# Outline

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



# Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list:

$$\Pi_{F1,\ F2,\ \ldots,\ Fn}\ (E)$$

- *E* is any relational-algebra expression
- Each of $F_1$, $F_2$, …, $F_n$ are are arithmetic expressions involving constants and attributes in the schema of *E*.
- Given relation ***instructor(ID, name, dept_name,*** salary) where salary is annual salary, get the ***name*** and ***monthly salary:***

$$\Pi_{\text{name, salary/12}}\ (\text{instructor})$$

# Aggregate Functions and Operations

- **Aggregation functions** take a collections of values and return a single values

  **avg**: average value
  **min**: minimum value
  **max**: maximum value
  **sum**: sum of values
  **count**: number of values

- **Aggregate operation** in relational algebra:

$$_{G_1, G_2, \ldots, G_n} \mathcal{G}_{F_1(A_1), F_2(A_2, \ldots, F_n(A_n)}(E)$$

  *E* is any relational-algebra expression

  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each $F_i$ is an aggregate function over a group
  - Each $A_i$ is an attribute name

- Note: Some books/articles use $\gamma$ instead of $\mathcal{G}$ (Calligraphic G)



# Aggregate Example

- Relation *r*:

| *A* | *B* | *C* |
|---|---|---|
| α | α | 7 |
| α | β | 7 |
| β | β | 3 |
| β | β | 10 |

$\mathcal{G}_{\textbf{sum(c)}}$ (r)

| **sum**(*c*) |
|---|
| 27 |

# Aggregate Operation – Example

- Find the average salary in each department

$_{dept\_name}\mathcal{G}_{\textbf{avg}(salary)}(instructor)$

| *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 |



# Aggregate Functions (Cont.)

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

$_{dept\_name}\mathcal{G}_{\textbf{avg}(salary)\ \textbf{as}\ avg\_sal}(instructor)$

# Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations can be expressed using the assignment operator

$$
\begin{aligned}
temp1 &\leftarrow R \times S \\
temp2 &\leftarrow \sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} (temp1) \\
result &= \Pi_{R \cup S} (temp2)
\end{aligned}
$$

The result of R x S potentially has duplicated attributes. For example:
r(A,B) X s(B,C)
results in tuples w/ attributes {A, B, B, C}.
"$\prod_{R \cup S}$" can be use to get rid of the extra B.

Duplicated *tuples* are different, not present in the relational algebra.



# Multiset Relational Algebra

- Pure relational algebra removes all duplicate tuples
  - e.g. after projection
- *Multiset relational algebra* retains duplicates, to match SQL semantics
  - SQL duplicate retention was initially for efficiency, but is now a feature
- Multiset relational algebra defined as follows:
  - selection: output has as many duplicates of tuple as input, if the tuple satisfies the selection
  - projection: one tuple per input tuple, even if it is a duplicate
  - cross product: If there are *m* copies of *t1* in *r*, and *n* copies of *t2* in *s*, there are *m* x *n* copies of *t1.t2* in *r* x *s*
  - Other operators similarly defined
    - union: *m* + *n copies*
    - intersection: min(*m, n*) copies
    - difference: max(0, *m* – *n*) copies

# Outline

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



# Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL
  - 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…**

# SQL: Data Definition Language

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

- The schema for each relation.
- *Keys*
- 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.



# Keys (more later)

- Let $K \subseteq R$ (R is a set of columns)
- K is a **superkey** of R if values for K are sufficient to identify a unique row of any possible table
  - *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 candidate key can be the **primary key for a** table
  - Typically one that is small and immutable (doesn't change often)
  - Chosen by app/user
- *Keys are unique!*

# 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)



# 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),
    building      varchar(15),
    budget        numeric(12,2) check (budget > 0),

    primary key (dept_name)
);
```

Might not be a key, but must be unique!

```
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
)
```

unique

Maybe not unique!



# 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: DML

- 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 (possibly foreign keys).



# 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)

# SQL Constructs: Insert/Delete/Update Tuples

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

```
update instructor
set salary = salary * 1.05
where salary ≤ 100000;

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

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

update instructor
set salary = salary * 1.05
where salary ≤ 100000;
```

  - The order is important
  - Can be done better using the <u>case</u> statement



# SQL Constructs: Insert/Delete/Update Tuples

- UPDATE <name> SET <field name> = <value> WHERE <condition>
  - Increase all salaries over $100,000 by 6%, others 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;
```

# Recap: 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: 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)
);
```

Might not be a key, but must be unique!

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

Maybe not unique!

# Basic Query Structure

**select** $A_1, A_2, \ldots, A_n$ — Attributes or expressions
**from** $r_1, r_2, \ldots, r_m$ — tables (or queries returning tables)
**where** $P$ — optional predicate

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* **desc**

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*

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;*

**select** *name*
**from** *instructor*
**where** *salary < 100000* **or** *salary >= 100000;*

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