

Triggery

# Trigger

- **Trigger** je procedurální kód, který lze spouštět automaticky před/namísto/po každém provedení příkazu INSERT, DELETE nebo UPDATE na vybrané tabulce.
- Spouští se automaticky na základě některé z uvedených událostí, ručně se nespouští.
- Trigger je navázán na tabulku, při zrušení tabulky příkazem DROP je zrušen i trigger.
- Použití: Pokročilé kontroly integrity databáze (např. správnosti vkládaných dat – při kontrole lze uvažovat související data z více tabulek), automatické archivování starých dat, kopírování dat, logování do souboru nebo tabulky (audit, žurnál), plnění atributu primárního klíče číslem ze sekvence apd.

# Trigger – vytvoření

```
CREATE [OR REPLACE] TRIGGER <název triggeru>
{BEFORE|AFTER|INSTEAD OF} {INSERT|DELETE|UPDATE}
[OF <seznam atributů>] ON <název tabulky>
[FOR EACH ROW [WHEN <podmínka>]]
[DECLARE
  -- u triggeru se uvádí DECLARE (oproti proc./funkcím)]
BEGIN
  výkonná sekce
[EXCEPTION
  sekce pro zpracování výjimek]
END;
```

- Pokud neuvedeme **FOR EACH ROW**, trigger se vyvolá jen jednou pro danou událost; jinak se vyvolá vícekrát – pro každý řádek tabulky ovlivňovaný **INSERTem/UPDATEm/DELETEDem** a v kódu triggeru budou pak dostupné pseudoobjekty **:NEW** a **:OLD** (s původními a novými hodnotami v řádku).

# Trigger – zrušení

- Syntaxe:

**DROP TRIGGER** <název triggeru>;

- nebo automaticky při rušení tabulky:

**DROP TABLE** <název tabulky>;

# Typy triggerů

- Existují následující typy triggerů:
  - **DDL triggers:** Před, po nebo místo zavolání nějakého DDL příkazu (například GRANT, REVOKE, TRUNCATE apod.) se zavolá trigger.
  - **Statement-level triggers:** Takový trigger se zavolá při změně tabulky pomocí DML příkazu (INSERT / UPDATE / DELETE). Nemá ale informace o tom, co se bude přidávat / měnit v databázi ... hodí se pouze pro logování jaký uživatel kdy zavolal nějakou DML operaci.
  - **Row-level triggers:** Tyto triggery se používají pro auditování změn v nějaké tabulce, protože se volají pro každý zpracovaný řádek.
  - **Compound triggers:** Kombinace statement a row-level triggers.
  - **System-level triggers:** Umožňují auditování server startup / shutdown, user logon / logoff apod.

# Trigger FOR EACH ROW

- Pokud je trigger prováděn pro každý řádek ovlivňovaný INSERTem/UPDATEm/DELETEm (FOR EACH ROW), v těle triggeru jsou dostupné předdefinované pseudoobjekty:
  - **:NEW** reprezentující nový záznam s novými hodnotami v řádku (po provedení INSERT/UPDATE/DELETE),
  - **:OLD** reprezentující původní záznam s původními hodnotami (před provedením I/U/D).
- Nepovinná část **WHEN <podmínka>** umožňuje spouštět FOR EACH ROW trigger navíc pouze pro takové řádky, jejichž atributy vyhovují zadané podmínce.

# Trigger FOR EACH ROW – příklad

```
CREATE OR REPLACE TRIGGER TR_zaloha_zamestnancu
BEFORE DELETE ON zamestnanec FOR EACH ROW
BEGIN
    INSERT INTO byvaly_zamestnanec(id, jmeno,
    prijmeni, ukonceni_pomeru)
    VALUES (:OLD.id, :OLD.jmeno, :OLD.prijmeni, SYSDATE);
END;
```

# BEFORE/AFTER ... FOR EACH ROW

- Pokud se jedná o BEFORE trigger (trigger je volán ještě před provedením I/U/D), údaje v záznamu :NEW lze v triggeru měnit (modifikovat výsledné hodnoty, které budou v řádku po provedení I/U/D). To samozřejmě pro DELETE nemá smysl.
- V AFTER triggeru nemůžeme již modifikovat hodnoty v :NEW (změna se do tabulky nepromítne), ale máme zde každopádně dostupné hodnoty z :OLD.
- Vyvoláním výjimky ven z BEFORE triggeru lze zabránit provedení následného dotazu I/U/D (lze zabránit např. vložení řádku při dosažení určitého „maximálního povoleného“ počtu záznamů, za určitých podmínek zabránit smazání...). Pro AFTER trigger také funguje zabránění provedení I/U/D vyvoláním a neošetřením výjimky.



# Použití triggeru pro zajištění autoinkrementace I.

- Chceme, aby se číslo ve sloupci tabulky automaticky navyšovalo pro každý nově vložený řádek (typické pro celočíselný primární klíč), aniž bychom při INSERTu museli sami volat sekvenci (<jméno sekvence>.NEXTVAL).
- Některé databázové systémy mají možnost označit takový sloupec jako AUTOINCREMENT (MySQL), Oracle nikoliv.
- V Oracle použijeme sekvenci a trigger, který zajistí, že se při každém INSERTu použije další číslo ze sekvence – viz. následující.

# Použití triggeru pro zajištění autoinkrementace II.


- Nadefinujeme sekvenci:

```
CREATE SEQUENCE AERO.zamestnanec_seq;
```

# Použití triggeru pro zajištění autoinkrementace II.

- Nadefinujeme trigger automaticky spouštěný před INSERTem pro každý řádek – trigger doplní do nově vkládaného řádku číslo ze sekvence:

```
CREATE OR REPLACE TRIGGER tr_zamestnanec_seq  
  BEFORE INSERT ON zamestnanec FOR EACH ROW  
BEGIN  
  if :new.zamestnanec_id is null then  
    :new.zamestnanec_id := zamestnanec_seq.nextval;  
  end if;  
END;
```



Poznámka: dříve bylo nutné použít tento zápis:

```
SELECT zamestnanec_seq.NEXTVAL  
INTO :NEW.zamestnanec_id FROM dual;
```

# Použití triggeru pro zajištění autoinkrementace IV.

- Volitelně můžeme také chtít zabránit pokusu o UPDATE hodnoty získané ze sekvence:

```
CREATE OR REPLACE TRIGGER  
AERO.tr_zamestnanec_seq_upd AFTER UPDATE OF  
zamestnanec_id ON AERO.zamestnanec FOR EACH ROW  
BEGIN  
    RAISE_APPLICATION_ERROR(-20010, 'Cannot update  
column zamestnanec_id in table AERO.zamestnanec  
as it uses sequence.');
```

**END;**

# Trigger pro více událostí (I/U/D)

- Trigger může být spouštěn např. po každém I, U a D:

```
CREATE OR REPLACE TRIGGER TR_zaznam_historie
AFTER INSERT OR UPDATE OR DELETE ON moje_tabulka FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO historie (typ_akce, cas, uzivatel, ...)
    VALUES ('INSERT', SYSDATE, USER, :NEW.atr1, ...);
  ELSIF UPDATING THEN
    -- ulozeni puvodnich hodnot z :OLD
    INSERT INTO historie (typ_akce, cas, uzivatel, ...)
    VALUES ('UPDATE-OLD', SYSDATE, USER, :OLD.atr1, ...);
    -- ulozeni novych hodnot z :NEW
    INSERT INTO historie (typ_akce, cas, uzivatel, ...)
    VALUES ('UPDATE-NEW', SYSDATE, USER, :NEW.atr1, ...);
  ELSIF DELETING THEN
    INSERT INTO historie (typ_akce, cas, uzivatel, ...)
    VALUES ('DELETE', SYSDATE, USER, :OLD.atr1, ...);
  END IF;
END;
```

Aktuální datum a čas

Jméno DB uživatele

# Použití triggeru pro zajištění integrity dat

- Pomocí triggerů lze provádět nejrůznější kontroly pro zajištění správnosti dat v databázi – přehledně na jednom místě, doplňková kontrola ke kontrole v aplikaci nad DB:

```
/* Trigger kontrolující interval  
<datum porizení letadla, čas odletu> */  
CREATE OR REPLACE TRIGGER AERO.TR_let_int_porizeni_casodl  
BEFORE INSERT ON AERO.let FOR EACH ROW  
DECLARE  
    datum_porizeni_letadla DATE;  
BEGIN  
    SELECT datum_porizeni INTO datum_porizeni_letadla  
    FROM AERO.letadlo  
    WHERE letadlo_id = :NEW.letadlo_id;  
  
    IF datum_porizeni_letadla > :NEW.cas_odletu THEN  
        RAISE_APPLICATION_ERROR(-20103,  
            'Letadlo nemuze odletat v dobu, kdy jeste neni koupeno.');
```