SQL - Basics

CS 4750
Database Systems

[A. Silberschatz, H. F. Korth, S. Sudarshan, Database System Concepts, Ch.3]

QUERY LANGUAGES

- Parts of a query language (such as Relational Algebra and SQL) are divided into two main categories:

Data Definition Language (DDL)

- Effects schema
- `CREATE TABLE`
- `ALTER TABLE`
- `DROP TABLE`

Data Manipulation Language (DML)

- Effects instance
- `SELECT`
- `INSERT`
- `UPDATE`
- `DELETE`

Structured Query Language (SQL)

- Standard language for relational database managements
- Domain-specific language
  - SQL only works on relational databases
  - Not for general purpose programming (e.g., Java, C/C++, Python)
- Include the ability to process sets of data
  - Querying data
  - Controlling access to the database and its objects
  - Guaranteeing database consistency
  - Updating rows in a table
  - Creating, replacing, altering, and dropping objects

Make it possible to work with data at the logical level

SQL (cont.)

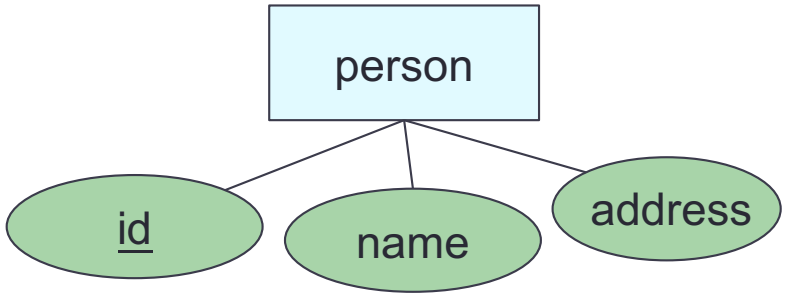
- Provides standard type; for example,
  - Numbers: `INT`, `FLOAT`, `DECIMAL(`*p*`, `*s*`)`
  - Strings:
    - `CHAR(`*n*`)` – fixed length *n*
    - `VARCHAR(`*n*`)` – variable length, max length *n*
    - `TEXT` – for large value; not support `DEFAULT` values, `NOT NULL`
  - `BOOLEAN`
  - `DATE, TIME, TIMESTAMP`
    - `DATE` – year, month, and day values
    - `TIME` – hour, minute, and second values
    - `TIMESTAMP` – year, month, day, hour, minute, and second values
  - `BLOB` (Binary Large Object) – varying-length binary string; for file, image, video, large object

SQL Statements

```
CREATE TABLE ...
DROP TABLE ...
ALTER TABLE ... ADD / REMOVE ...
INSERT INTO ... VALUES
DELETE FROM ... WHERE ...
UPDATE ... SET ... WHERE ...
SELECT ... FROM ...
... UNION ...
... INTERSECT ...
```

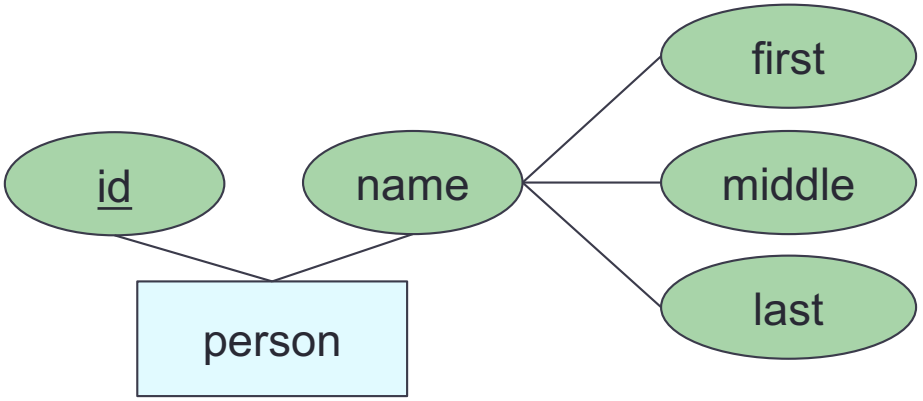
[More constraints and referential constraint maintenance later]

CREATE TABLE (Entity Set)



person(<u>id</u>, name, address)

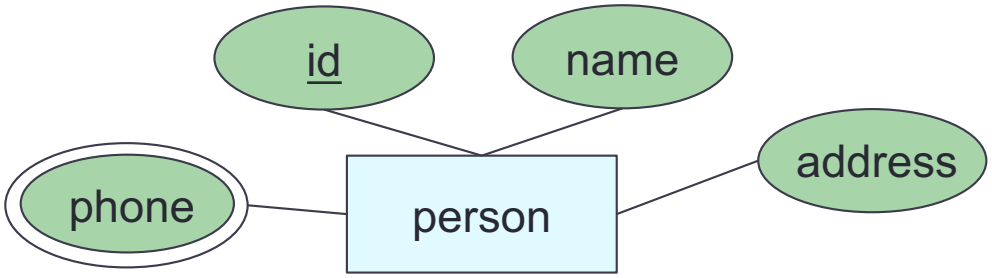
```
CREATE TABLE person (
        id INT,
        name VARCHAR(40),
        address VARCHAR(255),
        PRIMARY KEY (id) );
```



person(<u>id</u>, first_name,
middle_name, last_name)

```
CREATE TABLE person (
        id INT,
        first_name VARCHAR(20),
        middle_name VARCHAR(3),
        last_name VARCHAR(20),
        PRIMARY KEY (id) );
```

CREATE TABLE (Entity Set)



```
person(id, name, address)
person_phone(id, phone)
```

```
CREATE TABLE person ( id INT,
                      name VARCHAR(40),
                      address VARCHAR(255),
                      PRIMARY KEY (id) );

CREATE TABLE person_phone ( id INT REFERENCES person(id),
                            phone VARCHAR(10),
                            PRIMARY KEY (id, phone) );
```

CREATE TABLE (One-to-One)

Option 1: create tables for entity sets and a relationship set


```
CREATE TABLE product(pid VARCHAR(10), ...,
       PRIMARY KEY (pid) );

CREATE TABLE company(cname VARCHAR(100), ...,
       PRIMARY KEY (cname) );

CREATE TABLE makes(
     [ cname VARCHAR(100) UNIQUE REFERENCES company(cname),
       pid VARCHAR(100) UNIQUE REFERENCES product(pid),
       ... );
```

```
PRIMARY KEY (cname, pid)
```

Do we need all these tables?

CREATE TABLE (One-to-One)

Option 2: create tables for entity sets, store a primary key of one entity in another entity, no table for relationship set


```
CREATE TABLE product(pid VARCHAR(10), ...,
       PRIMARY KEY (pid) );

CREATE TABLE company(cname VARCHAR(100), ...,
       pid VARCHAR(10),
       PRIMARY KEY (cname),
       FOREIGN KEY (pid) REFERENCES product(pid) );
```

Make use of the one-to-one fact. Better design – fewer tables

CREATE TABLE (Many-to-Many)


```
CREATE TABLE product(pid VARCHAR(10), ...,
       PRIMARY KEY (pid) );

CREATE TABLE company(cname VARCHAR(100), ...,
       PRIMARY KEY (cname) );

CREATE TABLE makes(
       cname VARCHAR(100) REFERENCES company(cname),
       pid VARCHAR(100) REFERENCES product(pid), ...,
       PRIMARY KEY (cname, pid) );
```

CREATE TABLE (One-to-Many / Many-to-One)

Option 1: create tables for entity sets and a relationship set


```
CREATE TABLE product(pid VARCHAR(10), ...,
       PRIMARY KEY (pid) );

CREATE TABLE company(cname VARCHAR(100), ...,
       PRIMARY KEY (cname) );

CREATE TABLE makes(
       cname VARCHAR(100) REFERENCES company(cname),
       pid VARCHAR(100) REFERENCES product(pid), ...,
       PRIMARY KEY (pid) );
```

```
FOREIGN KEY (cname) REFERENCES company(cname)
```

Do we need all these tables?

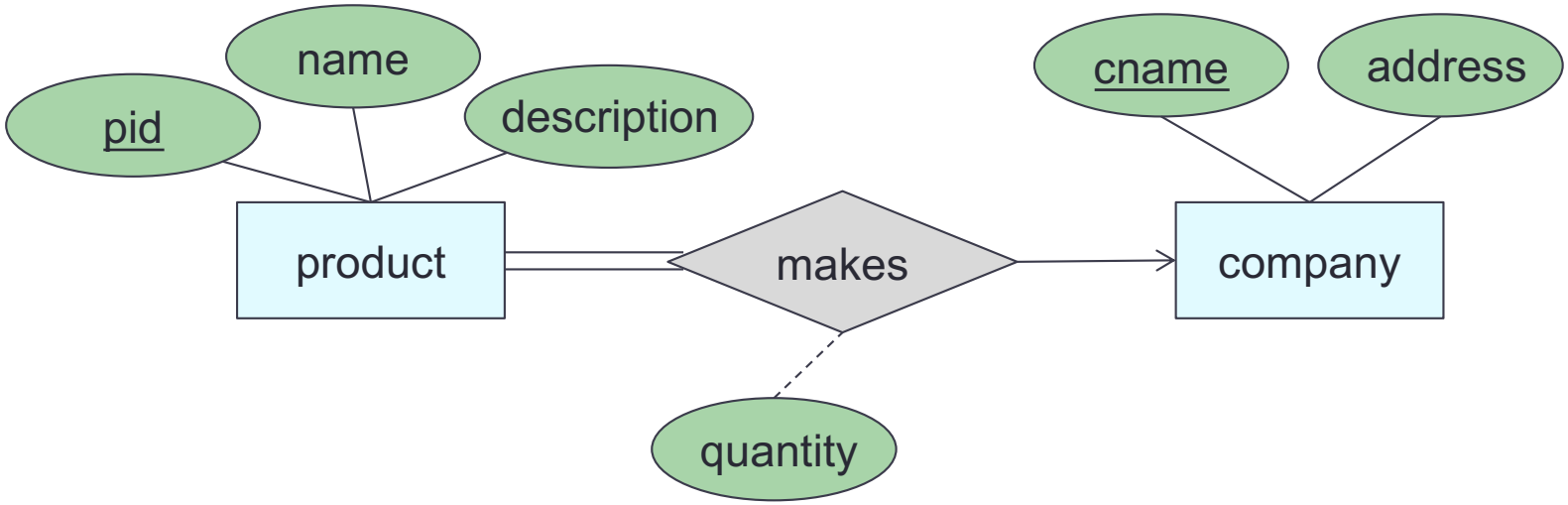
CREATE TABLE (One-to-Many / Many-to-One)

Option 2: create tables for entity sets, store a primary key of the "one" side in the "many" side, no table for relationship set


```
CREATE TABLE product(pid VARCHAR(10), ...,
       cname VARCHAR(100),
       PRIMARY KEY (pid),
       FOREIGN KEY (cname) REFERENCES company(cname) );

CREATE TABLE company(cname VARCHAR(100), ...,
       PRIMARY KEY (cname) );
```

CREATE TABLE (Total Participation)

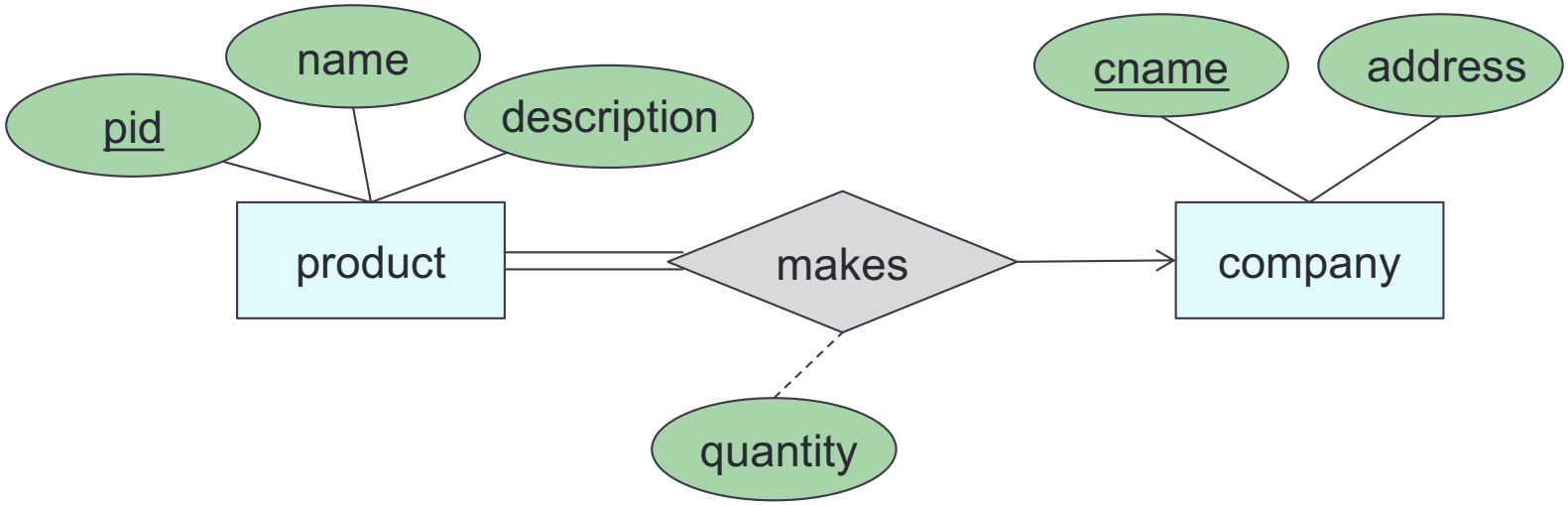
Option 1: use a FOREIGN KEY constraint




```
CREATE TABLE product(pid VARCHAR(10), ...,
       quantity INT,
       cname VARCHAR(100)
       PRIMARY KEY (pid),
       FOREIGN KEY (cname) REFERENCES company(cname) );

CREATE TABLE company(cname VARCHAR(100), ...,
       PRIMARY KEY (cname) );
```

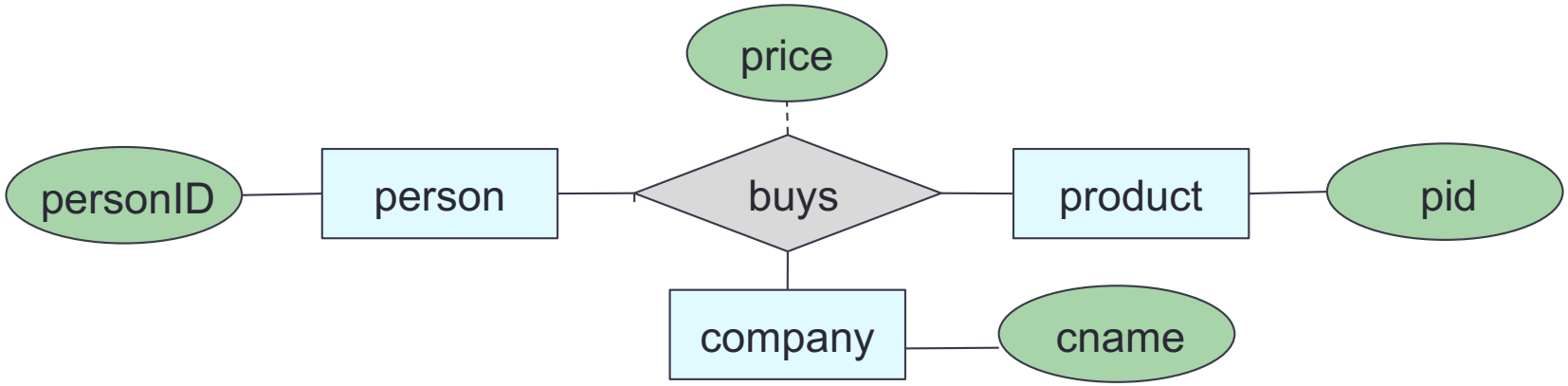
CREATE TABLE (Total Participation)

Option 2: use a NOT NULL constraint



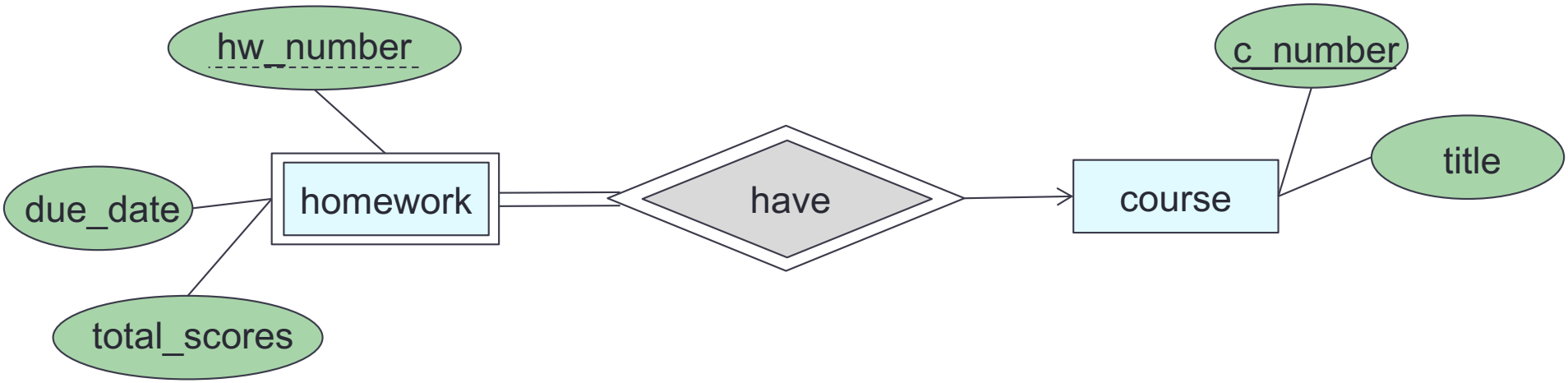

```
CREATE TABLE product(pid VARCHAR(10), ...,
       quantity INT,
       PRIMARY KEY (pid),
       cname VARCHAR(100) NOT NULL REFERENCES company(cname));

CREATE TABLE company(cname VARCHAR(100), ...,
       PRIMARY KEY (cname));
```

CREATE TABLE (Multi-Way Relations)




```
CREATE TABLE product(pid VARCHAR(10), ...,
       PRIMARY KEY (pid) );

CREATE TABLE company(cname VARCHAR(100), ...,
       PRIMARY KEY (cname) );

CREATE TABLE person VARCHAR(10), ...,
       PRIMARY KEY (personid) );

CREATE TABLE buys(
       cname VARCHAR(100) REFERENCES company(cname),
       pid VARCHAR(100) REFERENCES product(pid),
       personID VARCHAR(10) REFERENCE person(personID), ...,
       price FLOAT,
       PRIMARY KEY (cname, pid, personID,) );
```

CREATE TABLE (Weak Entity Set)



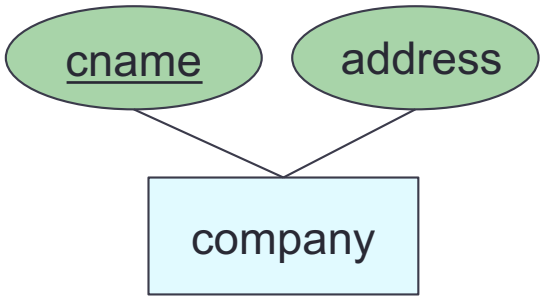

```
CREATE TABLE course(c_number VARCHAR(10), ...,
       PRIMARY KEY (c_number) );

CREATE TABLE homework(hw_number VARCHAR(10), ...,
       c_number VARCHAR(10) REFERENCES course(c_number),
       PRIMARY KEY (c_number, hw_number) );
```

DROP TABLE

```
DROP TABLE table_name;
DROP TABLE homework;
```

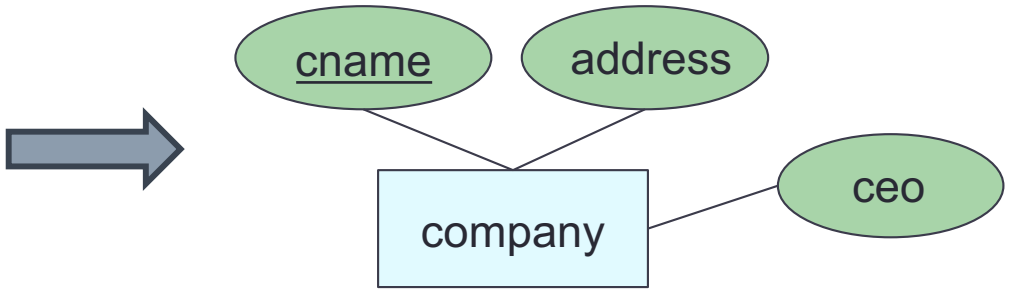
This cannot be undone !!

ALTER TABLE … ADD/DROP

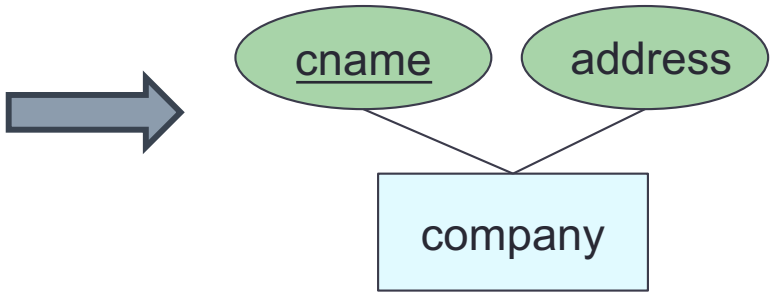



```
CREATE TABLE company (
        cname VARCHAR(40),
        address VARCHAR(255),
        PRIMARY KEY (cname));
```

```
ALTER TABLE company
ADD ceo VARCHAR(40);
```




```
ALTER TABLE company
DROP ceo;
```

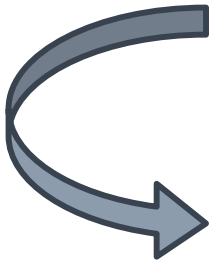


INSERT INTO … VALUE

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

```
INSERT INTO Student_lecture
VALUES ("9999", "85 Engineer's Way",
        "Database Systems", "Humpty");
```



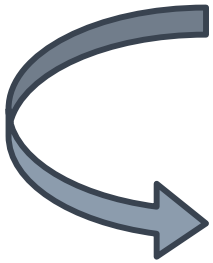
Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie
9999	85 Engineer's Way	Database Systems	Humpty

UPDATE … SET … WHERE

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie



```
UPDATE Student_lecture
SET Teaching_assistant = “Humpty”
WHERE S_id = “5678”;
```

Student_lecture

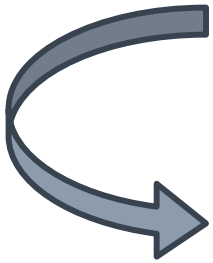
S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Humpty

DELETE FROM … WHERE

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

```
DELETE FROM Student_lecture
WHERE Teaching_assistant = "Humpty";
```



Student_lecture

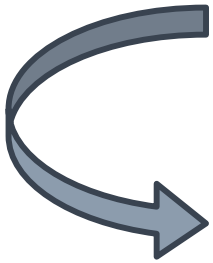
S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

SELECT * FROM ...

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

```
SELECT * FROM Student_lecture;
```



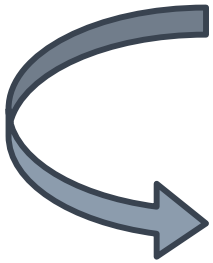
S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

SELECT Specific Attribute

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

```
SELECT S_id, Course FROM Student_lecture;
```



S_id	Course
1234	Database Systems
2345	Database Systems
3456	Cloud Computing
1234	Web Programming Lang.
5678	Software Analysis

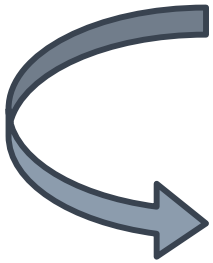
"Projection"

SELECT with WHERE Clause (1)

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

```
SELECT * FROM Student_lecture
WHERE Teaching_assistant = “Minnie” ;
```



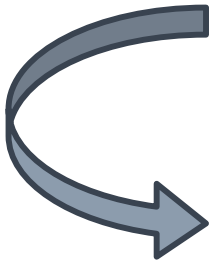
S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

SELECT with WHERE Clause (2)

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

```
SELECT S_id, Course FROM Student_lecture
WHERE Teaching_assistant = “Minnie” ;
```



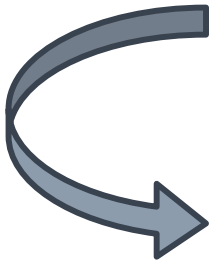
S_id	Course
1234	Database Systems
5678	Software Analysis

SELECT with WHERE Clause (3)

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

```
SELECT S_id, Course FROM Student_lecture
WHERE Teaching_assistant = “Minnie” OR
      Teaching_assistant = “Mickey” ;
```



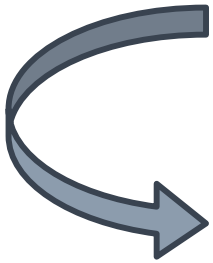
S_id	Course
1234	Database Systems
1234	Web Programming Lang.
5678	Software Analysis

SELECT DISTINCT

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

```
SELECT DISTINCT Teaching_assistant
FROM Student_lecture ;
```



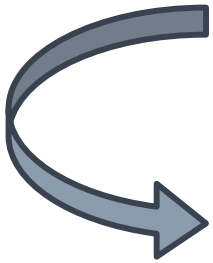
Teaching_assistant
Minnie
Humpty
Dumpty
Mickey

SELECT with Pattern Matching

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie

```
SELECT Teaching_assistant FROM Student_lecture
WHERE Course LIKE 'Data%';
```

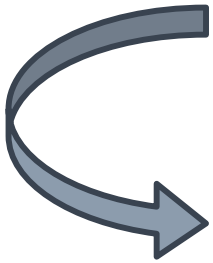


Teaching_assistant
Minnie
Humpty

ORDER BY (Ascending)

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie



```
SELECT S_id, Course FROM Student_lecture
WHERE Teaching_assistant <> "Dumpty"
ORDER BY S_id;
```

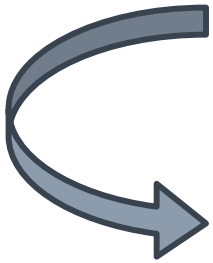
S_id	Course
1234	Database Systems
1234	Web Programming Lang.
2345	Database Systems
5678	Software Analysis

Ascending, by default
Or include ASC

ORDER BY (Descending)

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie



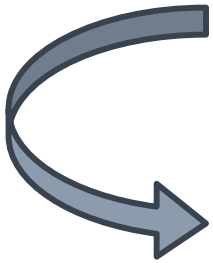
```
SELECT S_id, Course FROM Student_lecture
WHERE Teaching_assistant <> “Dumpty”
ORDER BY S_id DESC;
```

S_id	Course
5678	Software Analysis
2345	Database Systems
1234	Web Programming Lang.
1234	Database Systems

SELECT with Aliases

Student_lecture

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey
5678	2131 Forest Lake Ln.	Software Analysis	Minnie



```
SELECT S_id as ID, Course as "Course Name"
FROM Student_lecture
WHERE Teaching_assistant <> "Dumpty" ;
```

ID	Course Name
1234	Database Systems
1234	Web Programming Lang.
2345	Database Systems
5678	Software Analysis

UNION

- Combine two tables
- Each tables must have the same number of columns
- The columns must have similar / compatible data types
- The columns in both tables must be in the same order
- Tuples of the second table comes after the first

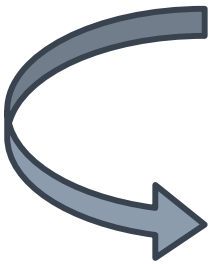
Student_lecture1

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty

Student_lecture2

S_id	Address	Course	Teaching_assistant
3456	900 S. Detroit	Cloud Computing	Dumpty

```
(SELECT S_id, Course FROM Student_lecture1) UNION
(SELECT S_id, Course FROM Student_lecture2)
```



S_id	Course
1234	Database Systems
2345	Database Systems
3456	Cloud Computing

INTERSECT

- Combine two tables
- Each table must have the same number of columns
- The columns must have similar / compatible data types
- The columns in both tables must be in the same order
- Tuples of the second table comes after the first

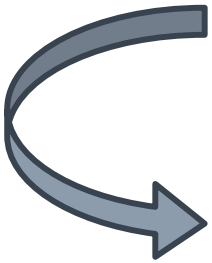
Student_lecture1

S_id	Address	Course	Teaching_assistant
1234	57 Hockanum Blvd	Database Systems	Minnie
2345	1400 E. Bellows	Database Systems	Humpty

Student_lecture2

S_id	Address	Course	Teaching_assistant
3456	900 S. Detroit	Cloud Computing	Dumpty
1234	57 Hockanum Blvd	Web Programming Lang.	Mickey

```
(SELECT S_id FROM Student_lecture1) INTERSECT
(SELECT S_id FROM Student_lecture2)
```



S_id
1234

Some DBMS (e.g., old versions of MySQL) don't support INTERSECT operator.
Good news!! Our CS server does ☺

Queries Involving Multiple Relations

- Combine the tables based on common columns

TAHiring

computingID	name	year	hours_worked
ht1y	Humpty	4	20
dt2y	Dumpty	3	20
md3y	Mickey	4	15
mn4e	Minnie	4	16
dh5h	Duhhuh	3	10

Payrate

year	hourly_rate
4	12
3	10

```
SELECT computingID, name, TAHiring.year, hourly_rate, hours_worked,
FROM TAHiring, Payrate
WHERE TAHiring.year = Payrate.year
```

computingID	name	year	hourly_rate	hours_worked
ht1y	Humpty	4	12	20
dt2y	Dumpty	3	10	20
md3y	Mickey	4	12	15
mn4e	Minnie	4	12	16
dh5h	Duhhuh	3	10	10

DBMS -- "Natural join"

Wrap-Up

- Keep the number of tables small – reduce the number of tables when you can (in many-to-one and one-to-many)
- Simple, commonly used SQL syntax

What's next?

- More SQL