RAWDATA
Section 1

SQL part 1

Henrik Bulskov & Troels Andreasen

Introduction to SQL

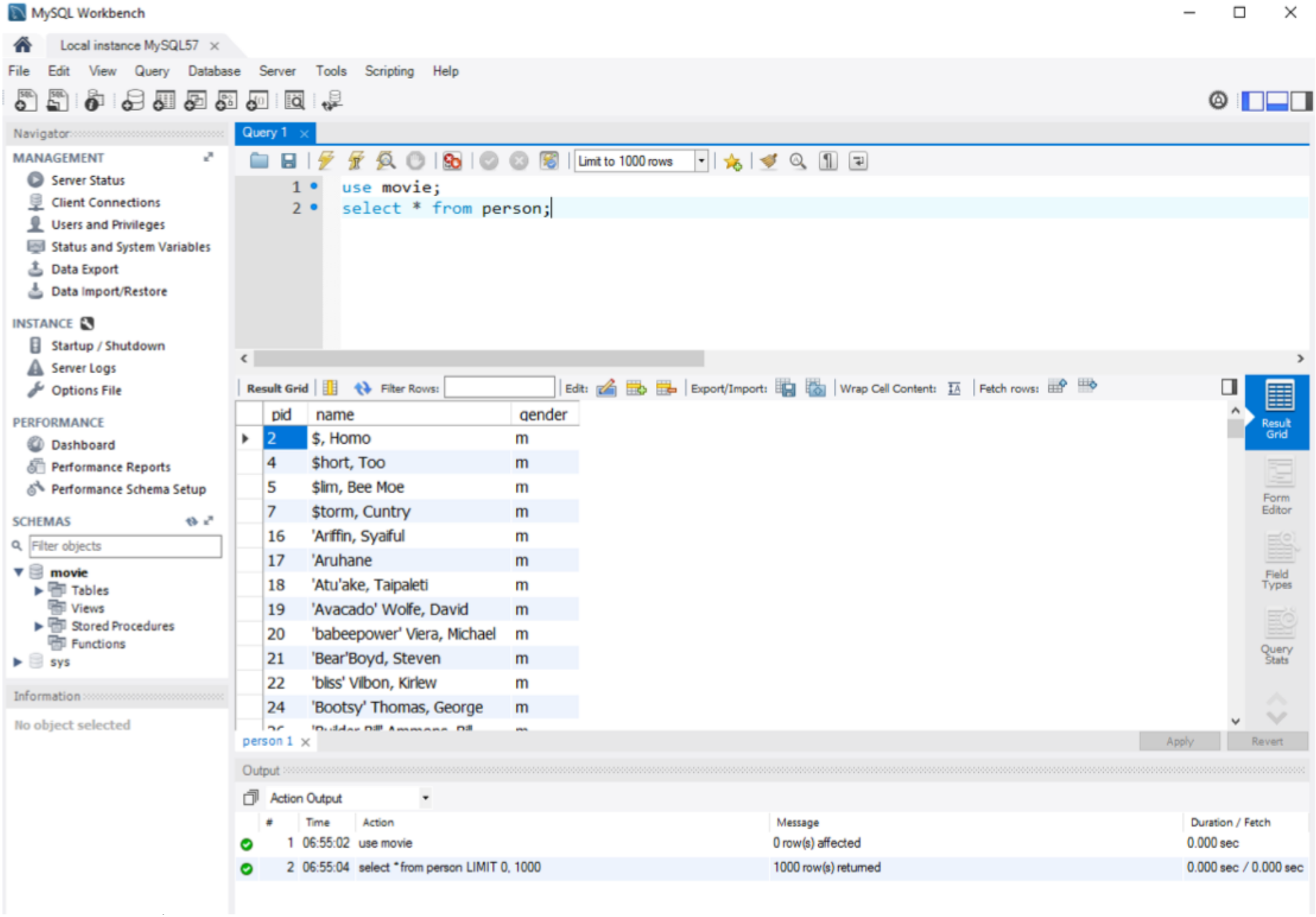
- ❑ Basic Query Structure (DML)
- ❑ Additional Basic Operations (DML)
- ❑ Null Values (DML)
- ❑ Aggregate Functions (DML)
- ❑ Nested Subqueries (DML)
- ❑ Modification of the Database (DML)
- ❑ Data Definition (DDL)

History

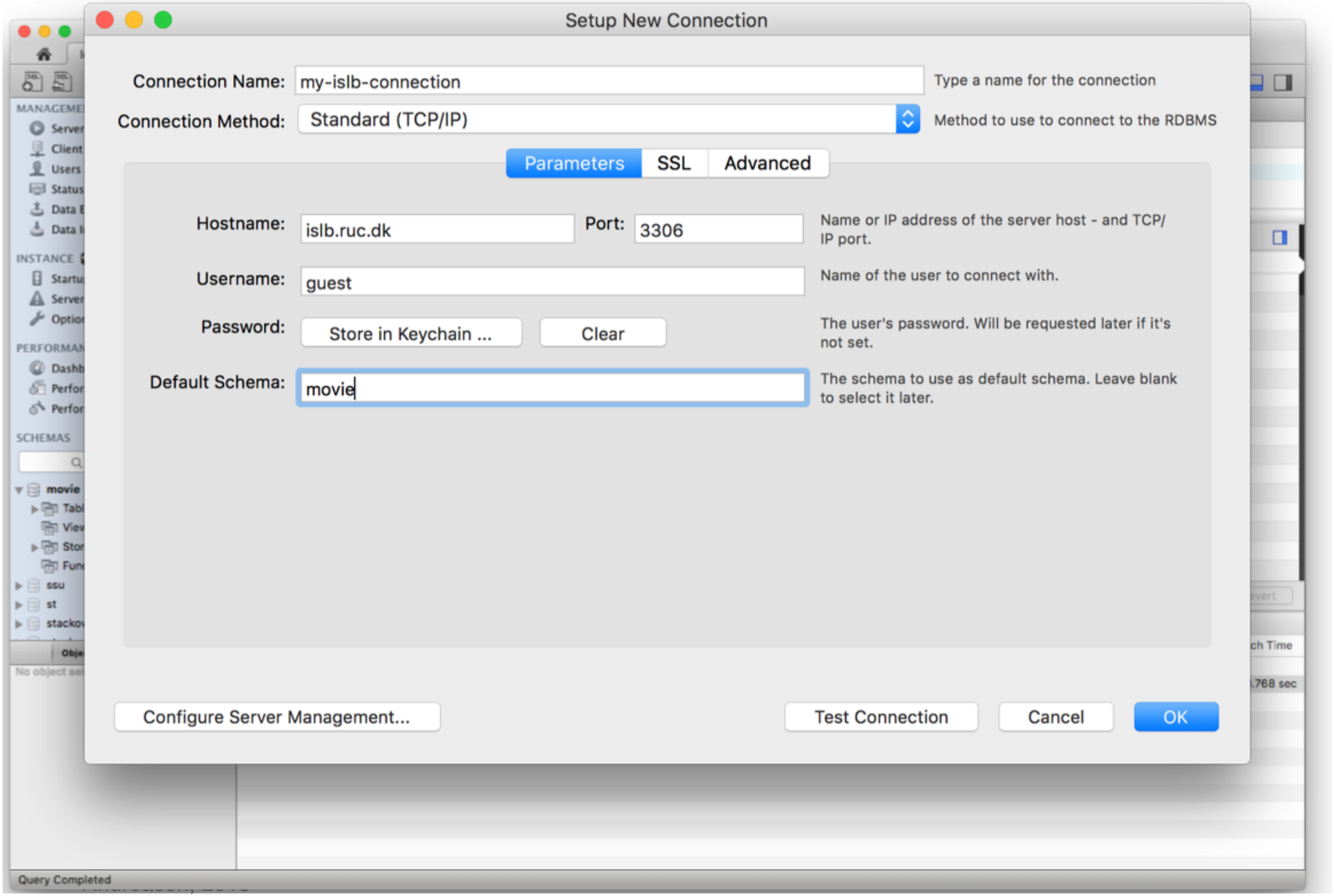
- IBM SEQUEL language developed as part of System R project at the IBM San Jose Research Laboratory during the 1970s
- SEQUEL = Structured English QUEry Language
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86, SQL-89, SQL-92
  - SQL:1999, SQL:2003, SQL:2008, SQL:2011, **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
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
Information
No object selected
Query 1
Limit to 1000 rows
1 use movie;
2 select * from person;
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
Fetch rows:
pid name gender
2 $, Homo m
4 $hort, Too m
5 $lim, Bee Moe m
7 $torm, Cuntry m
16 'Ariffin, Syaiful m
17 'Aruhane m
18 'Atu'ake, Taipaleti m
19 'Avacado' Wolfe, David m
20 'babeepower' Viera, Michael m
21 'Bear'Boyd, Steven m
22 'bliss' Vilbon, Kirlew m
24 'Bootsy' Thomas, George m
person 1
Apply
Revert
Result Grid
Form Editor
Field Types
Query Stats
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:
islb.ruc.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:
Store in Keychain ...
Clear
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

A movie database

From SQLwarmup (exercises)

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

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:

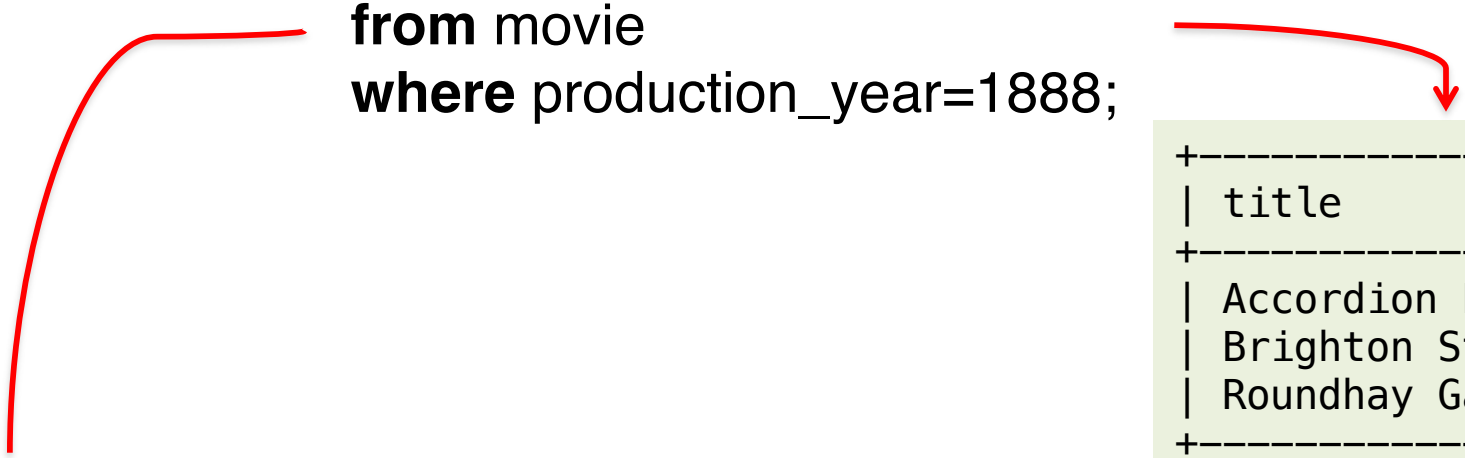
select A_1, A_2, ..., A_n
from r_1, r_2, ..., r_m
where P

  - 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;

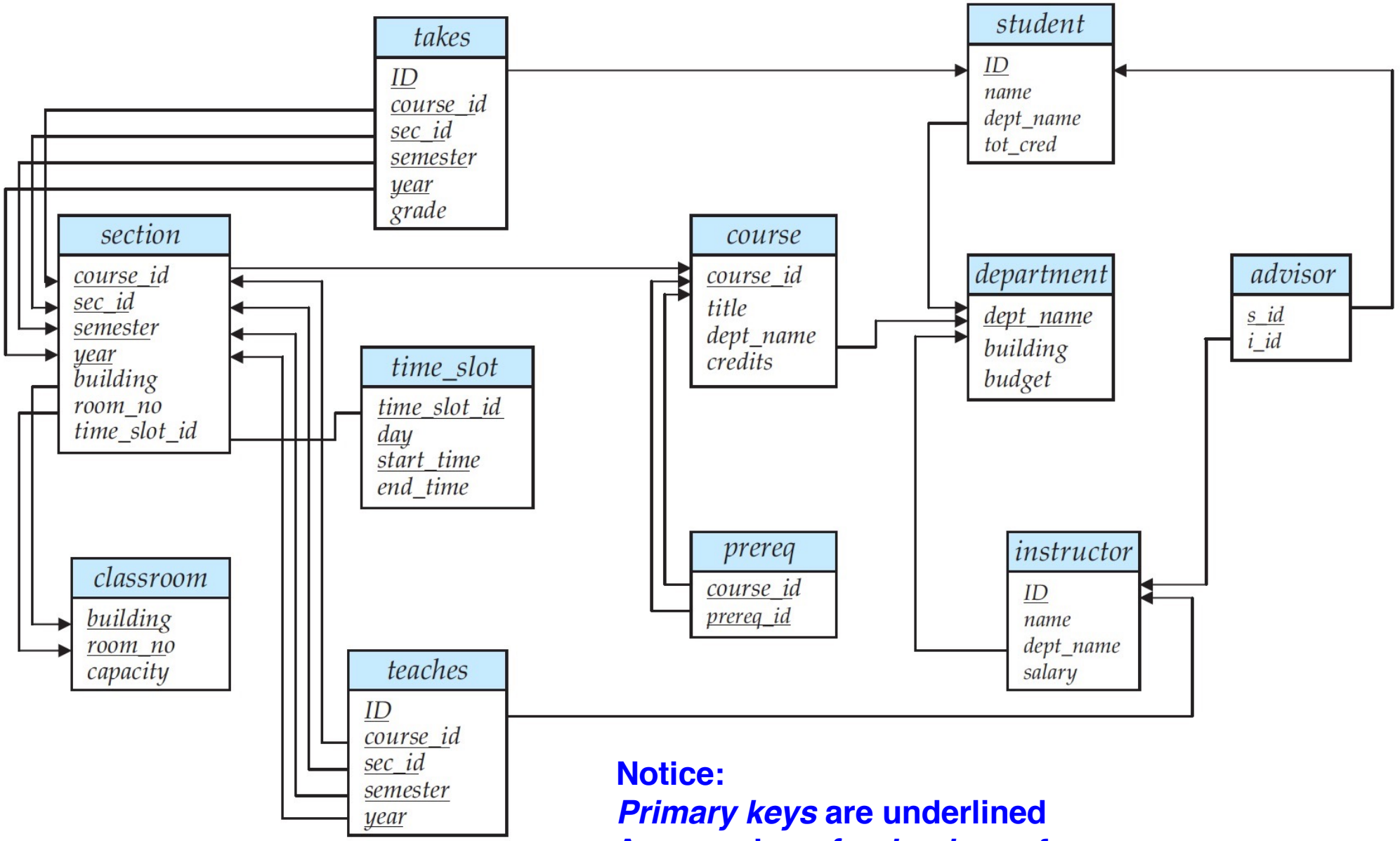
- **select** mid, title, production_year
  from movie;

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



Notice:
***Primary keys* are underlined**
Arrows show *foreign key* references

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* ≡ *NAME* ≡ *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 se
```

```
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:

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

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

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

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
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

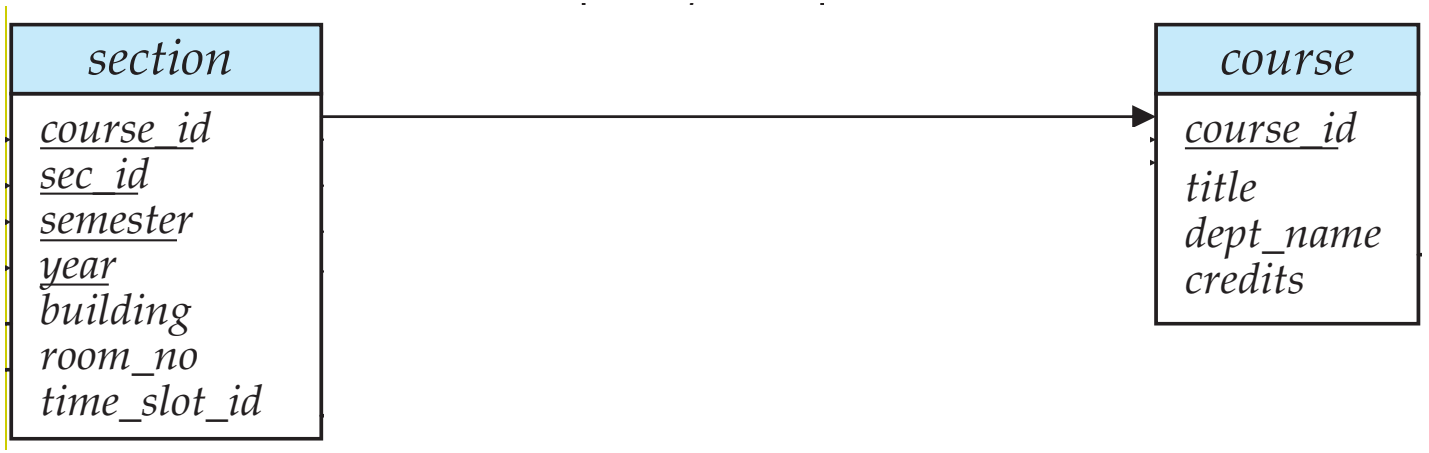
inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	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*;

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
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 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         |
```

```
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                                 |
|            9 | CALIFIERI_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, ≥ $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');
```

How to express this - without the tuple comparison?

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 + *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**

Why not just? :
where *dept_name* **= 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 < null or null <> null or null = 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

  Why distinct?

- 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

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.

Example queries on the movie database

- 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

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A **subquery** is a **select-from-where** expression that is nested within another query.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

- Nested Subqueries **in where-clause**, variants
  - **in**, **not in,**
  - **some, all,**
  - **exists, not exists**
  - **unique**
- Nested Subqueries **in from-clause**, variants
  - simple parenthetic expression
- **Others**
  - (**with-**clause)
  - scalar subqueries

Example Query (in, not in)

- 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);
```

Example Query ("tuple in relation")

- Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
                    (select course_id, sec_id, semester, year
                     from teaches
                     where teaches.ID= 10101);
```

- Note: Above query can be written in a much simpler manner. The formulation above is simply to illustrate SQL features.

Set Comparison (some)

- Find names of instructors with salary greater than the salary 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 < **some** | 0 / 5 / 6 |) = true (read: 5 < some tuple in the relation)

(5 < **some** | 0 / 5 |) = false

(5 = **some** | 0 / 5 |) = true

(5 ≠ **some** | 0 / 5 |) = true (since 5 ≠ 0)

Notice that (= **some**) ≡ **in**

So do we also have that (≠ **some**) ≡ **not in ?**

Example Query (all)

- 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** $r \Leftrightarrow \forall\ t \in r$ (F <comp> *t)*

(5 < **all** | 0 / 5 / 6 |) = false

(5 < **all** | 6 / 10 |) = true

(5 = **all** | 4 / 5 |) = false

(5 ≠ **all** | 4 / 6 |) = true (since 5 ≠ 4 and 5 ≠ 6)

Notice that (≠ **all**) ≡ **not in**

Is then also (= **all**) ≡ **in ?**

Test for Empty Relations (exists, not exists)

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

Correlation Variables (exists)

- 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);
```

- **Alias/tuple variable (like S and T above)**
  - **variable that refers to a relation (also called correlation name)**
- **Correlated subquery**
  - **subquery that refers to outer query (often using alias)**

Subqueries in the From Clause

- SQL allows a subquery expression to be used in the **from** 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) depts
where avg_salary > 42000;
```

- Note that we do not need to use the **having** clause in this example

- However the version with having is slightly more readable

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

- Notice: Subqueries in from must be renamed in MySQL

Scalar Subquery

- Scalar subquery is one which is used where a single value is expected
- in **select:**

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

- in **where:**

```
select name
from instructor
where  salary * 10 >
    (select budget  from department
      where department.dept_name = instructor.dept_name)
```

- Runtime error if subquery returns more than one result tuple

3.1 (d)

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.

b. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.

c. Find the highest salary of any instructor.

d. Find all instructors earning the highest salary (there may be more than one with the same salary).

e. Find the enrollment of each section that was offered in Autumn 2009.

f. Find the maximum enrollment, across all sections, in Autumn 2009.

g. Find the sections that had the maximum enrollment in Autumn 2009.

Data Definition Language

The SQL **data-definition language (DDL)** allows the specification of information about relations, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints

- In addition
  - The set of indices to be maintained for each relations.
  - Security and authorization information for each relation.
  - The physical storage structure of each relation on disk.

Create Table Construct

- An SQL relation is defined using the **create table** command:

  create table *r* (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - *r* is the name of the relation
  - each A_i is an attribute name in the schema of relation *r*
  - D_i is the data type of values in the domain of attribute A_i
- Example:

  create table *instructor* (
  ID **char**(5),
  name **varchar**(20) **not null,**
  dept_name **varchar**(20),
  salary **numeric**(8,2));

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

Domain Types in SQL

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point.
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.
- More are covered later (chapter 4)

Integrity Constraints in Create Table

- **not null**
- **primary key** (A_1, ..., A_n)
- **foreign key** (A_m, ..., A_n) **references** *r*

Example: Declare *ID* as the primary key
Declare *dept_name* as the foreing key

```
create table instructor (
    ID              char(5),
    name            varchar(20) not null,
    dept_name       varchar(20),
    salary          numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references department (dept_name))
```

- **primary key** declaration on an attribute automatically ensures **not null**
- **SQL Standard:** primary key specification in referenced relation is optional
- **MySQL syntax: primary key in referenced relation required**

And a Few More Relation Definitions

- **create table** *student* (
  ID **varchar**(5),
  name **varchar**(20) not null,
  dept_name **varchar**(20),
  tot_cred **numeric**(3,0),
  primary key (*ID*),
  foreign key *(dept_name*) **references** *department (dept_name))* ;
- **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, sec_id, semester, year),*
  foreign key (*ID*) **references** *student (ID),*
  foreign key (*course_id, sec_id, semester, year*)
  references *section (course_id, sec_id, semester, year)*);
  - Note: *sec_id* can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same course in the same semester

And more still

- **create table** *course* (
  course_id **varchar**(8) **primary key**,
  title **varchar(**50),
  dept_name **varchar**(20),
  credits **numeric**(2,0),
  foreign key *(dept_name*) **references** *department)*);
  - Primary key declaration can be combined with attribute declaration as shown above

DDL Example (from university_database.sql)

```
create table department
        (dept_name              varchar(20),
         building               varchar(15),
         budget                 numeric(12,2) check (budget > 0),
         primary key (dept_name)
        );

create table course
        (course_id              varchar(8),
         title                  varchar(50),
         dept_name              varchar(20),
         credits                numeric(2,0) check (credits > 0),
         primary key (course_id),
         foreign key (dept_name) references department (dept_name)
                on delete set null
        );

create table instructor
        (ID                     varchar(5),
         name                   varchar(20) not null,
         dept_name              varchar(20),
         salary                 numeric(8,2) check (salary > 29000),
         primary key (ID),
         foreign key (dept_name) references department (dept_name)
                on delete set null
        );
…
```

Drop and Alter Table Constructs

- **drop table [if exists]** *student*
  - Deletes the table and its contents

- **alter table**
  - **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 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

```
alter table student add dept_name2 varchar(20);
alter table student drop dept_name2;
```

Modification of the Database

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

Modification of the Database – Deletion

- 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 (Cont.)

- 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 deposit, 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)
  - But not allowed in MySQL

Modification of the Database – 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);

- When some values are unknown

  insert into *course* (*course_id*, *title*, *credits*)
  values ('CS-437', 'Database Systems', 4);

  What about dept_name for this new tuple?

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

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

Insertion (Cont.)

- Add all instructors to the *student* relation with tot_creds set to 0

  insert into *student*
  select *ID, name, dept_name, 0*
  from *instructor*

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

  insert into *table*1 **select** * **from** *table*1

  would cause problems, if *table1* did not have any primary key defined.

Modification of the Database – Updates

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others receive a 5% raise
  - 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 here …

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
```

Updates with Scalar Subqueries

- Recompute and update tot_creds value for all students

```
update student S
   set tot_cred = ( select sum(credits)
                    from takes natural join course
                    where 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
```