9. Týždeň
Trojhodnotová logika.
Kvantifikátory a NOT. Množinové operácie.

1) Trojhodnotová logika
- **a) UNKNOWN**
- **b) NULL**

2) Alternatívny prístup k chýbajúcim údajom

3) Kvantifikátory ALL, ANY (SOME), EXISTS a NOT

4) Množinové operácie
- **a) UNION [ALL]**
- **b) intersect**
- **c) except**

5) Príklady

1) Trojhodnotová logika

a) Neznáme (UNKNOWN = NULL)

Vieme, že Coddova základná požiadavka na reláciu je:

každý riadok v tabuľke obsahuje hodnotu pre každý stĺpec.

Date: "Null (3HL) a relačný model sú vzájomne nekompatibilné", pretože:
- "Typ", ktorý obsahuje null, nie je typom (pretože typy obsahujú hodnoty).
- "n-tica", ktorá obsahuje null, nie je n-tica (pretože n-tice obsahujú hodnoty).
- "Relácia", ktorá obsahuje null, nie je reláciou (pretože relácie obsahujú n-tice a n-tice neobsahujú null).

Takže, ak sú prítomné null-y, nemôžeme hovoriť o *skutočnom* relačnom modeli.

V teórii relačných DB výsledok porovnania skutočných hodnôt s UNKNOWN vychádza z definície OR a AND:

- OR je TRUE ak aspoň jeden operand je TRUE
- AND je FALSE ak aspoň jeden operand je FALSE

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOVN	UNKNOWN

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

	NOT
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Zapamätajme si: TRUE OR UNKNOWN/null je TRUE
a FALSE AND UNKNOWN/null je FALSE.

Ukážky na null, true, false a is null:

```
select null;  -- NULL
select true;  -- 1
select false; -- 0

select null is null;     -- 1
select null is not null; -- 0
select null = null;      -- null

select true is null;     -- 0
select true is not null; -- 1

select false is null;     -- 0
select false is not null; -- 1
```

Kým `null` je hodnota a `is null` funkcia, `unknown` hodnota neexistuje iba `is unknown` (a `is not unknown`) funkcia, ktorá pre `null` je pravdivá :

```
select null is unknown;     -- 1
select null is not unknown; -- 0
```

Preto nasledujúce kódové riadky sú nedovolené:

```
-- select unknown;
-- select unknown is unknown;
-- select unknown is not unknown;
-- select unknown is null;
-- select unknown is not null;
```

Kódové riadky

```
select * from (select 1 Jed)t1 where null;
select * from (select 1 Jed)t1 where TRUE AND null;
select * from (select 1 Jed)t1 where FALSE AND null;
select * from (select 1 Jed)t1 where TRUE OR null;        # 1
select * from (select 1 Jed)t1 where FALSE OR null;

select * from (select 1 Jed)t1 where NOT null;
select * from (select 1 Jed)t1 where NOT(TRUE AND null);
select * from (select 1 Jed)t1 where NOT(FALSE AND null);# 1
select * from (select 1 Jed)t1 where NOT(TRUE OR null);
select * from (select 1 Jed)t1 where NOT(FALSE OR null);
```

vrátia prázdnu tabuľku, okrem dvoch prípadov, kedy predikáty sa vyhodnotia ako TRUE.

Riadky

```
SELECT 1=null;
SELECT NOT(1=null);
```

vrátia `null`. Ako systém vyhodnotí `1=null`, môžeme kontrolovať aj pomocou `where` klauzuly. Z nasledujúcich kódových riadkov,

```
select * from (select 3 x) t1 where not(1=null);
select * from (select 3 x) t1 where     1=null ;
```

ktoré vrátia prázdnu tabuľku a nie trojku, je zrejmé, že výraz `1=null` sa vyhodnotí ako `null`, lebo `true` apriori nemôže byť a keby bol `false`, potom by výsledkom dopytu bola trojka.

Podobne

```
select * from (select 1 Jed) t1 where null = null;         -- prazdna tabulka
select * from (select 1 Jed) t1 where not (null = null);   -- prazdna tabulka
```

ale

```
 select * from (select 1 Jed) t1 where null is null;       -- 1
```

Nasledujúce dva kódové riadky s rovnakým výsledkom, ilustrujú ešte raz najdôležitejší prípad operácie OR a AND trojhodnotovej logiky s NULL.

true or null je true:

```
select * from (select 3 x) t1 join (select 4 y) t2 on     1=1  or 1=null;
```

true

	x	y
1	3	4

false and null je false:

```
select * from (select 3 x) t1 join (select 4 y) t2 on not(1=2 and 1=null);
```

false

2) Alternatívny prístup k chýbajúcim údajom

C.J.Date a H.Darwen v prvom vydaní práce *Database Explorations: Essays on The Third Manifesto and Related Matters*, navrhujú rôzne prístupy k problému chýbajúcich údajov, ktoré sa vyhnú použitiu alebo zjavnej potrebe null v zmysle SQL. Popíšeme tu jeden z nich.

Uvažujme tabuľku o zmluvných dodávateľoch (v SQL farebné bunky obsahujú NULL)

Dodavatel

idD	Meno	Vernosť	Firma
1	Jano	20	APV
2	Fero	10	
3	Stevo		VEGA
4	Zoli		

Typy/príčiny chýbajúcich hodnôt

Dodávateľ je nový a ešte sme neodhadli stupeň vernosti

Dodávateľ nemá firmu, pracuje z domu

Dodávateľ neuviedol názov firmy

Riešenie: DB_dodavatel_2 so 6-mi tabuľkami bez NULL

DodMen

IdD	Meno
1	Jano
2	Fero
3	Stevo
4	Zoli

DodVer

idD	Vernosť
1	20
2	10

DodFir

idD	Firma
1	APV
3	VEGA

DodBezV

idD
3
4

DodBezNF

idD
4

DodBezF

idD
2

Dané riešenie
- nepoužíva null a žiadnú inú hodnotu na označenie chýbajúcich hodnôt
- namiesto trojhodnotovej logiky opiera sa o dvojhodnotovú logiku

Napíšme dopyt pre DB_dodavatel_2, ktorý je ekvivalentný

```
SELECT idD, Firma FROM Dodavatel
  WHERE Firma IS NOT NULL;
```

Riešenie

```
SELECT idD, Firma FROM DodFirma;
```

3) Kvantifikátory (operátory) ALL, ANY (SOME), EXISTS bez a s NOT

ALL, ANY (<=> SOME)

- syntax:

```
... where skal.výraz porovn.operátor ALL / ANY ( VD )
```

kde porovn.operátor je jeden z operátorov: = <> > >= < <=

Pripomíname, že

ALL
- ALL porovnáva skalárny výraz s **každou hodnotou** zoznamu alebo VD a vráti TRUE ak porovnanie platí pre **každú** dvojicu

ANY
- ANY tiež porovnáva skalárny výraz s **hodnotou** zoznamu alebo VD a vráti TRUE ak porovnanie platí **aspoň pre jednu** dvojicu

EXISTS
- vráti TRUE, ak VD obsahuje **aspoň jeden riadok**, ktorý môže obsahovať aj samé NULL hodnoty
- na rozdiel od ALL a ANY, EXISTS môžeme podobne ako IN negovať s NOT aj bez eliminácie NULL hodnôt.

WHERE klauzula v NOT EXISTS je splnená, ak poddopyt nevráti **žiadny** riadok.

Negácia IN a EXISTS pomocou NOT a nerovná sa <>

[NOT] EXISTS (VD)

= ANY ⇔ IN [~ EXIST]

<> ALL ⇔ NOT IN [~ NOT EXIST ak NOT NULL]

Poznamenáme, že ALL, NOT IN ... a NOT EXISTS ... sú drahé operácie, lebo za nimi nasledujúci poddopyt musí tabuľku / zoznam **úplne preskenovať**, teda každý riadok sa musí skontrolovať, kým operácie =ANY, IN ... a EXISTS ... je možné ukončiť skôr po kontrole toho riadku, pre ktorý podmienka je splnená.

Porovnanie [NOT] IN a [NOT] EXISTS

```
USE DBmaz;
DROP TABLE IF EXISTS t1;
DROP TABLE IF EXISTS t2;
CREATE TABLE t1 (id1 INT);
CREATE TABLE t2 (id2 INT);
INSERT t1 VALUES (1),(2),(3),(null);
INSERT t2 VALUES (1),    (3),(null);
```

t1

id1
1
2
3
NULL

t2

id2
1
3
NULL

Kým dopyty ANY, IN a EXISTS (so zodpovedajúcim predikátom) vracajú rovnaké výsledky (dokonca aj JOIN s IS NOT NULL): = ANY ⇔ IN [~ EXIST]

(pamätáme sa, ako sa vyhodnotia výrazy: `1=null` alebo `null=null` ?)

```
SELECT t1.* FROM t1 WHERE t1.id1 = ANY (SELECT t2.id2 FROM t2);
SELECT t1.* FROM t1 WHERE t1.id1 IN (SELECT t2.id2 FROM t2);
SELECT t1.* FROM t1 WHERE    EXISTS (SELECT t2.*   FROM t2 WHERE t1.id1 = t2.id2);
SELECT t2.* FROM t2 LEFT JOIN t1 ON t1.id1 = t2.id2 WHERE t2.id2 IS NOT NULL;
```

id2
1
3

dopyty NOT IN a NOT EXISTS už nie:

```
SELECT t1.* FROM t1 WHERE t1.id1 <> ALL (SELECT t2.id2 FROM t2); -- nič, prázdna tabuľka;
SELECT t1.* FROM t1 WHERE t1.id1 NOT IN (SELECT t2.id2 FROM t2); -- nič
SELECT t1.* FROM t1 WHERE    NOT EXISTS (SELECT t2.* FROM t2 WHERE t1.id1 = t2.id2); -- 2,null
SELECT t1.* FROM t1 WHERE t1.id1 != ANY (SELECT t2.id2 FROM t2); -- 1 / 2 / 3
```

Ale po odstránení **NULL** hodnôt, znova vrácajú rovnaké výsledky:

```
SELECT t1.* FROM t1 WHERE t1.id1 <> ALL (SELECT t2.id2 FROM t2 WHERE t2.id2 IS NOT NULL); -- 2
SELECT t1.* FROM t1 WHERE t1.id1 NOT IN (SELECT t2.id2 FROM t2 WHERE t2.id2 IS NOT NULL); -- 2
SELECT t1.* FROM t1 WHERE    NOT EXISTS (SELECT t2.*   FROM t2 WHERE t1.id1 = t2.id2) AND
                                                                    t1.id1 IS NOT NULL; -- 2
```

Porovnanie dvojice (n-tice) vo WHERE klauzule

```
DROP TABLE t1;
DROP TABLE t2;
CREATE TABLE t1 (id1 INT, x int);
CREATE TABLE t2 (id2 INT, y int);
INSERT t1 VALUES (1, 5),(2, 6),(1, 7),(null, 8);
INSERT t2 VALUES (1, 5),       (1, 6),(null, 7);
```

t1

id1	x
1	5
2	6
1	7
NULL	8

t2

id2	y
1	5
1	6
NULL	7

```
SELECT t1.* FROM t1 JOIN t2 ON t1.id1 = t2.id2 AND t1.x = t2.y;
-- ⇔
SELECT t1.* FROM t1 WHERE (t1.id1, t1.x) = ANY (SELECT t2.id2, t2.y FROM t2); # MySQL; not SS
```

id1	x
1	5

4) Množinové operácie

Množinové operácie

`UNION, INTERSECT, EXCEPT` **(rozdiel)**

v SQL spájajú dva alebo viac dopytov s kompatibilnými výsledkami. Operácie `INTERSECT, EXCEPT` sa v MySQL objavili dosť neskoro, boli implementované až v roku 2023. Na dosiahnutie posledných dvoch operácií ukážeme rôzne techniky.

Výhodou množinových operácií je zjednodušenie, resp. sprehľadnenie zložitejších dopytov. Ich **nevýhodou** môže byť menej optimálny kód, beh ktorého trvá dlhšie, veď štandardne riadky tabuľky treba preskenovať, prejsť dvakrát, raz pre každý Select operand, a v prípade veľkých tabuliek to môže znamenať časovú stratu.

a) UNION [ALL]

SELECT ... UNION | INTERSECT | EXCEPT SELECT ...

UNION a UNION ALL operátory umožňujú spojiť viac výsledkov (dopytov) do jedného. Na rozdiel od JOIN, ktorý predovšetkým používame na vertikálne spojenie stĺpcov (a pochopiteľne aj riadkov), UNION sa používa na horizontálne spojenie riadkov, pritom:

- počty stĺpcov musia byť rovnaké
- dátové typy zodpovedajúcich stĺpcov mali by byť kompatibilné alebo pretypovateľné (string !)

```
select 1, 'a'
union
select now(), now();
```

1	a
1	a
2016-11-03 10:57:31	2016-11-03 10:57:31

`UNION ALL` na rozdiel od UNION vráti aj duplicitné riadky.

```
USE DBmAZ;
drop table if exists T1;
drop table if exists T2;
CREATE TABLE T1 (i INT, x CHAR);
CREATE TABLE T2 (j INT, y ChAR);

INSERT INTO T1 VALUES
(1,'a'), (2,'b'), (10,'x');
INSERT INTO T2 VALUES
(10,'x'),(20,'y'),(30,NULL);

SELECT * FROM T1;
SELECT * FROM T2;

### 1) UNION
SELECT * FROM T1 UNION ALL SELECT * FROM T2; # 6 riad.
SELECT * FROM T1 UNION     SELECT * FROM T2; # 5 riad.
```

i	x
1	a
2	b
10	x

j	y
10	x
20	y
30	NULL

i	x
1	a
2	b
10	x
10	x
20	y
30	NULL

i	x
1	a
2	b
10	x
20	y
30	NULL

b) intersect

Operátor **intersect** porovnáva výsledky viac `SELECT` príkazov a vráti DISTINCT hodnoty. V MySQL intersect môžeme dosiahnúť jednoducho pomocou `INNER JOIN`-u.

```
SELECT * FROM T1
INTERSECT
SELECT * FROM T2;

-- ⇔
SELECT i, x FROM T1 JOIN T2 ON T2.j = T1.i AND T2.y = T1.x;

-- ⇔
SELECT i, x FROM T1 JOIN T2 ON (i,x) = (j,y); # MySQL
-- ⇔
SELECT i, x FROM T1, T2 WHERE (T2.j, T2.y) = (T1.i, T1.x); # MySQL

-- ⇔ Atypické riešenie pomocou IN:
SELECT i, x FROM T1
   WHERE i IN ( SELECT j FROM T2 WHERE T2.j = T1.i AND T2.y = T1.x );
```

i	x
10	x

c) except

Operátor **except** porovnáva výsledky viac `SELECT` príkazov a vráti DISTINCT hodnoty. Na rozdiel od **intersect** nie je symetrická operácia – výsledok záleží na poradí operandov. Ukážeme štyri riešenia v MySQL aj pomocou

- NOT IN
- <>ALL
- NOT EXISTS
- OUTER JOIN ... IS NULL

a) Rozdiel T1/T2:

```
SELECT * FROM T1
EXCEPT
SELECT * FROM T2;

-- ⇔
```

Negujeme atypické riešenie s IN:

```
SELECT * FROM T1
   WHERE i NOT IN ( SELECT j FROM T2 WHERE T2.j = T1.i AND T2.y = T1.x );

-- ⇔
SELECT * FROM T1
   WHERE i <> ALL ( SELECT j FROM T2 WHERE T2.j = T1.i AND T2.y = T1.x );

-- ⇔
SELECT * FROM T1
   WHERE NOT EXISTS ( SELECT * FROM T2 WHERE T2.j = T1.i AND T2.y = T1.x );
```

i	x
1	a
2	b

Pri LEFT OUTER JOIN vo vrátenej časti T2 tabuľky NULL hodnotu obsahujú riadky, ktoré sú odlišné od T1 a práve tie sú potrebné, tie poskytujú tretie riešenie.

```
-- ⇔ BEST namiesto EXCEPT: T1\T2:
SELECT i, x FROM T1 LEFT OUTER JOIN T2
                    ON (i, x) = (j, y)
WHERE T2.j IS NULL OR T2.y IS NULL;
```

i	x	j	y
10	x	10	x
1	a	NULL	NULL
2	b	NULL	NULL

SELECT * FROM T1 LEFT OUTER JOIN T2 ON i = j

b) Rozdiel T2/T1:

```
SELECT * FROM T2 EXCEPT SELECT * FROM T1;
-- ⇔
SELECT * FROM T2
WHERE NOT EXISTS (SELECT * FROM T1 WHERE T2.j = T1.i AND T2.y = T1.x);

-- ⇔
SELECT * FROM T2
WHERE j NOT IN (SELECT i FROM T1 WHERE T2.j = T1.i AND T2.y = T1.x);

-- ⇔
SELECT j,y FROM T2 LEFT OUTER JOIN T1
#SELECT j, y FROM T2 LEFT OUTER JOIN T1
                    # ON i = j <=> ON x = y
                     ON (i, x) = (j, y)
WHERE T1.i is NULL;
```

j	y
20	y
30	NULL

5) Príklady

Zistime id a mená pacientov, ktorí navštívili všetkých lekárov.

```
A - výrok, A' - negácia A; P=pacient, L=lekár;
      A:  všetky P, ktorí navštívili všetkých L
    ~ A:  neexistuje P, ktorý nenavštívil všetkých L
A': existuje P, existuje L: P nenavštívil L

A = (A')' = Neexistuje P, neexistuje L: P nenavštívil L
            ( pre pacienta NIE je lekár, koho NEnavštívil )
```

1) Prvé riešenie pomocou Except

```
USE Poliklinika;

-- OK SQL Server 1. riesenie
SELECT P.idP, P.krstne FROM Pacienti P
WHERE NOT EXISTS(    -- neexistuje L, koho P nenavstivil
             SELECT L.idL FROM Lekari L
             Except  -- zoznam lekarov, ktorych pacient nenavstivil
             SELECT N.idL FROM Navstevy N
                    WHERE N.idP = P.idP
              );
```

	idP	krstne
1	6	Tana

2) Druhé riešenie bez Except pomocou korelovaného dopytu

Najprv testujeme vnorený dopyt korelovaného dopytu, kde p.IDp nahradíme s 3.

```
-- MySQL, SQL Server
-- Ciel - taky idP, pre ktory obdrzime prazdnu tabulku, teda
--        taky pacient, pre koho NIE je lekár, koho Nenavštívil
```

```
SELECT * FROM Lekari L         -- 1) vsetci lekari
       LEFT OUTER JOIN(        -- 3) vsetci lekari s OUTER JOIN
             SELECT N.idL, 3 jaj FROM Navstevy N
             WHERE N.idP = 3   -- 2) L, koho idp = 3 navstivil
                   ) AS n2 ON N2.idL=L.idL
WHERE N2.idL IS NULL;   -- 4) lekari, koho idp = 3 NEnavstivil
```

	idL	krstne	spec	datNar	idL	jaj
1	2	Zoli	Zubny	1961-11-14 00:00:00.000	NULL	NULL
2	4	Zuzka	Zubny	1970-04-02 00:00:00.000	NULL	NULL
3	5	Imro	Interny	1956-11-09 00:00:00.000	NULL	NULL

Keby sme namiesto konkrétnej hodnoty 3 písali dvakrát 6(Tana), obdržali by sme hľadanú prázdnu tabuľku. Všeobecný prípad dvakrát p.IDp neprešiel v starších verziách My SQL (2021).

```
--- OK MySQL 2022, SQL Server 2. riesenie
--- NO MySQL < 2022

SELECT P.idP, P.krstne FROM Pacienti P -- 5) vsetci pacienti
WHERE NOT EXISTS(                      -- 6) neexistuje L, koho P nenavstivil<=>P navstivil vsetkych lekarov
       -- Ciel - taky idP, pre ktory obdrzime prazdnu tabulku:
          SELECT * FROM Lekari L         -- 1) vsetci lekari
                 LEFT OUTER JOIN(        -- 3) vsetci lekari s OUTER JOIN
                   SELECT N.idL, p.IDp FROM Navstevy N
                   WHERE N.idP = p.IDp -- 2) L, koho idp = 3 navstivil
                         ) AS n2 ON N2.idL=L.idL
           WHERE N2.idL IS NULL          -- 4) lekari, koho idp = 3 NEnavstivil
          );
```

Kým v 2020 pre testovanie kódu na MS SQL Server online kompilátor rextester bol ešte úplne dostupný, v 2021 už nie
https://rextester.com/l/sql_server_online_compiler
http://sqlfiddle.com/

V 2023 https://onecompiler.com/

O týždeň nasleduje
Úložiska dát a Data science