Null hodnoty v SQL

1. Úvod

Null neboli **NULL** je speciální označení skutečnosti, že hodnota *neexistuje* popř. *není známá*, též jako *unknown* value (pozor zde na slovo value)**.**
Nikoliv to však neznamená označování prázdné hodnoty (třeba u řetězců), nebo hodnoty 0 při uvažování číselné domény (typu), avšak se k **Null**u někdy můžeme chovat jako k hodnotě 0, uvidíme později.
=> **Null** chápeme tedy jako stav, nikoliv hodnotu.

Důvod existence **Null**u je ke splnění požadavku RDMS (true relational database management systems) musí umět reprezentovat nějakým způsobem **chybějící informaci/nepoužitelnou informaci**.

Null má jeden syntax, avšak může mít více sémantik, zde jsou dvě nejběžnější vysvětleny na příkladech:

Příklad (chybějící hodnota): „Kolik knih vlastní Adam?“ Na tuto otázku může být odpověď 0 (víme, že žádné knihy nevlastní), nebo NULL (ještě nevíme kolik knih vlastní, informace je chybějící a může být doplněna později).

Příklad (nepoužitelná hodnota): Představte si stávající databázi o automobilech, která najednou potřebovala přidat podporu pro elektrická auta, tato databáze má nějaký atribut *miles-per-gallon*, což v případě elektrického auta je nepoužitelná hodnota, tedy NULL zde udává stav nepoužitelné hodnoty.

Pouze pro úplnost, neboť všichni známe deklaraci v SQL:

```
CREATE TABLE <name> (
        <some_atr> <type> NOT NULL,
        <other_atr> <type>
);
```

V případě atributu *some_atr* vyžadujeme explicitní hodnotu, zatímco u *other_atr* nevyžadujeme a informace může být i chybějící či nepoužitelná. Při vytváření nového záznamu (n-tice), kde hodnota není explicitně specifikována v případě *other_atr* bude naplněna stavem NULL označující chybějící/nepoužitelnou informaci, u *some_atr* by došlo samozřejmě k vyvolání chyby.

2. Propagace

Protože NULL není datová hodnota, ale je to stav označující chybějící hodnotu, tak používání matematických operátorů na NULL nám dává neznámý výsledek, který je tedy reprezentován NULLem.

```
10 * NULL      -- Použitím mat. op. je výsledek NULL
```

```
'Fish n' || NULL || 'Chips'  -- Zřetězení přes NULL je opět NULL
```

K zamyšlení, co udělá následující výraz?

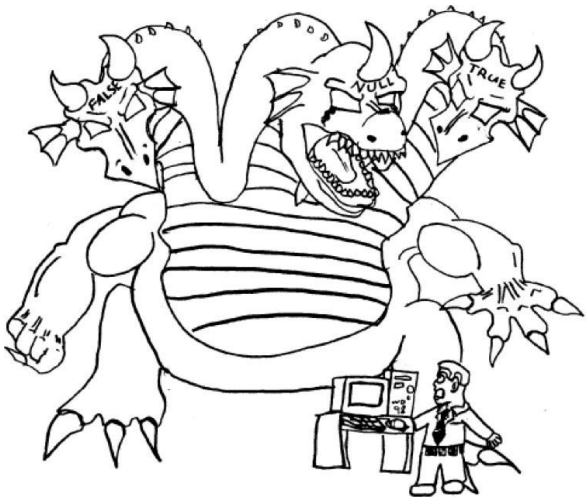
```
NULL / 0
```

Měl by vyvolat výjimku o pokus dělení nulou, že? Nemusí, většina platforem jako je MSSQL, MySQL, PostgreSQL a Oracle vrátí NULL. Toto chování není nikde definováno, jak se k tomu správně zachovat, je to tedy záležitostí platformy.

3. Three-valued logic (3VL)

NULL není prvkem žádné domény, tedy není brán jako hodnota, ale někdy též jako „placeholder“ pro nedefinovanou hodnotu. 3VL je postavena na tom, že ohodnocení e: VS -> {True, False, Unknown}, kde VS je množina výrokových symbolů.

p	NOT p
True	False
False	True
Unknown	Unknown



The monster that is three-valued logic has bitten more than one unwary SQL developer (Matt Williams)

4. Porovnávání

Porovnávání s NULLem nemůže vyústit ze tří stavů nic z **True** nebo **False**, ale **Unknown**.

```
SELECT 10 = NULL      -- Pravdivostní ohodnocení výrazu je Unknown
```

Celé chování pro představu je popsáno touto tabulkou, kde *p* a *q* jsou výrokové symboly, které mohou nabývat tří stavů {True, False, Unknown}.

P	q	p OR q	p AND q	p = q
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

Stejně jako instance nějaké třídy není ekvivalentní (díky adresy v paměti) s jinou instancí stejné třídy, stejně tak i NULL není ekvivalentní s jiným NULLem při jejich porovnávání. Až na pár výjimek, kde se k nim SQL chová, že nejsou od sebe distinktivní jsou operace a klauzule např. shlukování **GROUP BY** nebo **PARTITION BY**; **UNION**, **INTERSECT** a **EXCEPT**; **DISTINCT** v SELECT dotazech.

5. WHERE klauzule

3VL SQL se vyskytuje v jazyce DML (Data Manipulation Language) v porovnávacích predikátech příkazů a dotazů DML. Klauzule **WHERE** způsobí, že příkaz DML bude působit pouze na ty řádky, pro které je predikát vyhodnocen jako **True**. Pro řádky, pro které je predikát vyhodnocen jako **False** nebo **Unknown** nejsou příkazy **INSERT**, **UPDATE** nebo **DELETE** zpracovány, a jsou zahozeny pro **SELECT** dotazy.

```
SELECT *
FROM t
WHERE i = NULL;
```

Výše uvedený dotaz vždy vrátí nula řádků, protože porovnání sloupce *i* se sloupcem **Null** vždy vrátí hodnotu **Unknown**, a to i pro řádky, kde *i* je **Null**. Výsledek **Unknown** způsobí, že **SELECT** zahodí každý takový řádek.

6. 3VL specifické porovnávací predikáty

Základní porovnávací operátory SQL při porovnávání čehokoli s hodnotou **Null** vždy vrátí **Unknown**, takže standard SQL poskytuje dva speciální predikáty porovnání, specifické pro hodnotu **Null**. Predikáty **IS NULL** a **IS NOT NULL**, které testují, zda data jsou nebo nejsou **Null**. Takřka to samé pro predikáty **IS FALSE** a **IS NOT FALSE** (k nim i obdoba pro True), **IS UNKNOWN** a **IS NOT UNKNOWN**.

Predikáty mají následující pravdivostní tabulku:

p	p IS TRUE	p IS NOT TRUE	p IS FALSE	p IS NOT FALSE	p IS UNKNOWN	p IS NOT UNKNOWN
True	True	False	False	True	False	True
False	False	True	True	False	False	True
Unknown	False	True	False	True	True	False

7. Null v jiných konstrukcích

JOIN

Spojení se vyhodnocují pomocí stejných porovnávacích pravidel jako pro klauzule WHERE. Zatímco R ⋈ R = R platí pro každý výraz v relační algebře, spojení v SQL vyloučí všechny řádky, které mají kdekoli hodnotu Null.

CASE

SQL poskytuje dvě varianty podmíněných výrazů. Jeden se nazývá „simple CASE“ a funguje jako switch. Druhý se nazývá „searched CASE“ a funguje jako if-elseif.

a) Simple CASE výrazy používají implicitní porovnání, která se vyhodnocují pomocí stejných porovnávacích pravidel jako pro klauzule WHERE o kterých už jsme se už zmínili.

```
SELECT CASE i WHEN NULL THEN 'Is Null'  -- Toto nebude nikdy vráceno
              WHEN    0 THEN 'Is Zero'  -- Toto bude vráceno, pokud i = 0
              WHEN    1 THEN 'Is One'   -- Toto bude vráceno, pokud i = 1
              END
FROM t;
```

Protože i = NULL se vyhodnotí jako **Unknown**, tak řetězec ‚Is Null' nebude nikdy vrácen.

b) Searched CASE výrazy mohou ve svých podmínkách využít predikáty typu **IS NULL** a **IS NOT NULL**. Následující příklad ukazuje, jak využít „searched CASE" pro nalezení hodnoty NULL.

```
SELECT CASE WHEN i IS NULL THEN 'Null Result'-- Vráceno, pokud i = NULL
            WHEN     i = 0 THEN 'Zero'       -- Vráceno, pokud i = 0
            WHEN     i = 1 THEN 'One'        -- Vráceno, pokud i = 1
            END
FROM t;
```

Nyní už námi očekávaná funkcionalita, kterou jsme očekávali v query u simple case bude fungovat, všimněte si, že je tedy rozdíl na pořadí „SELECT CASE i WHEN" a „SELECT CASE WHEN i".

8. Funkce řešící problémy Null(u)

ISNULL – ISNULL(a, b) vrátí *b* pokud je *a* NULL jinak vrátí *a*; Transact-SQL konkrétně, ostatní platformy to nemají a používají již zmíněné predikáty

COALESCE – COALESCE(a, b, c) vrátí první prvek z leva, který není NULL; „pod pokličkou" se jedná o searched case

```
CASE WHEN value1 IS NOT NULL THEN value1
     WHEN value2 IS NOT NULL THEN value2
     WHEN value3 IS NOT NULL THEN value3
     ...
     END
```

NVL – NVL(a, b) vrátí zleva první non-NULL parametr, vrátí NULL pokud všechny jsou NULL; Oracle; NVL je zkratka za **N**ull **V**alue **L**ogic popř. **N**ull **V**a**L**ue

```
COALESCE (val1, ..., val{n})
-- Je ekvivalentní s
NVL (val1, NVL(val2, NVL(val3, …, NVL(val{n-1}, val{n}) … )))
```

CASE – vizte výše

9. Funkce (ne)pracující s Nullem

Agregační funkce jako jsou SUM(), AVG(), berou jako argument atribut z nějaké číselné domény. Pokud se v některém funkcí agregovaném sloupci objeví NULL stav, je ignorován a vypuštěn z výsledku (tzv. Null-elimination step). Výsledek agregačních funkcí může být také NULL (minimálně u MAX(), MIN()).

O něco jiné je chování COUNT(), představme si tabulku *Person* s atributy *id* a *age*, kde *id* je unikátní přirozené číslo a *age* může nabývat libovolného přirozeného čísla a připouští NULL. Celkový počet záznamů je 5, přičemž v jednom z nich *age* nabývá stavu NULL.

```
SELECT COUNT(*) FROM Person;
```

Vrátí všech 5 záznamů.

```
SELECT COUNT(age) FROM Person;
```

Vrátí pouze 4 záznamy. Povšimněme si faktu, že COUNT(*) je syntaktický cukr za COUNT(id, age) a funkce tak vrátila největší z čísel, které napočítala v každém sloupci zvlášť.

Pokud bychom chtěli náš druhý dotaz dostat ke stejnému výsledku jako ten první pouze přes sloupec *age*, mohlo by to vypadat nějak takto:

```
SELECT SUM(CASE WHEN age IS NULL THEN 1 ELSE 1) FROM Person;
```

Seznam obsažených článků:

https://en.wikipedia.org/wiki/Null_(SQL)

https://docs.bmc.com/docs/ars81/relational-algebra-and-qualifications-involving-null-values-225971553.html?fbclid=IwAR1VJofpj_mRWwemiVRCvzWk5Ci0Xb5wO4F4I7be8Id72br6woUgnH6U2hI

https://www.geeksforgeeks.org/sql-null-values/

https://towardsdatascience.com/how-to-deal-with-null-values-in-standard-sql-1bffce0c55cd

https://practice.geeksforgeeks.org/problems/explain-how-relational-algebra-operations-deal-with-null-values?fbclid=IwAR2rthlNkF16d6UHMP_bIzxHGJp2XhZ7XtqkNP3axViKUR0QaFEheBMfXEM

https://www.sqlshack.com/working-with-sql-null-values/

https://www.sqlshack.com/sql-partition-by-clause-overview/

https://www.oreilly.com/library/view/oracle-sqlplus-the/0596007469/ch04s03.html