

CMSC 424: Database Design

Introduction to databases

Relational Model

SQL

Why databases?

- An example from my research: Clinical data from diarrhea study
- Need to store data about:
 - Individual patients: age, clinical parameters, disease state, etc.
 - Laboratory results: microbiology, virology, parasitology, etc.
 - DNA samples: concentration, location on plate, etc.
 - Results of computational analysis: files, number of sequences in files, etc.
- Do you really need a database?

Flat-file solution

- Samples spreadsheet (Samples.csv)
 - sample identifier
 - country
 - case status
- Laboratory spreadsheet (Lab_out.csv)
 - clinical parameters
 - drugs administered
 - results of laboratory tests
- Computational analysis spreadsheet (454.csv)
 - DNA information
 - File name
 - # of sequences detected

Flat-file solution

- Why not all the data in one file?
 - file can get very big
 - different people may want to update the information separately (doctor, lab technician, bioinformatician)
- How do we match the files to each other?
 - all files need to refer to the same set of IDs (in our case "Sample ID")
- Will this work well enough?
 - Perhaps...depending on how you use it

Flat-file solution: querying

- Find the identifiers for patients from Gambia that have Giardia
 - fairly easy: all data in one file
`grep 'Gambia .* Giardia' samples.csv | cut -f 1`
 - Note: I need to know exactly how the file is organized
- Find the age of all sick children from Gambia
 - a bit harder: information is in two different files
`join -1 1 -2 1 Lab_Out.csv samples.csv | grep 'Gambia' | cut -f 7`
 - Note: again I need to know exactly how the files are organized
 - Note: I also need to keep the files sorted by the identifier, otherwise the 'join' command doesn't work

Flat-file solution: querying

- Find the average number of sequences found in children under 2 that have cholera
 - Huh?
 - Now I need to join information from three files
 - and do some math
 - perhaps I'll have to write some code
- Note: there's a lot of stuff you can do with just simple command-line operations in Unix

Flat-file solution: updating

- How do I add information about a new patient?
 - I need to update all three files
 - Make sure to not have any typos
- How about if I want to add more fields to one or more files (e.g. a new laboratory test)
 - I need to make sure I don't mess up the order of the columns in any of the files (otherwise my scripts won't work)
- How do I ensure certain constraints are met?
 - some fields should never be empty (e.g. identifier)
 - other fields must have reasonable values (e.g. $97 < \text{body temperature} < 110$)
 - records are sorted by their identifier (so that join command works)
 - age is a numeric value

The better solution

- Database management systems handle all the challenges we just encountered
- They also handle a lot more
 - Atomicity: certain groups of operations must operate as one – either they all succeed, or the whole block fails

For example: if I add a new patient to the three files, I want the record to be added to all, or none, even if the system crashes as I'm adding the information

- Durability: once an operation succeeds, the state of the database is appropriately changed

For example: I add a new patient to the three files, tell the user that I did, then the system crashes. When the system comes up it better have the new patient in the database.

DBMS...cont

- Databases also handle:
 - Concurrency: adequately handle multiple simultaneous requests

Example: A doctor and a lab technician want to simultaneously add the same patient to the database.

- The doctor looks in the sample spreadsheet and does not find the patient
- The doctor generates a new identifier (1001)
- The lab technician looks in the same spreadsheet and does not find the patient
- The lab technician generates a new identifier (1002)
- The doctor records the patient's information in the samples spreadsheet under identifier 1001
- The lab technician records the patient's information in the Lab_out spreadsheet under identifier 1002

DBMS...cont

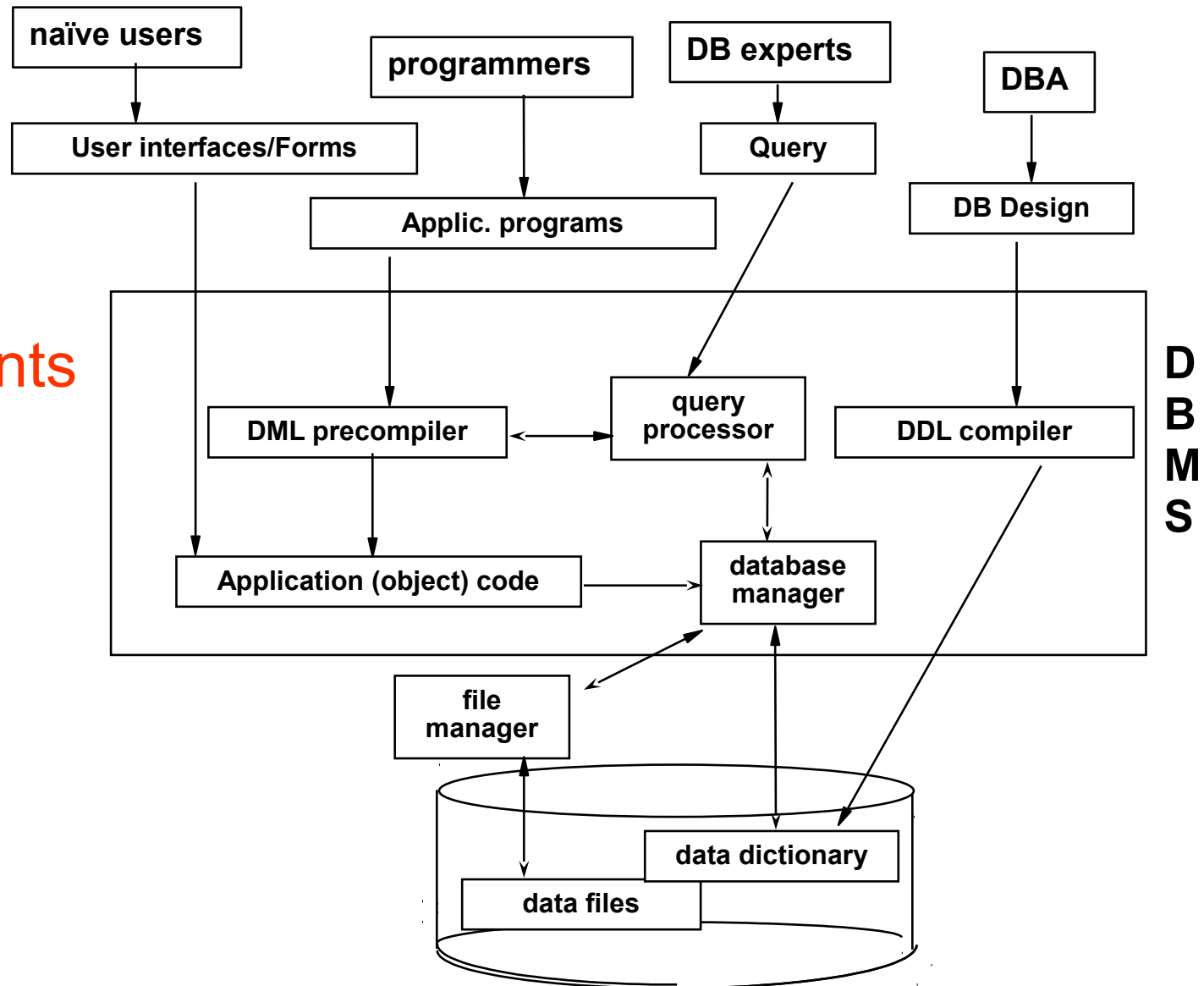
- Databases also handle
 - Security: only authorized users can see the data
 - Privacy: a user may only see the data that they need to access (e.g. lab technician cannot see clinical parameters)
- Most of these features are difficult or impossible to implement in a flat-file system

DBMS at a Glance

- Data Modeling
How the data are represented
- Data Retrieval
How the data are accessed/queried
- Data Storage
How the data are organized on disk
- Data Integrity
How concurrency and crashes are handled

Architecture of a DBMS

System Components & Interfaces



DML: Data Manipulation Language

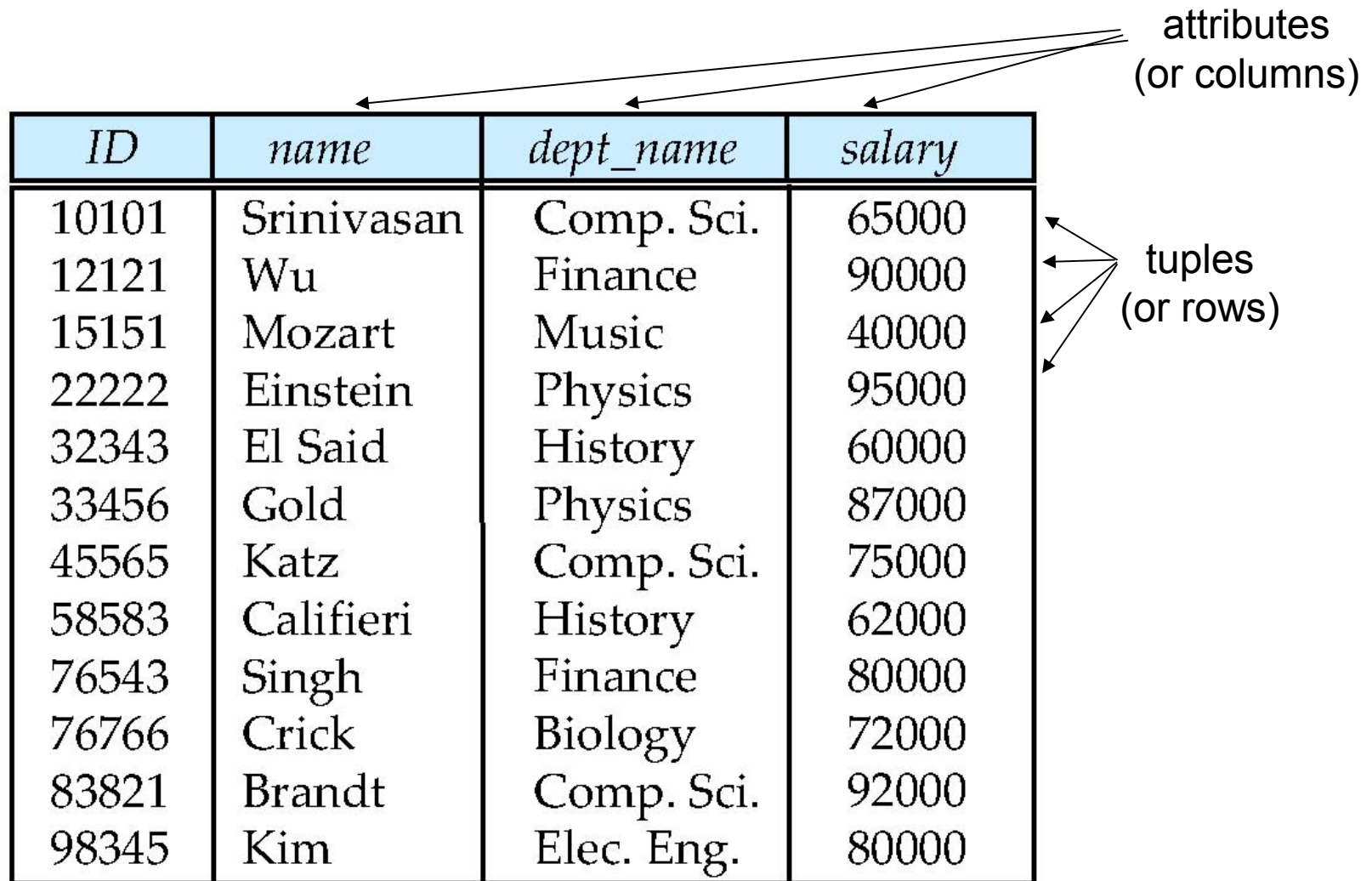
DDL: Data Definition Language

DBA: Data Base Administrator

Relational databases

- Organize the data in a series of "tables" or "relations"
- Conceptually same as the spreadsheets we used in our early example
- The column headers are called "attributes"
- The elements in the tables are called "rows" or "tuples"

Example of a Relation



The diagram illustrates a relation table with four columns and 12 rows. The columns are labeled *ID*, *name*, *dept_name*, and *salary*. The rows contain data for various individuals, including Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, and Kim. Annotations with arrows point to the columns, labeled "attributes (or columns)", and to the rows, labeled "tuples (or rows)".

<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

More about relational model

- Tuples assumed to be unordered
- Attributes not necessarily unique (many rows can have same value in a column)
- Key – an attribute, or combination of attributes, that can be used to select a unique tuple

e.g. the Sample ID field in my example
or Street name + Street number + Apartment number for an address

- Foreign key constraint/relationship – link between the keys of two or more tables

Relation Schema and Instance

- A_1, A_2, \dots, A_n are *attributes*
- $R = (A_1, A_2, \dots, A_n)$ is a *relation schema*

Example:

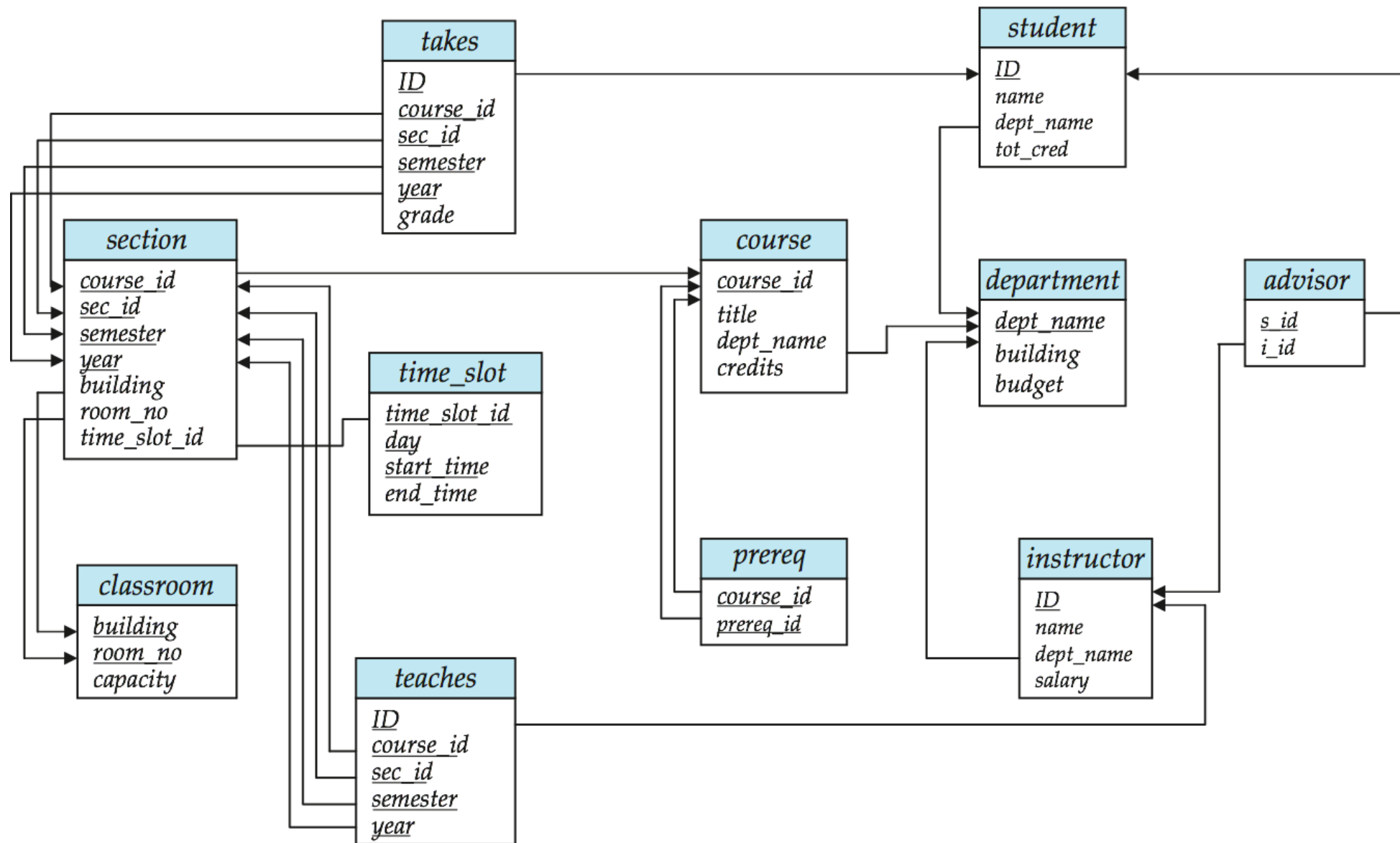
instructor = (*ID*, *name*, *dept_name*, *salary*)

- Formally, given sets D_1, D_2, \dots, D_n a **relation** r is a subset of

$$D_1 \times D_2 \times \dots \times D_n$$

Thus, a relation is a set of n -tuples (a_1, a_2, \dots, a_n) where each $a_i \in D_i$

Schema Diagram for University Database



Basic Query Structure

- 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.

The select Clause

- The **select** clause list the attributes desired in the result of a query
 - corresponds to the projection operation of the relational algebra
- 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.)
 - E.g., *Name* \equiv *NAME* \equiv *name*
 - Some people use upper case wherever we use bold font.

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
```

- The keyword **all** specifies that duplicates 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*

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

select *ID, 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 the result must satisfy
 - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept with salary > 80000

select *name*

from *instructor*

where *dept_name* = 'Comp. Sci.' **and** *salary* >

80000

- Comparison results can be combined using the logical connectives **and**, **or**, and **not**.
- Comparisons can be applied to results of arithmetic expressions.

The from Clause

- The **from** clause lists the relations involved in the query
 - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*
select *
from *instructor, teaches*
 - generates every possible instructor – teaches pair, with all attributes from both relations.
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

Cartesian Product

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

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
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

[illegible]

Getting started

- Oracle (available on grace systems)
\$ sqlplus
- MySQL (free software)
\$ mysql
- From here on it's a command line interface

Getting started...cont

- Select database

use database ;

- Note: commands end with ;

Database resources

- <http://www.dbis.informatik.uni-goettingen.de/Mondial/>
- You can use MySQL (easy to do at home)
- But... must ensure that code runs in Oracle on grace
- Some useful software:
 - MySQL Workbench – <http://wb.mysql.com>
Database management tool, also allows you to build schemas
 - Xampp - <http://sourceforge.net/projects/xampp/>
Includes webserver, mysql, php, etc. particularly useful for the project
 - PHPMyAdmin – http://www.phpmyadmin.net/home_page/index.php
Web-based administration of MySQL database