CS 61:
Database Systems

CRUD

Adapted from Silberschatz, Korth, and Sundarshan unless otherwise noted
Augmented by Claude Opus

CRUD

CRUD	SQL	Notes
Create	INSERT	Add new entries
Read	SELECT	Retrieve entries
Update	UPDATE	Adjust entries
Delete	DELETE	Remove entries

Agenda

1. Creating tables and their attributes
2. Inserting, deleting, and updating rows
3. Keys
4. Integrity constraints

Review: Data is held is relations (tables)

College schema

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

Each relation has attributes (fields)
Each field has a data type:

- **VARCHAR(n)**
- **INT**
- **FLOAT**
- **DATETIME**

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

You can create your own data types (out of scope for CS61)

How can you create a Boolean?

Use BOOLEAN, but it is a TINYINT (1 byte INT)

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),
PRIMARY KEY (`ID`);

Primary Key tells how to uniquely identify rows

- Easier to create tables graphically with MySQL Workbench (but MySQL Workbench simply runs this command 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

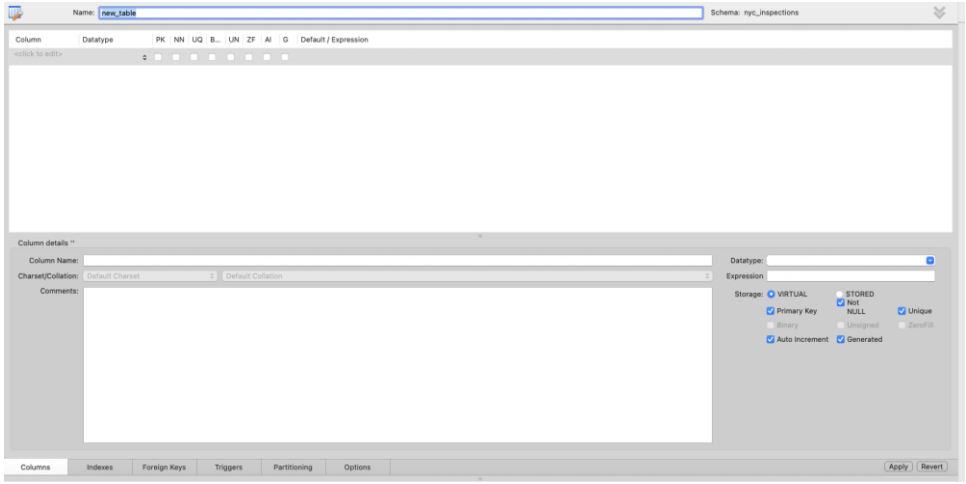
- **Alter**
  - **ALTER TABLE** *r* **ADD** *A D*
    - Where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*
    - All exiting tuples in the relation are assigned *null* as the value for the new attribute
    - MySQL can add DEFAULT <val>

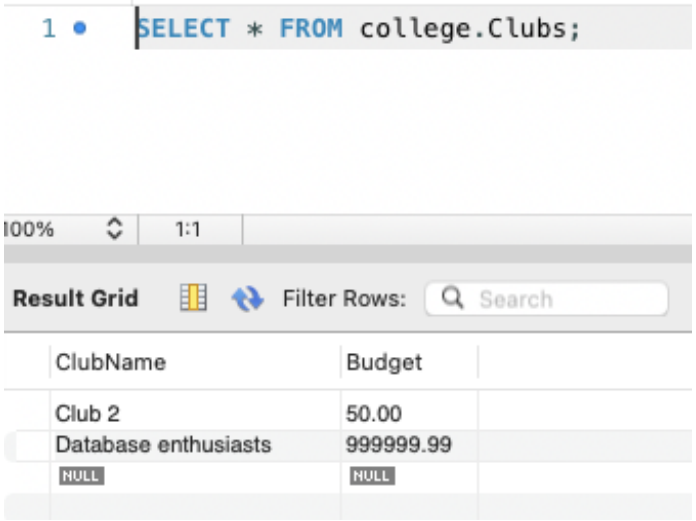
Add attribute A with domain D

  - **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

Practice 1: make a table in MySQL Workbench





1) Create a table for student clubs called Clubs
Click Create Table button
Add attributes:

- **ClubName (set to be Primary Key and Not Null)**
- **Budget (set 8 digits, with 2 decimal places, default to 0)**

Click Apply to create table
MySQL will show you the SQL it issues

2) Add some clubs using Result Grid

Click Apply after entering rows

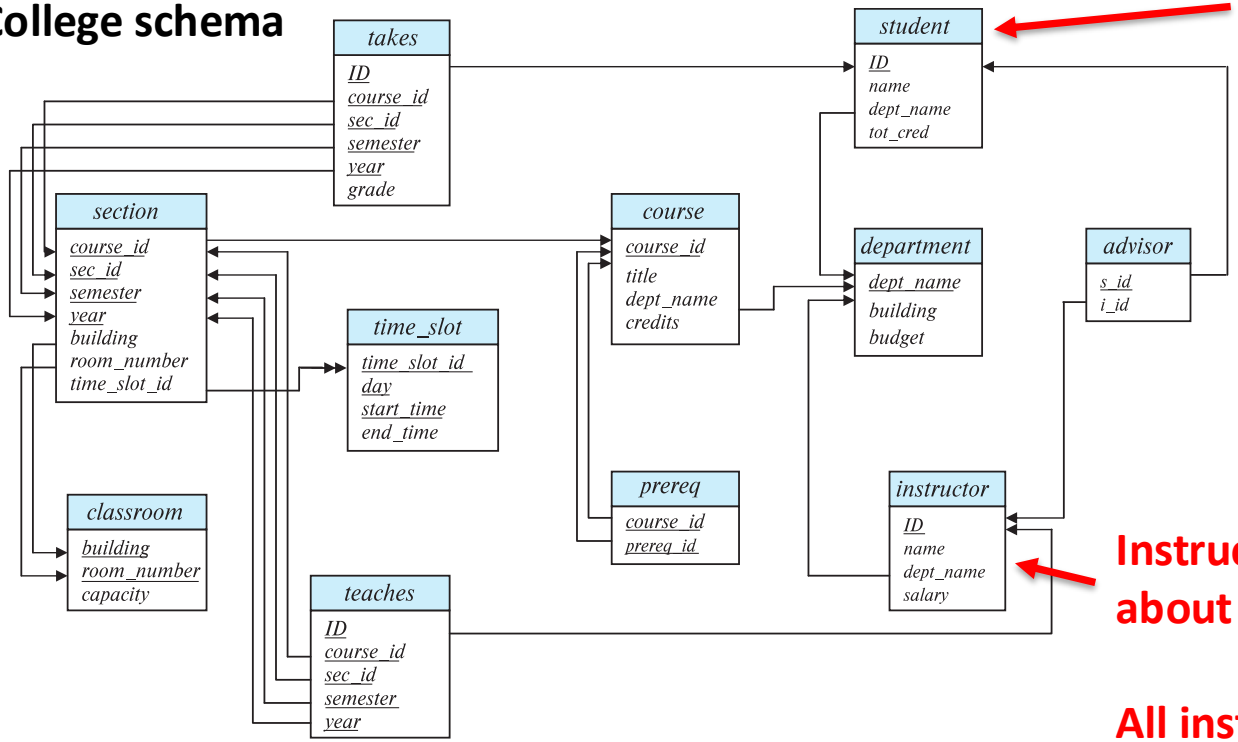
MySQL will show you the SQL it issues

Agenda

1. Creating tables and their attributes
2. Inserting, deleting, and updating rows
3. Keys
4. Integrity constraints

Review: College schema

College schema



Student relation holds data about students

All students listed, even if they have never taken a Course

Instructor relation holds data about instructors

All instructors listed, even if they have never taught a Course

Relations

- Each relation holds data about people, places, things or events (nouns)
- Relations (tables) consist of rows and columns
- Each row (tuple) represents one person, place, thing, or event
- Each column represents one attribute about a person, place, thing, or event (e.g., name)
- A column (FK) can refer to a column (PK) in another table, creating a relationship between tables

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, with $1M budget, building still to be determined

INSERT INTO department (dept_name, budget) **VALUES** ('Database Systems', 1000000)

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	1000000.00
Elec. Eng.	Taylor	85000.00
Finance	Painter	120000.00
History	Painter	50000.00
Music	Packard	80000.00
Physics	Watson	70000.00
NULL	NULL	NULL

We can also INSERT into a table using a SELECT 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

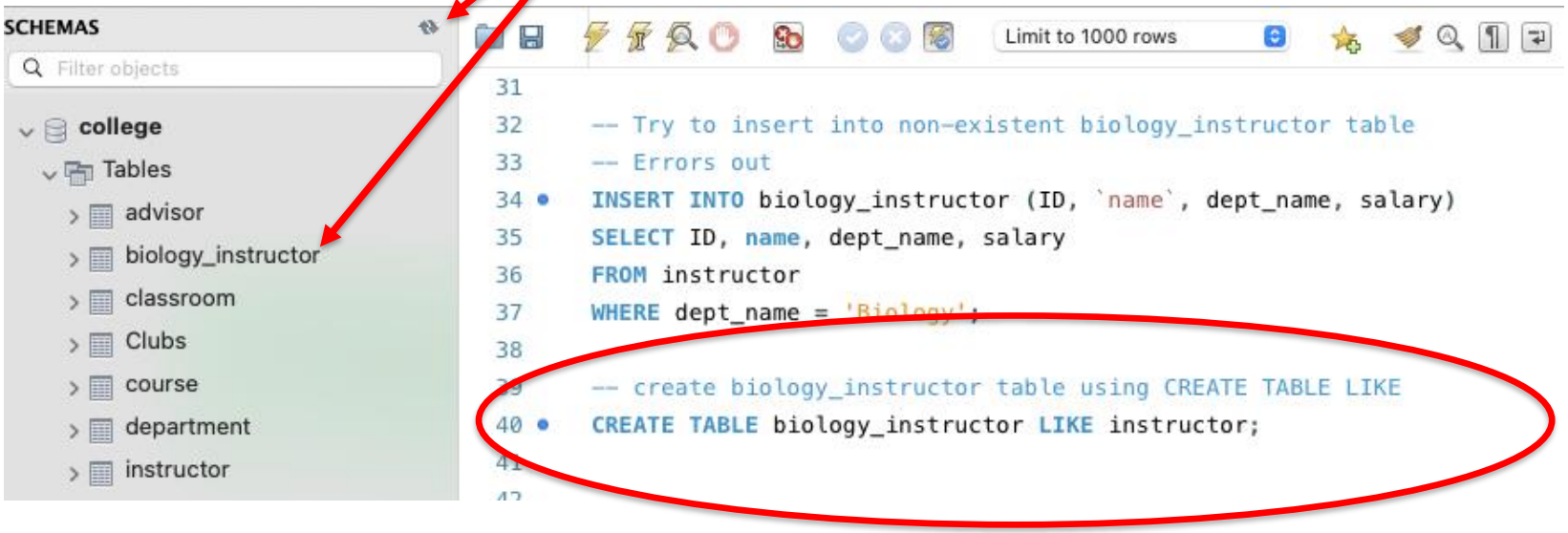
biology_instructor table

ID	name	dept_name	salary
76766	Crick	Biology	72000.00

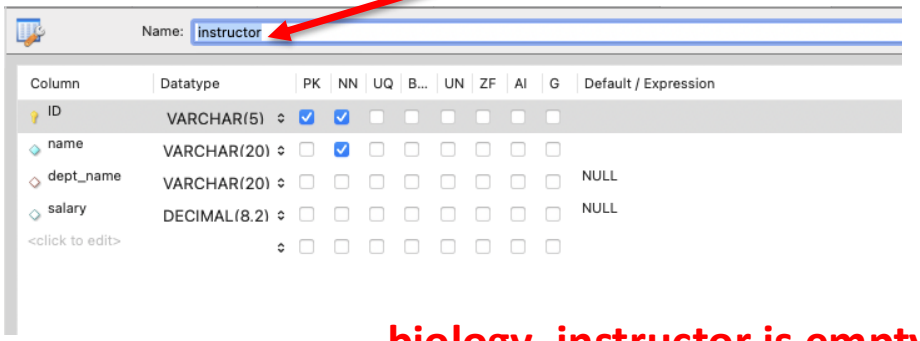
Assumes table called `biology_instructor` exists (error if not)

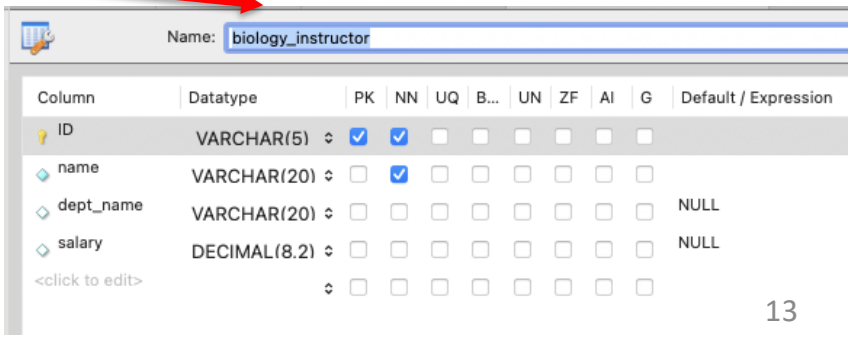
Can make a new table using CREATE TABLE LIKE

Refresh to see new table



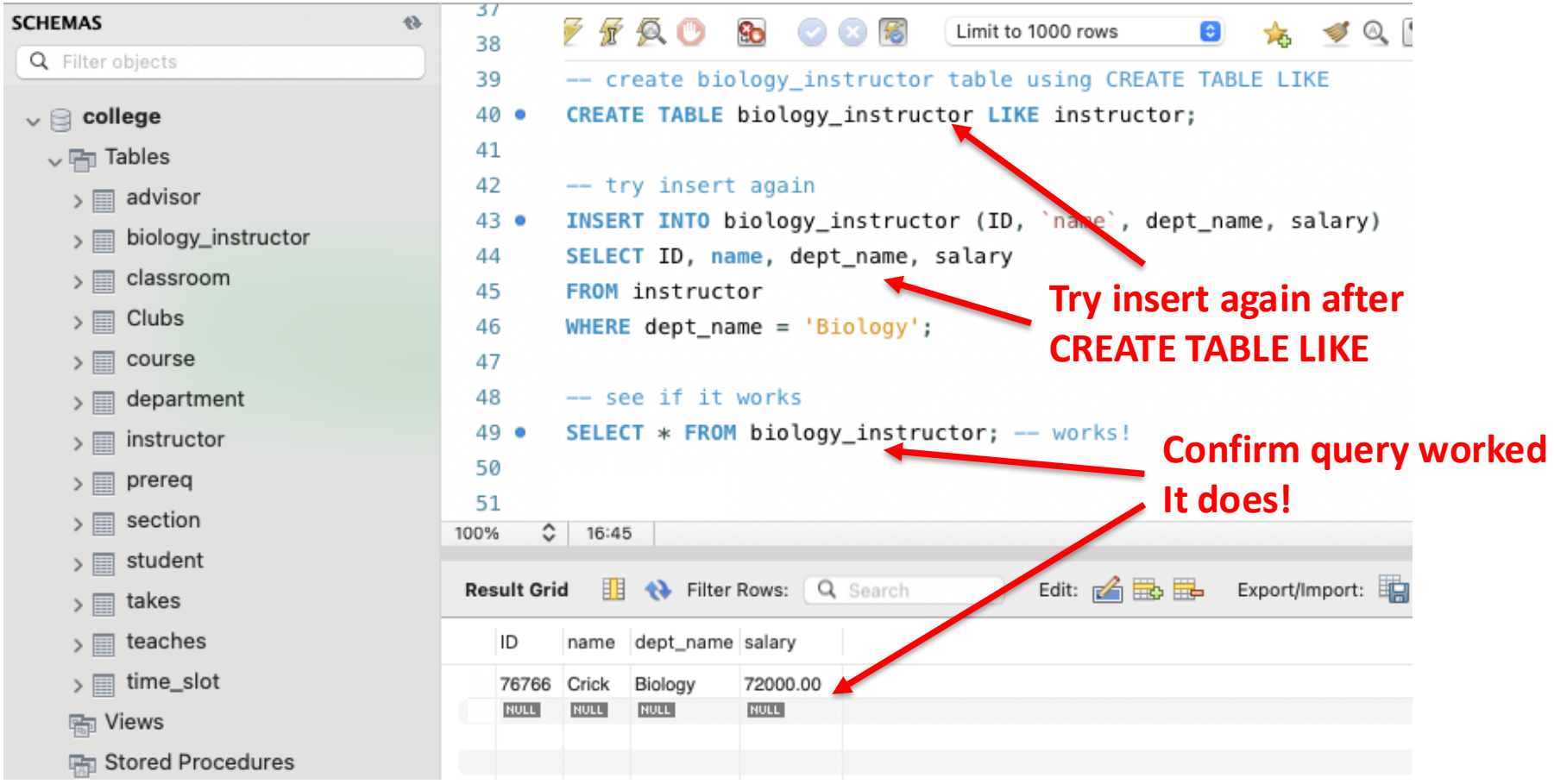
biology_instructor has same schema as instructor





biology_instructor is empty!

Can make a new table using CREATE TABLE LIKE



Alternatively, we can also create and fill table using a SELECT 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

Drop table, then remember to refresh to see new table in MySQL Workbench GUI

Example:

```
DROP TABLE biology_instructor;

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 to make and fill table 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

Update: the U 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 <= 90,000

UPDATE *instructor*
SET *salary* = *salary* * 1.05
WHERE *salary* <= 90000 ;

instructor before

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

instructor after

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	68250.00
12121	Wu	Finance	94500.00
15151	Mozart	Music	42000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	63000.00
33456	Gold	Physics	91350.00
45565	Katz	Comp. Sci.	78750.00
58583	Califieri	History	65100.00
76543	Singh	Finance	84000.00
76766	Crick	Biology	75600.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	84000.00

Not updated (Salary > 90,000)

Delete removes rows from a table

Delete: the D in CRUD

DELETE FROM table
WHERE P

Example: Delete all biology instructors

DELETE FROM *instructor*
WHERE *dept_name* = 'Biology';

Deletes: Crick

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

Practice 2: UPDATE

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)
   - NOTE: MySQL by default will not allow updates without reference to a primary key (covered soon)
   - For today, first run: SET SQL_SAFE_UPDATES = 0;
   - When finished run: SET SQL_SAFE_UPDATES = 1;
3. Update zipcode to NULL if zipcode = ‘’;

Agenda

1. Creating tables and their attributes
2. Inserting, deleting, and updating rows
3. Keys
4. Integrity constraints

We use multiple tables to avoid inconsistencies

Consider this table that combines instructor and department info

ID	name	dept_name	salary	building	budget
10101	Srinivasan	Comp. Sci.	68250.00	Taylor	100000.00
12121	Wu	Finance	94500.00	Painter	120000.00
15151	Mozart	Music	42000.00	Packard	80000.00
22222	Einstein	Physics	95000.00	Watson	70000.00
32343	El Said	History	63000.00	Painter	50000.00
33456	Gold	Physics	91350.00	Watson	70000.00
45565	Katz	Comp. Sci.	78750.00	Taylor	100000.00
58583	Califieri	History	65100.00	Painter	50000.00
76543	Singh	Finance	84000.00	Painter	120000.00
83821	Brandt	Comp. Sci.	92000.00	Taylor	100000.00
98345	Kim	Elec. Eng.	84000.00	Taylor	85000.00

Each instructor is listed with their name, dept, and salary

But, also the department's building and budget

Ex. Srinivasan, Katz, and Brandt are in CS

We see same building and budget for each CS instructor

What could go wrong?

- *Consistency anomalies* – What if budget set differently for each CS prof?
- *Update anomalies* – If data stored in multiple rows, must update all rows (ex. If update building, must update all rows)
- *Insertion anomalies* – If an instructor exists, but is not assigned to a department, they will not appear in the table (or could appear with null for department and budget)
- *Deletion anomalies* – If only instructor in a dept, if you delete that instructor, the department and building disappears

Instead of one big table, use multiple tables!

We use multiple tables to avoid inconsistencies

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

Store related data in a relation

- **Instructors in instructor table**
- **Departments in department table**

Avoid duplicating data, just enter each entity one time in its table
Recall Highlander from Day 1

Some thoughts on naming 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

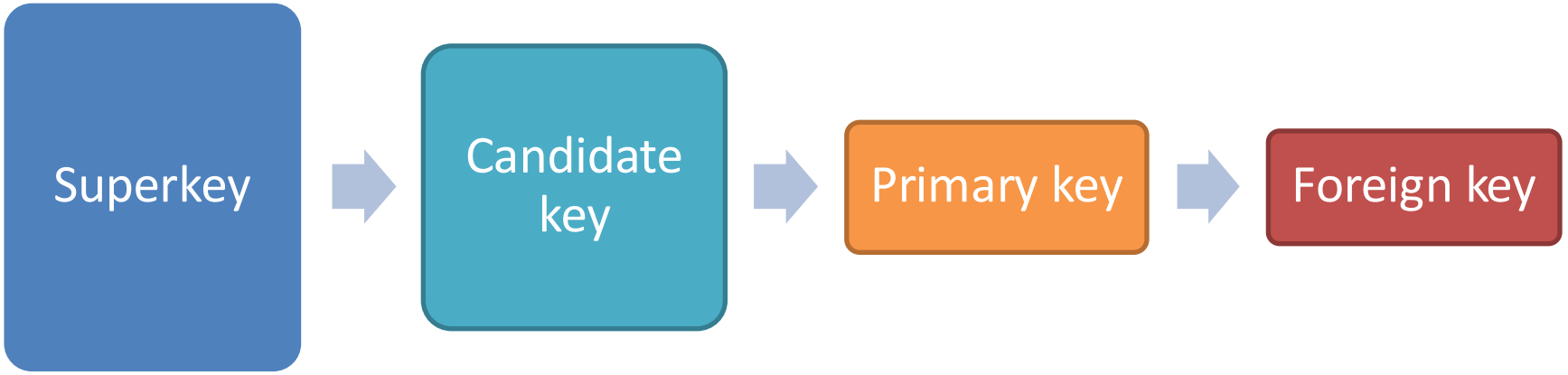
I also prefer:

- **Capital first letter then lower case, with capital letter for other words (DepartmentName) for table and attribute names**
- **I like plural table names (Departments not Department)**
- **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 in a relation

- A Candidate key chosen to identify each row

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

Primary Keys uniquely identify rows; sometimes there are “natural keys”

Primary keys

Primary keys:

- Single attribute or a combination of attributes (called a composite primary key)
- Uniquely identifies each or row in relation
- Function is to guarantee entity integrity, not to “describe” entity (if describing entity, use a non-key attribute)
- Works with foreign keys to implement relationships between entities

Natural key:

- Real-world identifier than can uniquely identify real-world objects
- Sometimes, but not always present (e.g., “CS61” is a natural key for this class)
- Familiar to end users and forms part of their day-to-day business vocabulary
- Can sometimes be used as the primary key of the entity being modeled

Surrogate key:

- System generated key
- Often generated with auto_increment

Primary Keys should have several desirable qualities

Primary keys

PK Characteristic	Notes
Unique values	PK must uniquely identify each tuple. Must be able to guarantee unique values, nulls not allowed

Based on Coronel and Morris

Primary Keys should have several desirable qualities

Primary keys

PK Characteristic	Notes
Unique values	PK must uniquely identify each tuple. Must be able to guarantee unique values, nulls not allowed
Nonintelligent	PK should not have embedded semantic meaning (if field has semantic meaning, use it as attribute, not key!). Example: StudentID f12345a better than Smith, Mary B.

Based on Coronel and Morris

Primary Keys should have several desirable qualities

Primary keys

PK Characteristic	Notes
Unique values	PK must uniquely identify each tuple. Must be able to guarantee unique values, nulls not allowed
Nonintelligent	PK should not have embedded semantic meaning (if field has semantic meaning, use it as attribute, not key!). Example: StudentID f12345a better than Smith, Mary B.
Does not change over time	Attributes with semantic meaning sometimes change over time (names are not good PKs)

Based on Coronel and Morris

Primary Keys should have several desirable qualities

Primary keys

PK Characteristic	Notes
Unique values	PK must uniquely identify each tuple. Must be able to guarantee unique values, nulls not allowed
Nonintelligent	PK should not have embedded semantic meaning (if field has semantic meaning, use it as attribute, not key!). Example: StudentID f12345a better than Smith, Mary B.
Does not change over time	Attributes with semantic meaning sometimes change over time (names are not good PKs)
Preferably single attribute	PK should minimum number of attributes possible (irreducible). Single attribute desirable, but not required. Single attribute PKs simplify FKs

Based on Coronel and Morris

Primary Keys should have several desirable qualities

Primary keys

PK Characteristic	Notes
Unique values	PK must uniquely identify each tuple. Must be able to guarantee unique values, nulls not allowed
Nonintelligent	PK should not have embedded semantic meaning (if field has semantic meaning, use it as attribute, not key!). Example: StudentID f12345a better than Smith, Mary B.
Does not change over time	Attributes with semantic meaning sometimes change over time (names are not good PKs)
Preferably single attribute	PK should minimum number of attributes possible (irreducible). Single attribute desirable, but not required. Single attribute PKs simplify FKs
Preferably numeric	Database can assign new tuples a unique value simply by incrementing the last value such as auto_increment (Mongo does this differently)

Based on Coronel and Morris

Primary Keys should have several desirable qualities

Primary keys

PK Characteristic	Notes
Unique values	PK must uniquely identify each tuple. Must be able to guarantee unique values, nulls not allowed
Nonintelligent	PK should not have embedded semantic meaning (if field has semantic meaning, use it as attribute, not key!). Example: StudentID f12345a better than Smith, Mary B.
Does not change over time	Attributes with semantic meaning sometimes change over time (names are not good PKs)
Preferably single attribute	PK should minimum number of attributes possible (irreducible). Single attribute desirable, but not required. Single attribute PKs simplify FKs
Preferably numeric	Database can assign new tuples a unique value simply by incrementing the last value such as auto_increment (Mongo does this differently)
Security compliant	Do not use attributes that have security risks such as social security numbers!

Based on Coronel and Morris

Problem using auto_increment to generate surrogate keys; consider UUID

Problem with using auto_increment for primary key

- Could be easy to guess
- Consider API route: http://<your domain>/api/employees/5
- Might guess there are IDs 4 and 6 (and beyond)
- Adversary could try plugging in random values to see what they can find

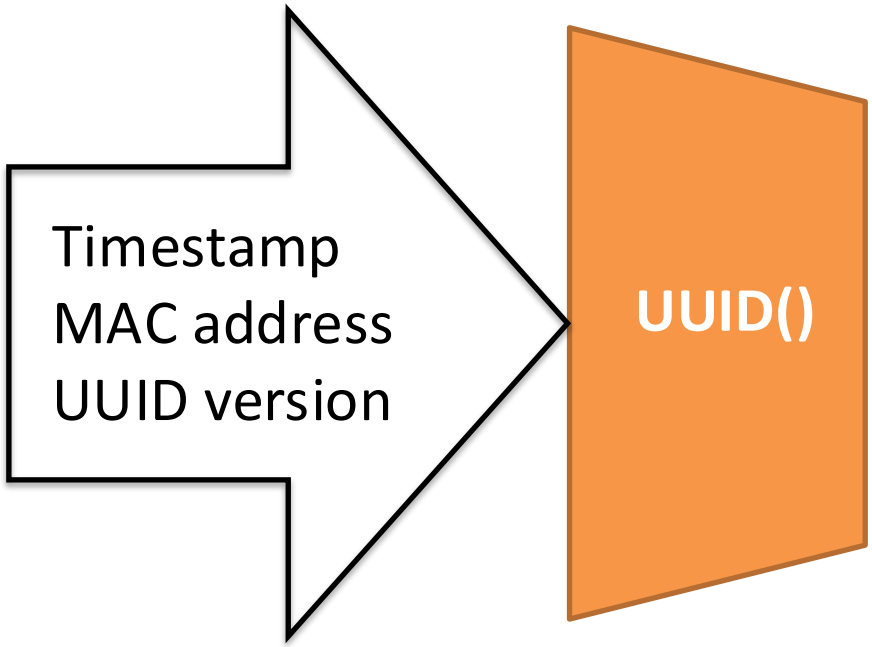
Universally Unique Identifier UUID() function will generate a 128-bit value unique across tables, databases, and servers

- UUID values do not expose the information about your data so they are safer to use in a URL
- Allow you to merge rows from different databases or distribute databases across servers
- Can be generated offline
- Can update parent and subtype in one transaction

Source: https://mysqlserverteam.com/mysql-8-0-uuid-support/

UUIDs are guaranteed to be unique, even if generated on different servers

Universally unique identifier (UUID)



12345678-1234-5678-1234-567812345678

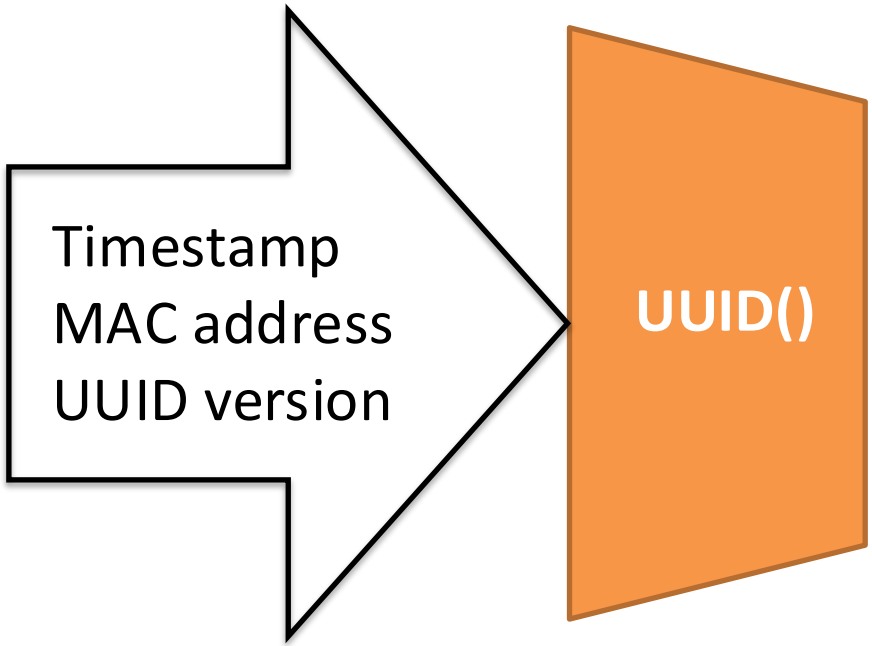
Output is 32-digit string of numbers plus four dashes, 36 characters total
Can use this as a primary key

Time will never be the same again
No two computers will have the same MAC address
Therefore, no UUIDs will be the same

Source: https://mysqlserverteam.com/mysql-8-0-uuid-support/

UUIDs have their downsides too: they are big and unordered!

Universally unique identifier (UUID)



12345678-1234-5678-1234-567812345678

Downsides:

- **Increased storage – 32 characters (plus 4 dashes) vs. Integer at 4 bytes**
- **Harder to debug:**
  SELECT * FROM Table
  WHERE ID = '12345678-1234-5678-1234-567812345678'
- **Performance issues: large key size and not ordered**

Source: https://mysqlserverteam.com/mysql-8-0-uuid-support/

MySQL has commands that solve these problems

Universally unique identifier (UUID)

```
CREATE TABLE UUIDDemo (
    PK BINARY(16),
    ItemName VARCHAR(20),
    PRIMARY KEY (PK));

-- insert items into table with PK of UUID
INSERT INTO UUIDDemo(PK, ItemName) VALUES (UUID_TO_BIN(UUID()), 'Item1');
INSERT INTO UUIDDemo(PK, ItemName) VALUES (UUID_TO_BIN(UUID()), 'Item2');
INSERT INTO UUIDDemo(PK, ItemName) VALUES (UUID_TO_BIN(UUID()), 'Item3');

-- see results
SELECT BIN_TO_UUID(PK), ItemName
    FROM UUIDDemo;
```

Can store UUID as 16 bytes

UUID_TO_BIN converts 36-character UUID to 16-byte binary

BIN_TO_UUID converts binary back to 36-character string

1:142

ult Grid | Filter Rows: Search | Export:

BIN_TO_UUID(PK)	ItemName
38301ff4-3416-11f1-851e-5a28b826936f	Item1
3a2ffa54-3416-11f1-851e-5a28b826936f	Item2
3c00241c-3416-11f1-851e-5a28b826936f	Item3

NOTE: time elements on left, change most rapidly

Can reverse with UUID_TO_BIN(UUID(), true)

PK then stored in ascending order

Source: https://mysqlserverteam.com/mysql-8-0-uuid-support/

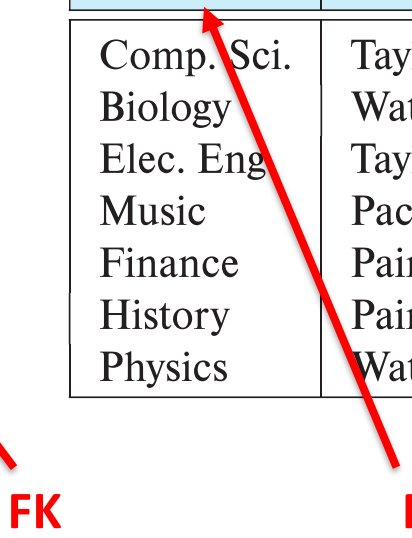
Foreign keys constrain attribute values to primary keys of another relation

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 usually is in practice)

NYC health inspections store the same data on restaurants for each inspection

USE nyc_data;
SELECT * **FROM** restaurant_inspections;

Each row is one health inspection *violation* for a restaurant

CAMIS	DBA	BORO	BUILDING	STREET	ZIPCODE	PHONE	CUISINE DESCRIPTION	INSPECTION DATE	ACTION
50172433	OAKBERRY ACAI & SMOOTHIES	Brooklyn	230	LIVINGSTON STR…	11201	9172143975		01/01/1900	
50176125	JUICE TIME / PAPA KANAFA	Queens	24-09	STEINWAY STREET	11103	9173464062		01/01/1900	
50010248	CAFE GRUMPY	Manhattan		LEXINGTON AVE…		2126612198	Coffee/Tea	05/22/2024	Violations were cited in the following area(s).
50168118	GYRO KING	Brooklyn	872A	FLATBUSH AVENUE	11226	8453215804		01/01/1900	
50178778	KASAMIA RESTAURANT & BAR	Queens	62-27	FRESH POND RO…	11379	7183506406		01/01/1900	
50135830	STUDIO DEM	Brooklyn	241	WYTHE AVENUE	11249	3477053234		01/01/1900	
50170174	GRASMERE BODY BUILDING INC.	Staten Isl…	2590	HYLAN BOULEVA…	10306	718691027		01/01/1900	
41503108	TULCINGO RESTAURANT	Queens		NATIONAL STREET		7186398880	Mexican	03/04/2025	Violations were cited in the following area(s).
50151973	532 COURT	Brooklyn	532	COURT STREET	11231	9175182169		01/01/1900	
50175335	HOKKAIDO BAKED CHEESE TART	Manhattan	220	8 AVENUE	10011	3476156148		01/01/1900	
50089033	THE LEGEND'S LOUNGE	Manhattan	2823	FREDERICK DOU…		9143544662		01/01/1900	
41403890	TOWERS CAFE	Brooklyn	760	BROADWAY	11206	7186303042	American	03/03/2023	Establishment re-opened by DOHMH.
50172761	NUMCHANA9 CORPORATION	Queens	215-01	73 AVENUE	11364	7183953666		01/01/1900	
50157923	GOLAN HEIGHTS	Manhattan	2553	AMSTERDAM AVE…	10033	7189093343		01/01/1900	
50154196	YULIANNA'S TAVERN	Manhattan	4476	BROADWAY	10040	9175290966		01/01/1900	
50105390	ABSURD CONCLAVE	Brooklyn	360	JEFFERSON STR…	11237	3342207148		01/01/1900	
41580023	SAGE	Brooklyn		GRAHAM AVENUE		7182186644	Thai	06/13/2023	Violations were cited in the following area(s).
50173758	ROMAINE EMPIRE INC FARMER'S FRIDGE	Manhattan	5141	BROADWAY	10034	3122290099		01/01/1900	
50134660	MANHATTAN ELITE	Manhattan		CHELSEA PIERS	10011	7327275030		01/01/1900	
50171803	FSW 3RD AVE INC	Manhattan	1911	3 AVENUE	10029	3478259092		01/01/1900	
50173129	LOU WAI LOU 888 INC	Queens	60-06	MAIN STREET	11355	7184618899		01/01/1900	
50173884	PANDA EXPRESS INC	Brooklyn	84	COURT STREET	11201	6267999898		01/01/1900	
50093744	MORI MORI	Brooklyn	55	WATER STREET	11201	9177437388	Salads	09/12/2025	Violations were cited in the following area(s).
50179211	RICO CHIMI	Brooklyn	1492	MYRTLE AVENUE	11237	6467275066		01/01/1900	
50153385	DOS TOROS	Queens		AIRPORT		5163989952		01/01/1900	
50172498	CAFE HERE	Brooklyn	499	VAN BRUNT STR…	11231	6318716795		01/01/1900	

Restaurants have ID (CAMIS), name (dba), boro, building, street, zip code, phone, cuisine description
This data is repeated each time a restaurant is inspected (inconsistencies could occur)

Make a Restaurants table to store data about each restaurant

DROP TABLE only if it exists (otherwise error if DROP TABLE and table does not exist)

```
DROP TABLE IF EXISTS Restaurants ;

CREATE TABLE Restaurants (
  RestaurantID INT NOT NULL,
  RestaurantName VARCHAR(100) NULL DEFAULT NULL,
  Boro VARCHAR(20) NULL DEFAULT NULL,
  Building VARCHAR(20) NULL DEFAULT NULL,
  Street VARCHAR(100) NULL DEFAULT NULL,
  ZipCode INT NULL DEFAULT NULL,
  Phone BIGINT NULL DEFAULT NULL,
  Latitude DOUBLE NULL DEFAULT NULL,
  Longitude DOUBLE NULL DEFAULT NULL,
  CuisineDescription VARCHAR(100) DEFAULT NULL,
  PRIMARY KEY (RestaurantID)
  );
```

Create attributes for:

- ***ID***
- ***Name***
- ***Boro***
- ***Building***
- ***Street***
- ***ZipCode***
- ***Phone***
- ***Latitude***
- ***Longitude***
- ***CusineDescription***

Set PRIMARY KEY to be RestaurantID

Make a Restaurants table to store data about each restaurant

DROP TABLE only if it exists (otherwise error if DROP TABLE and table does not exist)

```
DROP TABLE IF EXISTS Restaurants ;

CREATE TABLE Restaurants (
  RestaurantID INT NOT NULL,
  RestaurantName VARCHAR(100) NUL
  Boro VARCHAR(20) NULL DEFAULT NU
  Building VARCHAR(20) NULL DEFAUL
  Street VARCHAR(100) NULL DEFAULT
  ZipCode INT NULL DEFAULT NULL,
  Phone BIGINT NULL DEFAULT NULL,
  Latitude DOUBLE NULL DEFAULT NUL
  Longitude DOUBLE NULL DEFAULT NU
  CuisineDescription VARCHAR(100) DE
  PRIMARY KEY (RestaurantID)
  );
```

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
RestaurantID	INT	☑	☑	☐	☐	☐	☐	☐	☐	
RestaurantNa...	VARCHAR(100	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Boro	VARCHAR(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Building	VARCHAR(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Street	VARCHAR(100	☐	☐	☐	☐	☐	☐	☐	☐	NULL
ZipCode	INT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Phone	BIGINT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Latitude	DOUBLE	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Longitude	DOUBLE	☐	☐	☐	☐	☐	☐	☐	☐	NULL
CuisineDescr...	VARCHAR(100	☐	☐	☐	☐	☐	☐	☐	☐	NULL
<click to edit>		☐	☐	☐	☐	☐	☐	☐	☐	

Set PRIMARY KEY to be RestaurantID

First find distinct restaurants; some have been inspected many times

DISTINCT removes duplicates

```
147 •  SELECT DISTINCT camis AS RestaurantID,
148        dba AS RestaurantName, boro, building, street,
149        zipcode, phone, latitude, longitude, `cuisine description` AS CuisineDescription
150        FROM restaurant_inspections;
151
```

100% 18:147

Result Grid Filter Rows: Search Export: Fetch rows:

RestaurantID	RestaurantName	boro	building	street	zipcode	phone	latitude	longitude	CuisineDescripti...
50167752	ONE AND ONE	Manhattan	12	1 AVENUE	10009	6466577217	40.72335785103	-73.98834703895	
50177986	MEZEH	Manhattan	900	BROADWAY	10003	6465450078	40.73910196003	-73.98977327391	
50178407	ALADDIN HALAL RESTAURANT AND SWEET INC.	Queens	37-08	73 STREET	11372	6462907120	40.74844477194	-73.89264754085	
50167880	SILKY KITCHEN	Manhattan	12	JOHN STREET	10038	9178218869	40.70999655005	-74.00910046587	
50172123	WONDER	Queens	56-16	MYRTLE AVENUE	11385	9088086982	40.70018657781	-73.90549245842	
50161512	CGTENANT LLC	Manhattan	900	BROADWAY	10003	3056141532	40.73910196003	-73.98977327391	
50160730	HOUSE OF PASTA	Manhattan	511	EAST 12 STREET	10009	3479756151	40.7288710724	-73.98080529356	
40876618	BIJAN'S	Brooklyn	7981	HOYT STREET		7188555574	NULL	NULL	American
50169812	MAKEROOM COPR. LLC	Queens	64-02	MYRTLE AVENUE	11385	3477129813	40.70096540095	-73.89225905084	
50178782	KNEAD SOME LOVE NY	Manhattan	151	WEST 34 STREET	10001	7188442153	40.75050916477	-73.98947917044	
50176337	Country Donuts	Staten Isl...	877	HUGUENOT AVE...	10312	9084158637	40.53280507332	-74.19223454555	
50121480	PICCIOTTO SICILIAN STEER FOOD&CAFE	Queens	42-34	235 STREET	11363	7182338213	40.76766199457	-73.74790016564	
50178068	SWEET HOSPITALITY GROUP, LLC	Manhattan	434	LAFAYETTE STRE...	10003	3477743522	40.72917432573	-73.99202983084	
50177683	POPUPBAGELS INC	Manhattan	315	GREENWICH STR...	10013	9787614714	40.71699830807	-74.01080050265	
50159945	SEAPARK 59 CORP	Manhattan	41	WEST 40 STREET	10018	6466372137	40.75306937642	-73.98388067316	
50175558	AB HOT PIZZA INC.	Manhattan	200	DYCKMAN STREET	10040	3478845060	40.86487913685	-73.92686335619	

Now use previous query to fill the Restaurants table

NOTE: CAMIS in restaurant_inspections is called RestauantID in Restaurants

```
153 •  INSERT INTO Restaurants (RestaurantID, RestaurantName, Boro, Building, Street, ZipCode, Phone, Latitude, Longitude, CuisineDescription)
154        SELECT DISTINCT camis AS RestaurantID,
155            dba AS RestaurantName, boro, building, street,
156            zipcode, phone, latitude, longitude, `cuisine description` AS CuisineDescription
157        FROM restaurant_inspections
158        WHERE dba IS NOT NULL;
159
160 •  SELECT * FROM Restaurants;
161
```

100% 20:160

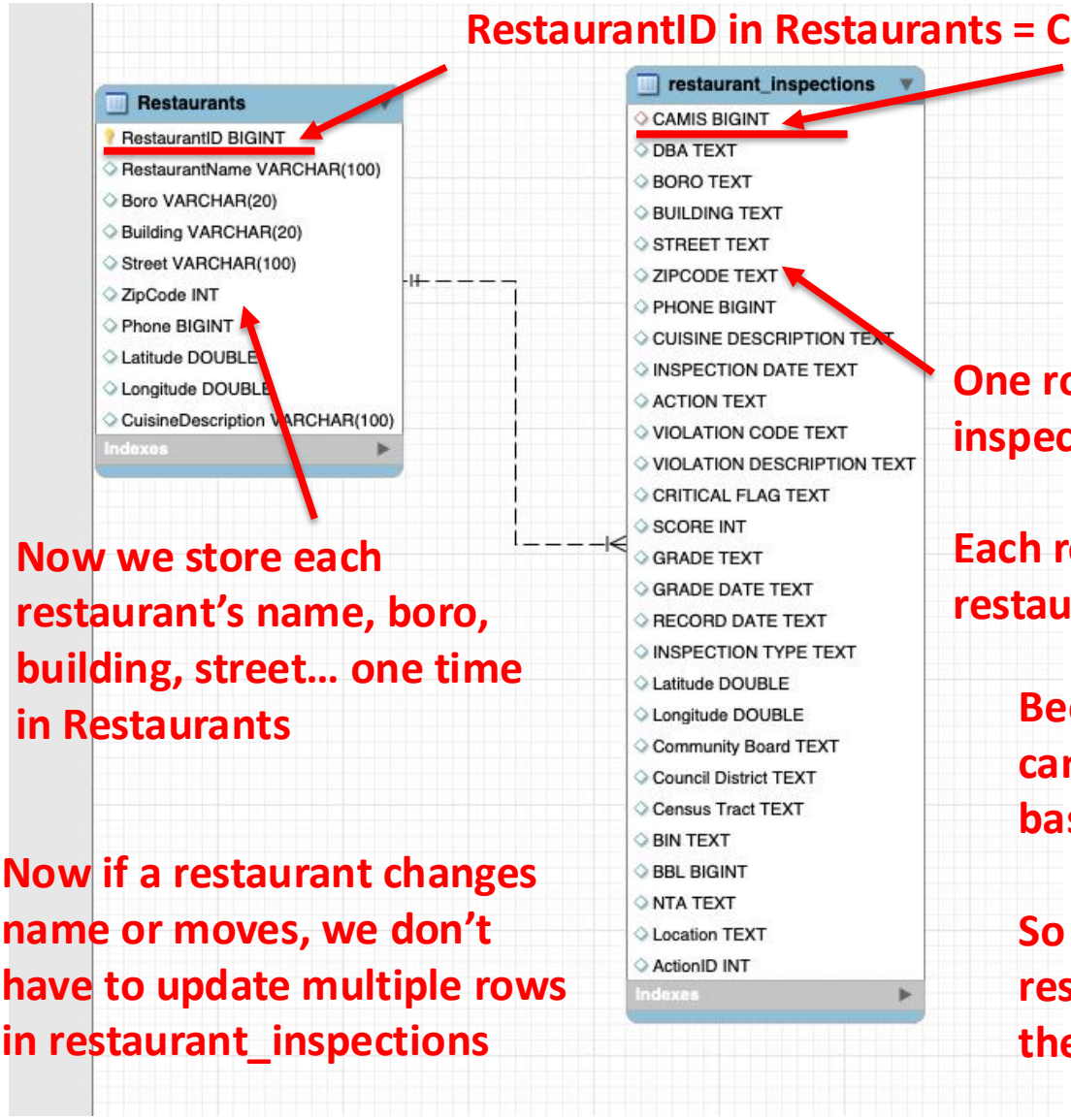
Now we can alter restaurant_inspections so we don't need to duplicate name, boro, street, … for each inspection

Result Grid | Filter Rows: Search | Edit: | Export/Import: | Fetch rows:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude	Longitude	CuisineDescription
40356018	RIVIERA CATERERS	Brooklyn	2780	STILLWELL AVENUE	11224	7183723031	40.57989566331	-73.98208665586	American
40356483	WILKEN'S FINE FOOD	Brooklyn	7114	AVENUE U	11234	7184443838	40.62011159093	-73.9069894897	Sandwiches
40356731	TASTE THE TROPICS ICE CREAM	Brooklyn	1839	NOSTRAND AVENUE	11226	7188560821	40.64079501746	-73.94848798153	Frozen Desserts
40357217	ASIA PLAZA CAFÉ	Bronx	2300	SOUTHERN BOULEVARD	10460	7187411426	40.850536752	-73.88245508146	American
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691423	-73.96958057845	American
40359705	NATHAN'S FAMOUS	Brooklyn	1310	SURF AVENUE	11224	7183332202	40.57553686942	-73.98165226411	Hotdogs
40360045	SEUDA FOODS	Brooklyn	705	KINGS HIGHWAY	11223	7183751500	40.60618692143	-73.9654664459	Jewish/Kosher
40361618	SAL'S DELI	Queens	129-08	20 AVENUE	11356	7186619498	40.78167383607	-73.83941597683	Sandwiches
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262063589	-73.96770961573	American
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STREET	10012	2129952570	40.72574361744	-73.99747810759	American
40362432	HO MEI	Queens	103-05	37 AVENUE	11368	7187796903	40.75330591158	-73.86429537578	Chinese
40362869	SHASHEMENE INT'L RESTAURA...	Brooklyn	195	EAST 56 STREET	11203	3474300871	40.65198322279	-73.92457494415	Caribbean
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542308	-73.99056473379	American
40363427	BAGELS N BUNS	Staten Isl...	2491	VICTORY BOULEVARD	10314	7187611900	40.61026786383	-74.14640121029	Sandwiches
40363834	CARVEL	Staten Isl...	1111	HYLAN BOULEVARD	10305	7188167807	40.59855313075	-74.07964600818	Frozen Desserts
40363920	NEW GOLDEN BILLION	Brooklyn	976	RUTLAND ROAD	11212	7187357707	40.66214059458	-73.92718021322	Chinese
40364179	MISS MAIME'S SPOONBREAD T...	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137126967	-73.96015994361	Soul Food

We can use CAMIS to look up Restaurant info

Now we have two tables, where RestauantID = CAMIS



Now we store each restaurant's name, boro, building, street… one time in Restaurants

Now if a restaurant changes name or moves, we don't have to update multiple rows in restaurant_inspections

One row in restaurant_inspections is one inspection violation of a restaurant

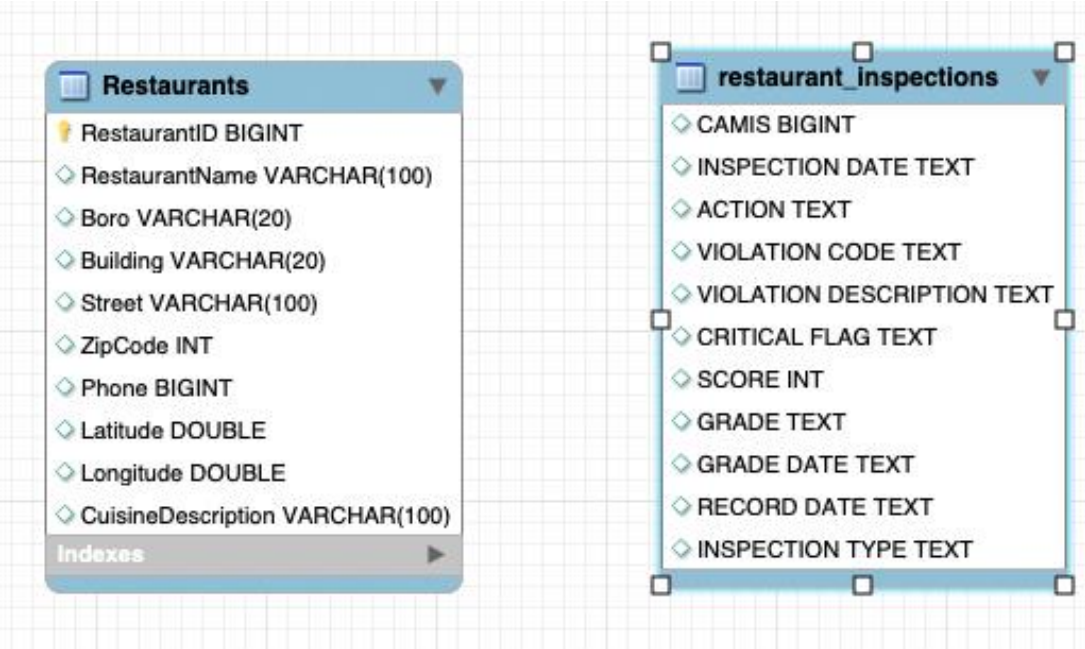
Each row, however, repeats the restaurant name, boro, building, street…

Because RestaurantID = CAMIS, we can look up the Restaurant's details based on CAMIS when needed

So we will keep CAMIS in restaurant_ inspections, but delete the other redundant attributes

Remove unneeded info from restaurant_inspections

```
ALTER TABLE restaurant_inspections
DROP COLUMN `CUISINE DESCRIPTION`,
DROP COLUMN PHONE,
DROP COLUMN Latitude,
DROP COLUMN Longitude,
DROP COLUMN ZIPCODE,
DROP COLUMN STREET,
DROP COLUMN BUILDING,
DROP COLUMN BORO,
DROP COLUMN DBA,
DROP COLUMN `Community board`,
DROP COLUMN `Council district`,
DROP COLUMN `Census tract`,
DROP COLUMN `BIN`,
DROP COLUMN `BBL`,
DROP COLUMN `NTA`,
DROP COLUMN `Location`
;
```



Restaurants has data about restaurants
Restaurant_inspections has data about each inspection
RestaurantID and CAMIS tie them together

Use CAMIS = RestaurantID to get restaurant details for each inspection

Original data restored

```
SELECT r.*, i.*
    FROM Restaurants r, restaurant_inspections i
    WHERE r.RestaurantID = i.CAMIS;
```

100% 8:176

Result Grid Filter Rows: Search Export: Fetch rows:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude	Longitude	CuisineDescripti...	CAMIS	INSPECTION DATE	ACTION
50167752	ONE AND ONE	Manhattan	12	1 AVENUE	10009	6466577217	40.72335785103	-73.98834703895		50167752	01/01/1900	
50177986	MEZEH	Manhattan	900	BROADWAY	10003	6465450078	40.73910196003	-73.98977327391		50177986	01/01/1900	
50178407	ALADDIN HALAL RESTAURANT AND SWEET...	Queens	37-08	73 STREET	11372	6462907120	40.74844477194	-73.89264754085		50178407	01/01/1900	
50167880	SILKY KITCHEN	Manhattan	12	JOHN STREET	10038	9178218869	40.70999655005	-74.00910046587		50167880	01/01/1900	
50172123	WONDER	Queens	56-16	MYRTLE AVENUE	11385	9088086982	40.70018657781	-73.90549245842		50172123	01/01/1900	
50161512	CGTENANT LLC	Manhattan	900	BROADWAY	10003	3056141532	40.73910196003	-73.98977327391		50161512	01/01/1900	
50160730	HOUSE OF PASTA	Manhattan	511	EAST 12 STREET	10009	3479756151	40.7288710724	-73.98080529356		50160730	01/01/1900	
40876618	BIJAN'S	Brooklyn	7981	HOYT STREET	0	7188555574	NULL	NULL	American	40876618	06/22/2022	Violations were cited in the following area(s
50169812	MAKEROOM COPR. LLC	Queens	64-02	MYRTLE AVENUE	11385	3477129813	40.70096540095	-73.89225905084		50169812	01/01/1900	
50178782	KNEAD SOME LOVE NY	Manhattan	151	WEST 34 STREET	10001	7188442153	40.75050916477	-73.98947917044		50178782	01/01/1900	
50176337	Country Donuts	Staten Isl...	877	HUGUENOT AVE...	10312	9084158637	40.53280507332	-74.19223454555		50176337	01/01/1900	
50121480	PICCIOTTO SICILIAN STEER FOOD&CAFE	Queens	42-34	235 STREET	11363	7182338213	40.76766199457	-73.74790016564		50121480	01/01/1900	
50178068	SWEET HOSPITALITY GROUP, LLC	Manhattan	434	LAFAYETTE STRE...	10003	3477743522	40.72917432573	-73.99202983084		50178068	01/01/1900	
50177683	POPUPBAGELS INC	Manhattan	315	GREENWICH STR...	10013	9787614714	40.71699830807	-74.01080050265		50177683	01/01/1900	
50159945	SEAPARK 59 CORP	Manhattan	41	WEST 40 STREET	10018	6466372137	40.75306937642	-73.98388067316		50159945	01/01/1900	
50175558	AB HOT PIZZA INC.	Manhattan	200	DYCKMAN STREET	10040	3478845060	40.86487913685	-73.92686335619		50175558	01/01/1900	
50177204	DANTE APERITIVO	Manhattan	51	BANK STREET	10014	3477075656	40.7367967442	-74.00401260566		50177204	01/01/1900	
50170422	THE TOWN HALL	Manhattan	123	WEST 43 STREET	10036	2125825472	40.75583616366	-73.98438172067		50170422	01/01/1900	

From Restaurants

From restaurant_inspections

Now if a restaurant moves or changes name, just update one row in one table

```
180 •  SELECT * FROM Restaurants
181    WHERE RestaurantID = 40758684;
182
```

Pick a Restaurant

100% 34:186

Result Grid | Filter Rows: Search | Edit: | Export/Import:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude	Longitude	CuisineDescription
40758684	JUICY LUCY'S	Manhattan	72	EAST 1 STREET	10003	2127775829	40.72337709977	-73.98870059301	Juice, Smoothies, Fruit Salads
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
184    -- see inspection results for this restaurant
185 •  SELECT r.RestaurantName, i.*
186        FROM Restaurants r, restaurant_inspections i
187        WHERE r.RestaurantID = i.CAMIS AND CAMIS = 40758684;
188
```

100% 23:186

Result Grid | Filter Rows: Search | Export:

RestaurantName	CAMIS	INSPECTION DATE	ACTION	VIOLATION CODE	VIOLATION DESCRIPTION	CRITIC
JUICY LUCY's	40758684	07/31/2025	Violations were cited in the following area(s).	03I	Juice packaged on premises with no or incomplete label, no warni...	Critical

Juicy Lucy's inspected one time, violation code 03I

Now if a restaurant moves or changes name, just update one row in one table

```
180 •  SELECT * FROM Restaurants
181    WHERE RestaurantID = 40758684;
182
```

Previous data

100% 34:186

Result Grid Filter Rows: Search Edit: Export/Import:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude	Longitude	CuisineDescription
40758684	JUICY LUCY'S	Manhattan	72	EAST 1 STREET	10003	2127775829	40.72337709977	-73.98870059301	Juice, Smoothies, Fruit Salads
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
183    -- Change scandalous name and move to building 73 from 72
184 •  UPDATE Restaurants
185        SET RestaurantName = 'NON-JUICY LUCY\'s', Building = 73
186        WHERE RestaurantID = 40758684;
187
188 •  SELECT * FROM Restaurants
189    WHERE RestaurantID = 40758684;
190
```

Change scandalous name and move building in Restaurant table (keep same RestaurantID)

Inspection table unchanged

100% 14:188

Result Grid Filter Rows: Search Edit: Export/Import:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude	Longitude	CuisineDescription
40758684	NON-JUICY LUCY's	Manhattan	73	EAST 1 STREET	10003	2127775829	40.72337709977	-73.98870059301	Juice, Smoothies, Fruit Salads
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Now if a restaurant moves or changes name, just update one row in one table

```
180 •  SELECT * FROM Restaurants
181    WHERE RestaurantID = 40758684;
182
```

Previous data

100% 34:186

Result Grid | Filter Rows: Search | Edit: | Export/Import:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude	Longitude	CuisineDescription
40758684	JUICY LUCY'S	Manhattan	72	EAST 1 STREET	10003	2127775829	40.72337709977	-73.98870059301	Juice, Smoothies, Fruit Salads
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
198    -- see inspection results not changed, even though restaurant details did change
199 •  SELECT r.RestaurantName, i.*
200        FROM Restaurants r, restaurant_inspections i
201        WHERE r.RestaurantID = i.CAMIS AND CAMIS = 40758684 ;
202
```

100% 18:199

Result Grid | Filter Rows: Search | Export:

RestaurantName	CAMIS	INSPECTION DATE	ACTION	VIOLATION CODE	VIOLATION DESCRIPTION	CRITICAL FLAG	SCOR
NON-JUICY LUCY's	40758684	07/31/2025	Violations were cited in the following area(s)	03I	Juice packaged on premises with no or incomplete label, no warni…	Critical	10

Same restaurant with new name, Non-Juicy Lucy's, keeps old inspection results, violation code 03I due to same RestaurantID (and CAMIS in restaurant_inspections)

Agenda

1. Creating tables and their attributes
2. Inserting, deleting, and updating rows
3. Keys
4. Integrity constraints

Integrity constrains ensure the data is consistent with our expectations

Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency

We can ensure:

- A checking account must have a balance greater than $10,000
- The salary of an employee must be at least $15.00 an hour
- A customer must have a unique phone number

Example:

```
CREATE TABLE Employees (
    EmployeeID  INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(20) NOT NULL,
    Phone INT UNIQUE,
    Salary INT,
    PRIMARY KEY (EmployeeID),
    CHECK (Salary > 0)
);
```

Note: MySQL versions before 8 ignored check statements!

EmployeeID set as PRIMARY KEY (can not be NULL) and auto_increment

Name is not NULL

New constraints:

- **Phone must be unique (e.g., multiple people cannot have the same phone number)**
- **CHECK on Salary ensures Salary is greater than 0, but NULL is accepted! (can also check when Salary declared)**

INSERT and UPDATE queries fail if constraints not met

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 (INT or 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

Health inspection questions

Does it make sense to have an inspection of a restaurant that is not in the Restaurants table?

How can we make sure every inspection is for a valid Restaurant?

What should we do with inspections if we delete a row in the Restaurants table?

- If we don't track the restaurant any longer, perhaps we should delete all inspections in restaurant_inspections
- Perhaps we should keep the inspection records, but maybe we should set the CAMIS to null
- Maybe we should set CAMIS to another default value
- (What we would probably do is add a column to Restaurants called Active or similar, and set to False if Restaurant goes out of business, then we can keep inspection results in restaurant_inspections)

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)

Table r_1 getting FK

Referenced table r_2

Attribute holding FK in table getting FK

Attribute in referenced table that serves as FK constraint

Values for attribute A_1 in table r_1 must be a primary key value in table r_2

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)

  ON DELETE CASCADE

  ON UDPATE CASCADE;

If referenced table changes, cascade same action (delete or update) to table getting foreign key

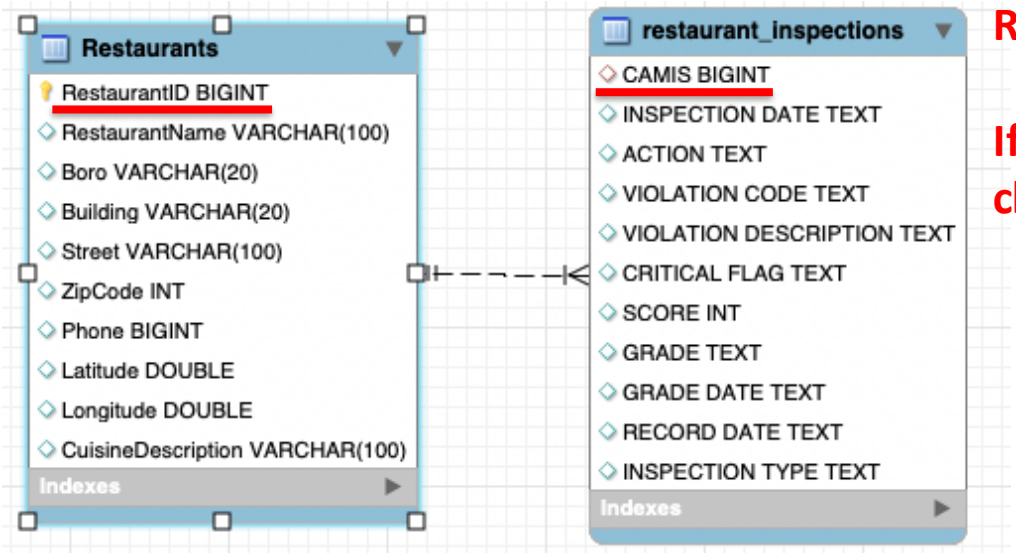
Add a foreign key constraint to an existing table with the ALTER TABLE command

Create a foreign key constraint

Each CAMIS value in restaurant_inspections must be in a row in Restaurants

```
ALTER TABLE restaurant_inspections
    ADD FOREIGN KEY(CAMIS) REFERENCES Restaurants(RestaurantID)
        ON DELETE CASCADE
        ON UPDATE CASCADE;
```

If Restaurants table row is deleted, delete the row in restaurant_inspections with CAMIS = RestaurantID

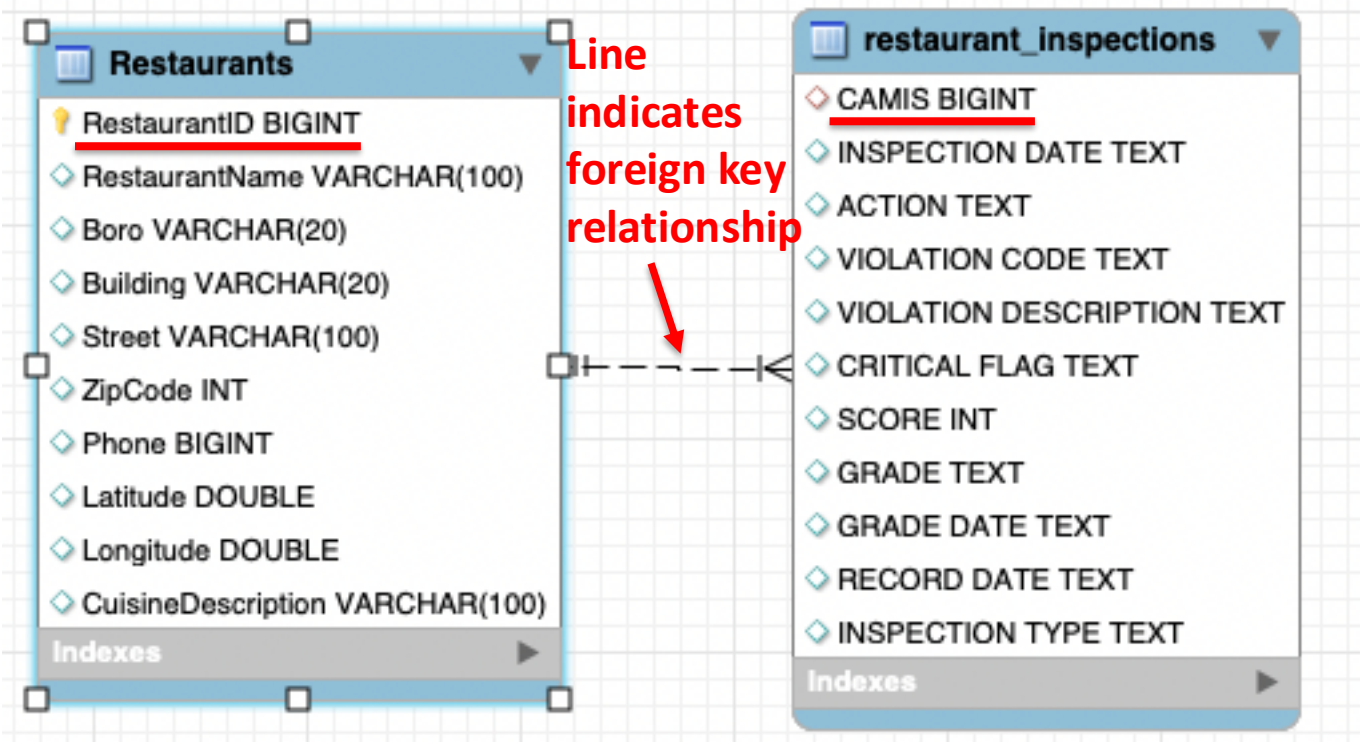


If Restaurants table RestaurantID is updated, change CAMIS in restaurant_inspections

Other choices instead of CASCADE:

- **SET NULL**
- **NO ACTION**
- **RESTRICT (prevent parent table changes if child table has rows with foreign key matching parent key)**

Each inspection CAMIS must reference an existing restaurant RestaurantID



Database will reject an insert into restaurant_inspections that has a CAMIS not in RestaurantIs

INSERT INTO restaurant_inspections
(CAMIS, `inspection date`)
VALUES (100, '4/1/2026');

No Restaurant has RestaurantID = 100

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails

Changes to Restaurants table cascades to restaurant_inspections table with FK

```
220 •  SELECT r.RestaurantID, r.RestaurantName, i.*
221        FROM Restaurants r, restaurant_inspections i
222        WHERE r.RestaurantID = i.CAMIS AND camis = 40758684;
223
```

100% 11:221

Result Grid Filter Rows: Search Export:

RestaurantID	RestaurantName	CAMIS	INSPECTION DATE	ACTION	VIOLATION CODE	VIOLATION DESCRIPTION
40758684	NON-JUICY LUCY's	40758684	07/31/2025	Violations were cited in the following area(s).	03I	Juice packaged on premises v

NON-JUICY LUCY'S has RestaurantID=CAMIS=40758684

Changes to Restaurants table cascades to restaurant_inspections table with FK

```
226 •  UPDATE Restaurants
227        SET RestaurantID = 1
228        WHERE RestaurantID = 40758684;
229
230    -- check RestaurantName update in Restaurants
231 •  SELECT * FROM Restaurants WHERE RestaurantID = 1; -- NOT JUICY LUCY's
```

Update NON-JUICY LUCY'S RestaurantID=1 in Restaurants

Recall we set ON UPDATE to CASCADE

100% 18:231

Result Grid | Filter Rows: Search | Edit: | Export/Import:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude	Longitude	CuisineDescription
1	NON-JUICY LUCY's	Manhattan	73	EAST 1 STREET	10003	2127775829	40.72337709977	-73.98870059301	Juice, Smoothies, Fruit Salads
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Changes to Restaurants table cascades to restaurant_inspections table with FK

```
233     -- check change cascaded to restaurant_inspections
234 •   SELECT r.RestaurantID, r.RestaurantName, i.*
235         FROM Restaurants r, restaurant_inspections i
236         WHERE r.RestaurantID = i.CAMIS AND camis = 1;
237
```

100% 23:235

Result Grid Filter Rows: Search Export:

RestaurantID	RestaurantName	CAMIS	INSPECTION DATE	ACTION	VIOLATION CODE	VIOLATION DESCRIPTION	CRITICAL FLAG	S
1	NON-JUICY LUCY's	1	07/31/2025	Violations were cited in the following area(s).	03I	Juice packaged on premises with no or incomplete label, no warni...	Critical	1

Change of RestaurantID cascaded to restaurant_inspections CAMIS due to Foreign Key constraint set to CASCADE changes

Deletes to Restaurants table cascades to restaurant_inspections table with FK

```
240     -- check delete from Restaurant cascades to delete from restaurant_inspections
241 •   DELETE FROM Restaurants
242         WHERE RestaurantID = 1;
243
244     -- confirm delete in Restaurants
245 •   SELECT * FROM Restaurants WHERE RestaurantID = 1; -- no results
246
```

Delete Lucy's from Restaurant and confirm deleted ON DELETE of Foreign Key set to CASCADE

100% 64:245

Result Grid Filter Rows: Search Edit: Export/Import:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude	Longitude	CuisineDescripti...
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
247     -- confirm delete cascaded to restaurant_inspections
248 •   SELECT * FROM restaurant_inspections WHERE CAMIS = 1; -- no results
249
```

100% 68:248

Result Grid Filter Rows: Search Export:

CAMIS	INSPECTION DATE	ACTION	VIOLATION CODE	VIOLATION DESCRIPTION	CRITICAL FLAG	SCORE	GRADE	GRADE DATE	RECORD DATE

Entries for Lucy's in restaurant_inspections also deleted due to Foreign Key delete cascade