CSIT115 Data Management and Security

SELECT Statement (5)

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

SELECT statement (5)

Outline

Nested queries

Correlated nested queries

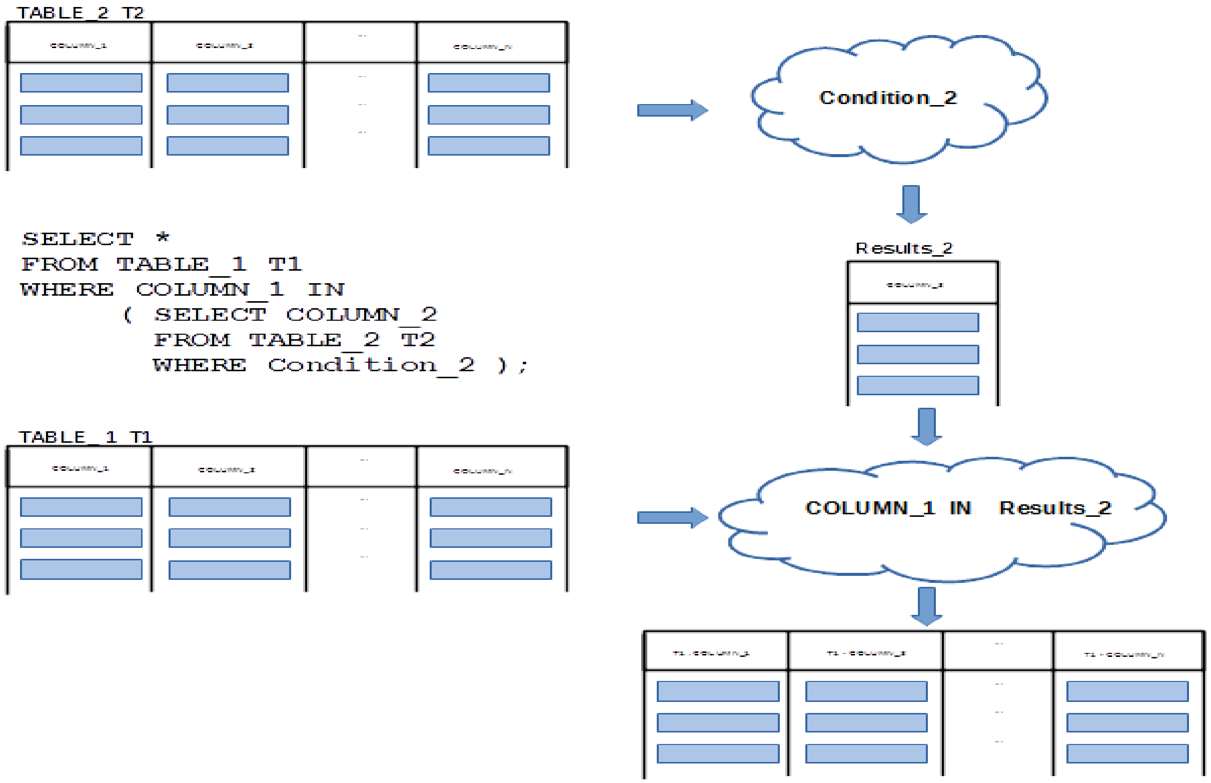
Queries with WITH clause

Relational views

Advanced DML statements

Nested queries

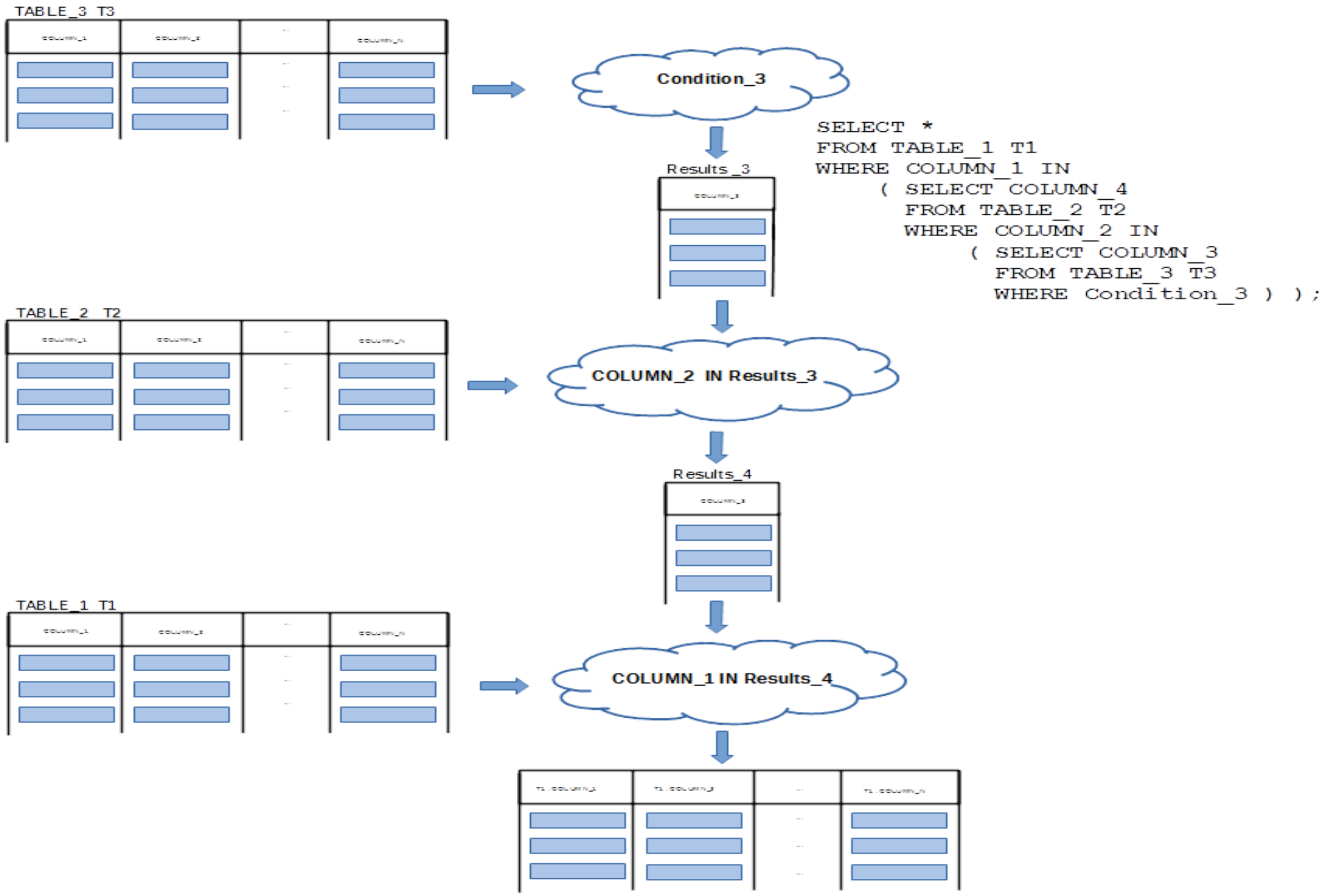
Nested query is a query with another query embedded in `WHERE`, `SELECT`, `FROM` clauses



```
SELECT *
FROM TABLE_1 T1
WHERE COLUMN_1 IN
      ( SELECT COLUMN_2
        FROM TABLE_2 T2
        WHERE Condition_2 );
```

Nested queries

Nested query is a query with another query embedded in `WHERE`, `SELECT`, `FROM` clauses



```
SELECT *
FROM TABLE_1 T1
WHERE COLUMN_1 IN
      ( SELECT COLUMN_4
        FROM TABLE_2 T2
        WHERE COLUMN_2 IN
              ( SELECT COLUMN_3
                FROM TABLE_3 T3
                WHERE Condition_3 ) );
```

Nested queries

Sample database

CREATE TABLKE 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 );
```

Nested queries

A class of nested queries is based on a concept of inline views

Inline views can be used to reduce the complexity of query implementation

An inner query is created first and its outcomes are used as a relational table (inline view) in an outer query

For example, a query find the titles of courses offered by a department chaired by Peter is decomposed into the following two queries:

- `Q1`: Find a department chaired by Peter
- `Q2`: Find the titles of courses offered by a department found in `Q1`

A query `Q1` is implemented first and then it is used in `WHERE` clause of query `Q2`

Nested queries

In nested queries SELECT statements are nested to theoretically unlimited level in WHERE clause

Nested query with a set membership operation IN

```
SELECT title
FROM COURSE
WHERE offered_by IN ( SELECT name
                      FROM DEPARTMENT
                      WHERE chair = 'Peter' );
```

If inner query returns more than one row the we must use IN istead of =

Nested queries

A way how we implement a query find the titles of courses offered by a department chaired by Peter is the following

Create a inner query as inline view Q

Inline view

```
( SELECT name
FROM DEPARTMENT
WHERE chair = 'Peter' ) Q
```

Create outer query that references an inline view Q created earlier

Query that references an inline view Q

```
SELECT title
FROM COURSE
WHERE offered_by IN Q.name
```

Replace a reference Q to an inline view with the inline view

Nested query with a set membership operation IN

```
SELECT title
FROM COURSE
WHERE offered_by IN ( SELECT name
                      FROM DEPARTMENT
                      WHERE chair = 'Peter' );
```

Nested queries

Another example, find the chairs of all departments that offer 12 credit point courses

An inner query as inline view

Inline view

```
( SELECT offered_by
FROM COURSE
WHERE credits = 12) Q;
```

Create outer query that references an inline view created earlier

Query that references an inline view Q

```
SELECT chair
FROM DEPARTMENT
WHERE name IN Q.offered_by
```

Replace a reference to an inline view with the inline view itself

Nested query with a set membership operation IN

```
SELECT chair
FROM DEPARTMENT
WHERE name IN ( SELECT offered_by
                FROM COURSE
                WHERE credits = 12 );
```

Nested queries

A query find the chairs of all departments that offer no courses is an example of ANTIJOIN operation

It is equivalent to a query find all rows in a relational table DEPARTMENT that cannot be joined with any row in a relational table COURSE

An inner query as inline view finds all departments that offer at least one course

Inline view

```
( SELECT offered_by
FROM COURSE ) Q;
```

An outer query finds all chairs of deparments that are NOT included in the results of an inner query

Query that references an inline view

```
SELECT chair
FROM DEPARTMENT
WHERE name NOT IN Q.offered_by
```

Nested queries

Finally, we replace a reference to an inline view Q with the inline view itself

Nested query with a negated set membership operation IN

```
SELECT chair
FROM DEPARTMENT
WHERE name NOT IN ( SELECT offered_by
                    FROM COURSE );
```

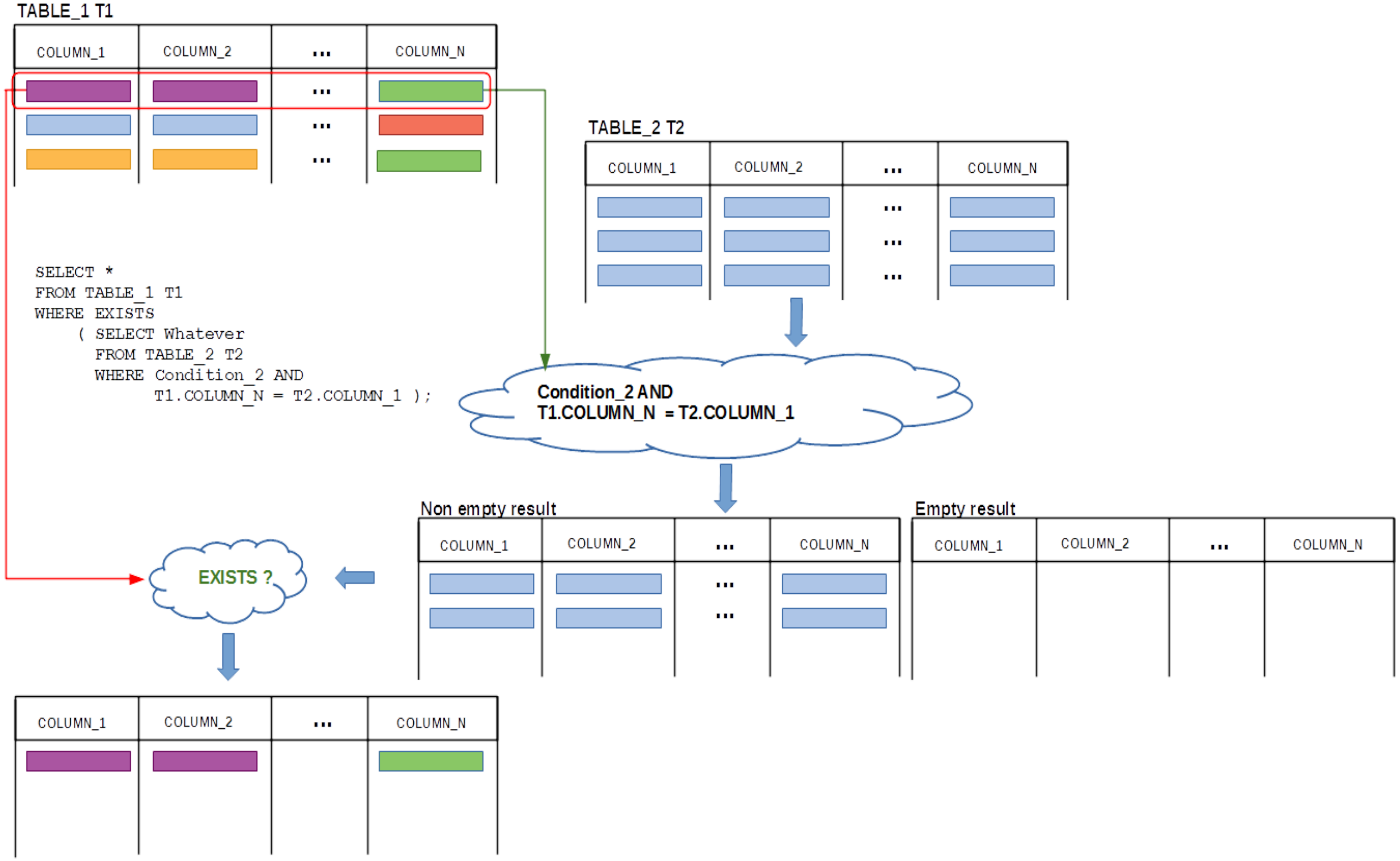
SELECT statement (5)

Outline

- Nested queries
- Correlated nested queries
- Queries with WITH clause
- Relational views
- Advanced DML statements

Correlated nested queries

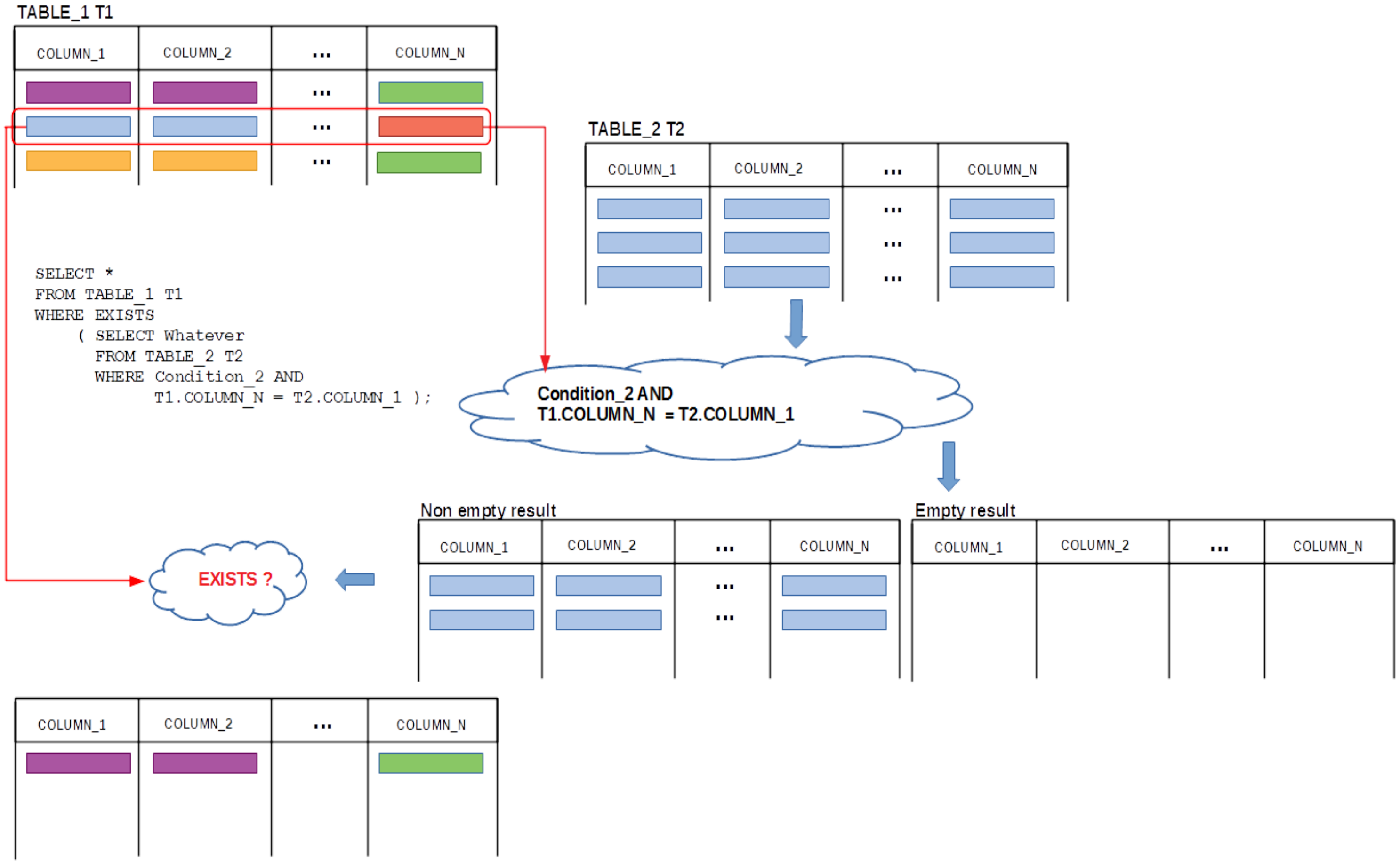
In a correlated nested query an inline view may reference the names of relational tables used in `SELECT` statement outer to the inline view



```
SELECT *
FROM TABLE_1 T1
WHERE EXISTS
     ( SELECT Whatever
       FROM TABLE_2 T2
       WHERE Condition_2 AND
             T1.COLUMN_N = T2.COLUMN_1 );
```

Correlated nested queries

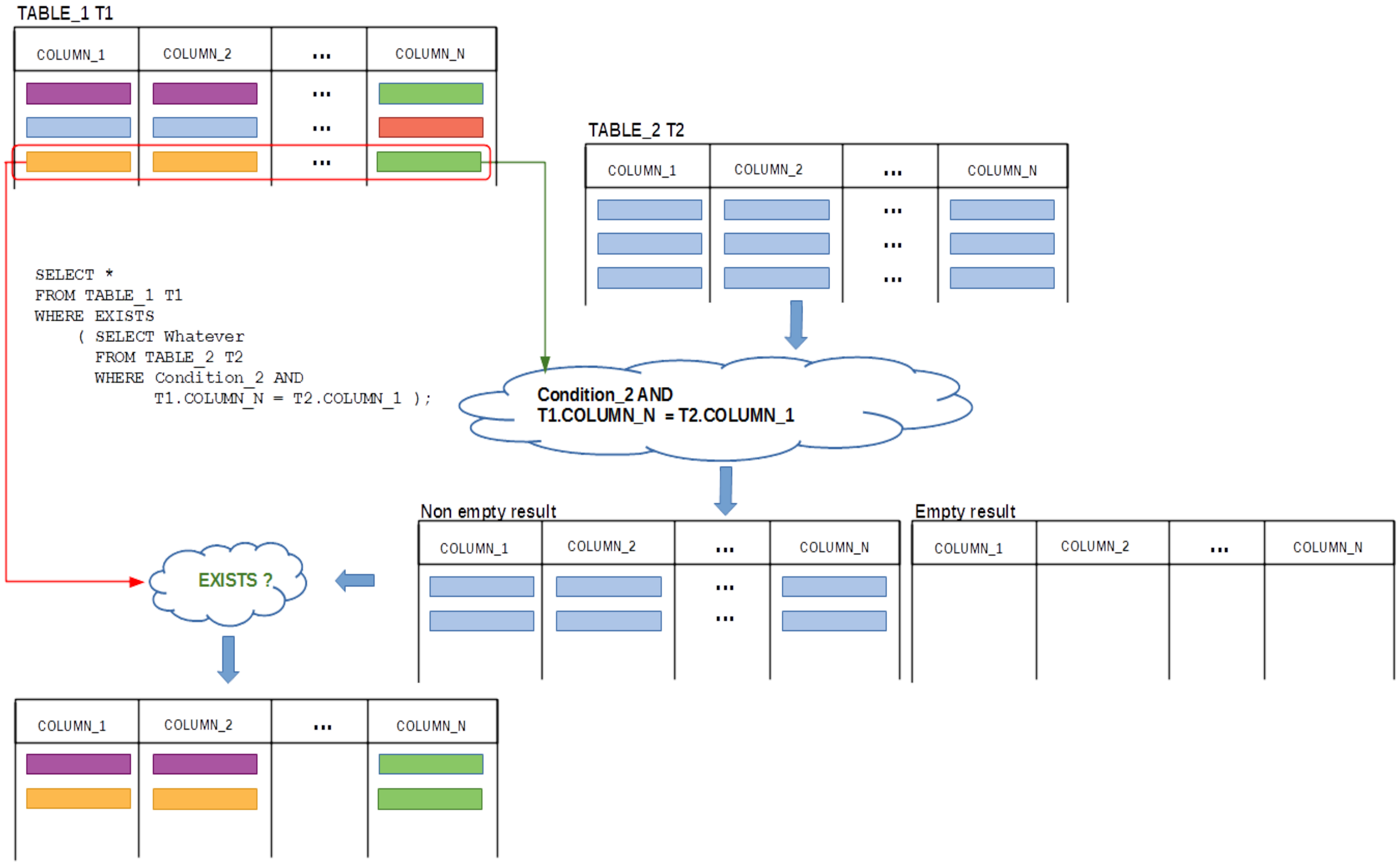
In a correlated nested query an inline view may reference the names of relational tables used in `SELECT` statement outer to the inline view



```
SELECT *
FROM TABLE_1 T1
WHERE EXISTS
      ( SELECT Whatever
        FROM TABLE_2 T2
        WHERE Condition_2 AND
              T1.COLUMN_N = T2.COLUMN_1 );
```

Correlated nested queries

In a correlated nested query an inline view may reference the names of relational tables used in `SELECT` statement outer to the inline view



```
SELECT *
FROM TABLE_1 T1
WHERE EXISTS
     ( SELECT Whatever
       FROM TABLE_2 T2
       WHERE Condition_2 AND
             T1.COLUMN_N = T2.COLUMN_1 );
```

TABLE_1 T1

COLUMN_1	COLUMN_2	...	COLUMN_N
🟪	🟪	...	🟩
🟦	🟦	...	🟥
🟧	🟧	...	🟩

TABLE_2 T2

COLUMN_1	COLUMN_2	...	COLUMN_N
🟦	🟦	...	🟦
🟦	🟦	...	🟦
🟦	🟦	...	🟦

Non empty result

COLUMN_1	COLUMN_2	...	COLUMN_N
🟦	🟦	...	🟦
🟦	🟦	...	🟦

Empty result

COLUMN_1	COLUMN_2	...	COLUMN_N

COLUMN_1	COLUMN_2	...	COLUMN_N
🟪	🟪		🟩
🟧	🟧		🟩

Correlated nested queries

For example, consider a query: Find the chairs of departments that offer 12 credit point courses

Such query can be re-phrased as an equivalent query find the chairs of departments such that there exists at least one course worth 12 credit points offered by a department we are looking for

An inner query ... course worth 12 credit points offered by a department we are looking for is implemented as the following inline view

Inline view

```
( SELECT *
FROM COURSE
WHERE credits = 12 AND offered_by = DEPARTMENT.name ) Q
```

An outer query find the chairs of departments such that there exists at least one course found in the inner quer is implemented in the folowing way

Query with an existential quantifier EXISTS that references and inline view

```
SELECT chair
FROM DEPARTMENT
WHERE EXISTS Q
```

Correlated nested queries

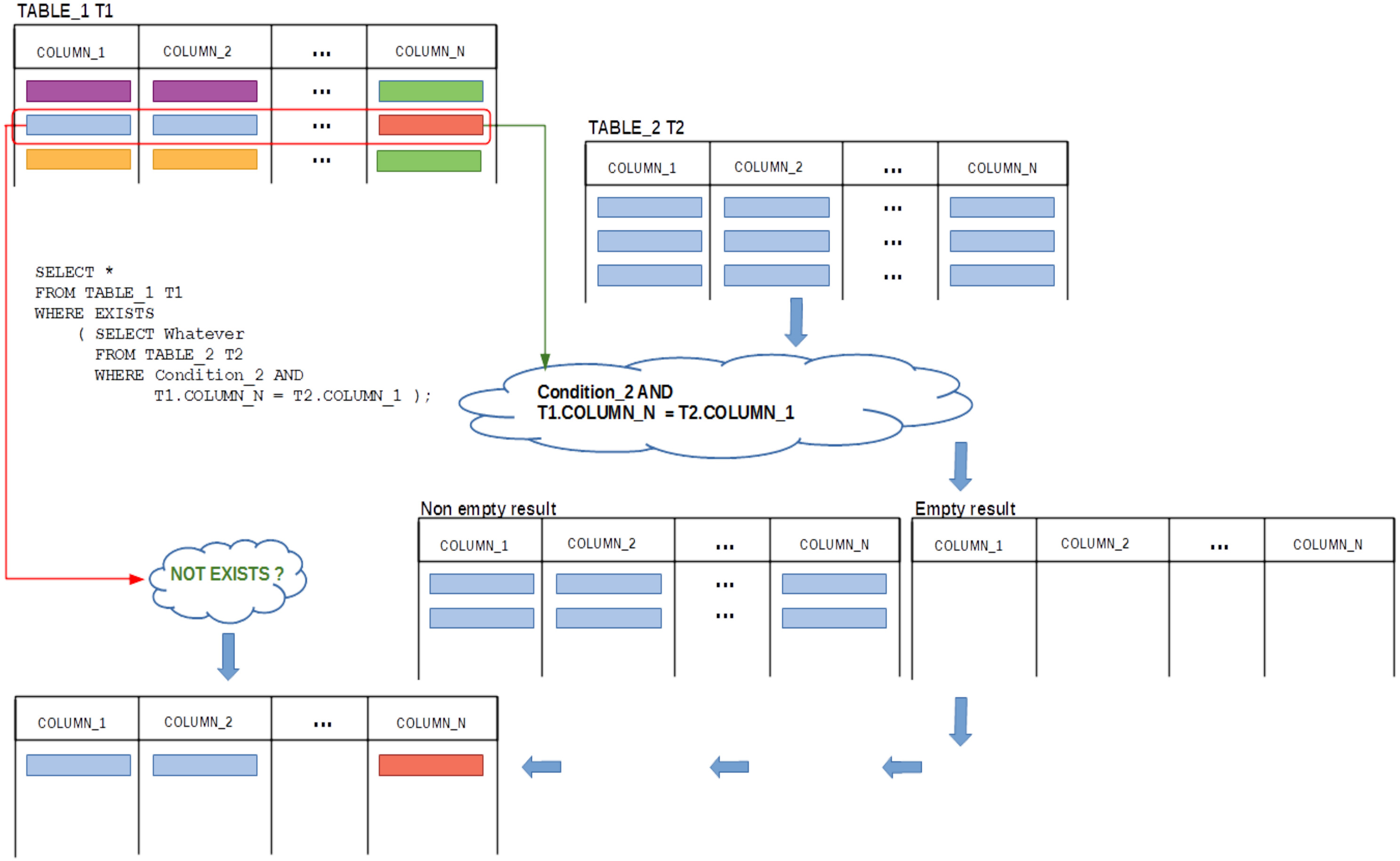
Finally, we replace a reference Q to an inline view with the inline view itself

Correlated nested query with an existential quantifier EXISTS

```
SELECT chair
FROM DEPARTMENT
WHERE EXISTS ( SELECT *
               FROM COURSE
               WHERE credits = 12 AND offered_by = DEPARTMENT.name );
```

Correlated nested queries

In a correlated nested query an inline view may reference the names of relational tables used in `SELECT` statement outer to the inline view



```
SELECT *
FROM TABLE_1 T1
WHERE EXISTS
     ( SELECT Whatever
       FROM TABLE_2 T2
       WHERE Condition_2 AND
             T1.COLUMN_N = T2.COLUMN_1 );
```

Correlated nested queries

Another example where inline view references a name of relational table used in SELECT statement outer to the inline view is the following

Find the chairs of all departments that offer no courses

It is equivalent to a query find the chairs of departments such that does not exist a course offered by a department we are looking for

An inner query finds ... a course offered by a department we are looking for

Inline view

```
( SELECT *
FROM COURSE
WHERE offered_by = DEPARTMENT.name ) Q;
```

An outer query finds all chairs of deparments such that does not exist a course found by an inner query

Query with a negated existential quantifier EXISTS that references an inline view

```
SELECT chair
FROM DEPARTMENT
WHERE NOT EXISTS Q
```

Correlated nested queries

Finally, we replace a reference to an inline view Q with the inline view itself

Correlated nested query with a negated existential quantifier EXISTS

```
SELECT chair
FROM DEPARTMENT
WHERE NOT EXISTS( SELECT *
                  FROM COURSE
                  WHERE offered_by = DEPARTMENT.name );
```

SELECT statement (5)

Outline

- Nested queries
- Correlated nested queries
- Queries with WITH clause
- Relational views
- Advanced DML statements

Queries with WITH clause

Consider a query: find the names of all departments together with the total number of courses offered by each department, include the departments that offer no courses

The query can be decomposed into the following two queries:

- find the names of departments and the numbers of courses offered by each department, and a query
- aggregate the results from the first query over the names of departments and count the total number of courses offered by each department

The first query can be implemented as a query definition DEPT_COURSE within WITH clause

WITH clause with a query definition

```
WITH DEPT_COURSE AS
                   ( SELECT name, cnum
                     FROM DEPARTMENT LEFT OUTER JOIN COURSE
                                     ON DEPARTMENT.name = COURSE.offered_by ),
```

Queries with **WITH** clause

The second query can be implemented a query definition DC_COUNT that references a query definition DEPT_COURSE within WITH clause

WITH clause with two query definitions

```
WITH DEPT_COURSE AS
                 ( SELECT name, cnum
                   FROM DEPARTMENT LEFT OUTER JOIN COURSE
                                  ON DEPARTMENT.name = COURSE.offered_by ),
     DC_COUNT AS
                 ( SELECT name, COUNT(cnum) total_courses
                   FROM DEPT_COURSE
                   GROUP BY name )
```

Queries with WITH clause

The final query is implemented as SELECT statement appended to a query definition DC_COUNT within WITH clause

WITH clause with two query definitions and SELECT statement

```
WITH DEPT_COURSE AS
                   ( SELECT name, cnum
                     FROM DEPARTMENT LEFT OUTER JOIN COURSE
                                     ON DEPARTMENT.name = COURSE.offered_by ),
     DC_COUNT AS
                   ( SELECT name, COUNT(cnum) total_courses
                     FROM DEPT_COURSE
                     GROUP BY name )
    SELECT name
    FROM DC_COUNT);
```

Queries with WITH clause

In another example WITH clause can be used to reduce the implementation complexity of the following query: find the chairs of departments that offer both 6 and 12 credit point courses

The query is decomposed into the following subqueries:

- Find the names of departments that offer 6 credit point courses
- Find the names of departments that offer 12 credit point courses
- Find the names of departments included in both results from the subqueries above
- Find the chairs of departments included in the previous subquery

The first subquery is implemented as the following query definition COURSE6CR within WITH clause

WITH clause with a query definition

```
WITH COURSE6CR AS
                ( SELECT offered_by
                  FROM COURSE
                  WHERE credits = 6 ),
```

Queries with WITH clause

A subquery: find the chairs of departments that offer 12 credit point courses is implemented as a query definition COURSE12CR appended to WITH clause

WITH clause with two query definitions

```
WITH COURSE6CR AS
                  ( SELECT offered_by
                    FROM COURSE
                    WHERE credits = 6 ),
     COURSE12CR AS
                  ( SELECT offered_by
                    FROM COURSE
                    WHERE credits = 12 );
```

Queries with WITH clause

A subquery: find the names of departments that offer both 6 and 12 credit point courses is implemented as a query definition COURSE6_12CR appended to WITH clause

WITH clause with three query definitions

```
WITH COURSE6CR AS
                 ( SELECT offered_by
                   FROM COURSE
                   WHERE credits = 6 ),
     COURSE12CR AS
                 ( SELECT offered_by
                   FROM COURSE
                   WHERE credits = 12 ),
     COURSE6_12CR AS
                 ( SELECT COURSE6CR.offered_by
                   FROM COURSE6CR JOIN COURSE12CR
                                 ON COURSE6CR.offered_by = COURSE12CR.offered_by ),
```

Queries with WITH clause

A subquery: find the chairs of departments that offer both 6 and 12 credit point courses is implemented as a query definition CHAIR appended to WITH clause

WITH clause with four query definitions

```
WITH COURSE6CR AS
                 ( SELECT offered_by
                   FROM COURSE
                   WHERE credits = 6 ),
     COURSE12CR AS
                 ( SELECT offered_by
                   FROM COURSE
                   WHERE credits = 12 ),
     COURSE6_12CR AS
                 ( SELECT COURSE6CR.offered_by
                   FROM COURSE6CR JOIN COURSE12CR
                                 ON COURSE6CR.offered_by = COURSE12CR.offered_by ),
     CHAIR AS
                 ( SELECT DEPARTMENT.chair
                   FROM COURSE6_12CR JOIN DEPARTMENT
                                     ON COURSE6_12CR.offered_by = DEPARTMENT.name )
```

Queries with WITH clause

Finally a query: find the chairs of departments that offer both 6 and 12 credit point courses is implemented as SELECT statement appended to WITH clause

WITH clause with four query definitions and SELECT statement

```
WITH COURSE6CR AS
                 ( SELECT offered_by
                   FROM COURSE
                   WHERE credits = 6 ),
     COURSE12CR AS
                 ( SELECT offered_by
                   FROM COURSE
                   WHERE credits = 12 ),
     COURSE6_12CR AS
                 ( SELECT COURSE6CR.offered_by
                   FROM COURSE6CR JOIN COURSE12CR
                                 ON COURSE6CR.offered_by = COURSE12CR.offered_by ),
     CHAIR AS
                 ( SELECT DEPARTMENT.chair
                   FROM COURSE6_12CR JOIN DEPARTMENT
                                    ON COURSE6_12CR.offered_by = DEPARTMENT.name )
SELECT *
FROM CHAIR;
```

SELECT statement (5)

Outline

Nested queries

Correlated nested queries

Queries with WITH clause

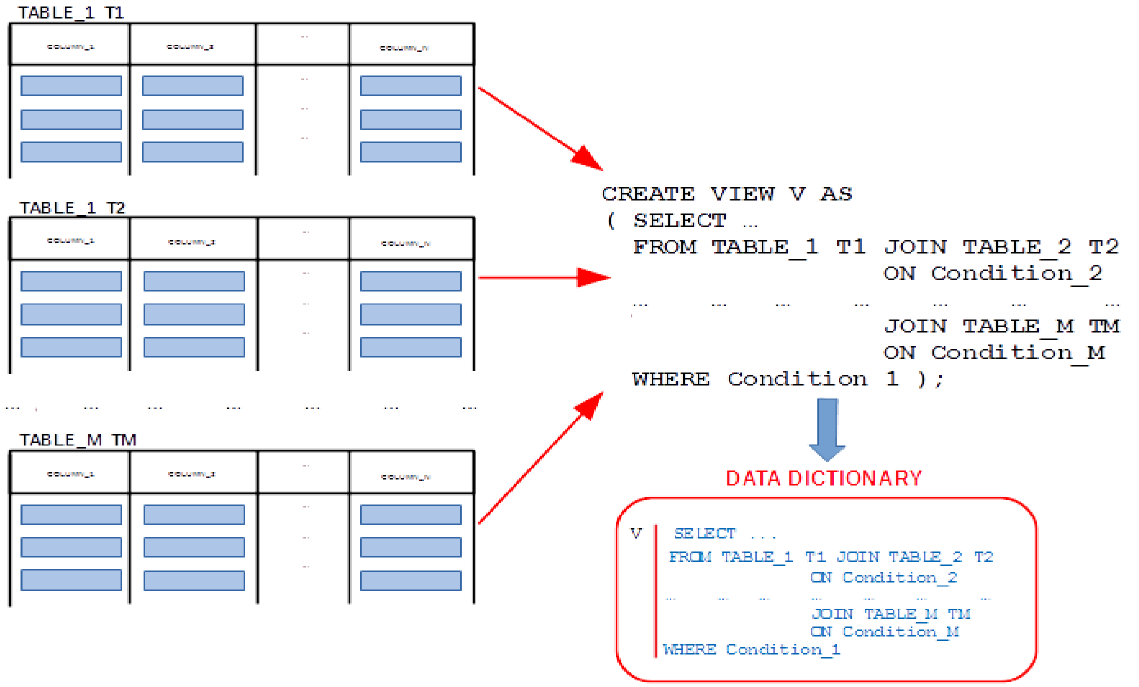
Relational views

Advanced DML statements

Relational views

A relational view is a virtual relational table (derived relational table) that occupies no presistent storage and it is computed from very beginning each time it is used in `SELECT` statement

A relational view is stored by a database management system as a pair (name of a view, `SELECT` statement that defines the structure and contents of the view)

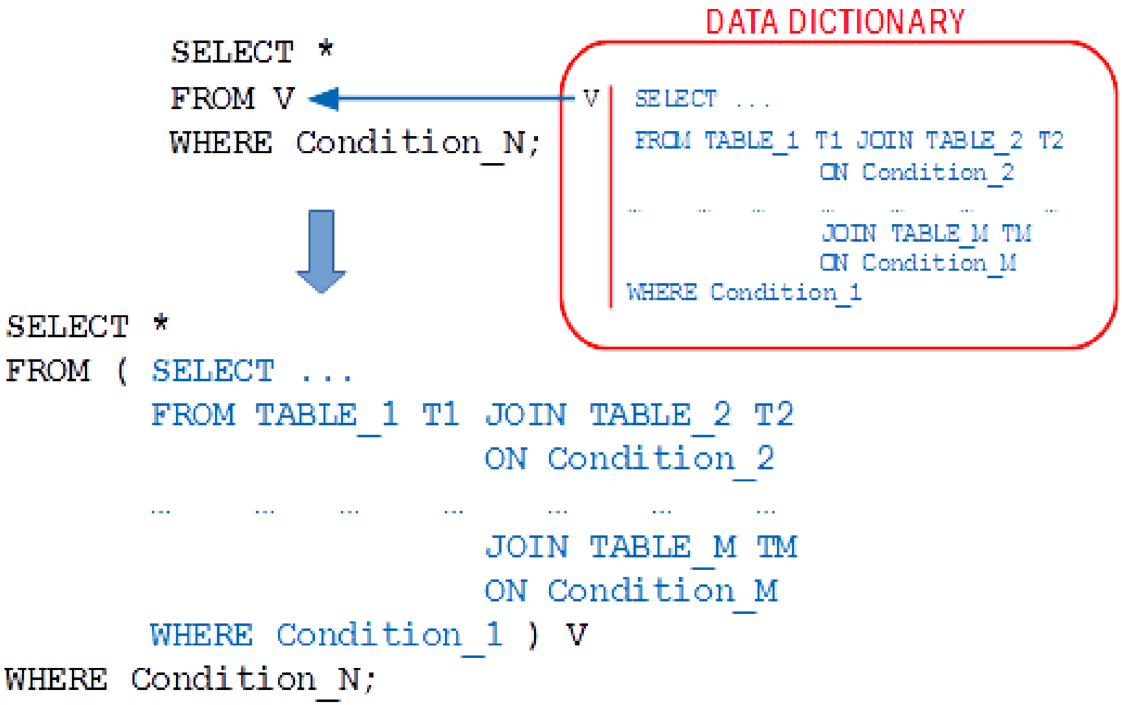


```
CREATE VIEW V AS
( SELECT ...
  FROM TABLE_1 T1 JOIN TABLE_2 T2
                  ON Condition_2
  ...     ...    ...     ...     ...     ...     ...
                  JOIN TABLE_M TM
                  ON Condition_M
  WHERE Condition 1 );
```

```
V | SELECT ...
  FROM TABLE_1 T1 JOIN TABLE_2 T2
                  ON Condition_2
  ...    ...   ...   ...   ...   ...   ...
                  JOIN TABLE_M TM
                  ON Condition_M
  WHERE Condition_1
```

Relational views

Each time a name of relational view is used in `SELECT` statement its definition replaces the name of view and it becomes an inline view



```
SELECT *
FROM V
WHERE Condition_N;
```

```
SELECT ...
FROM TABLE_1 T1 JOIN TABLE_2 T2
                ON Condition_2
...    ...    ...    ...    ...    ...    ...
                JOIN TABLE_M TM
                ON Condition_M
WHERE Condition_1
```

```
SELECT *
FROM ( SELECT ...
       FROM TABLE_1 T1 JOIN TABLE_2 T2
                       ON Condition_2
       ...    ...    ...    ...    ...    ...    ...
                       JOIN TABLE_M TM
                       ON Condition_M
       WHERE Condition_1 ) V
WHERE Condition_N;
```

Relational views

For example, create a relational view that contains the names of all departments together with the total number of courses offered by each department

CREATE VIEW statement

```
CREATE VIEW VDEPT( name, total_courses ) AS
( SELECT name, count(cnum)
  FROM DEPARTMENT LEFT OUTER JOIN COURSE
                  ON DEPARTMENT.name = COURSE.offered_by
                  GROUP BY name );
```

Relational views

Then a view VDEPT can be used to implement a query find the names of departments that offer more than 1 course

SELECT statement

```
SELECT name
FROM VDEPT
WHERE total_courses > 1;
```

The same query implemented with GROUP BY and HAVING clauses is the following

SELECT statement with LEFT OUTER JOIN operation

```
SELECT name, count(cnum)
FROM DEPARTMENT LEFT OUTER JOIN COURSE
                  ON DEPARTMENT.name = COURSE.offered_by
GROUP BY name
HAVING count(cnum) > 1;
```

Relational views

A relational view can be used to reduce the complexity of SELECT statements

For example, a query find the chairs of departments that offer both 6 and 12 credit point courses is decomposed into the following queries

V6: Find the names of departments that offer 6 credit point courses

V12: Find the names of departments that offer 12 credit point courses

VNAME: Find the names of departments included in both V6 and V12

VCHAIR: Find the chairs of departments included in VNAME

The views are implemented in the following way

V6: Find the names of departments that offer 6 credit point courses

CREATE VIEW statement

```
CREATE VIEW V6( name ) AS
( SELECT offered_by
  FROM COURSE
  WHERE credits = 6 );
```

Relational views

V12: Find the names of departments that offer 12 credit point courses

CREATE VIEW statement

```
CREATE VIEW V12( name ) AS
( SELECT offered_by
  FROM COURSE
  WHERE credits = 12 );
```

VNAME: Find the names of departments included in both V6 and V12

CREATE VIEW statement

```
CREATE VIEW VNAME( name ) AS
( SELECT V6.name
  FROM V6 JOIN V12
       ON V6.name = V12.name );
```

VCHAIR: Find the chairs of departments included in VNAME

CREATE VIEW statement

```
CREATE VIEW VCHAIR( chair ) AS
( SELECT DEPARTMENT.chair
  FROM VNAME JOIN DEPARTMENT
             ON VNAME.name = DEPARTMENT.NAME );
```

Relational views

The final query is

SELECT statement

```
SELECT *
FROM VCHAIR;
```

SELECT statement (5)

Outline

- Nested queries
- Correlated nested queries
- Queries with WITH clause
- Relational views
- Advanced DML statements

Advanced DML statements

Advanced Data Manipulation statements use subqueries implemented as `SELECT` statements inside DML statements

For example, delete all courses offered by a department chaired by Peter

DELETE statement with nested SELECT statemenmt

```
DELETE FROM COURSE
WHERE offered_by = ( SELECT name
                     FROM DEPARTMENT
                     WHERE chair = 'Peter' );
```

Delete all departments that offer no courses

DELETE statement with negated existential quantifier EXISTS and nested SELECT statement

```
DELETE FROM DEPARTMENT
WHERE NOT EXISTS ( SELECT '1'
                   FROM COURSE
                   WHERE COURSE.offered_by = DEPARTMENT.name );
```

Advanced DML statements

Increase the total number of staff members by 5 in all departments that offer more than 20 courses

UPDATE statement with nested SELECT statement

```
UPDATE DEPARTMENT
SET total_staff_number = total_staff_number + 5
WHERE name IN ( SELECT offered_by
                FROM COURSE
                GROUP BY offered_by
                HAVING COUNT(cnum) > 20 );
```

Add to table `DEPARTMENT` a column that contains the total number of courses offered by each department and insert the correct values into the column

ALTER TABLE statement that adds an attribute

```
ALTER TABLE DEPARTMENT ADD ( total_courses DECIMAL(2) );
```

UPDATE statement with nested SELECT statement

```
UPDATE DEPARTMENT
SET total_courses = ( SELECT COUNT(title)
                      FROM COURSE
                      WHERE COURSE.offered_by = DEPARTMENT.name );
```

Advanced DML statements

Create a table that contains the names of departments together with the total number of courses offered by each department and insert correct data into the table

CREATE TABLE statement with nested SELECT statement

```
CREATE TABLE DCNT AS
( SELECT name, COUNT(cnum) TOTC
  FROM DEPARTMENT LEFT OUTER JOIN COURSE
                  ON DEPARTMENT.name = COURSE.offered_by
  GROUP BY name );
```

Note, that NONE of the consistency constraints except `NULL/NOT NULL` constraints are copied from the relational tables `DEPARTMENT` and `COURSE` into a relational table `DCNT`

Advanced DML statements

To preserve the consistency constraints we create a relational table DCNT first ...

CREATE TABLE statement

```
CREATE TABLE DCNT(
 name          VARCHAR(50)  NOT NULL,
 total_courses DECIMAL(2)    NOT NULL,
  CONSTRAINT DCNT_pkey PRIMARY KEY(name),
              CONSTRAINT DCNT_fkey FOREIGN KEY (name) REFERENCES DEPARTMENT(name) );
```

... and we load data into the table next

INSERT statement with nested SELECT statement

```
INSERT INTO DCNT
( SELECT name, COUNT(cnum)
  FROM DEPARTMENT LEFT OUTER JOIN COURSE
                 ON DEPARTMENT.name = COURSE.offered_by
  GROUP BY name );
```

References

T. Connoly, C. Begg, Database Systems, A Practical Approach to Design, Implementation, and Management, Chapter 6.3.5 Subqueries, Chapter 6.3.8 EXISTS and NOT EXISTS, Chapter 7.4.1 Creating a View Pearson Education Ltd, 2015

D. Darmawikarta, SQL for MySQL A Beginner's Tutorial, Chapter 7 Subqueries, Chapter 9 Views, Brainy Software Inc. First Edition: June 2014

How to ... ? Cookbook, How to implement queries in SQL ? (Part 2) Recipe 6.3 How to implement nested queries ?

How to ... ? Cookbook, How to implement queries in SQL ? (Part 2) Recipe 6.4 Recipe 6.4 How to implement correlated nested queries ?