

RAWDATA

Section 1

SQL part 1

Henrik Bulskov & Troels Andreasen

Introduction to SQL

- Basic Query Structure (DML)
- Additional Basic Operations (DML)
- Null Values (DML)
- Aggregate Functions (DML)
- Nested Subqueries (DML)
- Modification of the Database (DML)
- Data Definition (DDL)

History

- IBM SEQUEL language developed as part of System R project at the IBM San Jose Research Laboratory during the 1970s
- SEQUEL = Structured English QUERy Language
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
 - SQL-86, SQL-89, SQL-92
 - SQL:1999, SQL:2003, SQL:2008, SQL:2011, **SQL:2016**
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and – in addition – special proprietary features.
- Notice
 - Not all examples in the book (and in these slides) will work on every particular system
 - Most will work with MySQL, though

Workbench

MySQL Workbench

Local instance MySQL57

File Edit View Query Database Server Tools Scripting Help

Navigator: Query 1

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

SCHEMAS

- Filter objects
- movie
 - Tables
 - Views
 - Stored Procedures
 - Functions
- sys

No object selected

Query 1

```
1 • use movie;
2 • select * from person;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows: | Result Grid | Form Editor | Field Types | Query Stats

	pid	name	gender
▶	2	\$, Homo	m
	4	\$hort, Too	m
	5	\$lim, Bee Moe	m
	7	\$torm, Cntry	m
	16	'Ariffin, Syaiful	m
	17	'Aruhane	m
	18	'Atu'ake, Taipaleti	m
	19	'Avacado' Wolfe, David	m
	20	'bableepower' Viera, Michael	m
	21	'Bear'Boyd, Steven	m
	22	'bliss' Vilbon, Kirlew	m
	24	'Bootsy' Thomas, George	m
...

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	06:55:02	use movie	0 row(s) affected	0.000 sec
2	06:55:04	select *from person LIMIT 0, 1000	1000 row(s) returned	0.000 sec / 0.000 sec

Workbench

Setup New Connection

Connection Name: my-islb-connection Type a name for the connection

Connection Method: Standard (TCP/IP) Method to use to connect to the RDBMS

Parameters SSL Advanced

Hostname: islbruc.dk Port: 3306 Name or IP address of the server host - and TCP/IP port.

Username: guest Name of the user to connect with.

Password: The user's password. Will be requested later if it's not set.

Default Schema: movie The schema to use as default schema. Leave blank to select it later.

Configure Server Management... Test Connection Cancel OK

Query Completed

MANAGEMENT
Server Client Users Status Data E Data I
INSTANCE
Startup Server Options
PERFORMANCE
Dashboard Performance Performance
SCHEMAS
movie
Table View Stored Procedure Function
ssu
st
stackover
Object
No object selected

- 3 tables: movie, casting, person:

- **movie(mid, title, production_year)**
- **casting(mid, pid)**
- **person(pid, name, gender)**

A movie database

From SQLwarmup (exercises)

Find the full version of this database on islb.ruc.dk (connect as guest/guest) or make your local copy by importing from `movie_database.sql`

Not all rows from the 3 tables are shown here

Full content are:

- movie: 887.047 rows
- casting: 6.995.056 rows
- person: 2.422.836 rows

movie

mid	title	production_year
2438281	Accordion Player	1888
2546319	Brighton Street Scene	1888
2831850	Hyde Park Corner	1889
3004390	Man Walking Around the Corner	1887
3127044	Passage de Venus	1874
3205178	Roundhay Garden Scene	1888
3214201	Sallie Gardner at a Gallop	1878
3462286	Traffic Crossing Leeds Bridge	1888

casting

pid	mid
1158190	2438281
1158190	3205178
2197644	3205178
2743436	3205178
3489247	3205178

person

pid	name	gender
1158190	Le Prince, Adolphe	m
2197644	Whitley, Joseph	m
2743436	Hartley, Annie	f
3489247	Whitley, Sarah	f

Basic Query Structure

- The SQL **data-manipulation language (DML)** provides the ability to query information, and insert, delete and update tuples
- A typical SQL query has the form:

```
select  $A_1, A_2, \dots, A_n$ 
from  $r_1, r_2, \dots, r_m$ 
where  $P$ 
```

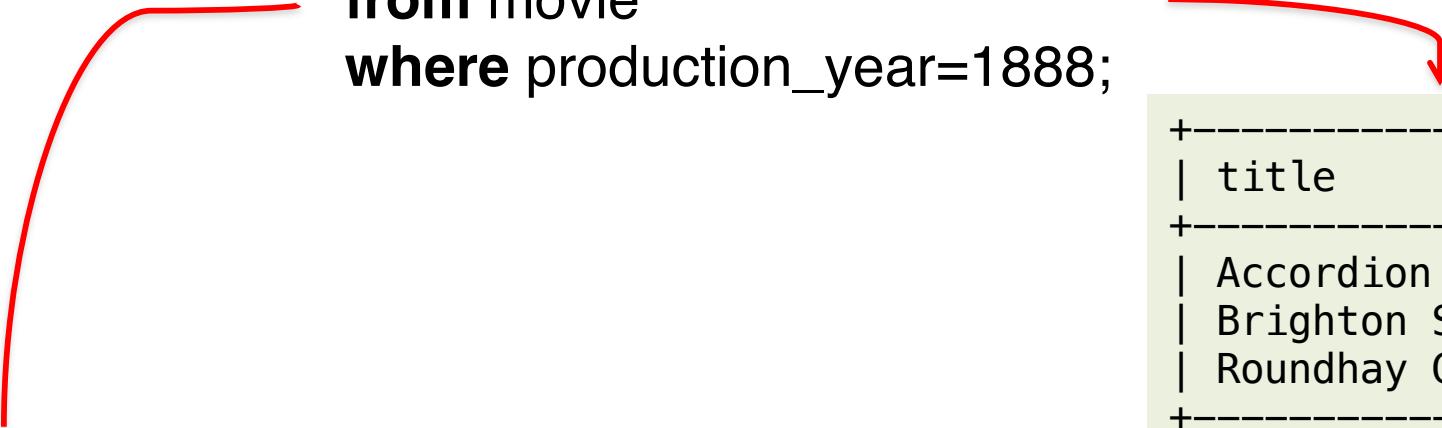
- A_i represents an attribute
- R_i represents a relation
- P is a predicate.

- The result of an SQL query is a relation.

Example Query

- A typical SQL query has the form:

```
select title  
from movie  
where production_year=1888;
```



title
Accordion Player
Brighton Street Scene
Roundhay Garden Scene

movie		
mid	title	production_year
2438281	Accordion Player	1888
2546319	Brighton Street Scene	1888
2831850	Hyde Park Corner	1889
3004390	Man Walking Around the Corner	1887
3127044	Passage de Venus	1874
3205178	Roundhay Garden Scene	1888
3214201	Sallie Gardner at a Gallop	1878
3462286	Traffic Crossing Leeds Bridge	1889

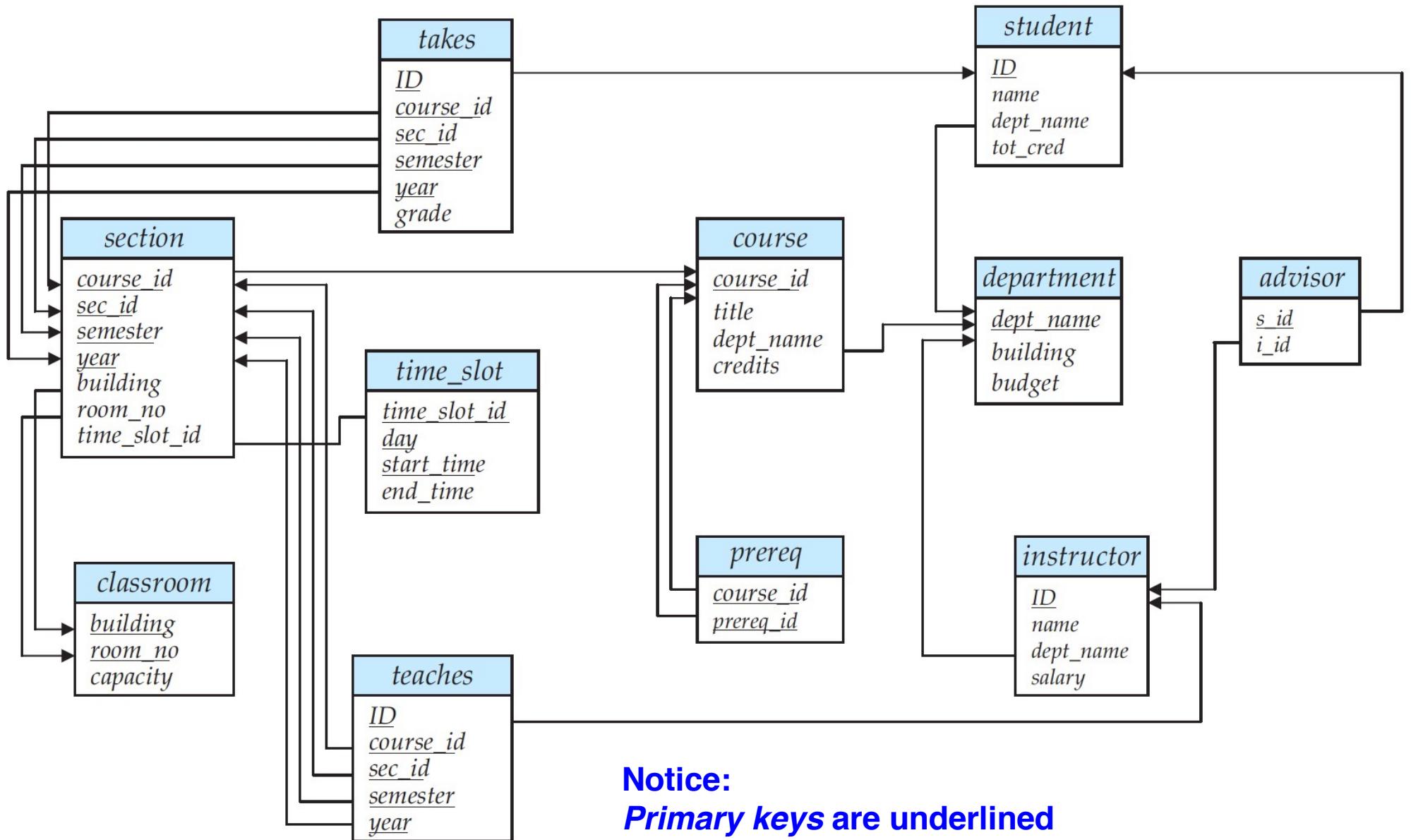
More examples

- select title, production_year
from movie
where production_year=1888;**
- select title
from movie
where production_year<1900;**
- select mid, title, production_year
from movie;**

movie		
mid	title	production_year
2438281	Accordion Player	1888
2546319	Brighton Street Scene	1888
2831850	Hyde Park Corner	1889
3004390	Man Walking Around the Corner	1887
3127044	Passage de Venus	1874
3205178	Roundhay Garden Scene	1888
3214201	Sallie Gardner at a Gallop	1878
3462286	Traffic Crossing Leeds Bridge	1889

Results?

A University Database



Primary and foreing keys

- Informal definitions ...
- **Primary key**
 - Column / attribute that identifies rows in the table
- **Foreing key**
 - Column / attribute that points to the primary key in another table
- Both can also be multiple columns/attributes
- We return to these later

- What would a similar diagram over the simple movie database look like?

The select Clause

- The **select** clause list the attributes desired in the result of a query
- Example: find the names of all instructors:

```
select name  
from instructor
```

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters interchangeably)
 - E.g. $Name \equiv NAME \equiv name$

```
mysql> select name from instructor;  
+-----+  
| name |  
+-----+  
| Srinivasan |  
| Wu |  
| Mozart |  
| Einstein |  
| El Said |  
| Gold |  
| Katz |  
| Califieri |  
| Singh |  
| Crick |  
| Brandt |
```

The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select.
- Find the names of all departments with instructor, and remove duplicates

```
select distinct dept_name ←  
      from instructor
```

Obviously needed when
duplicates are default
(**all** is default)

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

```
select all dept_name ←  
      from instructor
```

Not really useful since
all is default

The select Clause (Cont.)

```
mysql> select dept_name from instructor;  
+-----+  
| dept_name |  
+-----+  
| Biology   |  
| Comp. Sci. |  
| Comp. Sci. |  
| Comp. Sci. |  
| Elec. Eng. |  
| Finance   |  
| Finance   |  
| History   |  
| History   |  
| Music     |  
| Physics   |  
| Physics   |  
+-----+  
12 rows in set (0.00 sec)
```

```
mysql> select distinct dept_name from instructor;  
+-----+  
| dept_name |  
+-----+  
| Biology   |  
| Comp. Sci. |  
| Elec. Eng. |  
| Finance   |  
| History   |  
| Music     |  
| Physics   |  
+-----+  
7 rows in set (0.00 sec)
```

The select Clause (Cont.)

- An **asterisk *** in the select clause denotes “all attributes”

```
select *  
from instructor
```
- The **select** clause can contain **arithmetic expressions** involving the operation, +, −, *, and /, and operating on constants or attributes of tuples.
- The query:

```
select ID, name, dept_name, salary/12  
from instructor
```

would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.

The where Clause

- The **where** clause specifies conditions that each row in the result must satisfy
- Conditions can be combined using the **logical connectives and, or, and not.**
- Example
 - To find all instructors in Comp. Sci. dept with salary > 80000

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

The where Clause

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

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

```
mysql> select name from instructor
where dept_name = 'Comp. Sci.' and salary > 80000;
+-----+
| name |
+-----+
| Brandt |
+-----+
1 row in set (0.01 sec)
```

instructor:

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

SQLwarmup exercises, first part 1-10

- 1) List all titles of movies produced in year 1888
- 2) List title and year of movies produced in year 2019 or later
- 3) List title and year of movies produced in the 1930'ies.
(Hint: you can combine conditions using AND)
- 4) Find the names of all female actors. (Hint: gender should be "f")
- 5) Find the title and production year of all movies with a title starting with 007.
Hint: rather than "=" you can use the operator "like" and by specifying e.g. "An%" you will match all values that start with "An"
- 6) Count the number of movies.
Hint: rather than selecting an attribute you can simply select using a function "count(*)"
- 7) Count the number of movies produced in 2004. Hint: use "count(*)" again, but now in a query with a where-clause
- 8) Find the name and gender of all persons with a name starting with "Mikkelsen"
- 9) Find the name of the person with 7767 as pid
- 10) Find all mid's for movies that have casted the person with pid=7767

The from Clause

- The **from** clause lists the relations involved in the query

- Two tables in the **from** clause

```
select *  
from instructor, teaches
```

- generates every possible *instructor – teaches* pair, with all attributes from both relations
- This is called the **Cartesian product** of *instructor* and *teaches*

- Cartesian product is

- not very useful directly, but
- often useful combined with where-clause condition

Cartesian Product of: *instructor* and *teaches*

instructor

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
...

teaches

<i>ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

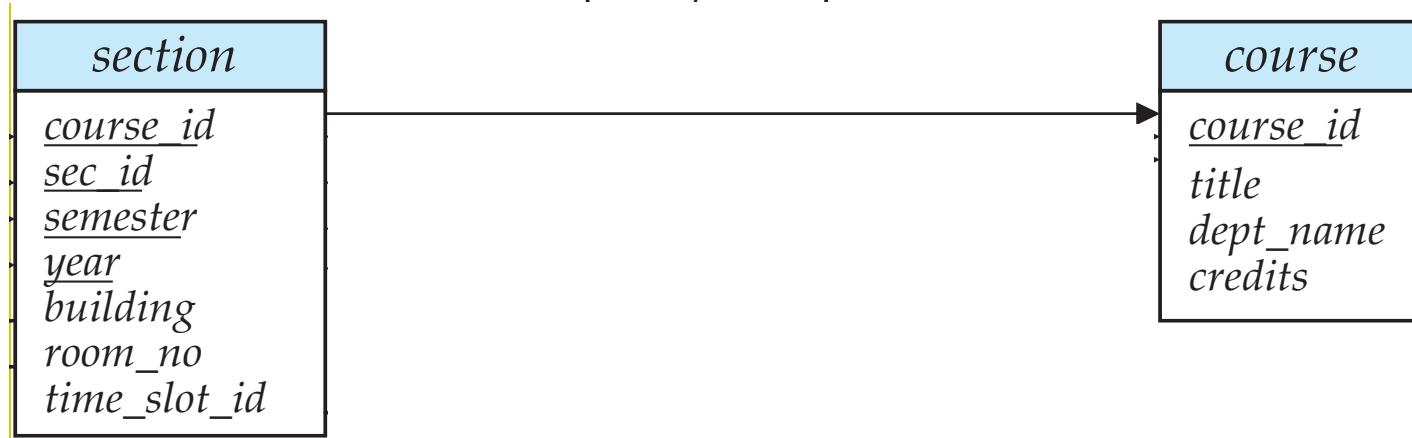
<i>inst.ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>teaches.ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...
...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...
...

Joins

- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

```
select section.course_id, semester, year, title
      from section, course
     where section.course_id = course.course_id and
           dept_name = 'Comp. Sci.'
```

From schema page 10



Joins

- For all instructors who have taught some course, find their names and the course ID of the courses they taught.

```
select name, course_id  
from instructor, teaches  
where instructor.ID = teaches.ID
```

From schema page 10



- The condition *instructor.ID = teaches.ID* is kind of “natural” here

Natural Join

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column
- **select ***
from *instructor* natural join *teaches*;

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

Natural Join Example

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

```
select name, course_id  
from instructor, teaches  
where instructor.ID = teaches.ID;
```

```
select name, course_id  
from instructor natural join teaches;
```

Natural Join (Cont.)

- Danger in natural join: **beware of unrelated attributes with same name which may get equated incorrectly**
- List the names of instructors along with the titles of courses that they teach
 - Incorrect version (makes course.dept_name = instructor.dept_name)
**select name, title
from instructor natural join teaches natural join course;**
 - Correct version
**select name, title
from instructor natural join teaches, course
where teaches.course_id = course.course_id;**
 - Another correct version
**select name, title
from (instructor natural join teaches)
join course using(course_id);**

The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:

old-name as new-name

—or just

old-name new-name

—thus keyword **as** is optional and may be omitted

- E.g.

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

- or just

```
select ID, name, salary/12 monthly_salary  
from instructor
```

```
mysql> select ID, name, salary/12 as monthly_salary from instructor;  
+----+-----+-----+  
| ID | name | monthly_salary |  
+----+-----+-----+  
| 10101 | Srinivasan | 5416.666667 |  
| 12121 | Wu | 7500.000000 |
```

```
mysql> select ID, name, salary/12 from instructor;
```

```
+----+-----+-----+  
| ID | name | salary/12 |  
+----+-----+-----+  
| 10101 | Srinivasan | 5416.666667 |  
| 12121 | Wu | 7500.000000 |
```

The Rename Operation

- Yet another example

```
select distinct T.name the_name
from instructor T, instructor S
where T.salary > S.salary and S.dept_name = 'Comp. Sci.'
```

- what is this?

```
mysql> select distinct T.name the_name
from instructor T, instructor S
where T.salary > S.salary and S.dept_name = 'Comp. Sci.';
```

```
+-----+
| the_name |
+-----+
| Wu
| Einstein
| Gold
| Katz
| Singh
| Crick
| Brandt
| Kim
+-----+
8 rows in set (0.01 sec)
```

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

String Operations (LIKE)

- SQL includes a string-matching operator for comparisons on character strings. The operator “like” uses patterns that are described using two special characters:
 - percent (%). The % character matches any substring.
 - underscore (_). The _ character matches any character.
- Find the names of all instructors whose name includes the substring “ri”.

```
select name  
from instructor  
where name like '%ri%'
```

```
mysql> select name from instructor where name like '%ri%';  
+-----+  
| name |  
+-----+  
| Srinivasan |  
| Califieri |  
| Crick |  
+-----+  
3 rows in set (0.00 sec)
```

String Operations (LIKE)

- Pattern matching examples:
 - ‘Intro%’ matches any string beginning with “Intro”.
 - ‘%Comp%’ matches any string containing “Comp” as a substring.
 - ‘___’ matches any string of exactly three characters.
 - ‘___ %’ matches what?
- Patterns are by default not case sensitive in MySQL.

String Operations (Cont.)

- SQL supports a variety of string operations such as
 - concatenation using function:
 - concat()
 - converting from upper to lower case (and vice versa)
 - upper() and lower()
 - finding string length, extracting substrings, etc.
 - length(), substring()

```
mysql> select length(name),concat(upper(name),'_',substr(dept_name,1,2))
from instructor;
+-----+-----+
| length(name) | concat(upper(name),'_',substr(dept_name,1,2)) |
+-----+-----+
|      10 | SRINIVASAN_Co
|       2 | WU_Fi
|       6 | MOZART_Mu
|       8 | EINSTEIN_Ph
|       7 | EL_SAID_Hi
|       4 | GOLD_Ph
|       4 | KATZ_Co
|       0 | GATTERTD_Hi
+-----+-----+
```

Ordering the Display of Tuples

- List in alphabetic order the names of all instructors

```
select distinct name  
from   instructor  
order by name
```

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
 - Example: **order by name desc**
- Can sort on multiple attributes
 - Example: **order by dept_name, name**

Where Clause Predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between \$90,000 and \$100,000 (that is, $\geq \$90,000$ and $\leq \$100,000$)

```
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 semester = 'Fall' and year = 2009)
union
(select course_id from section where semester = 'Spring' and year = 2010)
```

- Notice
 - **union** automatically eliminates duplicates
 - **union all** can be used to retain all duplicates

- Observe that the SQL standard also define
 - intersect** and **except**
 - but these are **not** supported in MySQL

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*
 - Example: $5 + \text{null}$ returns null
- In SQL the predicate **is null** can be used to check for null values.
 - Example: Find all instructors whose *dept_name* is null.

```
select *
from instructor
where dept_name is null
```

```
mysql> insert into instructor(id,name,salary) values ("12345","Merkel",77777);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM instructor where dept_name is null;
+-----+-----+-----+
| ID   | name    | dept_name | salary   |
+-----+-----+-----+
| 12345 | Merkel | NULL      | 77777.00 |
+-----+-----+-----+
```

Null Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
 - Example: $5 < \text{null}$ or $\text{null} \neq \text{null}$ or $\text{null} = \text{null}$
- Three-valued logic using the truth value *unknown*:
 - OR: (*unknown or true*) = *true*,
 (*unknown or false*) = *unknown*
 (*unknown or unknown*) = *unknown*
 - AND: (*true and unknown*) = *unknown*,
 (*false and unknown*) = *false*,
 (*unknown and unknown*) = *unknown*
 - NOT: (**not** *unknown*) = *unknown*
 - “**P is unknown**” evaluates to true if predicate *P* evaluates to *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

Null Values and Three Valued Logic

```
mysql> select * from instructor where dept_name is unknown;
+----+-----+-----+-----+
| ID | name | dept_name | salary |
+----+-----+-----+-----+
| 12345 | Merkel | NULL | 77777.00 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from instructor where dept_name='aaa' is
unknown;
+----+-----+-----+-----+
| ID | name | dept_name | salary |
+----+-----+-----+-----+
| 12345 | Merkel | NULL | 77777.00 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

Aggregate Functions

- These functions operate on the multiset (bag) 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 (Cont.)

- Find the average salary of instructors in the Computer Science department

```
select avg (salary)
from instructor
where dept_name= 'Comp. Sci.';
```

- Find the total number of instructors who teach a course in the Spring 2010 semester

```
select count (distinct ID)
from teaches
where semester = 'Spring' and year = 2010
```

- Find the number of tuples in the *course* relation

```
select count (*)
from course;
```

Aggregate Functions – Group By

- Find the average salary of instructors in each department

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

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

```
select * from instructor;
```

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

- Note: departments with no instructor will not appear in result

Aggregation (Cont.)

- Standard SQL Group By
 - Attributes in **select** clause outside of aggregate functions must be identical to attributes in **group by** list, as in
 - **select dept_name, avg (salary)**
from instructor
group by dept_name;
- MySQL: also allow this
 - **select avg (salary)**
from instructor
group by dept_name;
- What would be the interpretation of this variant?

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)
from instructor
group by dept_name
having avg (salary) > 42000;
```

- why not just use **where**?

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

Null Values and Aggregates

- ❑ One *null*-salary:

```
mysql> insert into instructor(id,name,dept_name)
      values ("12345","Merkel","Physics");
```

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

```
mysql> select count(id) from
instructor;
+-----+
| count(id) |
+-----+
|      13   |
+-----+
```

```
mysql> select count(salary)
from instructor;
+-----+
| count(salary) |
+-----+
|      12      |
+-----+
```

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
12345	Merkel	Physics	NULL
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

SQL - exercises

- You are strongly recommended to test your solutions to the SQL-queries in the exercises on the university database with a MySQL database (accessed from MySQL Workbench – or command prompt, if you prefer)
- Preparation – two alternatives to do experiments with the university-database and the movie-database
 - use **islb.ruc.dk**
 - observe islb is read-only (exercise 3.12 involve modifications)
 - use **your local MySQL server** (localhost)
 - download and install the schemas university-database.sql and movie_database.sql from Moodle

Data

- Give names of the female actors that have played most roles
- Give names of actors that have acted together with "Mikkelsen, Mads"

Not all rows from the 3 tables are shown here

Full content are:

- movie: 887.047 rows
- casting: 6.995.056 rows
- person: 2.422.836 rows

movie

mid	title	production_year
2438281	Accordion Player	1888
2546319	Brighton Street Scene	1888
2831850	Hyde Park Corner	1889
3004390	Man Walking Around the Corner	1887
3127044	Passage de Venus	1874
3205178	Roundhay Garden Scene	1888
3214201	Sallie Gardner at a Gallop	1878
3462286	Traffic Crossing Leeds Bridge	1888

casting

pid	mid
1158190	2438281
1158190	3205178
2197644	3205178
2743436	3205178
3489247	3205178

person

pid	name	gender
1158190	Le Prince, Adolphe	m
2197644	Whitley, Joseph	m
2743436	Hartley, Annie	f
3489247	Whitley, Sarah	f

3.1 (a,c), 3.11(a)

3.1 Write the following queries in SQL, using the university schema. (We suggest you actually run these queries on a database, using the sample data that we provide on the Web site of the book, db-book.com. Instructions for setting up a database, and loading sample data, are provided on the above Web site.)

- a. Find the titles of courses in the Comp. Sci. department that have 3 credits.
- c. Find the highest salary of any instructor.

3.11 Write the following queries in SQL, using the university schema.

- a. Find the names of all students who have taken at least one Comp. Sci. course; make sure there are no duplicate names in the result.