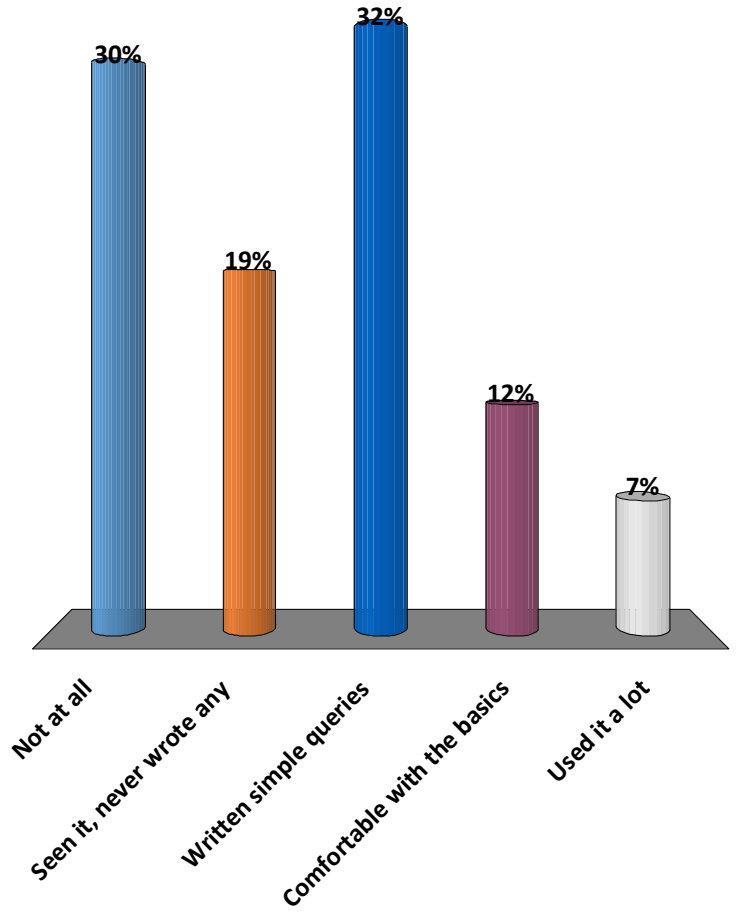
COMP 430/533
Intro. to Database Systems

SQL 1

Have you used SQL before?

A. Not at all
B. Seen it, never wrote any
C. Written simple queries
D. Comfortable with the basics
E. Used it a lot

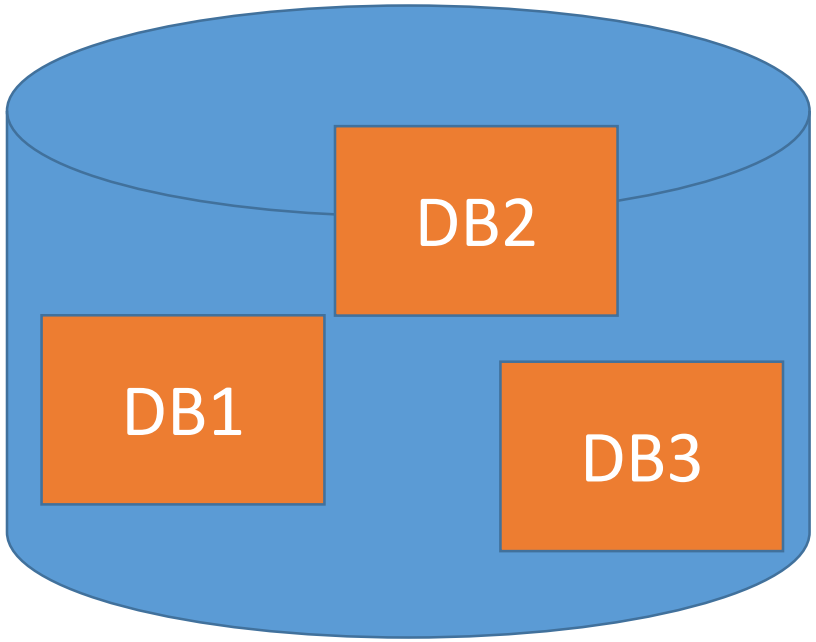


Basic concepts

Database Management System (DBMS)

Software for managing multiple databases – e.g., PostgreSQL

- Server that communicates with clients
- DB ops controlled by client, typically using SQL
- Manages users & access control
- Often given exclusive control over storage
  - Potentially no traditional file system
  - May replicate & distribute data over multiple disks or servers



We'll use three DB clients

Jupyter:

- SQL embedded within Jupyter notebooks

psql:

- Command-line
- .sql code file

user app:

- Called from main application language (Python, Java, …)
- E.g., something like SQLexecute*("…SQL CODE…")*

Table=Relation

Heading:
Names of each column.

Product

p_name	price	manufacturer
Gizmo	$19.99	GizmoWorks
Powergizmo	$39.99	GizmoWorks
Widget	$19.99	WidgetsRUs
HyperWidget	$203.99	Hyper

Row=*tuple*=*record*:
A single entry in the table.
#rows = *cardinality*

Cell=field:
One column of one row

Column=*attribute*=*field*:
A typed data entry in each row.
#columns = *arity=degree*

Table=Relation

Product

p_name	price	manufacturer
Gizmo	$19.99	GizmoWorks
Powergizmo	$39.99	GizmoWorks
Widget	$19.99	WidgetsRUs
HyperWidget	$203.99	Hyper

Table is a *multiset* of rows. Not ordered. Duplicate rows allowed.

Schema defines table attributes

Product

p_name	price	manufacturer
Gizmo	$19.99	GizmoWorks
Powergizmo	$39.99	GizmoWorks
Widget	$19.99	WidgetsRUs
HyperWidget	$203.99	Hyper

Product (p_name: string, price: float, manufacturer: string)

Creating a table

Product (p_name: string, price: money, manufacturer: string)

```
CREATE TABLE Product (
    p_name VARCHAR(50),
    price MONEY,
    manufacturer VARCHAR(50)
);
```

Some SQLs: CURRENCY, NUMERIC(6,2)

```
CREATE TYPE ProductType (
    p_name VARCHAR(50),
    price MONEY,
    manufacturer VARCHAR(50)
);
CREATE TABLE Product OF ProductType;
```

SQL attribute types

All types are atomic!

Traditionally. However, sets and arrays are allowed in some SQL versions.
We won't discuss these.

Mostly standard:

- CHAR(n), VARCHAR(n), NCHAR(n), NVARCHAR(n)
- INT, BIGINT, SMALLINT, FLOAT, DECIMAL(m,n)
- TIMESTAMP, DATE, TIME, INTERVAL
- …

Less standard:

- MONEY, BOOLEAN, UUID
- …

NULL data

Any cell can be NULL.

- Signals missing value.

Possible interpretations:

- Value doesn’t exist.
- Value exists but unknown.
- Value not applicable to this record.

Keys

Key = minimal set of attributes acting as a unique tuple identifier.

- Consider the universe of all potential relation data, not just what the table currently contains.

Product

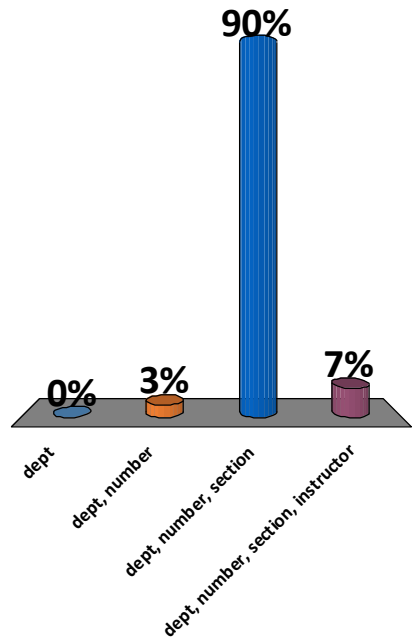
p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper

What is key?

A. dept
B. dept, number
✓ C. dept, number, section
D. dept, number, section, instructor

Course

dept	number	section	instructor
MATH	101	1	Jones
MATH	101	2	Smith
MATH	102	1	Williams
PHYS	101	1	Baker
PHYS	102	2	Baker

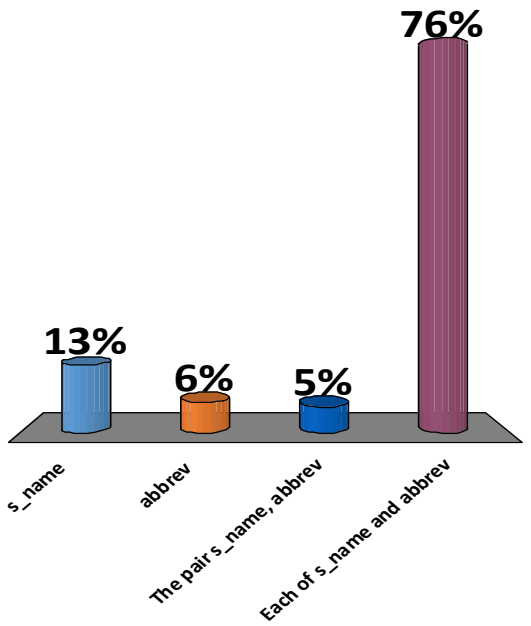


What is key?

A. s_name
B. abbrev
C. The pair s_name, abbrev
✓ D. Each of s_name and abbrev

State

s_name	abbrev	year_admitted
Alabama	AL	1819
Alaska	AK	1959
Arizona	AZ	1912
Arkansas	AR	1836
California	CA	1850



Primary keys

Every table should have one *primary key*.

- Each tuple must have distinct non-NULL values of primary key attributes.
- Guarantees table is a mathematical relation.

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper

Product (p_name: string, price: float, manufacturer: string)

Primary keys are a DB-checked constraint

Adding a new record with duplicate or NULL primary key will fail.

- Failure reported as exception or erroneous return value.

Product (p_name: string, price: float, manufacturer: string)

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper
~~Widget~~	~~15.99~~	~~GizmoWorks~~

We will see other kinds of DB-checked constraints.

What data should the primary key be?

Natural: Data field(s) needed in table anyways.

Synthetic: “Made-up” just for the purpose of being an identifier.

PostgreSQL: UUID type.

Pros/cons discussed later in design section.

Creating a table with primary key

Product (p_name: string, price: money, manufacturer: string)

```
CREATE TABLE Product (
    p_name VARCHAR(50),
    price MONEY,
    manufacturer VARCHAR(50),
    PRIMARY KEY (p_name)
);
```

```
CREATE TABLE Product (
    p_name VARCHAR(50) PRIMARY KEY,
    price MONEY,
    manufacturer VARCHAR(50)
);
```

NOT NULL constraint

Course (c_id: UUID, c_name: string, instructor: string)

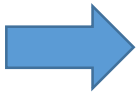
```
CREATE TABLE Course (
    c_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    c_name VARCHAR(50) NOT NULL,
    instructor VARCHAR(50)
);
```

Adding records to a table

```
INSERT INTO Product VALUES
  ('MiniGizmo', 15.99, 'GizmoWoks'),
  ('MiniWidget', 21.99, 'WidgetsRUs');
INSERT INTO Product (p_name, manufacturer) VALUES
  ('NanoWidget', 'WidgetsRUs');
```

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper



p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper
MiniGizmo	15.99	GizmoWorks
MiniWidget	21.99	WidgetsRUs
NanoWidget	NULL	WidgetsRUs

Activity: Creating a table

02a-table.ipynb

CREATE TABLE Course (…);
INSERT INTO Course …;

Course

dept	number	section	instructor
MATH	101	1	Jones
MATH	101	2	Smith
MATH	102	1	Williams
PHYS	101	1	Baker
PHYS	102	2	Baker

Course (dept, number, section, instructor)

Activity partial solution

Course (dept, number, section, instructor)

```
CREATE TABLE Course (
    dept CHAR(4),
    number CHAR(3),
    section INT DEFAULT 1,
    instructor VARCHAR(50),
    PRIMARY KEY (dept, number, section)
);
```

A convenient option.

Simple queries

SELECT-FROM-WHERE

```
SELECT   attributes
FROM     tables
WHERE    conditions;
```

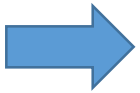
Results in a new table, which can be returned or stored.

Selecting some records

```
SELECT  *
FROM    Product
WHERE   Price > 20;
```

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper



p_name	price	manufacturer
Powergizmo	39.99	GizmoWorks
HyperWidget	203.99	Hyper

More selection examples

```
SELECT   *
FROM    Product
WHERE  price IS NOT NULL;
```

```
SELECT   *
FROM    Product
WHERE  price BETWEEN 20 AND 40;
```

Inclusive

```
SELECT   *
FROM    Product
WHERE  price > 20 AND manufacturer = ‘GizmoWorks’;
```

```
SELECT   *
FROM    Product
WHERE  manufacturer IN (‘GizmoWorks’, ‘WidgetsRUs’);
```

Selection with pattern matching

```
SELECT   *
FROM    Product
WHERE  p_name LIKE ‘%Gizmo%’;
```

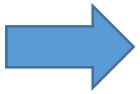
%	= Match any sequence of 0-or-more characters
_	= Match any single character
[abc]	= Match any one character listed
[a-c]	= Match any one character in range

Projecting some fields

```
SELECT   p_name, manufacturer
FROM    Product;
```

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper



p_name	manufacturer
Gizmo	GizmoWorks
Powergizmo	GizmoWorks
Widget	WidgetsRUs
HyperWidget	Hyper

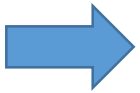
Answer (p_name, manufacturer)

Combining selection & projection

```
SELECT   p_name, manufacturer
FROM     Product
WHERE    price > 20;
```

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper



p_name	manufacturer
Powergizmo	GizmoWorks
HyperWidget	Hyper

Are results necessarily distinct?

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper

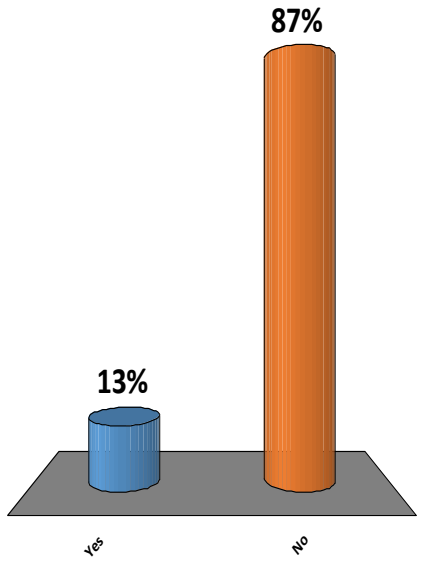
```
SELECT  manufacturer
FROM    Product;
```

A. Yes

✓ B. No

manufacturer
GizmoWorks
WidgetsRUs
Hyper

manufacturer
GizmoWorks
GizmoWorks
WidgetsRUs
Hyper

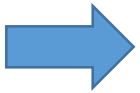


Making results distinct

```
SELECT DISTINCT   manufacturer
FROM    Product;
```

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper



manufacturer
GizmoWorks
WidgetsRUs
Hyper

Ensures results are a set.

Query semantics – set notation

```
SELECT [DISTINCT]  a1, a2, …, am
FROM  T
WHERE  FilterCondition;
```

$\{(a_1, a_2, \ldots, a_m) \mid \text{FilterCondition}\}$

Multisets by default.
Sets with DISTINCT.

Computation in SELECT clauses

```
SELECT   location, time, celsius * 1.8 + 32 AS fahrenheit
FROM     SensorReading;
```

```
SELECT   player_id, Floor(height) AS feet, (height – Floor(height)) * 12 AS inches
FROM     Player;
```

```
SELECT   student_id, CASE WHEN lastname < 'N' THEN 1 ELSE 2 END AS group
FROM     Student;
```

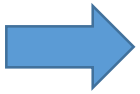
Use AS. Without it, SQL will create a default field name.

Sorting

```
SELECT   p_name, manufacturer
FROM     Product
ORDER BY   price DESC, manufacturer;
```

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper



p_name	manufacturer
HyperWidget	Hyper
Powergizmo	GizmoWorks
Gizmo	GizmoWorks
Widget	WidgetsRUs

Multiset semantics vs. Sorting

Table rows are unordered, except when they're ordered.

More accurately, unless you use ORDER BY:

- Can't assume anything about ordering.
- Ordering depends on implementation, which can vary.
- Queries don't necessarily maintain order of original table.

Subset of results

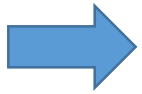
Some SQLs:
SELECT TOP 2 ...

```
SELECT   p_name, manufacturer
FROM     Product
LIMIT    2;
```

Takes first N rows. Usually used with ORDER BY.

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper



p_name	manufacturer
Gizmo	GizmoWorks
Powergizmo	GizmoWorks

Activity: Writing queries

02b-queries.ipynb

Some details

NULL semantics

NULL is not a value. It is the lack of a value.

In numeric operations:

- f(NULL) → NULL

In Boolean operations, we use 3-value logic (FALSE, UNKNOWN, TRUE):

- NULL = 'Houston' → UNKNOWN
- NULL = NULL → UNKNOWN

A	NOT A
F	T
U	U
T	F

A AND B		B		
		F	**U**	**T**
A	**F**	F	F	F
	U	F	U	U
	T	F	U	T

A OR B		B		
		F	**U**	**T**
A	**F**	F	U	T
	U	U	U	T
	T	T	T	T

Syntax details

- Strings use single quotes, not double ‘Houston’
- Equality test uses single =, not double x = 5
- Amount of whitespace doesn’t matter

Syntax details – case sensitivity

- Case insensitive
  - Keywords: SELECT, NULL
  - Table names: Product
  - Function names: Count()
- Depends on SQL version
  - Attribute names: product_id
- Case sensitive
  - String literals: ‘HOUSTON’, ‘Houston’, ‘houston’

Some useful online SQL references

Good for quick checks of statement syntax and options.

- www.w3schools.com/sql/
- www.tutorialspoint.com/sql/
- www.sqlcommands.net/
- www.techonthenet.com/sql/

Each flavor of DBMS has its own reference.

Using multiple tables

The need for multiple tables

Using a single table leads to repeating data.

- Provides the opportunity for inconsistency
- Requires more storage space & I/O time

Product

p_name	price	manufacturer	address	city	state
Gizmo	19.99	GizmoWorks	123 Gizmo St.	Houston	TX
Powergizmo	39.99	GizmoWorks	123 Gizmo St.	Houston	TX
Widget	19.99	WidgetsRUs	20 Main St.	New York	NY
HyperWidget	203.99	Hyper	1 Mission Dr.	San Francisco	CA

Product (p_name, price, manufacturer, address, city, state)

Using multiple tables

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper

Company

c_name	address	city	state
GizmoWorks	123 Gizmo St.	Houston	TX
WidgetsRUs	20 Main St.	New York	NY
Hyper	1 Mission Dr.	San Francisco	CA

Product (p_name, price, manufacturer)

Company (c_name, address, city, state)

Later in course: Deciding what fields belong in what tables.

Foreign keys & referential integrity

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper

Company

c_name	address	city	state
GizmoWorks	123 Gizmo St.	Houston	TX
WidgetsRUs	20 Main St.	New York	NY
Hyper	1 Mission Dr.	San Francisco	CA

Product's manufacturer is a *foreign key*.

- Foreign keys always refer to primary keys.
- We want to enforce *referential integrity* – each manufacturer is a c_name.

Creating a table with a foreign key

```
CREATE TABLE Product (
    p_name VARCHAR(50),
    price MONEY,
    manufacturer VARCHAR(50),
    PRIMARY KEY (p_name),
    FOREIGN KEY (manufacturer) REFERENCES Company (c_name )
);
```

```
CREATE TABLE Company (
    c_name VARCHAR(50),
    address VARCHAR(50),
    city VARCHAR(50),
    state CHAR(2),
    PRIMARY KEY (c_name)
);
```

Foreign key represents a dependence

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
…	…	…

Company

c_name	address	city	state
GizmoWorks	123 Gizmo St.	Houston	TX
…	…	…	…

Product conceptually dependent on Company to elaborate details.
Product data dependent on Company data being entered.

```
CREATE TABLE Product (
    ....
    FOREIGN KEY (manufacturer) REFERENCES Company (c_name)
);
```

```
CREATE TABLE Company (...);
```

Product def'n dependent on Company def'n.

Foreign keys vs. pointers

Foreign keys relate tables, or equivalently, sets of attributes.

- Repeats data to connect individual records.
- One relationship between tables vs. many pointers between table records.

Problems with data pointers:

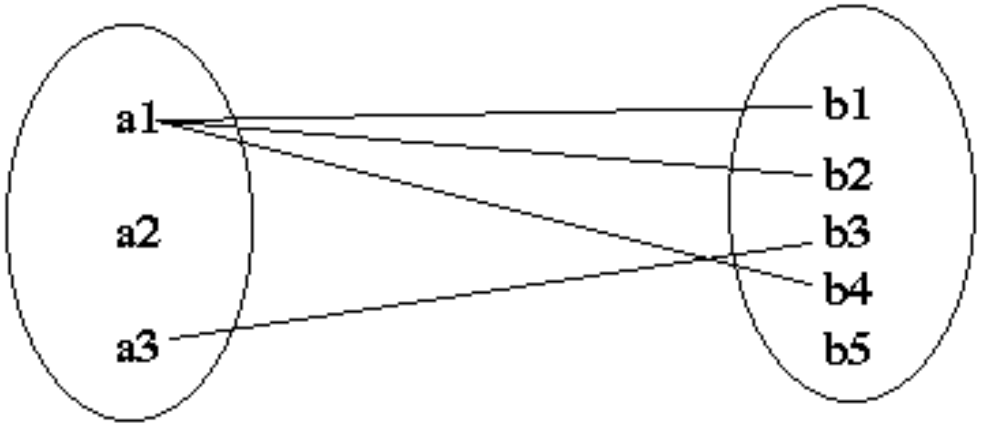
- Difficult to maintain pointers when data changes, moves, or gets copied, esp. with concurrency.
- We want to avoid making queries be directionally-biased.

In what ways do we relate tables?

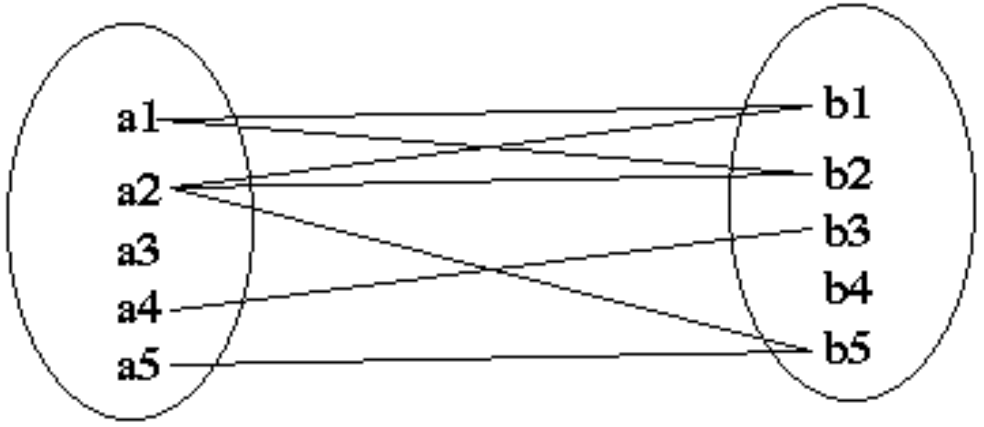
Explore design issues & common patterns later.

Two main building blocks:

- One-to-many relationships



- Many-to-many relationships



One-to-many / many-to-one relationships

Company

c_name	address	city	state
GizmoWorks	123 Gizmo St.	Houston	TX
WidgetsRUs	20 Main St.	New York	NY
Hyper	1 Mission Dr.	San Francisco	CA

Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper

Company (c_name, address, city, state)

Product (p_name, price, manufacturer)

Each Company can have many Products.

Each Product is made by exactly one Company.

Lookup tables – common use of one-to-many

Lookup table

Ref. integrity ensures Company's state is in list.

State

state
AL
…
TX
WV

Company

c_name	address	city	state
GizmoWorks	123 Gizmo St.	Houston	TX
WidgetsRUs	20 Main St.	New York	NY
Hyper	1 Mission Dr.	San Francisco	CA

State (state)

Company (c_name, address, city, state)

Each State can have many Companies.

Each Company is in one State.

Many-to-many relationships

Student

s_id	first_name	last_name
S01	John	Smith
S02	Mary	Wallace
S03	Sue	Roper
S04	Mark	Jones

Enrollment

s_id	crn	grade
S01	01234	A
S01	46117	C
S02	01234	B

Course

crn	dept	number
01234	COMP	140
15134	COMP	160
46117	ELEC	220

Each Student can be enrolled in many Courses.

Each Course has many Students.

Response Counter

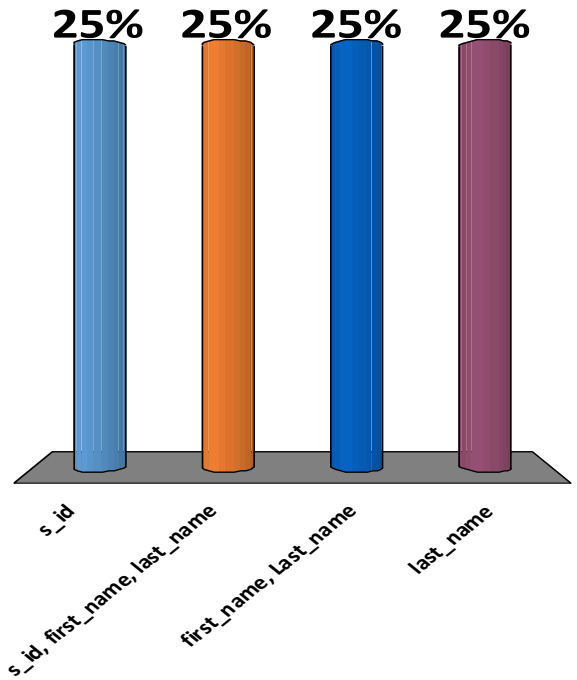
What is Student's primary key?

✓ A. s_id

B. s_id, first_name, last_name

C. first_name, Last_name

D. last_name



Student

s_id	first_name	last_name
S01	John	Smith
S02	Mary	Wallace
S03	Sue	Roper
S04	Mark	Jones

Response Counter

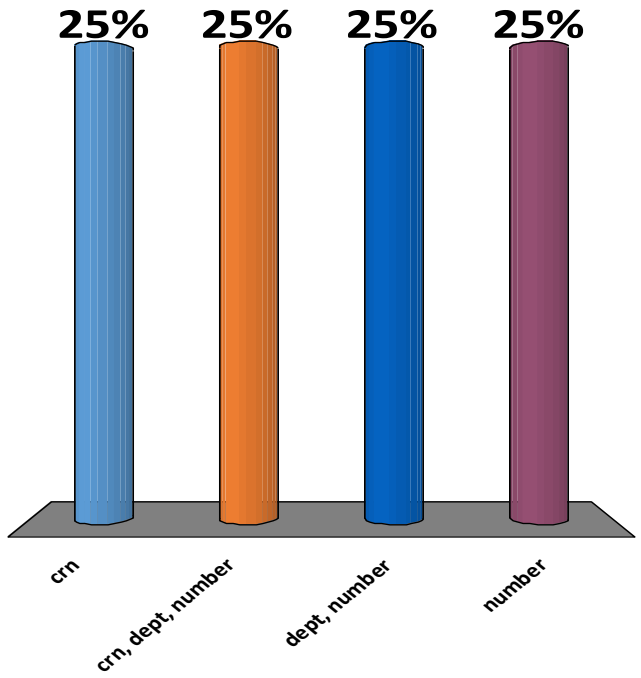
What is Course's primary key?

✓ A. crn

B. crn, dept, number

C. dept, number

D. number



Course

crn	dept	number
01234	COMP	140
15134	COMP	160
46117	ELEC	220

Response Counter

What is Enrollment's primary key?

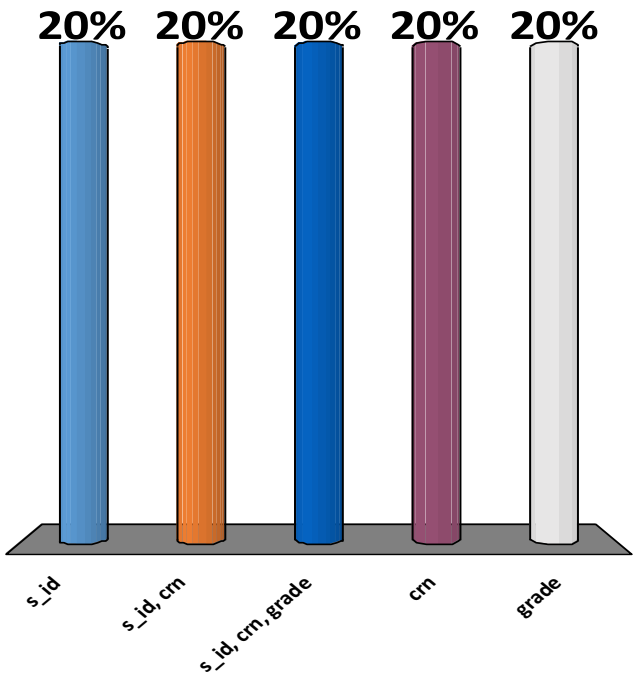
A. s_id

✓ B. s_id, crn

C. s_id, crn, grade

D. crn

E. grade



Enrollment

s_id	crn	grade
S01	01234	A
S01	46117	C
S02	01234	B

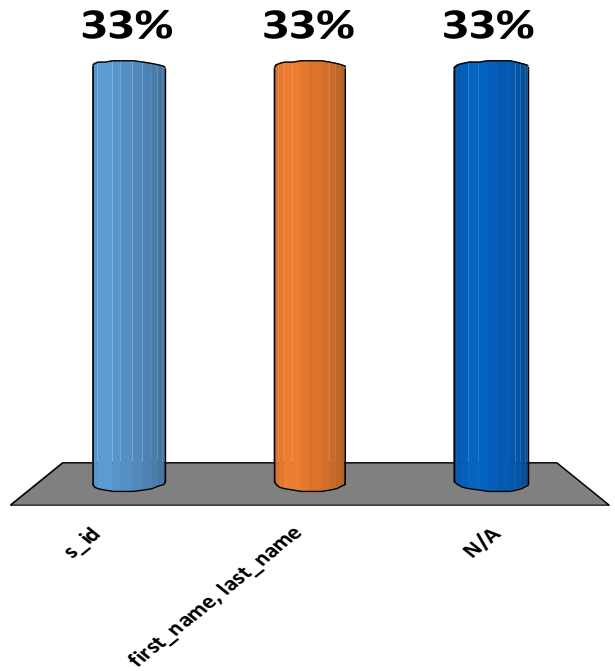
Response Counter

What is a foreign key in Student?

A. s_id

B. first_name, last_name

✓C. N/A



Student (s_id, first_name, last_name)

Enrollment (s_id, crn, grade)

Course (crn, dept, number)

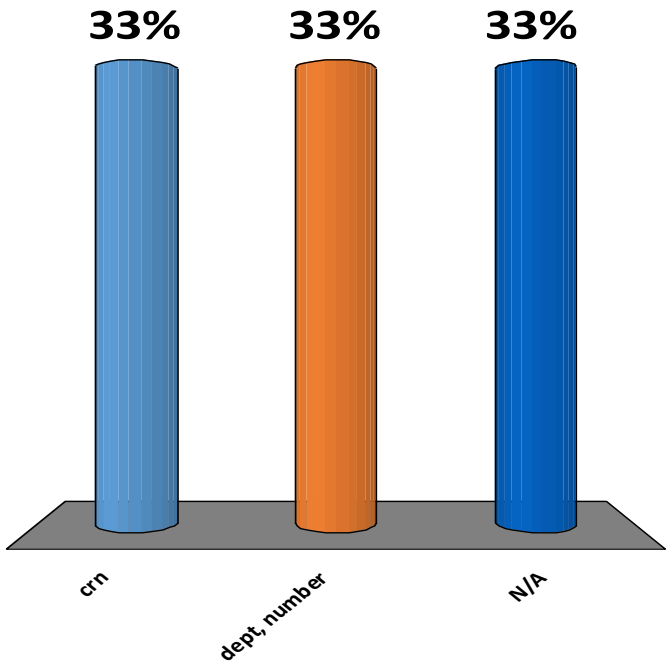
Response Counter

What is a foreign key in Course?

A. crn

B. dept, number

✓C. N/A



Student (s_id, first_name, last_name)

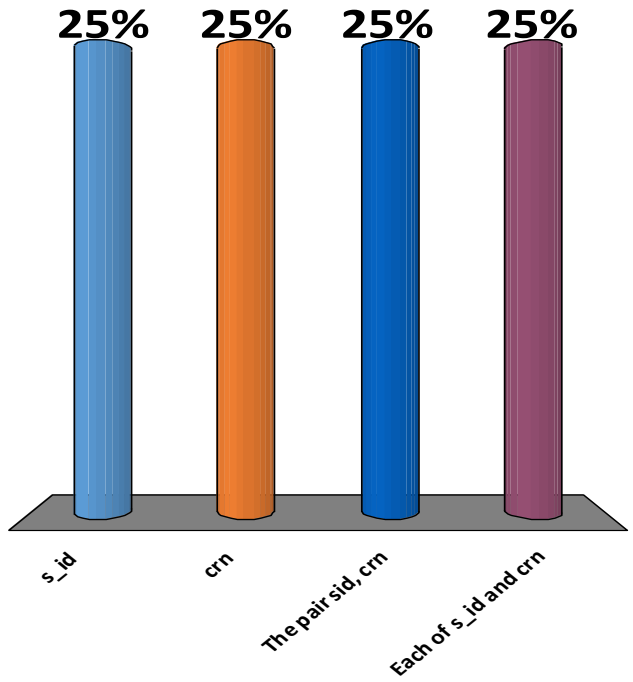
Enrollment (s_id, crn, grade)

Course (crn, dept, number)

Response Counter

What is a foreign key in Enrollment?

A. s_id
B. crn
C. The pair sid, crn
✓ D. Each of s_id and crn



Student (s_id, first_name, last_name)

Enrollment (s_id, crn, grade)

Course (crn, dept, number)

Creating a table with a foreign key

```
CREATE TABLE Student (
    s_id CHAR(10),
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    PRIMARY KEY (s_id)
);
```

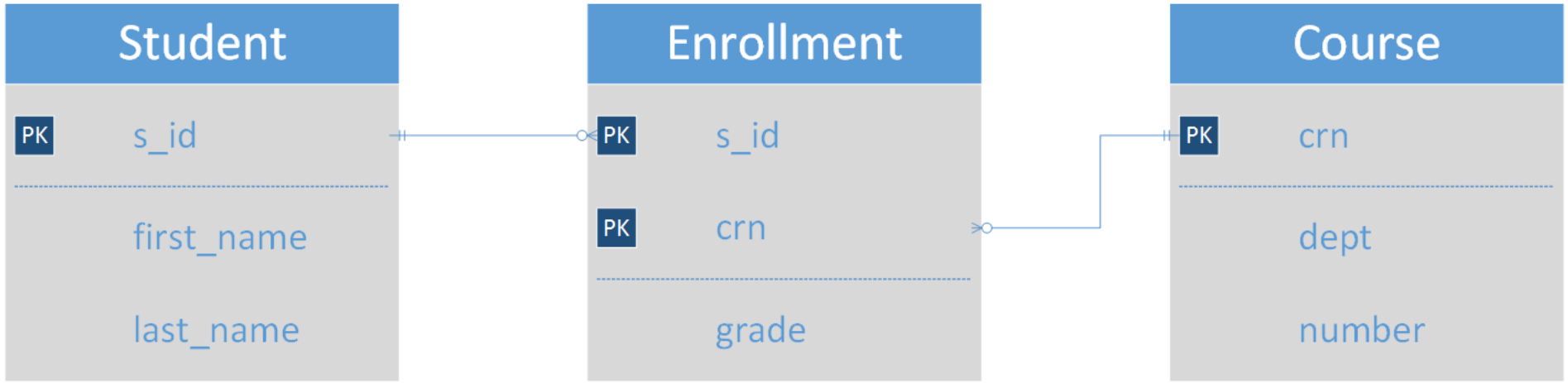
```
CREATE TABLE Course (
    crn CHAR(10),
    dept CHAR(4),
    number CHAR(3),
    PRIMARY KEY (crn)
);
```

```
CREATE TABLE Enrollment (
    s_id CHAR(10),
    crn CHAR(10),
    grade CHAR(2),
    PRIMARY KEY (s_id, crn),
    FOREIGN KEY (s_id) REFERENCES Student (s_id),
    FOREIGN KEY (crn) REFERENCES Course (crn)
);
```

Another representation – ER diagram

Style: Crow's foot

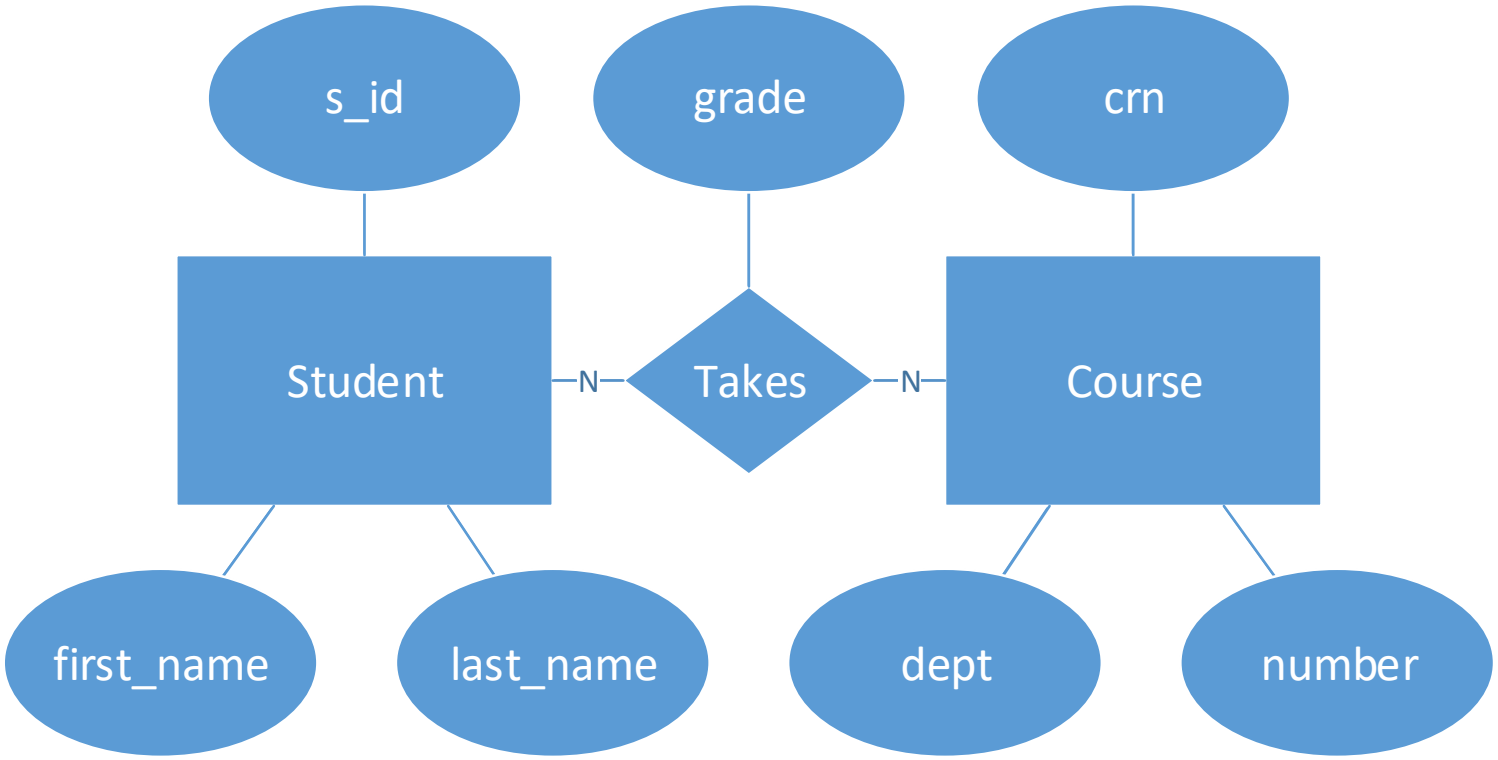
Conceptual level: Physical



A more abstract ER diagram

Style: Chen

Conceptual level: Logical



Joining tables – inner joins

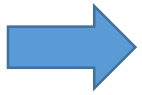
Product

p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper

Company

c_name	address	city	state
GizmoWorks	123 Gizmo St.	Houston	TX
WidgetsRUs	20 Main St.	New York	NY
Hyper	1 Mission Dr.	San Francisco	CA

By far, the most common kind of join. See other kinds later.



p_name	price	manufacturer	c_name	address	city	state
Gizmo	19.99	GizmoWorks	GizmoWorks	123 Gizmo St.	Houston	TX
Powergizmo	39.99	GizmoWorks	GizmoWorks	123 Gizmo St.	Houston	TX
Widget	19.99	WidgetsRUs	WidgetsRUs	20 Main St.	New York	NY
HyperWidget	203.99	Hyper	Hyper	1 Mission Dr.	San Francisco	CA

Joining tables – INNER JOIN syntax

Product

p_name	price	manufacturer

Company

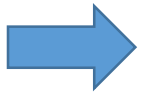
c_name	address	city	state

```
SELECT *
FROM Product
INNER JOIN Company ON manufacturer = c_name;
```

~~SELECT *~~
~~FROM Product, Company~~
~~WHERE manufacturer = c_name;~~

Old style – deprecated & error-prone

```
SELECT *
FROM Company
INNER JOIN Product ON manufacturer = c_name;
```



p_name	price	manufacturer	c_name	address	city	state
Gizmo	19.99	GizmoWorks	GizmoWorks	123 Gizmo St.	Houston	TX
Powergizmo	39.99	GizmoWorks	GizmoWorks	123 Gizmo St.	Houston	TX
Widget	19.99	WidgetsRUs	WidgetsRUs	20 Main St.	New York	NY
HyperWidget	203.99	Hyper	Hyper	1 Mission Dr.	San Francisco	CA

Resolving attribute name conflicts

Person (name, address, works_for)
Company (name, address)

```
SELECT Person.name, Person.address
FROM Person
INNER JOIN Company ON Person.works_for = Company.name;
```

```
SELECT p.name, p.address
FROM Person p
INNER JOIN Company c ON p.works_for = c.name;
```

Selection & Projection

```
SELECT p_name
FROM Product
INNER JOIN Company ON manufacturer = c_name
WHERE state = 'TX';
```

Join condition

Filter condition

1) Join

p_name	price	manufacturer	c_name	address	city	state
Gizmo	19.99	GizmoWorks	GizmoWorks	123 Gizmo St.	Houston	TX
Powergizmo	39.99	GizmoWorks	GizmoWorks	123 Gizmo St.	Houston	TX
Widget	19.99	WidgetsRUs	WidgetsRUs	20 Main St.	New York	NY
HyperWidget	203.99	Hyper	Hyper	1 Mission Dr.	San Francisco	CA

Selection & Projection

```
SELECT p_name
FROM Product
INNER JOIN Company ON manufacturer = c_name
WHERE state = 'TX';
```

Join condition

Filter condition

2) Selection

p_name	price	manufacturer	c_name	address	city	state
Gizmo	19.99	GizmoWorks	GizmoWorks	123 Gizmo St.	Houston	TX
Powergizmo	39.99	GizmoWorks	GizmoWorks	123 Gizmo St.	Houston	TX

Selection & Projection

```
SELECT p_name
FROM Product
INNER JOIN Company ON manufacturer = c_name
WHERE state = 'TX';
```

Join condition

Filter condition

3) Projection

p_name
Gizmo
Powergizmo

Semantics – set notation

SELECT [DISTINCT] $T_0.a_1, T_0.a_2, \ldots, T_n.a_m$
FROM T_0
INNER JOIN T_1 ON $JoinCondition_1$
…
INNER JOIN T_n ON $JoinCondition_n$
WHERE FilterCondition;

$\{(T_0.a_1, T_0.a_2, \ldots, T_n.a_m) \mid$
$JoinCondition_1$ AND
…
$JoinCondition_n$ AND
FilterCondition}

Multisets by default.
Sets with DISTINCT.

Activity: Writing multi-table queries

02c-queries.ipynb

Add, modify, delete data

Data not static

Databases are built to be used.

Forgetting this is a common student mistake.

Adding data

```
INSERT INTO Student VALUES
   ('S123456789', 'John', 'Doe', 'Houston', 'TX');

INSERT INTO Student VALUES
   ('S111111111', 'Mary', 'Jones', 'Houston', 'TX'),
   ('S222222222', 'Joe", 'Wallis', 'Kansas City', 'KS');

INSERT INTO Student (id, first_name, last_name) VALUES
   ('S987654321', 'Jane', 'Smith');
```

Saving query results into a table

```
INSERT INTO Student
SELECT …;

INSERT INTO Student (id, first_name, last_name)
SELECT …;
```

Existing table

```
SELECT …
INTO    new_table_name
…;
```

New table

Modifying & deleting data

```
UPDATE Student
SET city = ‘Austin’, state = ‘TX’
WHERE id = ‘S123456789’;
```

```
DELETE FROM Student;

DELETE FROM Student
WHERE …;
```

Foreign keys + DELETE: What's the issue?

Student (id, first_name, last_name, city, state)
Enrollment (student_id, crn)
Course (crn, dept_code, number, title)

```
DELETE FROM Student
WHERE id = 'S123456789';
```

Modifying & deleting data affects other tables

Student (id, first_name, last_name, city, state)
Enrollment (student_id, crn)
Course (crn, dept_code, number, title)

Need to maintain referential integrity!

```
DELETE FROM Student
WHERE id = 'S123456789';
```

By default, UPDATE & DELETE fail when they would break referential integrity.

Modifying & deleting data affects other tables

Student (id, first_name, last_name, city, state)
Enrollment (student_id, crn)
Course (crn, dept_code, number, title)

```
CREATE TABLE Enrollment (
    …
    FOREIGN KEY crn REFERENCES Course (crn)
    ON DELETE …,
    FOREIGN KEY student_id REFERENCES Student (id)
    ON UPDATE …
);
```

- **RESTRICT**
- **CASCADE**

- NO ACTION
- SET NULL
- SET DEFAULT

Modifying data & natural vs. synthetic keys

Natural	You can get a new SSN in cases of stolen identity.
	Account numbers can change when banks merge.
	Fits with ON UPDATE CASCADE.
Synthetic	Should never change.
	Fits with ON UPDATE RESTRICT / NO ACTION.

Usually don’t want to DELETE!

Soft delete

- I might want this data later, after all.
- Delete = Flag this data as “inactive”.
- E.g., closed customer account or discontinued product.
- Fits with ON DELETE RESTRICT / NO ACTION.

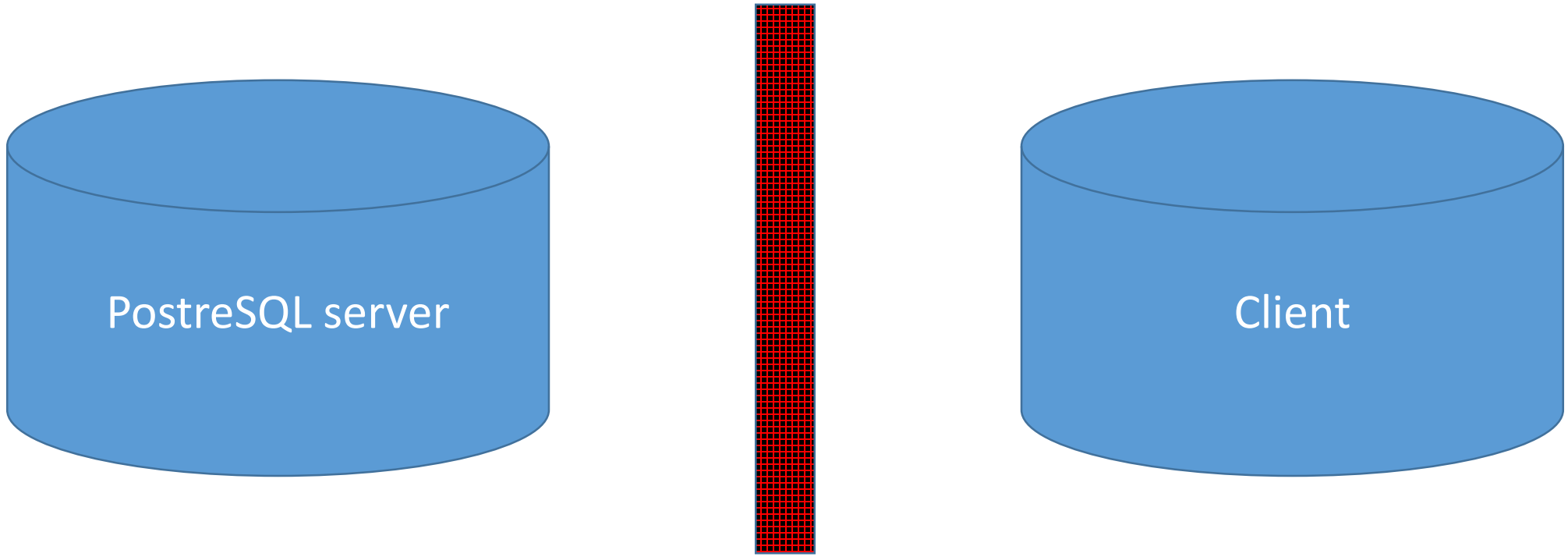
Hard delete

- I want to delete the data, so dang it, delete it!
- Fits with ON DELETE CASCADE.

Getting data from/to a file

Not a trivial thing!

- File on client or server?
- Security issues



Adding records to a table from a file

PostgreSQL

Server file: `COPY Product FROM '/my/path/product.csv' CSV HEADER;`

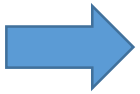
Must be logged in as a superuser, e.g., postgres. Server app must have file permission.

Client file: `\copy Product FROM '/my/path/product.csv' CSV HEADER`

Must use psql.

p_name	price	manufacturer

```
p_name, price, manufacturer
Gizmo,19.99,GizmoWorks
Powergizmo,39.99,GizmoWorks
Widget,19.99,WidgetsRUs
HyperWidget,203.99,Hyper
```



p_name	price	manufacturer
Gizmo	19.99	GizmoWorks
Powergizmo	39.99	GizmoWorks
Widget	19.99	WidgetsRUs
HyperWidget	203.99	Hyper

Add, modify, delete from schema

Database schemas not static

Needs change over time.

Think about long-term plans & maintenance.

Add, modify, delete

Tables

```
CREATE TABLE Student (…);
```

```
ALTER TABLE Student
RENAME TO Scholar;
```

```
DROP TABLE Student;
```

Attributes

```
ALTER TABLE Student
ADD COLUMN zip INT;
```

```
ALTER TABLE Student
RENAME COLUMN zip TO zip_code;
```

```
ALTER TABLE Student
ALTER COLUMN zip VARCHAR(9);
```

```
ALTER TABLE Student
DROP COLUMN state;
```

Constraints

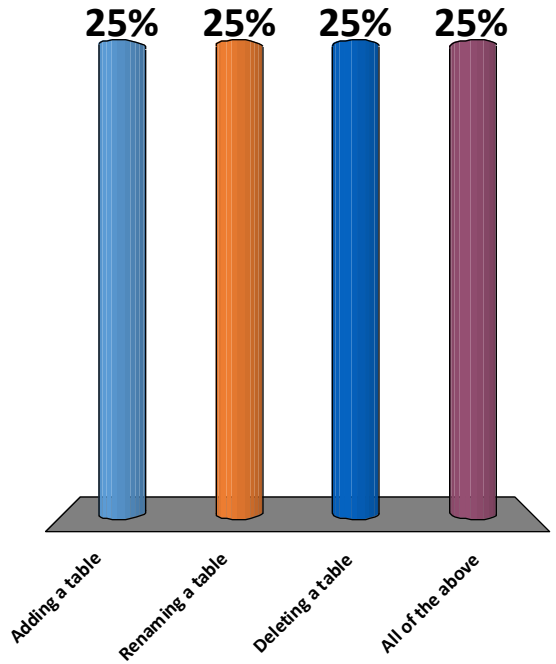
Similar. Later.

Response Counter

Which can change existing query semantics?

A. Adding a table
B. Renaming a table
C. Deleting a table
✓ D. All of the above

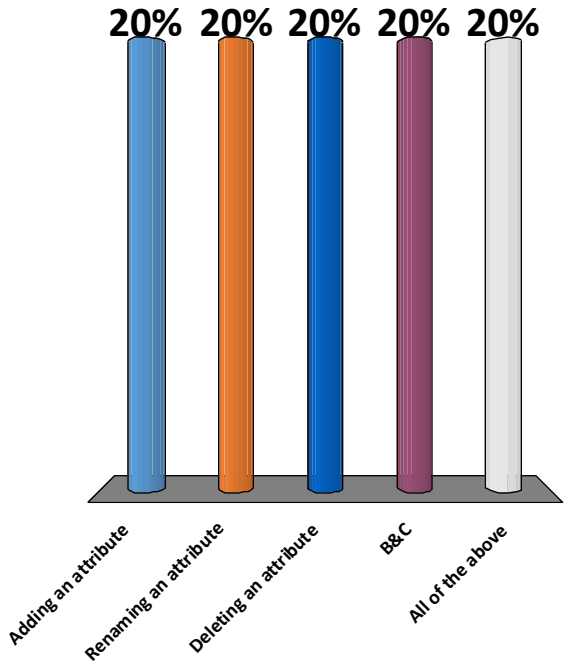
Should be obvious.



Response Counter

Which can change existing query semantics?

A. Adding an attribute
B. Renaming an attribute
C. Deleting an attribute
D. B&C
✓ E. All of the above




```
SELECT *
FROM …
…;
```

Dangerous – avoid SELECT *!

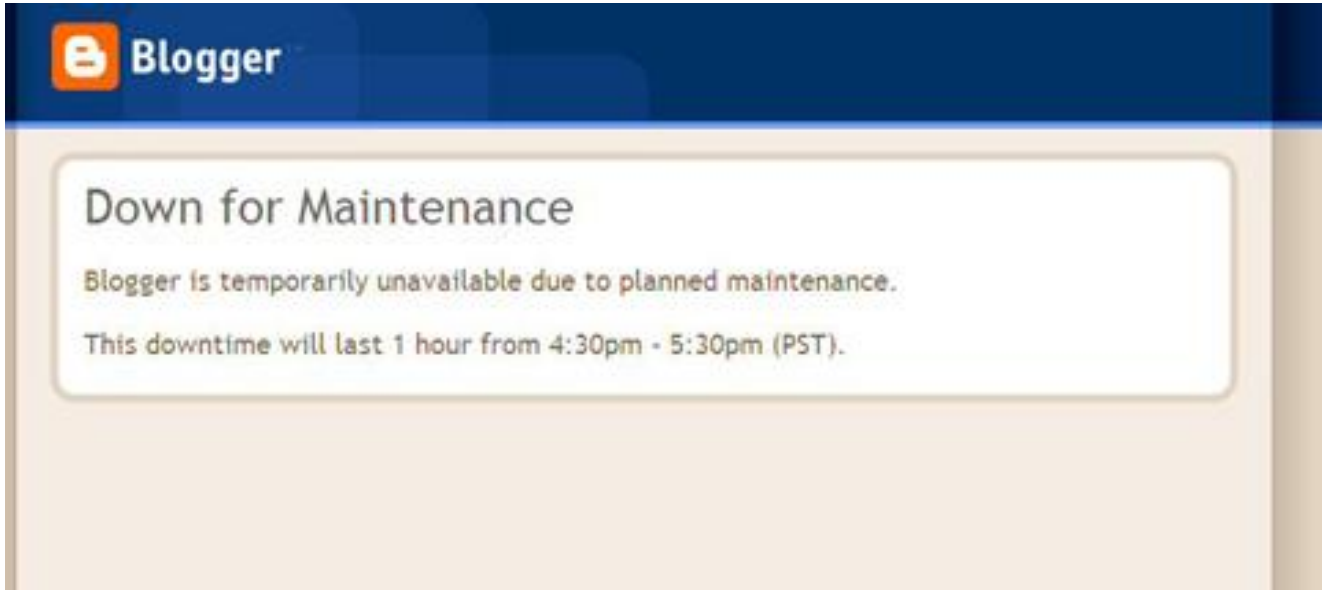
Specify columns explicitly.

Process for changing schemas

Traditional – Regular shutdowns

1. Make & test changes on a test server.
2. Have regularly scheduled system shutdown.
   a) Typically late at night to minimize user disruption.
   b) Disable user access to live server.
   c) Make & test changes on the live server.

Takes time. Lots of data, possibly replicated & distributed.



Reducing downtime

Instead of a full shutdown, just block users from DB temporarily.

- Shutdown only as long as needed.
- “Lock” the DB.

Only need to lock tables that are changing.

- See COMP 322 & 421 for more about locks.

For UX, assumes lock is only needed briefly.

Updating without any user disruption

E.g.:

- Create “shadow table” incorporating changes
- Create “triggers” in original table that forward data updates to shadow table
- Copy original table’s data to shadow table
- Rename shadow table to replace original table

Can choose to only let some users see the changes.

Some DBMS provide support for such “online” schema changes.

Structured Query Language (SQL) Overview

Syntax:	Mostly declarative
	Old-fashioned, COBOL-based
	Numerous “standards”: ANSI SQL, SQL92, SQL99
	Plus vendor-specific extensions & subsets
Semantics:	Mostly straight-forward
	Occasionally confusing
Implementation:	Highly optimized, parallelized
	Occasionally needs hints

- Data Definition Language (DDL)
  - Define relational *schemas* — declarative
  - Create/modify/delete *tables* and their *attributes* — imperative
- Data Manipulation Language (DML)
  - *Query* DB for desired data — declarative
  - Add/modify/delete data in tables — imperative
  - Define/use functions, procedures, triggers — imperative
- Data Control Language (DCL)
  - Control user priviledges — imperative
- Transaction Control Language (TCL)
  - Manage changes to data by DML — imperative

SQL

Procedural extensions to SQL