# Integrity rules
# in Oracle Databases



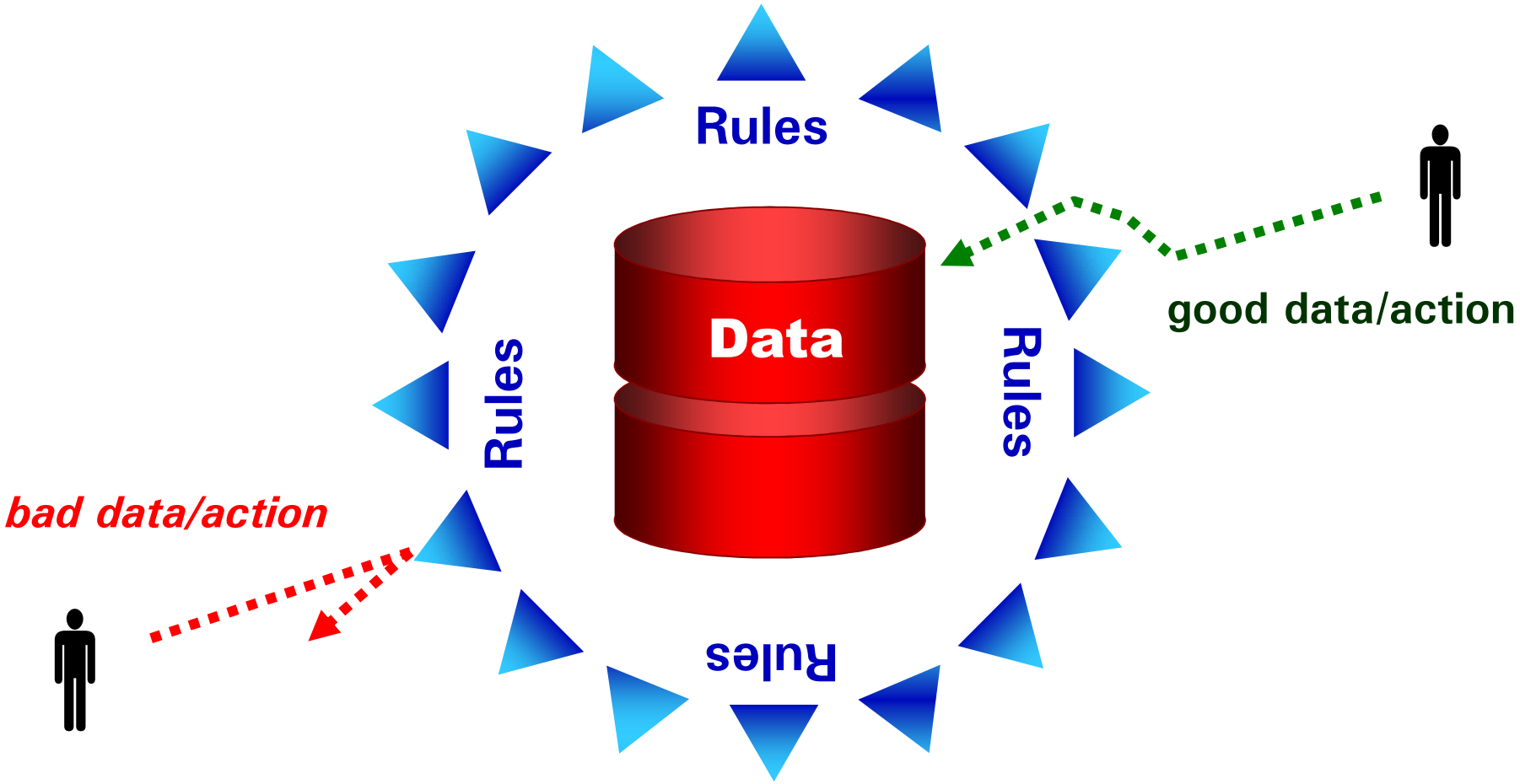
Rui Oliveira© rui.oliveira@ipleiria.pt

# References

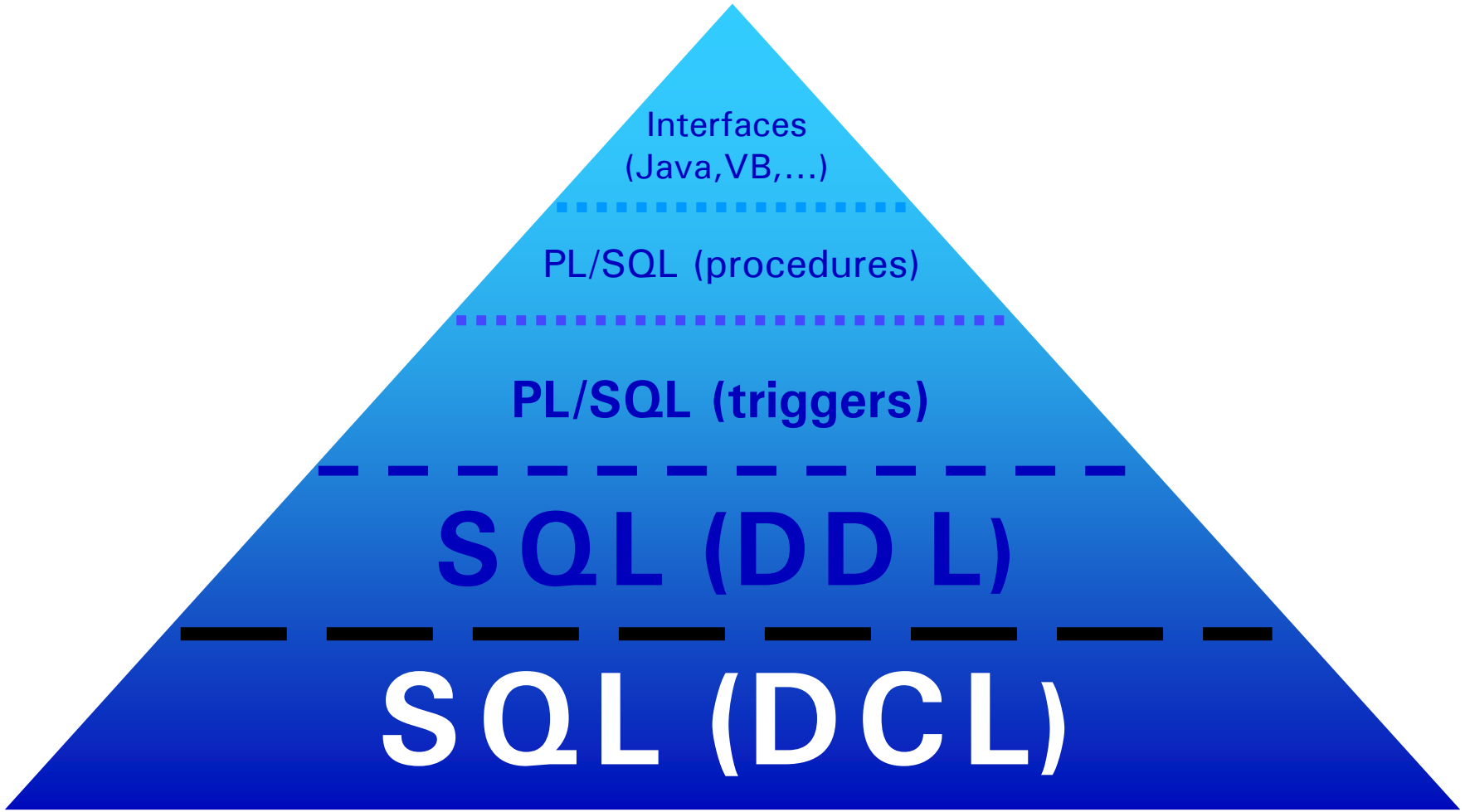
Further study required

- "*Oracle PL/SQL Programming"*, Steven Feuerstein, 5th edition, O'Reilly, 2009
- Manual "*Oracle Database SQL Language Reference 11g Release 2 (11.2)*", Oracle, 2011

# Integrity rules



# Integrity rules pyramid



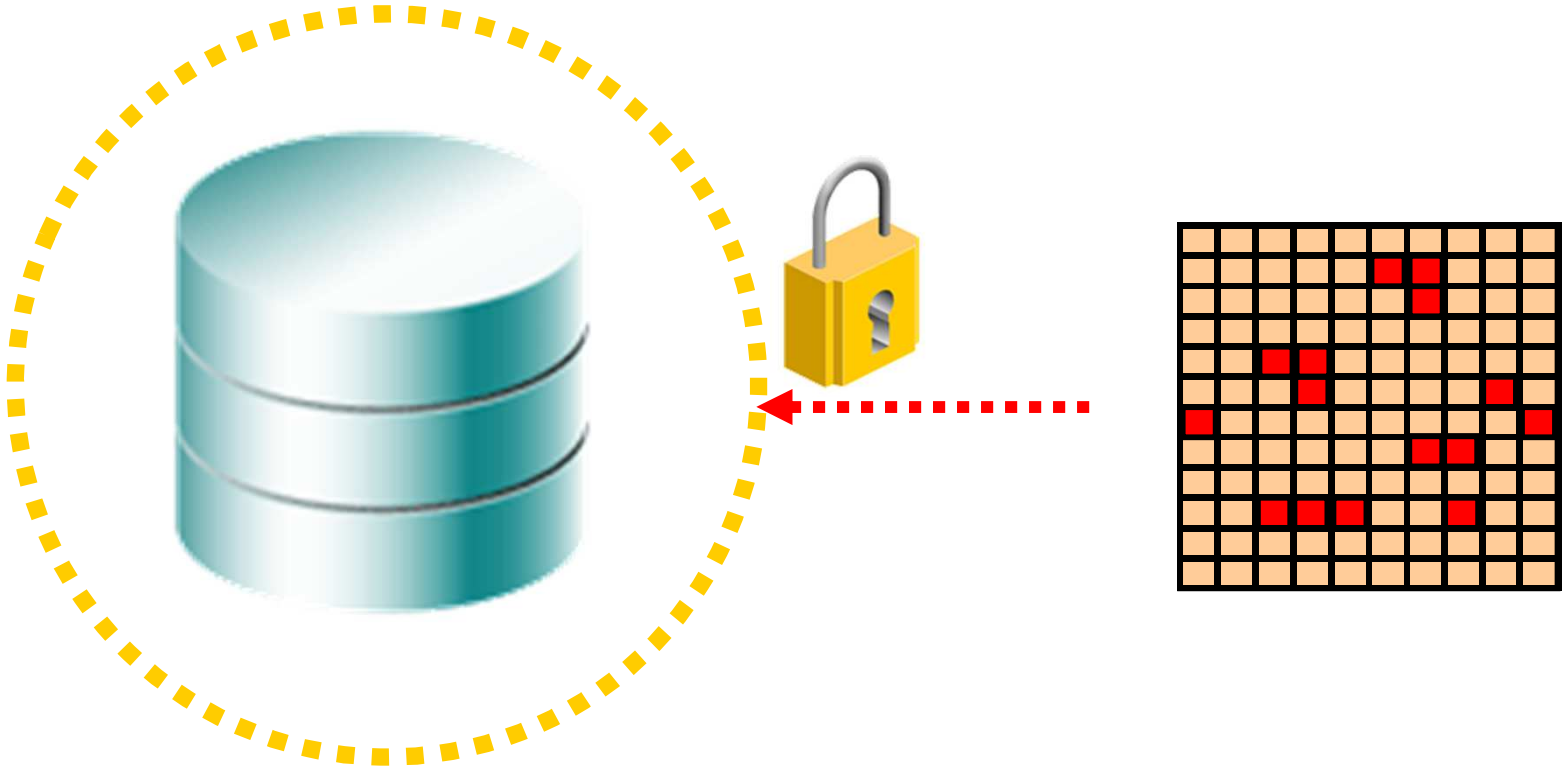
# Integrity rules

Common integrity rules

- Data types (NUMBER, INTEGER, etc.)
- Mandatory data existence (NOT NULL)
- Unique values (UNIQUE, PRIMARY KEY)
- Data domains (CHECK...)
- Referential integrity (FOREIGN KEY...)
- Privileges (GRANT, REVOKE,...)

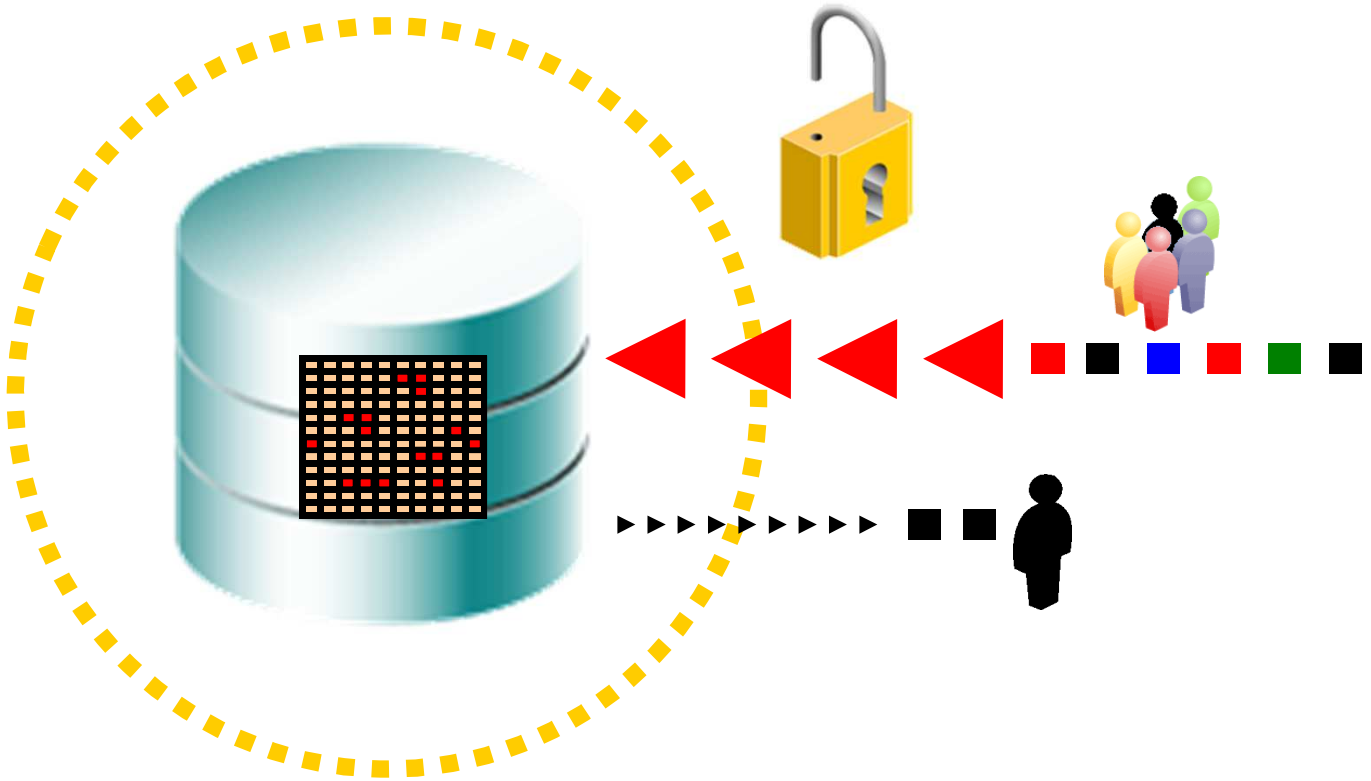
# Timing

When to enforce integrity rules? Preventive approach



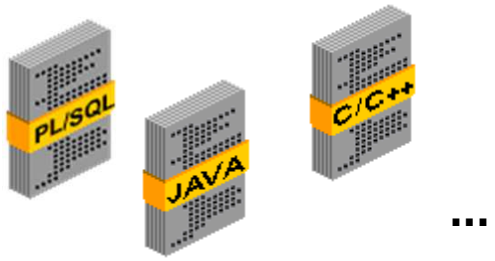
# Timing

When to enforce integrity rules? *Corrective* approach



# Risk factors

How to jeopardize data integrity?

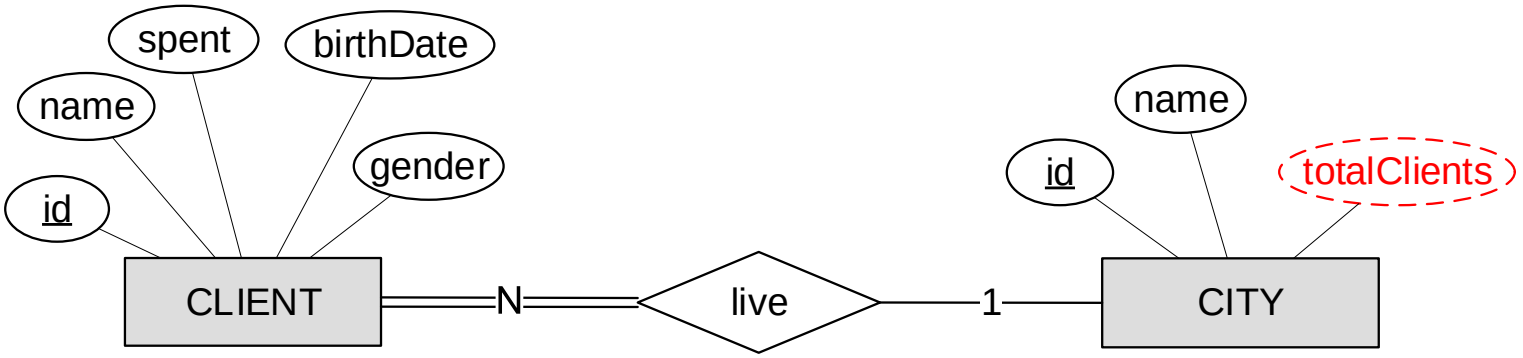



...

- Careless
- Uninterested
- *Interested*

- Bad programming
- Intentional

# The scenario




=

**CLIENTS**

| id | name | *cityId* | birthDate | spent | gender |
|---|---|---|---|---|---|
| 1 | António Freitas | *1* | 1980-04-06 | 1200 | M |
| 2 | Rita Marujo | *2* | 1983-01-06 | 1500 | F |
| 3 | Carlos da Silva | *3* | 1972-01-31 | 100 | M |
| 4 | Ana Oliveira | *1* | | 5400 | F |
| 5 | João Silva | *3* | 1978-12-04 | 650 | M |

**CITIES**

| id | name |
|---|---|
| 1 | Leiria |
| 2 | Lisboa |
| 3 | Coimbra |
| 4 | Guarda |

fk

# Fact

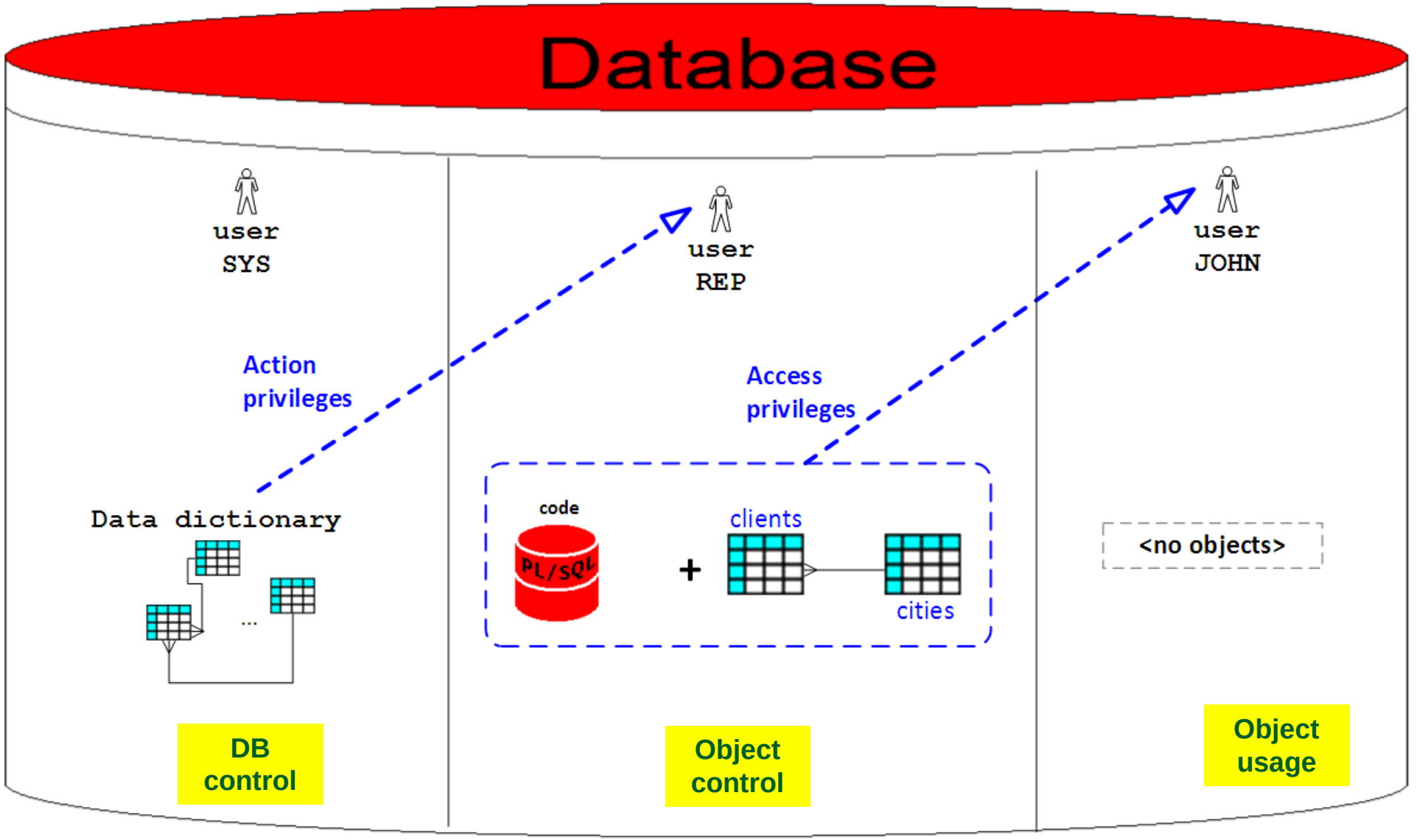
## When you create a table you are immediately enforce integrity rules.

**Example:**

```
CREATE TABLE cities (
id          NUMBER(2)       PRIMARY KEY,
name        VARCHAR2(100)   NOT NULL);

CREATE TABLE clients (
id          NUMBER(10),
...
cityId      NUMBER(2)       NOT NULL,
...
gender      CHAR(1),
CONSTRAINT pk_clients_id      PRIMARY KEY (id),
CONSTRAINT ck_clients_gender CHECK (gender IN ('M','F')),
CONSTRAINT fk_clients_cityId FOREIGN KEY (cityId) REFERENCES cities(id),
CONSTRAINT nn_clients_gender CHECK (gender IS NOT NULL) );
```

# Preparing the database for rule implementation



# Implementing rules: example #1

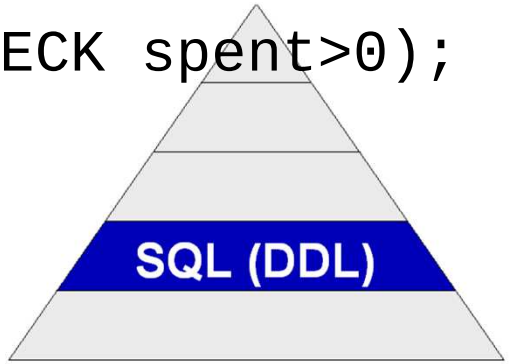
**RULE description**

- The amount spent by each client must always be greater than 0 (zero)

**RULE implementation**

**(REP user)**

- ```
  ALTER TABLE clients
  ADD CONSTRAINT ck_clients_spent (CHECK spent>0);
  ```


  ```

# Rule #2

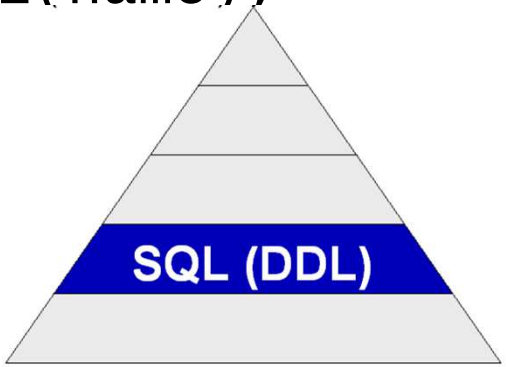
**RULE description**

- Ensure that city names are unique

**RULE implementation**

**(REP user)**

- ```
  ALTER TABLE cities
  ADD CONSTRAINT uq_cities_name UNIQUE(name);
  ```


  ```

# Rule #3

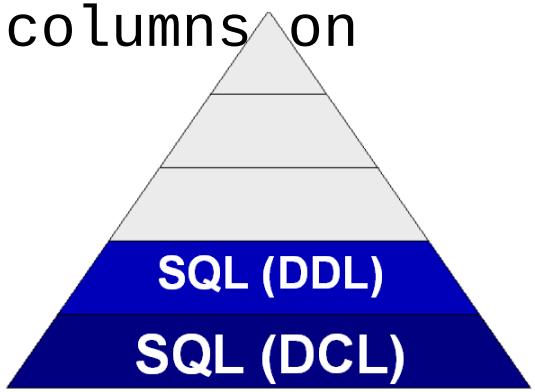
**RULE description**

- Ensure that user JOHN sees only clients' name and gender

**RULE implementation**

**(REP user)**

- Create object to limit the visible columns on the CLIENT table (it's SQL DDL)
- Allow JOHN to use that object (it's SQL DCL)



# Rule #3

**(SYS user)**

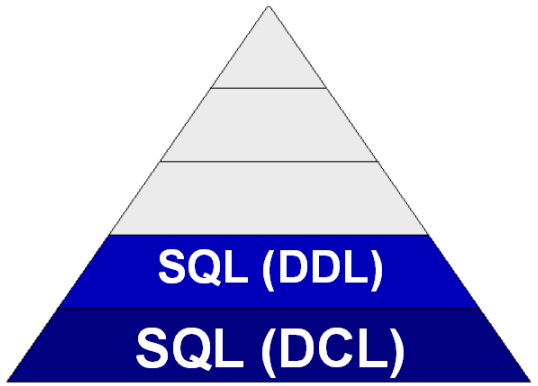
- `GRANT CREATE VIEW TO rep;` — SQL DCL

**(REP user)**

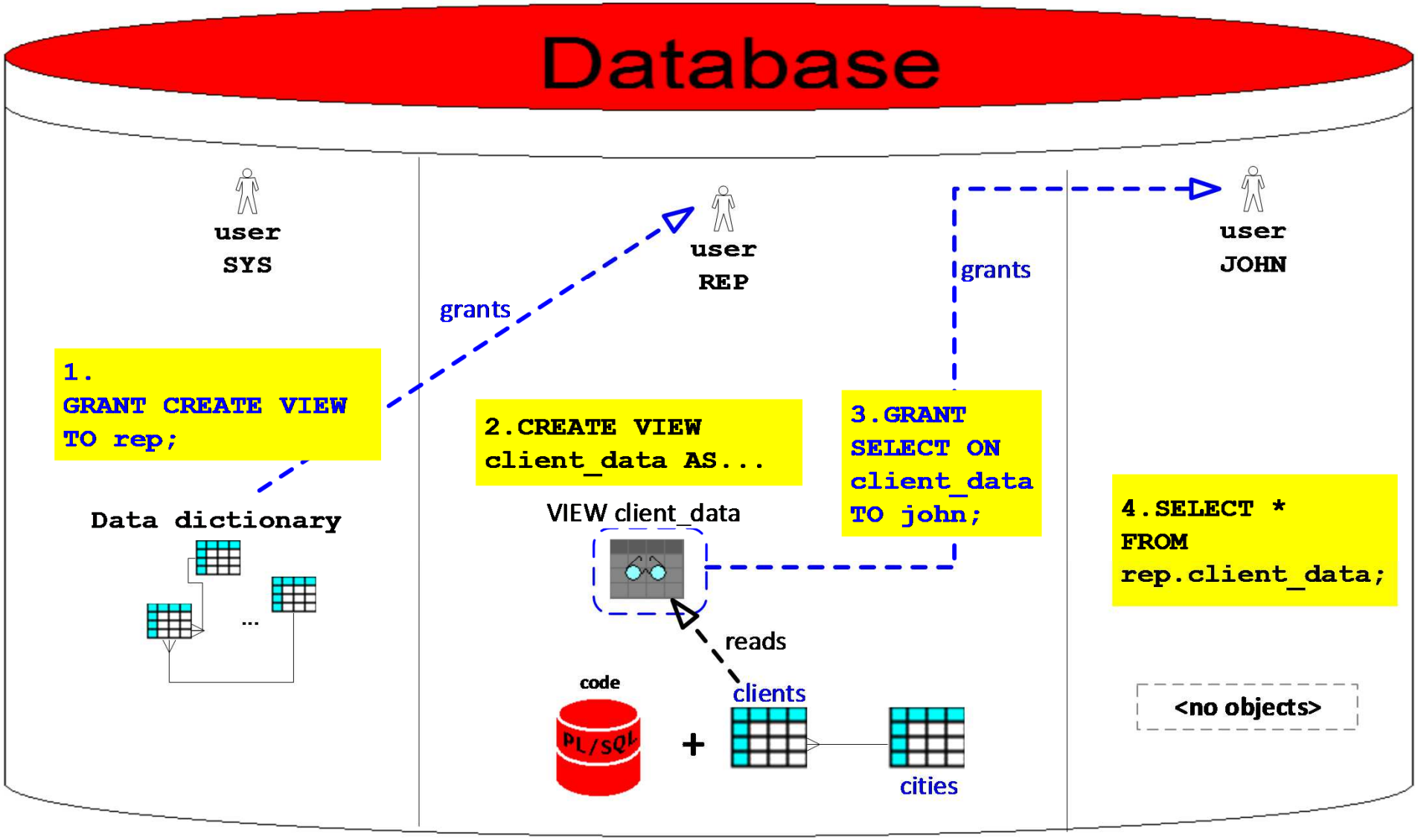
- 
```
CREATE VIEW client_data AS
    SELECT name, gender
    FROM clients;
```
— SQL DDL
- `GRANT SELECT ON client_data TO john;` — SQL DCL

**(JOHN user)**

- `SELECT * FROM rep.client_data;`



# Preparing the database for rule implementation



# Facts about *views*

**Views are database objects that store**

**only**

**a SELECT statement.**



Rui Oliveira© rui.oliveira@ipleiria.pt

# Rule #4

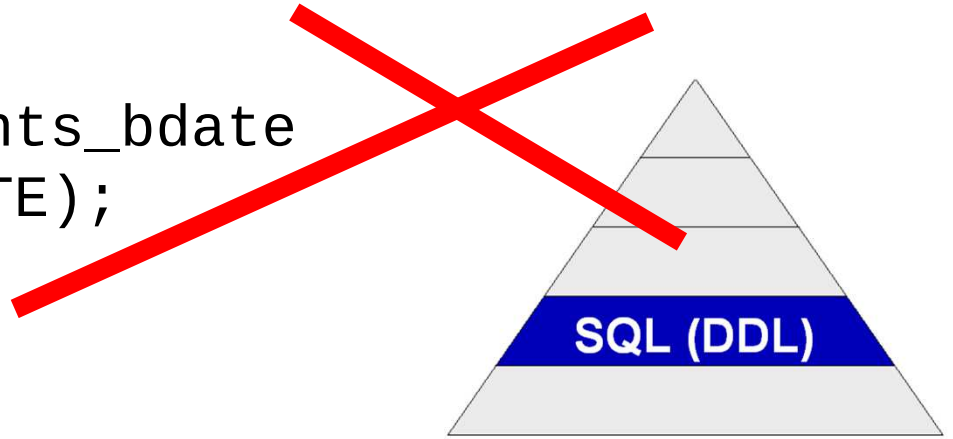
**RULE description**

- Ensure that a client's birthdate is never higher than the system data

**RULE implementation**

**(REP user)**

- 
```
ALTER TABLE clients
ADD CONSTRAINT ck_clients_bdate
CHECK (birthdate<SYSDATE);
```



# Triggers: how to enforce rule #4

```
CREATE OR REPLACE TRIGGER tri_biu_clients
BEFORE INSERT OR UPDATE OF birthDate ON clients
FOR EACH ROW
BEGIN
   IF (:NEW.birthDate>SYSDATE) THEN
      RAISE_APPLICATION_ERROR (-20001,'Invalid date');
   END IF;
END;
/
```

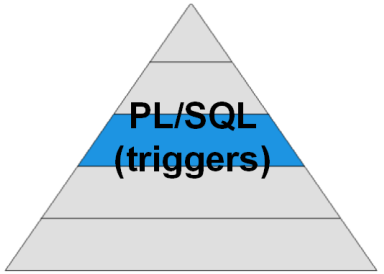
timing

target

event

number of times to execute

rule

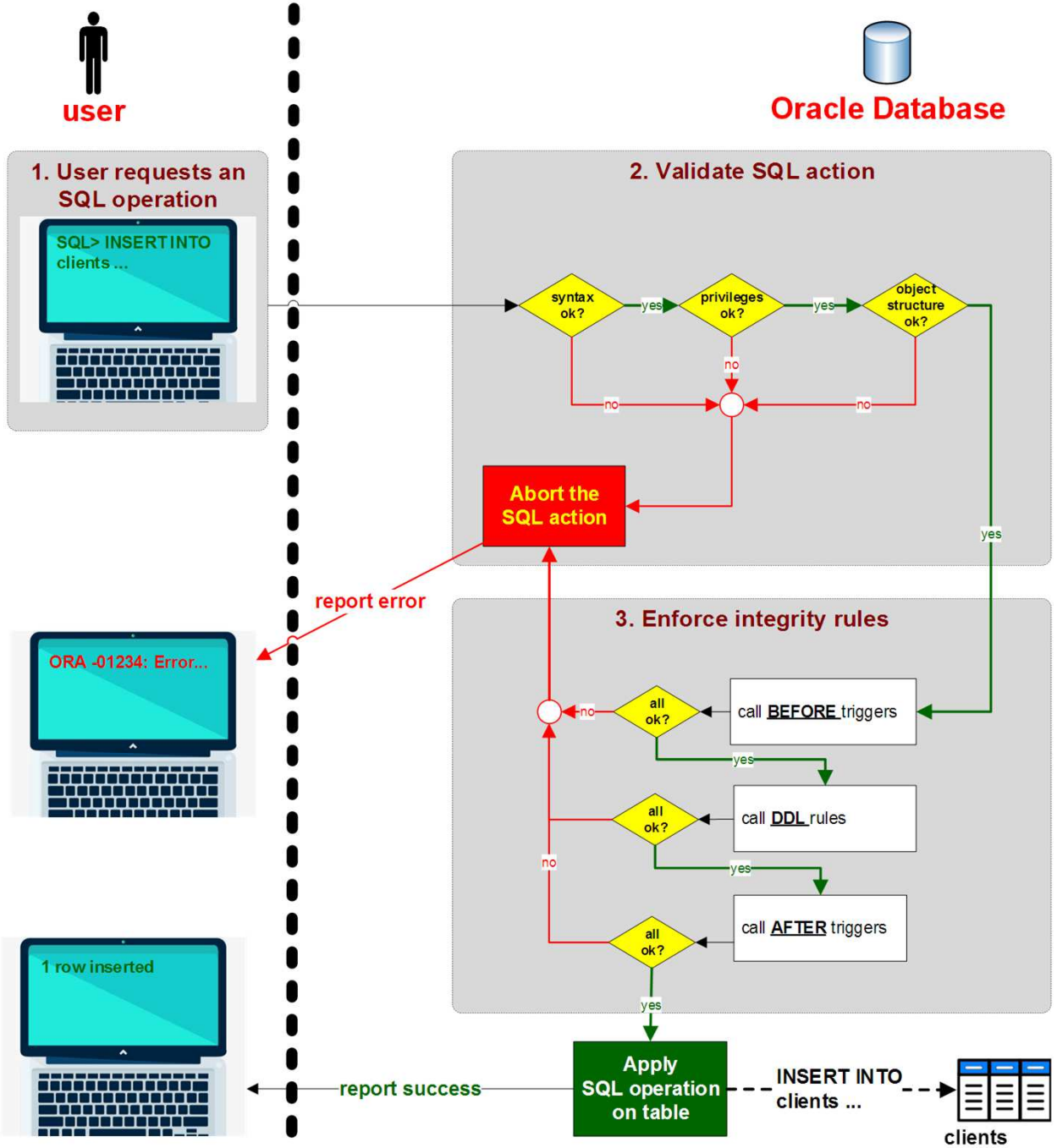


## *Facts about triggers*

**#1 - Triggers are database objects which store PL/SQL code**

**#2 - Triggers are *triggered* after users' data manipulation requests (inserts, deletes and updates)**

# Imposing integrity rules with *for each row* triggers



# *Triggers* : concepts

***target row*** = the row being inserted, updated or deleted

Example:

```
INSERT INTO cities (id,name)
VALUES (5,'Faro');
```

**CITIES**

| id | name |
|---|---|
| 1 | Leiria |
| 2 | Lisboa |
| 3 | Coimbra |
| 4 | Guarda |
| 5 | Faro |

***target row*** →

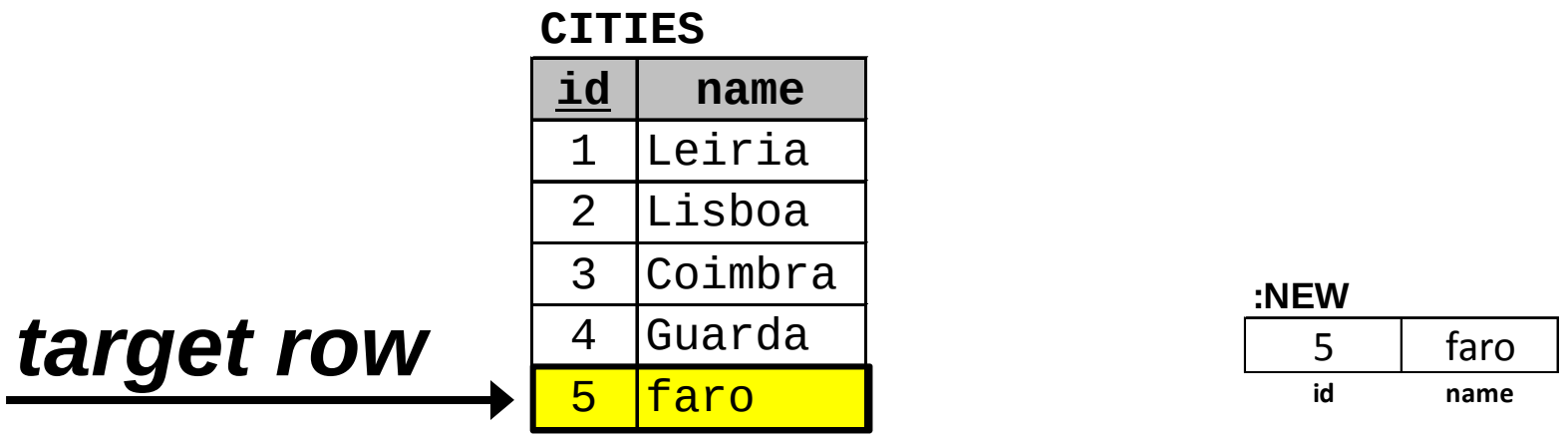
## Triggers control two pointers:

- **:NEW** is a pointer to the ***new*** data on the target row (inserts and updates only)
- **:OLD** is a pointer to the ***old*** data on the target row (updates and deletes only)

# *Triggers* :NEW and :OLD pointers

Example:

```
INSERT INTO cities (id,name)
VALUES (5,'faro');
```



# *Triggers* :NEW and :OLD pointers

Example:

```
UPDATE cities
SET name = 'Faro'
WHERE id=5;
```

**CITIES**

| id | name |
|---|---|
| 1 | Leiria |
| 2 | Lisboa |
| 3 | Coimbra |
| 4 | Guarda |
| 5 | faro |

***target row***

| | id | name |
|---|---|---|
| **:OLD** | 5 | faro |
| **:NEW** | 5 | Faro |

## Facts about *triggers*

**BEFORE triggers can be used to:**

- generate missing data on the target row
- cancel an SQL request if it disrespects a data integrity rule
- correct invalid data in the target row
- ...

"***BEFORE***" means "***BEFORE*** **DDL**" rules apply

## Facts about *triggers*

**AFTER triggers can be used to:**

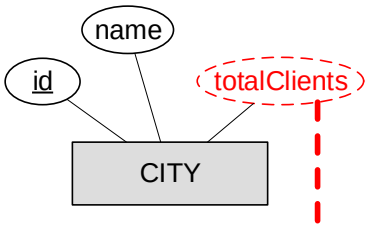
- update calculated values
- generate data outside of the target row
- ...

- "***AFTER***" means "***AFTER* DDL**" rules apply

# Rule #5

## RULE description

- For performance reasons, the *totalClients* attribute will be denormalized and stored on table **CITIES**.




**CITIES**

| id | name | totalClients |
|---|---|---|
| 1 | Leiria | |
| 2 | Lisboa | |
| 3 | Coimbra | |
| 4 | Guarda | |

# Rule #5

**RULE description** (continuation)

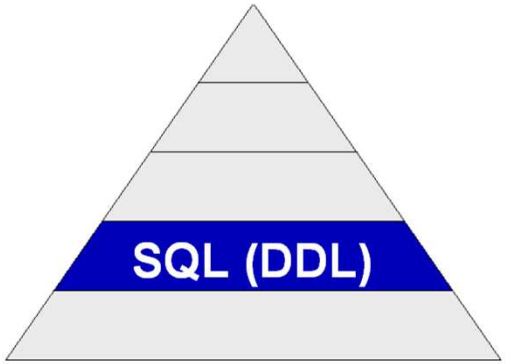
- *totalClients* integrity has to be kept:
  - When a new city is added, make sure *totalClients* starts at 0;
  - When a client is added, make sure *totalClients* is incremented for that client city;
  - When a client is removed, make sure *totalClients is* decremented for the old client city;
  - When a client changes his/her city, make sure *totalClients* is incremented for that client new city and decremented for that client old city;

# Rule #5

**RULE implementation (in REP user's account)**

*"For performance reasons, the totalClients attribute will be denormalized and stored on table* ***CITIES****."*

```
ALTER TABLE cities
ADD (totalClients NUMBER(4) NOT NULL);
```



# Rule #5

## RULE implementation (in REP user's account)

*"When a new city is added, make sure totalClients starts at 0;"*

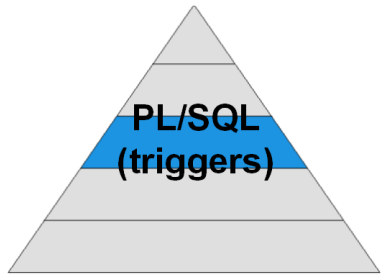
```
INSERT INTO cities(id,name)
VALUES(5,'Faro');
```

→

:NEW

| 5 | 'Faro' | |
|---|---|---|
| id | name | totalClients |

```
CREATE OR REPLACE TRIGGER tri_bi_cities
 BEFORE INSERT ON cities
 FOR EACH ROW
 BEGIN
     :NEW.totalClients := 0;
 END;
```

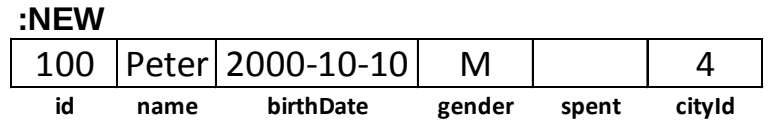


# Rule #5

**RULE implementation** (continuation)

*"When a client is added, make sure totalClients is incremented for that client city"*

```
INSERT INTO clients(id,name,birthdate,gender,cityId)
VALUES
(100,'Peter','2000-10-10','M',4);
```




```
CREATE OR REPLACE TRIGGER tri_ai_clients
 AFTER INSERT ON clients
 FOR EACH ROW
 BEGIN
    UPDATE cities SET totalClients = totalClients + 1
    WHERE id = :NEW.cityId;
 END;
```

# Rule #5

**RULE implementation** (continuation)

*"When a client is removed, make sure totalClients is decremented for that old client city"*

**DELETE FROM clients** ⟶ **:OLD**

| 4 | Ana Oliveira | | F | 5400 | 1 |
|---|---|---|---|---|---|
| **id** | **name** | **birthDate** | **gender** | **spent** | **cityId** |

**WHERE id = 4;**

```
CREATE OR REPLACE TRIGGER tri_ad_clients
AFTER DELETE ON clients
FOR EACH ROW
BEGIN
    UPDATE cities
    SET totalClients = totalClients - 1
    WHERE id = :OLD.cityId;
END;
```

# Rule #5

*"When a client changes his/her city(...)"*

```
UPDATE clients SET cityId=2 WHERE clients.id=4;
```

```
CREATE OR REPLACE TRIGGER tri_au_clients
AFTER UPDATE OF cityId ON clients
FOR EACH ROW
BEGIN
   UPDATE cities
   SET totalClients = totalClients + 1
   WHERE id = :NEW.cityId;

   UPDATE cities
   SET totalClients = totalClients – 1
   WHERE id = :OLD.cityId;
END;
```

→

| | id | name | birthDate | gender | spent | cityId |
|---|---|---|---|---|---|---|
| **:OLD** | 4 | Ana Oliveira | | F | 5400 | **1** |
| **:NEW** | 4 | Ana Oliveira | | F | 5400 | **2** |

# Rule #6

## RULE description

- The gender of each client will be automatically defined by the DB using the client's first name.
- If a client's gender is impossible to define, the gender will be set as "?"
- The integrity of *gender* has to be kept:
  - When the client name changes
  - When a new client is added

# Rule #6

## RULE implementation

<in class>



Rui Oliveira© rui.oliveira@ipleiria.pt

# Other motivations for using PL/SQL

Functions

# PL/SQL functions

What is a PL/SQL *function*?

- PL/SQL code stored in the database that users/applications can call *explicitly* (unlike triggers)

Goals

- Better code reusing
- Simplify/improve the coding logic

Requirements

- **Always** return a value
- May have entry values
- May query the database

# Functions in databases

Examples you already know/use:

- UPPER
- TO_CHAR
- TO_DATE
- NVL
- MONTHS_BETWEEN
- ...

# Functions in databases

Function *f_first_name* (retrieve the first name for rule #6)

```
CREATE FUNCTION f_first_name(p_name VARCHAR2)
RETURN VARCHAR2 IS
   v_pos INTEGER;
BEGIN
   v_pos := INSTR(p_name,1,' ');
   IF (v_pos>0) THEN
      RETURN (SUBSTR(p_name,v_pos));
   ELSE
      RETURN (p_name);
   END IF;
END;
```