Db2

Row Permission – Beispiel
Column Mask – Umgehung

© Walter E. Huth, 2024

Inhalt

- **Row Permisssion**
  - **Einführung**
  - **Beispiel [*]**
    – mit diversen, “wilden” Restriktionen des Zugriffs
    – mithilfe einer Globalen Variablen
  - **Zusätze**

- **Column Mask**
  - **Einführung**
  - **Beispiele, [*]**
    – wie man sie umgehen kann
  - **Zusätze**

[*] Realisierung/Tests auf Db2 für Linux bzw. Db2 für Windows – sollte(n) auch auf Db2 für z/OS funktionieren

© Walter E. Huth, 2024

Row Permission – Einführung

Definition:
- „Row Permission“ ist ein Datenbank-Objekt (d.h. erstellt via CREATE), das eine **Regel für den Zugriff auf** Tabellen-**Reihen** beschreibt. („Regel“ kann man als Erlaubnis oder als Restriktion interpretieren.)
- Die Regel wird spezifiziert mittels einer SQL **WHERE**-Bedingung, gemäß der ein Benutzer/eine Gruppe/eine Rolle auf eine Reihe zugreifen kann.

Beispiel:
```
CREATE PERMISSION perm-1 ON tab-1
FOR ROWS WHERE current_sqlid IN (...) OR col-1 LIKE 'A%'
ENFORCED FOR ALL ACCESS ENABLE ;
```

Zusätze:
- Zusätzlich erforderlich:
  `ALTER TABLE tab-1 ACTIVATE ROW ACCESS CONTROL ;`
- Obige CREATE und ALTER erfordern SECADM-Authority.
- Die WHERE-Bedingung w1 der Permission arbeitet wie ein Filter auf der Tabelle, „bevor“ die vom Benutzer spezifizierte WHERE-Bedingung w2 prozessiert wird. (Die WHERE-Bedingungen w1 und w2 werden mit AND verknüpft.)
- Definiert man mehrere Row Permissions auf derselben Tabelle, so werden deren Bedingungen mit OR verknüpft.
- Es gibt built-in Funktionen, die dieses Konzept unterstützen, z.B.:
  `VERIFY_GROUP_FOR_USER ( USER, racf-group-1, racf-group-2,.. )`
- RCAC (row and column access control) gibt‘s seit DB2 V10 (z/OS, LUW).

© Walter E. Huth, 2024

Beispiel: Schutz einzelner Personen

Beispiel-App: Die Personen einer Familie und deren Vorfahren („Stammbaum“) sind in einer Db2-Tabelle PERSON gespeichert.

Über das Web sollen Familien-Angehörige („Gäste“) Zugriff erhalten.

Zunächst müssen sich die Gäste mit eMail-Adresse und pw anmelden, sodann greift stellvertretend „uid1“ auf die Tabelle PERSON zu.

geheim! Klar, uid1 hat SELECT-Berechtigung.

Aber **zusätzlich**:
Einige gespeicherte Personen sollen nur für einen Teil der Gäste sichtbar sein!

(„user requirements“)

© Walter E. Huth, 2024

Nicht jeder Gast soll alle Personen sehen können!

Gegeben: Tabellen PERSON [id, name, geb_datum, ...] und GAST [email]

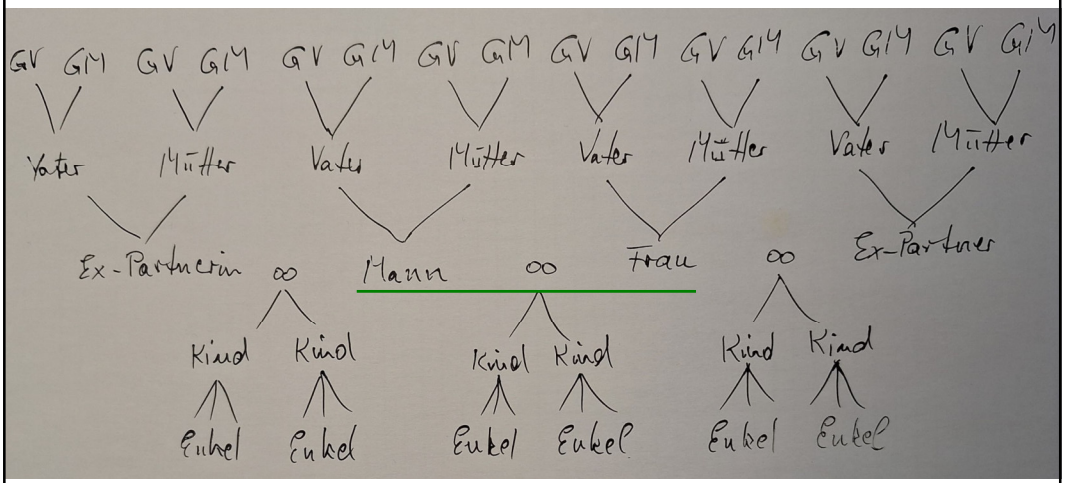
Restriktionen: Gäste sollen nicht alle gespeicherten Personen sehen können, z.B.:
- Ein Vetter des Mannes soll nur Verwandte des Mannes sehen können.
- Ein Vetter der Frau soll nur Verwandte der Frau sehen können.
- Aber: Beide Vettern sollen zusätzlich angezeigt bekommen:
  den Mann, die Frau, deren Geschwister und deren Nachkommen.
- Eines der unehelichen Kinder soll nur für bestimmte Gäste sichtbar sein.
- Jüdische Personen sollen nur für deren direkte Nachkommen sichtbar sein.
- . . . (Weitere Einschränkungen sollen einfach implementierbar sein.)

==> Pro Gast soll es möglich sein, eine „wilde“ Teilmenge von Personen freizuschalten bzw. zu sperren!

Lösungsidee: Pro Restriktion wird ein Schloss eingeführt, daraufhin:
Jede **Person** wird geschützt durch **n Schlösser (locks)**;
jeder **Gast** erhält **m Schlüssel (keys);**
Zugriff nur, wenn Gast für jedes Schloss einer Person einen Schlüssel besitzt.

© Walter E. Huth, 2024

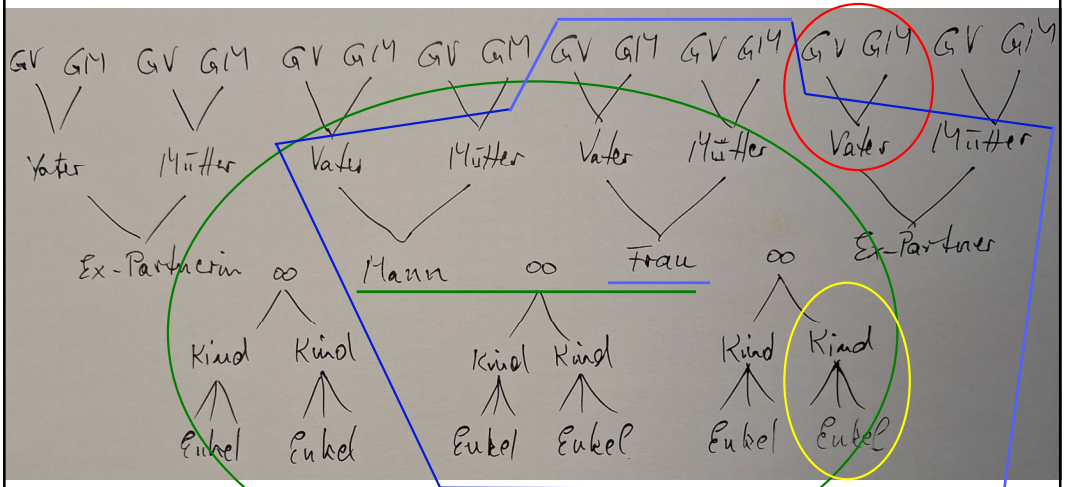
Veranschaulichung: Beispiel-Stammbaum



© Walter E. Huth, 2024

Beispiele möglicher Restriktionen

Restriktion-1 = Lock L1: Welche Personen im ersten Ansatz für alle Verwandten (Gäste) sichtbar sein sollen
Restriktion-2 = Lock L2: Wen Verwandte der Frau i.e.A. sehen sollen; analog L3: ..des Mannes..
Restriktion-4 = Lock L4: jüdisch gemäß NS-Definition – soll nur für direkte Nachkommen sichtbar sein
Restriktion-5 = Lock L5: uneheliches Kind u. dessen Nachkommen – soll nur für gewisse Gäste sichtbar sein



Beispiel:
Ein Vetter der Frau kann diese nur sehen, wenn er **Key L1** und **Key L2** besitzt,
Das uneheliche Kind und dessen Nachkommen kann er nur sehen, wenn er zusätzlich **Key L5** besitzt.

© Walter E. Huth, 2024

Benötigte Tabellen für Locks und Keys

PERSON

id	name	...
id1	Adam	...
id2	Berta	...
id3	Carl	...

PERSON_LOCKS

id	lock
id1	L1
id1	L2
id2	L1
id2	L3
id3	L3

GAST

email
gast1@web.de
gast2@gmx.de
gast3@gmail.com

GAST_KEYS

id	key
gast1@web.de	L1
gast1@web.de	L2
gast2@gmx.de	L3
gast2@gmx.de	L4
gast3@gmail.com	L1
gast3@gmail.com	L3

===> _ gast1 sieht nur id1
_ gast2 sieht nur id3
_ gast3 sieht id2 und id3

© Walter E. Huth, 2024

CREATE PERMISSION auf Tabelle PERSON

```
--   SECADM's activities:
--
--   Create global variable (prerequisite for - as used in - following permission):
CREATE VARIABLE v_email CHAR(35) ;   -- stores email address of a guest
--   Grant all on global variable to dbadmin/programmer's userid:
GRANT ALL ON VARIABLE v_email TO USER uid1 ;
--
--   Idea: A guest can access a row of a person, if no lock (of that person) is left
--                                               after reduction of all keys (of that guest).
CREATE PERMISSION person_perm ON person p
       FOR ROWS
       WHERE NOT EXISTS
             (
               SELECT pl.lock
                 FROM person_locks pl
                WHERE p.id = pl.id
               EXCEPT
               SELECT gk.key
                 FROM gast_keys gk
                WHERE gk.email = v_email
             )
             -- OR UPPER(USER) = 'DB2INST1' / 'SYSADM1'  -- for db2 administrator
       ENFORCED FOR ALL ACCESS ENABLE ;
--
--   Activate permission:
ALTER TABLE person ACTIVATE ROW ACCESS CONTROL ;
--
--   Remark: sequence required for DROP: 1. Permission, 2. Variable
```

© Walter E. Huth, 2024

Zugriff auf Tabelle PERSON

Wer greift zu?

```
--  Userid uid1 führt folg. Befehle aus:
```

Welcher Gast?

```
--  aktuellen Gast spezifizieren:
SET v_email = 'gast2@gmx.de' ;          [*]
```

Zugriff (Definition)

```
SELECT id, name, geb_datum, ...
  FROM person
 WHERE bedingung-1 ;
```

Restriktionen sind unsichtbar.

Zugriff (Realität)

```
-- Von Db2 ausgeführt wird:
SELECT id, name, geb_datum, ...
  FROM person
 WHERE bedingung-1
   AND where-bedingung-aus-row-permission-definition ; [#]
```

[*] Schema davor setzen, z.B.: .. DB2INST1.v_email .. oder .. SECADM1.v_email ..
[#] präziser: AND die OR-Kombinationen der WHERE-Bedingungen aller Row-Permission-Definitionen auf dieser Tabelle - inkl. der Bedingung 1=0 nach/aufgrund ALTER table ACTIVATE ROW ACCESS CONTROL

© Walter E. Huth, 2024

Verschiedene Gäste sehen verschied. Personen

```
--  Userid uid1:
--
SET v_email = 'gast2@gmx.de' ; [*]
--
--  Query:
--
SELECT id, name
  FROM person
 WHERE name LIKE '%a%' ;
--
--  Resultat:
--
ID  NAME
--  ----------
03  Carl
```

```
--  Userid uid1:
--
SET v_email = 'gast3@gmail.com'; [*]
--
--  Query:
--
SELECT id, name
  FROM person
 WHERE name LIKE '%a%'
--
--  Resultat:
--
ID  NAME
--  ----------
03  Carl
02  Berta
```

Resultat ist abhängig vom Inhalt der globalen Variablen, d.h. der eMail-Adresse des Gastes.

[*] Schema davor setzen, z.B.: .. DB2INST1.v_email .. oder .. SECADM1.v_email ..

© Walter E. Huth, 2024

Fazit, Bemerkungen

- **DER Vorteil: Wilde Kombination von Restriktionen wird unterstützt!**

- Füllen der Tabellen PERSON_LOCKS und GAST_KEYS
  - Bei diesem Beispiel gilt: Die allermeisten Einträge für PERSON_LOCKS kann man schnell/fehlerfrei mittels SQL INSERT ... SELECT ... bewerkstelligen. Für die wenigen Gäste sind die Einträge schnell manuell erledigt.
  - Bei anderen Anwendungen könnte es aufwendig werden.

- Schutz der einen Tabelle ausreichend!
  - Neben PERSON gibt‘s noch PAAR: speichert (Ehe-)Partner.
  - Wird nur via JOIN mit PERSON benutzt – ist automatisch mit geschützt

- Zugriffsberechtigungen für PERSON_LOCKS und GAST_KEYS: Restriktiv!

- Performance:
  - Jeder Zugriff erfolgt mit einer erweiterten WHERE-Bedingung.
  - Im vorgestellten Beispiel (~2000 Reihen in PERSON) nicht bemerkbar
  - Bei größeren Datenmengen: Indexe!

- RCAC (row and column access control)
  - überwacht ALLE (auch SYSADM‘s) SQL-Zugriffe, aber nicht Utilities.

© Walter E. Huth, 2024

Vergleich mit anderen Konstrukten

RCAC (row and column access control) im Vergleich zu

- **einer Lösung mit Views**
  Nach Manual gilt angeblich:
  - Je komplexer/feingliedriger die Restriktionen,
    desto schwieriger Definitionen und Wartung/Änderungen von Views
  - Anwendungen spezifizieren oft Views statt Tabellen.

- **LBAC (label-based access control)**
  - RCAC reine Db2 Lösung; LBAC erfordert RACF-Definitionen
  - LBAC für hierarchisch geordnete Gruppen
    (Generäle dürfen mehr als Gefreite.)
    => LBAC für Militär, Verwaltungen
    vs.
    RCAC für Firmen – und Stammbäume (;-))

© Walter E. Huth, 2024

Zusätze (aus einer Präsentation von G. Baklarz)

Beispiel-Anforderung:

The PAYROLL table can only be accessed through the HRPROCS.PROC1 stored procedure during the hours from 8:00 am to 5:00 pm

Lösung:

```
CREATE PERMISSION payroll-table-rules
ON PAYROLL

FOR ROWS WHERE CURRENT TIME BETWEEN '8:00' AND '17:00'
AND SYSIBM.ROUTINE_SPECIFIC_NAME = 'PROC1'
AND SYSIBM.ROUTINE_SCHEMA = 'HRPROCS'
AND SYSIBM.ROUTINE_TYPE = 'P'
ENFORCED FOR ALL ACCESS ENABLE;

ALTER TABLE payroll ACTIVATE ROW ACCESS CONTROL;
```

© Walter E. Huth, 2024

Zusätze (aus einer Präsentation von G. Baklarz)

teils LUW-spezifisch:

Using UDFs and Triggers with RCAC-Protected Tables

- UDFs must be defined as SECURED when referenced from within row and column access control definitions

```
ALTER FUNCTION ACCBALDISPLAY SECURED;
```

```
CREATE MASK EXAMPLEHMO.ACCT_BALANCE_MASK ON PATIENT FOR
COLUMN ACCT_BALANCE RETURN
CASE WHEN VERIFY_ROLE_FOR_USER(SESSION_USER,'ACCOUNTING') = 1
   THEN ACCBALDISPLAY(ACCT_BALANCE)
   ELSE 0.00
END
ENABLE;
```

- UDFs invoked on columns protected with a column mask must be defined as SECURED. For instance, the SELECT statement below will fail unless CALC is defined as a secure UDF

```
SELECT CALC(ACC_BALANCE) FROM PATIENT;
```

- Triggers must be defined as SECURED if the subject table is protected with row or column access control

```
ALTER TRIGGER T_LOG_CHANGES SECURED;
```

IBM Digital Technical Engagement IBM

© Walter E. Huth, 2024

Zusätze (aus einer Präsentation von G. Baklarz)

teils LUW-spezifisch:

Restrictions and Considerations

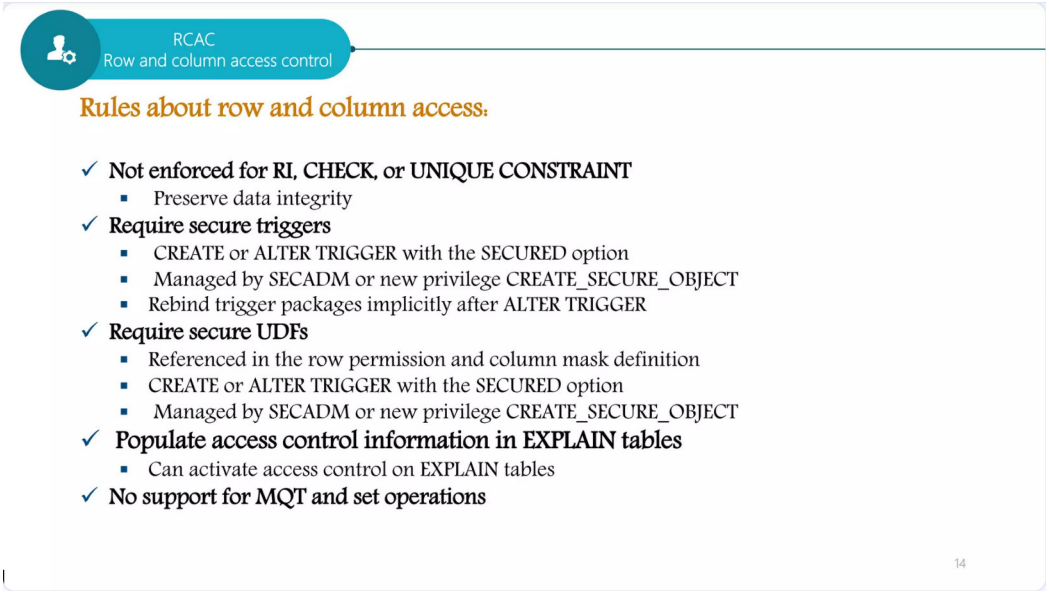
- You cannot create a mask on a column which
  - Is an XML object
  - Is a LOB column or a distinct type column that is based on a LOB
  - Is a column referenced in an expression that defines a generated column
- UDFs and TRIGGERs must be altered with SECURE keyword
  - Compromise between security vs. integrity
- Automatic Data Movement
  - No propagation of any existing row permissions or column masks takes place. Target table is automatically protected with the default no access row permission to protect against direct access to that target table
    - MQT, History Tables for Temporal tables, and detached table partitions for range-partitioned tables
- db2look can extract row permission and column mask definitions in order to mimic elsewhere
- EXPLAIN facility shows access plans with RCAC in place
  - Can override with NORCAC option

IBM Digital Technical Engagement IBM

© Walter E. Huth, 2024

Zusätze (aus Präsentation von N. Poorkamali)

teils LUW-spezifisch:



© Walter E. Huth, 2024

Inhalt

- **Row Permisssion**
  - **Einführung**
  - **Beispiel [*]**
    - **mit diversen, “wilden” Restriktionen des Zugriffs**
    - **mithilfe einer Globalen Variablen**
  - **Zusätze**
- **Column Mask**
  - **Einführung**
  - **Beispiele, [*]**
    - **wie man sie umgehen kann**
  - **Zusätze**

[*] Realisierung/Tests auf Db2 für Linux bzw. Db2 für Windows – sollte(n) auch auf Db2 für z/OS funktionieren

© Walter E. Huth, 2024

Column Mask – Einführung

Definition:
- „Column Mask“ ist ein Datenbank-Objekt (d.h. erstellt via CREATE), das eine **Regel** (i.a. Restriktion) **für den Zugriff auf Werte einer** Tabellen-**Spalte** beschreibt.
- Die Regel wird spezifiziert mittels einer SQL **CASE**-Expression, die beschreibt, unter welchen Bedingungen welche Spalten-Werte Benutzer/Gruppen/Rollen sehen dürfen.

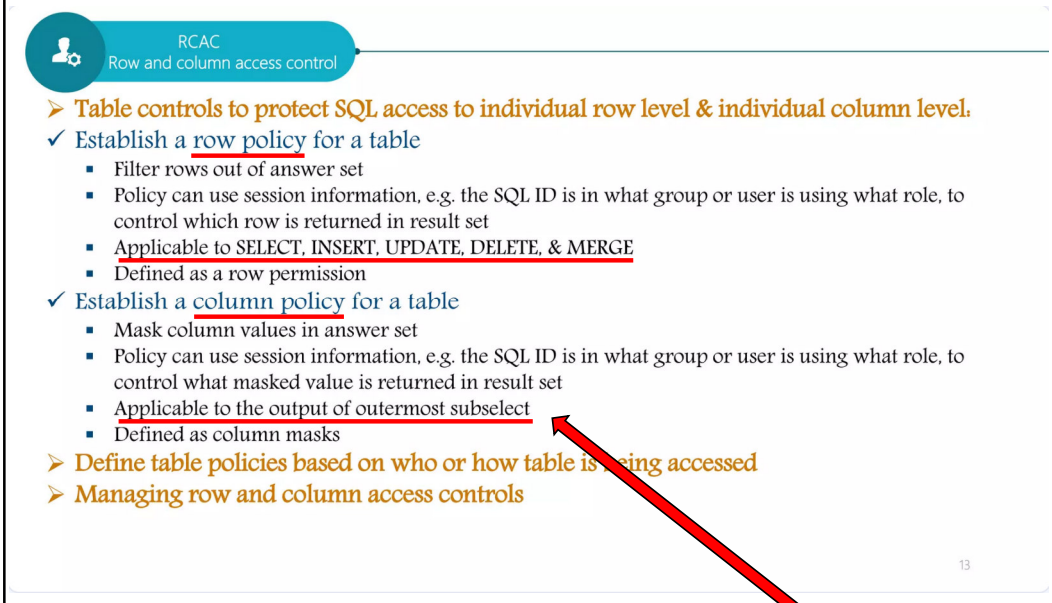
Beispiel:

```
CREATE MASK mask-1 ON tab-1 FOR COLUMN col-1 RETURN
CASE WHEN current_sqlid IN (...) OR col-1 LIKE 'A%'
     THEN col-1
     ELSE null
END
ENABLE ;
```

Zusätze:
- Zusätzlich erforderlich:
  `ALTER TABLE tab-1 ACTIVATE COLUMN ACCESS CONTROL ;`
- Obige CREATE und ALTER erfordern SECADM-Authority.
- Es gibt built-in Funktionen, die dieses Konzept unterstützen, z.B.:
  `VERIFY_GROUP_FOR_USER ( USER, racf-group-1, racf-group-2,.. )`
- RCAC (row and column access control) gibt‘s seit DB2 V10 (z/OS, LUW).

© Walter E. Huth, 2024

Row Permission vs. Column Mask



Quelle: Präsentation von N. Poorkamali; rote Hervorhebungen von W.Huth

Gefahr !

© Walter E. Huth, 2024

Column Mask umgehen – 1. Beispiel für Zahlen

Unterdrücke die Ausgabe von Boni über 500.00 !

```
CREATE MASK bonus_mask ON emp FOR COLUMN bonus RETURN
    CASE WHEN (bonus > 500.00) THEN NULL ELSE bonus END
    ENABLE;
ALTER TABLE emp ACTIVATE COLUMN ACCESS CONTROL ;
```

Eine geschickte Abfrage [*]
kann diese Boni dennoch anzeigen.

```
LASTNAME    BONUS   UNMASKED_BONUS
----------  ------  --------------
...
YAMAMOTO    500.00             500
YOSHIMURA   500.00             500
BROWN         NULL             600
HENDERSON     NULL             600
...
LUCCHESI      NULL             900
HAAS          NULL            1000
HEMMINGER     NULL            1000
```

[*] s. „Unmasking the masked data“ by Emil Kotrc, https://www.idug.org/news/unmasking-the-masked-data

© Walter E. Huth, 2024

Column Mask umgehen – 2. Beisp. für Zahlen (1)

```
PERSON_M
-----------------------
ID NAME        SALARY
-- ---------- ---------
01 Hugo          1000,00
02 Emil Hans     1900,50
03 Hans-Peter    3500,40
04 Emma Lotta    4000,09
05 Lena          7900,00
```

Unterdrücke die Ausgabe von Gehältern über 2000.00 !

```
CREATE MASK person_salary
   ON person_m FOR COLUMN salary RETURN
       CASE WHEN (salary > 2000)
            THEN NULL
            ELSE salary
       END
   ENABLE ;
ALTER TABLE person_m
      ACTIVATE COLUMN ACCESS CONTROL ;
```

```
SELECT * FROM person_m
==>
ID NAME        SALARY
-- ---------- ---------
01 Hugo          1000,00
02 Emil Hans     1900,50
03 Hans-Peter          -
04 Emma Lotta          -
05 Lena                -
```

© Walter E. Huth, 2024

Column Mask umgehen – 2. Beisp. für Zahlen (2)

Dennoch Gehalt abschätzen:

```
WITH intervalle ( x1, x2 ) AS
( SELECT    0,  999.99 FROM sysibm.sysdummy1
  UNION
  SELECT 1000, 1999.99 FROM sysibm.sysdummy1
  UNION
  SELECT 2000, 2999.99 FROM sysibm.sysdummy1
  .. etc. ..
  UNION
  SELECT 9000, 9999.99 FROM sysibm.sysdummy1
)
SELECT id, name, salary
      , x1 AS salary_von, x2 AS salary_bis
  FROM person_m, intervalle
 WHERE salary BETWEEN x1 AND x2
```

Variation der Idee von Emil Kotrc:
- ohne Rekursion
- mit Berücksichtigung von Nachkommastellen

nicht: „output of outermost subselect“

ID	NAME	SALARY	SALARY_VON	SALARY_BIS
01	Hugo	1000,00	1000	1999,99
02	Emil Hans	1900,50	1000	1999,99
03	Hans-Peter	-	3000	3999,99
04	Emma Lotta	-	4000	4999,99
05	Lena	-	7000	7999,99

© Walter E. Huth, 2024

Column Mask umgehen – Beispiel für Texte (1)

Spaltenwerte nicht anzeigen, wenn sie „e“ oder „E“ enthalten!

```
CREATE MASK person_name ON person_m FOR COLUMN name RETURN
       CASE WHEN (UPPER(name) LIKE '%E%') THEN NULL
            ELSE name
       END
       ENABLE ;
ALTER TABLE person_m ACTIVATE COLUMN ACCESS CONTROL ;
```

```
PERSON_M
--------------
ID NAME
-- -----------
01 Hugo
02 Emil Hans
03 Hans-Peter
04 Emma Lotta
05 Lena
```

```
SELECT id, name
FROM person_m

==>
ID NAME
-- ----------
01 Hugo
02 -
03 -
04 -
05 -
```

```
Query auf den folg. Seiten


==>
ID NAME        NAME_UNMASKED
-- ---------- -------------
01 Hugo        hugo
02 -           emil hans
03 -           hans-peter
04 -           emma lotta
05 -           lena
```

© Walter E. Huth, 2024

Column Mask umgehen – Beispiel für Texte (2)

Query zum Anzeigen aller Namen:

Simpel-Version, z.B. nur Kleinbuchstaben

```
WITH abc ( letter ) AS
( SELECT 'a' FROM sysibm.sysdummy1
  UNION
  SELECT 'b' FROM sysibm.sysdummy1
  .. etc. ..
  UNION
  SELECT 'z' FROM sysibm.sysdummy1
  UNION
  SELECT ' ' FROM sysibm.sysdummy1
  UNION
  SELECT '-' FROM sysibm.sysdummy1
)
, position ( nr ) AS
( SELECT 1 FROM sysibm.sysdummy1
  UNION ALL
  SELECT nr + 1 FROM position
    WHERE nr < (SELECT MAX(LENGTH(name))
                  FROM person_m  )
      AND nr < 2000                          [*]
)
    ..   Fortsetzung nächse Seite ..
```

```
ABC
------
LETTER
------
a
b
.. etc. ..
z
□              ←blank
-              ←minus
```

```
POSITION
--------
NR
--
1
2
3
.. etc. ..
n          = Spaltenlänge
```

[*] unterdrückt Warnung „infinite loop“

© Walter E. Huth, 2024

Column Mask umgehen – Beispiel für Texte (3)

Query zum Anzeigen aller Namen:

nicht: „output of outermost subselect“

```
        .. (Fortsetzung) ..

, name_vertikal (id, nr, letter) AS
(
SELECT id, nr, letter
  FROM person_m, position, abc
 WHERE letter =
       SUBSTR(LOWER(name),nr,1)
)
, solution (id, name_unmasked ) AS
(SELECT id
      , CAST (
        LISTAGG(letter, '' )
           WITHIN GROUP (ORDER BY nr)
        AS VARCHAR(10) )
        AS name_unmasked
  FROM name_vertikal
 GROUP BY id
)
SELECT s.id, name, name_unmasked
  FROM solution s, person_m p
 WHERE s.id = p.id
```

Resultat s. vorne

```
NAME_VERTIKAL
-------------
ID NR LETTER
-- -- ------
01  1 h
02  1 e
03  1 h
04  1 e
05  1 l
01  2 u
02  2 m
03  2 a
04  2 m
05  2 e
01  3 g
02  3 i
03  3 n
    3 m
05  3 n
01  4 o
02  4 l
03  4 s
04  4 a
05  4 a
01  5 □  ←blank
02  5 □  ←blank
03  5 -  ←minus
.. etc. ..
```

Kreuzprodukt von id und nr; danach kombiniert mit Buchstabe aus person_m.name

```
SOLUTION
--------
ID NAME_UNMASKED
-- -------------
01 hugo
02 emil hans
03 hans-peter
04 emma lotta
05 lena
```

© Walter E. Huth, 2024

Column Mask - Zusätze

im Katalog:

```
SELECT char(controlschema,6) AS schema
     , char(controlname,14)  AS name
     , controltype           AS type     --  LUW: R oder C; z/OS: R oder M
     , char(ruletext,60)     AS ruletext
  FROM syscat.controls                   --  z/OS: SYSIBM.SYSCONTROLS

SCHEMA NAME           TYPE RULETEXT
------ -------------- ---- ------------------------------------------------------------
xyz    PERSON_SALARY  C    CASE WHEN (SALARY > 2000) THEN NULL ELSE SALARY END
xyz    PERSON_NAME    C    CASE WHEN (UPPER(NAME) LIKE '%E%') THEN NULL ELSE NAME END
```

Ursache des Umgehungsproblems:

- **Column Mask betrifft nur den „output of outermost subselect“.**
- **Direkter Tabellen-Zugriff mit SQL (statt nur per Programm) erlaubt**

Flankierende Maßnahmen:

- **s. Artikel**
  „ Db2 for z/OS column mask … effective data security strategy“
  https://www.idug.org/news/db2-for-zos-column-mask-functionality-part-of-an-effective-data-security-strategy
  mit weiterem Link zu CIS (Center for Internet Security) -Dokument („CIS IBM Db2 13 for z/OS Benchmark“)

Db2 13 for z/OS **FL 506** hebt einige Restriktionen für Column Masks auf:
https://www.ibm.com/docs/en/db2-for-zos/13?topic=levels-function-level-506#db2z_fl_v13r1m506__e29163

© Walter E. Huth, 2024

Vielen Dank
für Ihre Aufmerksamkeit !

walter.huth (at) gmx.de
whuth.de

© Walter E. Huth, 2024