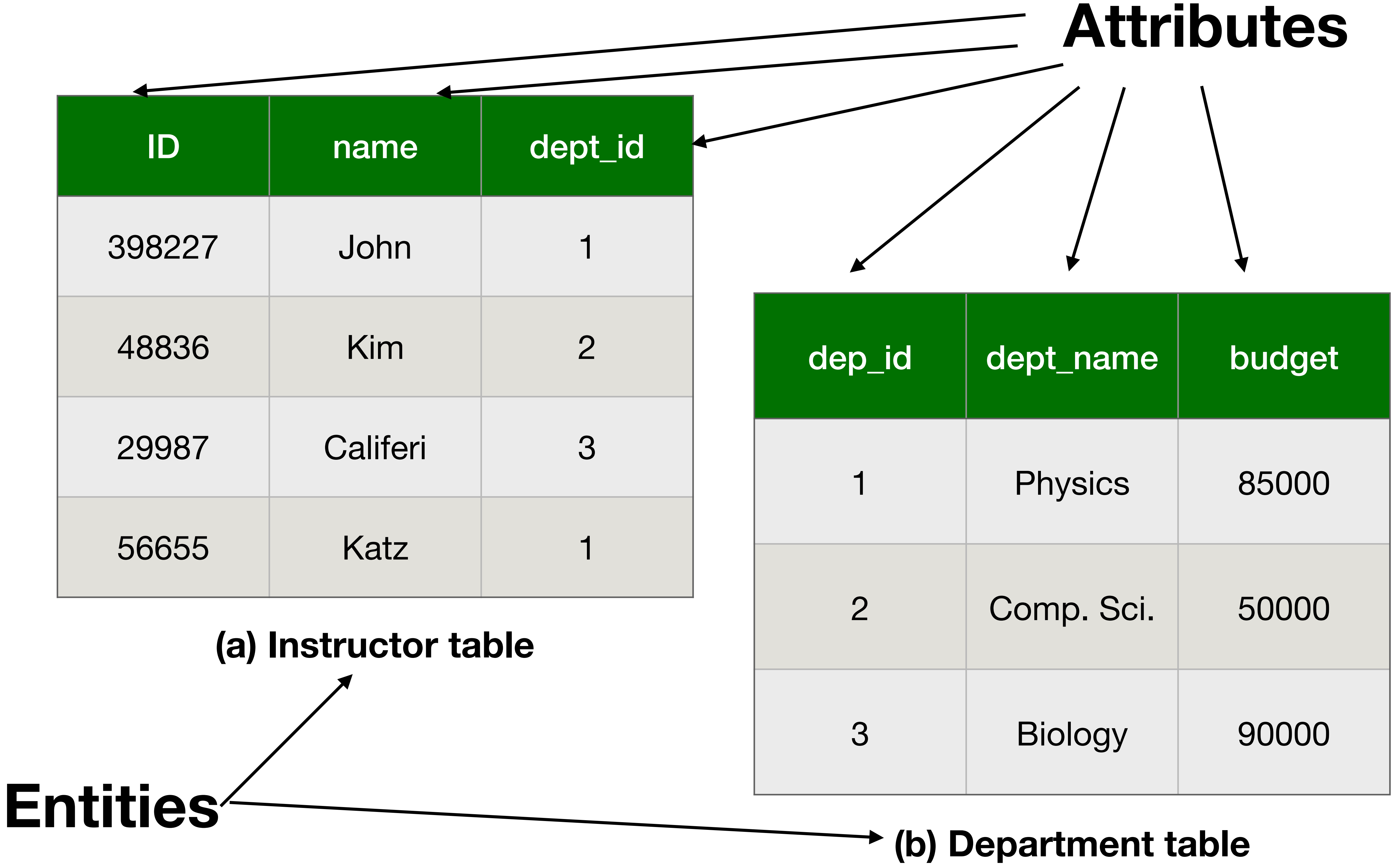
Database Systems
The Relational Model
The Basics

CSC675-775
Jose Ortiz
jortizco@sfsu.edu

The Basics



Attributes

ID	name	dept_id
398227	John	1
48836	Kim	2
29987	Califeri	3
56655	Katz	1

(a) Instructor table

dep_id	dept_name	budget
1	Physics	85000
2	Comp. Sci.	50000
3	Biology	90000

(b) Department table

Entities

Relational Databases

- A database that can be defined using the relational model.
- Relations:
  - **Relation schema:**
    - Name of the relation, the columns of the table and **domain** of their attributes.
  - **Relation instance:**
    - A table with rows and columns.

Relation Schema

- Example: Students(sid:char, name:varchar, login:varchar, age: integer, gpa: float)

- **Relation scheme** specifies:

  1. **Name** of the **relation**

  2. **Name** of each **column** / fields / attribute

  3. The **domain** for **each column**

Relation Instance

- A table where all rows contain the same number of attributes (columns)

Tuples

sid	Name	Login	Age	Gpa
53666	Raphael	raphael@cs	18	3.1
53688	John	john@eecs	28	2.0
53650	Chistian	chistian@math	58	3.9

cardinality (#rows) = 3,
degree (#columns) = 5

- Therefore, a **relation instance** (AKA relation) is simply a **set of tuples**

Relational Model and Query Language

- Simple and intuitive data representation
- Supports for high-level query language
- Queries written intuitively
- DBMS is responsible for efficient evaluation

DDL: Data Definition Language

SQL the most widely used query language

Creating a Database

- Relation schema

```
CREATE SCHEMA MyDatabase;

CREATE DATABASE MyDatabase;
```

Database

- Create a database/schema

```
CREATE SCHEMA MyDatabase;

CREATE DATABASE MyDatabase;
```

- Using the database

```
USE MyDatabase;
```

Creating Tables/Relations Using SQL

```
CREATE TABLE IF NOT EXISTS Department(
    did TINYINT UNSIGNED AUTO_INCREMENT,
    name VARCHAR(30) NOT NULL,
    budget DECIMAL(13,2) NOT NULL,
    CONSTRAINT Department_PK PRIMARY KEY (id)
};

CREATE TABLE IF NOT EXISTS Student (
    sid CHAR(9) NOT NULL,  -- Not computable
    name VARCHAR(30) NOT NULL,
    login VARCHAR(30) NOT NULL,
    age INT UNSIGNED NOT NULL,
    gpa FLOAT(3,2) DEFAULT 0.00,
    department INT UNSIGNED NOT NULL,
    CONSTRAINT Student_PK PRIMARY KEY (sid),
    CONSTRAINT Student_Department_FK  FOREIGN KEY
              (department) REFERENCES Department(id)

);
```

M:1 Relationship

Employee

ssn	name	dob	dept
617335456	John	08/03/75	12
345444567	Mary	03/06/90	12
345223456	Jane	05/12/99	13

Department

did	name	budget
12	eng	35000
13	marketing	50000
14	HR	30000

```
CREATE TABLE IF NOT EXISTS Department (
    did TINYINT(2) UNSIGNED AUTO_INCREMENT,
    name VARCHAR(30) NOT NULL,
    budget DECIMAL(13,2) NOT NULL,
    CONSTRAINT Department_PK PRIMARY KEY (did)
};

CREATE TABLE IF NOT EXISTS Employee(
    ssn CHAR(9) NOT NULL,
    name VARCHAR(30) NOT NULL,
    dob DATETIME NOT NULL,
    dept TINYINT(2) UNSIGNED NOT NULL,
    CONSTRAINT Employee_PK PRIMARY KEY (ssn),
    CONSTRAINT Employee_Department_FK  FOREIGN KEY
              (dept) REFERENCES Department(did)

);
```

M:N Relationship

Employee

ssn	name	dob
617335456	John	08/03/75
345444567	Mary	03/06/90
345223456	Jane	05/12/99

Manager

manager	dept
617335456	1234
345444567	1234
345223456	1235
345223456	1236

Department

did	name	Budget
1234	eng	35000
1235	market	50000
1236	HR	30000

```
CREATE TABLE IF NOT EXISTS Employee(
    ssn CHAR(9) NOT NULL,
    name VARCHAR(30) NOT NULL,
    dob DATETIME NOT NULL,
    dept TINYINT(3) UNSIGNED NOT NULL,
    CONSTRAINT Employee_PK PRIMARY KEY (ssn),

);
CREATE TABLE IF NOT EXISTS Department (
    did TINYINT UNSIGNED AUTO_INCREMENT,
    name VARCHAR(30) NOT NULL,
    budget DECIMAL(13,2) NOT NULL,
    CONSTRAINT Department_PK PRIMARY KEY (did)
};
CREATE TABLE IF NOT EXISTS Manager(
    manager CHAR(9) NOT NULL,
    dept TINYINT UNSIGNED NOT NULL,
    CONSTRAINT Manager_Employee_FK  FOREIGN KEY
              (manager) REFERENCES Employee(ssn) ON DELETE CASCADE,
    CONSTRAINT Manager_Department_FK  FOREIGN KEY
              (dept) REFERENCES Department(did) ON DELETE CASCADE

);
```

Altering Relations

sid	name	login	age	gpa	dep_id
NULL	NULL	NULL	NULL	NULL	NULL

Altering a relation

```
ALTER TABLE Student ADD COLUMN graduated BOOLEAN DEFAULT FALSE;
```

sid	name	login	age	gpa	dep_id	graduated
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
ALTER TABLE Student MODIFY sid INT(5) UNSIGNED AUTO_INCREMENT;
```

Destroying Relations

- Destroying a relation
  - DROP TABLE ENTITY
    - Schema and instance information are deleted
    - DROP TABLE Students
      - Destroys the relation students
- TRUNCATE TABLE ENTITY
  - Delete all the data in the table but keeps the relation.

DML: Data Manipulation Language

SQL the most widely used query language

Insert Data in DB

```
INSERT INTO table_name (col_1, col_2, col_3, … , col_n )
VALUES (value_1, value_2, value_3, … , value_m);

— Only if m_values == degree of the table (e.g. n);
INSERT INTO table_name
VALUES (value_1, value_2, value_3, … ,  value_m);

— Example inserting tuple into table students;
INSERT INTO Students (name, login, age, gpa, graduated, dep_id)
VALUES (‘Raphael’, ‘raphael@cs’, 20 ,  3.10, false, 1 );
```

sid	name	login	age	gpa	graduated	dep_id
53666	Raphael	raphael@cs	18	3.10	0	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Insert Data in DB

```
INSERT INTO table_name (col_1, col_2, col_3, … , col_n )
VALUES (value_1, value_2, value_3, … , value_m);

--Only if m_values == degree of the table (e.g. n);
INSERT INTO table_name
VALUES (value_1, value_2, value_3, … ,  value_m);

--With auto_increment in sid to avoid duplicates;
INSERT INTO Students (name, login, age, gpa, graduated, dep_id)
VALUES (‘John’, ‘john@cs’, 58 , 3.9, false, 3);
```

sid	name	login	age	gpa	Graduated	dep_id
53666	Raphael	raphael@cs	18	3.1	0	1
53667	John	john@eecs	58	3.9	0	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL

SELECT DATA FROM DB

```
SELECT (ATTRIBUTE_i, … , ATTRIBUTE_m) FROM TABLE;

SELECT (name, gpa) FROM Student;
```

name	gpa
Raphael	3.1
John	2.0
Christina	3.9

```
SELECT * FROM Student;
```

sid	name	Email	gpa	graduated	dept
53666	Raphael	raphael@cs	3.1	0	1
53667	John	john@eecs	2.0	0	3
53668	Christina	chris@cs	3.9	0	1

Select With Where Conditions

sid	name	Email	gpa	graduated	dept
53666	Raphael	raphael@cs	3.1	0	1
53667	John	john@eecs	2.0	0	3
53668	Christina	chris@cs	3.9	0	1

```
SELECT (ATTRIBUTE_i, … , ATTRIBUTE_m) FROM TABLE WHERE CONDITION;
```

```
SELECT (name, gpa, dep_id) FROM Student WHERE dept=1 ;
```

name	gpa	dept
Raphael	3.1	1
Chistina	3.9	1

```
SELECT * FROM Student WHERE gpa<=3.1;
```

sid	name	email	gpa	graduated	dept
53666	Raphael	raphael@cs	3.1	0	1
53667	John	john@eecs	2.0	0	3

Update data in DB

```
UPDATE TABLE
SET col_1= val_1, col_2 = val_2, … , col_n = val_n
WHERE condition;
```

sid	name	email	gpa	graduated	dept
53666	Raphael	raphael@cs	3.1	0	1
53667	John	john@eecs	2.0	0	3
53668	Christina	chris@cs	3.9	0	1

```
UPDATE Student SET gpa=4.0 WHERE name=‘Christina’;
```

sid	name	email	gpa	graduated	dept
53666	Raphael	raphael@cs	3.1	0	1
53667	John	john@eecs	2.0	0	3
53668	Christina	chris@cs	4.0	0	1

Updated value

Delete data in DB

```
DELETE FROM TABLE WHERE condition;
```

sid	name	email	gpa	graduated	dept
53666	Raphael	raphael@cs	3.1	0	1
53667	John	john@eecs	2.0	0	3
53668	Christina	chris@cs	3.9	0	1

```
DELETE FROM Student WHERE LENGTH(email)<>10;
```

sid	name	email	gpa	graduated	dept
53666	Raphael	raphael@cs	3.1	0	1

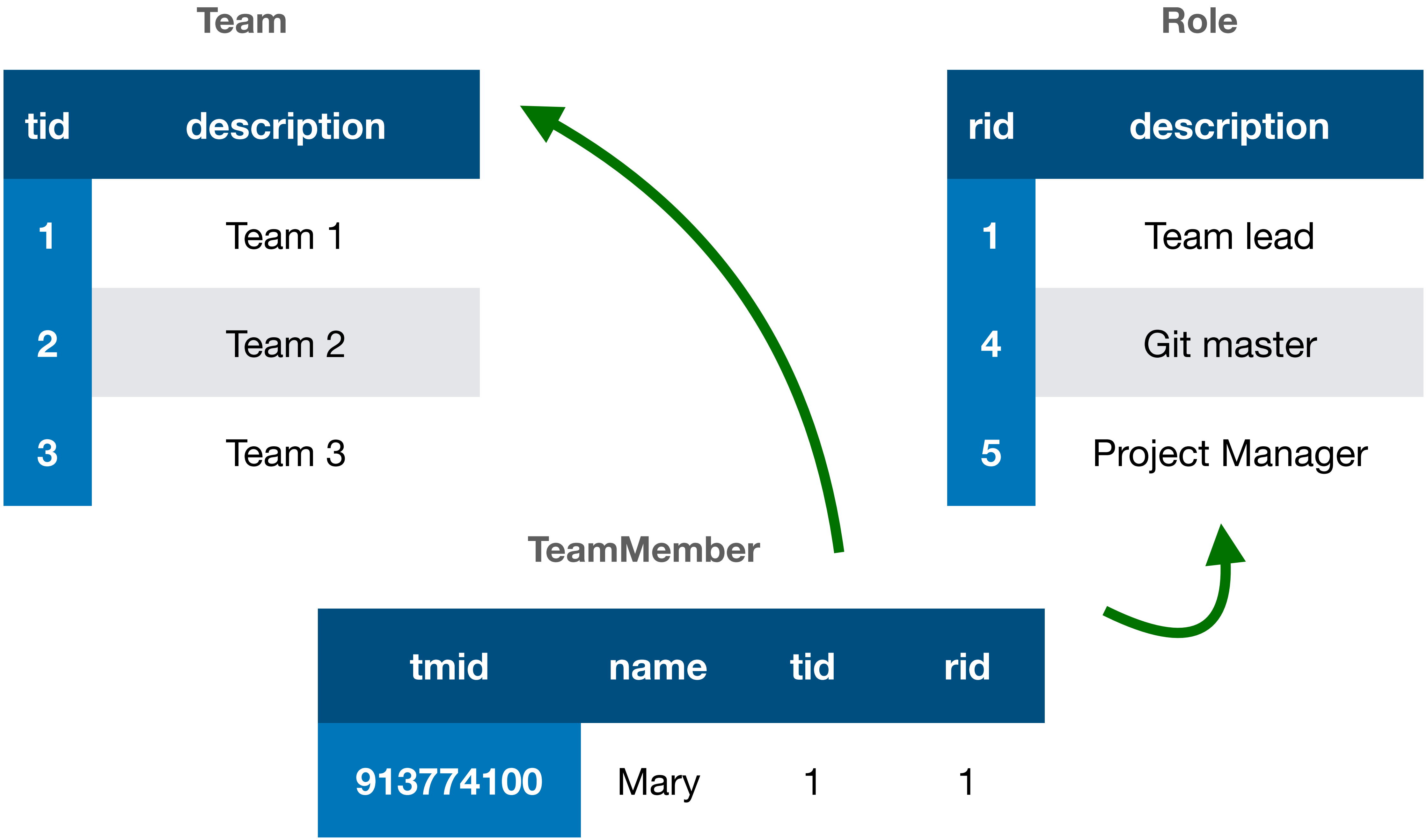
```
DELETE FROM Student WHERE name=‘Raphael’;
```

sid	name	email	gpa	graduated	dept
NULL	NULL	NULL	NULL	NULL	NULL

The Problem with Delete

Problem #1: DELETE FROM Role WHERE rid=2 AND rid=3;

Problem #2: DELETE FROM Team WHERE tid=1



Team

tid	description
1	Team 1
2	Team 2
3	Team 3

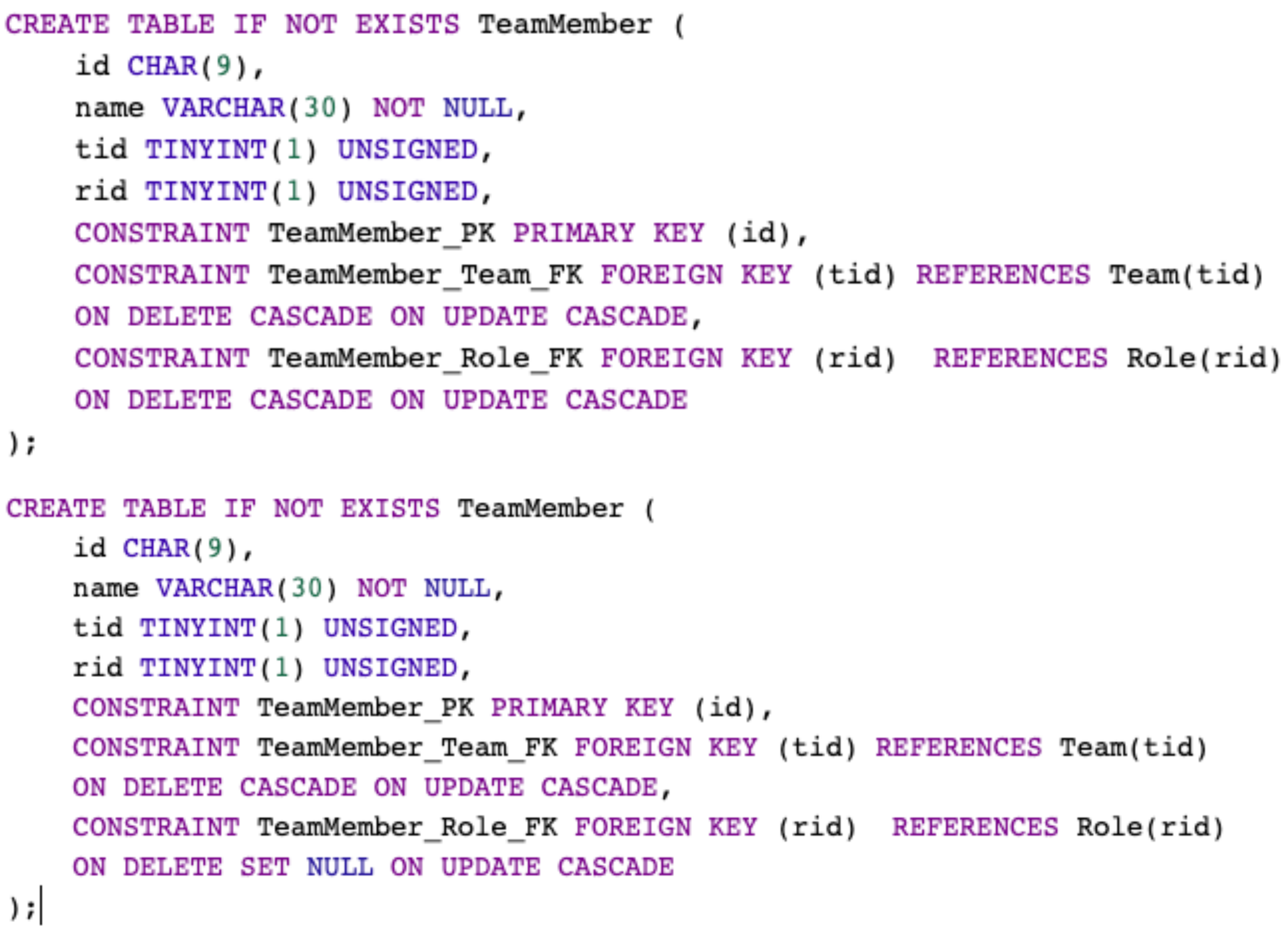
Role

rid	description
1	Team lead
4	Git master
5	Project Manager

TeamMember

tmid	name	tid	rid
913774100	Mary	1	1

ON DELETE CASCADE ON UPDATE CASCADE



```
CREATE TABLE IF NOT EXISTS TeamMember (
    id CHAR(9),
    name VARCHAR(30) NOT NULL,
    tid TINYINT(1) UNSIGNED,
    rid TINYINT(1) UNSIGNED,
    CONSTRAINT TeamMember_PK PRIMARY KEY (id),
    CONSTRAINT TeamMember_Team_FK FOREIGN KEY (tid) REFERENCES Team(tid)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT TeamMember_Role_FK FOREIGN KEY (rid)  REFERENCES Role(rid)
    ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS TeamMember (
    id CHAR(9),
    name VARCHAR(30) NOT NULL,
    tid TINYINT(1) UNSIGNED,
    rid TINYINT(1) UNSIGNED,
    CONSTRAINT TeamMember_PK PRIMARY KEY (id),
    CONSTRAINT TeamMember_Team_FK FOREIGN KEY (tid) REFERENCES Team(tid)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT TeamMember_Role_FK FOREIGN KEY (rid)  REFERENCES Role(rid)
    ON DELETE SET NULL ON UPDATE CASCADE
);
```

ALTER ON DELETE ON UPDATE

```
ALTER TABLE TeamMember
MODIFY CONSTRAINT TeamMember_Role_FK
ON DELETE SET NULL;
```

ALTER ON DELETE ON UPDATE

```
ALTER TABLE Student
DROP CONSTRAINT Student_Role_FK;

ALTER TABLE Student
ADD CONSTRAINT Student_Role_FK
FOREIGN KEY (rid)
REFERENCES Role(rid)
ON DELETE SET NULL
ON UPDATE CASCADE;
```

Advanced SQL

Find Student’s Team Role
Show student’s name and their role description in the team

Student

sid	name	tid	rid
913774100	Mary	1	1
976772277	John	2	2
916662266	Jose	2	1

Team

tid	description
1	CSC675 Team 1
2	CSC675 Team 2

Role

rid	description
1	Team lead
2	Git master

```
SELECT S.name AS student, R.description AS team_role
FROM  Student S, Role R
WHERE S.rid = R.rid
```

student	team_role
Mary	Team lead
John	Git master
Jose	Team lead

Find team leads. Show the name of the students, and the description of the teams

Student

sid	name	tid	rid
913774100	Mary	1	1
976772277	John	2	2
916662266	Jose	2	1

Team

tid	description
1	team 1
2	team 2

Role

rid	description
1	team lead
2	git master

```
SELECT S.name AS student, T.description AS team
FROM  Student S, Role R, Team T
WHERE S.rid = R.rid
AND S.tid = T.tid
AND R.description = ‘team lead’
```

student	team
Mary	team 1
Jose	team 2

Conceptual Evaluation Strategy for an SQL Query

- Computer the cross-product of **from-list**
- Discard resulting tuples if they fail **qualifications**
- Delete attributes that are not in **select-list**
- If **DISTINCT** is specified, eliminate duplicate rows

Find sailors (names) who have reserved boat ID 103.

```
SELECT DISTINCT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid=103
```

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96

Step 1: Computer the cross product of **from-lis**t: S and R

S.sid	S.sname	S.rating	S.age	R.sid	R.bid	R.day
22	Dustin	7	45	22	101	10-Oct-96
22	Dustin	7	45	58	103	12-Nov-96
58	Rusty	10	35	22	101	10-Oct-96
58	Rusty	10	35	58	103	12-Nov-96
31	Lubber	8	55	22	101	10-Oct-96
31	Lubber	8	55	58	103	12-Nov-96

```sql
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid=103
```

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96

Step 2: Discard resulting tuples if they **fail qualifications**: S.sid = R.sid AND r.bid=103

S.sid	S.sname	S.rating	S.age	R.sid	R.bid	R.day
~~22~~	~~Dustin~~	~~7~~	~~45~~	~~22~~	~~101~~	~~10-Oct-96~~
~~22~~	~~Dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~12-Nov-96~~
~~58~~	~~Rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10-Oct-96~~
58	Rusty	10	35	58	103	12-Nov-96
~~31~~	~~Lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10-Oct-96~~
~~31~~	~~Lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~12-Nov-96~~

```sql
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid=103
```

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96

Step 3: Delete attributes that are not in **select-list**

S.sid	S.sname	S.rating	S.age	R.sid	R.bid	R.day
~~22~~	~~Dustin~~	~~7~~	~~45~~	~~22~~	~~101~~	~~10-Oct-96~~
~~22~~	~~Dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~12-Nov-96~~
~~58~~	~~Rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10-Oct-96~~
~~58~~	Rusty	~~10~~	~~35~~	~~58~~	~~103~~	~~12-Nov-96~~
~~31~~	~~Lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10-Oct-96~~
~~31~~	~~Lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~12-Nov-96~~

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid=103
```

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96

S.sname
Rusty

Advanced SQL:

Find sailors (sailor ID) who have reserved at least one boat.

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	103	4-Oct-96

Boats

B:

bid	bname	color
101	Interlak	blue
103	Clipper	green

```
SELECT DISTINCT sid
 FROM Reserves
```

S.sid
22
58

Advanced SQL:

Find the names of the sailors who have reserved at least one boat.

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	103	4-Oct-96

Boats

B:

bid	bname	color
101	Interlak	blue
103	Clipper	green

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.id = R.id
```

S.sname
Dustin
Dustin
Rusty

Advanced SQL:

Find the names of the sailors who have reserved at least one boat.

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	103	4-Oct-96

Boats

B:

bid	bname	color
101	Interlak	blue
103	Clipper	green

```
SELECT DISTINCT S.sname
FROM Sailors S, Reserves R
WHERE S.id = R.id
```

S.sname
Dustin
Rusty

Advanced SQL:

Find the names of the sailors who have reserved the green boat.

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	103	4-Oct-96

Boats

B:

bid	bname	color
101	Interlak	blue
103	Clipper	green

```
SELECT DISTINCT S.sname AS sailorName
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = ‘green’
```

sailorName
Dustin
Rusty

Advanced SQL

- Strings in SELECT Clause

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

```
SELECT ’Sailor”s age: ’, S.age
FROM Sailors S
```

Expr1000	age
Sailor’s age	45
Sailor’s age	35
Sailor’s age	55

Advanced SQL

- Strings in SELECT Clause (CONCAT)

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

```
SELECT CONCAT(’Sailor”s age: ’, S.age)
FROM Sailors S
```

age
Sailor’s age: 45
Sailor’s age: 35
Sailor’s age: 55

Advanced SQL

Find sid's of sailors who've reserved a blue and a green boat.

- INTERSECTIONS

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	103	4-Oct-96

Boats

B:

bid	bname	color
101	Interlak	blue
103	Clipper	green

```
SELECT  R1.sid AS sailorID
FROM  Boats B1, Reserves R1
WHERE  R1.bid=B1.bid
            AND B1.color='blue'
INTERSECT
SELECT  R2.sid
FROM  Boats B2, Reserves R2
WHERE R2.bid=B2.bid
           AND B2.color='green'
```

sailorID
22

Advanced SQL

Find sid’s of sailors who’ve reserved a blue and a green boat.

- INTERSECTIONS

```
SELECT  R1.sid
FROM  Boats B1, Reserves
R1, Boats B2, Reserves R2
WHERE  R1.sid = R2.sid
AND R1.bid=B1.bid AND
R2.bid=B2.bid  AND
B1.color=‘blue’ AND
B2.color=‘green’
```

```
SELECT  R1.sid AS sailorID
FROM  Boats B1, Reserves R1
WHERE  R1.bid=B1.bid
           AND B1.color=‘blue’
INTERSECT
SELECT  R2.sid
FROM  Boats B2, Reserves R2
WHERE R2.bid=B2.bid
          AND B2.color=‘green’
```

Advanced SQL

Find sid’s of sailors who’ve reserved a blue but never a green boat.

- EXCEPT

```
SELECT  R1.sid
FROM  Boats B1, Reserves R1,
Boats B2, Reserves R2
WHERE  R1.sid = R2.sid AND
R1.bid=B1.bid AND
R2.bid=B2.bid
  AND B1.color=‘blue’
  AND B2.color<>‘green’
```

```
SELECT  R1.sid
FROM  Boats B1, Reserves R1
WHERE  R1.bid=B1.bid
              AND B1.color=‘blue’
EXCEPT
SELECT  R2.sid
FROM  Boats B2, Reserves R2
WHERE R2.bid=B2.bid
             AND B2.color=‘green’
```

Advanced SQL
Nested Queries

Nested Queries

Find names of sailors who've reserved boat #103:

```
SELECT  S.sname
FROM  Sailors S, Reserve R
WHERE  S.sid=R.sid AND R.bid=103
```

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  (SELECT  R.sid
                  FROM  Reserves R
                  WHERE  R.bid=103)
```

Nested Queries

Find the color of the boat reserved by Sailor Dustin:

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	103	4-Oct-96

Boats

B:

bid	bname	color
101	Interlak	blue
103	Clipper	green

```
SELECT B.color
FROM Boats B
WHERE B.bid IN (SELECT  R.bid
                FROM  Reserves R
                WHERE R.sid IN  (SELECT S.sid
                                 FROM Sailors S
                                 WHERE S.sname=“Dustin”))
```

B.color
blue
green

Nested Queries

Find names of sailors who've reserved boats that are not green

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	103	4-Oct-96

Boats

B:

bid	bname	color
101	Interlak	blue
103	Clipper	green

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  (SELECT  R.sid
                  FROM  Reserves R
                  WHERE  R.bid NOT IN  (SELECT B.bid
                                        FROM Boats B
                                         WHERE B.color=“green”))
```

sname
Dustin

Nested Queries

Find boats (name) that were not reserved by Sailor S

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	103	4-Oct-96

Boats

B:

bid	bname	color
101	Interlak	blue
103	Clipper	green

```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS
                (SELECT  B.bid
                FROM  Boats B)
                EXCEPT
                 (SELECT  DISTINCT R.bid
                FROM  Reserves R
                 WHERE  R.sid=S.sid)
```

The set of boats that were Not reserved by the sailor S. Should be {}

Nested Queries

Find boats (name) that were not reserved by Sailor S

The hard way without EXCEPT

```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS
            (SELECT  B.bid
            FROM  Boats B)
            EXCEPT
             (SELECT  DISTINCT R.bid
            FROM  Reserves R
             WHERE  R.sid=S.sid)
```

```
SELECT  S.sname   - - Sailors S for whom this set is empty.
FROM  Sailors S
WHERE  NOT EXISTS  (SELECT  B.bid
                     FROM  Boats B
                  WHERE  NOT EXISTS  (SELECT  R.bid
                                       FROM  Reserves R
                                       WHERE  R.bid=B.bid
                                       AND R.sid=S.sid))
```

Advanced SQL
Nested Queries and Set -Comparison Operators

Set-Comparison Operators

Find sailors whose rating is greater than that of every sailor called Dustin:

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55
75	Dustin	8	30

```
SELECT  S1.sname
FROM  Sailors S1
WHERE  NOT EXISTS  (SELECT  *
                    FROM  Sailors S2
                    WHERE  S2.sname=‘Dustin’ AND
                    S2.rating>S1.rating)
```

```
SELECT  *
FROM  Sailors S
WHERE  S.rating > ALL (SELECT  S2.rating
                       FROM  Sailors S2
                       WHERE S2.sname=‘Dustin’)
```

op ANY, *op* ALL (where *op*: <, <=, =, <>, >=, >)

58	Rusty	10	35

Set-Comparison Operators

Find sailors whose rating is greater than some/any sailor called Dustin:

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55
75	Dustin	8	30

```
SELECT  S1.sname
FROM  Sailors S1
WHERE  NOT EXISTS  (SELECT  *
                    FROM  Sailors S2
                    WHERE  S2.sname=‘Dustin’ AND
                    S2.rating>S1.rating)
```

```
SELECT  *
FROM  Sailors S
WHERE  S.rating > ANY(SELECT  S2.rating
                      FROM  Sailors S2
                      WHERE S2.sname=‘Dustin’)
```

op ANY, *op* ALL (where *op*: <, <=, =, <>, >=, >)

58	Rusty	10	35
31	Lubber	8	55
75	Dustin	8	30

Advanced SQL
Nested Queries and Aggregate Operators

Aggregate Operators

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

```
SELECT  COUNT (*)
FROM  Sailors S
```

Number of tuples in Sailors table.

```
SELECT  COUNT (DISTINCT S.sname)
FROM  Sailors S
```

Number of distinct names in Sailors table.

```
SELECT  AVG (S.age)
FROM  Sailors S
WHERE  S.rating=10
```

Average age of Sailors with rating 10.

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.rating = (SELECT  MAX(S2.rating)
                   FROM  Sailors S2)
```

Names of all the Sailors with the highest rating.

```
SELECT  AVG ( DISTINCT S.age)
FROM  Sailors S
WHERE  S.rating=10
```

The average of all the unique ages of Sailors with rating 10.

Advanced SQL
GROUP BY and HAVING

Consider This Problem

Find the average sailor age for <u>each rating level</u>

Sailors

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55
75	Dustin	8	30

SELECT [DISTINCT] *select-list*
FROM *relation-list*
WHERE *qualification*
GROUP BY *grouping-list*

Solution

Find the average sailor age for each rating level

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55
75	Dustin	8	30

```
SELECT AVG(S.age) AS avg_age
FROM    Sailors S
GROUP BY  S.rating
```

avg_age
45
35
42.5

GROUP BY and HAVING

Find the age of the youngest sailor for each rating level,
Show the first 20 results in descending order

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	8	55
75	Dustin	8	30

```
SELECT MIN(S.age) AS min_age
FROM    Sailors S
GROUP BY  S.rating
ORDER BY min_age DESC
LIMIT 20
```

min_age
45
35
30

GROUP BY and HAVING

How many sailors in each rating and age group?

Sailors

S:

sid	sname	rating	age
22	Dustin	7	45
58	Rusty	10	35
31	Lubber	7	45
75	Dustin	8	30

```
SELECT S.rating, S.age, COUNT(*) AS numSailors
FROM    Sailors S
GROUP BY  S.rating, S.age
```

rating	age	numSailors
7	45	2
10	35	1
8	30	1

GROUP BY and HAVING

```
SELECT       [DISTINCT]  select-list
FROM           relation-list
WHERE        qualification
GROUP BY   grouping-list
HAVING      group-qualification
```

Having clause specifies condition at the group level

Find the age of a youngest sailor who is eligible to vote (age >= 18), for each rating level with at least 2 such sailors

```
SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
```

rating	
7	35.0

Sailors

S:

~~sid~~	~~sname~~	rating	age
~~22~~	~~dustin~~	7	45.0
~~31~~	~~lubber~~	8	55.5
~~71~~	~~Zorba~~	~~10~~	~~16.0~~
~~64~~	~~Horatio~~	7	35.0
~~29~~	~~brutus~~	1	33.0
~~58~~	~~Rusty~~	10	35.0

rating	age
~~1~~	~~33.0~~
7	45.0
7	35.0
~~8~~	~~55.5~~
~~10~~	~~35.0~~

Step 1. Cross product of tables in the from-list.
Step 2. Eliminate tuples that do not meet the WHERE clause.
Step 3: Delete attributes that are not in select-list, Group by, or Having clause.
Step 4: Identify the groups according to the Group By clause.
Step 5: Eliminate the groups that do not meet the group-qualification in the Having clause.
Step 6. Generate one answer row for each remaining group.

GROUP BY And HAVING

For sailors above age 20, find the average age of the sailors for each rating level.

```
SELECT      S.rating, AVG(S.age) AS AvgAge
FROM        Sailors S
GROUP BY    S.rating
HAVING      S.age > 20
```

WRONG!!!

```
SELECT      S.rating, AVG(S.age) AS AvgAge
FROM        Sailors S
WHERE       S.age > 20
GROUP BY    S.rating
```

GOOD!!!

Advanced SQL
JOINS

SELECT With JOINS

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	103	4-Oct-96

Boats

B:

bid	bname	color
101	Interlak	blue
103	Clipper	green

```
SELECT  boat.bid, boat.color
FROM    Reserves reserve
JOIN  Boats boat ON reserve.bid = boat.bid
ORDER BY boat.bid ASC
```

bid	color
101	blue
101	blue
103	green
103	green

JOINS

- Method for linking data between one or more relations.
- Better performance than nested queries?
  - SQL Joins
    - INNER JOIN
    - LEFT JOIN
    - RIGHT JOIN
    - Some DBMS also provide support for CROSS JOIN

INNER JOIN

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	103	4-Oct-96

Boats

B:

bid	bname	color
101	Interlak	blue
103	Clipper	green

```
SELECT       boat.bid, boat.color
FROM         Reserves reserve
INNER JOIN  Boats boat
ON reserve.bid = boat.bid
ORDER BY boat.bid ASC
```

bid	color
101	blue
101	blue
103	green
103	green

LEFT JOIN

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	104	4-Oct-96

Boats

B:

bid	bname	color
101	Interlak	blue
103	Clipper	green

```
SELECT       boat.bid, boat.color
FROM         Reserves reserve
LEFT JOIN  Boats boat
ON reserve.bid = boat.bid
```

bid	Color
101	blue
103	green
104	NULL

RIGHT JOIN

Reserves

R:

sid	bid	Day
22	101	10-Oct-96
58	103	12-Nov-96
22	104	4-Oct-96

Boats

B:

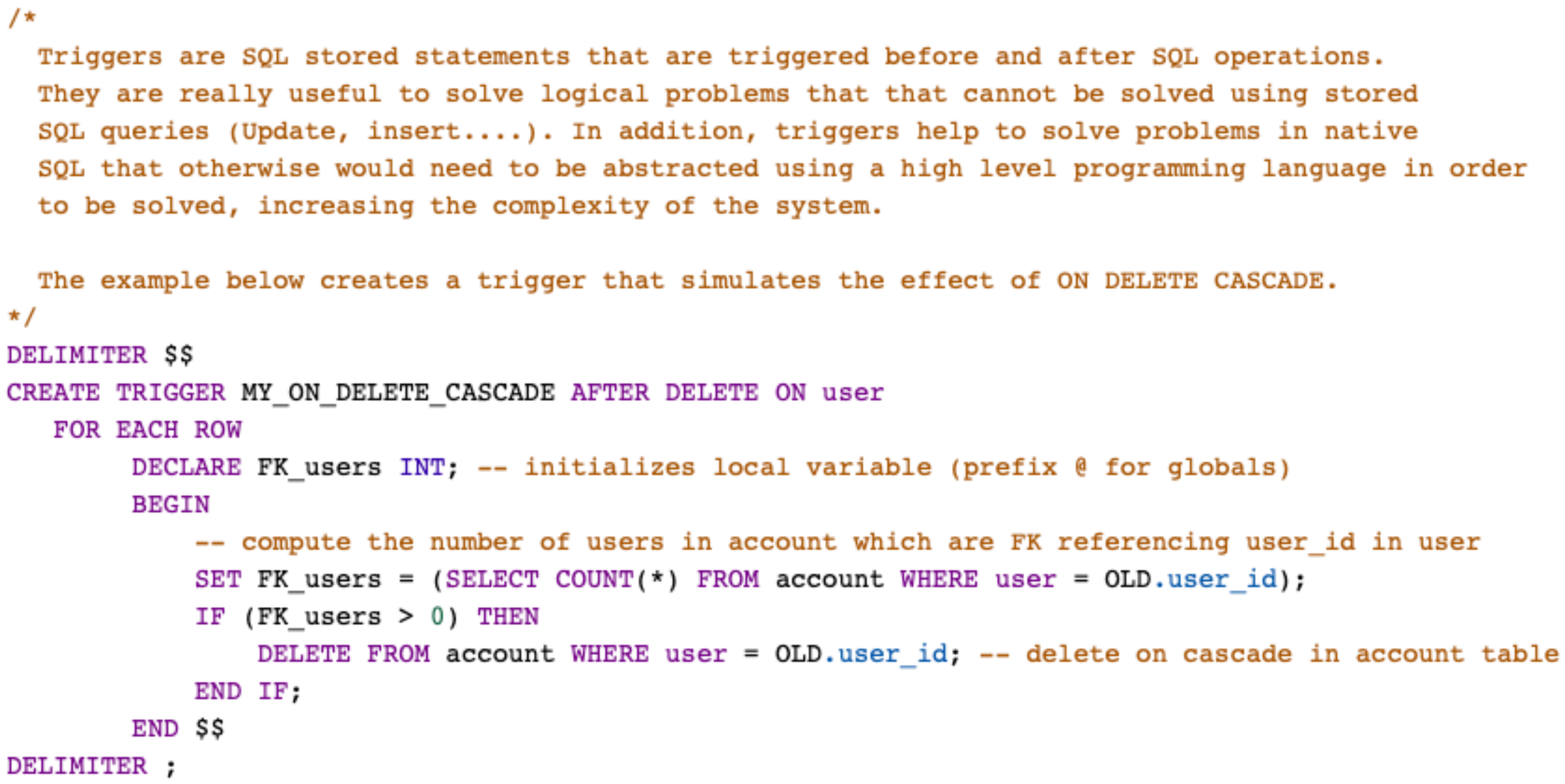
bid	bname	color
101	Interlak	blue
103	Clipper	green
105	Hercules	White

```
SELECT      boat.bid, boat.color
FROM        Reserves reserve
RIGHT JOIN  Boats boat
ON reserve.bid = boat.bid
```

bid	Color
101	blue
103	green
NULL	white

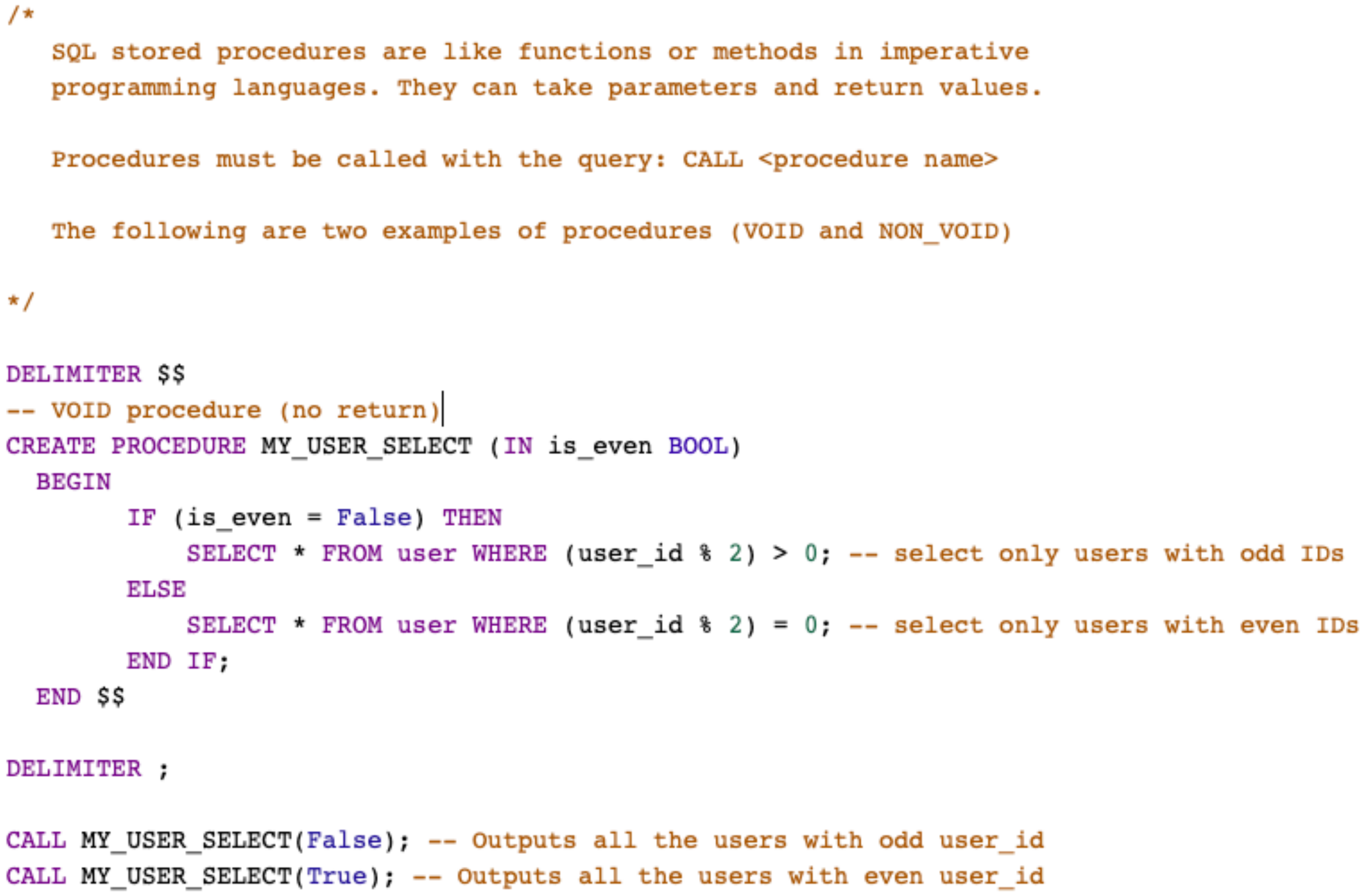
Triggers and Procedures

Triggers



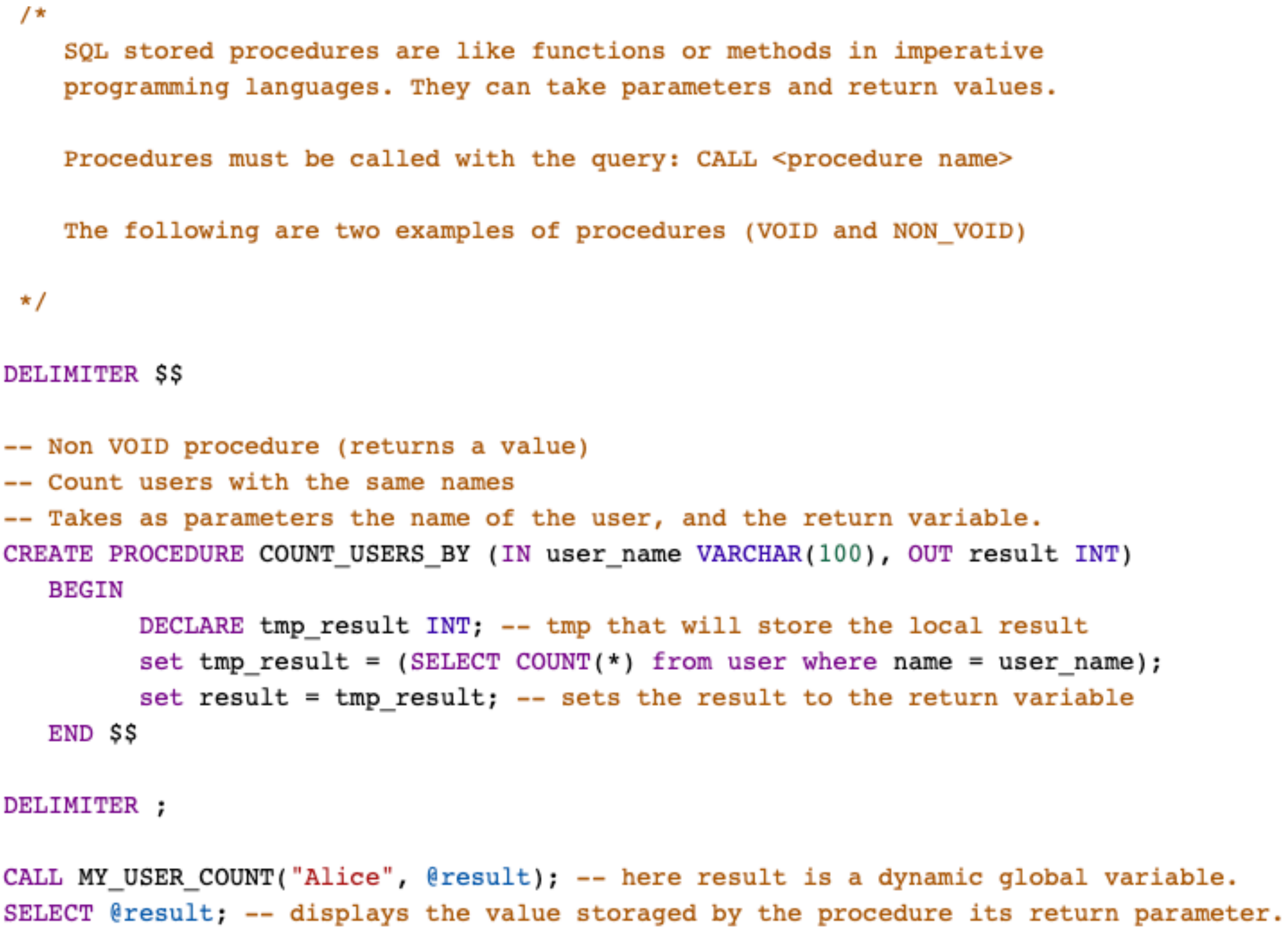
```
/*
  Triggers are SQL stored statements that are triggered before and after SQL operations.
  They are really useful to solve logical problems that that cannot be solved using stored
  SQL queries (Update, insert....). In addition, triggers help to solve problems in native
  SQL that otherwise would need to be abstracted using a high level programming language in order
  to be solved, increasing the complexity of the system.

  The example below creates a trigger that simulates the effect of ON DELETE CASCADE.
*/
DELIMITER $$
CREATE TRIGGER MY_ON_DELETE_CASCADE AFTER DELETE ON user
   FOR EACH ROW
        DECLARE FK_users INT; -- initializes local variable (prefix @ for globals)
        BEGIN
            -- compute the number of users in account which are FK referencing user_id in user
            SET FK_users = (SELECT COUNT(*) FROM account WHERE user = OLD.user_id);
            IF (FK_users > 0) THEN
                DELETE FROM account WHERE user = OLD.user_id; -- delete on cascade in account table
            END IF;
        END $$
DELIMITER ;
```

SQL Stored Procedures (VOID)



```
/*
   SQL stored procedures are like functions or methods in imperative
   programming languages. They can take parameters and return values.

   Procedures must be called with the query: CALL <procedure name>

   The following are two examples of procedures (VOID and NON_VOID)

*/

DELIMITER $$
-- VOID procedure (no return)
CREATE PROCEDURE MY_USER_SELECT (IN is_even BOOL)
  BEGIN
        IF (is_even = False) THEN
            SELECT * FROM user WHERE (user_id % 2) > 0; -- select only users with odd IDs
        ELSE
            SELECT * FROM user WHERE (user_id % 2) = 0; -- select only users with even IDs
        END IF;
  END $$

DELIMITER ;

CALL MY_USER_SELECT(False); -- Outputs all the users with odd user_id
CALL MY_USER_SELECT(True); -- Outputs all the users with even user_id
```

SQL Stored Procedures (Return Value)



```
 /*
    SQL stored procedures are like functions or methods in imperative
    programming languages. They can take parameters and return values.

    Procedures must be called with the query: CALL <procedure name>

    The following are two examples of procedures (VOID and NON_VOID)

 */

DELIMITER $$

-- Non VOID procedure (returns a value)
-- Count users with the same names
-- Takes as parameters the name of the user, and the return variable.
CREATE PROCEDURE COUNT_USERS_BY (IN user_name VARCHAR(100), OUT result INT)
   BEGIN
         DECLARE tmp_result INT; -- tmp that will store the local result
         set tmp_result = (SELECT COUNT(*) from user where name = user_name);
         set result = tmp_result; -- sets the result to the return variable
   END $$

DELIMITER ;

CALL MY_USER_COUNT("Alice", @result); -- here result is a dynamic global variable.
SELECT @result; -- displays the value storaged by the procedure its return parameter.
```