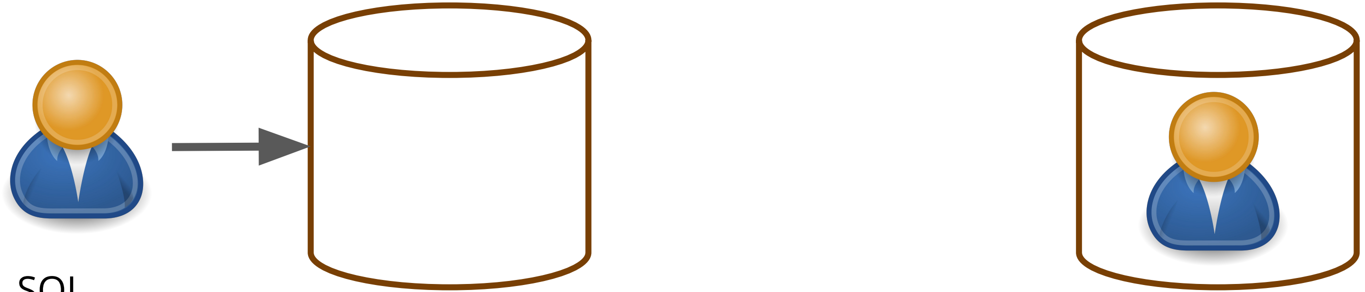
INFOH417 Database System Architectures

Mahmoud SAKR <mahmoud.sakr@ulb.be>

École polytechnique de Bruxelles

2021

What is this course for ?



SQL
Python
Web
…

Storage
Access control
Optimization
Distribution
…

Course Goals

- Understanding the query optimization and execution cycle
- Improving slow queries
- Describing the common index structures, knowing their capabilities and shortcomings
- Understanding cost based optimization, and the associated statistics and estimation methods
- Describing and being able to implement Abstract Data Types in extensible database systems
- Describing data and query distribution mechanisms, and being able to configure and run a distributed database system
- Comparing the architectures of SQL and NoSQL systems
- Comparing the architectures of database V.S. data flow systems

Course Topics

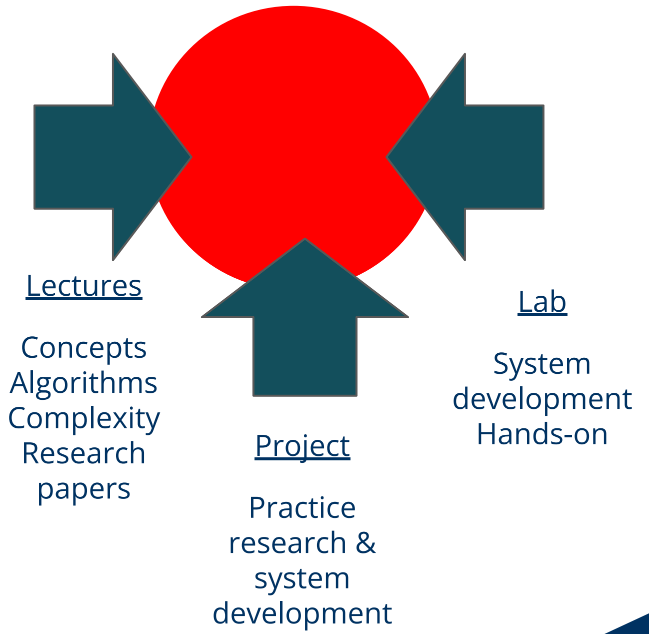
- Query Execution
- Refreshing SQL and Relational Algebra
- Query Optimization
- Indexes
- Advanced Databases
- Multidimensional index structures
- Distributed databases
- New DBMS Architectures (NoSQL)
- New Data Flow Architectures (Map-Reduce and Spark)

Prerequisites

- Relational databases
- SQL
- Relational Algebra
- General programming skills

Course Organization

- Lectures on Friday, S.UB4.132, 10h00-12h00
- Lab sessions on Thursday, S.UB4.130, 6x[10h00-12h00], 5x[16h00-18h00]
  - Taught by: Maxime Schoemans
  - make PostgreSQL from source
- Project
  - Implement data management features in PostgreSQL
- Check timeedit for schedule updates
- Grading
  - Group project, 4 members, 40%
  - Written exam, 60%
- Course notes, please enroll in Université virtuelle



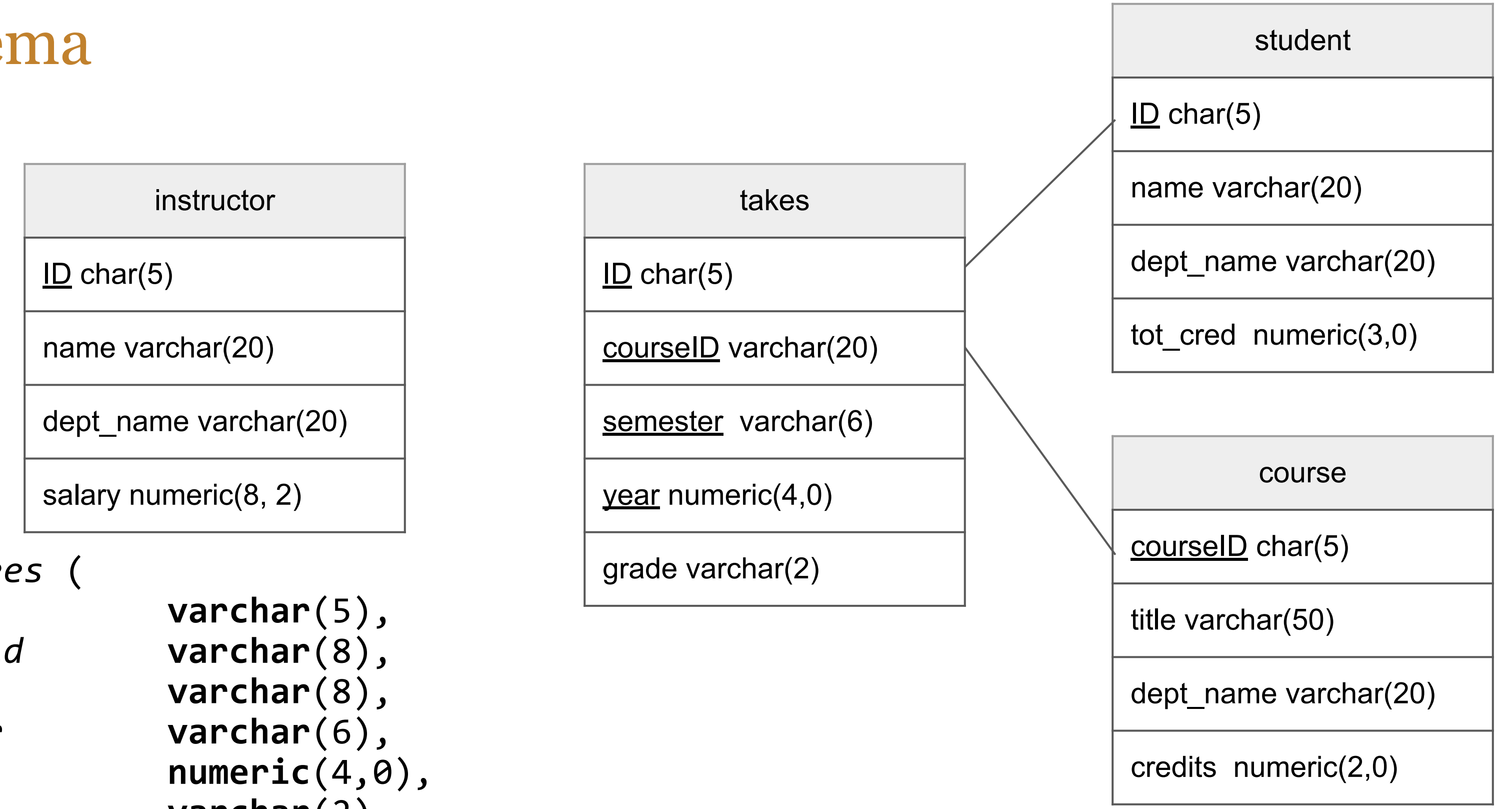
Recommended Readings

- A mixture of book chapters and research papers, which will be identified per lecture

Refreshing SQL

- Silberschatz, Korth and Sudarshan, Database Systems Concepts 6th Edition, Chapter 3.

Course Schema



```
create table takes (
        ID                varchar(5),
        course_id         varchar(8),
        sec_id            varchar(8),
        semester          varchar(6),
        year              numeric(4,0),
        grade             varchar(2),
        primary key (ID, course_id, semester, year) ,
        foreign key (ID) references  student,
        foreign key (course_id) references course);
```

Updates to tables

- **Insert**

  insert into *instructor* **values** ('10211', 'Smith', 'Biology', 66000);

- **Delete**
  - Remove all tuples from the *student* relation

    delete from *student*

- **Drop Table**

  drop table *r*

- **Alter**

  alter table *r* **add** *A D*

  - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*
  - All exiting tuples in the relation are assigned *null* as the value for the new attribute.

  alter table *r* **drop** *A*

  - where *A* is the name of an attribute of relation *r*
  - Dropping of attributes not supported by many databases.

The select Clause

Find the department names of all instructors, and remove duplicates

```
select distinct dept_name
from instructor
```

The keyword **all** specifies that duplicates should not be removed.

```
select all dept_name
from instructor
```

The select Clause (Cont.)

An asterisk in the select clause denotes "all attributes"

```
select *
from instructor
```

An attribute can be a literal with no **from** clause

```
select '437'

select '437' as FOO
```

The select Clause (Cont.)

The **select** clause can contain arithmetic expressions involving the operation, +, –, *, and /, and operating on constants or attributes of tuples.

```
select ID, name, salary/12 as monthlySalary
from instructor
```

The where Clause

To find all instructors in Comp. Sci. dept

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

Logical connectives **and, or,** and **not**

To find all instructors in Comp. Sci. dept with salary > 80000

The where Clause

To find all instructors in Comp. Sci. dept

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

Logical connectives **and, or,** and **not**

To find all instructors in Comp. Sci. dept with salary > 80000

```
select name
from instructor
where dept_name = 'Comp. Sci.'  and salary > 80000
```

The from Clause

Find the names of all instructors in the Art department who have taught some course and the course_id

```
select name, course_id
from instructor , teaches
where instructor.ID = teaches.ID  and  instructor.
dept_name = 'Art'
```

Self Join Example

Relation *emp-super*

person	supervisor
Bob	Alice
Mary	Susan
Alice	David
David	Marry

Find the supervisor of “Bob”

Find the supervisor of the supervisor of “Bob”

String Pattern matching

SIMILAR TO is similar to LIKE, and further supports regex operators

```
SELECT 'abc' LIKE 'abc';      true
SELECT 'abc' LIKE 'a%';       true
SELECT 'abc' LIKE '_b_';      true
SELECT 'abc' LIKE 'c';        false
```

```
SELECT 'abc' SIMILAR TO 'abc';            true
SELECT 'abc' SIMILAR TO 'a';              false
SELECT 'abc' SIMILAR TO '%(b|d)%';        true
SELECT 'aaac' SIMILAR TO 'a*b*c%';        true
SELECT 'aaac' SIMILAR TO 'a*b+c%';        false
```

https://www.postgresql.org/docs/13/functions-matching.html

Ordering the Display of Tuples

List in alphabetic order the names of all instructors

```
 select distinct name
from     instructor
order by name


order by name desc


order by  dept_name, name
```

Where Clause Predicates

Example: Find the names of all instructors with salary between \$90,000 and \$100,000 (that is, ≥ \$90,000 and ≤ \$100,000)

```
select name
from instructor
where salary between 90000 and 100000
```

Tuple comparison

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, ’Biology’);
```

Set Operations

Find courses that ran in Fall 2009 or in Spring 2010

```
(select course_id from section where sem = ‘Fall’ and year = 2009)
 union          ||
(select course_id from section where sem = ‘Spring’ and year = 2010)
```

Find courses that ran in Fall 2009 and in Spring 2010

```
(select course_id from section where sem = ‘Fall’ and year = 2009)
 intersect
(select course_id from section where sem = ‘Spring’ and year = 2010)
```

Find courses that ran in Fall 2009 but not in Spring 2010

```
(select course_id from section where sem = ‘Fall’ and year = 2009)
 except
(select course_id from section where sem = ‘Spring’ and year = 2010)
```

Set Operations (Cont.)

Set operations **union**, **intersect**, and **except**

Each of the above operations automatically eliminates duplicates

To retain all duplicates use the corresponding multiset versions **union all,** **intersect all** and **except all.**

Suppose a tuple occurs *m* times in *r* and *n* times in *s,* then, it occurs:

? times in *r* **union all** *s*

? times in *r* **intersect all** *s*

? times in *r* **except all** *s*

Set Operations (Cont.)

Set operations **union**, **intersect**, and **except**

Each of the above operations automatically eliminates duplicates

To retain all duplicates use the corresponding multiset versions **union all, intersect all** and **except all.**

Suppose a tuple occurs *m* times in *r* and *n* times in *s,* then, it occurs:

m + *n* times in *r* **union all** *s*

min(*m,n)* times in *r* **intersect all** *s*

max(0, *m* – *n)* times in *r* **except all** *s*

Null Values

null signifies an unknown value or that a value does not exist.

The result of any arithmetic expression involving *null* is *null*

Example: 5 + *null* returns null

The predicate **is null** can be used to check for null values.

Example: Find all instructors whose salary is null.

```
select name
from instructor
where salary is null
```

Null Values and Three Valued Logic

NULL	AND	OR	NOT
TRUE	NULL	TRUE	NULL
FALSE	FALSE	~~FALSE~~ Null	
NULL	NULL	NULL	

Aggregate Functions

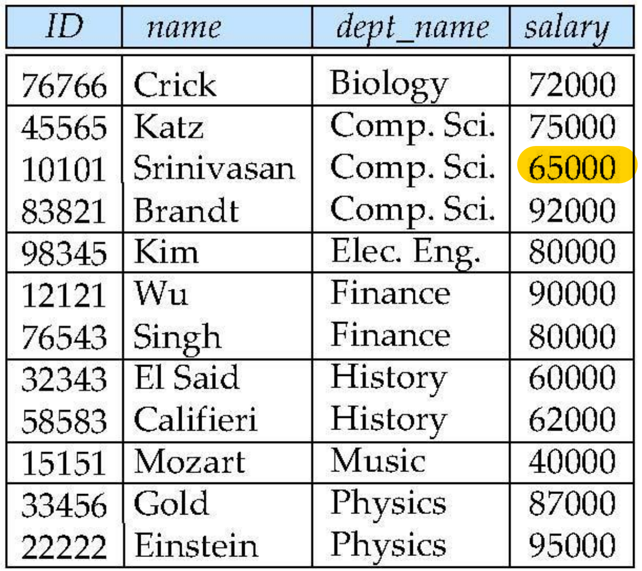
These functions operate on the multiset of values of a column of a relation, and return a value

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

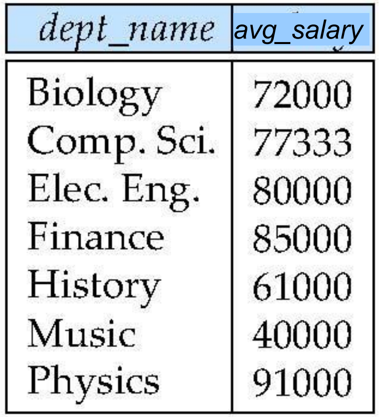
Aggregate Functions – Group By

Find the average salary of instructors in each department

```
select dept_name, avg (salary) as avg_salary
from instructor
group by dept_name;
```



ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000



dept_name	avg_salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregation (Cont.)

Attributes in **select** clause outside of aggregate functions must appear in **group by** list

```
/* erroneous query */
select dept_name, ID, avg (salary)
from instructor
group by dept_name;
```

Why ?

To have one average for each group formed by group by.

Aggregate Functions – Having Clause

Find the names and average salaries of all departments whose average salary is greater than 42000

```
select dept_name, avg (salary)  as moyen
from instructor
group by dept_name
having avg (salary) > 42000;
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

Null Values and Aggregates

Total all salaries

```
select sum (salary )
from instructor
```

Above statement ignores null amounts

Result is *null* if there is no non-null amount

All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes

What if collection has only null values?

count returns 0

all other aggregates return null

Nested Subqueries

The nesting can be done in the following SQL query

> **select** $A_1, A_2, \ldots, A_n$
> **from** $r_1, r_2, \ldots, r_m$
> **where** P

as follows:

> A_i can be replaced be a subquery that generates a single value.
>
> r_i can be replaced by any valid subquery
>
> P can be replaced with an expression of the form:
>
> *B* <operation> (subquery)
>
> Where *B* is an attribute and <operation> to be defined later.

Set Membership

Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = ’Fall’ and year= 2009 and
  course_id in (select course_id
                from section
                where semester = ’Spring’ and year= 2010);
```

Find courses offered in Fall 2009 but not in Spring 2010

Set Membership

Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = ’Fall’ and year= 2009 and
  course_id in (select course_id
                 from section
                 where semester = ’Spring’ and year= 2010);
```

Find courses offered in Fall 2009 but not in Spring 2010

```
select distinct course_id
from section
where semester = ’Fall’ and year= 2009 and
  course_id  not in (select course_id
                      from section
                      where semester = ’Spring’ and year= 2010);
```

Set Comparison – "some" Clause

Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept name = ’Biology’;
```

Same query using > **some** clause

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept name = ’Biology’);
```

Definition of "some" Clause

- F <comp> **some** $r \Leftrightarrow \exists\, t \in r$ such that (F <comp> t)
  Where <comp> can be: $<,\ \leq,\ >,\ =,\ \neq$

$(5 < \textbf{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{true}$ (read: 5 < some tuple in the relation)

$(5 < \textbf{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{false}$

$(5 = \textbf{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true}$

$(5 \neq \textbf{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true (since } 0 \neq 5)$

(= **some**) ≡ **in**
However, (≠ **some**) $\not\equiv$ **not in**

Set Comparison – “all” Clause

Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

Set Comparison – “all” Clause

Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept name = ’Biology’);
```

Definition of "all" Clause

F <comp> **all** *r* ⇔ ∀ *t* ∈ *r* (F <comp> *t*)

$$(5 < \textbf{all} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{false}$$

$$(5 < \textbf{all} \begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$$

$$(5 = \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$

$$(5 \neq \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$$

(≠ all) ≡ not in

However, **(= all) ≢ in**

Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

Use of “exists” Clause

Yet another way of specifying the query “Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester”

```
 select course_id
from section as S
where semester = ’Fall’ and year = 2009 and
  exists ???
```

Use of "exists" Clause

Yet another way of specifying the query "Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester"

```
 select course_id
from section as S
where semester = 'Fall' and year = 2009 and
  exists (select *
          from section as T
          where semester = 'Spring' and year= 2010
                and S.course_id = T.course_id);
```

Correlation name – variable S in the outer query

Correlated subquery – the inner query

Use of “not exists” Clause

Find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = ’Biology’)
                   except
                   (select T.course_id
                    from takes as T
                    where S.ID = T.ID));
```

- First nested query lists all courses offered in Biology
- Second nested query lists all courses a particular student took

Note that $X - Y = \emptyset \Leftrightarrow X \subseteq Y$

Note: Cannot write this query using = **all** and its variants

Test for Absence of Duplicate Tuples

The **unique** construct tests whether a subquery has any duplicate tuples in its result.

The **unique** construct evaluates to “true” if a given subquery contains no duplicates .

Find all courses that were offered at most once in 2009

```
select T.course_id
from course as T
where unique (select R.course_id
              from section as R
              where T.course_id= R.course_id
                    and R.year = 2009);
```

Subqueries in the Form Clause

Find the average instructors' salaries of those departments where the average salary is greater than $42,000."

```
    select dept_name, avg_salary
    from (select dept_name, avg (salary) as avg_salary
          from instructor
          group by dept_name)   as xxx
    where avg_salary > 42000;
```

Note that we do not need to use the **having** clause

With Clause CTE

The **with** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs.

Find all departments with the maximum budget

```
with max_budget (value) as
   (select max(budget)
    from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```

Subqueries in the Select Clause

Scalar subquery is one which is used where a single value is expected

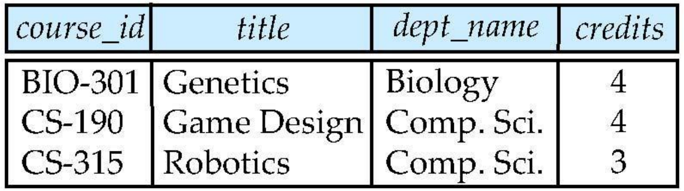
List all departments along with the number of instructors in each department

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

Runtime error if subquery returns more than one result tuple

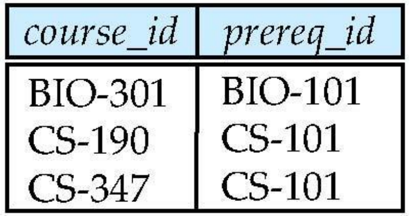
Join operations – Example

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

prereq information is missing for CS-315 and

course information is missing for CS-437

Left Outer Join

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

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

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

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*

Right Outer Join

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

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

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

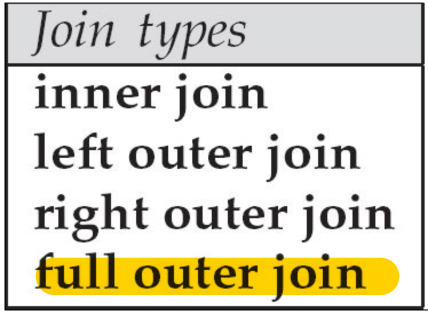
course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

Outer Join

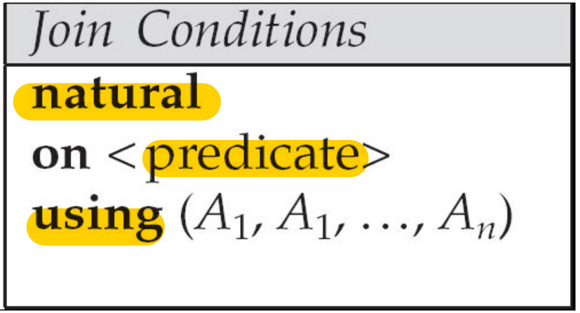
- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values.

Joined Relations

- **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, and what attributes are present in the result of the join.
- **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 $(A_1, A_1, \ldots, A_n)$

Full Outer Join

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

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

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

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

Credits

- The SQL slides come from:
- Silberschatz, Korth and Sudarshan, Database Systems Concepts 6th Edition, Ch3.