Advanced Data Management (CSCI 680/490)

Databases

Dr. David Koop

Python Features

- Iterators: for loops use to go through elements
  - `it = iter(d.values()); next(it)`
- Comprehensions: succinct computations over collections (map & filter)
  - `squares = [i**2 for i in range(10) if i % 3 != 1]`
- Exceptions: deal with errors when desired, allow aggregation
  - `try-except-else-finally`
- Object-Oriented Programming:
  - Class definitions (`__init__`, `self`)
  - Using object obj: `obj.field`, `obj.function()`

Databases & DBMSes

- Database:
  - Basically, just structured data/information stored on a computer
  - Very generic, doesn't specify specific way that data is stored
  - Can be single-file (or in-memory) or much more complex
- Database Management System (DBMS):
  - Software to manage databases
  - Instead of each program writing its own methods to manage data, abstract data management to the DBMS
  - Specify structure of the data (schema)
  - Provide query capabilities

Data Models

- The data model specifies:
  - what data can be stored (and sometimes how it is stored)
  - associations between different data values
  - what constraints can be enforced
  - how to access and manipulate the data
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semistructured data model (XML)
- Network Model

[A. Silberschatz et al.]

Relational Model & Relations

- Relations are basically tables of data
  - Each row represents a **tuple** in the relation
- A relational database is an **unordered** set of relations
  - Each relation has a unique name in the database
- Each row in the table specifies a relationship between the values in that row
  - The account ID "A-307", branch name "Seattle", and balance "275" are all related to each other

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
…	…	…

[D. Pinkston]

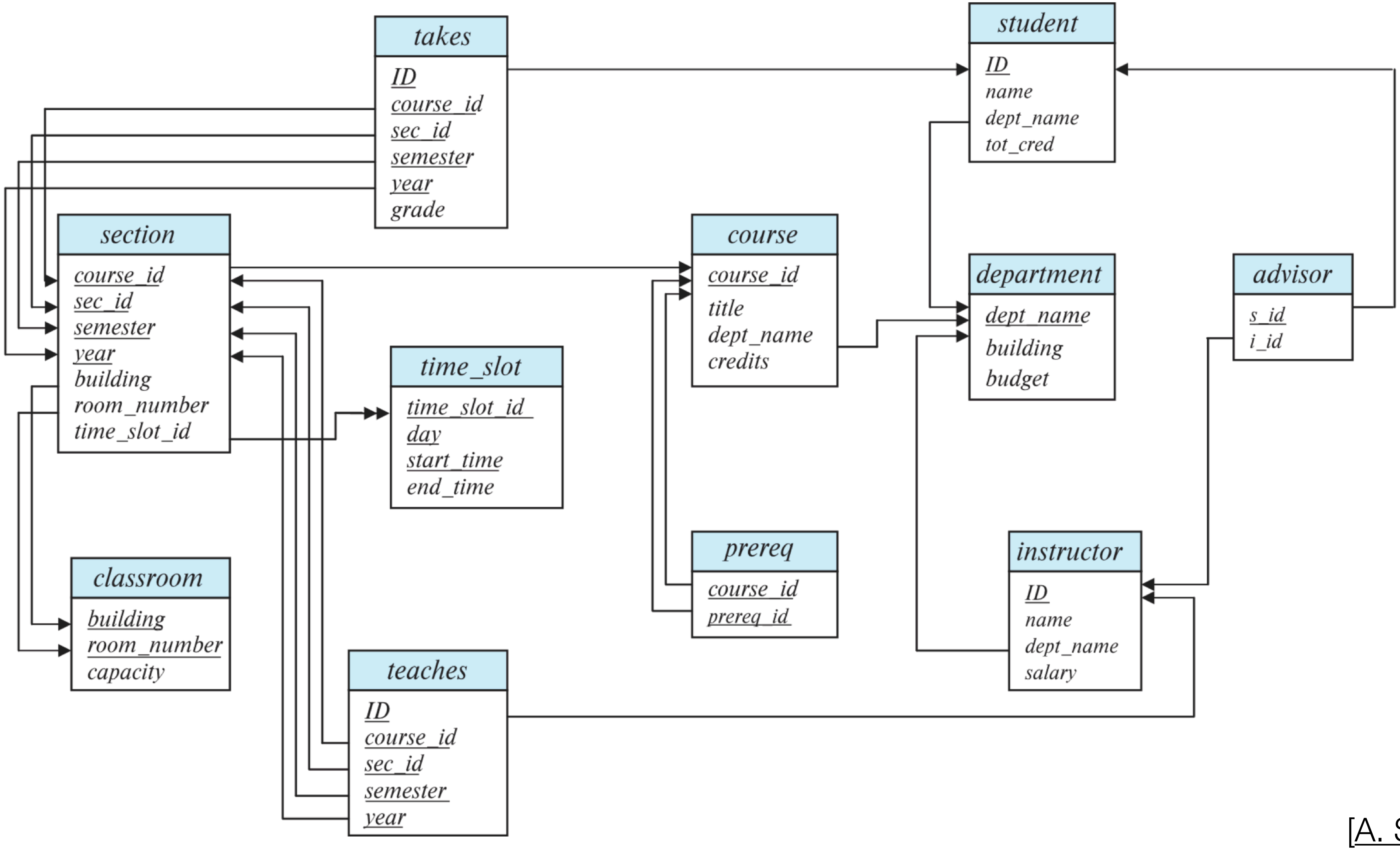
Database Schema

- Database schema: the logical structure of the database.
- Database instance: a snapshot of the data at a given instant in time.
- Example Schema
  - `instructor` (*ID*, *name*, *dept_name*, *salary*)

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

[A. Silberschatz et al.]

Schema Diagram with Keys



[A. Silberschatz et al.]

Relational Algebra

- Definition: A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.
- Six basic operators
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

[A. Silberschatz et al.]

Select Operation

- The select operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- p is called the selection predicate
- Example: select those tuples of the `instructor` relation where the instructor is in the "Physics" department.
  - Query: $\sigma_{dept\_name=\text{"Physics"}}$(`instructor`)
  - Result:

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

[A. Silberschatz et al.]

Project Operation

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

- Example: eliminate the `dept_name` attribute of instructor
- Query: $\prod_{\text{ID, name, salary}}$ (`instructor`)

[A. Silberschatz et al.]

Cartesian-Product Operation

- The **Cartesian-product** operation (denoted by X) allows us to combine information from any two relations.
- Example: the Cartesian product of the relations `instructor` and `teaches` is written as: `instructor` X `teaches`
- We construct a tuple of the result out of **each possible pair** of tuples: one from the instructor relation and one from the teaches relation
- Since the instructor ID appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came.
    - `instructor.ID` and `teaches.ID`

[A. Silberschatz et al.]

Join Operation

- The Cartesian-Product `instructor X teaches` associates every tuple of instructor with every tuple of teaches.
  - Most of the resulting rows have information about instructors who **did not** teach a particular course.
- To get only those tuples of `instructor X teaches` that pertain to instructors and the courses that they taught, we write:

$$\sigma_{\text{instructor.id = teaches.id}}(\texttt{instructor} \times \texttt{teaches})$$

  - We get only those tuples of `instructor X teaches` that pertain to instructors and the courses that they taught.

[A. Silberschatz et al.]

Equivalent Queries

- Example: Find information about courses taught by instructors in the Physics department
- Query 1:

$$\sigma_{dept\_name=\text{“Physics”}} (\texttt{instructor} \bowtie_{instructor.ID = teaches.ID} \texttt{teaches})$$

- Query 2

$$(\sigma_{dept\_name=\text{“Physics”}} (\texttt{instructor})) \bowtie_{instructor.ID = teaches.ID} \texttt{teaches}$$

- The **order** of joins is one focus of some of the work on query optimization

[A. Silberschatz et al.]

Assignment 1

- Due Monday, Feb. 7 at 11:59pm
- Using Python for data analysis on the Met's artwork
- Provided a1.ipynb file (right-click and download)
- Use basic python for now to demonstrate language knowledge
  - No pandas (for now)
- Use Anaconda or hosted Python environment
- Turn .ipynb file in via Blackboard
- Notes:
  - You will need to do some parsing of the data (converting to ints, splitting strings)

SQL

SQL History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO SQL: SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
- Not all examples work on all systems

[A. Silberschatz et al.]

Components of SQL

- **Data Definition Language (DDL)**: the specification of information about relations, including schema, types, integrity constraints, indices, storage
- **Data Manipulation Language (DML)**: provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- **Integrity**: the DDL includes commands for specifying integrity constraints.
- **View definition**: The DDL includes commands for defining views.
- Also: **Transaction control**, **embedded and dynamic SQL**, **authorization**

[A. Silberschatz et al.]

Create Table

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

  `create table r` $(A_1\ D_1,\ A_2\ D_2,\ \ldots,\ A_n\ D_n,\ (C_1),\ \ldots,\ (C_k))$

  - `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

C_i are integrity constraints

- Example:

```
create table instructor(
        ID              char(5),
        name            varchar(20),
        dept_name       varchar(20),
        salary          numeric(8,2));
```

[A. Silberschatz et al.]

Create Table

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

  `create table r` $(A_1\ D_1,\ A_2\ D_2,\ \ldots,\ A_n\ D_n,\ (C_1),\ \ldots,\ (C_k))$

  - `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

C_i are integrity constraints

- Example:

```
create table instructor(
        ID              char(5),
        name            varchar(20),
        dept_name       varchar(20),
        salary          numeric(8,2));
```

[A. Silberschatz et al.]

Create Table

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

  `create table r` $(A_1\ D_1,\ A_2\ D_2,\ \ldots,\ A_n\ D_n,\ (C_1),\ \ldots,\ (C_k))$

  - `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

C_i are integrity constraints

- Example:

```
create table instructor(
        ID              char(5),
        name            varchar(20),
        dept_name       varchar(20),
        salary          numeric(8,2));
```

[A. Silberschatz et al.]

Create Table

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

  `create table r` $(A_1\ D_1,\ A_2\ D_2,\ \ldots,\ A_n\ D_n,\ (C_1),\ \ldots,\ (C_k))$

  - `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

C_i are integrity constraints

- Example:

```
create table instructor(
    ID            char(5),
    name          varchar(20),
    dept_name     varchar(20),
    salary        numeric(8,2));
```

[A. Silberschatz et al.]

Integrity Constraints in Create Table

- Types of integrity constraints
  - **primary key** $(A_1, \ldots, A_n)$
  - **foreign key** $(A_m, \ldots, A_n)$ **references** r
  - **not null**
- SQL prevents any update to the database that violates an integrity constraint
- **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);
```

[A. Silberschatz et al.]

Updates to tables

- Insert: **insert into** `instructor` **values** `('10211', 'Smith', 'Biology', 66000);`
- Delete: **delete from** `student; -- remove all tuples from student`
- Drop Table: **drop table** `r`
- Alter: **alter table** `r` **add** *A D;* **alter table** `r` **drop** `A`
  - *A* is the name of the attribute to be added to relation `r`
  - *D* is the domain of *A*
  - All exiting tuples are assigned `null` for the new attribute's value
  - Dropping of attributes not widely supported

[A. Silberschatz et al.]

Basic Query Structure

- 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**

[A. Silberschatz et al.]

Select

- The **select** clause lists 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**
  - `Name` and `NAME` and `name` are equivalent
  - Some people use upper case for language keywords (e.g. `SELECT`)

[A. Silberschatz et al.]
    ```

Select

- SQL allows **duplicates** in relations as well as in query results.
- To eliminate duplicates, put the keyword **`distinct`** after **`select`**.
- Example: Find the department names of all instructors (no duplicates)
  - ```
    select distinct dept_name
       from instructor;
    ```
- The keyword **`all`** specifies that duplicates should not be removed
  - ```
    select all dept_name
       from instructor;
    ```

dept_name
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

[A. Silberschatz et al.]

Select

- An asterisk (*) in the select clause denotes "all attributes"
  - `select * from instructor;`
- An attribute can be a **literal** with no from clause (`select '437'`)
  - Result is a table with one column and a single row with value `'437'`
  - Can give the column a name using as: `select '437' as FOO`
- An attribute can be a literal with from clause:
  - `select 'A' from instructor`
  - Result is a table with one column and *N* rows (number of tuples in the instructors table), each row with value "A"

[A. Silberschatz et al.]

Select "Math"

- 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`.
- Can rename expressions using the **`as`** clause:
  - `select ID, name, salary/12 as monthly_salary`

[A. Silberschatz et al.]

Where

- The **where** clause specifies conditions that the result must satisfy
  - Confusingly corresponds to the **selection** predicate in relational algebra
- Example: Find all instructors in Comp. Sci. dept
  - ```
    select name
    from instructor
    where dept_name = 'Comp. Sci.'
    ```

[A. Silberschatz et al.]
    ```

Where

- The operands can be expressions with operators `<`, `<=`, `>`, `>=`, `=`, and **`<>`**
- SQL allows the use of the logical connectives `and`, `or`, and `not`
- Comparisons can be applied to results of arithmetic expressions
- Example: Find all instructors in Comp. Sci. with salary > 70000
  - ```
    select name
    from instructor
    where dept_name = 'Comp. Sci.' and salary > 70000
    ```

name
Katz
Brandt

[A. Silberschatz et al.]
    ```

From

- The **from** clause lists the relations involved in the query
  - Corresponds to the **Cartesian Product** operation in relational algebra
- Find the Cartesian product `instructor` X `teaches`
  - ```
    select *
    from instructor, teaches;
    ```
  - All possible `instructor` – `teaches` pair, with all attributes from both
  - Shared attributes (e.g., `ID`) are renamed (e.g., `instructor.ID`)
- Not very useful directly but useful combined with where clauses.

[A. Silberschatz et al.]
    ```

From

- Find the names of all instructors who have taught some course and that course_id
  - ```
    select name, course_id
    from instructor, teaches
    where instructor.ID = teaches.ID
    ```
- 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'
    ```

name	*course_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

[A. Silberschatz et al.]

The Rename Operation

- SQL allows renaming relations and attributes using the **as** clause:
  - *old-name* **as** *new-name*
- Example: Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.
  - ```
    select distinct T.name
    from instructor as T, instructor as S
    where T.salary > S.salary and S.dept_name = 'Comp. Sci.'
    ```
- Keyword as is optional and may be omitted
  - `instructor as T` is equivalent to `instructor T`

[A. Silberschatz et al.]
    ```

Set Operations

- Find courses that ran in Fall 2017 or in Spring 2018
- ```
  (select course_id from section where sem = 'Fall' and year = 2017)
    union
  (select course_id from section where sem = 'Spring' and year = 2018)
  ```
- Find courses that ran in Fall 2017 and in Spring 2018
- ```
  (select course_id from section where sem = 'Fall' and year = 2017)
    intersect
  (select course_id from section where sem = 'Spring' and year = 2018)
  ```
- Find courses that ran in Fall 2017 but not in Spring 2018
- ```
  (select course_id from section where sem = 'Fall' and year = 2017)
    except
  (select course_id from section where sem = 'Spring' and year = 2018)
  ```

[A. Silberschatz et al.]
  ```

Aggregate Functions

- 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 2018 semester
  - ```
    select count(distinct ID)
    from teaches
    where semester = 'Spring' and year = 2018;
    ```
- Find the number of tuples in the course relation
  - ```
    select count(*)
    from course;
    ```

[A. Silberschatz et al.]
    ```

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

[A. Silberschatz et al.]
    ```

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

[A. Silberschatz et al.]
    ```

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

[A. Silberschatz et al.]
    ```

Having Clause

- Filter groups based on predicates
- Predicates in the having clause are applied **after** the formation of groups whereas predicates in the where clause are applied **before** forming groups
- Example: Find the names and average salaries of all departments whose average salary is greater than 42,000
  - ```
    select dept_name, avg(salary) as avg_salary
    from instructor
    group by dept_name
    having avg(salary) > 42000;
    ```

[A. Silberschatz et al.]
    ```

Modification of the Database

- Deleting tuples from a given relation.
- Inserting new tuples into a given relation
- Updating values in some tuples in a given relation

[A. Silberschatz et al.]

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');
    ```

[A. Silberschatz et al.]

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');
    ```

[A. Silberschatz et al.]

Insertion

- Add a new tuple to course
  - ```
    insert into course
        values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
    ```
- or…
  - ```
    insert into course(course_id, title, dept_name, credits)
        values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
    ```
- Add a new tuple to `student` with `tot_creds` set to `null`
  - ```
    insert into student
        values ('3003', 'Green', 'Finance', null);
    ```

[A. Silberschatz et al.]
    ```

Insertion

- Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18,000.
  - ```
    insert into instructor
       select ID, name, dept_name, 18000
       from student
       where dept_name = 'Music' and total_cred > 144;
    ```
- The select-from-where statement is evaluated fully before any of its results are inserted into the relation.
- If not queries like

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

would cause problems

[A. Silberschatz et al.]
```

Updates

- Give a 5% salary raise to all instructors
  - ```
    update instructor
    set salary = salary * 1.05
    ```
- Give a 5% salary raise to those instructors who earn less than 70000
  - ```
    update instructor
    set salary = salary * 1.05
    where salary < 70000;
    ```
- Give a 5% salary raise to instructors whose salary is less than average
  - ```
    update instructor
    set salary = salary * 1.05
    where salary < (select avg(salary) from instructor);
    ```

[A. Silberschatz et al.]
    ```

Updates

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Use two update statements:
  - ```
    update instructor
    set salary = salary * 1.03
    where salary > 100000;
    ```
  - ```
    update instructor
    set salary = salary * 1.05
    where salary <= 100000;
    ```
  - Order matters!

[A. Silberschatz et al.]

Joins

- Join operations take two relations and return another relation.
- From relational algebra, this is a Cartesian product + selection
- Want tuples in the two relations to match (under some condition)
- The join operations typically used as subquery expressions in the from clause
- Three types of joins:
  - Natural join
  - Inner join
  - Outer join

[A. Silberschatz et al.]

Natural Join

- Natural join matches tuples with the same values for all **common** attributes, and retains only **one copy** of each common column.
- List the names of instructors along with the course ID of the courses that they taught
  - ```
    select name, course_id
    from students, takes
    where student.ID = takes.ID;
    ```
- Same query in SQL with "natural join" construct
  - ```
    select name, course_id
    from student natural join takes;
    ```

[A. Silberschatz et al.]

Example: Student Schedules

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

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

[A. Silberschatz et al.]

Example: Natural Join

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

[A. Silberschatz et al.]

Natural Join Danger

- Beware of unrelated attributes with same name which get equated incorrectly
- Example: List the names of students instructors along with the titles of courses that they have taken
  - ```
    select name, title
    from student natural join takes natural join course;
    ```
- Wrong… only lists courses when the student took courses in their department (major)
- Correct:
  - ```
    select name, title
    from student natural join takes, course
    where takes.course_id = course.course_id;
    ```

[A. Silberschatz et al.]

Outer Join

- Joins so far are inner joins
- Outer joins returns tuples from one (or both) relations that do not match tuples in the other relation
- Fills in missing values with null
- Three forms of outer join:
  - **left** outer join
  - **right** outer join
  - full **outer** join

[A. Silberschatz et al.]

Join Examples

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

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

prereq

Left Join

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 Join

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

[A. Silberschatz et al.]

Join Examples

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`

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

`prereq`

(Full) Outer Join

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

Inner Join

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

[A. Silberschatz et al.]