CMSC424: Database Design
Relational Model; SQL

Instructor: Amol Deshpande
amol@cs.umd.edu

Outline

- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

Context

- Data Models
  - Conceptual representation of the data
- Data Retrieval
  - How to ask questions of the database
  - How to answer those questions
- Data Storage
  - How/where to store data, how to access it
- Data Integrity
  - Manage crashes, concurrency
  - Manage semantic inconsistencies

Relational Data Model

Introduced by Ted Codd (late 60's – early 70's)

- *Before = "Network Data Model" (Cobol as DDL, DML)*
- *Very contentious: Database Wars (Charlie Bachman vs. Ted Codd)*

Relational data model contributes:

1. *Separation of logical, physical data models (data independence)*
2. *Declarative query languages*
3. *Formal semantics*
4. *Query optimization (key to commercial success)*

1st prototypes:

- *Ingres → CA*
- *Postgres → Illustra → Informix → IBM*
- *System R → Oracle, DB2*

Key Abstraction: Relation

Account =

bname	acct_no	balance
Downtown	A-101	500
Brighton	A-201	900
Brighton	A-217	500

Terms:

- Tables (aka: Relations)

Why called Relations?

*Closely correspond to mathematical concept of a **relation***

Relations

Account =

bname	acct_no	balance
Downtown	A-101	500
Brighton	A-201	900
Brighton	A-217	500

Considered equivalent to…

{ (Downtown, A-101, 500),
(Brighton, A-201, 900),
(Brighton, A-217, 500) }

Relational database semantics defined in terms of mathematical relations

Relations

Account =

bname	acct_no	balance
Downtown	A-101	500
Brighton	A-201	900
Brighton	A-217	500

Considered equivalent to…

{ (Downtown, A-101, 500),
(Brighton, A-201, 900),
(Brighton, A-217, 500) }

Terms:

- Tables (aka: Relations)
- Rows (aka: tuples)
- Columns (aka: attributes)
- Schema (e.g.: Acct_Schema = (bname, acct_no, balance))

Definitions

Relation Schema (or Schema)

A list of attributes and their domains

E.g. **account**(account-number, branch-name, balance)

Programming language equivalent: A variable (e.g. x)

Relation Instance

A particular instantiation of a relation with actual values

Will change with time

bname	acct_no	balance
Downtown	A-101	500
Brighton	A-201	900
Brighton	A-217	500

Programming language equivalent: Value of a variable

Definitions

Domains of an attribute/column

The set of permitted values

e.g., bname must be String, balance must be a positive real number

We typically assume domains are **atomic,** i.e., the values are treated as indivisible (specifically: you can't store lists or arrays in them)

Null value

A special value used if the value of an attribute for a row is:

- unknown (e.g., don't know address of a customer)
- inapplicable (e.g., "spouse name" attribute for a customer)
- withheld/hidden

Different interpretations all captured by a single concept – leads to major headaches and problems

Tables in a University Database

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building,
room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(Id, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

Keys

- Let K ⊆ R
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of any possible relation r(R)
  - *Example: {ID} and {ID,name} are both superkeys of instructor.*
- Superkey K is **a candidate key** if K is minimal (i.e., no subset of it is a superkey)
  - *Example: {ID} is a candidate key for Instructor*
- One of the candidate keys is selected to be the **primary key**
  - Typically one that is small and immutable (doesn't change often)
- Primary key typically highlighted (e.g., underlined)

Tables in a University Database

classroom(building, room_number, capacity)

department(dept_name, building, budget)

course(course_id, title, dept_name, credits)

instructor(ID, name, dept_name, salary)

Tables in a University Database

takes(ID, course_id, sec_id, semester, year, grade)

What about ID, course_id?

No. May repeat:

(“1011049”, “CMSC424”, “101”, “Spring”, 2014, D)

(“1011049”, “CMSC424”, “102”, “Fall”, 2015, null)

What about ID, course_id, sec_id?

May repeat:

(“1011049”, “CMSC424”, “101”, “Spring”, 2014, D)

(“1011049”, “CMSC424”, “101”, “Fall”, 2015, null)

What about ID, course_id, sec_id, semester?

Still no: (“1011049”, “CMSC424”, “101”, “Spring”, 2014, D)

(“1011049”, “CMSC424”, “101”, “Spring”, 2015, null)

Tables in a University Database

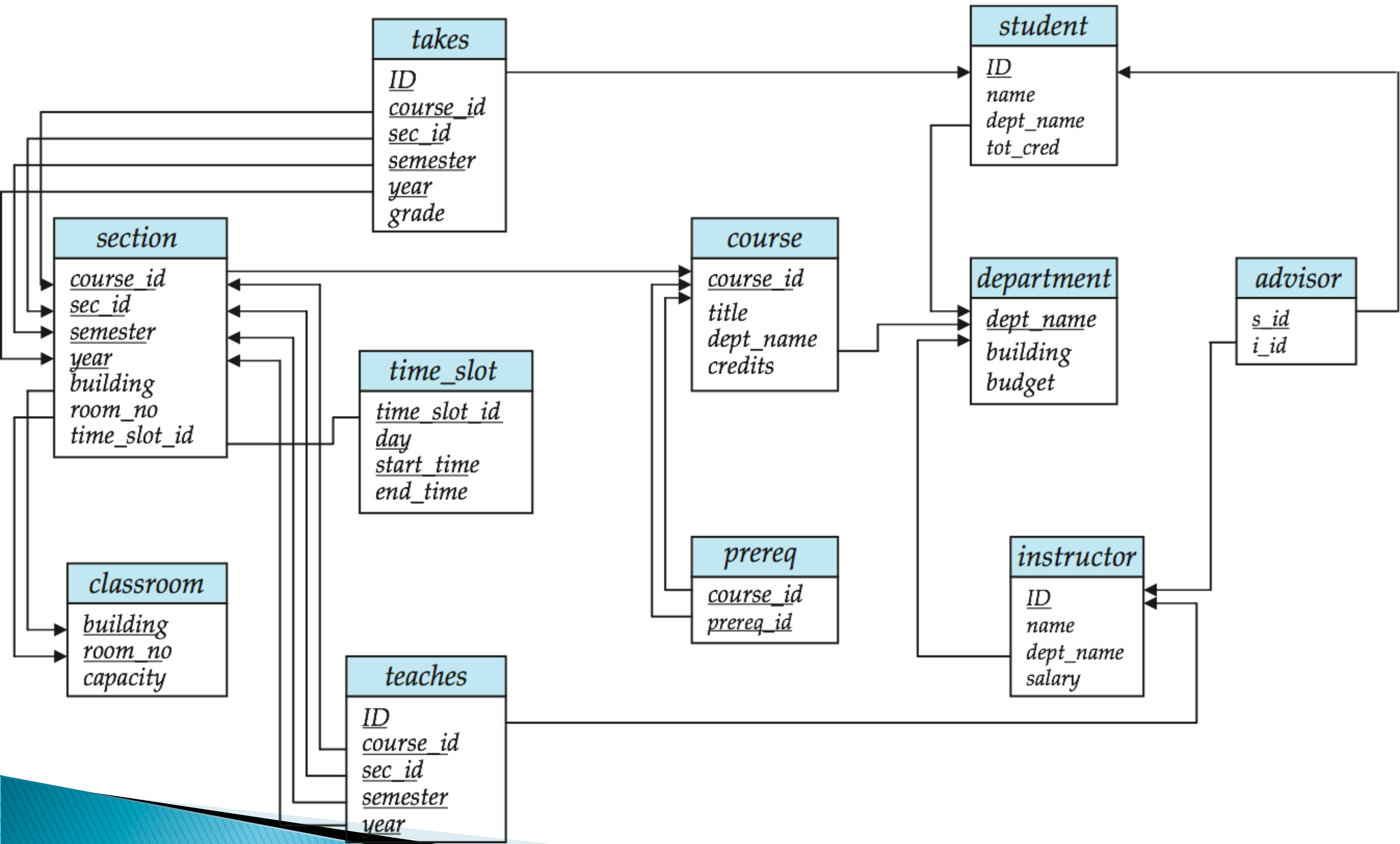
classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building,
room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

Keys

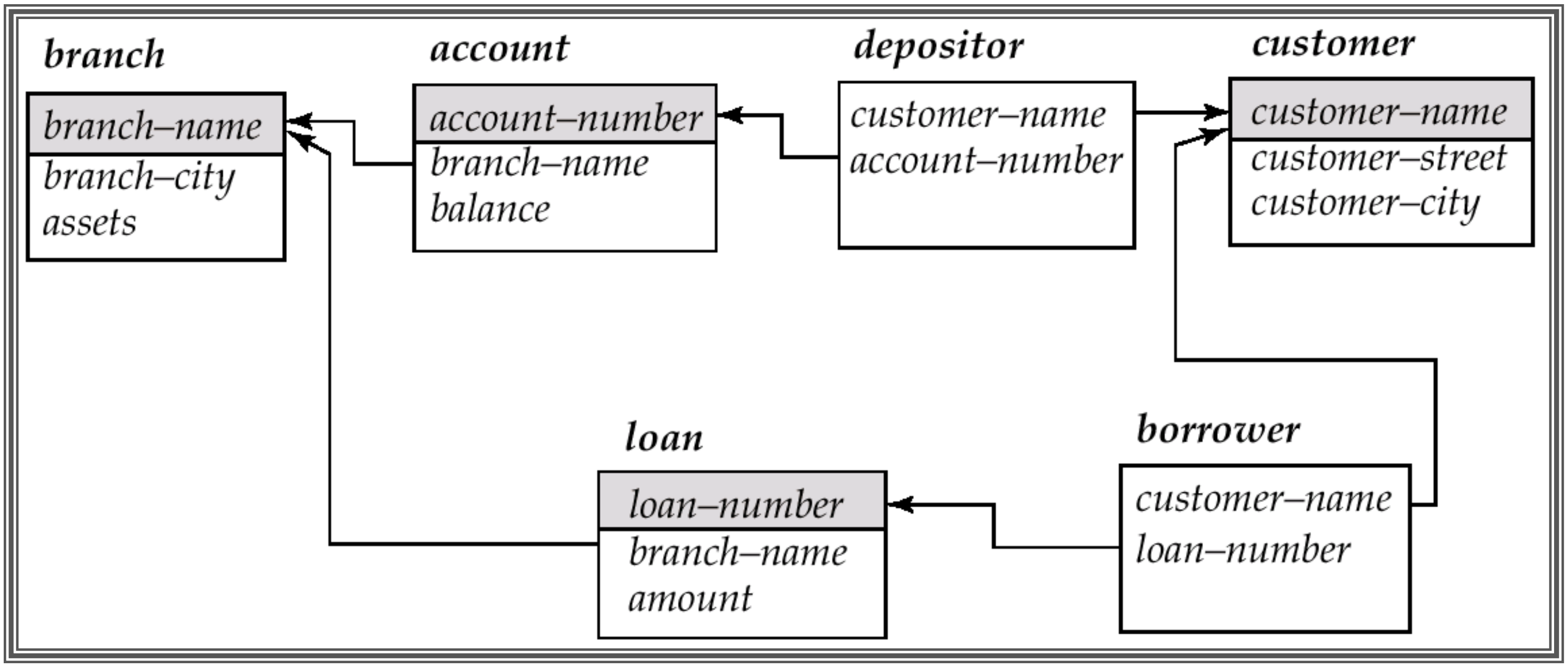
- **Foreign key:** Primary key of a relation that appears in another relation
  - {ID} from *student* appears in *takes, advisor*
  - *student* called ***referenced*** relation
  - *takes* is the ***referencing*** relation
  - Typically shown by an arrow from referencing to referenced

- **Foreign key constraint**: the tuple corresponding to that primary key must exist
  - Imagine:
    - Tuple: ('student101', 'CMSC424') in *takes*
    - But no tuple corresponding to 'student101' in *student*
  - Also called ***referential integrity constraint***

Schema Diagram for University Database



Schema Diagram for the Banking Enterprise



Examples

- Married(person1_ssn, person2_ssn, date_married, date_divorced)
- Account(cust_ssn, account_number, cust_name, balance, cust_address)
- RA(student_id, project_id, superviser_id, appt_time, appt_start_date, appt_end_date)
- Person(Name, DOB, Born, Education, Religion, …)
  - *Information typically found on Wikipedia Pages*

Examples

- Married(person1_ssn, person2_ssn, date_married, date_divorced)
- Account(cust_ssn, account_number, cust_name, balance, cust_address)
  - If a single account per customer, then: cust_ssn
  - Else: (cust_ssn, account_number)
    - In the latter case, this is not a good schema because it requires repeating information
- RA(student_id, project_id, superviser_id, appt_time, appt_start_date, appt_end_date)
  - Could be smaller if there are some restrictions – requires some domain knowledge of the data being stored
- Person(Name, DOB, Born, Education, Religion, …)
  - *Information typically found on Wikipedia Pages*
  - *Unclear what could be a primary key here: you could in theory have two people who match on all of those*

Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

Relational Query Languages

- Example schema: *R(A, B)*
- Practical languages
  - SQL
    - select A from R where B = 5;
  - Datalog (sort of practical)
    - q(A) :- R(A, 5)
- Formal languages
  - Relational algebra

    $\pi_A (\sigma_{B=5} (R))$
  - Tuple relational calculus

    $\{ t : \{A\} \mid \exists\ s : \{A, B\} (R(A, B) \wedge s.B = 5) \}$
  - Domain relational calculus
    - Similar to tuple relational calculus

Relational Operations

- Some of the languages are “procedural” and provide a set of operations
  - Each operation takes one or two relations as input, and produces a single relation as output
  - Examples: SQL, and Relational Algebra

- The “non-procedural” (also called “declarative”) languages specify the output, but don’t specify the operations
  - Relational calculus
  - Datalog (used as an intermediate layer in quite a few systems today)

Select Operation

Choose a subset of the tuples that satisfies some predicate
Denoted by σ in relational algebra

Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{A=B \land D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Project

Choose a subset of the columns (for all rows)
Denoted by $\prod$ in relational algebra

Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\prod_{A,D}(r)$

A	D
α	7
α	7
β	3
β	10

A	D
α	7
β	3
β	10

Relational algebra following "set" semantics – so no duplicates
SQL allows for duplicates – we will cover the formal semantics later

Set Union, Difference

Relation r, s

r:

A	B
α	1
α	2
β	1

s:

A	B
α	2
β	3

$r \cup s$:

A	B
α	1
α	2
β	1
β	3

$r - s$:

A	B
α	1
β	1

Must be compatible schemas

What about intersection ?

Can be derived

$r \cap s = r - (r - s);$

Cartesian Product

Combine tuples from two relations

If one relation contains N tuples and the other contains M tuples, the result would contain N*M tuples

The result is rarely useful – almost always you want pairs of tuples that satisfy some condition

Relation r, s

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

$r \times s$:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Joins

Combine tuples from two relations if the pair of tuples satisfies some constraint

Equivalent to Cartesian Product followed by a Select

Relation r, s

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

$r \bowtie_{A=C} s$:

A	B	C	D	E
α	1	α	10	a
~~α~~	~~1~~	~~β~~	~~10~~	~~a~~
~~α~~	~~1~~	~~β~~	~~20~~	~~b~~
~~α~~	~~1~~	~~γ~~	~~10~~	~~b~~
~~β~~	~~2~~	~~α~~	~~10~~	~~a~~
β	2	β	10	a
β	2	β	20	b
~~β~~	~~2~~	~~γ~~	~~10~~	~~b~~

Natural Join

Combine tuples from two relations if the pair of tuples agree on the common columns (with the same name)

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Figure 2.5 The *department* relation.

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

Figure 2.4 Unsorted display of the *instructor* relation.

department $\bowtie$ instructor:

ID	*name*	*salary*	*dept_name*	*building*	*budget*
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
12121	Wu	90000	Finance	Painter	120000
15151	Mozart	40000	Music	Packard	80000
22222	Einstein	95000	Physics	Watson	70000
32343	El Said	60000	History	Painter	50000
33456	Gold	87000	Physics	Watson	70000
45565	Katz	75000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
76543	Singh	80000	Finance	Painter	120000
76766	Crick	72000	Biology	Watson	90000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000

Figure 2.12 Result of natural join of the *instructor* and *department* relations.

Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

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 standard SQL:
  - SQL-86, SQL-89, SQL-92
  - SQL:1999, SQL:2003, SQL:2008
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system.
- Several alternative syntaxes to write the same queries

Different Types of Constructs

- **Data definition language** (DDL): Defining/modifying schemas
  - **Integrity constraints:** Specifying conditions the data must satisfy
  - **View definition:** Defining views over data
  - **Authorization:** Who can access what
- **Data-manipulation language** (DML): Insert/delete/update tuples, queries
- **Transaction control:**
- **Embedded SQL:** Calling SQL from within programming languages
- **Creating indexes, Query Optimization control…**

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
- Also: other information such as
  - 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.

SQL Constructs: Data Definition Language

- CREATE TABLE <name> (<field> <domain>, …)

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

```
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
)
```

SQL Constructs: Data Definition Language

- CREATE TABLE <name> (<field> <domain>, …)

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

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

SQL Constructs: Data Definition Language

- drop table student
- delete from student
  - Keeps the empty table around
- alter table
  - alter table student add address varchar(50);
  - alter table student drop tot_cred;

SQL Constructs: Insert/Delete/Update Tuples

- INSERT INTO <name> (<field names>) VALUES (<field values>)

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

  insert into *instructor (name, ID)* **values** ('Smith', '10211');

  -- NULL for other two

  insert into *instructor (ID) values ('10211');*

  -- FAIL

- DELETE FROM <name> WHERE <condition>

  delete from department **where** budget < 80000;

  - Syntax is fine, but this command **may be rejected** because of referential integrity constraints.

SQL Constructs: Insert/Delete/Update Tuples

- DELETE FROM <name> WHERE <condition>

  delete from department **where** budget < 80000;

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Figure 2.5 The *department* relation.

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

Instructor relation

We can choose what happens:
(1) Reject the delete, or
(2) Delete the rows in Instructor (may be a cascade), or
(3) Set the appropriate values in Instructor to NULL

SQL Constructs: Insert/Delete/Update Tuples

- DELETE FROM <name> WHERE <condition>

delete from department **where** budget < 80000;

```
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
        on delete set null
    );
```

We can choose what happens:
(1) Reject the delete (nothing), or
(2) Delete the rows in Instructor (on delete cascade), or
(3) Set the appropriate values in Instructor to NULL (on delete set null)

SQL Constructs: Insert/Delete/Update Tuples

- DELETE FROM <name> WHERE <condition>
  - Delete all classrooms with capacity below average

    delete from classroom **where** capacity <

    (**select avg(**capacity) **from** classroom);

  - Problem: as we delete tuples, the average capacity changes
  - Solution used in SQL:
    - First, compute **avg** capacity and find all tuples to delete
    - Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)
  - E.g. consider the query: delete the smallest classroom

SQL Constructs: Insert/Delete/Update Tuples

- UPDATE <name> SET <field name> = <value> WHERE <condition>
  - Increase all salaries's over $100,000 by 6%, all other receive 5%.
  - Write two update statements:

```
update instructor
 set salary = salary * 1.06
 where salary > 100000;


update instructor
 set salary = salary * 1.05
 where salary ≤ 10000;
```

  - The order is important
  - Can be done better using the case statement

SQL Constructs: Insert/Delete/Update Tuples

- UPDATE <name> SET <field name> = <value> WHERE <condition>
  - Increase all salaries's over $100,000 by 6%, all other receive 5%.
  - Can be done better using the <u>case</u> statement

```
update instructor
set salary =
        case
            when salary > 100000
                then salary * 1.06
            when salary <= 100000
                then salary * 1.05
        end;
```

Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

Setting up the PostgreSQL database

- Follow the instructions posted on the course website to set up the University database in PostgreSQL

https://github.com/umddb/cmsc424-fall2015/tree/master/postgresql-setup

Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

Basic Query Structure

select $A_1, A_2, \ldots, A_n$ ← Attributes or expressions

from $r_1, r_2, \ldots, r_m$ ← Relations (or queries returning tables)

where P ← Predicates

Find the names of all instructors:
select *name*
from *instructor*

Remove duplicates:
select distinct *name*
from *instructor*

Order the output:
select distinct *name*
from *instructor*
order by *name* **asc**

Apply some filters (predicates):
select *name*
from *instructor*
where salary > 80000 **and** dept_name = 'Finance';

Basic Query Constructs

Find the names of all instructors:
select *name*
from *instructor*

Select all attributes:
select *
from *instructor*

Expressions in the select clause:
select *name, salary < 100000*
from *instructor*

More complex filters:
select *name*
from *instructor*
where (dept_name != ‘Finance’ **and** salary > 75000)
or (dept_name = ‘Finance’ **and** salary > 85000);

A filter with a subquery:
select *name*
from *instructor*
where dept_name in (**select** dept_name **from**
department **where** budget < 100000);

Basic Query Constructs

Renaming tables or output column names:
select *i.name, i.salary * 2* **as** *double_salary*
from *instructor i*
where *i.salary < 80000* **and** *i.name like '%g_';*

Find the names of all instructors:
select *name*
from *instructor*

More complex expressions:
select *concat(name, concat(', ', dept_name))*
from *instructor;*

Careful with NULLs:
select *name*
from *instructor*
where *salary < 100000* **or** *salary >= 100000;*

Wouldn't return the instructor with NULL salary (if any)

Multi-table Queries

Cartesian product:

select *

from *instructor, department*

Use predicates to only select "matching" pairs:

select *

from *instructor i, department d*

where *i.dept_name = d.dept_name;*

Identical (in this case) to using a natural join:

select *

from *instructor* **natural join** *department;*

Natural join does an equality on common attributes – doesn't work here:

select *

from *instructor* **natural join** *advisor;*

Instead can use "on" construct (or where clause as above):

select *

from *instructor* **join** *advisor* **on** *(i_id = id);*

Multi-table Queries

3-Table Query to get a list of instructor-teaches-course information:

select *i.name* **as** *instructor_name, c.title* **as** *course_name*
from *instructor i, course c, teaches*
where *i.ID = teaches.ID and c.course_id = teaches.course_id;*

Beware of unintended common names (happens often)
You may think the following query has the same result as above – it doesn't

select *name*, *title*
from *instructor* **natural join** *course* **natural join** *teaches;*

I prefer avoiding "natural joins" for that reason

Note: On the small dataset, the above two have the same answer, but not on the large dataset. Large dataset has cases where an instructor teaches a course from a different department.

Set operations

Find courses that ran in Fall 2009 or 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);

In both:

(**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2009)
intersect
(**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2010);

In Fall 2009, but not in Spring 2010:

(**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2009)
except
(**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2010);

Set operations: Duplicates

Union/Intersection/Except eliminate duplicates in the answer (the other SQL commands don't) (e.g., try 'select dept_name from instructor').

Can use "union all" to retain duplicates.

NOTE: The duplicates are retained in a systematic fashion (for all SQL operations)

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

- $m + n$ times in *r* **union all** *s*
- min(m,n) times in *r* **intersect all** *s*
- max(0, $m - n$) times in *r* **except all** *s*

NULLs

Union/Intersection/Except eliminate duplicates in the answer (the other SQL commands don't) (e.g., try 'select dept_name from instructor').

Can use "union all" to retain duplicates.

NOTE: The duplicates are retained in a systematic fashion (for all SQL operations)

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

- $m + n$ times in *r* **union all** *s*
- min(*m,n)* times in *r* **intersect all** *s*
- max(0, *m – n)* times in *r* **except all** *s*

Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

SQL: Nulls

The "dirty little secret" of SQL

(major headache for query optimization)

Can be a value of any attribute

e.g: branch =

bname	**bcity**	**assets**
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

What does this mean?

(unknown) We don't know Waltham's assets?

(inapplicable) Waltham has a special kind of account without assets

(withheld) We are not allowed to know

SQL: Nulls

Arithmetic Operations with `Null`

`n + NULL = NULL` (similarly for all *arithmetic ops*: `+, -, *, /, mod,` …)

e.g: branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT bname, assets * 2 as a2
FROM branch
```

=

bname	a2
Downtown	18M
Perry	3.4M
Mianus	.8M
Waltham	NULL

SQL: Nulls

Boolean Operations with `Null`

`n < NULL = UNKNOWN` (similarly for all *boolean ops*: `>, <=, >=, <>, =, ...`)

e.g: branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT *
FROM branch
WHERE assets = NULL
```

=

bname	bcity	assets

Counter-intuitive: NULL * 0 = NULL

Counter-intuitive: select * from movies
where length >= 120 or length <= 120

SQL: Nulls

Boolean Operations with `Null`

`n < NULL = UNKNOWN` (similarly for all *boolean ops*: `>, <=, >=, <>, =, …`)

e.g: branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT *
FROM branch
WHERE assets IS NULL
```

=

bname	bcity	assets
Waltham	Boston	NULL

SQL: Unknown

Boolean Operations with `Unknown`

`n < NULL = UNKNOWN` (similarly for all *boolean ops*: `>, <=, >=, <>, =, …`)

`FALSE OR UNKNOWN = UNKNOWN`

`TRUE AND UNKNOWN = UNKNOWN`

Intuition: substitute each of TRUE, FALSE for unknown. If different answer results, results is unknown

`UNKNOWN OR UNKNOWN = UNKNOWN`

`UNKNOWN AND UNKNOWN = UNKNOWN`

`NOT (UNKNOWN) = UNKNOWN`

Can write:

```
SELECT …
FROM …
WHERE booleanexp IS UNKNOWN
```

UNKNOWN tuples are not included in final result

Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

Aggregates

Other common aggregates:
max, min, sum, count, stdev, …

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

Find the average salary of instructors in the Computer Science
select ***avg(****salary)*
from *instructor*
where *dept_name = 'Comp. Sci';*

Can specify aggregates in any query.

Find max salary over instructors teaching in S'10
select max(*salary)*
from *teaches* **natural join** *instructor*
where *semester* = ' Spring' **and** *year* = 2010;

Aggregate result can be used as a scalar.
Find instructors with max salary:
select *
from *instructor*
where *salary* **= (select max***(salary)* **from** *instructor);*

Aggregates

Aggregate result can be used as a scalar.
Find instructors with max salary:

select *
from *instructor*
where *salary* **= (select max***(salary)* **from** *instructor);*

Following doesn't work:

select *
from *instructor*
where *salary* **= max***(salary);*

select *name,* **max***(salary)*
from *instructor*
where *salary* **= max***(salary);*

Aggregates: Group By

Split the tuples into groups, and computer the aggregate for each group

select *dept_name*, **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

Aggregates: Group By

Attributes in the select clause must be aggregates, or must appear in the group by clause. Following wouldn't work

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

"having" can be used to select only some of the groups.

select *dept_name*, ID, **avg** (*salary*)
from *instructor*
group by *dept_name*
having avg(*salary***)** *> 42000;*

Aggregates and NULLs

Given

branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

Aggregate Operations

```
SELECT SUM (assets) =
FROM branch
```

SUM
11.1 M

`NULL` *is ignored for SUM*

Same for `AVG` *(3.7M),* `MIN` *(0.4M),* `MAX` *(9M)*

Also for COUNT(assets) -- returns 3

But `COUNT (*)` *returns*

COUNT
4

Aggregates and NULLs

Given

branch =

<u>bname</u>	<u>bcity</u>	<u>assets</u>

`SELECT SUM (assets) =`
`FROM branch`

<u>SUM</u>
NULL

- *Same as* `AVG, MIN, MAX`
- *But* `COUNT (assets)` *returns*

<u>COUNT</u>
0

Summary

- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)