

course:

Database Systems (NDBI025)

SS2011/12

lecture 4:

SQL – data definition & modification, views

doc. RNDr. Tomáš Skopal, Ph.D.

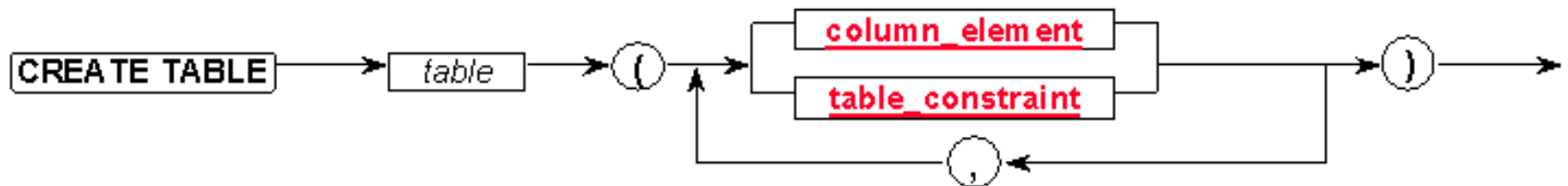
RNDr. Michal Kopecký, Ph.D.

Today's lecture outline

- data definition
 - definition of tables (their schemes) and integrity constraints – CREATE TABLE
 - altering the definitions – ALTER TABLE
- data manipulation
 - INSERT, UPDATE, DELETE
- database views

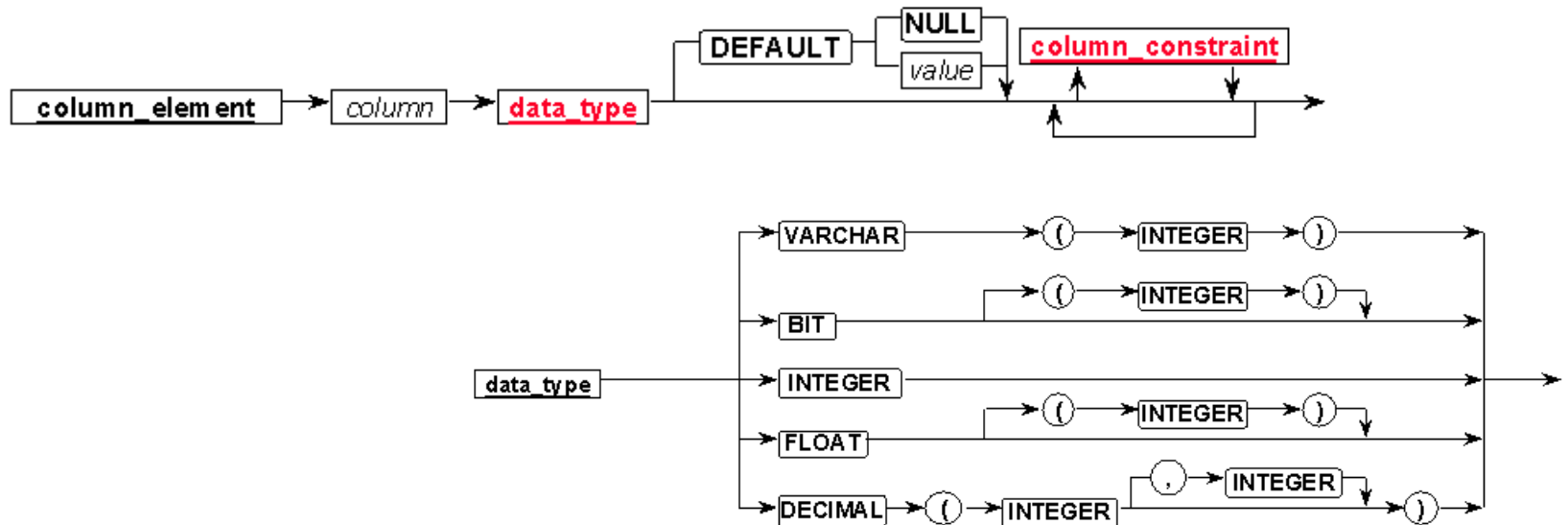
CREATE TABLE – basic construction

- construction of table schema and an empty table
- table name, columns defined, column-scope integrity constraints (IC), table-scope integrity constraints



CREATE TABLE – column definition

- each column has a data_type
- optionally
 - default value within a new record (DEFAULT NULL | value)
 - column-scope integrity constraints

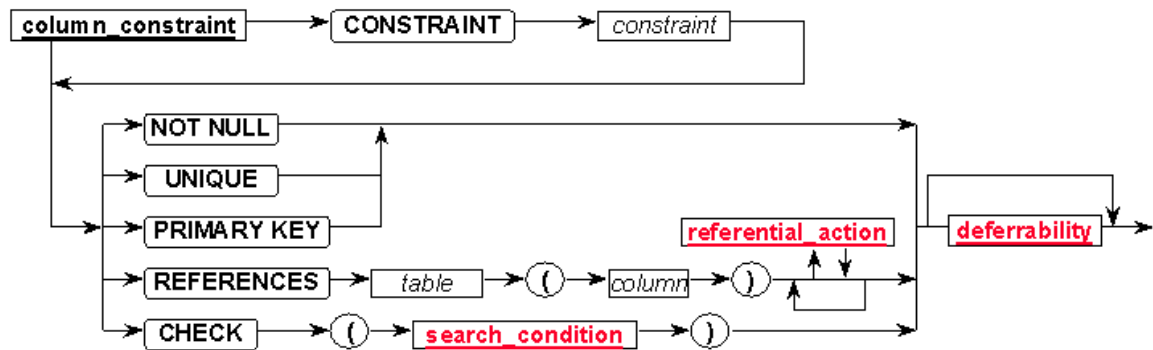


Example – simple table (w/o IC)

```
CREATE TABLE Product  
  (Id INTEGER, Name VARCHAR(128), Price DECIMAL(6,2),  
   ProductionDate DATE, Available BIT DEFAULT TRUE, Weight FLOAT)
```

CREATE TABLE – column-scope integrity constraint

- column-based IC allows to limit the domain of values in column in record (new or modified)
 - named **CREATE TABLE** ... (... , **CONSTRAINT** constraint ...)
 - unnamed
- 5 types limiting a valid value
 - NOT NULL** – value cannot be null
 - UNIQUE** – value must be unique (w.r.t all records in the table)
 - PRIMARY KEY** – primary key definition (key = **NOT NULL** + **UNIQUE**, primary = primary index)
 - REFERENCES** – one-column foreign key (both columns must share definition)
 - CHECK** – generic condition, similarly as in **SELECT ... WHERE**
 - applied only on the inserted/updated row(s)
 - data inserted/update only if TRUE
- if an IC is not validated, the record is not updated



Example – definition of tables with column-scope ICs

CREATE TABLE Product

(Id **INTEGER CONSTRAINT** pk **PRIMARY KEY**, Name **VARCHAR**(128) **UNIQUE**,
Price **DECIMAL**(6,2) **NOT NULL**, ProductionDate **DATE**, Available **BIT DEFAULT**
TRUE, Weight **FLOAT**,
Producer **INTEGER REFERENCES** Producer (Id))

CREATE TABLE Producer

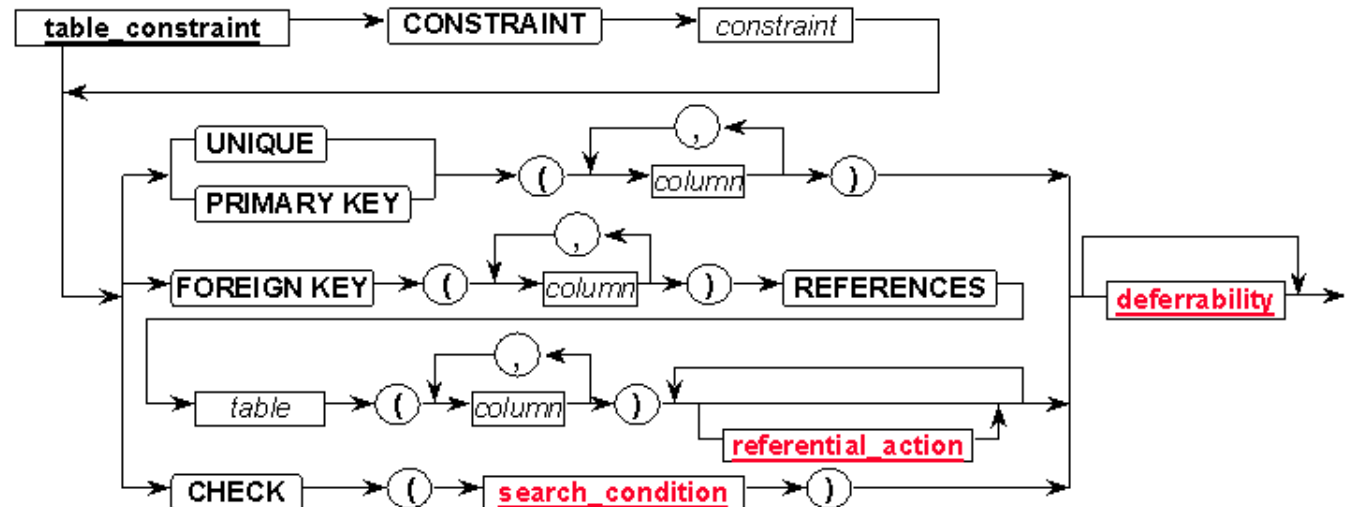
(Id **INTEGER PRIMARY KEY**, ProducerName **VARCHAR**(128),
HQ **VARCHAR**(256))

Example – foreign key (single table)

```
CREATE TABLE Employee  
  (IdEmp INTEGER PRIMARY KEY, Name VARCHAR(128),  
   Boss INTEGER REFERENCES Employee (IdEmp))
```

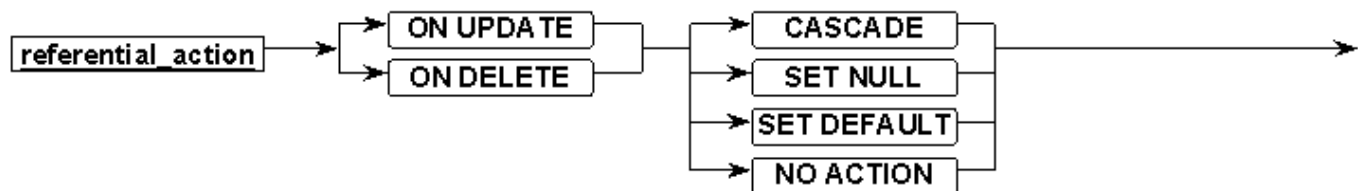

CREATE TABLE – table-scope integrity constraint

- generalization of column-scope IC to use for multiple columns
 - apart NOT NULL, this is only meaningful for single column
- **UNIQUE** – n-tuple of values is unique
- **FOREIGN KEY** – the same as **REFERENCES** in column-scope IC
- **CHECK**



Referential integrity

- when updating **referencing** or **referenced** table, violation of foreign keys may occur
 - record update with foreign key value that is not valid (not present in column of referenced table)
 - delete/update of a record in the referenced table (so that value in referencing table becomes not covered)
- when a violation of foreign keys occurs, there are two options
 - if there is no referential action defined, an error message appears, aborting the action (SQL 89)
 - referential action is triggered, **referential_action** (SQL 92)
 - **ON UPDATE, ON DELETE** – definition of when to trigger the action, either by update or delete of row in the referenced table
 - **CASCADE** – the row with the referencing value is treated the same (i.e., updated or deleted)
 - **SET NULL** – referencing value in the row is set to NULL
 - **SET DEFAULT** – referencing value in the row is set to a default value defined in CREATE TABLE
 - **NO ACTION** – default, no action takes place, resp., DBMS displays a message (SQL 89)



Example – table with table-scope IC

CREATE TABLE Product

(Id **INTEGER PRIMARY KEY**, Name **VARCHAR**(128) **UNIQUE**, Price **DECIMAL**(6,2) **NOT NULL**, ProductionDate **DATE**, Available **BIT DEFAULT TRUE**, Weight **FLOAT**, Producer **VARCHAR**(128), ProducerHQ **VARCHAR**(256), **CONSTRAINT** fk **FOREIGN KEY** (Producer, ProducerHQ) **REFERENCES** Producer (ProducerName,HQ))

CREATE TABLE Producer

(ProducerName **VARCHAR**(128), HQ **VARCHAR**(256), Subject **VARCHAR**(64), **CONSTRAINT** pk **PRIMARY KEY**(ProducerName, HQ))

Example – CHECK

```
CREATE TABLE Product
(Id INTEGER PRIMARY KEY, Name VARCHAR(128) UNIQUE, Price
DECIMAL(6,2) NOT NULL, ProductionDate DATE, Available BIT DEFAULT
TRUE, Weight FLOAT,
CONSTRAINT chk CHECK
(Weight > 0))
```

Example – ON DELETE, ON UPDATE

CREATE TABLE Product

(Id **INTEGER CONSTRAINT** pk **PRIMARY KEY**, Name **VARCHAR**(128) **UNIQUE**,
Price **DECIMAL**(6,2) **NOT NULL**, ProductionDate **DATE**, Available **BIT DEFAULT**
TRUE, Weight **FLOAT**,
Producer **INTEGER REFERENCES** Producer (Id)
ON DELETE CASCADE)

CREATE TABLE Producer

(Id **INTEGER PRIMARY KEY**, ProducerName **VARCHAR**(128),
HQ **VARCHAR**(256))

ALTER TABLE

- table scheme definition altering
 - column – add/remove column, change of DEFAULT value
 - IC – add/remove IC
- however, there may be data that won't allow the IC change (e.g., primary key definition)

ALTER TABLE table-name

... **ADD** [**COLUMN**] column-name column-definition

... **ADD** constraint-definition

... **ALTER** [**COLUMN**] column-name SET

... **ALTER** [**COLUMN**] column-name DROP

... **DROP COLUMN** column-name

... **DROP CONSTRAINT** constraint-name

Example – ALTER TABLE

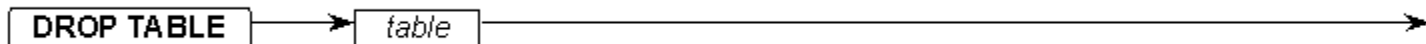
```
CREATE TABLE Product  
(Id INTEGER PRIMARY KEY, Name VARCHAR(128) UNIQUE, Price  
DECIMAL(6,2) NOT NULL, ProductionDate DATE, Available BIT DEFAULT TRUE,  
Weight FLOAT,  
CONSTRAINT chk CHECK  
(AND Weight > 0))
```

```
ALTER TABLE Product DROP CONSTRAINT chk
```

```
ALTER TABLE Product ADD CONSTRAINT chk CHECK  
(Weight > 10))
```

DROP TABLE

- **DROP TABLE** *table*
- complementary to **CREATE TABLE** *table*
- the table content is deleted and also the definition (i.e., scheme)
 - if we need to delete only the content we use **DELETE FROM** *table* (see next)

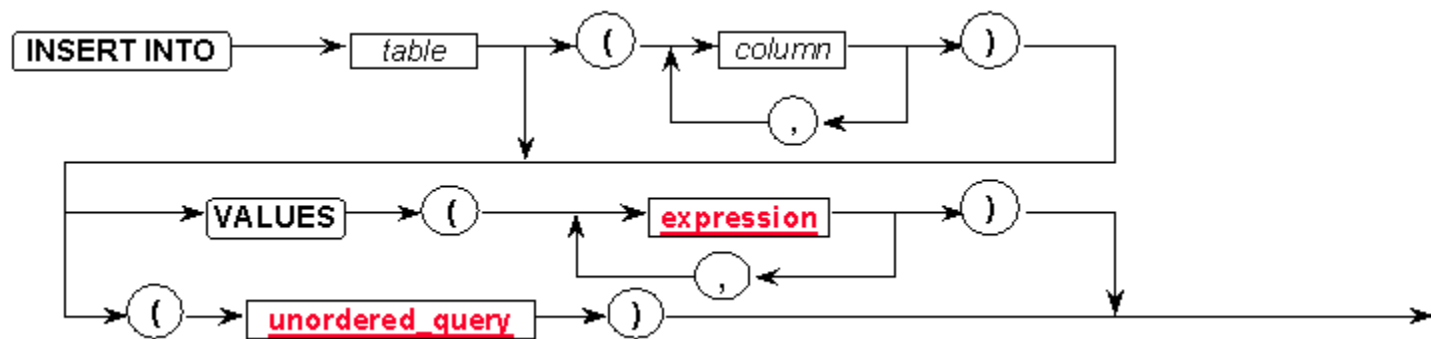


Data modification

- apart SELECT, the SQL DML offers three commands for data manipulation
 - INSERT INTO – insertion of rows
 - DELETE FROM – deletion of rows
 - UPDATE – update of rows

INSERT INTO

- insertion of row by enumeration, two options
 - **INSERT INTO** table (col1, col3, col5) **VALUES** (val1, val3, val5)
 - **INSERT INTO** table **VALUES** (val1, val2, val3, val4, val5)
- insertion of multiple rows that result from a SELECT
 - **INSERT INTO** table1 | (column list) | **(SELECT ... FROM ...)**



Example – INSERT INTO

CREATE TABLE Product

(Id **INTEGER CONSTRAINT** pk **PRIMARY KEY**, Name **VARCHAR**(128)
UNIQUE, Price **DECIMAL**(6,2) **NOT NULL**, ProductionDate **DATE**, Available
BIT DEFAULT TRUE, Weight **FLOAT**,
Producer **INTEGER REFERENCES** Producer (Id))

INSERT INTO Product **VALUES** (0, 'Chair', 86, '2005-5-6', TRUE, 3, 123456)

INSERT INTO Product (Id, Name, Price, ProductionDate, Weight, Producer)
VALUES (0, 'Chair', 86, '2005-5-6', 3, 123456)

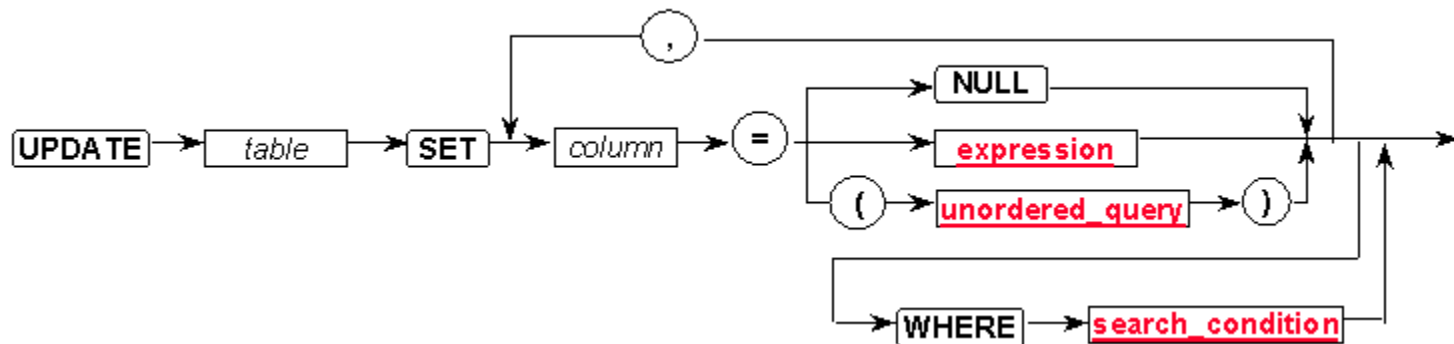
Example – INSERT INTO

```
CREATE TABLE ProductInStore  
  (Id INTEGER PRIMARY KEY, Name VARCHAR(128) UNIQUE, Price  
   DECIMAL(6,2))
```

```
INSERT INTO ProductInStore VALUES  
  (SELECT Id, Name, Price FROM Product WHERE Available = TRUE)
```

UPDATE

- update of rows matching a condition
- the values of chosen columns are set to
 - NULL
 - value given by expression (also constant)
 - query result



Example – UPDATE

UPDATE Product **SET** Name = 'Notebook' **WHERE** Name = 'Laptop'

UPDATE Product **SET** Price = Price * 0.9 **WHERE** **CAST**(ProductionDate **AS** VARCHAR(32)) < '2009-01-01'

UPDATE Product **AS** V1 **SET** Weight = (**SELECT** **AVG**(Weight) **FROM** Product **AS** V2 **WHERE** V1.Name = V2.Name)

DELETE FROM

- deletes rows that match condition
- **DELETE FROM** *table* deletes all rows

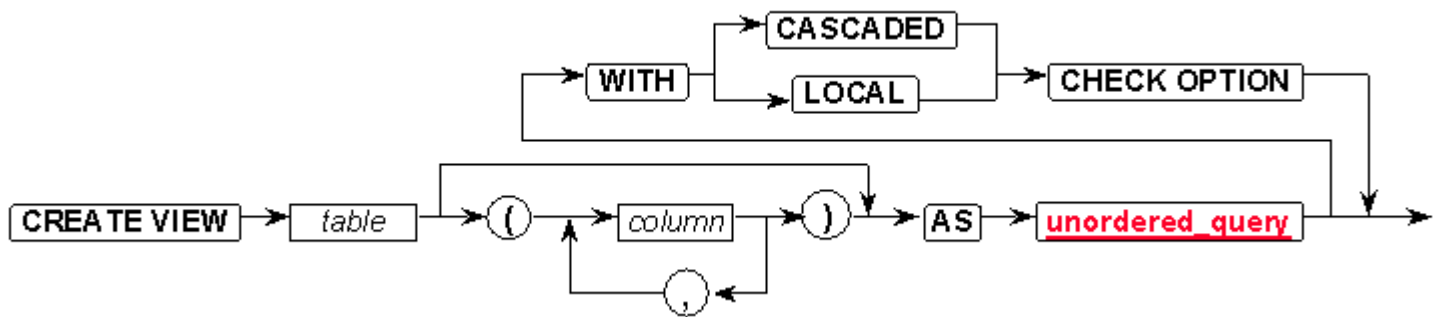
Example:

DELETE FROM Product **WHERE** Price > 100



Database views

- **named query that could be** used as a table (its result, resp.)
- evaluated dynamically
 - could be used also for data inserts/updates/deletes
- CHECK OPTION ensures that after row insert/update into the view the change will be visible
 - at least in this view
 - in all dependent views



Example – views

```
CREATE VIEW NewProductInStore AS  
  SELECT * FROM Product WHERE Available = TRUE AND CAST(ProductionDate  
  AS VARCHAR(32)) > '2009-01-01'  
WITH LOCAL CHECK OPTION
```

```
INSERT INTO NewProductInStore VALUES  
  (0, 'SwingChair', 135, '2010-05-06', TRUE, 4, 3215)  
- inserted (transitively into table Product)
```

```
INSERT INTO NewProductInStore VALUES  
  (0, 'Box', 135, '1999-11-07', TRUE, 4, 3215)  
!! Error – this record cannot be inserted, as it wouldn't be visible (too old box)
```