Chapter 5
Relational Databases and SQL: Further Issues

- Data Definition Language (DDL):
  schema generation
- Data Manipulation Language (DML):
  - queries
  - insertions, deletions, modifications
- Database behavior?

5.1 Database Schema

The database schema is the complete model of the structure of the application domain (here: relational schema):

- relations
  - names of attributes
  - domains of attributes
  - keys
- additional constraints
  - value constraints
  - referential integrity constraints
- storage issues of the physical schema: indexes, clustering etc. also belong to the schema

5.1.1 Schema Generation in SQL

Definition of Tables

Basic form: attribute names and domains

```
CREATE TABLE <table>
   (<col> <datatype>,
     ⋮
    <col> <datatype>)
```

domains: NUMBER, CHAR(n), VARCHAR2(n), DATE ...

```
CREATE TABLE City
      ( Name          VARCHAR2(35),
        Country       VARCHAR2(4),
        Province      VARCHAR2(32),
        Population    NUMBER,
        Longitude     NUMBER,
        Latitude      NUMBER );
```


Integrity constraints

Simple constraints on individual attributes are given with the attribute definitions as "column constraints":

- domain definitions are already integrity constraints
- further constraints on individual attribute values
  more detailed range restrictions:
  City: CHECK (population > 0) or CHECK (longitude BETWEEN -180 AND 180)
- NULL values allowed? : Country: name NOT NULL
- Definition of key/uniqueness constraints:
  Country: code PRIMARY KEY or name UNIQUE

Integrity constraints (Cont'd)

Constraints on several attributes are given separately as "table constraints":

```
CREATE TABLE <table>
   (<column definitions>,
    <table-constraint>, ... ,<table-constraint>)
```

- table-constraints have a name
- must state which columns are concerned

```
CREATE TABLE City
 ( Name VARCHAR2(35),
   Country VARCHAR2(4),
   Province VARCHAR2(32),
   Population NUMBER CONSTRAINT CityPop CHECK (Population >= 0),
   Longitude NUMBER CONSTRAINT CityLong CHECK (Longitude BETWEEN -180 AND 180),
   Latitude NUMBER CONSTRAINT CityLat CHECK (Latitude BETWEEN -90 AND 90),
   CONSTRAINT CityKey PRIMARY KEY (Name, Country, Province));
```

... for details see "Practical Training SQL".

Integrity constraints (Cont'd)

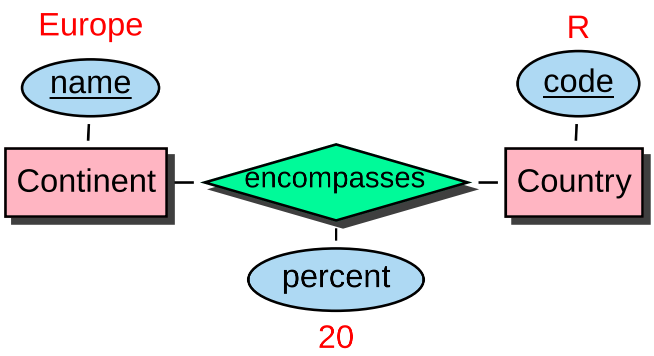
- up to now: only intra-table constraints

General Assertions

- inter-table constraints
  e.g., "sum of inhabitants of provinces equals the population of the country",
  "sum of inhabitants of all cities of a country must be smaller the than population of the country"
- SQL standard: CREATE ASSERTION
- not supported by most systems
- other solution: later

5.1.2 Referential Integrity Constraints

- important part of the schema
- relate foreign keys with their corresponding primary keys:



encompasses		
Country	**Continent**	**Percent**
VARCHAR(4)	VARCHAR(20)	NUMBER
R	Europe	20
R	Asia	80
D	Europe	100
...	...	...

encompasses.country → country.code and
encompasses.continent → continent.name

other examples:

city.country → country.code and
country.(capital,province,code) → city.(name,province,country)

Referential Integrity Constraints: SQL Syntax

- as column constraints (only single-column keys):

```
<column-name> <datatype> REFERENCES <table>(<column>)
```

- as table constraints (also compound keys):

```
CONSTRAINT <name> FOREIGN KEY (<column-list>)
                  REFERENCES <table>(<column-list>)
```

```
CREATE TABLE is_member
 (Country       VARCHAR2(4) REFERENCES Country(Code),
  Organization  VARCHAR2(12) REFERENCES Organization(Abbreviation),
  Type          VARCHAR2(30));
```

```
CREATE TABLE City
( Name VARCHAR2(35),
  Country VARCHAR2(4) REFERENCES Country(Code),
  Province VARCHAR2(32),
  Population NUMBER ..., Longitude NUMBER ..., Latitude NUMBER ...,
  CONSTRAINT CityKey PRIMARY KEY (Name, Country, Province),
  FOREIGN KEY (Country,Province) REFERENCES Province (Country,Name) );
```

5.1.3 Virtual Tables: Views

Views are tables that are not materialized, but *defined* by a query against the database:

```
CREATE VIEW <name> AS <query>
```

```
CREATE OR REPLACE VIEW symm_borders AS
SELECT * FROM borders
UNION
SELECT Country2, Country1, Length FROM borders;
```

```
SELECT country2
FROM symm_borders
WHERE country1='D';
```

- classical views: the content of a view is always computed when it is queried.
- *Materialized Views*: view is materialized and automatically maintained
  → *view maintenance problem*: when a base table changes, what modifications have to be applied to which views?

5.2 SQL: Data Manipulation Language

... everything is based on the structure of the SELECT-FROM-WHERE clause:

- Deletions:

  ```
  DELETE FROM ... WHERE ...
  ```

- Updates:

  ```
  UPDATE <table>
  SET <attribute> = <value>, ..., <attribute> = <value>
  WHERE ...
  ```

  value can be a subquery (also a correlated one)

- Insertions:

  ```
  INSERT INTO <table> VALUES (...)
  INSERT INTO <table> (SELECT ... FROM ... WHERE ...)
  ```

5.3 Beyond Relational Completeness

- The Relational Algebra and SQL are only *relationally complete*.
- can e.g. not compute the transitive closure of a relation
- applications require a more complex behavior:
  - SQL als the "core query language"
  - with something around it ...

Making SQL Turing-complete

- embedded SQL in C/Pascal:

  `EXEC SQL SELECT ... FROM ... WHERE ...`

  embedded into Java: JDBC (Java Database Connectivity)
- SQL-92: Procedural Extensions to SQL:
  - CREATE procedures and functions as compiled things *inside* the database
  - standardized concepts, but product-specific syntax
  - basic programming constructs of a "typical" Turing-complete language:
    Variables, BEGIN ... END, IF ... THEN ... ELSIF ..., WHILE ... LOOP ..., FOR ... LOOP
  - SQL can be used inside PL/SQL statements

"Impedance Mismatch" between DB and Programming Languages

(cf. Slide 3)

Set-oriented (relations) vs. value-oriented (variables)

- how to handle the result of a query in C/Pascal/Java?

Iterators (common programming pattern for all kinds of collections)

- explicit:
  - new/init(<query>)/open()
  - first(), next(), isempty()
  - fetch() (into a record/tuple variable)
- implicit (PL/SQL's "Cursor FOR LOOP"):

```
FOR <record-variable> IN <query>
LOOP
   do something with <record-variable>
END LOOP;
```

... for details see "Practical Training SQL".

5.4 Integrity Maintenance

- if a tuple is changed/inserted/deleted it is immediately checked whether all constraints in the current database state are satisfied afterwards.
  Otherwise the operation is rejected.
- if a constraint is defined/enabled, it is immediately checked whether it is satisfied by the current database state.
  Otherwise the operation is rejected.

Any further possibilities?

Integrity Maintenance (Cont'd): referential integrity

Consider again country - organization - is member:

is_member.organization → organization.abbrev
is_member.country → country.code

- deletion of a membership entry: no problem
- deletion of a country: any membership entries for it are now "dangling"

⇒ remove them!

Referential Actions

FOREIGN KEY is_member(country) REFERENCES country(code) ON DELETE CASCADE

- ON DELETE CASCADE: delete referencing tuple
- ON DELETE RESTRICT: referenced tuple cannot be deleted
- ON DELETE NO ACTION: referenced tuple can be deleted if the same transaction also deletes the referencing tuple
- ON DELETE SET NULL: foreign key of referencing tuple is set to NULL
- ON DELETE SET DEFAULT: foreign key of referencing tuple is set to a default value
- same for ON UPDATE

Referential Actions

Country

Name	Code	Capital	Province
Germany	D	Berlin	Berlin
United States	USA	Washington	Distr. Columbia
...	...	...	...

CASCADE

NO ACTION

City

Name	Country	Province
Berlin	D	Berlin
Washington	USA	Distr. Columbia
...	...	...

```
1.    DELETE FROM City
      WHERE Name='Berlin';

2.    DELETE FROM Country
      WHERE Name='Germany';

3.    UPDATE Country
      SET code='DE'
      WHERE code='D';
```

Referential Actions: Problems

Country

Name	Code	Capital	Province
Germany	D	Berlin	Berlin
United States	US	Washington	Distr.Col.
...	...	...	...

CASCADE

Province

Name	Country	Capital
Berlin	D	Berlin
Distr.Col.	US	Washington
...	...	...

SET NULL

CASCADE

City

Name	Country	Province
Berlin	D	B
Washington	USA	Distr.Col.
...	...	...

```
DELETE FROM Country
  WHERE Code='D'
```

... ambiguous semantics!

see `http://dbis.informatik.uni-goettingen.de/RefInt`.

... active behavior/reaction on events!

5.5 Active Databases/Triggers

- reacting on an event
  - external event/signal
  - internal event: modification/insertion/deletion
  - internal event: time
- if a condition is satisfied
- then do something/execute an action

ECA: Event-Condition-Action rules

ECA-Rules

Consider database updates only: one or more tuples of a table are changed.

- Granularity:
  - execute action once for "all updates together" (e.g., afterwards, update a sum)
  - execute action for each changed tuple (e.g. cascading update)
- Timepoint:
  - after execution of original update
  - before execution of original update
  - instead of original update
- Actions:
  - can read the before- and after value of the updated tuple
  - read and write other tables

Triggers

The SQL standard provides "Triggers" for implementation of ECA rules:

`CREATE TRIGGER`

- specify event:
  `ON {DELETE | UPDATE | INSERT} ON <table>`
- specify condition `WHEN <condition>`
- specify granularity `FOR EACH STATEMENT | ROW`
- specify action

Actions are programmed using the above-mentioned procedural extensions to SQL.

Applications

- implement application-specific *business rules*
- integrity maintenance
- monitoring of assertions

... for details see "Practical Training SQL".