3.2.4 Enforcing constraints

- Constraints → SQL definition of the schema
- Up to now: primary Key, foreign key, NOT NULL

- Different kinds of integrity constraints
  - value constraints on attributes
  - cardinalities
  - semantic constraints
  - referential constraints

- SQL-DDL:
  - Column constraint: Specify constraint as part of column definition
  - Table constraint: More than one row involved, specify constraint after the column definitions
Constraints may be violated when DB is changed (update, insert, delete)

⇒ Exception

ORA-02291: integrity constraint (SYS_C0067174) violated - parent key not found

Constraint name (optional):

    CONSTRAINT <name> <def>

Advantage: error message shows violated constraint in a readable way

ORA-02291: integrity constraint
    (FK_Dep.SYS_C0067174) violated - parent key not found
PRIMARY KEY
- Only once per table
- Not required, but omission is bad style
  May be column constraint (single attribute) or multicolumn constraint

NOT NULL
- Simplest constraint on attribute values, column constraint

Default values

<attributeName> <attributeType> DEFAULT <value>
e.g. ... population INTEGER DEFAULT 0

this is not: NULL
UNIQUE
• Column contains only unique values
• Left over from SQL-89 (no primary key constraint)
• Should be used for candidate keys
• Column constraint or table constraint

CHECK Clause
Enumeration:
    CHECK (VALUES IN ('X', 'Y', 'Z'))
Interval restriction:
    CHECK (population >= 0),
    CHECK (population < 40000000)
equivalent to
    CHECK (population >= 0 AND population < ...

Multicolumn constraints

CREATE TABLE Accounts (  
... amount DECIMAL(9,2),  
  credit DECIMAL(7,2),...,  
CONSTRAINT accountIsPos  
  CHECK amount + credit > 0 )

General constraint syntax

for column (except NOT NULL) and table constraints

CREATE TABLE <tab> (<listOfColumnSpecs>  
  [[CONSTRAINT <constraintName>]]  
  <constraint on columns or table>]0..n  
)

Constraint may be UNIQUE / PRIMARY KEY, CHECK, REFERENCES
Enforcing cardinalities

CREATE TABLE Country
(name VARCHAR(32) NOT NULL, C_ID VARCHAR(4) PRIMARY KEY, population INT, area INT, GNP INT, capital VARCHAR(25) NOT NULL, region_name VARCHAR(30) NOT NULL, ...)

Since every country has a capital
CREATE TABLE Region
(name VARCHAR(30) ,
C_ID VARCHAR(4) NOT NULL,
area Int,
population Int,
capital VARCHAR(25),
CONSTRAINT region_pk
    PRIMARY KEY (name,C_ID),
CONSTRAINT
    fk_ctry FOREIGN KEY (C_ID)
    REFERENCES Country
);

CREATE TABLE City
(name VARCHAR(25) NOT NULL,
C_ID VARCHAR(4),
reg_name VARCHAR(30),
population INT,
longitude NUMERIC(5,2),
latitude NUMERIC(5,2),
CONSTRAINT city_pk
    PRIMARY KEY
    (name,reg_name,C_ID),
CONSTRAINT region_country_fk
    FOREIGN KEY (R_ID,C_ID)
    REFERENCES Region
);

ALTER TABLE Country
    ADD CONSTRAINT fk_capital FOREIGN KEY
    (name, R_ID) REFERENCES City;

Constraint on Country
assumes table City
Preserving referential integrity

Row and primary key deleted, what to do with foreign keys?
Do nothing: exception
Define actions on referenced tables:

ON DELETE CASCADE
delete all referenced tuples
if a department disappears, all referenced employees are deleted (??)
ON DELETE SET NULL
ON DELETE DEFAULT
ON UPDATE CASCADE // not in Oracle!
update key in referencing table
e.g. new department name, propagate it to table with FK
ON UPDATE SET NULL
ON UPDATE SET DEFAULT
Circular relationships

Example

Table must be created in order to be referenced

How to define circular constraints?

*Specify constraints after table definition*

- Define tables without constraints.
- Use ALTER TABLE to define a constraint a posteriori
Circular constraint

ALTER TABLE Person
ADD CONSTRAINT birthPlaceReference
FOREIGN KEY (birthplace)
REFERENCES city(id);

Which constraint is still missing?

ALTER TABLE City
MODIFY COLUMN mayor NOT NULL
FOREIGN KEY;
3.2.5 Deferred constraints

The Chicken-Egg problem

CREATE TABLE chicken(cID INT PRIMARY KEY, 
                     eID INT);

CREATE TABLE egg(eID INT PRIMARY KEY, 
                  cID INT);

ALTER TABLE chicken
    ADD CONSTRAINT chickenREFegg
    FOREIGN KEY (eID) REFERENCES egg(eID);

ALTER TABLE egg
    ADD CONSTRAINT eggREFchicken
    FOREIGN KEY (cID) REFERENCES chicken(cID)

What happens if an egg / chicken is inserted?
Deferred constraints

Insertion violates foreign key constraint

```
INSERT INTO chicken VALUES(1, 2);
```
ORA-02291: integrity constraint
(chickenREFegg.SYS_C0067174) violated - parent key not found

```
INSERT INTO egg VALUES(2, 1);
```
ORA-02291: integrity constraint
(eggREFchicken.SYS_C0067174) violated - parent key not found

Defer constraint checking!

```
ALTER TABLE chicken
ADD CONSTRAINT chickenREFegg
FOREIGN KEY (eID) REFERENCES egg(eID)
INITIALLY DEFERRED DEFERRABLE;
```
Deferred constraints and transactions

Deferred constraints checked at the end of a transaction (*)

```
INSERT INTO chicken VALUES(1, 2);
-- constraint not checked here
INSERT INTO egg VALUES(2, 1);
COMMIT;  -- but here
```

Variants

```
INITIALLY DEFERRED DEFERRABLE
INITIALLY IMMEDIATE DEFERRABLE
SET CONSTRAINT <name>
    [DEFERED|IMMEDIATE]
```

allow checking at arbitrary times

(*) Transaction: unit of work consisting of one or more operations on the DB

© HS-2010
3.3 Assertions and Triggers

**Def.:** An **Assertions** is an integrity constraint defined independently from table definitions.

Similar to CHECK table constraints:
when evaluated to FALSE: exception

- **Semantics**
  Table assigned constraints always hold for empty tables

```
CREATE ASSERTION atLeastOneRegion
  CHECK (--always at least one region for each
          -- country in region table
          SELECT ... )
```

Most current DBS do not support sophisticated constraints, e.g. table independent assertions ...
Assertions and triggers  SQL / DDL

**Trigger**

\((<\text{predicate}>, <\text{action}>))\)-rule

Semantics:
- if \(<\text{predicate}>)\ is true before | after DB state is changed
- \(<\text{action}>)\ is performed

\[
\text{CREATE TRIGGER salaryCheck }
\text{AFTER INSERT ON Employee }
\text{REFERENCING NEW ROW AS c }
\text{FOR EACH ROW WHEN }
\text{EXISTS (SELECT * FROM Employee e }
\text{WHERE c.boss=e.emp# AND }
\text{e.salary < c.salary) }
\text{<do something>; -- e.g. print warning }
\text{-- may be expressed simpler }
\]
**Trigger example**

If `parts_on_hand` in inventory table too low: reorder!

### Triggering Statement

AFTER UPDATE OF `parts_on_hand` ON `inventory`  

### Trigger Restriction

WHEN `(new.parts_on_hand < new.reorder_point)`

### Triggered Action

```
FOR EACH ROW
DECLARE  
    NUMBER X;  
BEGIN  
    SELECT COUNT(*) INTO X  
    FROM pending_orders  
    WHERE part_no=:new.part_no;  

    IF X = 0 THEN  
        INSERT INTO pending_orders  
        VALUES (new.part_no, new.reorder_quantity, sysdate);  
    END IF;  
END;
```

© Oracle
Triggering applications

Triggers very **useful for triggering actions outside DB**

Triggering **audit trails**

*If table is changed store an entry about this event in a special place (the audit trail)*

Triggering an **application program**

*If a customer has ordered a book in the online shop and she has a non-NUL email address, send a mail!*
Trigger events

Trigger events are database events:

- INSERT, UPDATE, or DELETE statement on some table (*)
- Any creation or altering of schema objects
- A database startup or instance shutdown
- A specific error message or any error message
- A user logon or logoff

Action is performed BEFORE or AFTER the event happens or INSTEAD OF some event.

No external events ("msg arrives").

(*) also on some views
Standard Query Language: Standards

SQL-92 compliance levels:

1. Entry SQL: basically SQL-89, essential
2. Intermediate SQL,
3. Full SQL

No implementation of SQL-92 on level 2 or 3

SQL 1999 (SQL3) levels:

- Core SQL: essential for standard compliance
- Additional Features, e.g. object features

---

First standard:
SQL-89

Important:
SQL-92

Core
SQL: 1999

Enhanced
SQL: 1999

Slight
extension:
SQL: 2003

Newest
draft: SQL 2008
Standards in CS

Standards: not “nice to have” but **inevitable**

Heavy influenced by strategies of SW-Industry

All known implementations do **not conform** to every aspect of the standard

Standards may hinder scientific and technical improvement (!)
3.4 Data Types and name spaces in SQL / DDL

Database-name space?  schema name space?

Name structure:
<cat>.<database>.<schema>.<table>.<column>
**Databases and schemas in Postgres**

**DB cluster**

set of databases $\equiv$ catalog / SQL 99

$\downarrow$

**Database**

physically separated set of schemas

$\downarrow$

**Schema**

logical construct; set of "database objects"

$\downarrow$

**Tables, functions, triggers, .....**

Namespace: `database.schema.table`

without effect in Postgres
CREATE SCHEMA <schemaName>

e.g CREATE SCHEMA Mondial
creates a namespace, in which relations (tables) have unambiguous names

Proposed by SQL-2, but no DBS supports the full naming scheme

Only <table>.<column> names are supported by all systems, confusing terminology in many systems
Name spaces

**Oracle:**

Database = set of physical storage areas
("tablespaces")

Name of **schema** = dbsUsername,
all objects may be prefixed with <dbUsername>

**MySQL:**

Database = **directory in File system**
where data reside

**Schema** not defined in MySQL
3.4.2 SQL Data types

Primitive attribute (column) types

- Base types of the SQL and/or DB system
- No constructed types contradict “first normal form“ – introduced by SQL99
- Types for numbers, characters, strings, date / time, Binary objects

Numeric datatypes in SQL-2 - the first standard

- `NUMERIC (p,s)` exact number, basically same
- `DECIMAL (p,s)` as `DECIMAL`
- `INTEGER` alias: `INT`
- `SMALLINT`
- `FLOAT (p,s)` approximate number
- `REAL` implementation dependent precision
- `DOUBLE PRECISION`
SQL Built-in types

More datatypes in SQL-2: **Character** etc

Literal

CHARACTER [(n)]  CHAR
   // fixed length character string
CHARACTER VARYING (n)
   VARCHAR (n) 'Hello SQL'
   // variable length string, n=maximum
NATIONAL CHARACTER (n)  |  NCHAR (n)
NCHAR VARYING (n)

BIT [(n)], BIT VARYING, BOOLEAN

DATE  DATE '2001-5-2'
TIME  TIME '01:00:05.011'
TIMESTAMP composed of year, month, day,
   hour, minute, second
INTERVAL FirstUnitofTime  [LastUnitofTime]_{0}^{\max}

   e.g. '1 day 12 hours 59 min 10 sec'
Syntax diagram for ANSI / SQL-2 character data type
“Large Objects”

Large Character / Binary Objects since SQL 1999

Restricted, implementation defined restriction of maximum character string length
Char(n) / VARCHAR(n), typically 4000 Bytes

CHARACTER LARGE OBJECT | CLOB
NATIONAL CHARACTER LARGE OBJECT | NCLOB
BINARY LARGE OBJECT | BLOB

Typically up to 2 GB or even more.
Useful for images, videos, …

No blobs in Postgres ... but binary data type
and arbitrary long 'text' data type.
Postgres specific data types

**Net specific**

- **macaddr**  
  MAC address

- **inet**  
  IPV4 / V6 address

**Geometric types**

- **point**  
- **lseg**  
  line segment

- **path**  
  closed or open path

- **polygon, box**

- **circle**

**Miscellaneous**

- **serial**  
  autoincremented 32-Bit-Integer

**Constructed types**

- arrays and more....
<table>
<thead>
<tr>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2(size)</td>
<td>Variable-length character data</td>
</tr>
<tr>
<td>CHAR(size)</td>
<td>Fixed-length character data</td>
</tr>
<tr>
<td>NUMBER(p,s)</td>
<td>Variable-length numeric data</td>
</tr>
<tr>
<td>DATE</td>
<td>Date and time values</td>
</tr>
<tr>
<td>LONG</td>
<td>Variable-length character data up to 2 gigabytes</td>
</tr>
<tr>
<td>CLOB</td>
<td>Single-byte character data up to 4 gigabytes</td>
</tr>
<tr>
<td>RAW(n), LONG RAW</td>
<td>Raw binary data (up to 2 KB</td>
</tr>
<tr>
<td>BLOB</td>
<td>Binary data up to 4 gigabytes</td>
</tr>
<tr>
<td>e.g. X'49FE'</td>
<td></td>
</tr>
<tr>
<td>BFILE</td>
<td>Binary data stored in an external file; up to 4 gigabytes</td>
</tr>
</tbody>
</table>
### Differences

**Numeric types in different DBS:**

**Oracle**

\[
\text{NUMBER}(p, s) \quad \text{Variable-length numeric data}
\]

**MySQL:**

- TINYINT[(M)]
- SMALLINT[(M)]
- MEDIUMINT[(M)]
- INT[(M)]
- BIGINT[(M)]
- FLOAT(precision)
- FLOAT[(M,D)]
- DOUBLE[(M,D)]
- DOUBLE PRECISION[(M,D)]
- REAL[(M,D)]
- DECIMAL[(M[,D])]  
- NUMERIC[(M[,D])]  

Many differences from standard

- always use standard types
- Makes database less dependent from the database system vendor
Generated columns (SQL 2003)

Extension in SQL 2003: Generated columns
"Identity column" using internal sequence:

```sql
CREATE TABLE employees (  
  EMP_ID INTEGER GENERATED ALWAYS AS IDENTITY  
  START WITH 100  
  INCREMENT 1...  
  ...)
```

Instance of the more general concept
"Generated column"
Generated columns (SQL 2003)

Any number of columns of a base table can be designated as **generated columns**.

Each generated column must be associated with a **scalar expression**. All column references in such expressions must be to **columns of the base table** containing that generated column.

**Values** for generated columns are **computed and assigned automatically** every time a row is inserted into such tables.
CREATE TABLE EMPLOYEES ( 
    EMP_ID INTEGER,
    SALARY DECIMAL(7,2),
    BONUS DECIMAL(7,2),
    TOTAL_COMP GENERATED ALWAYS AS ( 
        SALARY + BONUS 
    ),
    HR_CLERK GENERATED ALWAYS AS ( 
        CURRENT_USER 
    )
)
SQL/DDL Domains

Domain = named sets of values and value representation

CREATE DOMAIN <domainName> <TypeDef>

CREATE DOMAIN Money DECIMAL (10,2)

not really representation independent, but useful in order to avoid semantically meaningless operations, e.g. comparing money with length attributes

Not supported in most Systems (neither Oracle nor MySQL, exception Postgres, SAP-DB)
**SQL / DDL:** Type definitions (user defined type)

Distinct type:
- Similar to domain definition
- Strong typing

**Syntax:**
```sql
CREATE TYPE <typeName> as <typeDef>
[FINAL];
```

**Examples:**
```sql
CREATE TYPE Euro AS DECIMAL(8,2) FINAL;
CREATE TYPE Mark AS DECIMAL(8,2) FINAL;
CREATE Type Address AS(
  street varchar (25),
  zipCode Integer,....);
```
3.5 Metadata management

Meta data

All definitions and other data on data are called metadata

Stored in system data structures

Data structures for metadata called the catalogue or data dictionary in particular when used for more than one DB

In most systems stored as tables

Makes metadata first class:

may be queried und modified in the same way as the data tables

Select <Table_Name> from User_Tables;
Metadata management: example

Querying the catalog using SQL (ORACLE)

```
SQL> SELECT constraint_name, search_condition, delete_rule
        FROM user_constraints
        WHERE table_name = 'Region';
```

<table>
<thead>
<tr>
<th>CONSTRAINT_NAME</th>
<th>SEARCH_CONDITION</th>
<th>DELETE_RULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_C001360</td>
<td>TITLE IS NOT NULL</td>
<td></td>
</tr>
<tr>
<td>PLAUSIBLE_YEAR</td>
<td>year &gt; TO_DATE('01.01.1900','DD.MM.YYYY')</td>
<td></td>
</tr>
<tr>
<td>ALLOWEDPRICE</td>
<td>pricePDay &gt;= 0) AND (pricePDay &lt; 100.0)</td>
<td></td>
</tr>
<tr>
<td>SYS_C001363</td>
<td>TAPE_MOVIE</td>
<td>CASCADE</td>
</tr>
</tbody>
</table>

No standard for metadata management! completely different in Postgres!
Postgres Information Schema

All kinds of metadata on schemas of db
.. tables, columns, ... sql_features (implemented)
e.g....

```
SELECT * FROM information_schema.Columns WHERE table_schema = 'video'
ORDER BY table_name
```

geo;video;rental;tape_id;1;;NO;integer;;;32... pg_catalog;int4;;...;
geo;video;rental;from_date;3;;NO;date;;; ... pg_catalog;date;;...;
geo;video;rental;until_date;4;;YES;date;;;;pg_catalog;date;;...;...
.....
Virtual tables: views

More SQL Schema Definition Statements

**CREATE TABLE** defines base tables

**Def.:** A **view** is a virtual table, has a definition, but no extent, definition is executed when table is accessed

```
CREATE VIEW <name> AS <SQL-select>
```

* e.g. `CREATE VIEW GNP_Ratio
  SELECT c_id, name, GNP, GNP/popul
  FROM Country;
`

May be used as ordinary tables for reads, updates are much more involved
View and Materialized views

View: a construct for defining virtual tables

Views are used in statements just as ordinary tables

SELECT name, age FROM myView WHERE…

But updates?

Materialized view

Result auf executing view defining expression is a temporary table. Performance improvement!

Makes sense if basically read Operations on view
3.6 Modifying and deleting definitions

ALTER TABLE

ALTER TABLE<tableName><redefinition>;

Add a column:

ALTER TABLE City
ADD (mayor CHAR(20));

Modify type:

ALTER TABLE City
MODIFY (mayor CHAR(30));

Many more variants of ALTER statement
see manual
Deletion of schema elements

Table delete

Delete table only if not referenced,
drop references first!

DROP TABLE <tableName> restrict;

Delete table and references:

DROP TABLE <tableName> CASCADE;

DROP TABLE <tableName> [, <tableName>]0..n
constraints;

Data, metadata and indexes are deleted.

Delete from <table> only deletes data
Oracle

PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY, REFERENCES, CHECK supported, uses sequence objects

Postgres

very similar to ORACLE (SQL99), SERIAL type as an abbreviation of sequence objects

MySQL (V3.x)

PRIMARY KEY, NOT NULL, UNIQUE supported
FOREIGN KEY, REFERENCES, CHECK accepted (for compatibility) but not supported.

Improved in V 5.X
Summary

Standard Query Language (SQL)
- Data definition language (DDL)
- Data manipulation language (DML)
In almost all current DBMS
All SQL implementations differ from standard
Core SQL99 is basically supported by high-end DBS

Important terms and concepts
- Data types
- Create, change, delete tables
- Referential integrity
- Integrity constraints
- TRIGGERS