CSIT115 Data Management and Security

# SELECT Statement (3)

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

# SELECT statement (3)

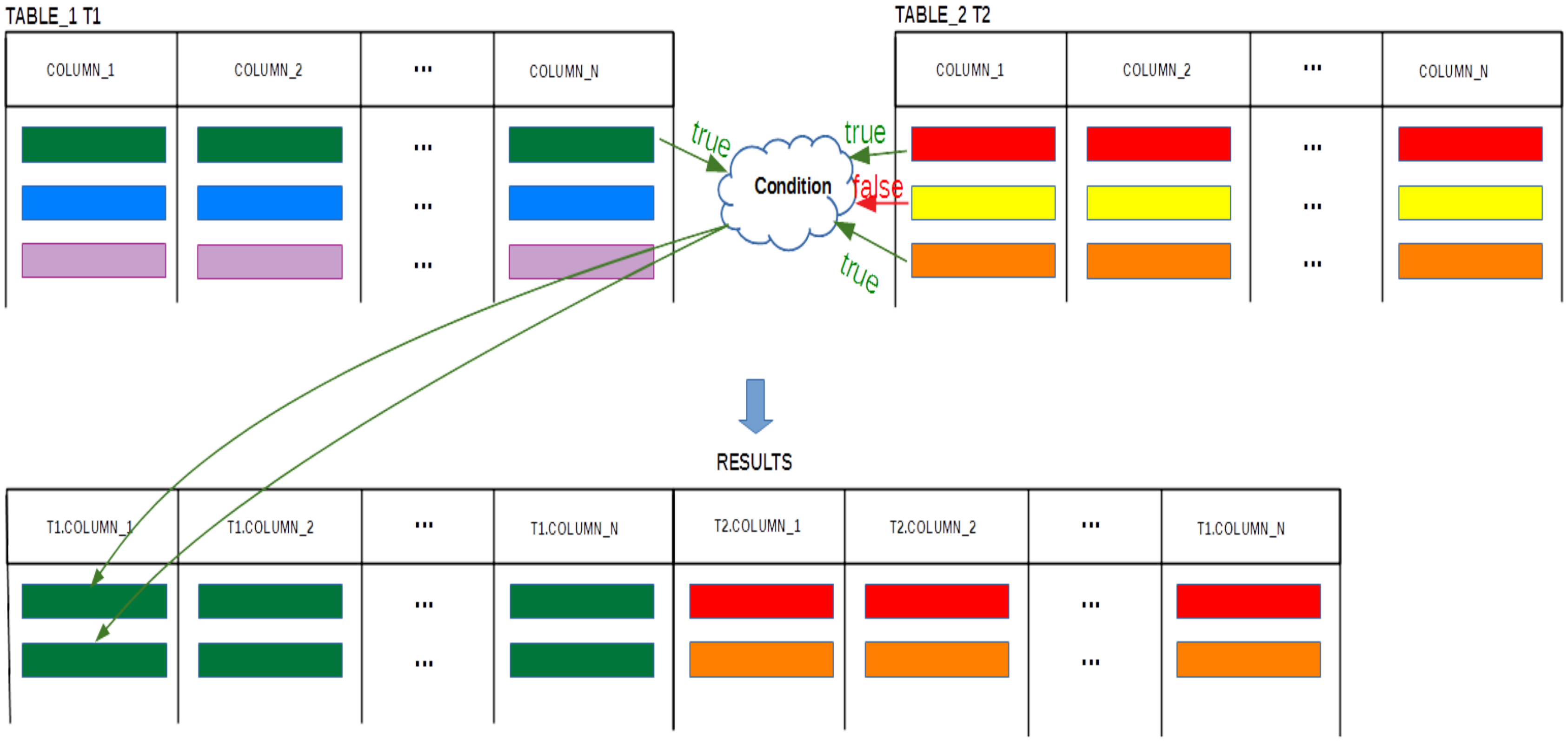
## Outline

- Join queries
- Natural join queries
- Column name join queries
- Cross join queries
- Join queries over more than 2 tables
- Self-join queries

# Join queries

Join operation "connects" the rows from two relational tables

```
SELECT *
FROM TABLE_1 T1 JOIN TABLE_2 T2
ON Condition;
```



# Join queries

Join operation "connects" the rows from two relational tables

```
SELECT *
FROM TABLE_1 T1 JOIN TABLE_2 T2
ON Condition;
```

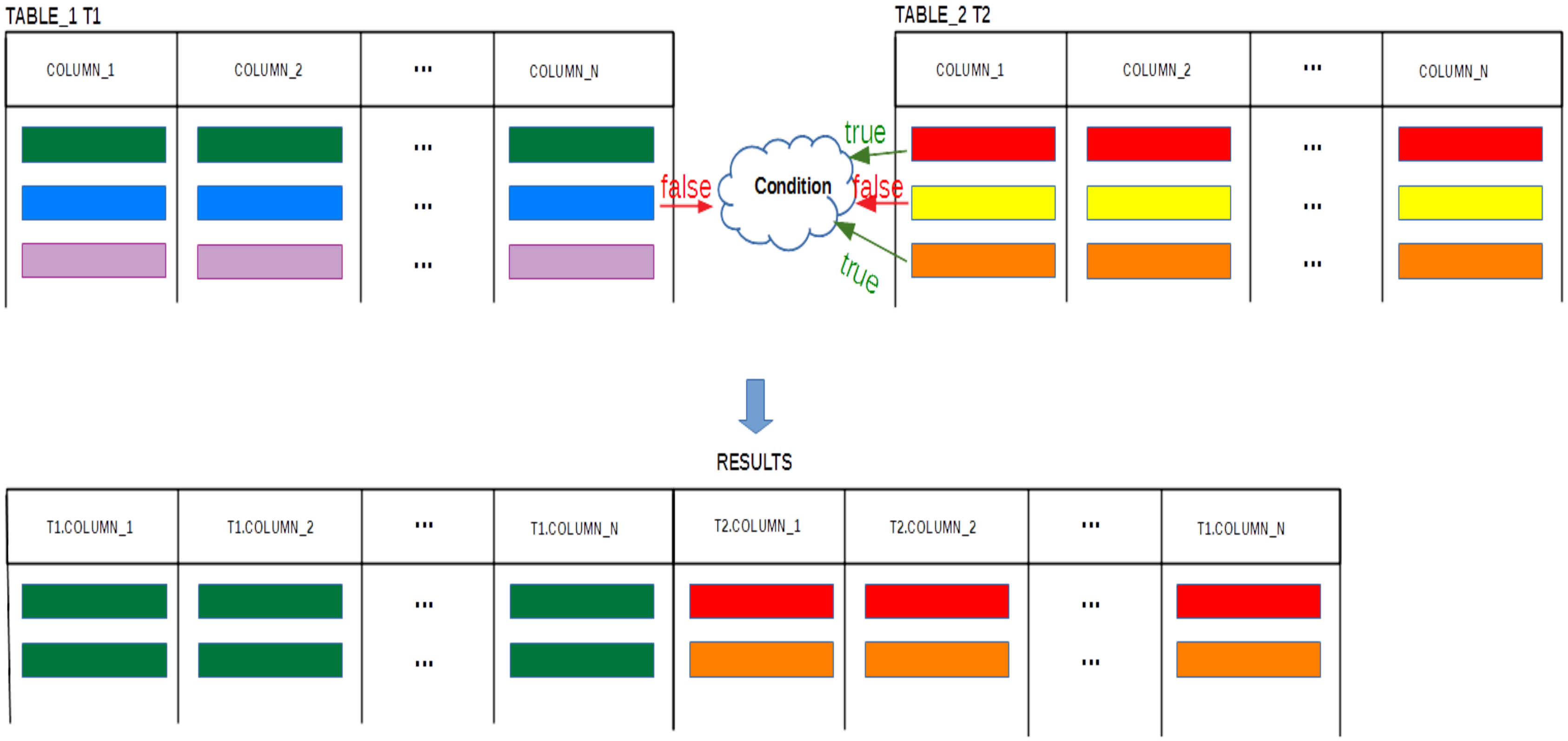


TABLE_1 T1

| COLUMN_1 | COLUMN_2 | ... | COLUMN_N |
|---|---|---|---|
| 🟩 | 🟩 | ... | 🟩 |
| 🟦 | 🟦 | ... | 🟦 |
| 🟪 | 🟪 | ... | 🟪 |

TABLE_2 T2

| COLUMN_1 | COLUMN_2 | ... | COLUMN_N |
|---|---|---|---|
| 🟥 | 🟥 | ... | 🟥 |
| 🟨 | 🟨 | ... | 🟨 |
| 🟧 | 🟧 | ... | 🟧 |

RESULTS

| T1.COLUMN_1 | T1.COLUMN_2 | ... | T1.COLUMN_N | T2.COLUMN_1 | T2.COLUMN_2 | ... | T1.COLUMN_N |
|---|---|---|---|---|---|---|---|
| 🟩 | 🟩 | ... | 🟩 | 🟥 | 🟥 | ... | 🟥 |
| 🟩 | 🟩 | ... | 🟩 | 🟧 | 🟧 | ... | 🟧 |

# Join queries

Join operation "connects" the rows from two relational tables

```
SELECT *
FROM TABLE_1 T1 JOIN TABLE_2 T2
ON Condition;
```

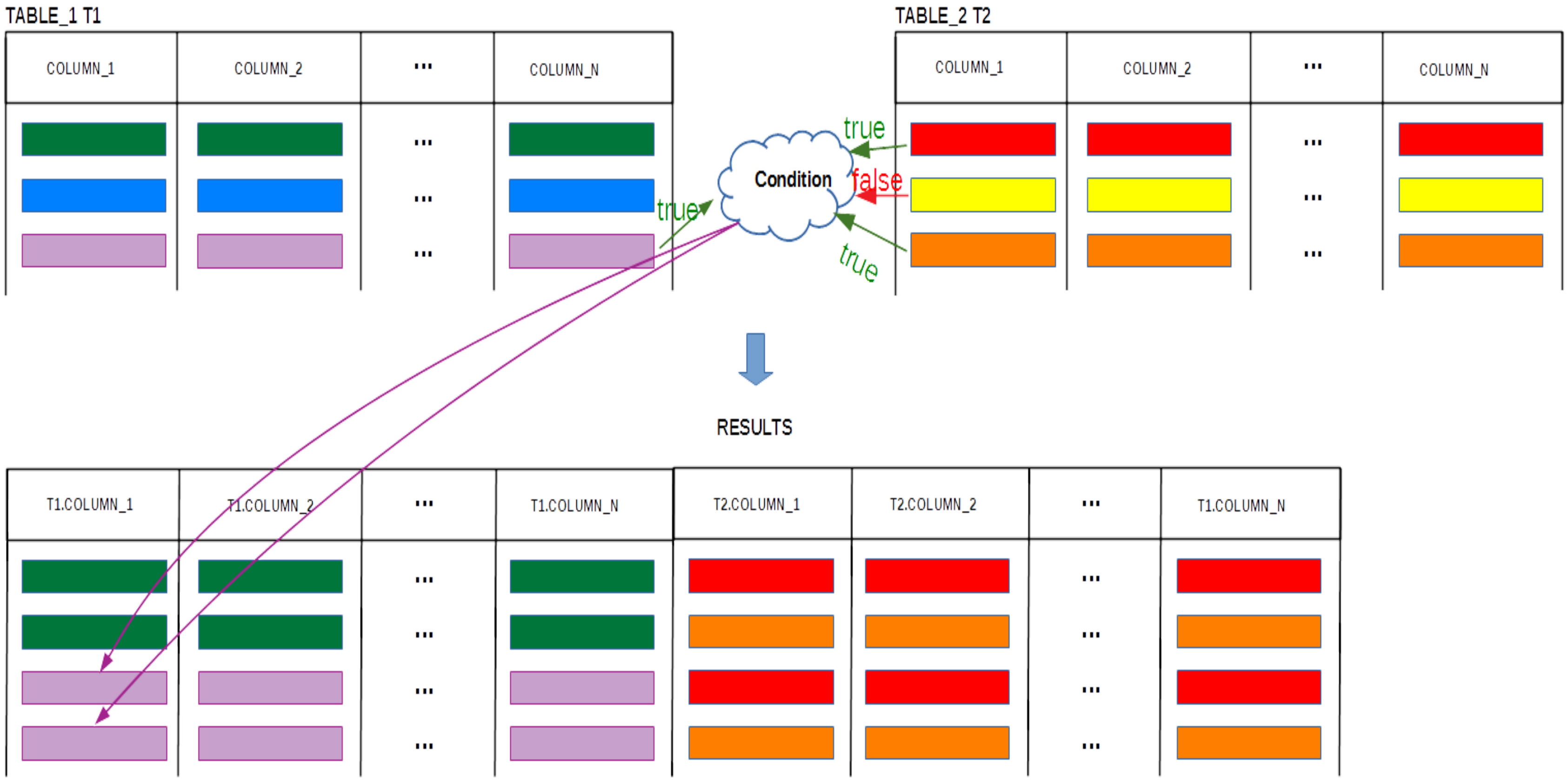


TABLE_1 T1

| COLUMN_1 | COLUMN_2 | ... | COLUMN_N |
|---|---|---|---|
| 🟩 | 🟩 | ... | 🟩 |
| 🟦 | 🟦 | ... | 🟦 |
| 🟪 | 🟪 | ... | 🟪 |

TABLE_2 T2

| COLUMN_1 | COLUMN_2 | ... | COLUMN_N |
|---|---|---|---|
| 🟥 | 🟥 | ... | 🟥 |
| 🟨 | 🟨 | ... | 🟨 |
| 🟧 | 🟧 | ... | 🟧 |

RESULTS

| T1.COLUMN_1 | T1.COLUMN_2 | ... | T1.COLUMN_N | T2.COLUMN_1 | T2.COLUMN_2 | ... | T1.COLUMN_N |
|---|---|---|---|---|---|---|---|
| 🟩 | 🟩 | ... | 🟩 | 🟥 | 🟥 | ... | 🟥 |
| 🟩 | 🟩 | ... | 🟩 | 🟧 | 🟧 | ... | 🟧 |
| 🟪 | 🟪 | ... | 🟪 | 🟥 | 🟥 | ... | 🟥 |
| 🟪 | 🟪 | ... | 🟪 | 🟧 | 🟧 | ... | 🟧 |

# Join queries

## Sample database

CREATE TABLE statement

```
CREATE TABLE DEPARTMENT(
 name              VARCHAR(50)        NOT NULL,
 code              CHAR(5)            NOT NULL,
 total_staff_number DECIMAL(2)        NOT NULL,
 chair             VARCHAR(50)            NULL,
 budget            DECIMAL(9,1)       NOT NULL,
  CONSTRAINT dept_pkey PRIMARY KEY(name),
  CONSTRAINT dept_ckey1 UNIQUE(code),
  CONSTRAINT dept_ckey2 UNIQUE(chair),
  CONSTRAINT dept_check1 CHECK (total_staff_number BETWEEN 1 AND 50) );
```

CREATE TABLE statement

```
CREATE TABLE COURSE(
 cnum              CHAR(7)           NOT NULL,
 title             VARCHAR(200)      NOT NULL,
 credits           DECIMAL(2)        NOT NULL,
 offered_by        VARCHAR(50)           NULL,
  CONSTRAINT course_pkey PRIMARY KEY(cnum),
  CONSTRAINT course_check1 CHECK (credits IN (6, 12)),
  CONSTRAINT course_fkey1 FOREIGN KEY(offered_by)
                        REFERENCES DEPARTMENT(name) ON DELETE CASCADE );
```

# Join queries

Consider the following query: Find the titles of all courses offered by a department chaired by Peter

Relational schema

```
DEPARTMENT
name | code | total_staff_number | chair | budget
```

- There are no titles of courses in a relational table `DEPARTMENT` !
- The titles of courses are in a relational table `COURSE`

Relational schema

```
COURSE
cnum | title | credits | offered_by
```

- To implement the query we must use two tables: `DEPARTMENT` and `COURSE`
- The rows from a table `DEPARTMENT` must be joined with (connected to) the respective rows in a relational table `COURSE` over a condition `DEPARTMENT.name = COURSE.offered_by`

SELECT statement with JOIN operation

```sql
SELECT COURSE.title
FROM COURSE JOIN DEPARTMENT
            ON DEPARTMENT.name = COURSE.offered_by
WHERE DEPARTMENT.chair = 'Peter';
```

# Join queries

Implementation of the query Find the titles of all courses offered by a department chaired by Peter has the following syntactical variations

SELECT statement with JOIN operation

```
SELECT title
FROM COURSE JOIN DEPARTMENT
            ON name = offered_by
WHERE chair = 'Peter';
```

SELECT statement with JOIN operation

```
SELECT C.title
FROM COURSE C JOIN DEPARTMENT D
            ON D.name = C.offered_by
WHERE D.chair = 'Peter';
```

SELECT statement with JOIN operation

```
SELECT COURSE.title
FROM COURSE, DEPARTMENT
WHERE DEPARTMENT.name = COURSE.offered_by AND DEPARTMENT.chair = 'Peter';
```

SELECT statement with JOIN operation

```
SELECT title
FROM COURSE, DEPARTMENT
WHERE name = offered_by AND chair = 'Peter';
```

# ANSI SQL Syntax

The following implementations of the query Find the titles of all courses offered by a department chaired by Peter are consistent with ANSI SQL standard

SELECT statement with JOIN operation (ANSI SQL standard)

```
SELECT COURSE.title
FROM COURSE JOIN DEPARTMENT
            ON DEPARTMENT.name = COURSE.offered_by
WHERE DEPARTMENT.chair = 'Peter';
```

SELECT statement with JOIN operation (ANSI SQL standard)

```
SELECT title
FROM COURSE JOIN DEPARTMENT
            ON name = offered_by
WHERE chair = 'Peter';
```

SELECT statement with JOIN operation (ANSI SQL standard)

```
SELECT C.title
FROM COURSE C JOIN DEPARTMENT D
            ON D.name = C.offered_by
WHERE D.chair = 'Peter';
```

# SELECT statement (3)

## Outline

- Join queries
- Natural join queries
- Column name join queries
- Cross join queries
- Join queries over more than 2 tables
- Self-join queries

# Natural join queries

Consider a query: Find the names of all employees from a department chaired by James Bond over the relational tables

Relational schema

```
DEPARTMENT
                    dname | code | total staff number | chair | budget
```

Relational schema

```
EMPLOYEE
                    enum | ename | dname
```

A natural join query

SELECT statement with NATURAL JOIN operation

```
SELECT ename
FROM EMPLOYEE NATURAL JOIN DEPARTMENT
WHERE chair = 'James Bond';
```

- is equivalent to a join query

SELECT statement with JOIN operation (equivalent to a statment above)

```
SELECT ename
FROM EMPLOYEE JOIN DEPARTMENT
              ON EMPLOYEE.dname = DEPARTMENT.dname
WHERE chair = 'James Bond';
```

# SELECT statement (3)

## Outline

- Join queries
- Natural join queries
- Column name join queries
- Cross join queries
- Join queries over more than 2 tables
- Self-join queries

# Column name join queries

Consider a query: Find the names of all employees from a department chaired by James Bond over the relational tables

Relational schema

```
DEPARTMENT
                         dname | code | total staff number | chair | budget
```

Relational schema

```
EMPLOYEE
                         enum | ename | dname
```

A column name join query

SELECT statement with column name JOIN operation

```
SELECT ename
FROM EMPLOYEE JOIN DEPARTMENT
              USING(dname)
WHERE chair = 'James Bond';
```

- is equivalent to a join query

SELECT statement with JOIN operation (equivalent to a statement above)

```
SELECT ename
FROM EMPLOYEE JOIN DEPARTMENT
              ON  EMPLOYEE.dname = DEPARTMENT.dname
WHERE chair = 'James Bond';
```

# SELECT statement (3)

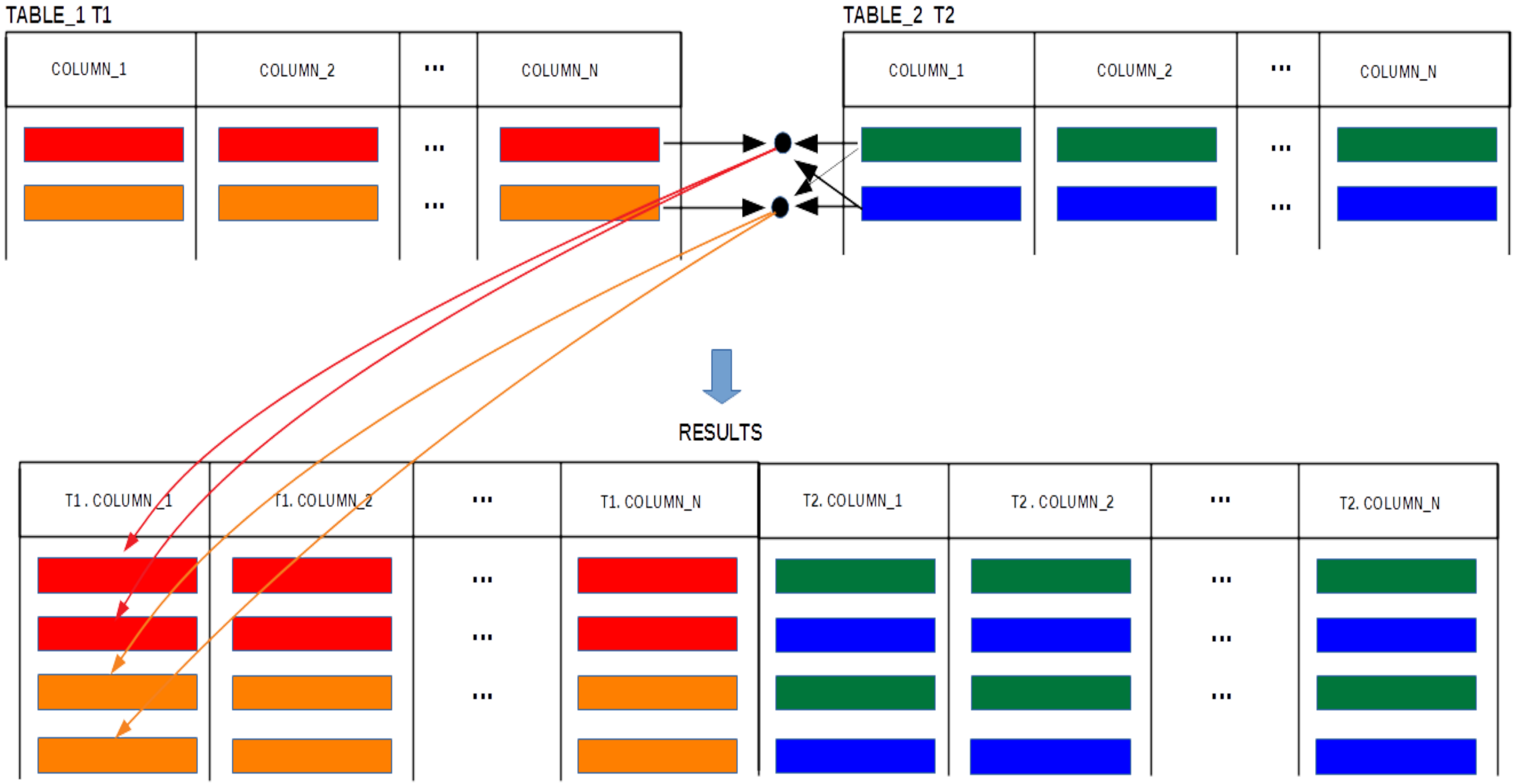
## Outline

- Join queries
- Natural join queries
- Column name join queries
- Cross join queries
- Join queries over more than 2 tables
- Self-join queries

# Cross join queries

Cross join operation "connects" all rows from a relational table with all rows from another relational table

```
SELECT *
FROM TABLE_1  T1 CROSS JOIN TABLE_2 T2;
```



# Cross join queries

Consider a query: Find all pairs of the names of employees and the names of chair people over the relational tables

```
DEPARTMENT                                                          Relational schema
                         dname | code | total staff number | chair | budget
```

```
EMPLOYEE                                                            Relational schema
                         enum | ename | dname
```

A cross join query

SELECT statement with CROSS JOIN operations

```
SELECT ename, chair
FROM EMPLOYEE CROSS JOIN DEPARTMENT;
```

- is equivalent to the following join queries

SELECT statement equivalent to a statement above

```
SELECT ename, chair
FROM EMPLOYEE JOIN DEPARTMENT;
```

SELECT statement equivalent to a statement above

```
SELECT ename, chair
FROM EMPLOYEE, DEPARTMENT;
```

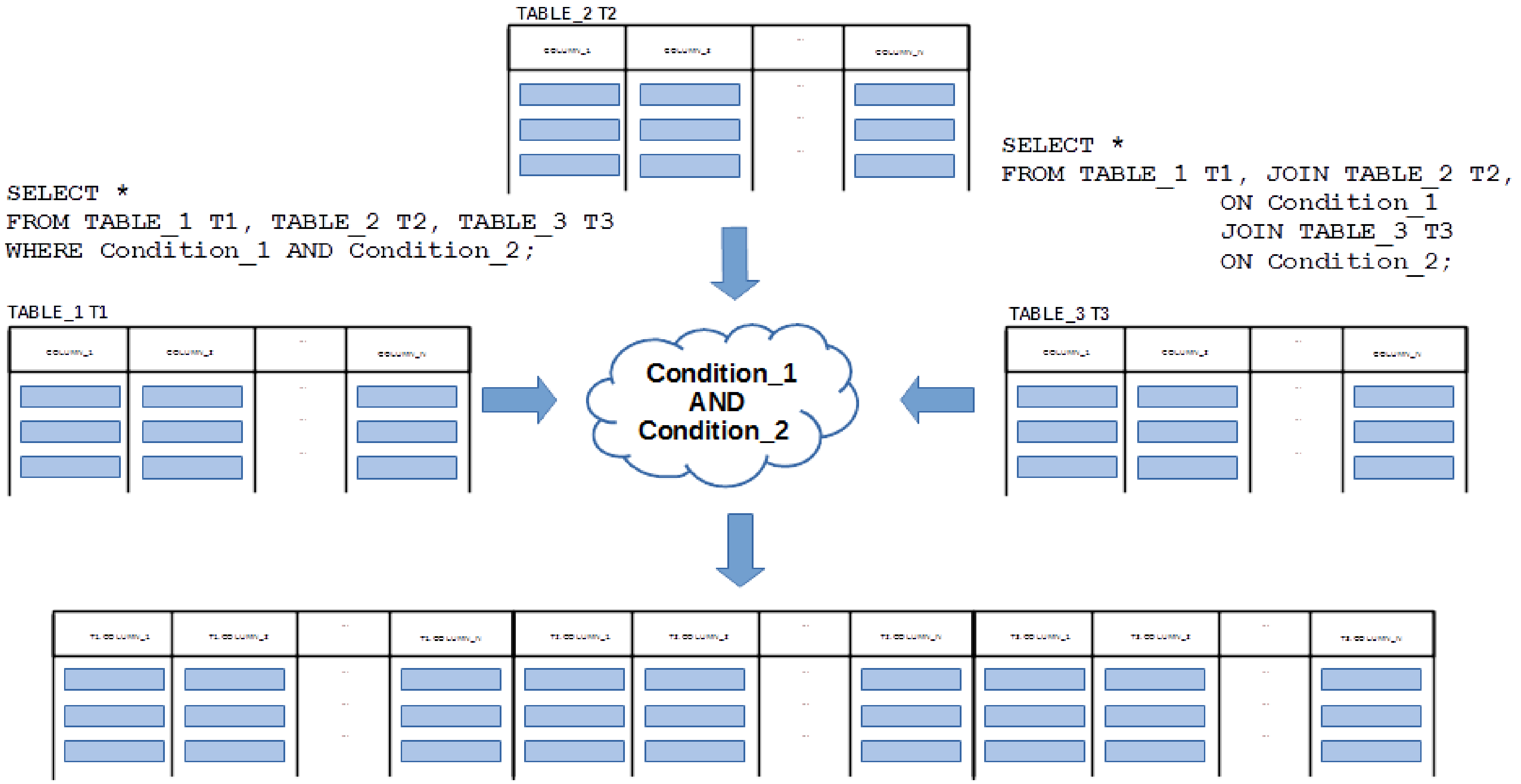
# SELECT statement (3)

## Outline

- Join queries
- Natural join queries
- Column name join queries
- Cross join queries
- Join queries over more than 2 tables
- Self-join queries

# Join queries over more than 2 tables

A sample join of three relational tables `TABLE_1`, `TABLE_2`, and `TABLE_3`

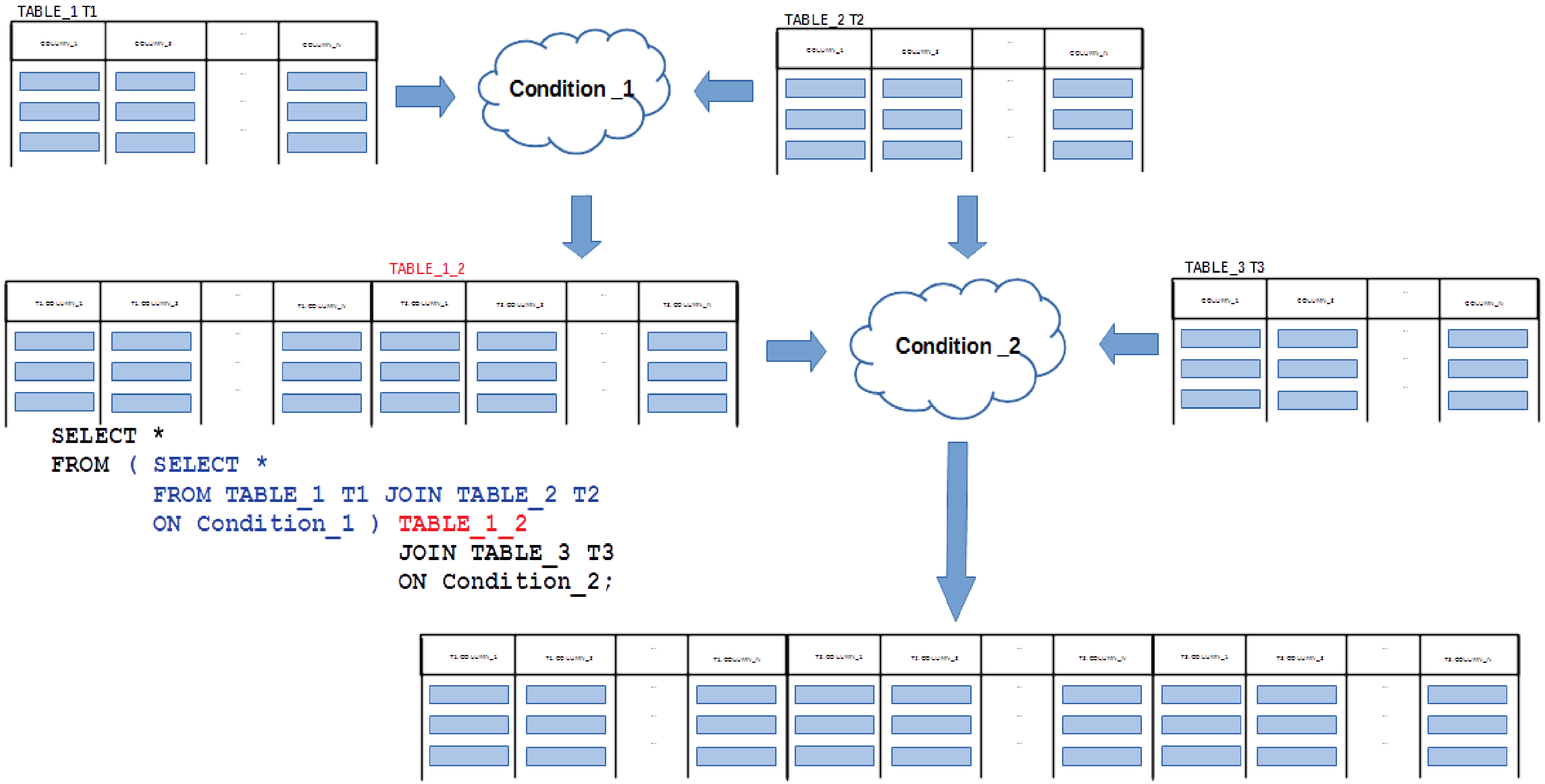


```
SELECT *
FROM TABLE_1 T1, TABLE_2 T2, TABLE_3 T3
WHERE Condition_1 AND Condition_2;
```

```
SELECT *
FROM TABLE_1 T1, JOIN TABLE_2 T2,
                 ON Condition_1
                 JOIN TABLE_3 T3
                 ON Condition_2;
```

# Join queries over more than 2 tables

Application of inline view to simplify join of three relational tables `TABLE_1`, `TABLE_2`, and `TABLE_3`



```
SELECT *
FROM ( SELECT *
       FROM TABLE_1 T1 JOIN TABLE_2 T2
       ON Condition_1 ) TABLE_1_2
                        JOIN TABLE_3 T3
                        ON Condition_2;
```

# Join queries over more than 2 tables

Consider the relational tables with the following schemas

Relational schema

```
COURSE
       cnum | title | credits
```

Relational schema

```
STUDENT
       snum | name | degree
```

Relational schema

```
ENROLMENT
       cnum | snum | edate | result
```

A query find the names of all students who enrolled Java course can be implemented as the following join query

SELECT statement that joins three relational tables

```sql
SELECT STUDENT.name
FROM COURSE JOIN ENROLMENT
            ON COURSE.cnum = ENROLMENT.cnum
            JOIN STUDENT
            ON ENROLMENT.snum = STUDENT.snum
WHERE COURSE.title = 'Java';
```

# Join queries over more than 2 tables

Implementation of a query find the names of all students who enrolled Java course has the following syntactical variations

SELECT statement that joins three relational tables

```
SELECT STUDENT.name
FROM COURSE JOIN ENROLMENT
            ON COURSE.cnum = ENROLMENT.cnum
            JOIN STUDENT
            ON ENROLMENT.snum = STUDENT.snum
WHERE COURSE.title = 'Java';
```

SELECT statement that joins three relational tables

```
SELECT STUDENT.name
FROM COURSE, ENROLMENT, STUDENT
WHERE COURSE.cnum = ENROLMENT.cnum AND ENROLMENT.snum = STUDENT.snum AND
       COURSE.title = 'Java';
```

SELECT statement that joins three relational tables

```
SELECT STUDENT.name
FROM ( SELECT *
       FROM COURSE JOIN ENROLMENT
                   ON COURSE.cnum = ENROLMENT.cnum
       WHERE COURSE.title = 'Java' ) CE JOIN STUDENT
                                        ON CE.snum = STUDENT.snum
WHERE COURSE.title = 'Java';
```

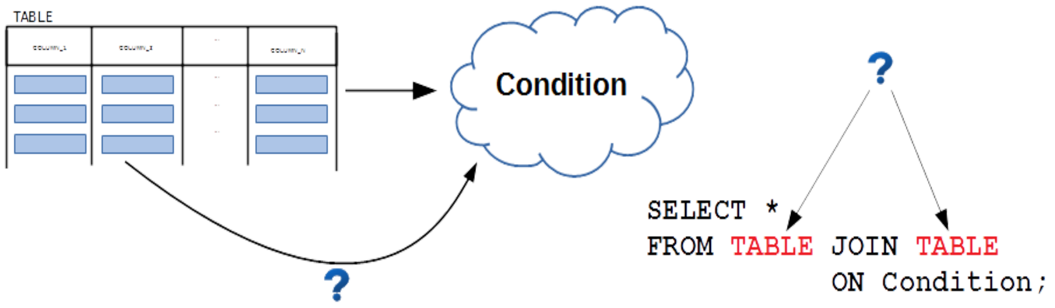
# SELECT statement (3)

## Outline

- Join queries
- Natural join queries
- Column name join queries
- Cross join queries
- Join queries over more than 2 tables
- Self-join queries

# Self-join queries

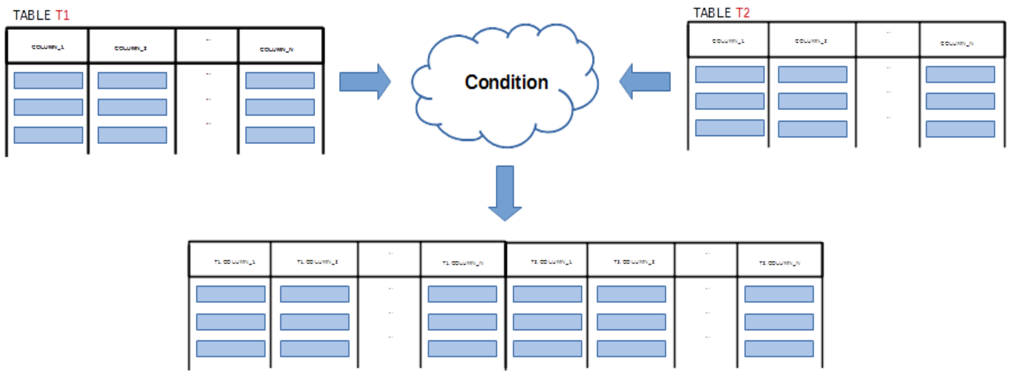
What if a relational table must be joined with itself ?



```
SELECT *
FROM TABLE JOIN TABLE
          ON Condition;
```

A table that must be joined with itself obtains two different alias names

```
SELECT *
FROM TABLE T1 JOIN TABLE T2
ON Condition ;
```



# Self-join queries

Consider a relational table with the following schema

Relational schema

```
EMPLOYEE
        enum, name, manager
```

and the following contents

Relational table

```
+------+-------+---------+
| enum | name  | manager |
+------+-------+---------+
|   10 | John  |    NULL |
|   20 | Peter |      10 |
|   30 | Mary  |      10 |
|   40 | Mike  |      20 |
|   50 | Kate  |      20 |
|   60 | Greg  |      50 |
|   70 | Phil  |      50 |
+------+-------+---------+
```

Consider a query find a name of manager of employee number 40

We can "plan" the implementation in the following way

- (1) Find an employee number of manager of employee number 40
- (2) Find a name of employee found in the previous query

# Self-join queries

Implementation of a plan

(1) Find employee number of manager of employee number 40

(2) Find a name of employee found in the previous query

is the following

SELECT statement

```
SELECT manager
FROM EMPLOYEE
WHERE enum = 40;
```

Result

```
20
```

SELECT statement

```
SELECT name
FROM EMPLOYEE
WHERE enum = 20;
```

Result

```
Peter
```

Is it possible to implement the query as one `SELECT` statement ?

YES ! In more than one way !

# Self-join queries

Solution 1

Assume that we have two identical relational tables E1 and E2

Relational tables

```
E1                                     E2
+------+-------+---------+             +------+-------+---------+
| enum | name  | manager |             | enum | name  | manager |
+------+-------+---------+             +------+-------+---------+
|   10 | John  |    NULL |             |   10 | John  |    NULL |
|   20 | Peter |      10 |    -------->|   20 | Peter |      10 |
|   30 | Mary  |      10 |    |        |   30 | Mary  |      10 |
|   40 | Mike  |      20 |-----        |   40 | Mike  |      20 |
|   50 | Kate  |      20 |             |   50 | Kate  |      20 |
|   60 | Greg  |      50 |             |   60 | Greg  |      50 |
|   70 | Phil  |      50 |             |   70 | Phil  |      50 |
+------+-------+---------+             +------+-------+---------+
```

To find a name of manager of employee number 40 we take a row

Row

```
|   40 | Mike  |      20 |
```

- from a table E1 and we join over a condition E1.manager = E2.enum with a row

Row

```
|   20 | Peter |      10 |
```

- from a table E2

# Self-join queries

So, how do we implement such "magic" ?

SELECT statement with self-join

```
SELECT E2.name
FROM EMPLOYEE E1 JOIN EMPLOYEE E2
                 ON E1.manager = E2.enum
WHERE E1.enum = 40;
```

Relational tables

```
E1                                     E2
+------+-------+---------+             +------+-------+---------+
| enum | name  | manager |             | enum | name  | manager |
+------+-------+---------+             +------+-------+---------+
|   10 | John  |    NULL |             |   10 | John  |    NULL |
|   20 | Peter |      10 |    -------->|   20 | Peter |      10 |
|   30 | Mary  |      10 |    |        |   30 | Mary  |      10 |
|   40 | Mike  |      20 |-----        |   40 | Mike  |      20 |
|   50 | Kate  |      20 |             |   50 | Kate  |      20 |
|   60 | Greg  |      50 |             |   60 | Greg  |      50 |
|   70 | Phil  |      50 |             |   70 | Phil  |      50 |
+------+-------+---------+             +------+-------+---------+
```

# Self-join queries

## Solution 2

We use inline views technique to combine the following queries

SELECT statement

```
SELECT manager
FROM EMPLOYEE
WHERE enum = 40;
```

SELECT statement

```
SELECT name
FROM EMPLOYEE
WHERE enum = 20;
```

We use the first SELECT statement to create an inline view E40

Inline view

```
( SELECT manager
FROM EMPLOYEE
WHERE enum = 40 ) E40
```

Then we join an inline view E40 with a relational table EMPLOYEE

SELECT statement with inline view

```
SELECT EMPLOYEE.name
FROM EMPLOYEE JOIN ( SELECT manager
                     FROM EMPLOYEE
                     WHERE enum = 40 ) E40
              ON EMPLOYEE.enum = E40.manager;
```

# Self-join queries

In another query we find the names of all employees directly managed by Kate

Relational table

```
+------+-------+---------+
| enum | name  | manager |
+------+-------+---------+
|   10 | John  |    NULL |
|   20 | Peter |      10 |
|   30 | Mary  |      10 |
|   40 | Mike  |      20 |
|   50 | Kate  |      20 |
|   60 | Greg  |      50 |
|   70 | Phil  |      50 |
+------+-------+---------+
```

We "plan" the implementation in the following way

- (1) Find an employee number of an employee Kate
- (2) Find the names of employees who have a number found in the previous query in a column `manager`

# Self-join queries

Solution 1

Assume that we have two identical relational tables E1 and E2

Relational tables

```
E1                                     E2

+------+-------+---------+             +------+-------+---------+
| enum | name  | manager |             | enum | name  | manager |
+------+-------+---------+             +------+-------+---------+
|   10 | John  |    NULL |             |   10 | John  |    NULL |
|   20 | Peter |      10 |             |   20 | Peter |      10 |
|   30 | Mary  |      10 |             |   30 | Mary  |      10 |
|   40 | Mike  |      20 |             |   40 | Mike  |      20 |
|   50 | Kate  |      20 |    -------- |   50 | Kate  |      20 |
|   60 | Greg  |      50 |<---|        |   60 | Greg  |      50 |
|   70 | Phil  |      50 |<---|        |   70 | Phil  |      50 |
+------+-------+---------+             +------+-------+---------+
```

To find a number of employee Kate we take a row

Row

```
|   50 | Kate  |      20 |
```

- from a table E2 and we join it over a condition E2.enum = E1.manager with the rows

Row

```
|   60 | Greg  |      50 |
```

Row

```
|   70 | Phil  |      50 |
```

- from a table E1

# Self-join queries

So, how do we implement such "magic" ?

SELECT statement with self-join

```
SELECT E1.name
FROM EMPLOYEE E1 JOIN EMPLOYEE E2
                 ON E1.manager = E2.enum
WHERE E2.name = 'Kate';
```

Relational tables

```
E1                                     E2
+------+-------+---------+             +------+-------+---------+
| enum | name  | manager |             | enum | name  | manager |
+------+-------+---------+             +------+-------+---------+
|   10 | John  |    NULL |             |   10 | John  |    NULL |
|   20 | Peter |      10 |             |   20 | Peter |      10 |
|   30 | Mary  |      10 |             |   30 | Mary  |      10 |
|   40 | Mike  |      20 |             |   40 | Mike  |      20 |
|   50 | Kate  |      20 |    -------- |   50 | Kate  |      20 |
|   60 | Greg  |      50 |<---|        |   60 | Greg  |      50 |
|   70 | Phil  |      50 |<---|        |   70 | Phil  |      50 |
+------+-------+---------+             +------+-------+---------+
```

# Self-join queries

## Solution 2

We use inline views technique to combine the following queries

SELECT statement

```
SELECT enum
FROM EMPLOYEE
WHERE name = 'Kate';
```

SELECT statement

```
SELECT name
FROM EMPLOYEE
WHERE manager = 50;
```

We use the first `SELECT` statement to create an inline view `KATE`

Inline view

```
( SELECT enum
  FROM EMPLOYEE
  WHERE name = 'Kate' ) KATE
```

Then we join an inline view `KATE` with a relational table `EMPLOYEE`

SELECT statement with inline view

```
SELECT EMPLOYEE.name
FROM EMPLOYEE JOIN ( SELECT enum
                     FROM EMPLOYEE
                     WHERE name = 'Kate' ) KATE
              ON EMPLOYEE.manager = KATE.enum;
```

# References

T. Connoly, C. Begg, Database Systems, A Practical Approach to Design, Implementation, and Management, Chapters 6.3.7 Multi-table Queries, Pearson Education Ltd, 2015

D. Darmawikarta, SQL for MySQL A Beginner's Tutorial, Chapter 6, pages 55 - 61 Brainy Software Inc. First Edition: June 2014

How to ... ? Cookbook, How to implement queries in SQL ? (Part 1), Recipe 5.4 How to implement simple join queries ?

How to ... ? Cookbook, How to implement queries in SQL ? (Part 2) Recipe 6.1 How to implement self join queries ?