Unit 2: SQL

2.1. DML: Queries and Data Manipulation

2.2. SQL Exercises (Lab. sessions)

2.3. DDL: Data Definition Language

Unit 2.3. DDL: Data Definition Language

SQL

DDL (Data Definition Language): Creation, modification, and deletion of the components of the relational DB schema.

DCL (Data Control Language): Dynamically changes the database properties

DML (Data Manipulation Language): Queries and database updates.

Unit 2.3. DDL: Data Definition Language

1. Data Definition Language (DDL)
2. **Schema components**
3. Table definition
4. Table modification
5. Table deletion
6. View definition
7. View deletion
8. Authorizations

SQL Commands for defining relational schemas

- create **schema**: gives name to a relational schema and declares the user who is the owner of the schema.
- create **domain**: defines a new data domain.
- create **table**: defines a table, its schema, and its associated constraints.
- create **view**: defines a view or derived relation in the relational schema.
- create **assertion**: defines general integrity constraints.

- **grant**: defines user authorizations for the operations over the DB objects.

All these commands have the opposite operation (**DROP** / **REVOKE**) and modification (**ALTER**).

Schema definition

CREATE SCHEMA [*schema_name*] [**AUTHORIZATION** *user*]
[*list_of_schema_elements*]

A schema element can be any of the following:

- Domain definition.
- Table definition.
- View definition.
- Constraint definition.
- Authorization definition.

Cascade: automatically drops objects (tables, functions, etc.) that are contained in the schema.

Removal of a relational schema definition:

DROP SCHEMA *schema_name* { **RESTRICT** | **CASCADE** };

Unit 2.3. DDL: Data Definition Language

1. Data Definition Language (DDL)
2. Schema components
3. **Table definition**
4. Table modification
5. Table deletion
6. View definition
7. View deletion
8. Authorizations

Create table

CREATE TABLE *table_name*

(*column_definition_list* [*table_constraint_definition_list*]);

Where a ***column_definition*** is done as follows:

column_name { *datatype* | *domain* }

[**DEFAULT** { *literal* | *system_function* | **NULL**}]

[*column_construct_definition_list*]

Constraints over a column

The constraints that can be defined over a **column** are:

- NOT NULL: not null value constraint.
- Constraint definition for single column *PK*, *Uni*, *FK*.
- General constraint definition with the *check* clause.

[**CONSTRAINT** *constraint_name*]
{ **NOT NULL**
| **PRIMARY KEY**
| **UNIQUE**
| **REFERENCES** *table_name* [(*column_names*)]
[**ON DELETE**
{ **CASCADE** | **SET NULL** | **SET DEFAULT** | **NO ACTION** }]
[**ON UPDATE**
{ **CASCADE** | **SET NULL** | **SET DEFAULT** | **NO ACTION** }]
| **CHECK** (*conditional_expression*) }
[*When_check_constraint*]

The operation is not allowed if the PK is violated (default option)

Example

```
CREATE TABLE climb
(climbname VARCHAR2(35) CONSTRAINT PK_climb PRIMARY KEY,
 height NUMBER(4),
 category CHAR(1),
 slope NUMBER(3,2),
 stagenum NUMBER(2) NOT NULL
          CONSTRAINT FK_climb_stage REFERENCES stage (stagenum),
 cnum NUMBER(3)
          CONSTRAINT FK_climb_cyc REFERENCES cyclist (cnum)
);
```

Constraints over a table

The constraints that can be defined over a **table** are:

- Constraint definition for single column PK, Uni, FK.
- General constraint definition with the check clause.

```
[ CONSTRAINT constraint_name ]
    {     UNIQUE ( list_of_column_names )
        | PRIMARY KEY ( list_of_column_names )
        | FOREIGN KEY ( list_of_column_names )
            REFERENCES table_name [( list_of_column_names )]
              [ ON DELETE
                  { CASCADE | SET NULL | SET DEFAULT | NO ACTION }]
              [ ON UPDATE
                  { CASCADE | SET NULL | SET DEFAULT | NO ACTION }]
        | CHECK ( conditional_expression )    }
        [ When_to_check_the_constraint ]
```

Example

```
CREATE TABLE wear
        (cnum NUMBER(3) NOT NULL CONSTRAINT FK_wear_cyc
                REFERENCES cyclist (cnum),
        stagenum NUMBER(2)
            CONSTRAINT FK_wear_stage REFERENCES stage (stagenum),
        code CHAR(3)
            CONSTRAINT FK_wear_jer REFERENCES jersey (code),
        CONSTRAINT PK_wear PRIMARY KEY ( stagenum, code )
);
```

Types of referential integrity

R (FK) → S (UK)

- **Complete (match full):**
  In a tuple of R all the values must have a null value or none of them. In the latter case, there must exist a tuple in S taking the same values for the attributes in UK as the values in the attributes of FK.

- **Partial (match partial):**
  If in a tuple of R one or more attributes of FK do not have a non-null value, then there must exist a tuple in S taking the same values for the attributes of UK as the values in the non-null attributes of FK.

- **Weak (default value.** The clause match is not included):
  If in a tuple of R all the values for the attributes of FK have a non-null value, then there must exist a tuple in S taking the same values for the attributes of UK as the values in the attributes of FK.

This is the only type supported by Oracle

When to check the constraints

[[NOT] DEFERRABLE]

[INITIALLY { IMMEDIATE | DEFERRED }]

- **deferrable** indicates that the constraint state can be modified between **deferred** (evaluated at the end of the transaction) and **immediate** (evaluated after every update of the database).
- **not deferrable** (default option) indicates that the constraint state cannot be modified and then it is assumed to be immediate forever and for every transaction.

For the deferrable option, we can specify how it starts for every transaction:

- INITIALLY IMMEDIATE
- INITIALLY DEFERRED

To change the mode of one or more constraints:

SET CONSTRAINT { *list_of_constraint_names* | **ALL** }
{ **IMMEDIATE** | **DEFERRED** }

Unit 2.3. DDL: Data Definition Language

1. Data Definition Language (DDL)
2. Schema components
3. Table definition
4. **Table modification**
5. Table deletion
6. View definition
7. View deletion
8. Authorizations

4. Table modification

```
ALTER TABLE table_name
   {     ADD ( column_definition)
       | MODIFY [ COLUMN ] ( column_name )
             { DROP DEFAULT |
              SET DEFAULT { string | system_function | NULL} |
              ADD constraint_definition |
              DROP constraint_name }
        | DROP [ COLUMN ]  column_name
        { RESTRICT | CASCADE }
   }
```

Example:

ALTER TABLE cyclist ADD (height NUMBER(3))

5. Table deletion

The SQL instruction to delete one table is

DROP TABLE *table_name* { **CASCADE CONSTRAINTS** }

To delete all the Foreign Keys constraints referencing this table

1. Data Definition Language (DDL)
2. Schema components
3. Table definition
4. Table modification
5. Table deletion
6. **View definition**
7. View deletion
8. Authorizations

6. View definition

- A view is a virtual table which is derived from other tables (base or virtual).
- Can be queried like any other base table.

If not specified, name coincides with the ones returned by the SELECT_statement

CREATE VIEW *view_name*
    [**(***list_of_column_names* **)**]
AS SELECT_statement
[**WITH CHECK OPTION**]

No update or insertion will be allowed if it violates the view definition

Example (Cycling race schema)

We are going to write many queries using stages with climbs.

```
CREATE VIEW stages_with_climbs AS
        SELECT *
        FROM stage
        WHERE stagenum IN (SELECT stagenum FROM climb);
```

Now, we can write queries using this view:

List the km of the longest stage including at least one maintain pass

```
        SELECT MAX(km)
        FROM stages_with_climbs;
```

Updating views

Updates are transferred to the original tables with some limitations. The update can be performed if there is no ambiguity.

A view is not updatable if:

- It contains set operators (UNION, INTERSECT,...).
- It contains the DISTINCT operator
- It contains aggregated functions (SUM, AVG, ..)
- It contains the clause GROUP BY

If the view uses two or more tables, only will be allowed the modifications affecting the table containing the primary key what could be primary key of the view

Updating views

View containing the winners of the stages and the number of won stages.

```
CREATE VIEW Winners AS
        SELECT C.cnum, C.name, COUNT(*) AS Won_stages
        FROM Cyclist C, Stage E
        WHERE C.cnum=E.cnum
        GROUP BY C.cnum, C.name ;
```

Can be updated ?

```
        INSERT INTO Winners VALUES (150, ‘Pepe Pérez’, 5);
```

No, because it contains an aggregated function and a GROUP BY. Each view row comes from one cyclist and a set of stages

7. View deletion

DROP VIEW *view_name* { CASCADE CONSTRAINTS }

To delete all the Foreign Keys constraints referencing this view

Unit 2.3. DDL: Data Definition Language

1. Data Definition Language (DDL)
2. Schema components
3. Table definition
4. Table modification
5. Table deletion
6. View definition
7. View deletion
8. **Authorizations**

Access Control

- Each object that is created in SQL has an owner.

  The owner of one database schema is the owner of all its components.

- The owner is the only person who can perform any operation on the object.

- To give other users access to the object, the owner must explicitly grant them the necessary privileges using the GRANT statement.

GRANT

GRANT

{ **ALL** |

SELECT |

INSERT [(*list_of_columns*] |

DELETE |

UPDATE [(*list_of_columns*)] }

ON *table_name* **TO** { *list_of_users* | **PUBLIC** }

[**WITH GRANT OPTION**]

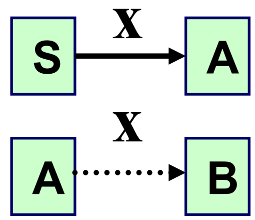
To all the users

Allows the user to pass the privileges to other users

The **REVOKE** statement is used to take away privileges that were granted with the GRANT statement. It has the same syntax than GRANT.

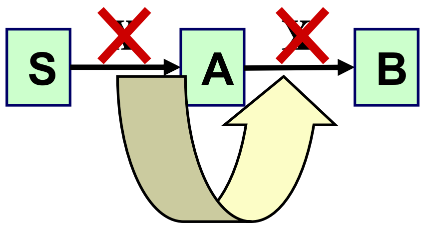
Management of transferrable authorizations

- It is necessary to know the access authorizations of each user (some authorizations will be transferable to other users).

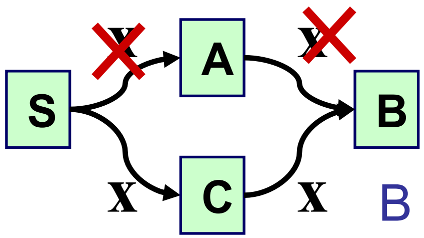


- One authorization can be transferred to other user in mode transferable or not

- When one authorization is revoked:
  - If the authorization was transferable, it is necessary to revoke all the transferred authorizations



- if one user receives more than one authorization, each of the authorizations can be independently revoked.



B maintains the authorization x

Exercise 1

Consider the following schema:

Actor (**act_code**: char(9), **name**: char(40), **age**: integer)

PK: {act_code}   NNV: {name}

Panel_member (**mem_code**: char(9), **name**: char(40), **speciality**: char(15))

PK: {mem_code} NNV: {name}

Role (**role_code**: char(2), **description**: real, **duration**: real, **mem_code** : char(9))

PK: {role_code}   NNV: {description, duration, mem_code}

FK: {mem_code} → Panel_member ON UPDATE RESTRICT

Performance (**role_code**: char(2), **act_code**: char(9), **per_date**: date)

PK: {role_code, act_code} NNV: {per_date}

FK: {role_code} -> Role   ON DELETE CASCADE,
                                                    ON UPDATE NO ACTION

FK: {act_code} -> Actor   ON DELETE NO ACTION

Exercise 1

Consider the following view

```
CREATE VIEW Young_Actor
        SELECT A.act_code, A.name, A.age
        FROM Actor A
        WHERE A.age <20;
```

and the following DML instruction:

```
INSERT INTO Young_Actor (act_code, name, age)
        VALUES (18, ‘Pepe’, 25);
```

Indicate the state of the database after the execution of the instruction above. Assume that before this instruction all tables were empty.

The system would insert that row in the relation ACTOR but this row wouldn’t be visible through the view YOUNG_ACTOR as the condition is not met. Note that the view does not include the WITH CHECK OPTION.
(If the view had been defined with the WITH CHECK OPTION clause, then the insertion would have been rejected by the DBMS.)

Exercise 2

Given the following DDL command in SQL

```
CREATE TABLE Performance
        ( role_code CHAR(2) PRIMARY KEY
                REFERENCES Role(role_code) ON DELETE CASCADE,
        act_code CHAR(9) PRIMARY KEY,
        per_date DATE NOT NULL,
        FOREIGN KEY act_code REFERENCES Actor(act_code));
```

Indicate whether the definition of the *Performance* relation is correct. In other case, spot out the errors and write the command again in a correct way

There is only one error: a relation cannot have two primary keys. The corrected definition would be:

```
CREATE TABLE Performance
( role_code CHAR(2) REFERENCES Role(role_code) ON DELETE CASCADE,
act_code CHAR(9),
per_date DATE NOT NULL,
PRIMARY KEY (role_code,act_code),
FOREIGN KEY act_code REFERENCES Actor(act_code));
```