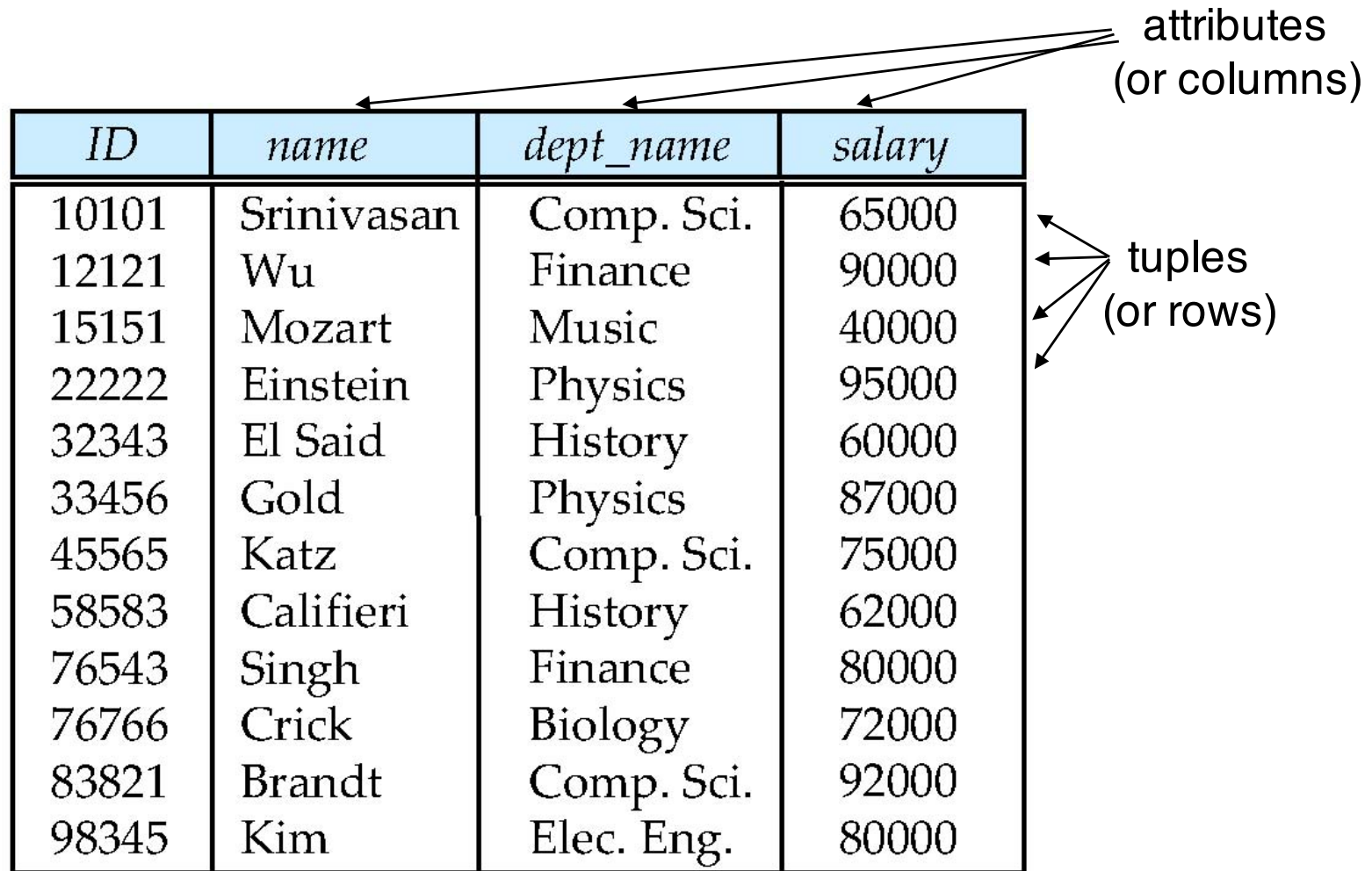


## **Chapter 2: Intro to Relational Model**

# Example of a Relation



The diagram shows a table representing a relation. The table has four columns: *ID*, *name*, *dept\_name*, and *salary*. The first row is the header, and the subsequent rows contain data. 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

# Attribute Types

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- The special value ***null*** is a member of every domain. Indicated that the value is “unknown”
- The null value causes complications in the definition of many operations

# 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  $a_i \in D_i$

- The current values (**relation instance**) of a relation are specified by a table
- An element  $t$  of  $r$  is a *tuple*, represented by a *row* in a table

# Relations are Unordered

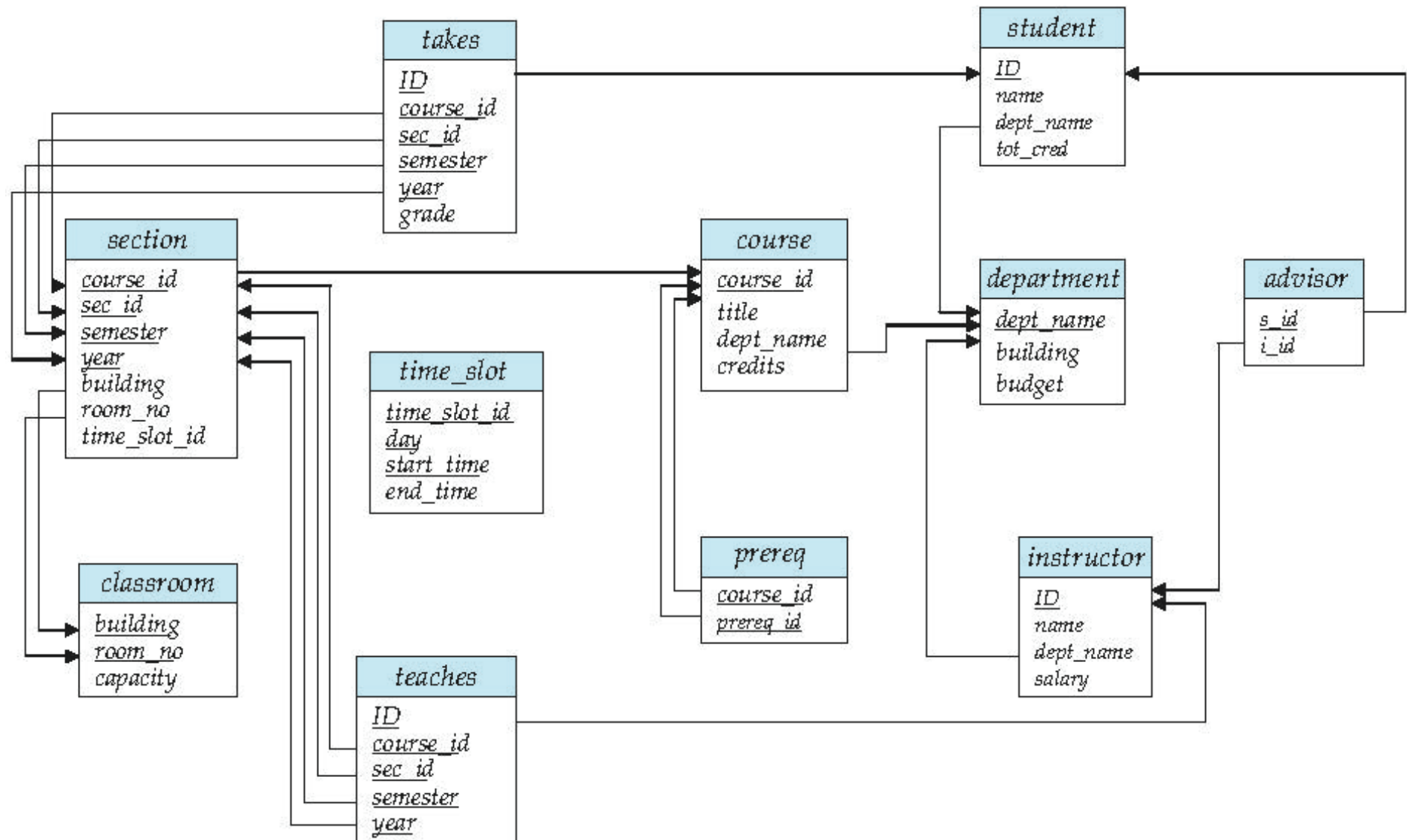
- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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

# Keys

- Let  $K \subseteq R$
- $K$  is a **superkey** of  $R$  if values for  $K$  are sufficient to identify a unique tuple of each possible relation  $r(R)$ 
  - Example:  $\{ID\}$  and  $\{ID, name\}$  are both superkeys of *instructor*.
- Superkey  $K$  is a **candidate key** if  $K$  is minimal  
Example:  $\{ID\}$  is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**.
  - which one?
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation
  - **Referenced** relation
  - Example – *dept\_name* in *instructor* is a foreign key from *instructor* referencing *department*

# Schema Diagram for University Database



# Relational Query Languages

- Procedural vs .non-procedural, or declarative
- “Pure” languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- The above 3 pure languages are equivalent in computing power
- We will concentrate in this chapter on relational algebra
  - Not Turing-machine equivalent
  - consists of 6 basic operations



# Select Operation – selection of rows (tuples)

■ Relation r

A	B	C	D
$\alpha$	$\alpha$	1	7
$\alpha$	$\beta$	5	7
$\beta$	$\beta$	12	3
$\beta$	$\beta$	23	10

■  $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
$\alpha$	$\alpha$	1	7
$\beta$	$\beta$	23	10

## Project Operation – selection of columns (Attributes)

- Relation  $r$ :

$A$	$B$	$C$
$\alpha$	10	1
$\alpha$	20	1
$\beta$	30	1
$\beta$	40	2

- $\Pi_{A,C}(r)$

$A$	$C$
$\alpha$	1
$\alpha$	1
$\beta$	1
$\beta$	2

 $=$ 

$A$	$C$
$\alpha$	1
$\beta$	1
$\beta$	2

# Union of two relations

■ Relations  $r, s$ :

$A$	$B$
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

$A$	$B$
$\alpha$	2
$\beta$	3

$s$

■  $r \cup s$ :

$A$	$B$
$\alpha$	1
$\alpha$	2
$\beta$	1
$\beta$	3

# Set difference of two relations

- Relations  $r, s$ :

$A$	$B$
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

$A$	$B$
$\alpha$	2
$\beta$	3

$s$

- $r - s$ :

$A$	$B$
$\alpha$	1
$\beta$	1

# Set intersection of two relations

■ Relation  $r, s$ :

$A$	$B$
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

$A$	$B$
$\alpha$	2
$\beta$	3

$s$

■  $r \cap s$

$A$	$B$
$\alpha$	2

Note:  $r \cap s = r - (r - s)$

# joining two relations -- Cartesian-product

■ Relations  $r, s$ :

$A$	$B$
$\alpha$	1
$\beta$	2

$r$

$C$	$D$	$E$
$\alpha$	10	a
$\beta$	10	a
$\beta$	20	b
$\gamma$	10	b

$s$

■  $r \times s$ :

$A$	$B$	$C$	$D$	$E$
$\alpha$	1	$\alpha$	10	a
$\alpha$	1	$\beta$	10	a
$\alpha$	1	$\beta$	20	b
$\alpha$	1	$\gamma$	10	b
$\beta$	2	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b
$\beta$	2	$\gamma$	10	b

# Cartesian-product – naming issue

■ Relations  $r, s$ :

$A$	$B$
$\alpha$	1
$\beta$	2

$r$

$B$	$D$	$E$
$\alpha$	10	a
$\beta$	10	a
$\beta$	20	b
$\gamma$	10	b

$s$

■  $r \times s$ :

$A$	$r.B$	$s.B$	$D$	$E$
$\alpha$	1	$\alpha$	10	a
$\alpha$	1	$\beta$	10	a
$\alpha$	1	$\beta$	20	b
$\alpha$	1	$\gamma$	10	b
$\beta$	2	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b
$\beta$	2	$\gamma$	10	b

# Renaming a Table

- Allows us to refer to a relation, (say  $E$ ) by more than one name.

$$\rho_x(E)$$

returns the expression  $E$  under the name  $X$

- Relations  $r$

$A$	$B$
$\alpha$	1
$\beta$	2

$r$

- $r \times \rho_s(r)$

$r.A$	$r.B$	$s.A$	$s.B$
$\alpha$	1	$\alpha$	1
$\alpha$	1	$\beta$	2
$\beta$	2	$\alpha$	1
$\beta$	2	$\beta$	2



# Composition of Operations

- Can build expressions using multiple operations
- Example:  $\sigma_{A=C}(r \times s)$

■  $r \times s$

A	B	C	D	E
$\alpha$	1	$\alpha$	10	a
$\alpha$	1	$\beta$	10	a
$\alpha$	1	$\beta$	20	b
$\alpha$	1	$\gamma$	10	b
$\beta$	2	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b
$\beta$	2	$\gamma$	10	b

sigma A=C(r) X S

■  $\sigma_{A=C}(r \times s)$

A	B	C	D	E
$\alpha$	1	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b

$$\sigma_{A=C}(r) \times s$$

# Joining two relations – Natural Join

- Let  $r$  and  $s$  be relations on schemas  $R$  and  $S$  respectively. Then, the “natural join” of relations  $R$  and  $S$  is a relation on schema  $R \cup S$  obtained as follows:
  - Consider each pair of tuples  $t_r$  from  $r$  and  $t_s$  from  $s$ .
  - If  $t_r$  and  $t_s$  have the same value on each of the attributes in  $R \cap S$ , add a tuple  $t$  to the result, where
    - ▶  $t$  has the same value as  $t_r$  on  $r$
    - ▶  $t$  has the same value as  $t_s$  on  $s$

# Natural Join Example

- Relations  $r, s$ :

A	B	C	D
$\alpha$	1	$\alpha$	a
$\beta$	2	$\gamma$	a
$\gamma$	4	$\beta$	b
$\alpha$	1	$\gamma$	a
$\delta$	2	$\beta$	b

$r$

B	D	E
1	a	$\alpha$
3	a	$\beta$
1	a	$\gamma$
2	b	$\delta$
3	b	$\epsilon$

$s$

- Natural Join

- $r \bowtie s$

A	B	C	D	E
$\alpha$	1	$\alpha$	a	$\alpha$
$\alpha$	1	$\alpha$	a	$\gamma$
$\alpha$	1	$\gamma$	a	$\alpha$
$\alpha$	1	$\gamma$	a	$\gamma$
$\delta$	2	$\beta$	b	$\delta$

$$\Pi_{A, r.B, C, r.D, E}(\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$$

# Notes about Relational Languages

- Each Query input is a table (or set of tables)
- Each query output is a table.
- All data in the output table appears in one of the input tables
- Relational Algebra is not Turing complete
- Can we compute:
  - SUM
  - AVG
  - MAX
  - MIN

# Summary of Relational Algebra Operators

Symbol (Name)	Example of Use
$\sigma$ (Selection)	$\sigma \text{ salary} \geq 85000 \text{ (instructor)}$
	Return rows of the input relation that satisfy the predicate.
$\Pi$ (Projection)	$\Pi ID, salary \text{ (instructor)}$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
$\times$ (Cartesian Product)	$\text{instructor} \times \text{department}$
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
$\cup$ (Union)	$\Pi name \text{ (instructor)} \cup \Pi name \text{ (student)}$
	Output the union of tuples from the <i>two</i> input relations.
$-$ (Set Difference)	$\Pi name \text{ (instructor)} - \Pi name \text{ (student)}$
	Output the set difference of tuples from the two input relations.
$\bowtie$ (Natural Join)	$\text{instructor} \bowtie \text{department}$
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.

**End of Chapter 2**

# **Chapter 3: Introduction to SQL**

# Outline

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database



# 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 (language name became Y2K compliant!)
  - 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 here may work on your particular system.

# 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
- And as we will see later, 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.

# 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. (ex., **numeric**(3,1), allows 44.5 to be stored exactly, but not 444.5 or 0.32)
- **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 in Chapter 4.

# Create Table Construct

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

```
create table  $r$  ( $A_1 D_1, A_2 D_2, \dots, A_n D_n,$   
                (integrity-constraint1),  
                ...,  
                (integrity-constraintk))
```

- $r$  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),  
    dept_name varchar(20),  
    salary     numeric(8,2))
```

# Integrity Constraints in Create Table

- **not null**
- **primary key** ( $A_1, \dots, A_n$ )
- **foreign key** ( $A_m, \dots, A_n$ ) **references**  $r$

*Example:*

```
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);
```

**primary key** declaration on an attribute automatically ensures **not null**

# 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*);
  
- **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*,  
    **foreign key** (*course\_id*, *sec\_id*, *semester*, *year*) **references** *section*);
  
- 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),  
    title        varchar(50),  
    dept_name    varchar(20),  
    credits      numeric(2,0),  
    primary key (course_id),  
    foreign key (dept_name) references department);
```

# Updates to tables

## ■ Insert

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

## ■ Delete

- Remove all tuples from the *student* relation
  - ▶ **delete from** *student*

## ■ Drop Table

- **drop table** *r*

## ■ Alter

- **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 exiting 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.



# 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 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 (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 department names of all instructors, and remove duplicates

```
select distinct dept_name  
from instructor
```

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

- An attribute can be a literal with no **from** clause

**select** '437'

- Results is a table with one column and a single row with value “437”
- Can give the column a name using:

**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”

# The select Clause (Cont.)

- 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 “*salary/12*” using the **as** clause:

```
select ID, name, salary/12 as monthly_salary
```