



Politecnico  
di Torino



# Table management

---

The SQL Language

# The SQL Language: Table management

---

- Creating a table
- Altering a table
- Deleting a table
- The data dictionary
- Data integrity

# Creating tables

---

Table management

# CREATE

```
CREATE TABLE TableName  
(AttributeName Domain [DefaultValue ] [Constraints]  
{ , AttributeName Domain [DefaultValue ] [Constraints]}  
OtherConstraints  
);
```

- It allows
  - defining all attributes (i.e., columns) of the table
  - defining integrity constraints on the table data
- Domain
  - it defines the data type of an attribute
  - predefined domains of the SQL language (elementary domains)
  - user-defined domains (starting from the predefined domains)
- Constraints
  - integrity constraints for the specific attribute
- OtherConstraints
  - general integrity constraints on the table

# Domain definitions

---

- *Default*
  - it allows specifying a default value for the attribute
- *GenericValue*
  - a value compatible with the attribute domain
- \*USER
  - user identifier
- NULL
  - standard default value

**DEFAULT** < *GenericValue* | USER | CURRENT\_USER |  
SESSION\_USER | SYSTEM\_USER | NULL >

# Elementary domains

Data type	SQL
Text	CHARACTER [VARYING] [(Length)] [CHARACTER SET CharacterFamilyName] VARCHAR (Length) TEXT
Binary	BIT [VARYING] [(Length)] BLOB BINARY
Boolean	BOOLEAN
Integer numbers	INTEGER SMALLINT BIGINT
Real numbers	NUMERIC [( Precision, Scale )] DECIMAL [( Precision, Scale )] FLOAT [(n)] REAL DOUBLE PRECISION

# Elementary domains: real numbers

---

- Exact representations
  - NUMERIC and DECIMAL are base-ten numbers
  - Precision: total number of digits
  - Scale: number of decimal places
  - Example: for number 123.45 precision is 5, scale is 2
- Approximate numeric domains
  - FLOAT (n): n specifies precision
  - it is the number of bits used to store the mantissa of a floating point number represented in scientific notation
  - it is a value ranging from 1 to 53 (the default value is 53)

# Domini elementari

Tipologia di dato	SQL
Time	TIMESTAMP [(Precision)] [WITH TIME ZONE] DATE DATETIME
JSON	JSON
Spatial	SDO_GEOMETRY GEOMETRY POINT LINESTRING POLYGON



The definition of data types in SQL differs depending on the DBMS used



# Domini elementari

Tipologia di dato	SQL
Time	TIMESTAMP [(Precision)] [WITH TIME ZONE] DATE DATETIME
JSON	JSON
Spatial	POINT POLYGON GEOMETRY GEOGRAPHY GEOMETRY POINT POLYGON GEOMETRY GEOGRAPHY

- Stores the year, the month, the day, the hour, the minutes, the seconds and possibly the fraction of second
- it uses 19 characters, plus the characters needed to represent the precision

YYYY-MM-DD hh:mm:ss:p



The definition of data types in SQL differs depending on the DBMS used

# Defining a domain (1/2)

---

```
CREATE DOMAIN DomainName AS DataType  
    [ DefaultValue ] [ Constraint ]
```

- It defines a new domain that may be used in attribute definitions
  - *DataType* is an elementary domain
- Example

```
CREATE DOMAIN Grade AS SMALLINT  
    DEFAULT NULL  
    CHECK (Grade >= 18 and Grade <=30)
```

# Definition of supplier-product database

- Creating the Supplier Table

```
CREATE TABLE S (  
  SId          CHAR(5),  
  SName       CHAR(20),  
  NEmployees  SMALLINT,  
  City        CHAR(15));
```

- Creating the Supply Table

```
CREATE TABLE SP (  
  SId          CHAR(5),  
  PId          CHAR(6),  
  Qty         INTEGER);
```

- Creating the Product Table

```
CREATE TABLE P (  
  PId          CHAR(6),  
  PName       CHAR(20),  
  Color       CHAR(6),  
  Size        SMALLINT,  
  Store       CHAR(15));
```



The definition of integrity constraints is missing

# Altering tables

---

Table management

# ALTER TABLE

- The following “alterations” are possible
  - adding a new column
  - defining a new default value for an existing column (attribute)
    - for example, replacing a previous default value
  - deleting an existing column (attribute)
  - defining a new integrity constraint
  - deleting an existing integrity constraint

```
ALTER TABLE TableName
< ADD COLUMN <Attribute-Definition> |
  ALTER COLUMN AttributeName
    < SET <Default-Value-Definition> | DROP DEFAULT > |
  DROP COLUMN AttributeName
    < CASCADE | RESTRICT > |
  ADD CONSTRAINT [ConstraintName]
    < Unique-Constraint-Definition > |
    < Integrity-Constraint-Definition > |
    < Check-Constraint-Definition > |
  DROP CONSTRAINT [ConstraintName]
    < CASCADE | RESTRICT >
```

- RESTRICT (default option)
  - the element (column or constraint) is not removed if it appears in the definition of some other element
- CASCADE
  - all elements with a dependency on a deleted element will be removed, until there are no unresolved dependencies

# Deleting tables

---

Table management

# DROP TABLE

```
DROP TABLE TableName  
[ RESTRICT | CASCADE];
```

- All rows in the table are deleted along with the table
- RESTRICT
  - the table is not deleted if it appears in the definition of some table, constraint or view
  - default option
- CASCADE
  - if the table appears in the definition of some view, the latter is also deleted

# Data Dictionary

---



# The data dictionary

---

- Metadata are information (data) about data
  - they may be stored in database tables
- The data dictionary contains the metadata of a relational database
  - it contains information about the database objects
  - it is managed directly by the relational DBMS
  - it may be queried by means of SQL commands
- It contains various information
  - descriptions of all database structures (tables, indices, views)
  - SQL stored procedures
  - user privileges
  - statistics
    - on the database tables
    - on the database indices
    - on the database views
    - on the evolution of the database

# Information about tables

---

- For each database table, the data dictionary contains
  - table name and physical structure of the file storing the table
  - name and data type for each attribute
  - name of all indices created on the table
  - integrity constraints

# Data dictionary tables

---

- Data dictionary information is stored in several tables
  - each DBMS uses different names for different tables
- The data dictionary may be queried by means of SQL commands

# The Oracle data dictionary

---

- In Oracle 3 collections of information are defined for the data dictionary
  - **USER\_\***: metadata related to the current user's data
  - **ALL\_\***: metadata related to all users' data
  - **DBA\_\***: metadata about system tables
- **USER\_\*** contains different tables and views, including:
  - **USER\_TABLES** contains metadata to the user tables
  - **USER\_TAB\_STATISTICS** contains statistics computed on the user tables
  - **USER\_TAB\_COL\_STATISTICS** contains statistics computed on user table columns

# Querying the data dictionary no. 1

---

- Show the name of user-defined tables and the number of tuples stored in each table

```
SELECT Table_Name, Num_Rows  
FROM USER_TABLES;
```

R

Table_Name	Num_Rows
S	5
P	6
SP	12

# Querying the data dictionary no.2 (1/2)

- For each attribute in the supplier-product table, show the attribute name, the number of distinct values and the number of tuples with a NULL value

```
SELECT Column_Name, Num_Distinct, Num_Nulls
FROM USER_TAB_COL_STATISTICS
WHERE Table_Name = 'SP'
ORDER BY Column_Name;
```

R

Column_Name	Num_Distinct	Num_Nulls
SId	4	0
PId	6	0
Qty	4	0

# Data Integrity

---

Table management

# Integrity constraints

---

- Data in a database are correct if they satisfy a set of correctness rules
  - rules are called *integrity constraints*
  - example: Qty  $\geq 0$
- Data update operations define a new state for the database, which may not necessarily be correct
- Checking the correctness of a database state may be done
  - by *application procedures*, performing all required checks
  - through the definition of *integrity constraints* on the tables
  - through the definition of *triggers*



# Application procedures

---

Each application includes all required correctness checks

## Pros

- “flexible” approach

## Cons

- checks may be “circumvented” by interacting directly with the DBMS
- a coding error may have significant effects on the database
- the knowledge about integrity constraints is typically “hidden” inside applications

# Table integrity constraints

---

- Integrity constraints are
  - defined in the **CREATE** or **ALTER TABLE** statements
  - stored in the system data dictionary
- Each time data are updated, the DBMS automatically verifies that the constraints are satisfied

# Table integrity constraints

---

## Pros

- *declarative* definition of constraints, whose verification is delegated to the system
  - the data dictionary describes all of the constraints in the system
- unique centralized check point
  - constraint verification may not be circumvented

## Cons

- they may slow down application execution
- it is not possible to define constraints of an arbitrary type
  - example: constraints on aggregated data

# Triggers

---

- Triggers are procedures executed automatically when specific data updates are performed
  - defined through the **CREATE TRIGGER** command
  - stored in the system data dictionary
- When a modification event occurs on data under the trigger's control, the procedure is automatically executed

# Trigger

---

## Pros

- they allow defining complex constraints
  - normally used in combination with constraint definition on the tables
- unique centralized check point
  - constraint verification may not be circumvented

## Cons

- complex
- they may slow down application execution

# Fixing violations

---

- If an application tries to execute an operation that causes a constraint violation, the system may
  - block the operation, causing an error in the application execution
  - execute a compensating action so that a new correct state is reached
    - example: when a supplier is deleted, its supplies are also deleted

# Integrity constraints in SQL-92

---

- The SQL-92 standard introduced the possibility to specify integrity constraints in a declarative way, delegating to the system the verification of their consistency
  - table constraints
    - restrictions on the data allowed in table columns
  - referential integrity constraints
    - manage references among different tables
      - based on the concept of foreign key

# Table Constraints

---

- They are defined on one or more columns of a table
- They are defined in the creation instructions of:
  - Tables
  - Domains
- Type of constraints:
  - Primary key
  - Admissibility of NULL values
  - Uniqueness
  - General tuple constraints
- They are checked after each SQL statement that operates on the table subject to the constraint
  - Entering new data
  - Changing the value of constrained columns
- If a constraint is violated, the SQL statement that caused the violation results in an execution error



# Primary Key

- A primary key is a set of attributes that uniquely identifies rows in a tables
- Only one primary key may be specified for a given table
- Primary key definition
  - composed of a single attribute

*AttributeName Domain PRIMARY KEY*

- composed of one or more attributes

**PRIMARY KEY** (*ListOfAttributes*)

# Primary Key examples

a single attribute

```
CREATE TABLE S (  
    SId          CHAR(5) PRIMARY KEY,  
    SName       CHAR(20),  
    NEmployees  SMALLINT,  
    City        CHAR(15))
```

one or more attributes

```
CREATE TABLE SP (  
    SId          CHAR(5),  
    PId          CHAR(6),  
    Qty         INTEGER,  
    PRIMARY KEY (SId, PId));
```

# Admissibility of the NULL value

- The **NULL** value indicates absence of information
- When a value must always be specified for a given attribute

*AttributeName Domain* **NOT NULL**

- NULL value is not allowed

# NOT NULL: example

---

```
CREATE TABLE S (SId          CHAR(5),  
                 SName       CHAR(20) NOT NULL,  
                 NoEmployees SMALLINT,  
                 City        CHAR(15));
```

# UNIQUE

- An attribute or a set of attributes may not take the same value in different rows of the table
  - for a single attribute

*AttributeName Domain* **UNIQUE**

- for one or more attributes

**UNIQUE** (*ListOfAttributes*)

- It is possible to repeat the **NULL** value (it is seen as a different value in each row)

# Candidate key

---

- A candidate key is a set of attributes that may serve as a primary key
  - it is unique
  - it does not allow the NULL value
- The combination **UNIQUE NOT NULL** defines a candidate key that does not allow null values

*AttributeName Domain* **UNIQUE NOT NULL**

# Unique constraint: example

---

```
CREATE TABLE P (    PId  CHAR(6),
                    PName  CHAR(20) NOT NULL UNIQUE,
                    Color  CHAR(6),
                    Size   SMALLINT,
                    Store   CHAR(15));
```

# General Tuple Constraints

- They allow expressing general conditions on each tuple
  - tuple or domain constraints

*AttributeName Domain CHECK (Condition)*

- Predicates that can be specified in the WHERE clause can be specified as a condition
- The database is correct if the condition is true



# General tuple constraints: example

---

```
CREATE TABLE S (SId          CHAR(5) PRIMARY KEY,  
                SName       CHAR(20) NOT NULL,  
                NoEmployees  SMALLINT  
                CHECK (NoEmployees>0),  
                City         CHAR(15));
```

# Referential Integrity Constraints

- They manage the link between tables by means of the value of attributes
- The foreign key is defined in the **CREATE TABLE** statement of the referencing table

**FOREIGN KEY** (*ListReferencingAttributes* )

**REFERENCES** *TableName* [(*ListReferencedAttributes* )]

- If the referenced attributes have the same name as the referenced attributes, they are not required

# Example: Defining a Foreign Key

---

```
CREATE TABLE SP (  
    SId          CHAR(5),  
    PId          CHAR(6),  
    Qty          INTEGER,  
    PRIMARY KEY (SId, PId),  
    FOREIGN KEY (SId)  
        REFERENCES S(SId),  
    FOREIGN KEY (PId)  
        REFERENCES P(PId));
```

# Politiche di gestione dei vincoli

- Integrity constraints are checked after each SQL command that may cause their violation
- Insert or update operations on the referencing table that violate the constraints are not allowed
- In the CREATE TABLE statement of the referencing table

**FOREIGN KEY** (*ListReferencingAttributes* )  
**REFERENCES**

*TableName* [(*ListReferencedAttributes* )]

[**ON UPDATE**

<**CASCADE** | **SET DEFAULT** | **SET NULL** | **NO ACTION**>]

[**ON DELETE**

<**CASCADE** | **SET DEFAULT** | **SET NULL** | **NO ACTION**>]

- Update or delete operations on the referenced table have the following outcome on the referencing table:
  - **CASCADE**: the update or delete operation is propagated
  - **SET NULL/DEFAULT**: a null or default value is set in the columns for the tuples whose values are no longer present in the referenced table
  - **NO ACTION**: the offending action is not executed

# Example: Product-Supply DB

- table **P**: describes the available products
  - Primary key: PId
  - Product name cannot have NULL or duplicate values
  - The size is always greater than zero
- table **S**: describes suppliers
  - Primary key: Sid
  - Supplier name cannot have NULL or duplicate values
  - The number of employees is always greater than zero
- table **SP**: describes supplies, relating products to the suppliers who supply them
  - Primary key: (SId, PId)
  - Quantity cannot be null and is greater than zero
  - Referential integrity constraints

# Constraint Management: Example 1

---

- SP (referencing table)
  - insert (new tuple PId, SId) -> No
  - update (SId) -> No
  - delete (tuple) -> Ok
- S (referenced table)
  - insert (new tuple) -> Ok
  - update (SId) -> cascaded update (cascade)
  - delete (tuple) -> cascaded update (cascade)  
prevent action (no action)

# SQL Example: Product-Supply DB

```
CREATE TABLE P ( PId          CHAR(6) PRIMARY KEY,
                 PName        CHAR(20) NOT NULL UNIQUE,
                 Color        CHAR(6),
                 Size          SMALLINT
                        CHECK (Size > 0),
                 Store        CHAR(15));

CREATE TABLE S (SId          CHAR(5) PRIMARY KEY,
                 SName        CHAR(20) NOT NULL UNIQUE,
                 NoEmployees  SMALLINT
                        CHECK (NoEmployees>0),
                 City         CHAR(15));

CREATE TABLE SP (SId        CHAR(5),
                 PId        CHAR(6),
                 Qty        INTEGER
                 CHECK (Qty IS NOT NULL and Qty>0),
                 PRIMARY KEY (SId, PId),
                 FOREIGN KEY (SId)
                        REFERENCES S(SId)
                 ON DELETE NO ACTION
                 ON UPDATE CASCADE,
                 FOREIGN KEY (PId)
                        REFERENCES P(PId)
                 ON DELETE NO ACTION
                 ON UPDATE CASCADE);
```

# Constraint Management: Example 2

---

- Employees (EId, EName, City, DId)
- Departments (DId, DName, City)
  
- Employees (referencing table)
  - insert (new tuple) -> No
  - update (DId) -> No
  - delete (tuple) -> Ok
  
- Departments (referenced table)
  - insert (new tuple) -> Ok
  - update (DId) -> cascaded update (cascade)
  - delete (tuple) -> cascaded update (cascade)
    - prevent action (no action)
    - set to unknown value (set null)
    - set to default value (set default)