Structured Query Language (SQL) Tutorial

SWEN304/SWEN439

Lecturer: Dr Hui Ma

Engineering and Computer Science

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Slides by: Pavle Mogin & Hui Ma

Outline

- SQL Constraints:
  - `CHECK` constraint
  - Referential integrity constraint

    `MATCH PARTIAL|MATCH FULL|MATCH SIMPLE`
- Using the same table in different context
- Correlated queries

Specifying CHECK Constraint

```
CREATE TABLE COURSE (
CourId  CHAR(4) CONSTRAINT cspk PRIMARY KEY

    CHECK constraint comes here
                                     ,
CName  CHAR(15) NOT NULL,
Points  INT NOT NULL,
Dept  CHAR(25)   );
```

- Suppose we would like to define an additional constraint on CourId that restricts this attribute values to those that follow the pattern:
  - the first character is a capital letter,
  - the next three characters are numbers between 100 and 999

Additional Constraint on CourId

```
CREATE TABLE COURSE (
   CourId  CHAR(4) CONSTRAINT cspk PRIMARY KEY
     CONSTRAINT valcon
       CHECK( (SUBSTR(CourId,1,1) BETWEEN ‘A’ AND ‘Z’)
      AND (SUBSTR(CourId,2,3) SIMILAR TO ‘[1-9][0-9][0-9]’)),
   CName  CHAR(15) NOT NULL,
   Points  INT NOT NULL,
   Dept  CHAR(25) );
```

- Here, we used regular expressions to define a CHECK constraint of the *format* (*pattern*) type

Referential Integrity – A Formal Definition

- Relations $r(N_2)$ and $r(N_1)$ satisfy referential integrity $N_2[Y] \subseteq N_1[X]$ if

$$(\forall u \in r(N_2))(\exists v \in r(N_1))(u[Y] = v[X] \vee$$
$$(\exists i \in \{1, \ldots, m\})(u[B_i] = \omega))$$

- Either tuples u and v are equal on X and Y values, or there exists at least one attribute in Y whose u value is null

Referential Integrity (1)

- TEXT_BOOK ({Title, ISBN, C_Code, C_Num}, {ISBN })
- COURSE ({C_Code, C_Num, C_Name },
  {C_Code + C_Num,})
- How do we specify the referential integrity constraint:

  $TEXT\_BOOK\ [C\_Code, C\_Num] \subseteq COURSE\ [C\_Code, C\_Num]$

Need:

- Referring and Referred relational variables and fields
- No Match clause or Match: `FULL|PARTIAL|SIMPLE`
- Action: `NO ACTION, CASCADE, SET NULL, SET DEFAULT`

Referential Integrity (2) MATCH

- MATCH clause:

  SIMPLE /no MATCH clause specified (Default):

  - Two tuples either match on primary key/foreign key values, or the foreign key has at least one null valued component

  PARTIAL:

  - Two tuples either match on primary key/foreign key values, or the not null valued foreign key components match the corresponding components of at least one primary key value

  FULL:

  - Two tuples either match on primary key/foreign key values, or all foreign key components are null

Referential Integrity (3)

```
CREATE TABLE COURSE (
    C_Code CHAR(4) NOT NULL DEFAULT ‘stat’,
    C_Num INT CHECK(C_NUM BETWEEN 100 AND 999)  DEFAULT 100,
    C_Name CHAR(25),
    PRIMARY KEY (C_Code, C_Num )
    );
```

```
CREATE TABLE TEXT_BOOK (
    Title  CHAR(30) NOT NULL,
    ISBN  INT PRIMARY KEY,
    C_Code CHAR(4),
    C_Num  INT(2)  CHECK(C_Num > 99 AND C_Num < 1000),
    FOREIGN KEY (C_Code, C_Num) REFERENCES COURSE
        [MATCH <condition>]  ON DELETE <action>
    );
```

Referential Integrity (4a)

[MATCH <condition>] ON DELETE <action>

- **No MATCH clause** (Default)
- **<action>: NO ACTION (RESTRICT)**

TEXTBOOK

Title	ISBN	C_Code	C_Num
FDBS	1111	comp	302

COURSE

C_Code	C_Num	Name
comp	302	DB

```
INSERT INTO TEXTBOOK VALUES ('Disc. Log.', 2222, 'math', null);
```

??

```
DELETE FROM COURSE WHERE C_Code = 'comp' AND C_Num = 302;
```

??

Referential Integrity (4b - Answer)

[MATCH <condition>] ON DELETE <action>

- **No MATCH clause** (Default)
- **<action>: NO ACTION** (RESTRICT)

TEXTBOOK

Title	ISBN	C_Code	C_Num
FDBS	1111	comp	302

COURSE

C_Code	C_Num	Name
comp	302	DB

INSERT INTO TEXTBOOK VALUES (‘Disc. Log.’, 2222, ‘math’, null);

Successful, because of MATCH default

DELETE FROM COURSE WHERE C_Code = ‘comp’ AND C_Num = 302;

Rejected, because of NO ACTION

Referential Integrity (5a)

[MATCH <condition>] ON DELETE <action>

- **MATCH <condition>: PARTIAL**
- **<action>: CASCADE,**

TEXTBOOK

Title	ISBN	C_Code	C_Num
FDBS	1111	comp	302

COURSE

C_Code	C_Num	Name
comp	302	DB

```
INSERT INTO TEXTBOOK VALUES (‘Disc. Log.’, 2222, ‘math’, null);
```

??

```
INSERT INTO TEXTBOOK VALUES (‘Disc. Log.’, 2222, ‘comp’, null);
```

??

```
DELETE FROM COURSE WHERE C_Code = ‘comp’ AND C_Num = 302;
```

??

Referential Integrity (5b - Answer)

[MATCH <condition>] ON DELETE <action>

- **MATCH <condition>: PARTIAL**
- **<action>: CASCADE,**

TEXTBOOK

Title	ISBN	C_Code	C_Num
FDBS	1111	comp	302

COURSE

C_Code	C_Num	Name
comp	302	DB

INSERT INTO TEXTBOOK VALUES (‘Disc. Log.’, 2222, ‘math’, null);

Rejected, because of MATCH PARTIAL

INSERT INTO TEXTBOOK VALUES (‘Disc. Log.’, 2222, ‘comp’, null);

Successful, because of MATCH PARTIAL

DELETE FROM COURSE WHERE C_Code = ‘comp’ AND C_Num = 302*;*

Successful, because of CASCADE (all tuples will be deleted)

Referential Integrity (6a)

[MATCH <condition>] ON DELETE <action>

- **MATCH <condition>: FULL**
- **<action>: SET NULL,**

TEXTBOOK

Title	ISBN	C_Code	C_Num
FDBS	1111	comp	302

COURSE

C_Code	C_Num	Name
comp	302	DB

INSERT INTO TEXTBOOK VALUES (‘Disc. Log.’, 2222, comp, null);

??

INSERT INTO TEXTBOOK VALUES (‘Disc. Log.’, 2222, null, null);

?

DELETE FROM COURSE WHERE C_Code = ‘comp’ AND C_Num = 302;

??

Referential Integrity (6b - Answer)

[MATCH <condition>] ON DELETE <action>

- **MATCH <condition>: FULL**
- **<action>: SET NULL,**

TEXTBOOK

Title	ISBN	C_Code	C_Num
FDBS	1111	comp	302

COURSE

C_Code	C_Num	Name
comp	302	DB

`INSERT INTO` TEXTBOOK `VALUES` ('Disc. Log.', 2222, 'comp', null);

Rejected, because of MATCH FULL

`INSERT INTO` TEXTBOOK `VALUES` ('Disc. Log.', 2222, null, null);

Successful, because of MATCH FULL

`DELETE FROM` COURSE `WHERE` C_Code = comp `AND` C_Num = 302*;*

Successful, because of `SET NULL` (course tuple will be deleted, and the foreign key of the textbook tuple will be nullified)

Referential Integrity (7a)

[MATCH <condition>] ON DELETE <action>

- **<action>: SET DEFAULT,**

TEXTBOOK

Title	ISBN	C_Code	C_Num
FDBS	1111	comp	302

COURSE

C_Code	C_Num	Name
comp	302	DB
stat	100	STAT

```
DELETE FROM COURSE WHERE C_Code = ‘comp’ AND C_Num = 302;
```

??

Referential Integrity (7b - Answer)

[MATCH <condition>] ON DELETE <action>

- **<action>: SET DEFAULT**

TEXTBOOK

Title	ISBN	C_Code	C_Num
FDBS	1111	comp	302

COURSE

C_Code	C_Num	Name
comp	302	DB
stat	100	STAT

```
DELETE  FROM COURSE  WHERE C_Code = 'comp' AND C_Num = 302;
```

Successful, because of SET DEFAULT (course tuple will be deleted, and the foreign key of the **textbook tuple** will be set to *stat 100*)

University Database

STUDENT

LName	FName	StudId	Major	TrPts
Smith	Susan	131313	Comp	54
Bond	James	007007	Math	120
Smith	Susan	555555	Comp	30
Cecil	John	010101	Math	90

COURSE

CName	CrsId	Points	Dept
DB Sys	C302	15	Comp
SofEng	C301	15	Comp
DisMat	M214	22	Math
Pr&Sys	C201	22	Comp

GRADES

StudId	CrsId	Grade
007007	C302	A+
555555	C302	ω
007007	C301	A
007007	M214	A+
131313	C201	B-
555555	C201	C
131313	C302	ω
007007	C201	A
010101	C201	ω

A Question for You (Tricky Null Value)

- What is wrong with the following query:

```
SELECT *
FROM GRADES
WHERE Grade = Null;
```

since it returns an empty table

StudId	CrsId	Grade

- Answer:

a) There is a mistake, only I do not know where

b) PostgreSQL is rubbish

c) Null is not a real value. It can be anything. So, to the questions whether

Null = Null, or

Grade = Null

PostgreSQL answers “I don’t know”.

Multiple uses of the same table

- SQL allows multiple occurrences of the same table in a `FROM` clause

- In that case, each occurrence of the same table has a different role, or a different context of usage

- Aliases are used to denote the context of usage

Multiple Uses of the Same Table

- **Query**: Retrieve student *ids* and *TrPts* of students that have greater number of transfer points than the student with *StudentId = 131313*

```
SELECT s1.StudId, s1.TrPts
FROM STUDENT s1, STUDENT s2
WHERE s1.TrPts > s2.TrPts     AND
            s2.StudId = 131313 ;
```

Not an Equi Join

- The context of s2 is "number of points of the student with StudentId = 131313"
- The context of s1 is "list of students having greater number of points than student with StudentId = 131313"

Nested Queries

- Some queries require comparing a tuple to a collection of tuples (e.g., *students doing courses that have more than 100 students*)

- This task can be accomplished by embedding a SQL query into `WHERE` clause of another query
  - The embedded query is called **nested query,**
  - The query containing the nested query is called **outer query**

- The comparison is made by using `IN`, θ `ANY`, θ `SOME`, and θ `ALL` operators, where $\theta \in \{ =, <, <=, >=, >, <> \}$

- Note: `IN` ⇔ `=ANY` and `IN` ⇔ `=SOME`

Correlated Nested Queries

- Let the variable *s* contain the current tuple of the outer query
- If the nested query doesn't refer to *s* :
  - The nested query computes the same result for each tuple in *s*
  - The outer query and the nested query are said to be **uncorrelated**
- If a condition in the `WHERE` clause of the nested query refers to some attributes of a relation declared in the outer query, the two queries are said to be **correlated**
  - Have to compute the inner query for each tuple considered by the outer query
  - Correlated nested queries consume more computation time than uncorrelated ones

Nested Queries

- Is the following nested query correlated or not?
  - Select first names of the students that didn't enroll M214

```
SELECT s1.FName
FROM  STUDENT s1
WHERE s1.StudId IN
      ((SELECT  s2.StudId FROM STUDENT s2)
       EXCEPT
       (SELECT StudId FROM  GRADES
        WHERE CourId  = 'M214'));
```

FName
Susan
Susan
John

Correlated Nested Queries

- Consider the relation schemas:

EMPLOYEE ({EmpId, EmpName, Salary }, {EmpId })

PROJECT ({ProjId, ProjName }, {ProjId })

WORKS_ON ({EmpId, ProjId, NoOfHours}, {EmpId + ProjId })

Correlated Nested Queries (1)

- **Query**: Retrieve names of employees that work more hours on a project than the average number of hours on that same project :

```
SELECT e.EmpName
FROM EMPLOYEE e, WORKS_ON w
WHERE e.EmpId = w.EmpId   AND
           NoOfHours > (SELECT AVG(NoOfHours )
                           FROM  WORKS_ON w1
                           WHERE  w.ProjId = w1.ProjId );
```

- Here, in the nested query, the task of *w* is to focus on the current project, and allow computing requested average
- *w.ProjId* is a correlated attribute

Correlated Nested Queries (2)

- Query: Retrieve names of employees that are ranked first three according to their salaries (each employee has a different salary)

```
SELECT e.EmpName
FROM EMPLOYEE e
WHERE 3 > (
    SELECT COUNT(*)
    FROM  EMPLOYEE e1
    WHERE e1.Salary > e.Salary);
```

- In the nested query, only the employees that have higher salary than the current employee are selected, and current employee will be selected in the outer query only if the number of employees, selected in the inner query, is less than 3
- e.Salary is a correlated attribute

Another Query

- Query: show the project name and the average salary of employees who worked on projects that took a total of more than 1000 hours.

```
CREATE VIEW ExpensiveProjects AS
   (SELECT ProjName, AVG(Salary)
    FROM WORKS_ON NATURAL JOIN EMPLOYEE NATURAL
    JOIN PROJECT
    WHERE  Projid IN
         (SELECT ProjId
           FROM (SELECT ProjId, SUM(NoOfHours) AS TotHours
                      FROM WORKS_ON
                      GROUP BY ProjId) AS ProjHours
           WHERE TotHours > 1000)
     GROUP BY ProjName
 );
```

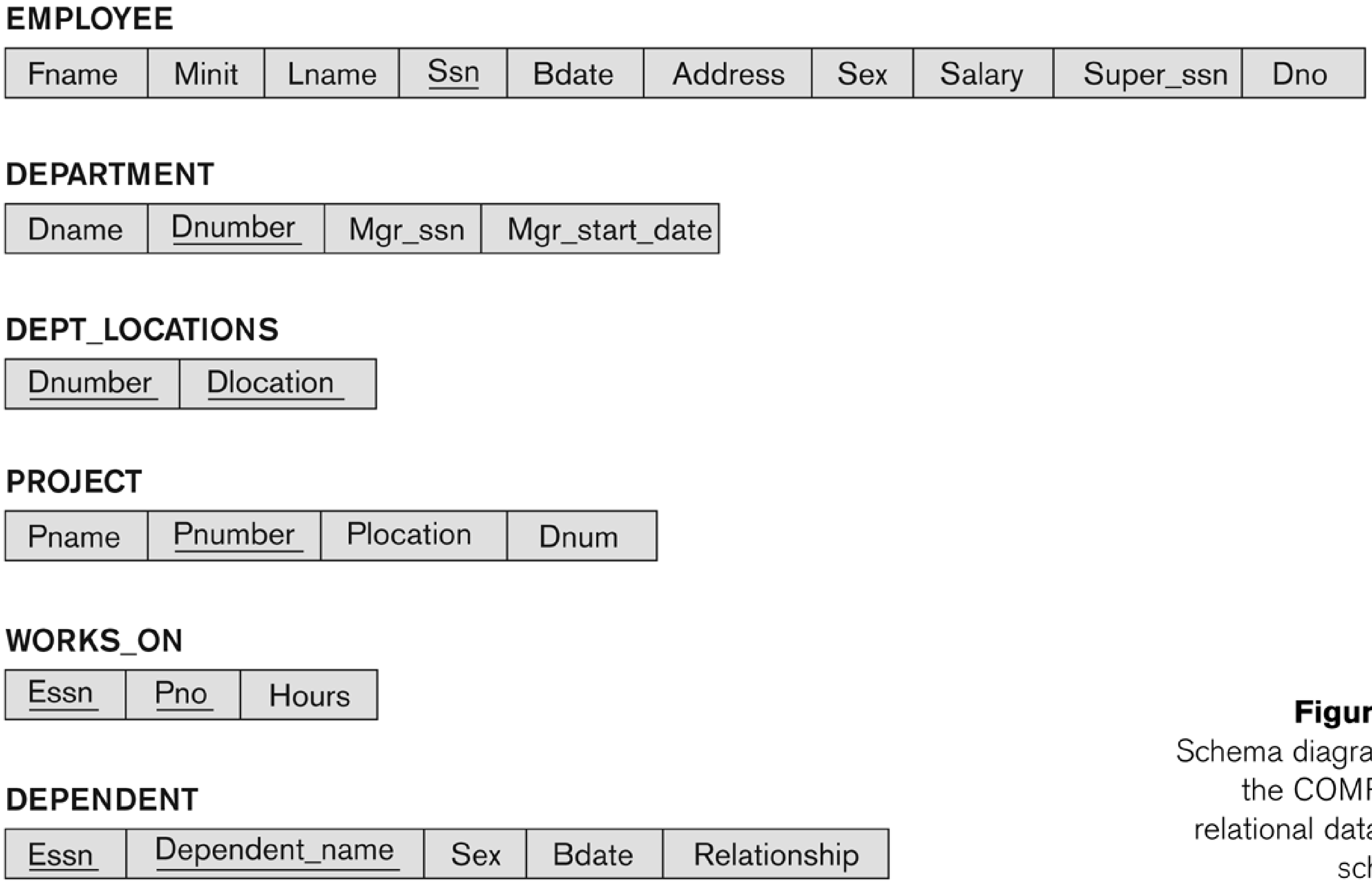
Another Query

- Query: show the project name and the average salary of employees who worked on projects that took a total of more tha 1000 hours.

```
CREATE VIEW ExpensiveProjects AS
  (SELECT ProjName, AVG(Salary)
    FROM WORKS_ON NATURAL JOIN EMPLOYEE NATURAL JOIN
(SELECT ProjId, SUM(NoOfHours) AS TotHours
FROM WORKS_ON
GROUP BY ProjId) AS ProjHours
    WHERE TotHours > 1000
   GROUP BY ProjName
   );
```

Exercises

COMPANY Database Schema



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 5.5
Schema diagram for the COMPANY relational database schema.

Exercises

In SQL, specify the following queries on the COMPANY database using the concept of nested queries.

1) Retrieve the names of all employees who work in the department that has the employee with the highest salary among all employees.
2) Retrieve the names of all employees whose supervisor’s supervisor has '888665555' for Ssn.
3) Retrieve the names of employees who make at least $10,000 more than the employee who is paid the least in the company.

Exercises

1) Retrieve the names of all employees who work in the department that has the employee with the highest salary among all employees.

```
SELECT LNAME
FROM EMPLOYEE
WHERE DNO = (SELECT DNO
               FROM EMPLOYEE
               WHERE SALARY = (SELECT MAX(SALARY)
                               FROM EMPLOYEE) )
```

Exercises

2) Retrieve the names of all employees whose supervisor’s supervisor has '888665555' for Ssn.

```
SELECT LNAME
FROM EMPLOYEE
WHERE SUPERSSN IN
        (SELECT SSN
          FROM EMPLOYEE
          WHERE SUPERSSN = ‘888665555’ )
```

Exercises

3) Retrieve the names of employees who make at least $10,000 more than the employee who is paid the least in the company

```
SELECT LNAME
FROM EMPLOYEE
WHERE SALARY >= 10000 +
            ( SELECT MIN(SALARY)
               FROM EMPLOYEE)
```