Deklarativní IO – shrnutí minulé přednášky

- Existují následující typy omezení:
  - NOT NULL
  - UNIQUE Key
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK
- Pro zobrazení všech definic a názvů IO se použije dotaz na pohledy **data dictionary** (USER_CONSTRAINTS, USER_CONS_COLUMS, ...)

Program přednášky:

- ošetření IO pomocí triggerů
- procedurální ošetření IO
- bezpečnost dat
  pohledy
  uživatelská schémata a privilegia
  databázové role
- transakční zpracování

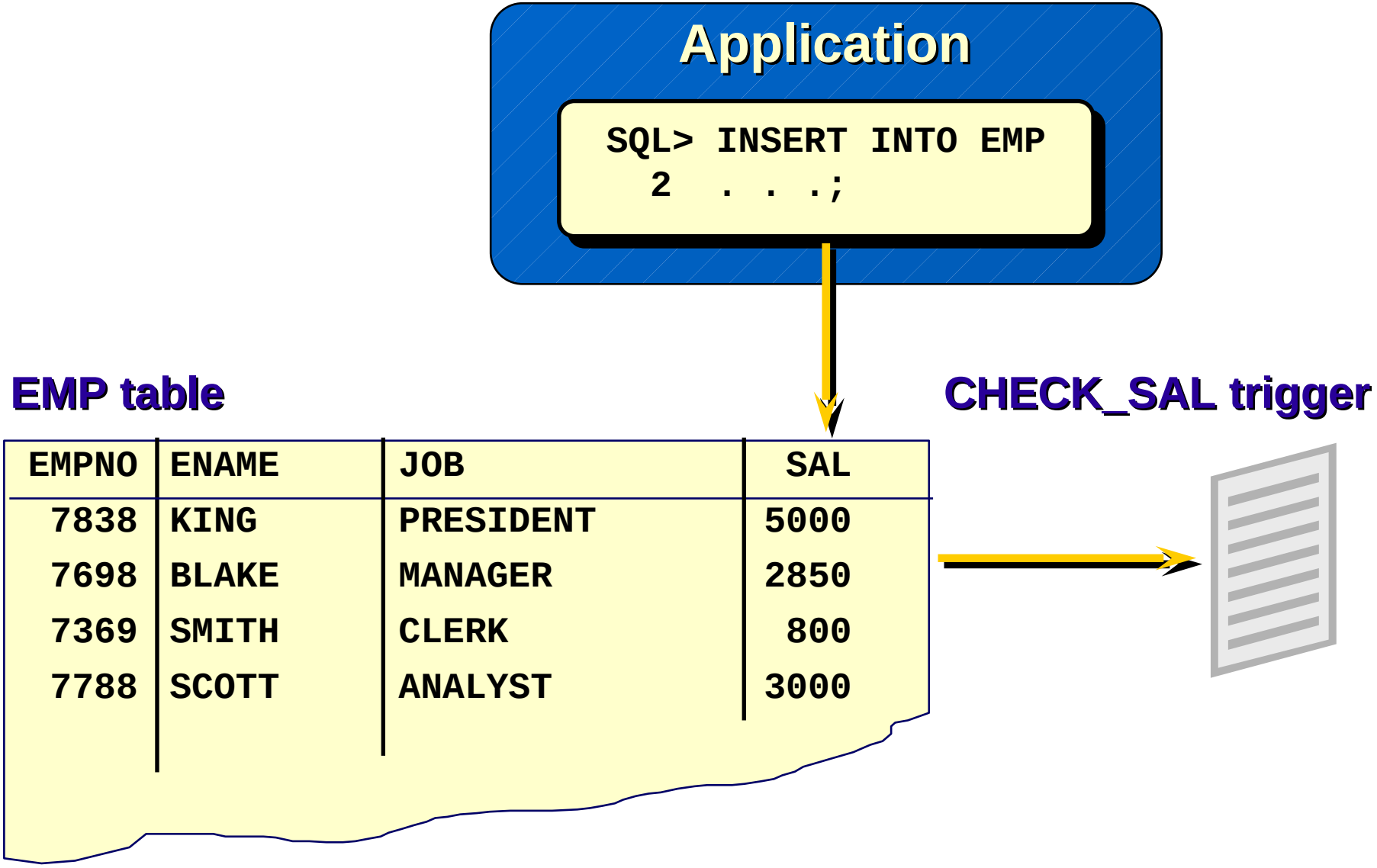
Procedurální IO – na straně serveru

- můžeme využít rozšíření funkcionality DB stroje o uložené **programové jednotky** a **triggery**
- příklady – Oracle, Postgresql
- procedury, funkce, balíky (packages), triggery
- složitější IO lze implementovat pomocí triggerů
- příklad – Oracle, jazyk PL/SQL

Triggery

- PL/SQLvázaný na konkrétní objekt (tabulku) a akci (insert, update, delete, ...)
- Triggery – databázové a aplikační

Databázový trigger - příklad



Triggery - vlastnosti

- časování
  - nad tabulkou: BEFORE, AFTER
  - nad pohledem: INSTEAD OF
- událost: INSERT, UPDATE, or DELETE
- jméno objektu
- typ triggeru: Row nebo statement
- omezující podmínka: When
- tělo triggeru tvoří PL/SQL blok

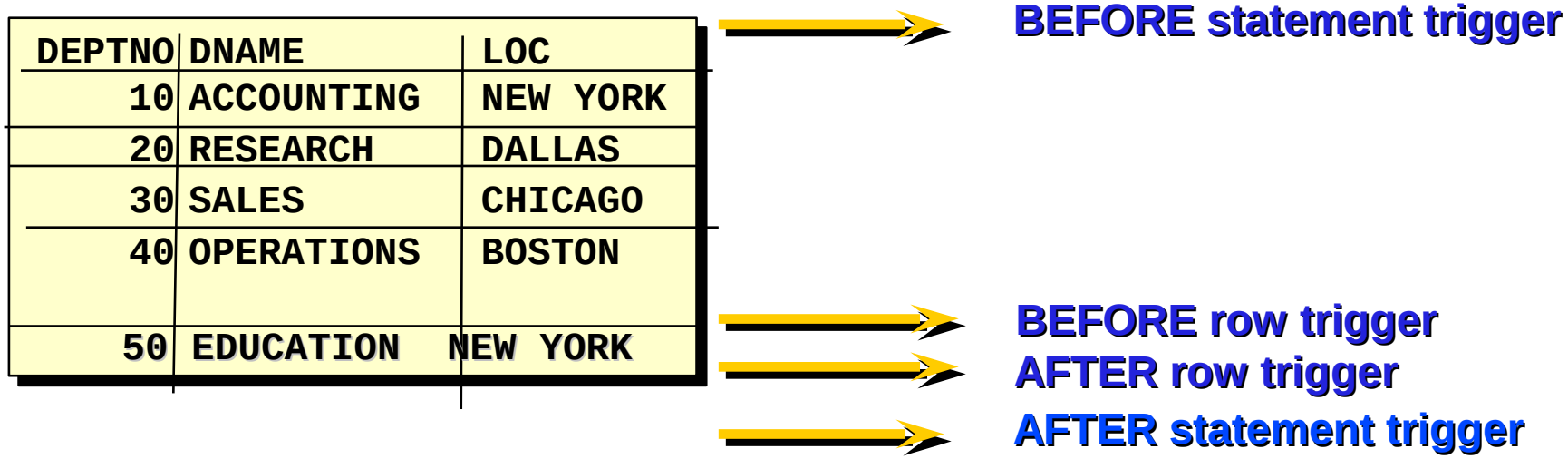
posloupnost spuštění triggerů

DML příkazj se týká pouze jednoh řádku.

DML Statement

```
SQL> INSERT INTO dept (deptno, dname, loc)
  2  VALUES (50, 'EDUCATION', 'NEW YORK');
```

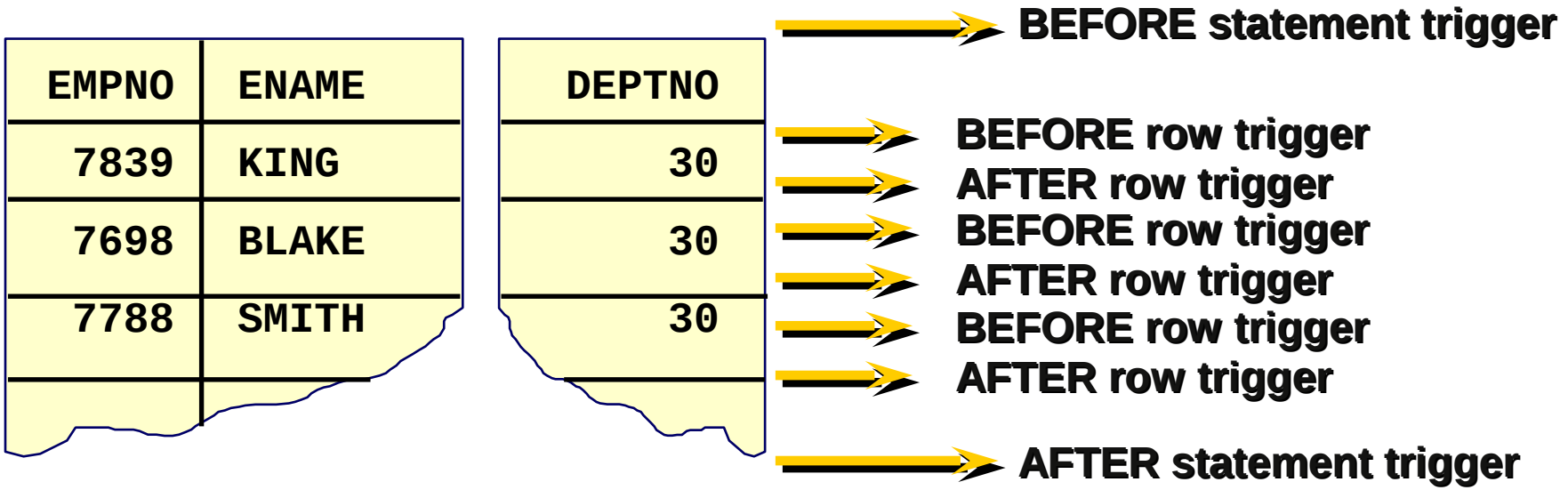
Triggering Action



Posloupnost spouštění triggerů

DML příkaz ovlivní více řádků.

```
SQL> UPDATE emp
  2  SET sal = sal * 1.1
  3  WHERE deptno = 30;
```



Trigger - syntaxe

```
CREATE [OR REPLACE] TRIGGER trigger_name
   timing
      event1 [OR event2 OR event3]
         ON table_name
trigger_body
```

Příklad triggeru

```
SQL> CREATE OR REPLACE TRIGGER secure_emp
  2  BEFORE INSERT ON emp
  3  BEGIN
  4   IF (TO_CHAR (sysdate,'DY') IN ('SAT','SUN')) OR
  5      (TO_CHAR(sysdate,'HH24') NOT BETWEEN
  6                               '08' AND '18')
  7   THEN RAISE_APPLICATION_ERROR (-20500,
  8   'You may only insert into EMP during business hours.');
  9   END IF;
 10  END;
 11  /
```

Chování triggeru

```
SQL> INSERT INTO emp (empno, ename, deptno)
  2  VALUES           (7777, 'BAUWENS', 40);
INSERT INTO emp (empno, ename, deptno)
            *
ERROR at line 1:
ORA-20500: You may only insert into EMP during normal hours.
ORA-06512: at "A_USER.SECURE_EMP", line 4
ORA-04088: error during execution of trigger
'A_USER.SECURE_EMP'
```

Procedurální IO na straně serveru – alternativa k použití triggerů

- založeno na procedurálním rozšíření
- kód je skryt v procedurách na serveru
- procedury hlídají (složitější) IO
- aplikace pracuje jako „front – end“
- DML je nahrazeno voláním procedur
- dotazy se realizují nad pohledy
- aplikace přístupových práv a rolí
- **Poznámka:** Existují i jiné alternativy architektury IS, které tuto problematiku řeší.

Procedura - syntaxe

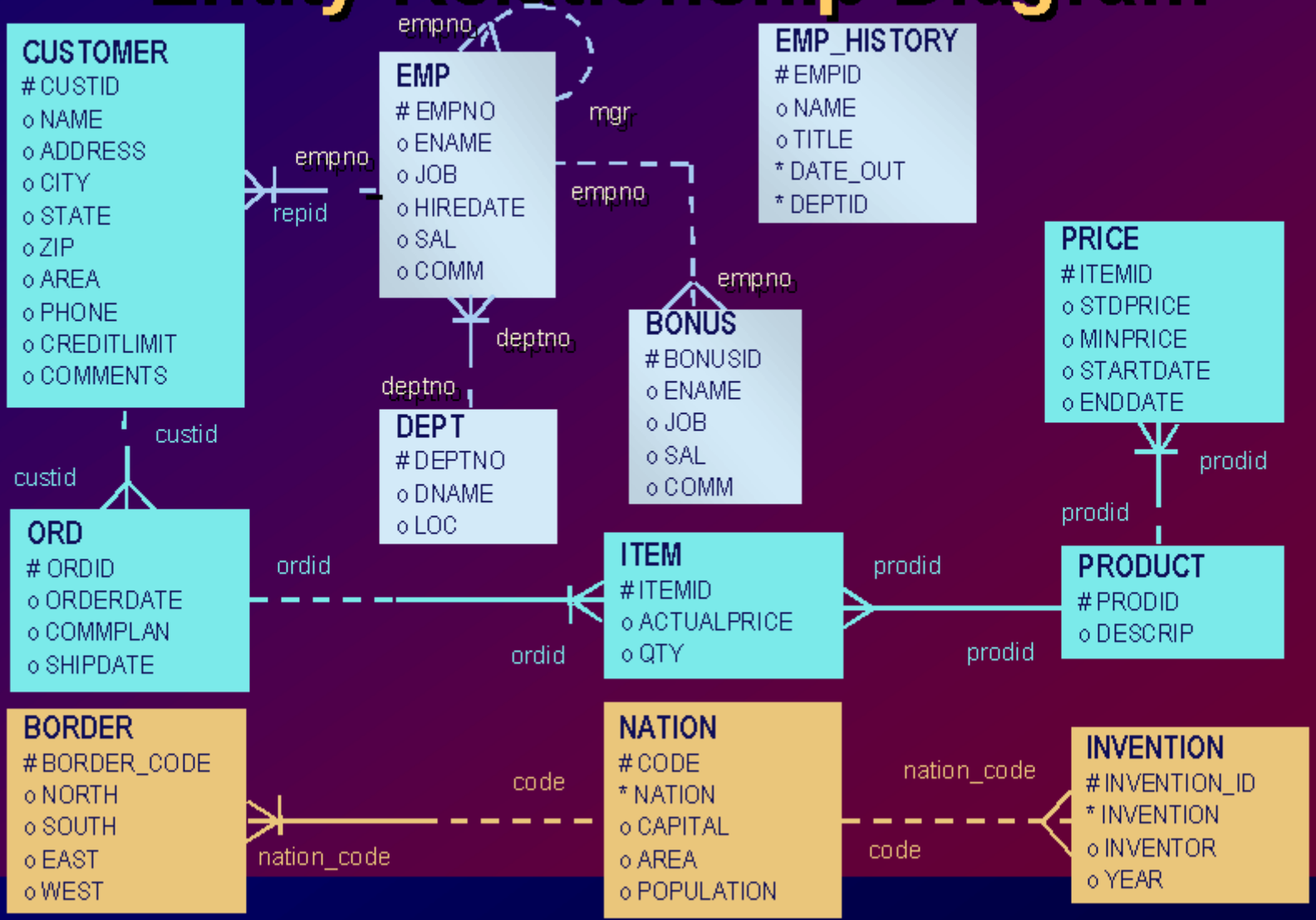
```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
IS|AS
PL/SQL Block;
```

Příklad – použití procedur

- vyrobíme procedury pro

  vložení / změnu / smazání produktu
- procedury později sdružíme do balíku (package)
- pro čtení vytvoříme pohledy
- pomocí SQL DCL zajistíme bezpečnost aplikace

Entity Relationship Diagram



ORACLE

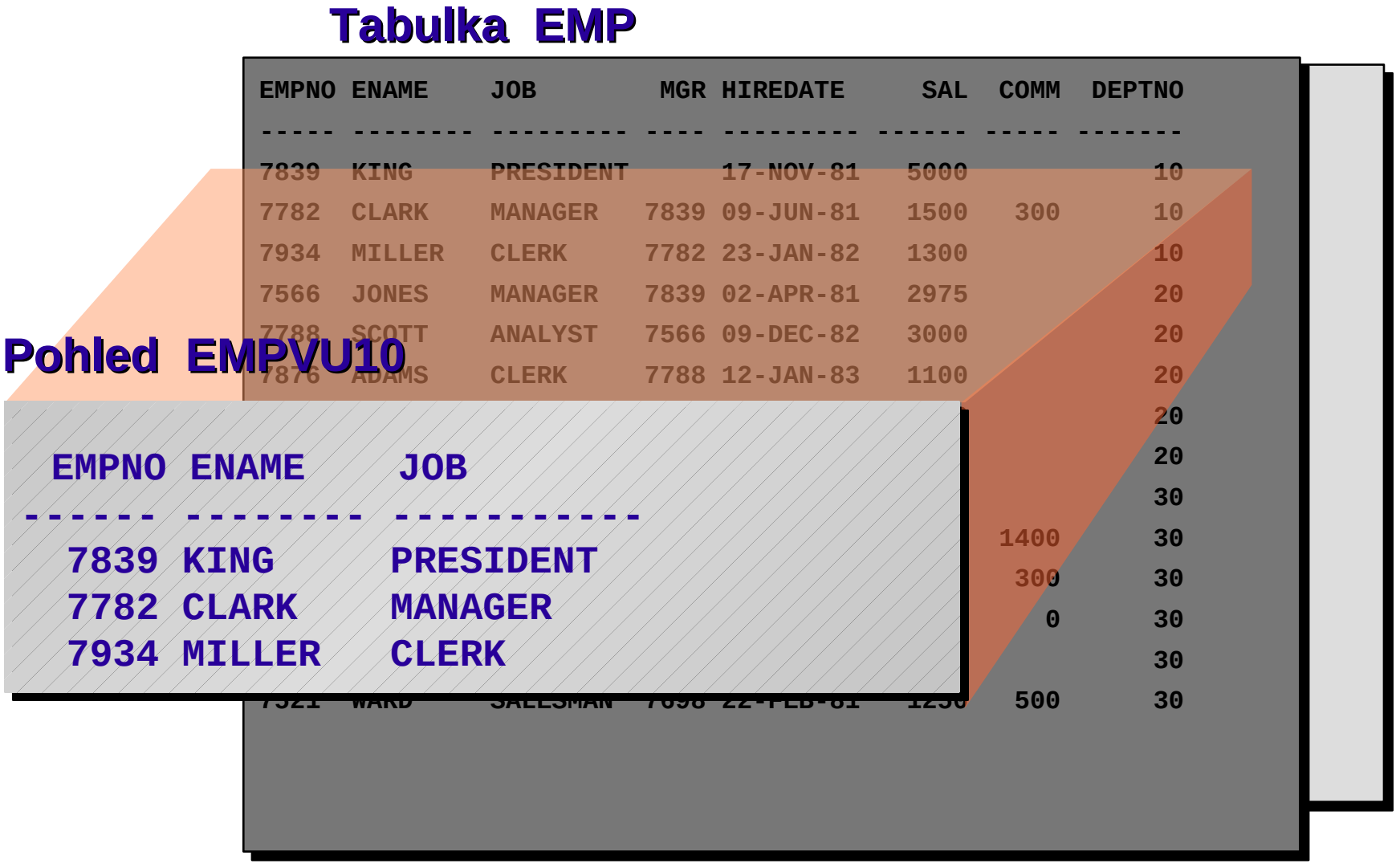
```
Create or Replace Procedure
  DEL_PROD(v_ProdId PRODUCT.PRODID%Type)
As
Begin
   Delete PRODUCT Where Prodid = v_Prodid;
   If Sql%NotFound Then
      Raise_Application_Error
      (-20203, 'Výrobek zadaného čísla '||v_Prodid||' neexistuje');
   End If;
End DEL_PROD;
/
```

```
Create or Replace Procedure
  UPD_PROD(v_Prodid PRODUCT.PRODID%Type,
                   v_Descrip PRODUCT.DESCRIP%Type)
As
Begin
   Update PRODUCT
         Set DESCRIP = v_Descrip
         Where Prodid = v_Prodid;
   If Sql%NotFound Then
      Raise_Application_Error
      (-20203, 'Výrobek zadaného čísla '||v_Prodid||' neexistuje');
   End If;
End UPD_PROD;
/
```

```
Create or Replace Procedure
  ADD_PROD(v_ProdId PRODUCT.PRODID%Type,
                   v_Descrip PRODUCT.Descrip%Type)
As
   Cons_vio Exception ;
   Pragma Exception_Init (Cons_vio, -1);
begin
   Insert Into PRODUCT (Prodid, Descrip)
              Values(v_ProdId, v_Descrip);
Exception
   When Cons_vio Then
        Raise_Application_Error
        (-20203,'Zadané číslo má již jiný výrobek');
End ADD_PROD;
/
```

```
Create or Replace Package Prod_Pack
-- specifikace modulu
As
Procedure ADD_PROD( v_Prodid  PRODUCT.PRODID%Type,
          v_Descrip PRODUCT.Descrip%Type);
Procedure UPD_PROD(v_Prodid PRODUCT.PRODID%Type,
           v_Descrip PRODUCT.DESCRIP%Type);
Procedure DEL_PROD (v_ProdId PRODUCT.PRODID%Type);
End Prod_Pack;
/
Create or Replace Package Body Prod_Pack
As
Procedure ADD_PROD( v_Prodid  PRODUCT.PRODID%Type,
          v_Descrip PRODUCT.Descrip%Type)
Is
    ...
```

Co je pohled?


Tabulka EMP
EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO
7839 KING PRESIDENT 17-NOV-81 5000 10
7782 CLARK MANAGER 7839 09-JUN-81 1500 300 10
7934 MILLER CLERK 7782 23-JAN-82 1300 10
7566 JONES MANAGER 7839 02-APR-81 2975 20
7788 SCOTT ANALYST 7566 09-DEC-82 3000 20
7876 ADAMS CLERK 7788 12-JAN-83 1100 20
20
20
30
1400 30
300 30
0 30
30
500 30
Pohled EMPVU10
EMPNO ENAME JOB
7839 KING PRESIDENT
7782 CLARK MANAGER
7934 MILLER CLERK

Proč používat pohledy?

- Pro omezení přístupu do databáze
- Pro zjednodušení složitých dotazů
- Pro prezentaci různých pohledů na stejná data

Vytváření pohledu

- Do příkazu CREATE VIEW můžete vložit poddotaz.

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW pohled
  [(alias[, alias]...)]
AS poddotaz
[WITH CHECK OPTION [CONSTRAINT omezení]]
[WITH READ ONLY]
```

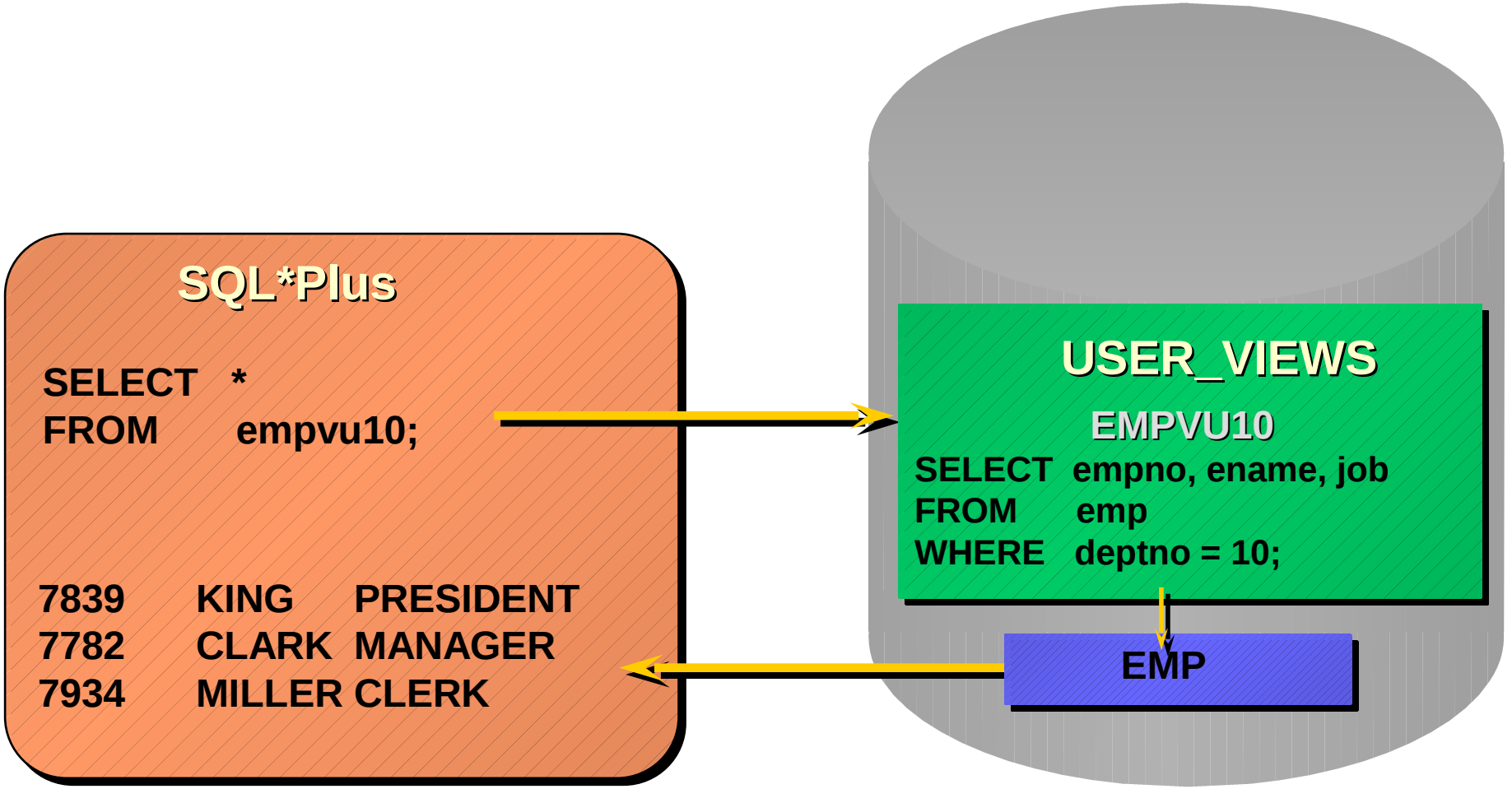
- Poddotaz může obsahovat komplexní syntaxi SELECT.
- Poddotaz nemůže obsahovat klauzuli ORDER BY.

Vytváření komplexního pohledu

Komplexní pohled obsahuje skupinové funkce, spojení, výrazy atd. Z principu věci nepodporuje DML. Lze ale použít INSTEAD-OF triggery.

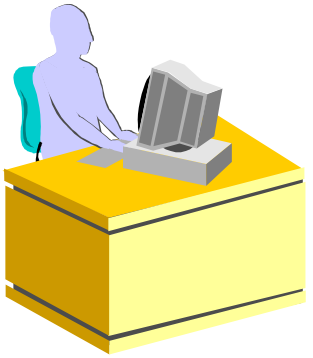
```
SQL> CREATE VIEW      dept_sum_vu
  2                   (name, minsal, maxsal, avgsal)
  3  AS SELECT        d.dname, MIN(e.sal), MAX(e.sal),
  4                   AVG(e.sal)
  5  FROM             emp e, dept d
  6  WHERE            e.deptno = d.deptno
  7  GROUP BY         d.dname;
View created.
```

Dotaz na pohled

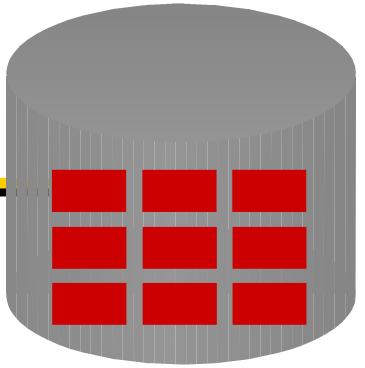


Řízení přístupu uživatelů

Administrátor databáze



Uživatelské jméno a heslo
práva



Uživatelé

Práva

- Zabezpečení databáze
  - zabezpečení systému
  - zabezpečení dat
- Systémová práva: získání přístupu do databáze
- Objektová práva: manipulace s obsahem objektů databáze
- Schéma: sada objektů, jako jsou tabulky, pohledy a sekvence

Systémová práva

- **Administrátor databáze má systémová práva vysoké úrovně.**
  - **vytváření nových uživatelů**
  - **odstraňování uživatelů**
  - **odstraňování tabulek**
  - **zálohování tabulek**

Systémová práva uživatele

- Jakmile je uživatel vytvořen, administrátor mu může přidělit specifická systémová práva.

```
GRANT právo [, právo...]
TO uživatel [, uživatel...];
```

- Tvůrce aplikace má následující systémová práva:
  - CREATE SESSION
  - CREATE TABLE
  - CREATE SEQUENCE
  - CREATE VIEW
  - CREATE PROCEDURE

Přidělování systémových práv

Admistrátor může přidělit uživateli specifická systémová práva.

```
SQL> GRANT  create table, create sequence, create view
  3  TO     scott;
Grant succeeded.
```

Objektová práva

- **Objektová práva se liší objekt od objektu.**
- **Vlastník má k objektu všechna práva.**
- **Vlastník může přidělit určitá práva ke svému objektu.**

```
GRANT        objektové_právo [(sloupce)]
ON           objekt
TO           {uživatel|role|PUBLIC}
[WITH GRANT OPTION];
```

Přidělování objektových práv

- Přidělíme práva dotazů pro tabulku EMP.

```
SQL> GRANT    select
  2  ON       emp
  3  TO       sue, rich;
Grant succeeded.
```

- Přidělíme práva pro aktualizaci určitých sloupců uživatelům a rolím.

```
SQL> GRANT    update (dname, loc)
  2  ON       dept
  3  TO       scott, manager;
Grant succeeded.
```

Použití klíčových slov WITH GRANT OPTION a PUBLIC

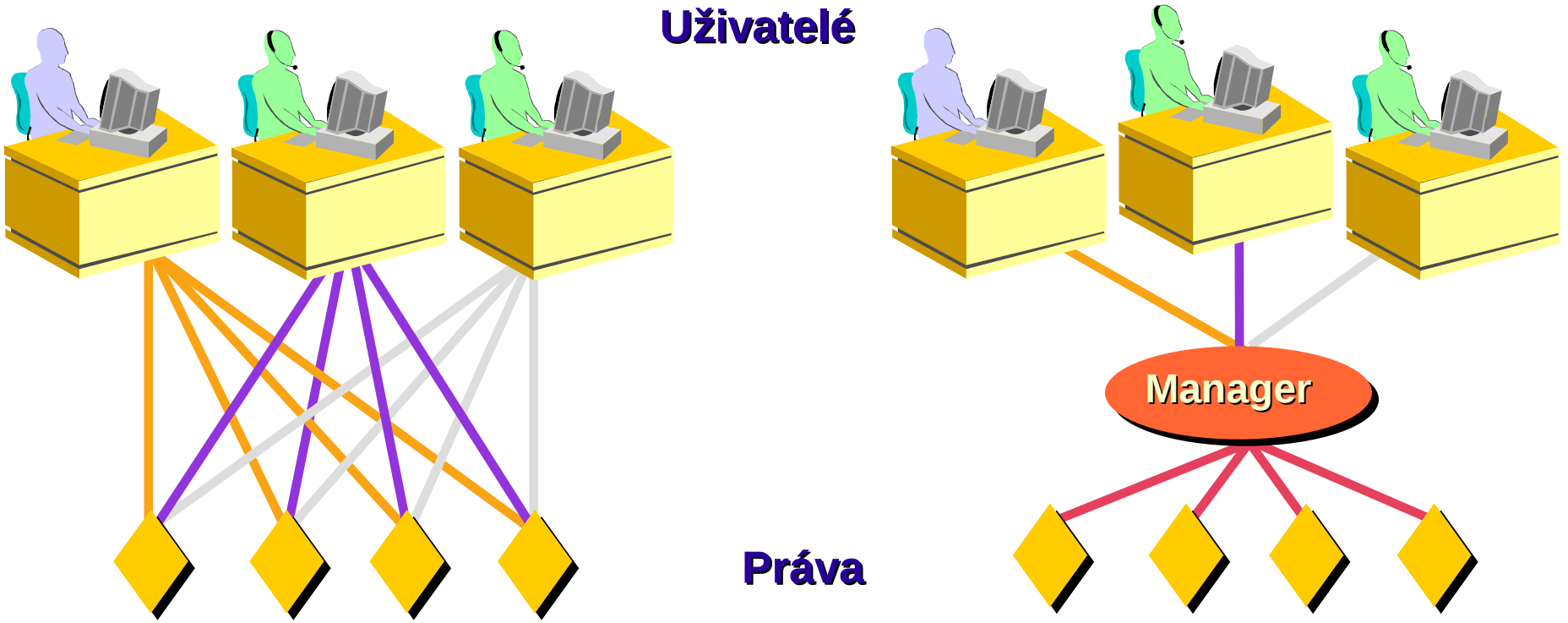
- Umožníme uživateli předávat práva dále.

```
SQL> GRANT    select, insert
  2  ON       dept
  3  TO       scott
  4  WITH GRANT OPTION;
Grant succeeded.
```

- Umožníme všem uživatelů provádět dotazy v tabulce DEPT uživatele Alice.

```
SQL> GRANT    select
  2  ON       alice.dept
  3  TO       PUBLIC;
Grant succeeded.
```

Co je role?



Přidělování práv
bez rolí

Přidělování práv
s použitím rolí

Vytváření role

```
SQL> CREATE ROLE manager;
Role created.
```

```
SQL> GRANT create table, create view
2           to manager;
Grant succeeded.
```

```
SQL> GRANT manager to BLAKE, CLARK;
Grant succeeded.
```

Práva a role - shrnutí

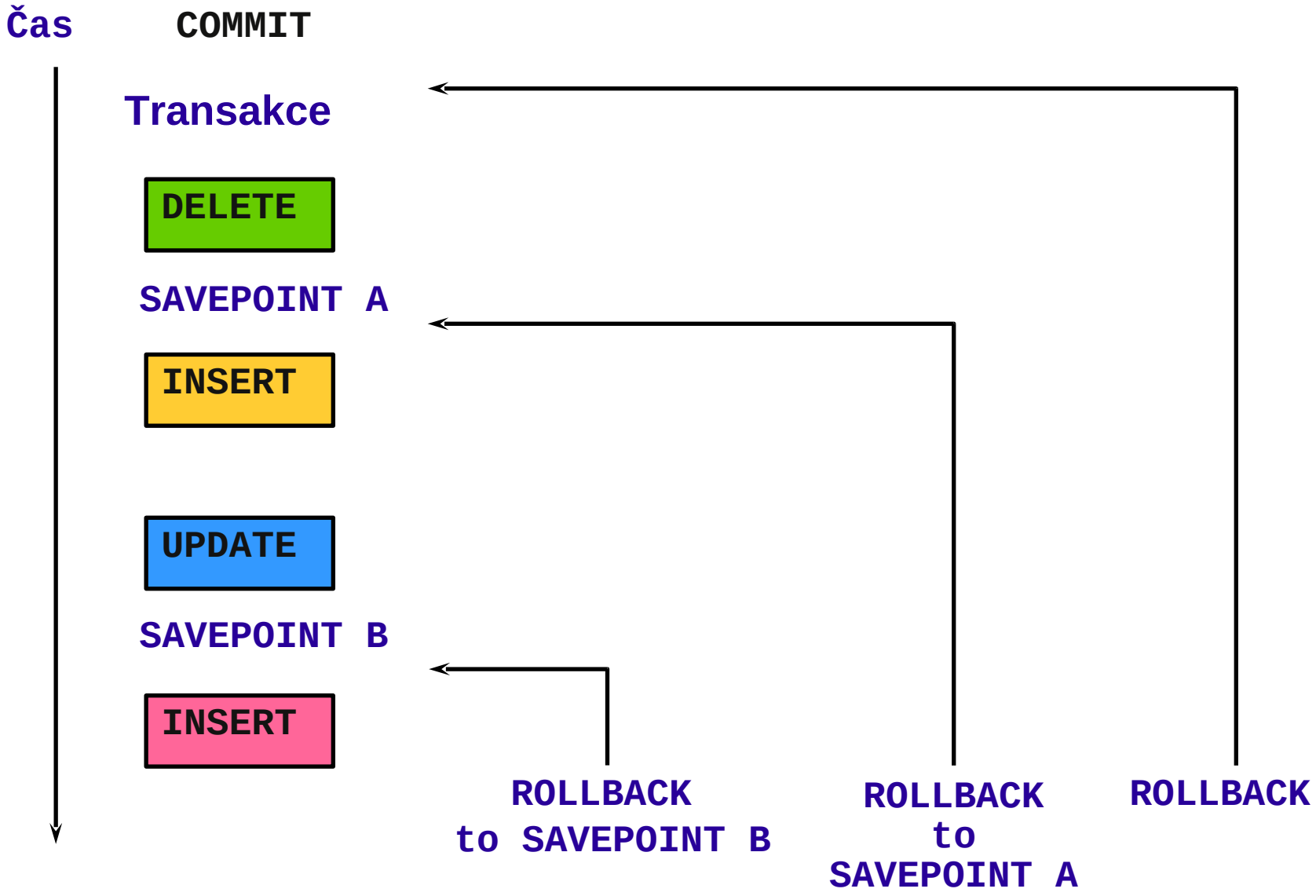
CREATE USER	**Umožňuje administrátoru vytvořit uživatele**
GRANT	**Umožňuje uživateli přidělit jiným uživatelům práva pro přístup ke svým objektům**
CREATE ROLE	**Umožňuje administrátoru vytvořit sadu práv**
ALTER USER	**Umožňuje uživatelům měnit jejich hesla**
REVOKE	**Odnímá uživatelům práva pro objekt**

Řízení transakcí – příkazy COMMIT a ROLLBACK

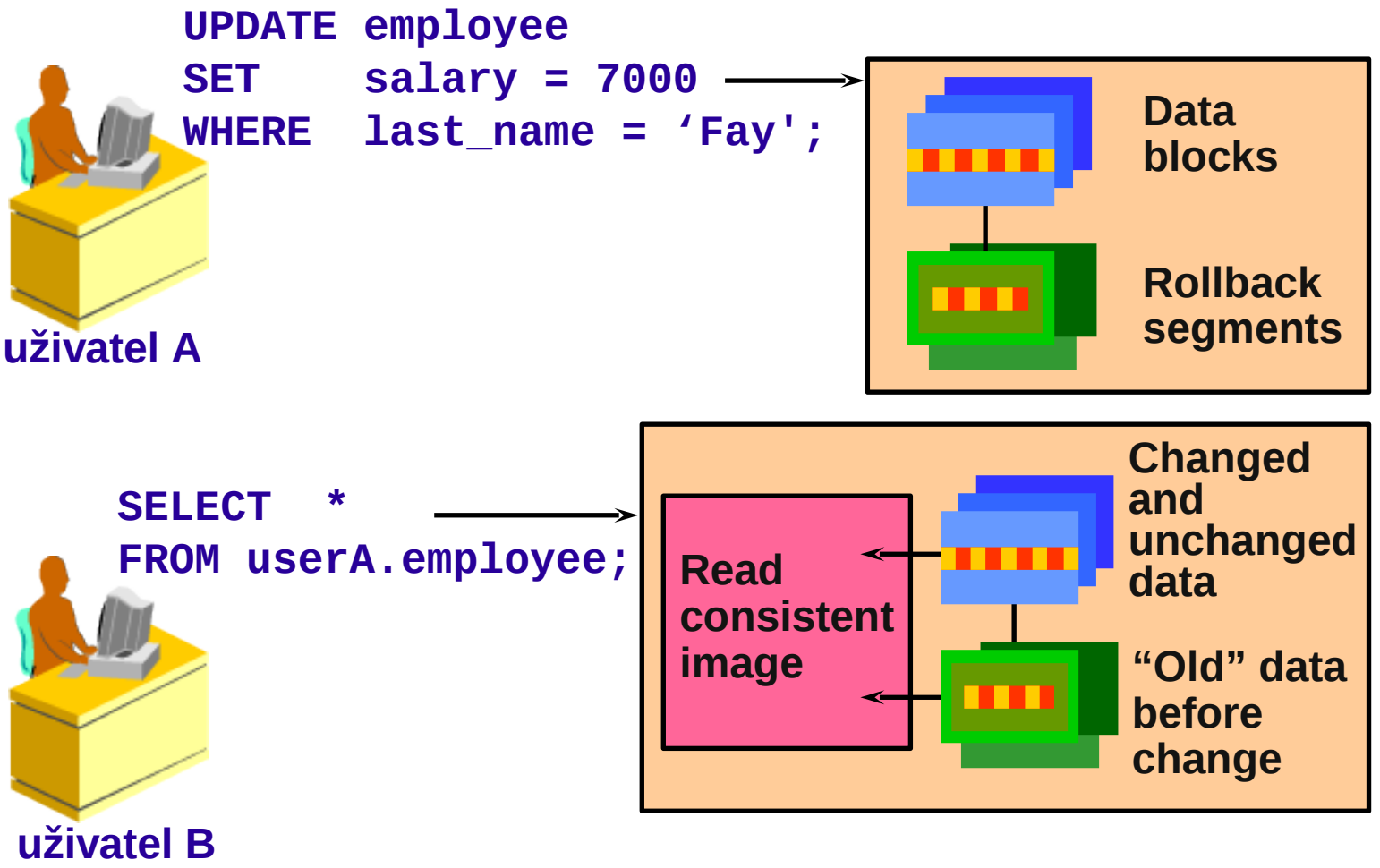
- COMMIT a ROLLBACK zajišťují konzistenci dat.
- Uživatel vidí změněná data pouze v relaci (session), ve které je měnil.
- Seskupení logicky souvisejících operací.

- Implicitní commit :
  - DDL příkaz (`CREATE, ...`).
  - DCL příkaz (`GRANT, REVOKE`).
  - Normální ukončení klienta (SLQ*'Plus.
- Implicitní rollback – abnornální ukončení klienta

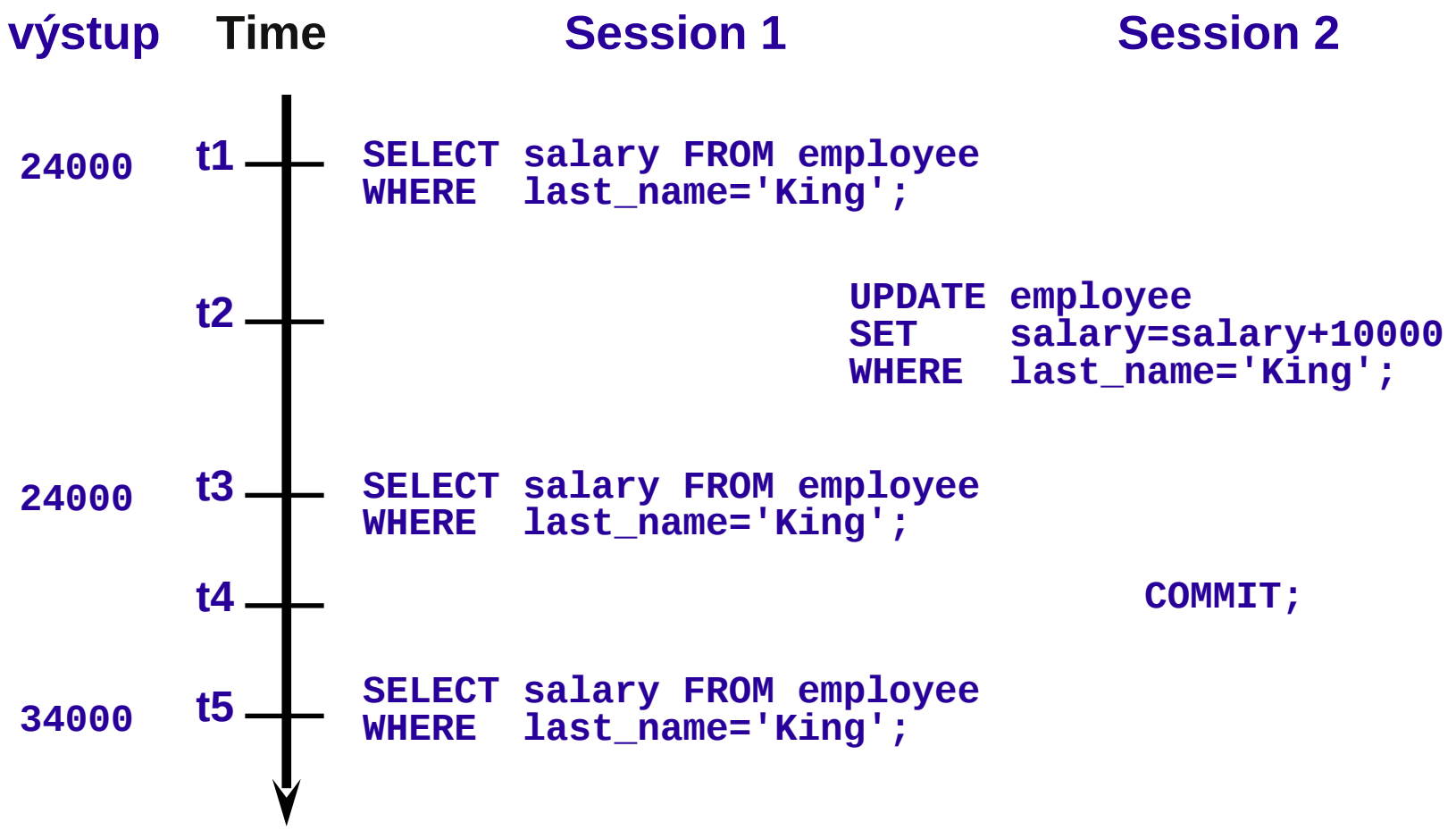
Řízení transakce



Implemtace konzistentího čtení



Konzistentní čtení - příklad



Vlastnosti transakcí

- Atomicity
- Consistency
- Independence (Isolation)
- Durability