MYSQL Datatypes

Numeric Types	TINYINT,SMALLINT,MEDIUMINT,INT,BIGINT	
	FLOAT(P) ,FLOAT(M,D) ,REAL(M,D) ,DOUBLE(P)	
	DECIMAL(5,2) ,numeric()	
	bit	
Date and time	DATE,TIME,DATETIME,TIMESTAMP,YEAR	
String	Char,varchar	
	Binary,varbinary	
	Blob	
	TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT	

Create schema

CREATE {DATABASE | SCHEMA}
[IF NOT EXISTS]
db_name

[*create_specification*] ...

create_specification: [DEFAULT] CHARACTER SET [=] *charset_name* |
[DEFAULT] COLLATE [=] *collation_name*

DDL

- CREATE
- DROP
- ALTER
- TRUNCATE
- COMMENT
- RENAME

CREATE

- CREATE [TEMPORARY] TABLE [IF NOT EXISTS] *tbl_name* (*create_definition*,...) [*table_options*]

- *create_definition*: *col_name column_definition*

DROP

- DROP {DATABASE | SCHEMA} [IF EXISTS] *db_name*

- DROP [TEMPORARY] TABLE [IF EXISTS] *tbl_name* [, *tbl_name*] ...

ALTER

- ALTER TABLE *tbl_name*

[*alter_specification* [, *alter_specification*] ...]

alter_specification: *table_options*

- ADD [COLUMN] *col_name column_definition* [FIRST | AFTER *col_name*]
- ADD [CONSTRAINT [*symbol*]] PRIMARY KEY
- ADD [CONSTRAINT [*symbol*]] UNIQUE
- ADD [CONSTRAINT [*symbol*]] FOREIGN KEY [*index_name*] (*col_name*,...) *reference_definition*
- ADD CHECK (*expr*)

- *alter_specification*: *table_options*
  - CHANGE [COLUMN] *old_col_name new_col_name column_definition* [FIRST|AFTER *col_name*]
  - DROP [COLUMN] *col_name*
  - DROP PRIMARY KEY
  - DROP FOREIGN KEY *fk_symbol*
  - MODIFY [COLUMN] *col_name column_definition* [FIRST | AFTER *col_name*]

TRUNCATE

- TRUNCATE [TABLE] *tbl_name*

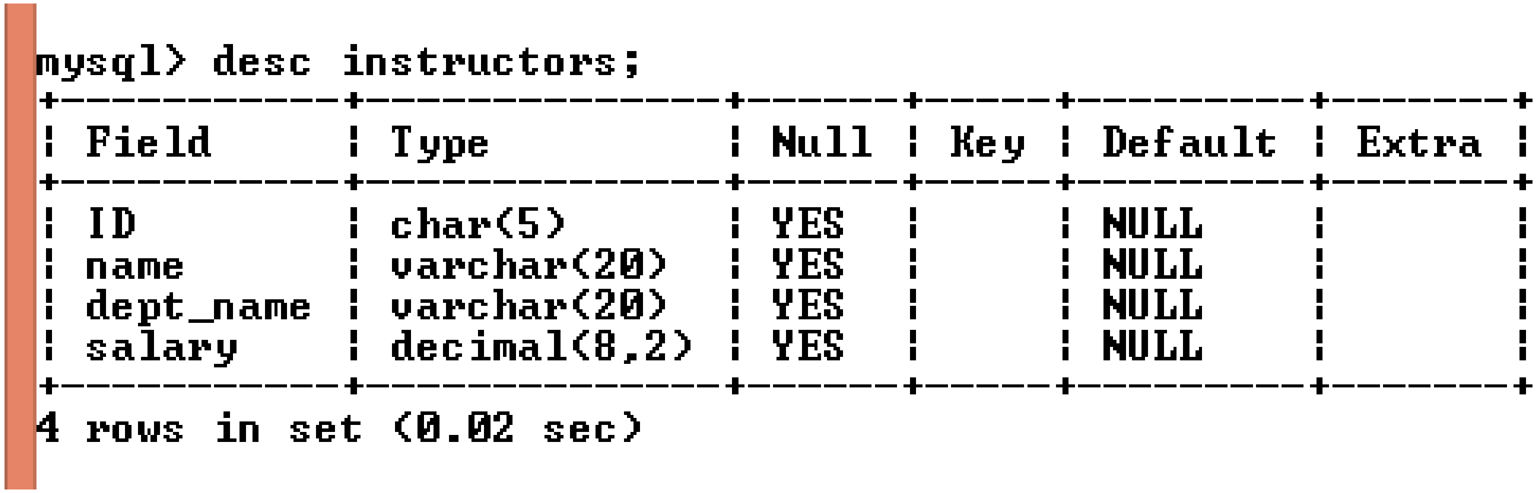
RENAME

- RENAME TABLE *tbl_name* TO *new_tbl_name* [, *tbl_name2* TO *new_tbl_name2*] ...

- **create database** College ;
- **create table** *instructors* (

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

- desc instructors;



```
mysql> desc instructors;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| ID        | char(5)      | YES  |     | NULL    |       |
| name      | varchar(20)  | YES  |     | NULL    |       |
| dept_name | varchar(20)  | YES  |     | NULL    |       |
| salary    | decimal(8,2) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
4 rows in set (0.02 sec)
```

Constraints

- Primary key
- Unique
- Not null
- Default
- Check
- Foreign key

Primary key

- PRIMARY KEY constraint for a table enforces the table to accept unique data for a specific column and this constraint creates a unique index for accessing the table faster.

```
CREATE TABLE IF NOT EXISTS
newauthor
(aut_id varchar(8) NOT NULL ,
 aut_name varchar(50) NOT NULL,
country varchar(25) NOT NULL,
 home_city varchar(25) NOT NULL, PRIMARY KEY (aut_id));
```

```
CREATE TABLE IF NOT EXISTS
Newauthor
(aut_id varchar(8) NOT NULL PRIMARY KEY,
aut_name varchar(50) NOT NULL,
country varchar(25) NOT NULL,
home_city varchar(25) NOT NULL);
```

MySQL CREATE TABLE PRIMARY KEY on multiple columns

```
CREATE TABLE IF NOT EXISTS newauthor
(aut_id varchar(8) NOT NULL ,
aut_name varchar(50) NOT NULL,
 country varchar(25) NOT NULL,
 home_city varchar(25) NOT NULL,
PRIMARY KEY (aut_id, home_city));
```

MySQL UNIQUE CONSTRAINT

```
CREATE TABLE IF NOT EXISTS newauthor
(aut_id varchar(8) NOT NULL ,
aut_name varchar(50) NOT NULL,
 country varchar(25) NOT NULL,
home_city varchar(25) NOT NULL,
UNIQUE (aut_id));
```

MySQL CREATE TABLE with not NULL CONSTRAINT

-
  ```
  CREATE TABLE IF NOT EXISTS
  newauthor(aut_id varchar(8) NOT NULL ,
  aut_name varchar(50) NOT NULL,
  country varchar(25) NOT NULL,
   home_city varchar(25) NOT NULL,
  UNIQUE (aut_id));
  ```

-
  ```
  CREATE TABLE IF NOT EXISTS
  Newauthor
  (aut_id varchar(8) NOT NULL UNIQUE ,
  aut_name varchar(50) NOT NULL,
  country varchar(25) NOT NULL,
  home_city varchar(25) NOT NULL);
  ```

MySQL CREATE TABLE with DEFAULT CONSTRAINT

- CREATE TABLE IF NOT EXISTS

newpublisher (pub_id varchar(8) NOT NULL UNIQUE DEFAULT ‘ ' ,

pub_name varchar(50) NOT NULL DEFAULT ‘ ' ,

pub_city varchar(25) NOT NULL DEFAULT ‘ ' ,

country varchar(25) NOT NULL DEFAULT 'India',
country_office varchar(25) ,

no_of_branch int(3),

estd date ,

PRIMARY KEY (pub_id));

MySQL CREATE TABLE to check values with CHECK CONSTRAINT

```
CREATE TABLE IF NOT EXISTS
newbook_mast
 (book_id varchar(15) NOT NULL UNIQUE, book_name
varchar(50) ,
 isbn_no varchar(15) NOT NULL UNIQUE ,
 cate_id varchar(8) ,
aut_id varchar(8) ,
pub_id varchar(8) ,
dt_of_pub date ,
pub_lang varchar(15) ,
 no_page decimal(5,0) CHECK(no_page>0) ,
book_price decimal(8,2) , PRIMARY KEY (book_id) );
```

MySQL creating table with FOREIGN KEY CONSTRAINT

```
CREATE TABLE IF NOT EXISTS
newbook_mast
(book_id varchar(15) NOT NULL PRIMARY KEY, book_name varchar(50)
,
 isbn_no varchar(15) NOT NULL ,
 cate_id varchar(8) ,
aut_id varchar(8) ,
pub_id varchar(8) ,
dt_of_pub date ,
 pub_lang varchar(15) ,
 no_page decimal(5,0) ,
book_price decimal(8,2) ,
 FOREIGN KEY (aut_id) REFERENCES newauthor(aut_id));
```

MySQL CREATE TABLE with FOREIGN KEY CONSTRAINT on multiple columns

```
CREATE TABLE IF NOT EXISTS
newbook_mast
 (book_id varchar(15) NOT NULL PRIMARY KEY, book_name varchar(50) ,
 isbn_no varchar(15) NOT NULL ,
cate_id varchar(8),
aut_id varchar(8) ,
pub_id varchar(8) ,
 dt_of_pub date ,
 pub_lang varchar(15) ,
 no_page decimal(5,0) ,
 book_price decimal(8,2) ,
FOREIGN KEY(aut_id) REFERENCES newauthor(aut_id),
FOREIGN KEY(pub_id) REFERENCES newpublisher(pub_id) );
```

MySQL CREATE TABLE with CASCADE and RESTRICT

```
CREATE TABLE IF NOT EXISTS
Newpurchase
 (invoice_no varchar(12) PRIMARY KEY,
 invoice_dt date , ord_no varchar(25) ,
ord_date date ,
receive_dt date , book_id varchar(8) ,
book_name varchar(50) ,
pub_lang varchar(8) ,
 cate_id varchar(8) ,
receive_qty int(5) ,
purch_price decimal(12,2) ,
 total_cost decimal(12,2) ,
FOREIGN KEY(ord_no,book_id) REFERENCES neworder(ord_no,book_id) ON
UPDATE CASCADE ON DELETE RESTRICT)
```

- On delete/update
  - Set null
  - Cascade
  - Set default
  - No action
  - RESTRICT

DML Queries

- Select
- Insert
- Update
- delete

insert

INSERT INTO *table_name*
(column1, *column2*, *column3*, ...)
VALUES (*value1*, *value2*, *value3*, ...);

INSERT INTO *table_name*
VALUES (*value1*, *value2*, *value3*, ...);

```
INSERT INTO members
(full_names,gender,physical_address,contact_number)
VALUES ('Leonard
Hofstadter','Male','Woodcrest',0845738767);
```

update

- UPDATE *table_name*
  SET *column1 = value1*, *column2 = value2*, ...
  WHERE *condition*;

- UPDATE Customers
  SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
  WHERE CustomerID = 1;

- UPDATE Customers
  SET ContactName='Juan'
  WHERE Country='Mexico';

delete

- DELETE FROM *table_name* WHERE *condition*;
- DELETE FROM Customers WHERE CustomerName=‘ABC';

Retrieval Queries in SQL (cont.)

- Basic form of the SQL SELECT statement is called a *mapping* or a *SELECT-FROM-WHERE block*

```
SELECT    <attribute list>
FROM      <table list>
WHERE     <condition>
```

  – <attribute list> is a list of attribute names whose values are to be retrieved by the query
  – <table list> is a list of the relation names required to process the query
  – <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

Relational Database Schema--Figure 5.5

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

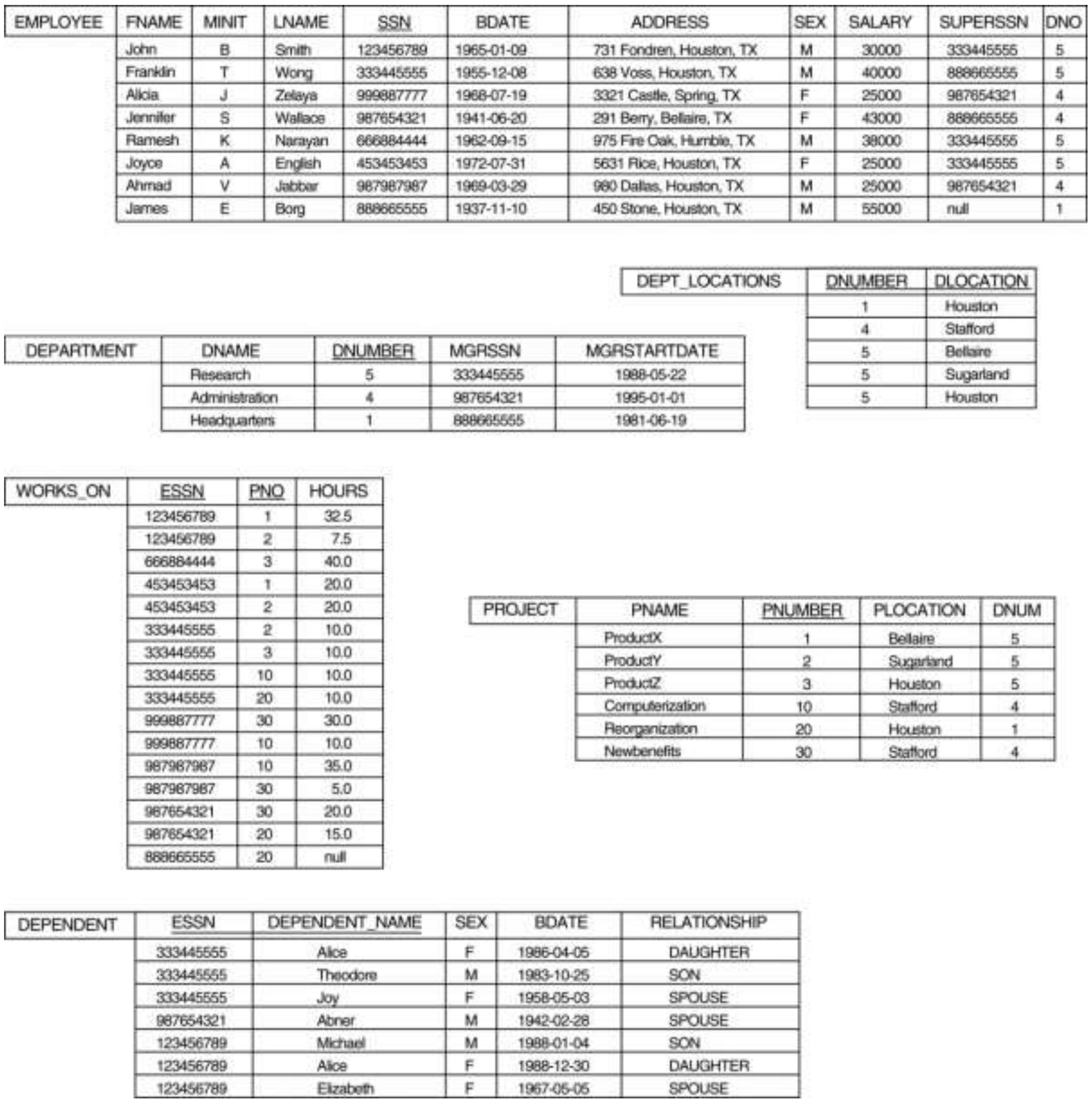
WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Populated Database--Fig.5.6



EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

Simple SQL Queries

- Basic SQL queries correspond to using the SELECT, PROJECT, and JOIN operations of the relational algebra
- All subsequent examples use the COMPANY database
- Example of a simple query on *one* relation
- Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

```
Q0:  SELECT     BDATE, ADDRESS
     FROM                 EMPLOYEE
     WHERE    FNAME='John' AND MINIT='B'
    AND                    LNAME='Smith'
```

  - Similar to a SELECT-PROJECT pair of relational algebra operations; the SELECT-clause specifies the *projection attributes* and the WHERE-clause specifies the *selection condition*
  - However, the result of the query *may contain* duplicate tuples

Aliases, * and DISTINCT, Empty WHERE-clause

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in *different relations*
  A query that refers to two or more attributes with the same name must *qualify* the attribute name with the relation name by *prefixing* the relation name to the attribute name

<u>Example:</u>

- EMPLOYEE.LNAME, DEPARTMENT.DNAME

The from Clause

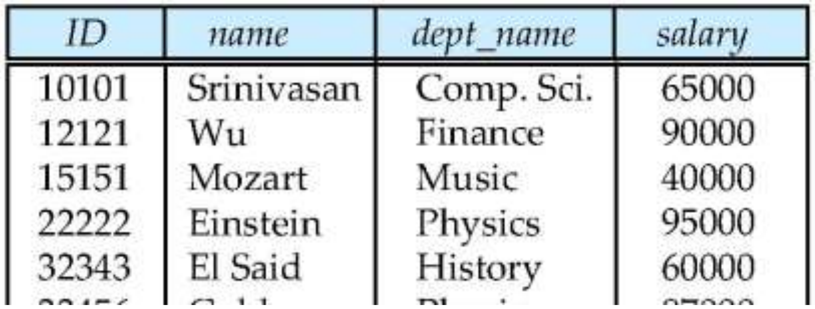
- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*

  select ∗
  from *instructor, teaches*

  - generates every possible instructor – teaches pair, with all attributes from both relations.
  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

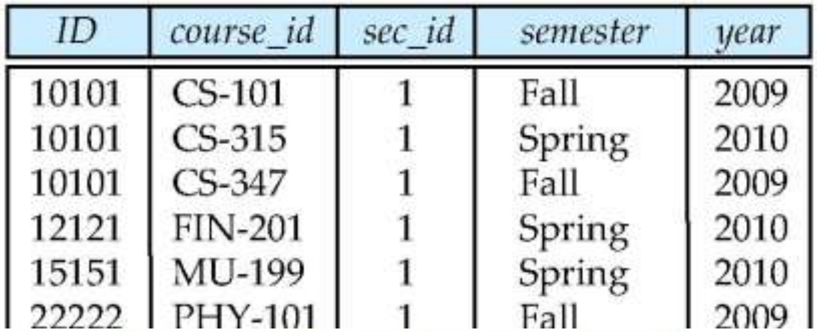
Cartesian Product

instructor

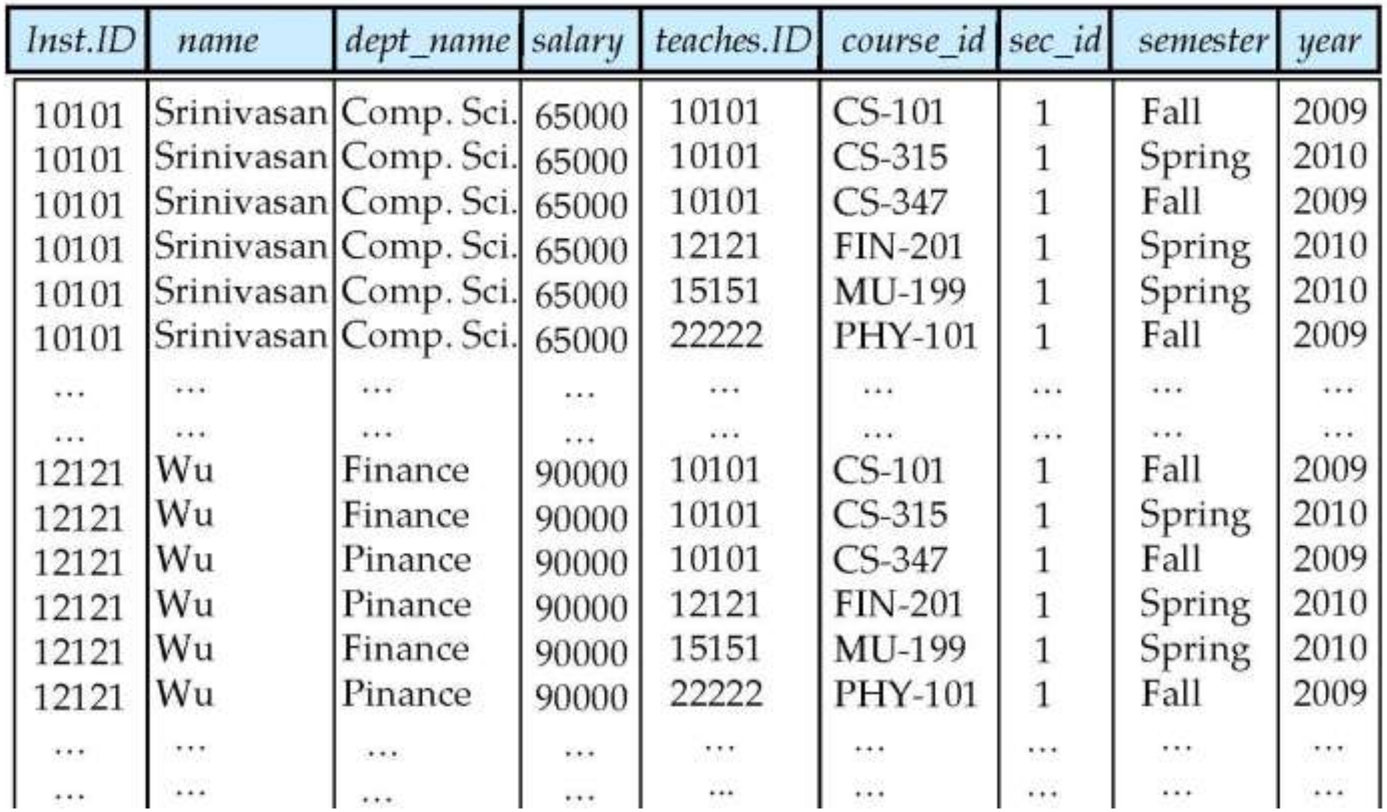


ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

teaches



ID	*course_id*	*sec_id*	*semester*	*year*
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



Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Simple SQL Queries (cont.)

- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1: SELECT   FNAME, LNAME, ADDRESS
    FROM     EMPLOYEE, DEPARTMENT
    WHERE    DNAME='Research' AND DNUMBER=DNO
```

  - Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
  - (DNAME='Research') is a *selection condition* (corresponds to a SELECT operation in relational algebra)
  - (DNUMBER=DNO) is a *join condition* (corresponds to a JOIN operation in relational algebra)

Simple SQL Queries (cont.)

- Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

```
Q2: SELECT   PNUMBER, DNUM, LNAME, BDATE, ADDRESS
    FROM                PROJECT, DEPARTMENT, EMPLOYEE
    WHERE    DNUM=DNUMBER AND MGRSSN=SSN AND
    PLOCATION='Stafford'
```

  - In Q2, there are *two* join conditions
  - The join condition DNUM=DNUMBER relates a project to its controlling department
  - The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department

ALIASES

- Some queries need to refer to the same relation twice
- In this case, *aliases* are given to the relation name
- Query 8: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

```
Q8:   SELECT   E.FNAME, E.LNAME, S.FNAME,
               S.LNAME
      FROM             EMPLOYEE E S
      WHERE    E.SUPERSSN=S.SSN
```

  - In Q8, the alternate relation names E and S are called *aliases* or *tuple variables* for the EMPLOYEE relation
  - We can think of E and S as two *different copies* of EMPLOYEE; E represents employees in role of *supervisees* and S represents employees in role of *supervisors*

ALIASES (cont.)

– Aliasing can also be used in any SQL query for convenience
  Can also use the AS keyword to specify aliases

```
Q8:     SELECT  E.FNAME, E.LNAME, S.FNAME,
        S.LNAME
        FROM            EMPLOYEE AS E, EMPLOYEE AS S
        WHERE   E.SUPERSSN=S.SSN
```

USE OF *

- To retrieve all the attribute values of the selected tuples, a * is used, which stands for *all the attributes*
  Examples:

```
Q1C:    SELECT   *
        FROM  EMPLOYEE
        WHERE    DNO=5
```

USE OF DISTINCT

- SQL does not treat a relation as a set; *duplicate tuples can appear*
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

```
Q11:  SELECT    SALARY
                FROM     EMPLOYEE

Q11A:           SELECT   DISTINCT SALARY
                FROM     EMPLOYEE
```

- The keyword **all** specifies that duplicates should not be removed.

```
select all SALARY
from EMPLOYE
```

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** *TEMP*

- 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*

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

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.'

- Comparison results can be combined using the logical connectives **and, or,** and **not**
  - To find all instructors in Comp. Sci. dept with salary > 80000

    select *name*
    from *instructor*
    where *dept_name* = 'Comp. Sci.' **and** *salary* > 80000

- Comparisons can be applied to results of arithmetic expressions.

UNSPECIFIED WHERE-clause

- A *missing WHERE-clause* indicates no condition; hence, *all tuples* of the relations in the FROM-clause are selected
- This is equivalent to the condition WHERE TRUE
- Query 9: Retrieve the SSN values for all employees.

```
Q9:SELECT     SSN
         FROM  EMPLOYEE
```

- If more than one relation is specified in the FROM-clause *and* there is no join condition, then the *CARTESIAN PRODUCT* of tuples is selected