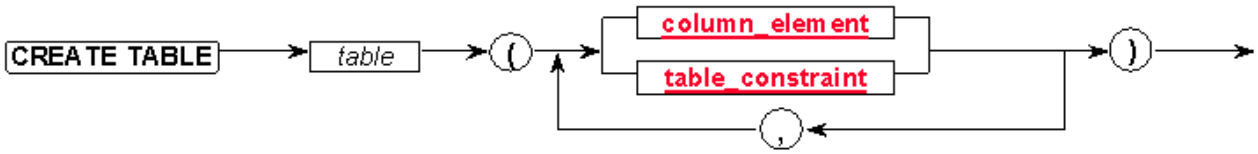
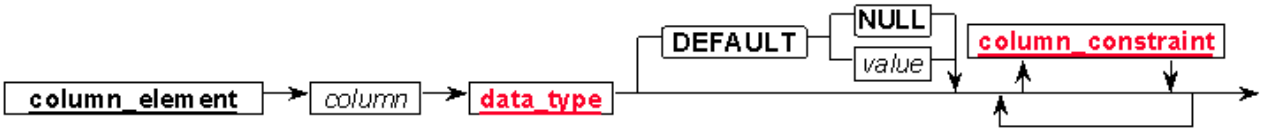
CREATE TABLE
- CREATE TABLE = definice tabulek a jejich IO
- ALTER TABLE = změna definice schématu
- aktualizace - INSERT INTO = vkládání
  - UPDATE = modifikace
  - DELETE = mazání

CREATE TABLE
Základní konstrukce
- vytvoření schématu a prázdnétabulky (pokud již není tabulka s daným jménem založena)
- tabulka má definovány - sloupce (atributy a související atributová IO)
  - tabulková IO



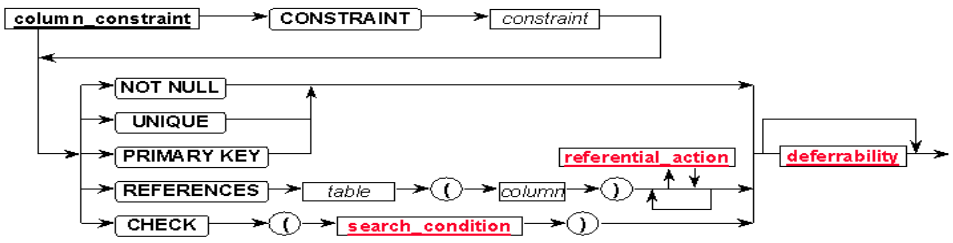
Definice sloupce - každý sloupec má vždy přiřazen
- datový typ data_type - VARCHAR (int = *délka řetězce*) = typ char
  - BIT (int) =
  - INTEGER = int
  - FLOAT (int)
  - DECIMAL (int,int = *desetinné číslo*)
- nepovinně je možno specifikovat
  - implicitní hodnotu v nově vytvořeném záznamu (**DEFAULT NULL** | ***value***)



  - sloupcové integritní omezení

PŘ - **CREATE TABLE** Výrobek (
Id **INTEGER**,
Název **VARCHAR**(128),
Cena **DECIMAL**(6,2),
Datum Výroby **DATE**,
JeNaSkladě **BIT DEFAULT** TRUE,
Hmotnost **FLOAT**
);

LOKÁNÍ INTEGRITNÍ OMEZENÍ SLOUPCE / ATRIBUTU
- sloupcové IO umožňuje omezit množinu platných hodnot daného atributu v rámci záznamu (nového nebo modifikovaného)
  - pojmenované **CREATE TABLE** ... (..., **CONSTRAINT** constraint ...)
  - nepojmenované
- 5 typů omezení na platnou hodnotu atributu **NOT NULL** = hodnota musí být nenulová
  - **UNIQUE** – hodnota musí být unikátní (v rámci všech řádků v tabulce)
  - **PRIMARY KEY** – definuje primární klíč (sémanticky totéž jako **NOT NULL** + **UNIQUE**)
  - **REFERENCES** – definuje jednoatributový cizí klíč (oba atributy musí být kompatibilní)
  - **CHECK** – obecná podmínka, stejně jako u příkazu **SELECT** ... **WHERE**
    - s tím rozdílem, že se vyhodnotí pouze na vkládaném řádku (řádcích)
    - při vyhodnocení podmínky na TRUE je hodnota atributu platná



PŘ – atributové IO

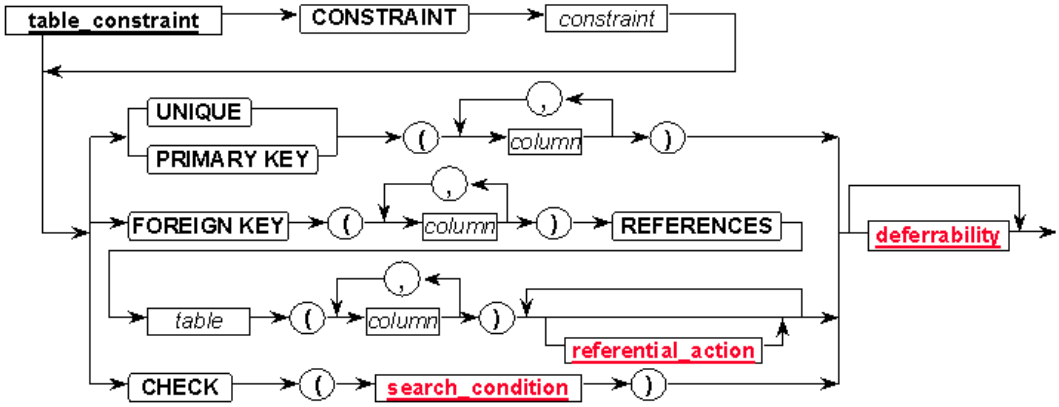
```
CREATE TABLE Výrobek (
        Id INTEGER CONSTRAINT pk PRIMARY KEY,
        Název VARCHAR(128) UNIQUE,
        Cena DECIMAL(6,2) NOT NULL,
        DatumVýroby DATE,
        JeNaSkladě BIT DEFAULT TRUE,
        Hmotnost FLOAT,
        Výrobce INTEGER REFERENCES Výrobce (Id)
);

CREATE TABLE Výrobce (
        Id INTEGER PRIMARY KEY,
        JménoVýrobce VARCHAR(128),
        Sídlo VARCHAR(256)
);
```

PŘ – cizí klíč

```
CREATE TABLE Zaměstnanec (
        IdZam INTEGER PRIMARY KEY,
        Jméno VARCHAR(128),
        Šéf INTEGER REFERENCES Zaměstnanec (IdZam)
);
```

GLOBÁLNÍ IO = IO celé tabulky
= zobecnění atributových IO pro kombinace hodnot více sloupců (kromě NOT NULL, to má smysl pouze u jednotlivých atributů)
- **UNIQUE**–n-ticehodnot je unikátní
- **FOREIGN KEY**–stejnějako **REFERENCES**u atributu
- **CHECK**



PŘ -

```
CREATE TABLE Výrobek (
        Id INTEGER PRIMARY KEY,
        Název VARCHAR(128) UNIQUE,
        Cena DECIMAL(6,2) NOT NULL,
        DatumVýroby DATE,
        JeNaSkladě BIT DEFAULT TRUE,
        Hmotnost FLOAT,
        Výrobce VARCHAR(128),
        SídloVýrobce VARCHAR(256),
        CONSTRAINT fk FOREIGN KEY (Výrobce, SídloVýrobce) REFERENCES Výrobce
                (JménoVýrobce, Sídlo)
);

CREATE TABLE Výrobce (
        JménoVýrobce VARCHAR(128),
        Sídlo VARCHAR(256),
        OborČinnosti VARCHAR(64),
        CONSTRAINT pk PRIMARY KEY(JménoVýrobce, Sídlo)
);
```

PŘ – CHECK

```
CREATE TABLE Výrobek (
        Id INTEGER PRIMARY KEY,
        Název VARCHAR(128) UNIQUE,
        Cena DECIMAL(6,2) NOT NULL,
        DatumVýroby DATE,
        JeNaSkladě BIT DEFAULT TRUE,
        Hmotnost FLOAT, CONSTRAINT chk CHECK ((Id = 0 OR Id > ALL (SELECT Id
                FROM Výrobek)) AND Hmotnost > 0)
);
```

REFERENČNÍ INTEGRITA

- při aktualizaci tabulky **referující** (ta, pro kterou je IO definováno) nebo **referované** může nastat porušení integrity cizích klíčů
    - pokus o vložení/aktualizaci záznamu s hodnotou cizího klíče, která se nevyskytuje v sloupci referované tabulky
    - pokus o smazání záznamu z referované tabulky, když pro mazanou hodnotu klíče existuje reference
- při porušení integrity cizích klíčů mohou nastat dva případy chování
    - hlášení chyby aktualizace, pokud není definována referenční akce (SQL 89)
    - vykonání referenční akce **referential_action** (SQL 92)
        - **ON UPDATE**,**ON DELETE** = podmínka spuštění akce, při modifikaci referované hodnoty nebo smazání řádku v referované tabulce
        - **CASCADE** = dotčený záznam s referujícíhodnotou se takésmaže, resp. aktualizuje novou hodnotou
        - **SET NULL** = referující hodnotou dotčeného záznamu se nastaví na NULL
        - **SET DEFAULT** = referující hodnotou dotčeného záznamu se nastaví implicitní hodnotu definovanou v CREATE TABLE
        - **NO ACTION** = implicitní, neprovede se nic, resp. SŘBD ohlásí chybu

PŘ – ON UPDATE, ON DELETE

```
CREATE TABLE Výrobek (
        Id INTEGER CONSTRAINT pk PRIMARY KEY,
        Název VARCHAR(128) UNIQUE,
        Cena DECIMAL(6,2) NOT NULL,
        DatumVýroby DATE,
        JeNaSkladě BIT DEFAULT TRUE,
        Hmotnost FLOAT,
        Výrobce INTEGER REFERENCES Výrobce (Id)
                ON DELETE CASCADE
);

CREATE TABLE Výrobce (
        Id INTEGER PRIMARY KEY,
        JménoVýrobce VARCHAR(128),
        Sídlo VARCHAR(256)
);
```

ALTER TABLE
- změna definice schématu atributy – přidání/odebrání atributu, změna DEFAULT hodnoty
  – přidání/odebrání IO
- pozor, v tabulce už mohou být data, která nedovolí změnit IO (např. zavést IO primární klíč)

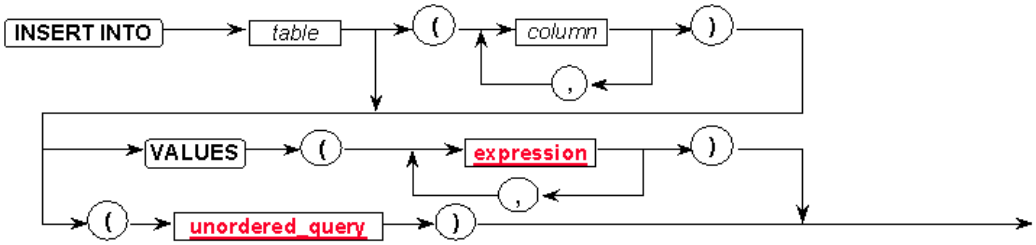
ALTER TABLE table-name ...
- **ADD** [**COLUMN**] column-name column-definition ... = přidej sloupec
- **ADD** constraint-definition ... = změna
- **ALTER** [**COLUMN**] column-name SET ... =
- **ALTER** [**COLUMN**] column-name DROP ... =
- **DROP COLUMN** column-name ... = smaž sloupec
- **DROP CONSTRAINT** constraint-name =

PŘ -
```
CREATE TABLE Výrobek (
        Id INTEGER PRIMARY KEY,
        Název VARCHAR(128) UNIQUE,
        Cena DECIMAL(6,2) NOT NULL,
        DatumVýroby DATE,
        JeNaSkladě BIT DEFAULT TRUE,
        Hmotnost FLOAT,
        CONSTRAINT chk CHECK ((Id = 0 OR Id > ALL (SELECT Id FROM Výrobek)) AND
                Hmotnost > 0)
);
ALTER TABLE Výrobek DROP CONSTRAINT chk
ALTER TABLE Výrobek ADD CONSTRAINT chk CHECK ((Id = 0 OR Id > ALL (SELECT Id FROM
        Výrobek)) AND Hmotnost > 10)
);
```

DROP TABLE
- **DROP TABLE *table***
- komplementárník příkazu **CREATE TABLE *table***
- smaže se jak obsah tabulky, tak i její definice, tj. schéma tabulky
- **DELETE FROM *table*** (viz dále) = pokud chceme vymazat pouze obsah tabulky (ne záhlaví – resp. schéma), použijeme příkaz

MODIFIKACE DAT
SQL obsahuje kromě SELECT tři příkazy pro manipulaci s daty
- INSERT INTO = vložení řádků
- DELETE FROM = vymazání řádků
- UPDATE = aktualizace hodnot v řádcích

INSERT INTO
- vkládání řádku výčtem hodnot, dvě možnosti
  - **INSERT INTO** table (col1, col3, col5) **VALUES** (val1, val3, val5)
  - **INSERT INTO** table **VALUES** (val1, val2, val3, val4, val5)
- vkládání více řádků, jejichž hodnoty vzniknou jako výsledek dotazu
  - **INSERT INTO** table1 | (výčet atributů) |(**SELECT** ... **FROM** ...)



PŘ - **CREATE TABLE** Výrobek (
Id **INTEGER CONSTRAINT** pk **PRIMARY KEY**,
Název **VARCHAR**(128) **UNIQUE**,
Cena **DECIMAL**(6,2) **NOT NULL**,
DatumVýroby **DATE**,
JeNaSkladě **BIT DEFAULT** TRUE,
Hmotnost **FLOAT**,
Výrobce **INTEGER REFERENCES** Výrobce (Id)
);

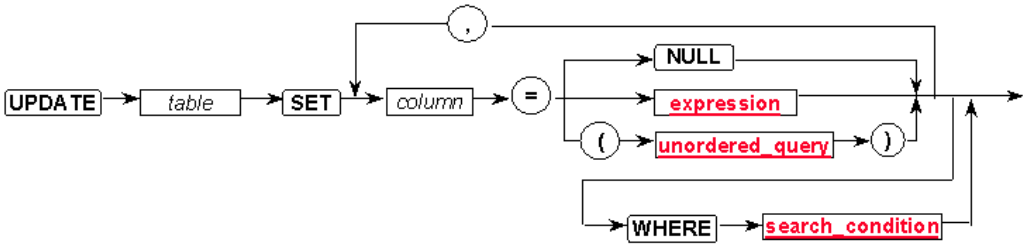
INSERT INTO Výrobek **VALUES** (0, 'Koště', 86, '2005-5-6', TRUE, 3, 123456)

INSERT INTO Výrobek (Id, Název, Cena, DatumVýroby, Hmotnost, Výrobce) **VALUES** (0, 'Koště', 86, '2005-5-6', 3, 123456)

CREATE TABLE VýrobekNaSkladě (
Id **INTEGER PRIMARY KEY**,
Název **VARCHAR**(128) **UNIQUE**,
Cena **DECIMAL**(6,2)
);

INSERT INTO VýrobekNaSkladě **VALUES** (**SELECT** Id, Název, Cena **FROM** Výrobek **WHERE** JeNaSkladě = TRUE)

UPDATE
- aktualizace záznamů splňujících podmínku
- hodnoty zvolených atributů vybraných záznamů jsou zvlášť nastaveny na
  - NULL
  - hodnotu expression (např. konstanta)
  - výsledek dotazu



PŘ - **UPDATE** Výrobek **SET** Název = 'Notebook' **WHERE** Název = 'Laptop'

UPDATE Výrobek **SET** Cena = Cena * 0.9 **WHERE CAST**(DatumVýroby **AS** VARCHAR(32)) < '2003-01-01'

UPDATE Výrobek **AS** V1 **SET** Hmotnost = (**SELECT AVG**(Hmotnost) **FROM** Výrobek **AS** V2 **WHERE** V1.Název = V2.Název)

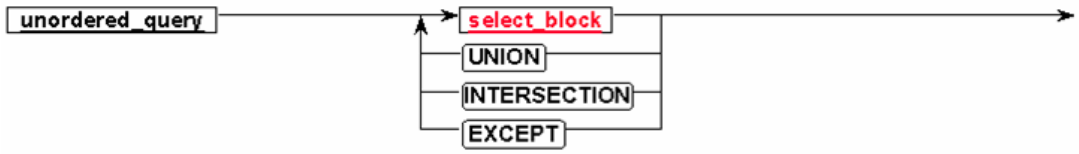
DELETE FROM
- vymaže záznamy, které splňují podmínku
- **DELETE FROM *table*** = vymaže všechny záznamy

PŘ - **DELETE FROM** Výrobek **WHERE** Cena > 100

DOTAZY V SQL - ZÁKLADNÍ KONSTRUKCE DOTAZU

NETŘÍDĚNÝ DOTAZ – schema
- příkaz **SELECT** = hlavní logika dotazování
- případně z příkazů - **UNION** = sjednocení
  - **INTERSECTION** = průnik
  - **EXCEPT** = rozdíl

  dvou nebo více výsledků získaných dotazem popsaným v příkazu **SELECT**
- výsledky nemají definované uspořádání (resp. jejich pořadí je určeno implementací vyhodnocení dotazu)

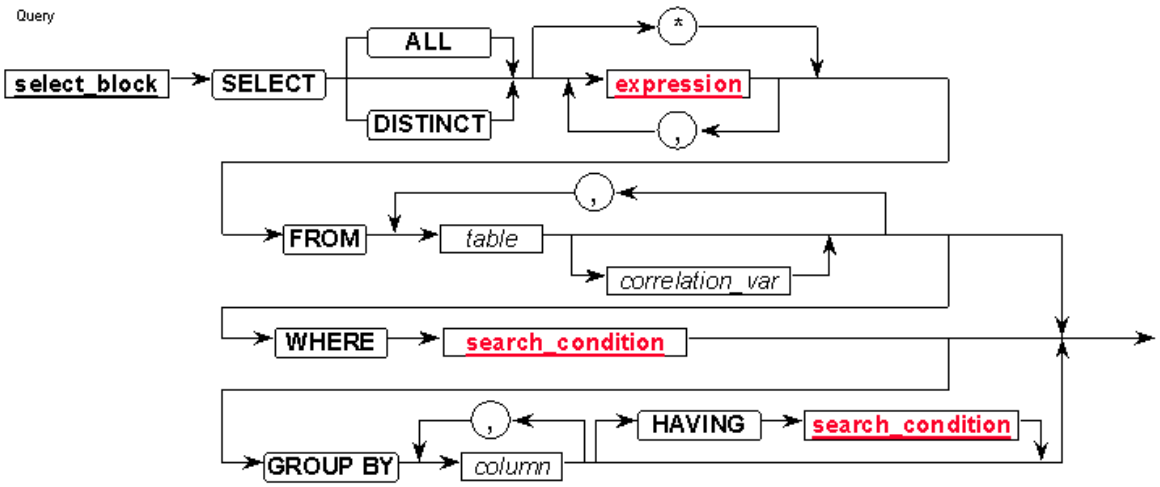


TŘÍDĚNÝ DOTAZ - výsledek netříděného dotazu lze setřídit
- klauzule **ORDER BY** - třídění podle sloupce (*column*)
- třídit lze podle definovaného uspořádání - vzestupně = **ASC**
  - sestupně = **DESC**
- lze definovat více sekundárních třídících kritérií, která se uplatní v případě nedefinovaného lokálního pořadí (shodné primární tříděné hodnoty)

SCHÉMA PŘÍKAZU SELECT
Příkaz **SELECT** se skládá z
- 2 až 5 klauzulí
- + případně ještě z klauzule **ORDER BY** (ta není specifická pouze pro **SELECT**)

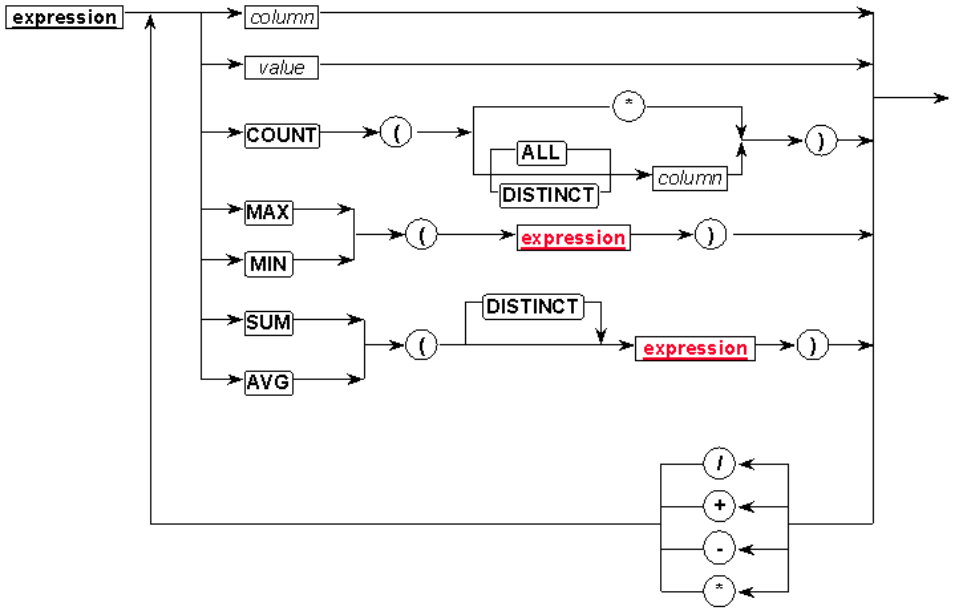
klauzule **SELECT** = projekce výstupního schématu, případně definice nových odvozených a agregačních sloupců
klauzule **FROM** = na které tabulky (v případě SQL ≥ 99 i vnořené dotazy, pohledy) se dotazujeme
klauzule **WHERE** = podmínka, kterou musí záznam (řádek) splňovat, aby se dostal do výsledku (logicky patří ke klauzuli **FROM**)
klauzule **GROUP BY** = přes které atributy se má výsledek popsaný pouze předchozími klauzulemi agregovat
klauzule **HAVING** = podmínka, kterou musí agregovaný záznam splňovat, aby se dostal do výsledku (patří ke klauzuli **GROUP BY**)

Logické pořadí vyhodnocení (resp. asociativita SELECT klauzulí):
FROM →**WHERE** →**GROUP BY** →**HAVING** →projekce **SELECT** (→**ORDER BY**)



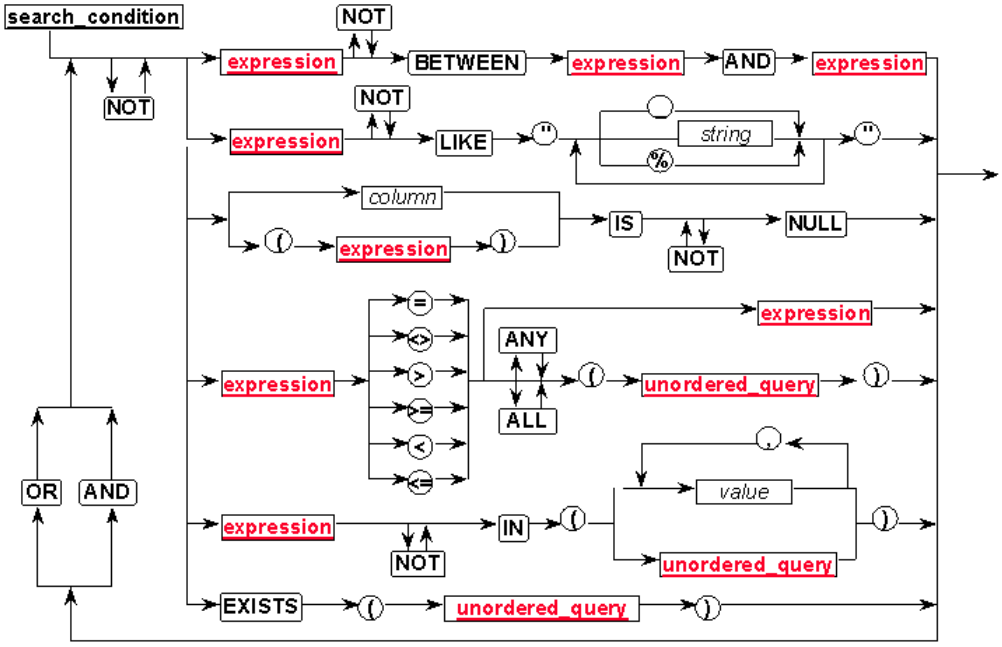
SELECT - EXPRESSION

Kontext - SELECT ALL | DISTINCT **expression** FROM ...
mnohokrát uvnitř **search_condition**



SELECT – SEARCH CONDITION

Kontext - SELECT ... FROM ... WHERE **search_condition**
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING **search_condition**



Tabulky použité v příkladech

Tabulka LETY

Let	Společnost	Destinace	Počet cestujících
OK251	CSA	New York	276
LH438	Lufthansa	Stuttgart	68
OK012	CSA	Milano	37
OK321	CSA	London	156
AC906	Air Canada	Torronto	116
KL7621	KLM	Rotterdam	75
KL1245	KLM	Amsterdam	130

Tabulka LETADLA

Letadlo	Společnost	Kapacita
Boeing 717	CSA	106
Airbus A380	KLM	555
Airbus A350	KLM	253

SELECT … FROM …
- nejjednodušší forma dotazu

SELECT [ALL] | DISTINCT expression **FROM** table1, table2, ...

- výraz může obsahovat
  - sloupce (hvězdička * je zástupce pro všechny neuvedené sloupce)
  - konstanty
  - agregace na výrazech
    - pokud se vyskytne alespoň jedna agregace, lze ve výrazu použít samostatně pouze agregované sloupce (ty uvedené v klauzuli **GROUP BY**), ostatní sloupce lze pouze „zabalit"do agregačních funkcí
    - pokud není definována klauzule **GROUP BY**, seskupuje se do jediné skupiny (celý zdroj určený klauzulí FROM se agreguje do jediného řádku) –tj. odpovídá agregaci podle prázdné množiny

  - **DISTINCT** = eliminuje duplikátní řádky ve výstupu
  - **ALL** (resp. bez specifikace) = povoluje ve výstupu i duplikátní řádky

  (pozor, má vliv na agregační funkce – u **DISTINCT** vstupuje do agregačních funkcí méně hodnot)
- **FROM** - obsahuje jednu nebo více tabulek, na kterých se dotaz provádí
  - pokud je specifikováno více tabulek, provede se kartézský součin

---

PŘ

Které společnosti přepravují cestující?
SELECT DISTINCT 'Spol.:', Společnost **FROM** Lety

Jaké páry letadel mohu vytvořit (bez ohledu na vlastníka) a jaká bude celková kapacita párů:
SELECT L1.Letadlo, L2.Letadlo, L1.Kapacita + L2.Kapacita **FROM** Letadla **AS L1**, Letadla **AS L2**

Kolik společností přepravuje cestující?
SELECT COUNT(**DISTINCT** Společnost) **FROM** Lety

Kolika lety se přepravují cestující?
SELECT COUNT(Společnost) **FROM** Lety
SELECT COUNT(*) **FROM** Lety

Kolik je letadel, jakou mají maximální, minimální, průměrnou a celkovou kapacitu?
SELECT COUNT(*), **MAX**(Kapacita), **MIN**(Kapacita), **AVG**(Kapacita), **SUM**(Kapacita) **FROM** Letadla

---

SELECT … FROM … WHERE

= logická podmínka selekce, tj. řádek tabulky (nebo kart. součinu, případně joinu), který podmínku splňuje, se dostane do výsledku
- jednoduché podmínky lze kombinovat logickými spojkami **AND**, **OR**, **NOT**
- lze se ptát
  - srovnávacím predikátem (=, <>, <, >, <=, >=) na hodnoty dvou atributů
  - na interval expr1 **[NOT] BETWEEN** (expr2 **AND** expr3)
  - řetězcovým predikátem **[NOT] LIKE** "*maska*", kde maska je řetězec obsahující speciální znaky **%** (reprezentující libovolný podřetězec) a _ (reprezentující libovolný znak)
  - testem na nedefinovanou hodnotu (**expr1) IS [NOT] NULL**
  - predikátem příslušnosti do množiny **expr1 [NOT] IN** (**unordered_query**)
  - jednoduchým existenčním kvantifikátorem **EXISTS** (**unordered_query**) testující prázdnost
  - rozšířenými kvantifikátory
    - existenčním **expr1** = | <> | < | > | <= | >= **ANY**(**unordered_query**)
      (tj. platí, že **alespoň jeden** prvek/řádek z **unordered_query** splňuje daný srovnávací predikát aplikovaný na **expr1**)
    - všeobecným **expr1** = | <> | < | > | <= | >= **ALL**(**unordered_query**)
      (tj. platí, že **všechny** prvky/řádky z **unordered_query** splňují daný srovnávací predikát aplikovaný na **expr1**)

PŘ

Kterými lety cestuje více než 100 cestujících?
SELECT Let, Počet_cestujících **FROM** Lety **WHERE** Počet_cestujících > 100

Kterými letadly by mohli letět cestující dané společnosti, pokud chceme mít naplněnost letadla alespoň 30%?

SELECT Let, Letadlo, (100 * Lety.Počet_cestujících / Letadla.Kapacita) **AS** Naplnenost **FROM** Lety, Letadla
WHERE Lety.Společnost = Letadla.Společnost **AND** Lety.Počet_cestujících <= Letadla.Kapacita
AND Naplnenost >= 30

Do kterých destinací se dá letět Airbusem nějaké společnosti (bez ohledu na naplněnost)?
SELECT DISTINCT Destinace **FROM** Lety **WHERE** Lety.Společnost **IN** (**SELECT** Společnost **FROM** Letadla
WHERE Letadlo **LIKE** "Airbus%")
nebo např.
SELECT DISTINCT Destinace **FROM** Lety, Letadla **WHERE** Lety.Společnost = Letadla.Společnost **AND**
Letadlo **LIKE** "Airbus%"

Cestující kterých letů se vejdou do libovolného letadla (bez ohledu na vlastníka letadla)?
SELECT * **FROM** Lety **WHERE** Počet_cestujících <= **ALL** (**SELECT** Kapacita **FROM** Letadla)

SPOJENÍ VŠEHO DRUHU
- Kartézský součin = „každý s každým"
- Přirozené spojení = spojení prvků relace přes stejné hodnoty **všech atributů** sdílených mezi A a B
- Vnitřní spojení = zobecnění přirozeného spojení, spojuje se přes danou podmínku
- Levé/pravé vnitřní spojení = spojení omezené na levou/pravou stranu (ve spojení nás zajímají pouze atributy A/B)
- Plné/levé/pravé vnější spojení = využívá doplnění nulových hodnot k těm prvkům, které nebylo možno „normálně" spojit

Jak lze spojovat bez speciálních konstrukcí (SQL 86):
- (vnitřní) spojení na podmínku a přirozené spojení lze realizovat jako omezený kartézský součin, tj.
SELECT... **FROM** table1, table2 **WHERE** table1.A = table2.B
- levé/pravé polospodní se upřesní v klauzuli **SELECT**, tj. projekcí

Nové konstrukce SQL 92:
- **kartézský součin** = **SELECT**... **FROM** table1 **CROSS JOIN** table2 ...
- **přirozené spojení** = **SELECT**... **FROM** table1 **NATURAL JOIN** table2 **WHERE**...
- **vnitřní spojení** = **SELECT**... **FROM** table1 **INNER JOIN** table2 **ON search_condition WHERE**...
- **sjednocení spojení** = vrací řádky první tabulky doplněné NULL hodnotami v atributech druhé tabulky + řádky druhé tabulky doplněné o NULL hodnoty v atributech první tabulky
SELECT... **FROM** table1 **UNION JOIN** table2 **WHERE**...
- **levé, pravé, plavnější spojení** =
SELECT... **FROM** table1 **LEFT**| **RIGHT**| **FULL OUTER JOIN** table2 **ONsearch_condition**... **WHERE**...

PŘ
Vrať dvojice let-letadlo uspořádané vzestupně podle volného místa v letadle po obsazení cestujícími (uvažujeme pouze ta letadla, kam se cestující z letu vejdou)?
SELECT Let, Letadlo, (Kapacita – Počet_cestujících) **AS** Volnych_mist **FROM** Lety **INNER JOIN** Letadla **ON** (Lety.Společnost = Letadla.Společnost **AND** Počet_cestujících <= Kapacita) **ORDER BY** Volnych_mist

Které lety nemohou být uskutečněny (protože společnost nevlastní vhodné/žádné letadlo)?
SELECT Let, Destinace **FROM** Lety **LEFT OUTER JOIN** Letadla **ON** (Lety.Společnost = Letadla.Společnost **AND** Počet_cestujících <= Kapacita) **WHERE** Letadla.Společnost **IS NULL**

SELECT /…/ GROUP BY … HAVING
- agregační klauzule, díky které se řádky dosavadní „tabulky" výsledku dotazu (tj. po fázi **FROM** a **WHERE**) poslučují podle totožných hodnot v definovaných sloupcích do skupin „super-řádků"
- výstupem je potom tabulka „super-řádků", kde slučující sloupce mají definované hodnoty původních řádků, ze kterých vznikly (protože byly pro všechny řádky v super-řádku stejné), kdežto v ostatních sloupcích by hodnota super-řádku byla nejednoznačná (různé hodnoty v původních řádcích), takže pro tyto řádky jsou dvě možnosti
  - buď se ve výsledku dotazu vůbec nebudou nevyskytovat
  - anebo se jim jedinečná hodnota vyrobí nějakou agregací z hodnot původních
- zobecnění použití agregačních funkcí uvedených dříve (**COUNT**, **MAX**, **MIN**, **AVG**, **SUM**), kde výsledkem není jednořádková tabulka (jako v případě nepoužití klauzule **GROUP BY**), ale tabulka s tolika řádky, kolik je super-řádků vzniklých po fázi **GROUP BY**
- z této „super-řádkové"tabulky lze pomocí klauzule **HAVING** odfiltrovat nezajímavé řádky (zde tedy nezajímavé super-řádky), podobně jako se pomocí **WHERE** filtrovaly výsledky pocházející z

„FROM-fáze“. POZOR, lze používat pouze agregované hodnoty sloupců

PŘ

Jakou (kladnou) přepravní kapacitu mají jednotlivé společnosti?

SELECT Společnost, SUM(Kapacita) FROM Letadla GROUP BY Společnost

Kterým společnostem se vejdou najednou všichni cestující do letadel (bez ohledu na destinaci, tj. v jednom letadle mohou být cestující více letů)?

SELECT Společnost, **SUM**(Počet_cestujících) **FROM** Lety **GROUP BY** Společnost **HAVING**
SUM(Počet_cestujících) <= (**SELECT SUM**(Kapacita) **FROM** Letadla **WHERE** Lety.Společnost = Letadla.Společnost)

VNOŘENÉ DOTAZY

- standard SQL92 (full) rozšiřuje možnost použití vnořených dotazů také v klauzuli **FROM**
- zatímco v SQL 86 bylo dovoleno jejich užití pouze v predikátech **ANY**, **ALL**, **IN**, **EXISTS**
- dva druhy použití
  - **SELECT** ... **FROM** (**unordered_query**) **AS** q1 **WHERE** ...
    - umožňuje výběr přímo z výsledku jiného dotazu (místo tabulky)
    - poddotaz je pojmenován q1 a s tímto identifikátorem se pracuje v dalších klauzulích jako s identifikátorem tabulky
  - **SELECT** ... **FROM** ((**unordered_query**) **AS** q1 **CROSS** | **NATURAL** | **INNER** | **OUTER** | **LEFT** | **RIGHT JOIN** (**unordered_query**) **AS** q2 **ON** (**expression**)
    - použití ve spojení všeho druhu

PŘ

Pro společnosti vlastnící letadla vrať součet všech cestujících a kapacit letadel.

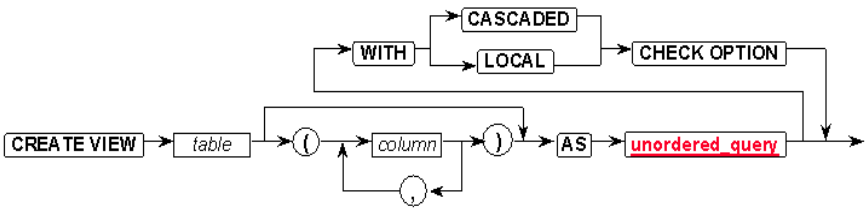
SELECT Lety.Společnost, **SUM**(Lety.Počet_cestujících), **MIN**(Q1. CelkováKapacita)
FROM Lety, (**SELECT SUM**(Kapacita) **AS** CelkováKapacita, Společnost **FROM** Letadla **GROUP BY** Společnost) **AS** Q1
WHERE Q1.Společnost = Lety.Společnost
GROUP BY Lety.Společnost;

Jaké jsou trojice různých letů, v rámci nichž se počty cestujících liší max. o 50?

SELECT Lety1.Let, Lety1.Počet_cestujících, Lety2.Let, Lety2.Počet_cestujících, Lety3.Let, Lety3.Počet_cestujících
FROM Lety **AS** Lety1
INNER JOIN (Lety **AS** Lety2 **INNER JOIN** Lety **AS** Lety3 **ON** Lety2.Let < Lety3.Let) **ON** Lety1.Let < Lety2.Let
WHERE abs(Lety1.Počet_cestujících – Lety2.Počet_cestujících) <=50 **AND**
abs(Lety2.Počet_cestujících – Lety3.Počet_cestujících) <=50 **AND**
abs(Lety1.Počet_cestujících – Lety3.Počet_cestujících) <=50
ORDER BY Lety1.Let, Lety2.Let, Lety3.Let;

POHLEDY = pojmenovaný dotaz, který lze dále použít jako tabulku

- generuje se dynamicky, podle dat vypočtených v okamžiku použití
  lze do něj - vkládat
  - mazat data
- CHECK OPTION zajistí, že po vložení/modifikaci záznamu do pohledu bude tato změna „vidět“
  - alespoň v tomto pohledu
  - ve všech závislých pohledech



PŘ

CREATE VIEW NovéVýrobkyNaSkladě **AS**
SELECT * **FROM** Výrobky **WHERE** JeNaSkladě = TRUE **AND CAST**(DatumVýroby **AS** VARCHAR(32)) > ‘2003-01-01’ **WITH LOCAL CHECK OPTION**

INSERT INTO NovéVýrobkyNaSkladě **VALUES** (0, ‘Hrábě’, 135, ‘2004-05-06’, TRUE, 4, 3215)
- vloží se (tranzitivně do tabulky Výrobky)

INSERT INTO NovéVýrobkyNaSkladě **VALUES** (0, ‘Lopata’, 135, ‘1999-11-07’, TRUE, 4, 3215)
!! Chyba – takovýto záznam se nemůže vložit, protože by nebyl „vidět“ v pohledu (moc stará lopata)