CS 61:
Database Systems

Multiple table CRUD

Agenda

1. Creating tables and their attributes
2. Inserting, deleting, and updating rows
3. Keys
4. Relational algebra part 2
5. Joins

SQL has several familiar data types we can use for attribute domains

Domain types

Domain type	Description
CHAR(n)	Fixed length character string, with user-specified length n, normally use varchar instead!
VARCHAR(n)	Variable length character strings, with user-specified maximum length n
SMALLINT	2-byte integer, max value 32,767
INT	4-byte integer, max value 2,147,483,647
BIGINT	8-byte integer, max value 9,223,372,036,854,775,807
NUMERIC(p,d) or DECIMAL(p,d)	Fixed point number, with user-specified precision of p total 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; truncate if too big)
REAL/DOUBLE PRECISION	Floating point and double-precision floating point numbers, max value 2.2250738585072014E- 308
FLOAT(n)	Floating point number, with user-specified precision of at least n digits, max value 1.175494351E-38
DATETIME	Format: YYYY-MM-DD HH:MM:SS

Create table SQL command sets up the schema for new relations

Create table

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

CREATE TABLE *r*

(A_1 D_1, A_2 D_2, ..., A_n D_n,
(integrity-constraint$_1$),
...,
(integrity-constraint$_k$))

Relation name r

Name/domain (data type) pairs, one for each attribute

Constrain the values an attribute can have, more on this soon!

- Example:

CREATE TABLE *instructor* (
ID **CHAR**(5),
name **VARCHAR**(20),
dept_name **VARCHAR**(20),
salary **NUMERIC**(8,2))

- Easier to create tables graphically with MySQL Workbench (but MySQL Workbench simply runs this commands for you)

Relations can be altered or deleted using DDL commands

Alter/delete relations and data

- **Delete Table**
  - **DROP TABLE** *r*

Delete relation r, both data and schema

- **Empty table**
  - **TRUNCATE TABLE** r

Delete data in relation r, but keep its schema

- **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

Add attribute A with domain D

  - **ALTER TABLE** *r* **DROP** *A*
    - where *A* is the name of an attribute of relation *r*
    - Dropping of attributes not supported by some databases

Delete attribute A from table r

Agenda

1. Creating tables and their attributes
2. Inserting, deleting, and updating rows
3. Keys
4. Relational algebra part 2
5. Joins

INSERT allows us to add new rows to a table

Insert: the C in CRUD

INSERT INTO table **VALUES** (v_1, v_2, …, v_n)

- v_1 … v_n must match order of attributes in table exactly
- Values for all attributes must be present

OR

INSERT INTO table (A_1, A_2, …, A_n) **VALUES** (v_1, v_2, ….v_n)

- v1 and A1 must match but can be in different order from table schema

Example: add a new department for database systems, building and budget are still to be determined

INSERT INTO department (dept_name)
VALUES ('Database Systems')

department table

dept_name	building	budget
Biology	Watson	90000.00
Comp. Sci.	Taylor	100000.00
Elec. Eng.	Taylor	85000.00
Finance	Painter	120000.00
History	Painter	50000.00
Music	Packard	80000.00
Physics	Watson	70000.00

dept_name	building	budget
Biology	Watson	90000.00
Comp. Sci.	Taylor	100000.00
Database Systems	NULL	NULL
Elec. Eng.	Taylor	85000.00
Finance	Painter	120000.00
History	Painter	50000.00
Music	Packard	80000.00
Physics	Watson	70000.00

We can also INSERT into a table using a SELECT nested query

Insert: the C in CRUD

INSERT INTO table (A_1, A_2, …, A_n)
SELECT B_1, B_2, …, B_n
FROM other table
WHERE condition

B_1 … B_n domains must match A_1 … A_n

Example:

```
INSERT INTO biology_instructor (ID, `name`, dept_name, salary)
SELECT ID, name, dept_name, salary
FROM instructor
WHERE dept_name = 'Biology';
```

instructor table

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

biology_instructor table

ID	name	dept_name	salary
76766	Crick	Biology	72000.00

Assumes table called `biology_instructor` exists

We can also create a table using a SELECT nested query

Insert: the C in CRUD

INSERT INTO table (A_1, A_2, …, A_n)

SELECT B_1, B_2, …, B_n

FROM other table

WHERE condition

B_1 … B_n domains must match A_1 … A_n

Example:

CREATE TABLE biology_instructor

SELECT ID, name, dept_name, salary

FROM instructor

WHERE dept_name = 'Biology';

instructor table

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

Use CREATE TABLE make table and fill with subquery results

biology_instructor table

ID	name	dept_name	salary
76766	Crick	Biology	72000.00

UPDATE allows us to change rows in a table

Insert: the C in CRUD

UPDATE table **SET** $A_1=v_1$, $A_2=v_2$

WHERE P

Example: Give a 5% salary raise to instructors whose salary is less than average

```
UPDATE instructor
SET salary = salary * 1.05
WHERE salary < (SELECT AVG (salary)
                FROM instructor);
```

instructor table

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

Avg is 74,833.33

Updates:

- **Srinivasan**
- **Mozart**
- **El Said**
- **Califieri**
- **Crick**

Note: subquery in the WHERE clause

Practice: UPDATE

Insert: the C in CRUD

The restaurant_inspections tables has columns for latitude and longitude, most of the time these values are included, sometimes they are null or zero

1. Examine latitude attribute
    - Find how many restaurants have a NULL latitude and how many have a non-NULL latitude
    - Find how many have a zero for latitude

2. Update latitude and longitude to NULL if latitude is zero (assumes longitude is invalid too)

Delete removes rows from a table

Delete: the D in CRUD

DELETE FROM table
WHERE P

Example: Delete all tuples in the instructor relation instructors associated with a department located in the Watson building

DELETE FROM *instructor*
WHERE *dept name* **IN**
(**SELECT** *dept name*
FROM *department*
WHERE *building* = 'Watson');

Deletes:
Crick
Einstein
Gold

instructor table

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

department table

dept_name	building	budget
Biology	Watson	90000.00
Comp. Sci.	Taylor	100000.00
Elec. Eng.	Taylor	85000.00
Finance	Painter	120000.00
History	Painter	50000.00
Music	Packard	80000.00
Physics	Watson	70000.00

Practice: DELETE

Delete: the D in CRUD

The restaurant_inspections table has rows where the restaurant name (dba) is NULL

1. Find out how many restaurants have NULL for dba
2. Delete those restaurants
3. Confirm those restaurants have been deleted

Agenda

1. Creating tables and their attributes
2. Inserting, deleting, and updating rows
3. Keys
4. Relational algebra part 2
5. Joins

Some thoughts on same conventions

instructor table

ID	*name*	*dept_name*	*salary*
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

department table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

My preference: use TableNameID (e.g., InstructorID) not just ID

Can be confusing when combining multiple tables if just use ID

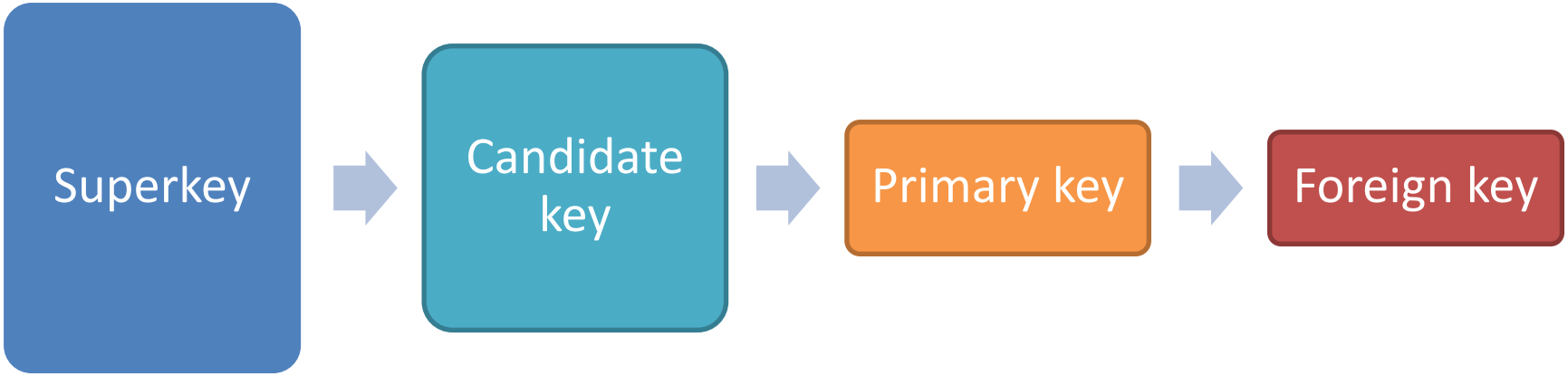
I also prefer:

- **Capital first letter then lower case, with capital letter for other words (DepartmentName) for table and attribute names**
- **Spell out name (e.g., "Section" not "sec"), can be confusing later, does "sec" mean security or section?**
- **Other people disagree! YMMV**

Keys uniquely identify table rows (tuples) based on their attributes

- Keys uniquely identify table rows and can be comprised of multiple attributes
- Let K $\subseteq$ R (R is the set of attributes in relation r, K is a subset of 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*
  - More formally: if t_1 and t_2 are tuples in r, and t1 ≠ t2, then t1.K ≠ t2.K
- If K is a superkey, then so is any superset of K
- Superkey *K* is a **candidate key** if *K* is minimal (no subset of K is also a superkey)
  - Example: {*ID*} is a candidate key for *Instructor, {ID, name}* is not
- Database designer chooses a candidate key to be the **primary key (PK)**
  - Must choose wisely (two instructors could have the same name, so use ID)
  - Choose primary keys based on attributes that rarely change
- Typically list primary key attributes first in relation schema and underline
  - Example: classroom(<u>building</u>, <u>room_number</u>, capacity)
  - Classroom primary key is comprised of building and room number

Key summary



- Uniquely identifies a row
- Can have more attributes than necessary to identify row

- Superkey with minimal number of attributes
- Can be more than one candidate key for a relation

- Candidate key chosen to identify each row

- Values in one table must match primary key in another table

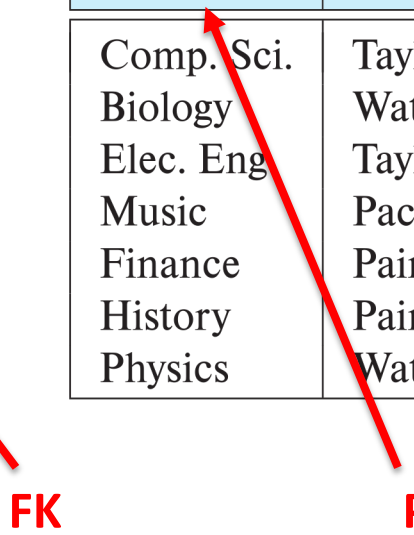
Foreign keys constrain attribute values to primary keys of another relation

instructor table

ID	*name*	*dept_name*	*salary*
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

department table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000



FK in one relation is PK in another

Foreign key (FK) constraint: attribute A for each tuple of relation r_1 (dept_name in instructor) must contain the value of the primary key of some tuple in relation r_2 (dept_name in department).

Referential integrity constraint: value of attribute must be the value of any tuple's attribute of another relation (not *necessarily* PK, but is in practice)

Integrity constrains ensure attributes have values we expect; set when creating table

Integrity constraints

- Some integrity constraints
  - **PRIMARY KEY**(A_1, ..., A_n)
  - **FOREIGN KEY**(A_i ...,A_j) **REFERENCES** r(A_k ...,A_l)
  - **NOT NULL**
- SQL prevents any update to the database that violates an integrity constraint

Must be non-null and unique for each tuple (no duplicates)

Attribute value must be a primary key in relation *r*

Attribute cannot be null

Can use auto_increment to create an increasing ID if numeric (BIGINT)

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(dept_name));
```

Instructor's *dept_name* must be value of a primary key in *department* table

***name* can't be null**

Instructor is uniquely identified by primary key *ID*

If update violates any constraint, SQL will reject command

Integrity constrains ensure attributes have values we expect

Integrity constraints

- **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 (ID),*
  foreign key (*course_id, sec_id, semester, year*) **references** *section(course_id, sec_id,semester,year)*);

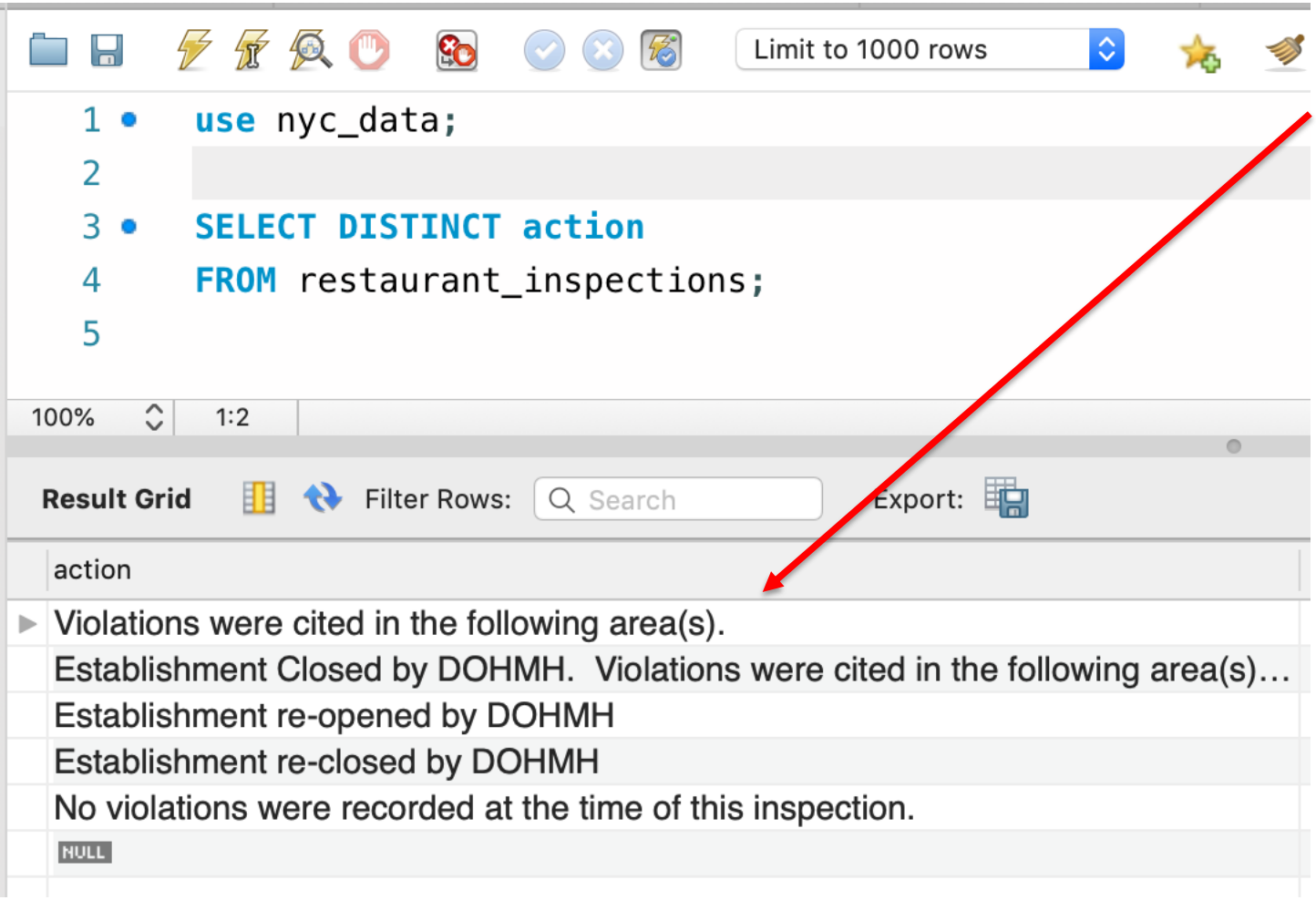
Composite primary key (made of multiple attributes)

SQL will reject if integrity constraints are not met

Value for foreign key attributes must be in a tuple in the *section* relation

We can create and populate tables using one statement

Limited number of actions in restaurant_inspections



There are only fives types of actions recorded over all inspections

Instead of storing the text for each action, we can create a table for Actions with an ID for each action and a description

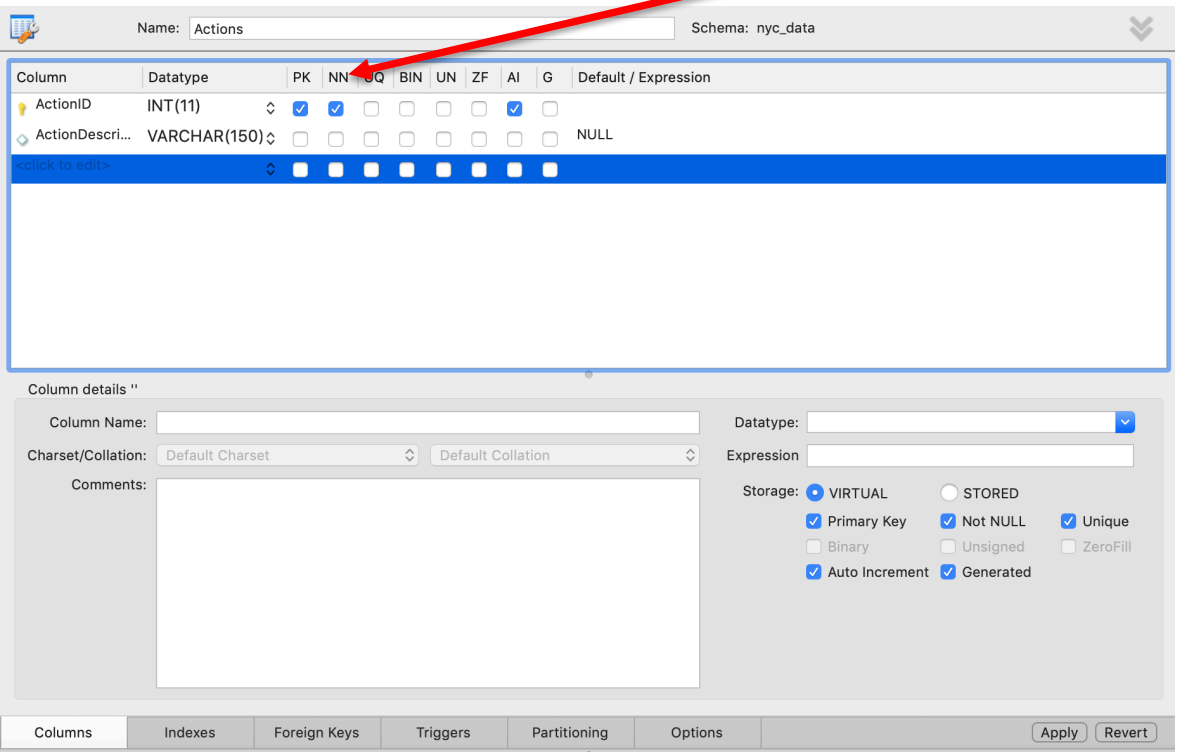
In the inspection table we can use the action ID as a foreign key

We can create and populate tables using one statement

CREATE TABLE Actions (ActionID **INT NOT NULL AUTO_INCREMENT**,
ActionDescription **VARCHAR(150)**,
PRIMARY KEY (ActionID))

SELECT DISTINCT Action **AS** ActionDescription
FROM restaurant_inspections
WHERE Action is not null;

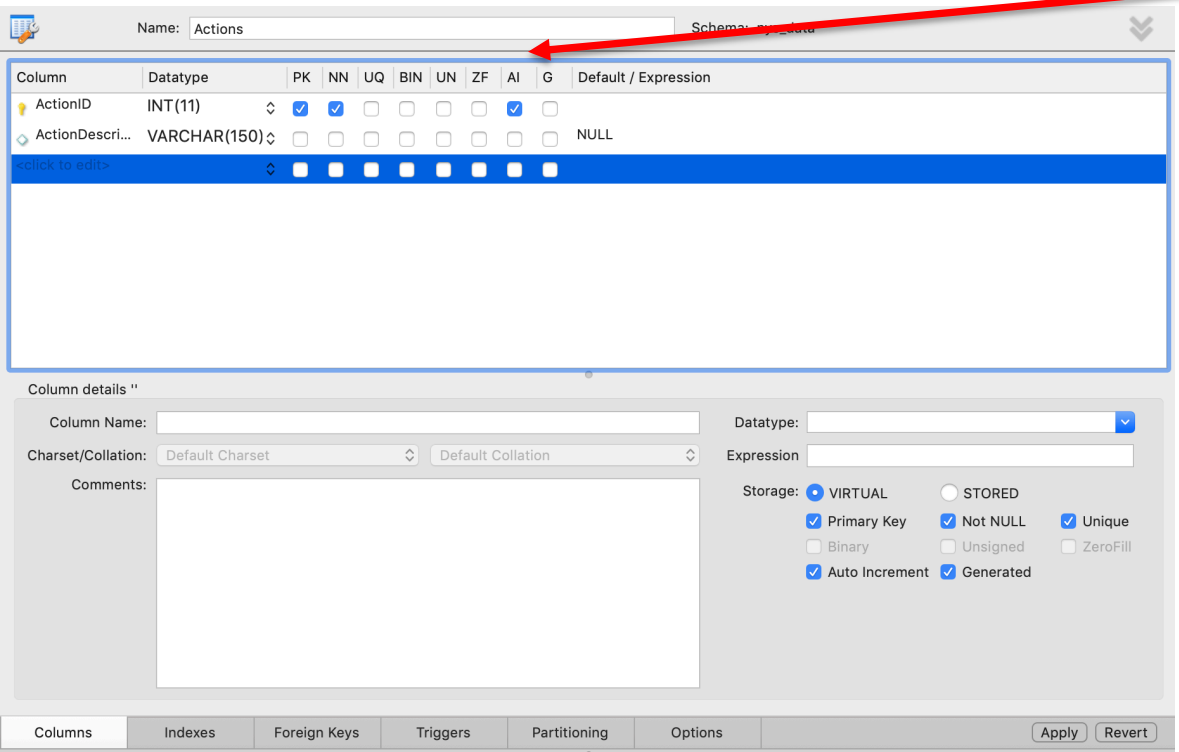
ActionID is primary key (PK), so it must be non-null (NN)



We can create and populate tables using one statement

CREATE TABLE Actions (ActionID **INT NOT NULL AUTO_INCREMENT**,
ActionDescription **VARCHAR(150)**,
PRIMARY KEY (ActionID))

SELECT DISTINCT Action **AS** ActionDescription
FROM restaurant_inspections
WHERE Action is not null;



ActionID is primary key (PK), so it must be non-null (NN)

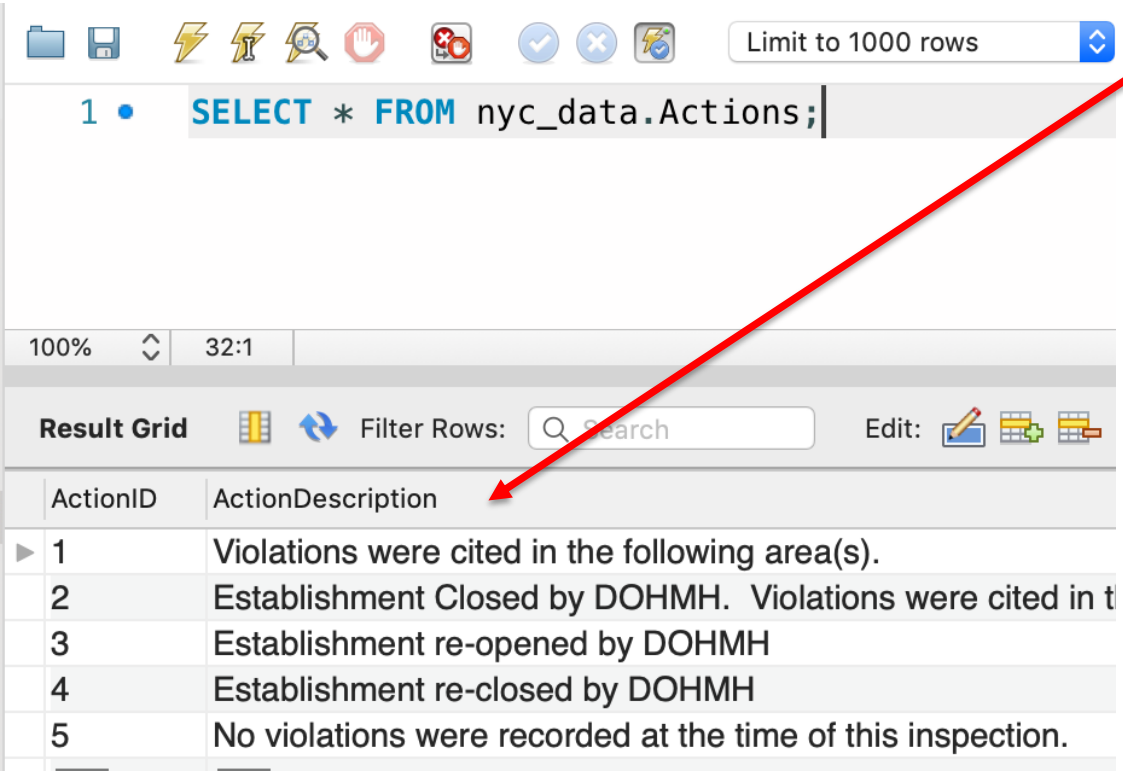
Can use auto_increment to create a unique increasing integer ID for each entry

We can create and populate tables using one statement

CREATE TABLE Actions (ActionID **INT NOT NULL AUTO_INCREMENT**,
ActionDescription **VARCHAR(150)**,
PRIMARY KEY (ActionID))

SELECT DISTINCT Action **AS** ActionDescription
FROM restaurant_inspections
WHERE Action is not null;

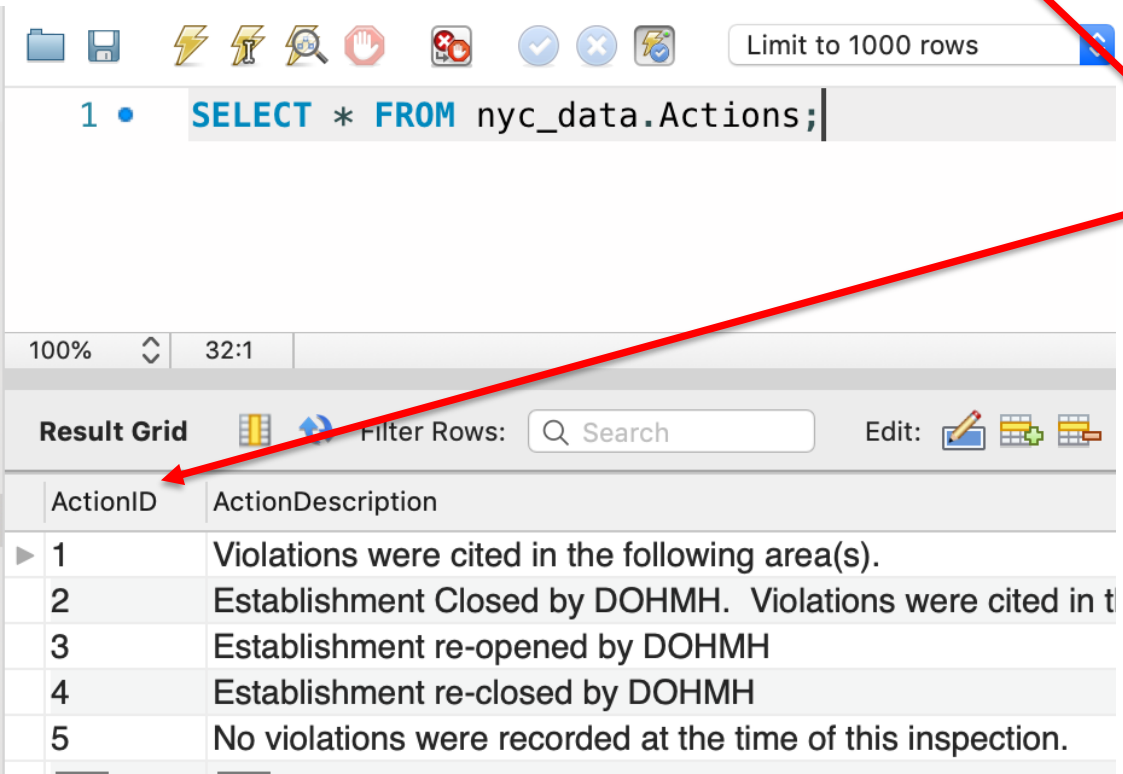
Select clause fills new table with data from restaurant_inspections



We can create and populate tables using one statement

CREATE TABLE Actions (ActionID **INT NOT NULL AUTO_INCREMENT**,
ActionDescription **VARCHAR(150)**,
PRIMARY KEY (ActionID))

SELECT DISTINCT Action **AS** ActionDescription
FROM restaurant_inspections
WHERE Action is not null;



SELECT clause fills new table with data from restaurant_inspections

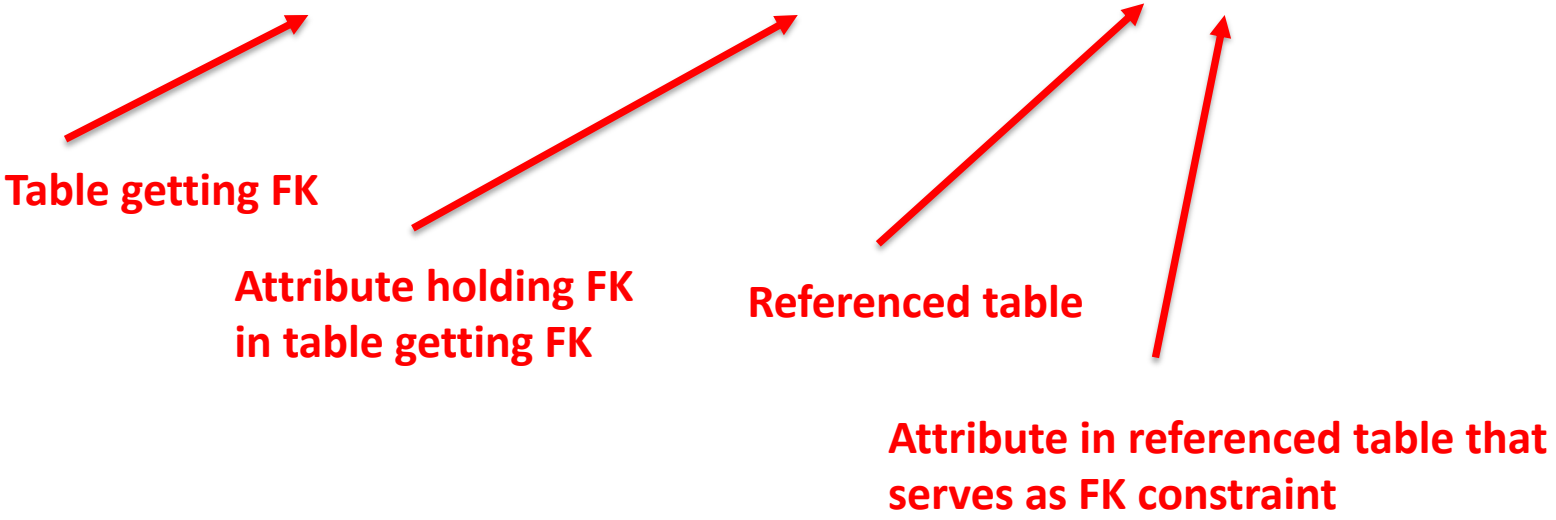
Auto_increment fills ActionID with increasing integer values for us

Notice we did not specify ActionID in the SELECT clause, MySQL filled it for us

Add a foreign key constraint to an existing table with the ALTER TABLE command

Create a foreign key constraint

- **Add foreign key constraint**
  - ALTER TABLE r_1 ADD FOREIGN KEY (A_1) REFERENCES r_2(A_2);



Agenda

1. Creating tables and their attributes
2. Inserting, deleting, and updating rows
3. Keys
4. Relational algebra part 2
5. Joins

I'll use the textbook's instructor and teaches tables

instructor table

ID	*name*	*dept_name*	*salary*
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

teaches table

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-315	1	Spring	2018
45565	CS-101	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

Teaches table lists courses and sections that are taught by instructors

Cartesian Product: combines every pair of tuples from two different relations

Cartesian Product: r X s

r → instructor X teaches ← *s*

Each tuple from *instructor* matched with each tuple from *teaches*

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018

Result has attributes from both relations

Note: ID appears in both *instructor* and *teaches* table, some systems prefix with table name

This is probably <u>not</u> what we want!

Most rows about an instructor who did NOT teach a course

Combine Cartesian product with SELECT to produce a JOIN operation

Join operation

$$\sigma_{instructor.id = teaches.id}(instructor \times teaches))$$

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017
76766	Crick	Biology	72000	76766	BIO-301	1	Summer	2018
83821	Brandt	Comp. Sci.	92000	83821	CS-190	1	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-190	2	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-319	2	Spring	2018
98345	Kim	Elec. Eng.	80000	98345	EE-181	1	Spring	2017

Now we get courses taught by instructors

Attributes from both relations combined into a new relation

JOIN: returns attributes from r and s where attributes in predicate θ match

Join notation: r $\bowtie_\theta$ s

Given relations *r* (*R*) and *s* (*S*)

Let "theta" be a predicate on attributes R "union" S

The join operation r $\bowtie_\theta$ s is defined as $r \bowtie_\theta s = \sigma_\theta (r \times s)$

θ = instructor.id=teaches.id

$\sigma_{instructor.id = teaches.id}$ (*instructor* x *teaches*))

Same as: *instructor* $\bowtie_{Instructor.id = teaches.id}$ *teaches*

Same procedure, just different notation

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017

Agenda

1. Creating tables and their attributes
2. Inserting, deleting, and updating rows
3. Keys
4. Relational algebra part 2
5. Joins

JOIN tables in FROM clause using predicate in WHERE, return attributes in SELECT

Join tables

$\Pi_{\text{name, course\_id}}$ (
instructor $\bowtie_{\textit{Instructor.id = teaches.id}}$ *teaches)*

SELECT *name, course_id*
FROM *instructor , teaches*
WHERE *instructor.ID = teaches.ID*

Conceptual sequence of events

1. Perform Cartesian product over all relations in **FROM** clause
   - Result is Cartesian product like in slide 29
   - If three tables, number of tuples = $|t_1| * |t_2| * |t_3|$,
     where $|x|$ = number of tuples in table x
   - This result is not particularly useful
   - Real databases do not actually go to this trouble (too time consuming)
2. Apply predicates in **WHERE** clause to result from step 1 (gives rows wanted)
3. Project attributes from **SELECT** clause (gives columns wanted)

Can use aliases for table and attribute names

Joins with alias and 'and' in where

Find the names of all instructors in the Finance department and the courses they have taught

```
SELECT name, course_id AS `course number`
FROM instructor i, teaches t
WHERE i.ID = t.ID
      AND  i.dept_name = 'Finance'
```

Can alias table names

Could have said '*from instructor as i*', but '*as*' is not required here

This is an "old style" join, next class we will look at another method

Can now reference table and attribute names by alias

Attribute will be called '*course number*' instead of '*course_id*' thanks to AS keyword

Use backtick (single quote character near the 1 key on your keyboard) for multiple word names `course number`

Practice

Rows in restaurant_inspections table are inspections of restaurants and each restaurant may have been inspected multiple times

1. Create and populate a table called Restaurants from restaurant_inspections with one row for each *distinct* restaurant inspected with these attributes: RestaurantID, RestaurantName, Building, Street, Boro, and CuisineID (CuisineID initially NULL)
   - What did you choose for the primary key (Hint: no need for auto_increment)
   - Will this table have the same number of rows as restaurant_inspections?
   - How many rows did yours have?
2. Create a table called Cuisine that holds each of the different types of cuisine restaurants may have
   - Decide on a primary key
   - Populate the table with distinct cuisine types from restaurant_inspections
3. Add a foreign key constraint to the Restaurants table that references the CuisineID attribute in the Cuisine table
   - Try to assign a CuisineID to a restuarant where the cuisine does not exist
4. Run my script “day4_create_nyc_inspections_schema.sql” before next class to create a new schema with several tables based on restaurant_inspections