5. SQL Querying

Views: Making Access Easy

- Views are *virtual* tables defined by SFW-expressions.
- ```
  CREATE VIEW V AS
     SELECT ... FROM ... WHERE ...;
  ```
- Tables introduced by the `CREATE TABLE`-clause, i.e. the *real* tables, are called *base* tables.
- A view can be used wherever we expect a table in an SFW-expression.

```
CREATE VIEW Capital-Info AS
   SELECT Capital, Inhabitants
   FROM Country, City
   WHERE Country.CoCode = City.CoCode;
SELECT * FROM Capital-Info
   WHERE Inhabitants =
      ( SELECT MAX(Inhabitants) FROM Capital-Info );
```
```

Dynamics: Insert, Delete and Update

A new EU-member.

```
INSERT INTO Membership (LCoCode, Organization, Status)
   VALUES ('PL', 'EU', 'member')
```

```
INSERT INTO Country ( CoCode )
   SELECT DISTINCT M.CoCode
      FROM Membership M
      WHERE NOT EXISTS (
         SELECT L.CoCode
            FROM Country L
            WHERE L.CoCode = M.CoCode)
```

Sequence-numbers

```
CREATE TABLE Country
   CoID INTEGER GENERATED ALWAYS AS IDENTITY
   ( START WITH 1
   INCREMENT BY 1
   MINVALUE 1
   MAXVALUE 100000
   NO CYCLE),
   ⋮

INSERT INTO Country
   (CoName, CoCode, Capital)
   VALUES ( 'Bavaria', 'BY', 'Munich')
```

```
DELETE FROM Stadt
```

```
DELETE FROM City
   WHERE Longitude < 90;
```

```
UPDATE City
   SET Inhabitants = Inhabitants * 1.1,
      ...WHERE Inhabitants > 1000;
```

Some Integrity Issues

- *Integrity constraint* define the legal states of a database.
- The DBMS guarantees that all integrity constraints known are obeyed.
- Primary key and foreign key clauses express integrity constraints.
  No null-values allowed for primary keys!
- Other kinds are *domain* constraints:

```
CREATE TABLE Location (
   Continent VARCHAR(35) NONNULL
   Percentage NUMBER DEFAULT 100;
```

- More general ones are based on the *check*-clause:

```
CREATE TABLE City (
   Longitude   NUMBER,
   Latitude    NUMBER,
   CHECK (Longitude BETWEEN -180 AND 180),
   CHECK (Latitude BETWEEN -90 AND 90);
```

Foreign Keys: the (nearly) complete story

Examples

```
CREATE TABLE Province (
   PrName   VARCHAR(35),
   CoCode   VARCHAR(4),
   PRIMARY KEY (PrName, CoCode),
   FOREIGN KEY (CoCode) REFERENCES Country (CoCode) )

CREATE TABLE Border (
   CoCode1  VARCHAR(4),
   CoCode2  VARCHAR(4),
   Length   INTEGER,
   PRIMARY KEY (CoCode1, CoCode2),
   FOREIGN KEY (CoCode1) REFERENCES Country (CoCode),
   FOREIGN KEY (CoCode2) REFERENCES Country (CoCode) )

CREATE TABLE City ( ...
   PRIMARY KEY (CiName, CoCode, PrName),
   FOREIGN KEY (CoCode) REFERENCES Country (CoCode),
   FOREIGN KEY (CoCode, PrName) REFERENCES Province (CoCode, PrName) )
```

Referential actions

Without any additional provisions the DBMS will cancel all operations on a databse which are going to violate integrity constraints.

Good news: for referential integrity we can, in many cases, specify actions to correct violations; these are the referential actions.

For a given `FOREIGN KEY`-clause, the table containing the clause is called *(child) C*-table and the referenced table is called *(parent) P*-table.

Referential actions are part of the C-table; they state what has to happen when a `DELETE`- or `UPDATE`-operation executed with respect to the corresponding P-table violates referential integrity.

```
CREATE TABLE Province (
   ⋮
   FOREIGN KEY (CoCode) REFERENCES Country (CoCode)
      ON DELETE CASCADE ON UPDATE NO ACTION )


CREATE TABLE City (
   ⋮
   PRIMARY KEY (CiNAME)
   FOREIGN KEY (CoCode, PrName)
      REFERENCES Province (CoCode, PrName)
      ON DELETE SET NULL ON UPDATE SET DEFAULT )
```

```
DROP TABLE T4; DROP TABLE T3; DROP TABLE T2; DROP TABLE T1;

CREATE TABLE T1( k1 NUMERIC, PRIMARY KEY(k1) );

CREATE TABLE T2( k2 NUMERIC, k1 NUMERIC,
PRIMARY KEY(k2), FOREIGN KEY(k1) REFERENCES T1(k1) ON DELETE CASCADE );

CREATE TABLE T3( k3 NUMERIC, k1 NUMERIC,
PRIMARY KEY(k3), FOREIGN KEY(k1) REFERENCES T1(k1) ON DELETE CASCADE );

CREATE TABLE T4( k4 NUMERIC, k2 NUMERIC, k3 NUMERIC, PRIMARY KEY(k4),
FOREIGN KEY(k2) REFERENCES T2(k2) ON DELETE CASCADE,
FOREIGN KEY(k3) REFERENCES T3(k3) ON DELETE SET NULL );

INSERT INTO T1(k1) VALUES(1); INSERT INTO T1(k1) VALUES(2);
INSERT INTO T2(k2,k1) VALUES(21,1); INSERT INTO T2(k2,k1) VALUES(22,2);
INSERT INTO T3(k3,k1) VALUES(31,1); INSERT INTO T3(k3,k1) VALUES(32,2);
INSERT INTO T4(k4,k2,k3) VALUES(41,21,31);
INSERT INTO T4(k4,k2,k3) VALUES(42,22,32);
INSERT INTO T4(k4,k2,k3) VALUES(43,22,31);

delete from T1 where k1 = 2;
```

Trigger: A Powerful Machanism Based on Events

T1	K1
	1

T2	K2	K1
	a	1

T3	K3	K1
	b	1

T4	K4	K2	K3
	c	a	b

```
/* DELETE CASCADE bei T2->T1 and T3->T1 */
 CREATE TRIGGER t1delete_t2undt3
    AFTER DELETE ON T1 REFERENCING OLD as oldrow
    FOR EACH ROW
    BEGIN DELETE FROM T2 WHERE k1=oldrow.k1;
          DELETE FROM T3 WHERE k1=oldrow.k1; END

/* DELETE CASCADE bei T4->T2 */
 CREATE TRIGGER t2delete_t4
    AFTER DELETE ON T2 REFERENCING OLD as oldrow
    FOR EACH ROW DELETE FROM T4 WHERE k2=oldrow.k2;

/* DELETE RESTRICT bei T4->T3 */
 CREATE TRIGGER t3delete_t4
    AFTER DELETE ON T3 REFERENCING OLD as oldrow
    FOR EACH ROW DELETE FROM T4 WHERE k3 = oldrow.k3
```

On Trigger

- Trigger are *Event-Condition-Action (ECA)*-rules; they tell us what will be done when the event occurs and the condition is fulfilled.
- Powerful mechanism to check and possibly correct violations of integrity.
- Executing a trigger may activate another trigger, and so on. Danger of nontermination!

Outlook: Analysis

Online Analytical Processing (OLAP): ROLLUP and CUBE

SALES

Model	Year	Color	Sales
Chevy	1990	red	5
Chevy	1990	white	87
Chevy	1990	blue	62
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	blue	49
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	blue	71
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	blue	63
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	blue	55
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	blue	39

$\Longrightarrow$ *ROLLUP*

ROLLUP

Model	Year	Color	Sales
Chevy	1990	blue	62
Chevy	1990	red	5
Chevy	1990	white	87
Chevy	1990	ALL	154
Chevy	1991	blue	49
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	ALL	198
Chevy	1992	blue	71
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	ALL	156
Chevy	ALL	ALL	508
Ford	1990	blue	63
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	ALL	189
Ford	1991	blue	55
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	ALL	116
Ford	1992	blue	39
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	ALL	128
Ford	ALL	ALL	433
ALL	ALL	ALL	941

SALES

Model	Year	Color	Sales
Chevy	1990	red	5
Chevy	1990	white	87
Chevy	1990	blue	62
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	blue	49
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	blue	71
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	blue	63
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	blue	55
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	blue	39

$\Longrightarrow$ *CUBE*

DATA CUBE

Model	Year	Color	Sales
Chevy	1990	blue	62
Chevy	1990	red	5
Chevy	1990	white	87
Chevy	1990	ALL	154
Chevy	1991	blue	49
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	ALL	198
Chevy	1992	blue	71
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	ALL	156
Chevy	ALL	blue	182
Chevy	ALL	red	90
Chevy	ALL	white	236
Chevy	ALL	ALL	508
Ford	1990	blue	63
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	ALL	189
Ford	1991	blue	55
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	ALL	116
Ford	1992	blue	39
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	ALL	128
Ford	ALL	blue	157
Ford	ALL	red	143
Ford	ALL	white	133
Ford	ALL	ALL	433
.	.	.	
.	.	.	
.	.	.	
.	.	.	
.	.	.	
.	.	.	
ALL	1990	blue	125
ALL	1990	red	69
ALL	1990	white	149
ALL	1990	ALL	343
ALL	1991	blue	106
ALL	1991	red	104
ALL	1991	white	110
ALL	1991	ALL	314
ALL	1992	blue	110
ALL	1992	red	58
ALL	1992	white	116
ALL	1992	ALL	284
ALL	ALL	blue	339
ALL	ALL	red	233
ALL	ALL	white	369
ALL	ALL	ALL	941

OLAP Useful for *Datawarehouse*-applications, i.e. applications based on *historical* data. This is in contrast to *Online Transaction Processing* (OLTP), i.e. applications on the *production* data.

OLAP in SQL

```
SELECT Model, Year, Color, sum(Sales)
FROM SALES
GROUP BY ROLLUP(Model, Year, Color)

SELECT Model, Year, Color, sum(Sales)
FROM SALES
GROUP BY CUBE(Model, Year, Color)
```