

Chapter 5

Relational Databases and SQL: Further Issues

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

213

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

214

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,
    Latitude      NUMBER,
    Longitude     NUMBER );
```

215

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`

216

Integrity constraints (Cont'd)

Multi-attribute constraints 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;
- e.g. multi-column keys and foreign keys.

```
CREATE TABLE City
```

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

... for details see “Practical Training SQL”.

217

Integrity constraints (Cont'd)

- up to now: only intra-table constraints

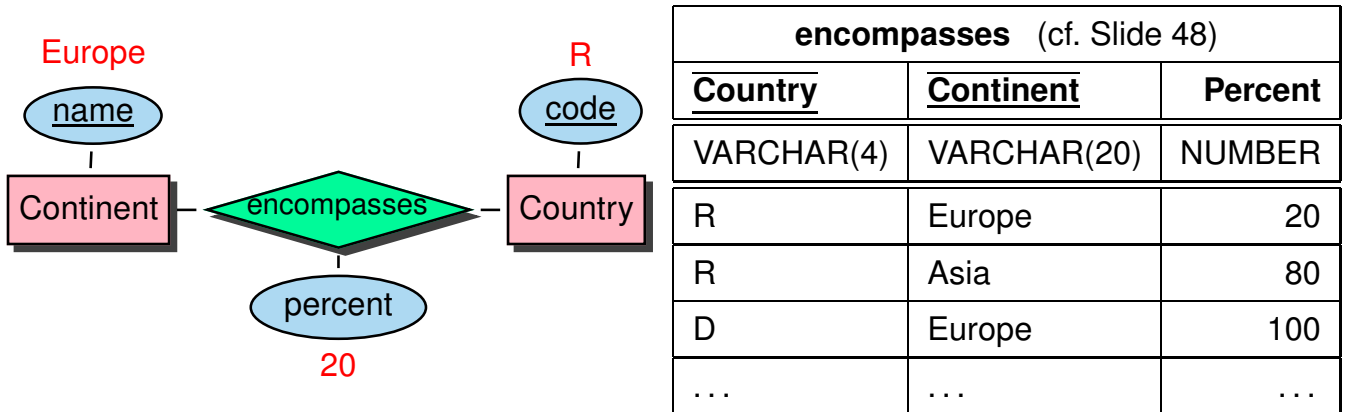
General Assertions

- 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 than population of the country”
- SQL standard: CREATE ASSERTION
- not supported by most systems
- other solution: later

218

5.1.2 Referential Integrity Constraints

- important part of the schema; especially for tables corresponding to relationship types;
- relate **foreign keys** with their corresponding **primary keys**:



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

Tables corresponding to entity types have foreign keys that correspond to 1:n relationships:

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

219

Referential Integrity Constraints: SQL Syntax

- as column constraints (only single-column foreign keys):
[CONSTRAINT <name>] <column-name> <datatype> REFERENCES <table>(<column>)
- as table constraints (also compound foreign keys):
[CONSTRAINT <name>] FOREIGN KEY (<column-list>)
REFERENCES <table>(<column-list>)

```
CREATE TABLE encompasses
  (Country      VARCHAR2(4) REFERENCES Country(Code),
   Continent    VARCHAR2(12) REFERENCES Continent(Name),
   percent      NUMBER CHECK (0 < percent <= 100),
   PRIMARY KEY (Country, Continent));

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

220

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?

221

5.2 SQL: Data Manipulation Language

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

- Deletions:

```
DELETE FROM <table> WHERE ...
```

- specifies in which table to delete,
- where-clause can contain arbitrary subqueries to other tables

- Updates:

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

- specifies in which table to update,
- value can be a subquery (also a correlated one)

- Insertions:

```
INSERT INTO <table> VALUES (<const1>, ..., <constn>)
INSERT INTO <table> (SELECT ... FROM ... WHERE ...)
```

- where the <const_i> are constants (strings, numbers, dates, ...).

222

5.3 SQL: The DATE Datatype ... and Customization

- many applications in business and administration use dates
- computations on dates (e.g., “last of the third month after ...”, “number of days between”)

⇒ SQL provides comprehensive datatypes DATE, TIME, TIMESTAMP

A More General View I: Datatypes

DATE etc. are just some (important and typical) examples of *built-in* datatypes

- specific operators (and behavior, cf. the XMLTYPE datatype in the SQLX standard)
- handled via one or more lexical representations as strings

A MORE GENERAL VIEW II: INTERNATIONALIZATION AND CUSTOMIZATION

Database systems are used anywhere in the world (like most software), *and* their contents is exchanged all over the world

- people use different languages (e.g. for error messages!)
- people use different representations
 - even for numbers: 3,1415 vs. 1.000.000 (german), 3.14 vs. 1,000,000 (anywhere else)
 - for dates: '31.12.2007', '12/31/2007' or '12-31-2007' (USA), '01-JAN-2003' etc., '01 Janeiro 2003' even language dependent.

SQL: INTERNATIONALIZATION AND CUSTOMIZATION

This issue is handled syntactically differently (but using the same idea) between different products.

Oracle: Natural Language Support

NLS_LANG (language and localization issues in general), NLS_NUMERIC_CHARACTERS (decimal point/dezimalkomma) and NLS_DATE_FORMAT (date format), NLS_SORT (sorting order)

- ALTER SESSION SET NLS_LANGUAGE = 'Language Territory.CharacterSet';
Language: error messages, etc, Territory: more detailed formats (America/Canada/UK) including default for decimal point and date format.
`ALTER SESSION SET NLS_LANGUAGE = 'portuguese'`
- ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ',.'; (german style),
ALTER SESSION SET NLS_NUMERIC_CHARACTERS = '.,'; (english style),
- ALTER SESSION SET NLS_DATE_FORMAT = 'string-pattern', e.g. 'DD.MM.YYYY', 'DD-MON-YY', 'DD hh:mm:ss'

225

SQL: Internationalization and Customization

Then, e.g., INSERT INTO Politics VALUES('D','18.01.1871','federal republic') is correctly interpreted. In the output, DATE values are always represented in the currently specified format.

⇒ SQL provides comprehensive datatypes DATE, TIME, TIMESTAMP

- semantics: year/month/date/hour/minute/second
timestamp: additionally fractions of seconds as decimal
(Oracle: only DATE and TIMESTAMP)
built-in calendar knows about length of months, leap years etc.
- operators on date and time:
 - `date + days`
 - `MONTHS_BETWEEN(date1, date2), ADD_MONTHS(date, n), LAST_DAY(date)`
 - `SYSDATE`

to_char(string,pattern) and to_date(string,pattern) functions

```
SELECT to_char(independence,'MM/DD/YYYY') from Politics; -- 01/18/1871
SELECT to_char(independence,'DAY') from Politics; -- wednesday
SELECT to_date('25-FEB-2012','DD-MON-YYYY')+5 from dual; -- 01-MAR-12
```

226

The DATE Datatype: Example

```
CREATE TABLE Politics
( Country VARCHAR2(4),
  Independence DATE,
  Government VARCHAR2(120));
ALTER SESSION SET NLS_DATE_FORMAT = 'DD MM YYYY';
INSERT INTO politics VALUES
('B','04 10 1830','constitutional monarchy');
```

All countries that have been founded between 1200 und 1600:

```
SELECT Country, Independence
FROM Politics
WHERE Independence BETWEEN
'01 01 1200' AND '31 12 1599'
ORDER BY Independence;
```

Country	Independence
THA	01 01 1238
MC	01 01 1419
E	01 01 1492
NL	01 01 1579

227

5.4 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 ...

228

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

229

“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”.

230

5.5 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?

231

Integrity Maintenance (Cont'd): referential integrity

Consider again country - organization - is member:

`isMember.organization` → `organization.abbreviation`
`isMember.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 `isMember(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

232

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';

233

Referential Actions: Problems

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

CASCADE

SET NULL

CASCADE

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

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>.

234

... active behavior/reaction on events!

5.6 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

235

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

236

Triggers

The SQL standard provides “Triggers” for implementation of ECA rules:

```
CREATE TRIGGER
```

- specify event:
ON {DELETE | UPDATE | INSERT} OF <table> <pl/sql-block>
- specify condition: WHEN <condition>
- specify granularity: FOR EACH STATEMENT | ROW
- specify action by pl/sql-block.

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

Applications

- implementation of application-specific *business rules*,
- integrity maintenance,
- monitoring of assertions.

... for details see “Practical Training SQL”.