UCS310
Database Management System

Introduction to SQL

Lecture-10

Date: 31 Jan 2023

Dr. Sumit Sharma
Assistant Professor
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology, Patiala

Recap

- Nested Subqueries
- Set Membership – *where*
  - in, not in
- Set Comparisons – *where*
  - some, all
- exists, unique
- Subqueries in *from*
- *with* clause

Subqueries in the Select Clause

Scalar Subquery

- Scalar subquery is one which is used where a single value is expected
- Scalar subquery can be used in the **select, where and having** clause

- List all departments along with the number of instructors in each department

```
select dept_name,
        ( select count(*)
          from instructor
          where department.dept_name = instructor.dept_name)
    as num_instructors
from department;
```

- **Runtime error if the subquery returns more than one result tuple**

Modification of the Database

Modification of the Database

- Deletion of tuples from a given relation
- Insertion of new tuples into a given relation
- Updating of values in some tuples in a given relation

Deletion

delete from *r*

where *P;*

- **delete** statement first finds all tuples *t* in *r* for which *P(t)* is true, and then deletes them from *r*
- **where** is optional

- **NOTE: delete** command operates on only one relation

Deletion Example

- Delete all instructors

```
delete from instructor
```

- Delete all instructors from the Finance department

```
delete from instructor
where dept_name= 'Finance';
```

- *Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building.*

```
delete from instructor
where dept_name in (select dept_name
                    from department
                    where building = 'Watson');
```

Deletion in Nested select

- Delete all instructors whose salary is less than the average salary of instructors

```
delete from instructor
where salary < (select avg (salary)
                from instructor);
```

  - Problem: as we delete tuples from *instructor*, the average salary changes
  - Solution used in SQL:
    1. First, compute **avg** (salary) and find all tuples to delete
    2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

Insertion

- Add a new tuple to *course*

```
insert into course
    values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- or equivalently

```
insert into course (course_id, title, dept_name, credits)
    values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- Add a new tuple to *student* with *tot_creds* set to null

```
insert into student
    values ('3003', 'Green', 'Finance', null);
```

Insertion

- Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of ₹18,000

```
insert into instructor
    select ID, name, dept_name, 18000
    from   student
    where  dept_name = 'Music' and total_cred > 144;
```

- The **select from where** the statement is evaluated fully before any of its results are inserted into the relation.

  Otherwise queries like

  insert into *table1* **select** * **from** *table1*

  would cause problems, might insert an infinite number of tuples if the primary key constraint on student were absent

Updates

- Give a 5% salary raise to all instructors

```
update instructor
   set salary = salary * 1.05
```

- Give a 5% salary raise to those instructors who earn less than 70000

```
update instructor
   set salary = salary * 1.05
 where salary < 70000;
```

- Give a 5% salary raise to instructors whose salary is less than average

```
update instructor
set salary = salary * 1.05
where salary <  (select avg (salary)
                 from instructor);
```

Updates

- Increase salaries of instructors whose salary is over ₹100,000 by 3%, and all others by a 5%
  - Write two **update** statements:

```
update instructor
   set salary = salary * 1.03
   where salary > 100000;

update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

  - The order is important
  - Can be done better using the **case** statement

Case Statement for Conditional Updates

- Same query as before but with case statement

```
update instructor
set salary = case
                when salary <= 100000 then salary * 1.05
                else salary * 1.03
             end
```

- The general form of the case statement is as follows.

```
case
    when pred1 then result1
    when pred2 then result2
    . . .
    when predn then resultn
else result0
end
```

Updates with Scalar Subqueries

- Scalar subqueries in SQL update statements can be used in the **set** clause
- Recompute and update tot_creds value for all students

```
update student S
set tot_cred = (select sum(credits)
                from takes, course
                where takes.course_id = course.course_id  and
                      S.ID= takes.ID.and
                      takes.grade <> 'F' and
                      takes.grade is not null);
```

- Sets *tot_creds* to null for students who have not taken any course
- Instead of **sum**(*credits*), use:

```
case
   when sum(credits) is not null then sum(credits)
   else 0
end
```

Joined Relations

Joined Relations

- **Join operations** take two relations and return as a result another relation

- A join operation is a **Cartesian product** that requires that tuples in the two relations match (under some conditions). It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the **from** clause
- Three types of joins:
  - Natural join
  - Inner join
  - Outer join

Natural Join in SQL

- Natural join **matches tuples with the same values for all common attributes** and retains only one copy of each common column

- List the names of instructors along with the course ID of the courses that they taught

```
select name, course_id
from  students, takes
where student.ID = takes.ID;
```

- Same query in SQL with “natural join” construct

```
select name, course_id
from student natural join takes;
```

Natural Join in SQL

- The from clause can have multiple relations combined using natural join:

  select *A1, A2, … An*
  from r_1 **natural join** r_2 **natural join** … **natural join** r_n
  where *P* ;

Takes Relation

ID	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	*null*

Student Relation

ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

student **natural join** *takes*

ID	*name*	*dept_name*	*tot_cred*	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B-
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A-
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	*null*

Join Condition

- The **on** condition allows a general predicate over the relations being joined
- This predicate is written like a **where** clause predicate except for the use of the keyword **on**

- Query example

```
select *
from student join takes on student_ID = takes_ID
```

  - The **on** condition above specifies that a tuple from *student* matches a tuple from *takes* if their *ID* values are equal

- Equivalent to:

```
select *
from student , takes
where student_ID = takes_ID
```

Outer Join

- An extension of the join operation that avoids loss of information
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join
- Uses *null* values.
- Three forms of outer join:
  - left outer join
  - right outer join
  - full outer join

Outer Join Example

- Relation *course*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- Relation *prereq*

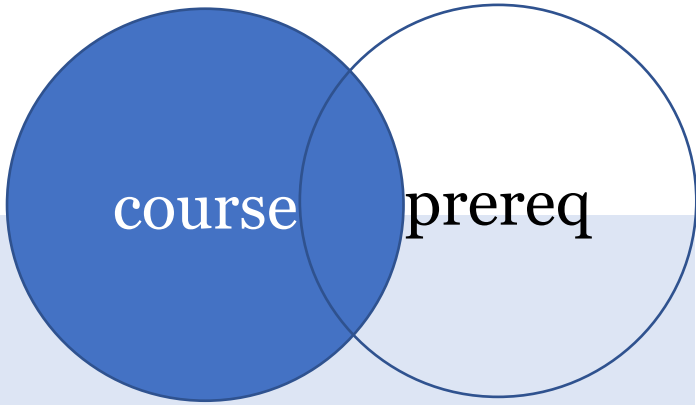
course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that

  course information is missing CS-347

  prereq information is missing CS-315

Left Outer Join



- *course* **natural left outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

- In relational algebra: *course* ⟕ *prereq*

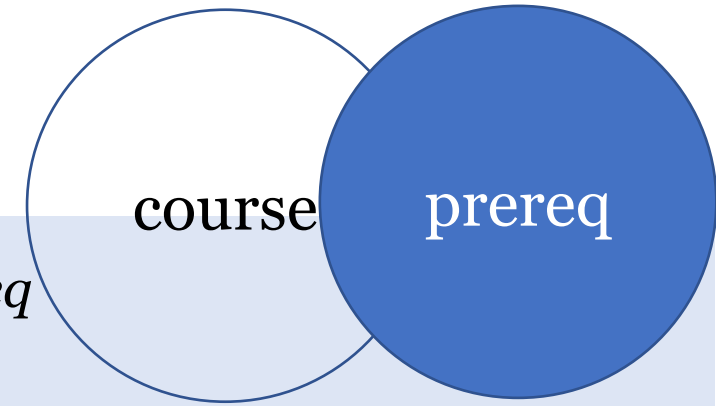
course

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Right Outer Join



- *course* **natural right outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

- In relational algebra: *course* ⟖ *prereq*

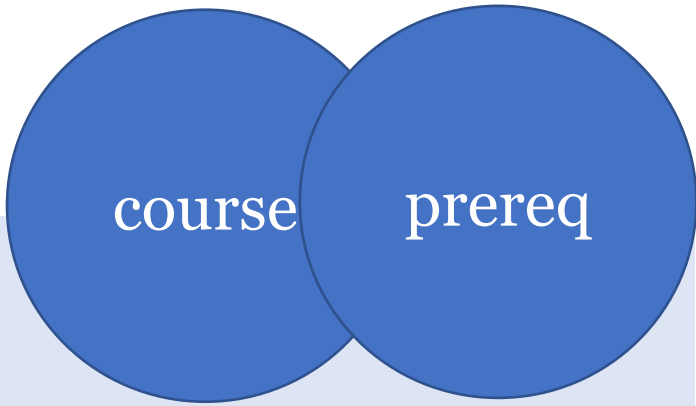
course

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Full Outer Join



- *course* **natural full outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

- In relational algebra: *course* ⟗ *prereq*

course

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Joined Types and Conditions

- **Join operations** take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- **Join condition** – defines which tuples in the two relations match.
- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated

Join types
inner join
left outer join
right outer join
full outer join

Join conditions
natural
on <predicate>
using $(A_1, A_2, \ldots, A_n)$

Integrity Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency
  - A checking account must have a balance greater than ₹10,000.00
  - A salary of a bank employee must be at least ₹4.00 an hour
  - A customer must have a (non-null) phone number

Constraints

- **not null**
- **primary key**
- **unique**
- **check** (P), where P is a predicate
- **default**
- **foreign key**

Not Null Constraints

- **not null**
  - Declare *name* and *budget* to be **not null**

    name **varchar**(20) **not null**
    budget **numeric**(12,2) **not null**

Unique Constraints

- **unique** ($A_1, A_2, \ldots, A_m$)
  - The unique specification states that the attributes A_1, A_2, …, A_m form a candidate key
  - No two tuples in the relation can be equal on all the listed attributes
  - Candidate keys are permitted to be null (in contrast to primary keys).

The Check Clause

- The **check** (P) clause specifies a predicate P that must be satisfied by every tuple in a relation.
- Example: ensure that the semester is one of fall, winter, spring, or summer

```
create table section
   (course_id varchar (8),
    sec_id varchar (8),
    semester varchar (6),
    year numeric (4,0),
    building varchar (15),
    room_number varchar (7),
    time slot id varchar (4),
     primary key (course_id, sec_id, semester, year),
     check (semester in ('Fall', 'Winter', 'Spring', 'Summer')))
```

Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: If "Biology" is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for "Biology"

- Let A be a set of attributes. Let R and S be two relations that contain attributes A and where A is the primary key of S. A is said to be a **foreign key** of R if for any values of A appearing in R these values also appear in S.

Referential Integrity

- Foreign *keys can be* specified as part of the SQL **create table** statement

  foreign key (*dept_name*) **references** *department*

- By default, a foreign key references the primary-key attributes of the referenced table

- SQL allows a list of attributes of the referenced relation to be specified explicitly

  foreign key (*dept_name*) **references** *department* (*dept_name*)

Cascading Actions in Referential Integrity

- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.
- An alternative, in case of delete or update is to cascade

```
create table course (
    (...
     dept_name varchar(20),
     foreign key (dept_name) references department
        on delete cascade
        on update cascade,
     . . .)
```

- Instead of cascade we can use :
  - **set null**,
  - **set default**

Complex Check Conditions

- The predicate in the check clause can be an arbitrary predicate that can include a subquery

  check (*time_slot_id* **in** (**select** *time_slot_id* **from** *time_slot*))

  The check condition states that the time_slot_id in each tuple in the *section* relation is actually the identifier of a time slot in the *time_slot* relation

  - The condition has to be checked not only when a tuple is inserted or modified in *section*, but also when the relation *time_slot* changes

Index Creation

- Many queries reference only a small proportion of the records in a table.
- It is inefficient for the system to read every record to find a record with a particular value
- An **index** on an attribute of a relation is a data structure that allows the database system to find those tuples in the relationships that have a specified value for that attribute efficiently, without scanning through all the tuples of the relation

- We create an index with the **create index** command

  create index <name> **on** <relation-name> (attribute);

Index Creation Example

- **create table** *student*
  (*ID* **varchar** (5),
  name **varchar** (20) **not null**,
  dept_name **varchar** (20),
  tot_cred **numeric** (3,0) **default** 0,
  primary key (*ID*))

- **create index** *studentID_index* **on** *student*(*ID*)

- The query:

  select *
  from *student*
  where *ID* = '12345'

  can be executed by using the index to find the required record, without looking at all records of *student*

Thanks!