Dealing with Data, Class 5

Spring, 2020

Why Databases?

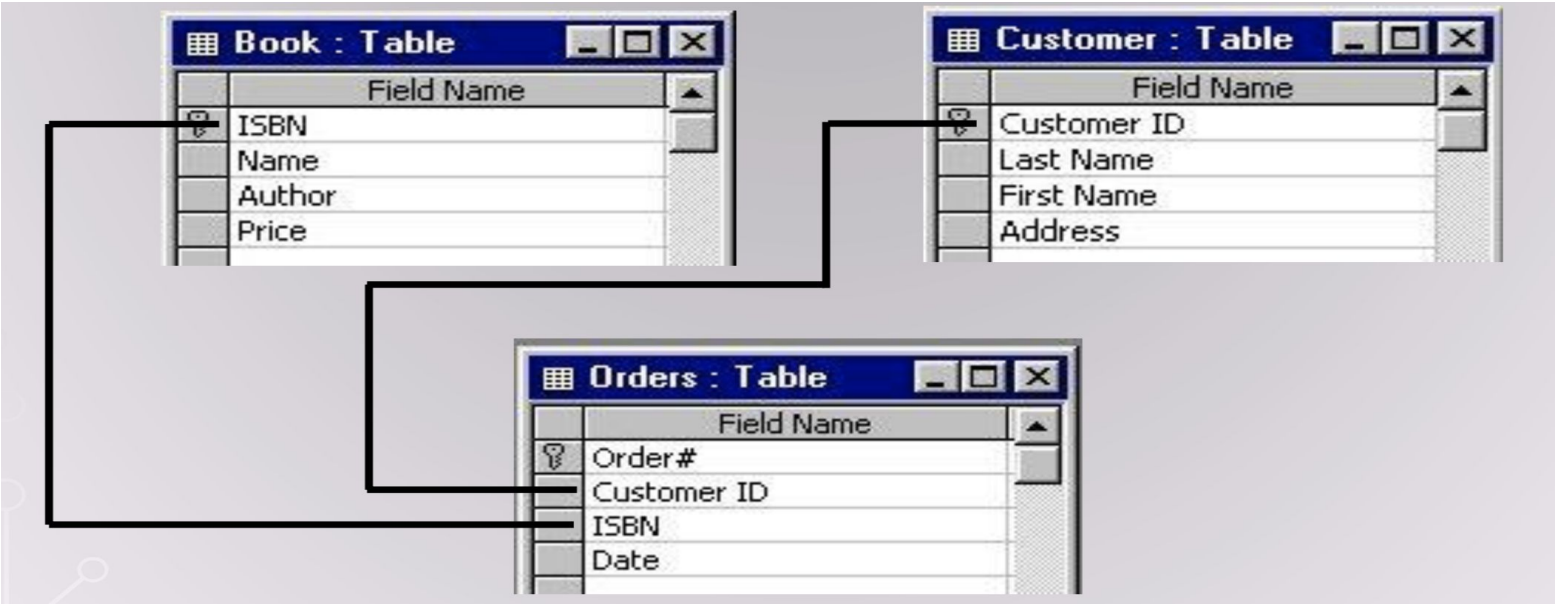
- Insertions, Deletions, and Updates

E.g.: If we update Jeff Bezo's address in Order #1, Orders #6 and #8 will remain unchanged

Order#	Date	Customer ID	Last Name	First Name	Address	ISBN	Book Name	Author	Price
1	9/1/03	C1001	Bezos	Jeff	1 Amazon Plaza	#0465039138	Code and other laws of cyberspace	Lessig, Lawrence	$25.00
2	9/2/03	C1004	Sproull	Lee	Dean's Office, Stern School, New York	#1573928895	Digital Copyright: Protecting Intellectual Property on the Internet	Litman, Jessica	$55.00
3	9/3/03	C1002	Student	Pat	Tisch LC-12, New York	#0072952849	MIS in the Information Age	Haag, Stephen	$98.75
4	9/4/03	C1003	Gates	Bill	Microsoft Corporation, Redmond	#0738206679	Linked: The New Science of Networks	Barabasi, Albert-Laszlo	$34.95
5	9/5/03	C1003	Gates	Bill	Microsoft Corporation, Redmond	#0738206083	Smart Mobs: The Next Social Revolution	Rheingold, Howard	$29.95
6	9/6/03	C1001	Bezos	Jeff	1 Amazon Plaza	#0738206083	Smart Mobs: The Next Social Revolution	Rheingold, Howard	$29.95
7	9/7/03	C1002	Student	Pat	Tisch LC-12, New York	#1573928895	Digital Copyright: Protecting Intellectual Property on the Internet	Litman, Jessica	$55.00
8	9/8/03	C1001	Bezos	Jeff	1 Amazon Plaza	#0738206083	Smart Mobs: The Next Social Revolution	Rheingold, Howard	$29.95

ERD (Entity Relationship Diagrams)

- All data are stored in tables, and each cell contains a single value; This prevents insertion, deletion, and update anomalies



Relationships Between Tables

An **entity** is a collection of objects with the same properties:

- Students (name, id, age, sex)
- Courses (course id, section id, description, location)

Entities are descriptions of **instances**:

- An instance of entity “Students” might be: (“John Doe”, “N1832”, “25”, “M”)

Relationships Between Tables

An **primary key** is an attribute whose value is unique in each instance:

- E.g.: **Student ID** in the **Students** entity

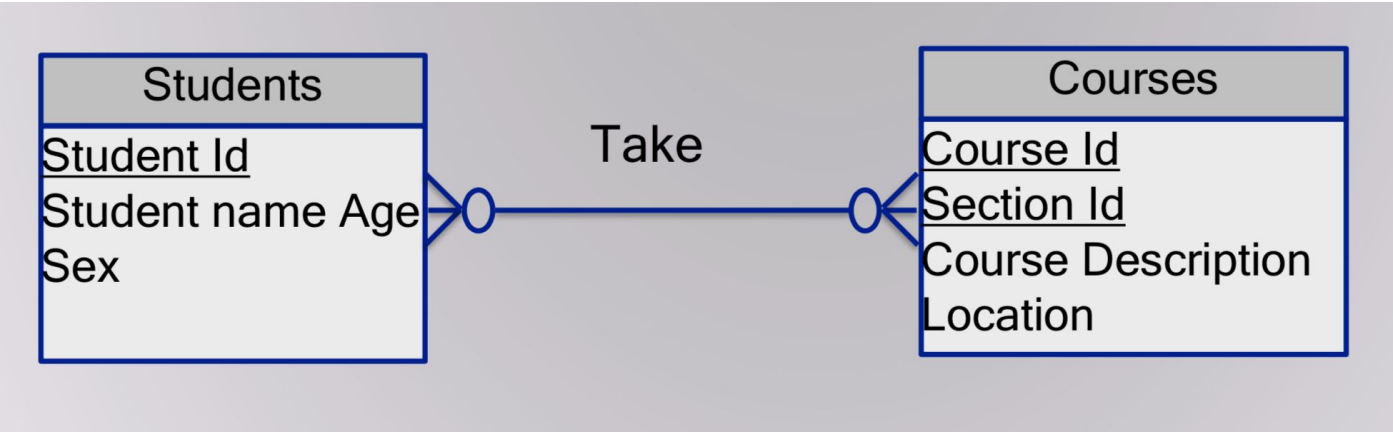
A **Composite Primary Key** is a primary key that consists of two or more attributes, whose values together (but not separately) are unique for each instance of an entity:

- E.g.: **Course ID** and **Section ID** in the **Courses** entity

Relationships Between Tables

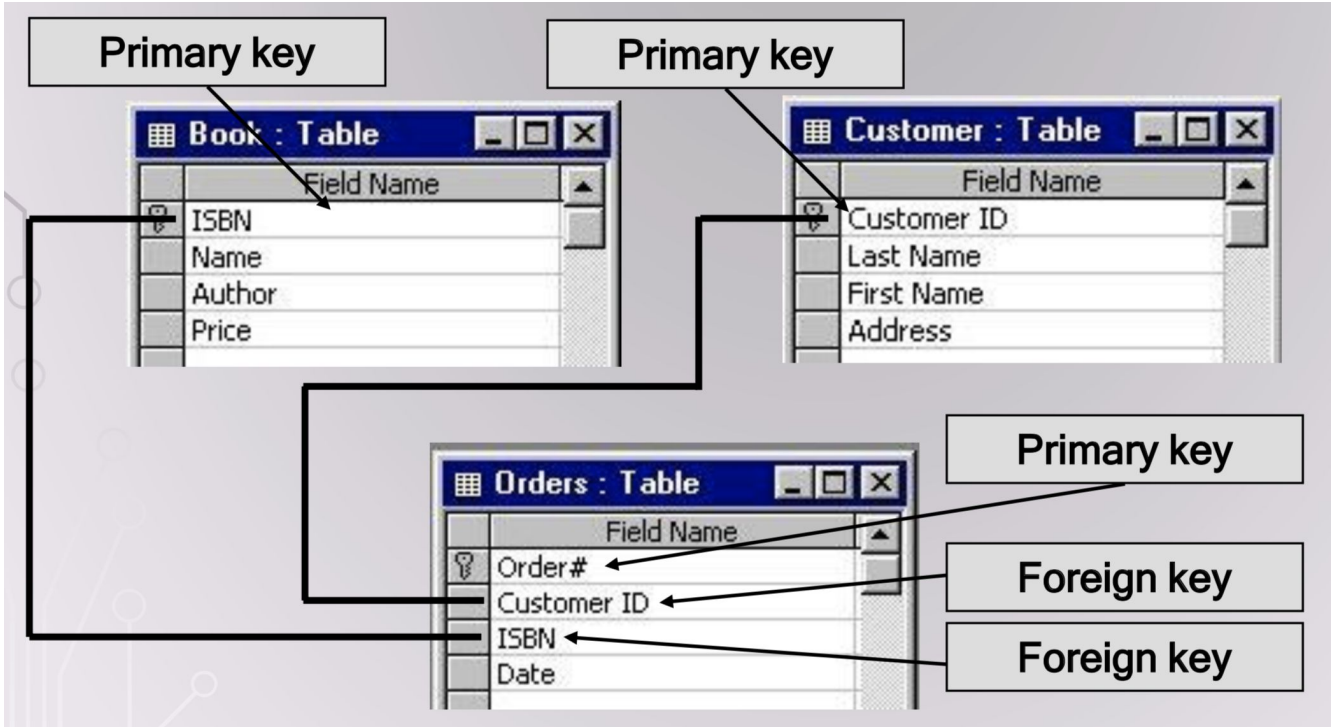
An **relationship** is an association among entities:

- E.g.: **Students** take **Courses**



Relationships are shown as a line connecting associated entities, labeled with the name of the relationship

Relationships Between Tables

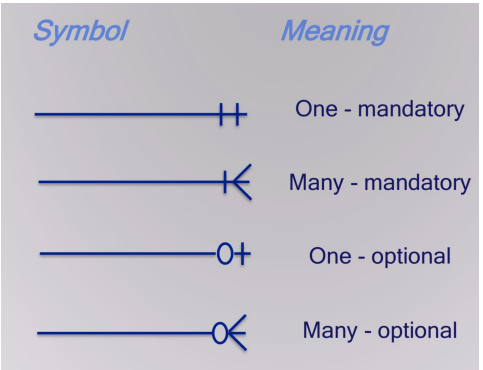


Relationships Between Tables

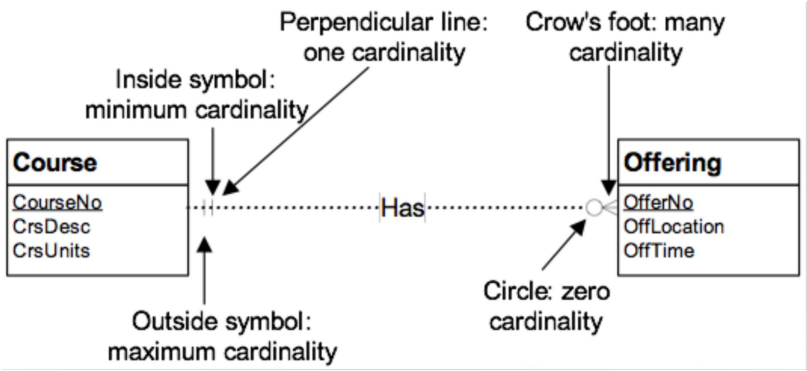
Cardinalities describe the number of instances that participate in a relationship:

- E.g.: A student may take 0, 1, or more (“many”) courses.
- A course can be taken by 0, 1, or more (“many”) students.

Cardinality Notation:



Cardinality Notation Example



- The “1:1” next to “Course” means “each offering has at least one, and at most one, Course”
- The “0-many” next to “Offering” means “each course may have zero offerings and may have many offerings”

From Narrative to ERD to Tables

1. Identify Entities and Attributes
2. Define the Primary Keys
3. Identify Relationships and Determine Cardinalities
4. Refine and Iterate

E.g.: You are designing an ERD for a Water Utility Company.

- Each customer has a number, name, billing address, type, applicable rate, and meter collection
- The available meter data includes number, address, size, and model
- An employee periodically reads each meter, and each reading includes a number, timestamp, consumption level, and employee number
- Bills are based on the most recent meter readings and applicable rates
- The rate includes a number, description, fixed and variable dollar amounts, and consumption threshold

From Narrative to ERD to Tables

Identifying Entities:

- Find nouns that describe people, places, things, and events

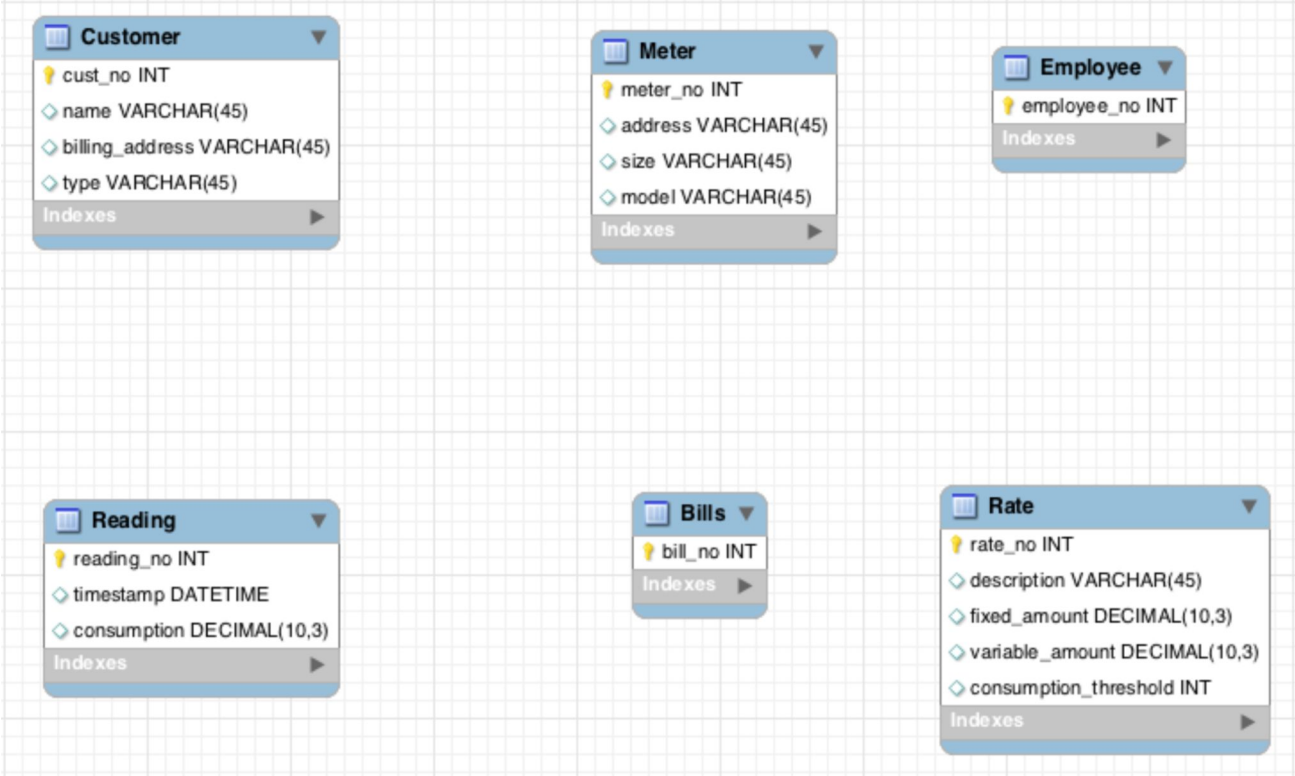
Identifying Attributes:

- Look for details about the entities; Consider something an attribute unless it contains other details

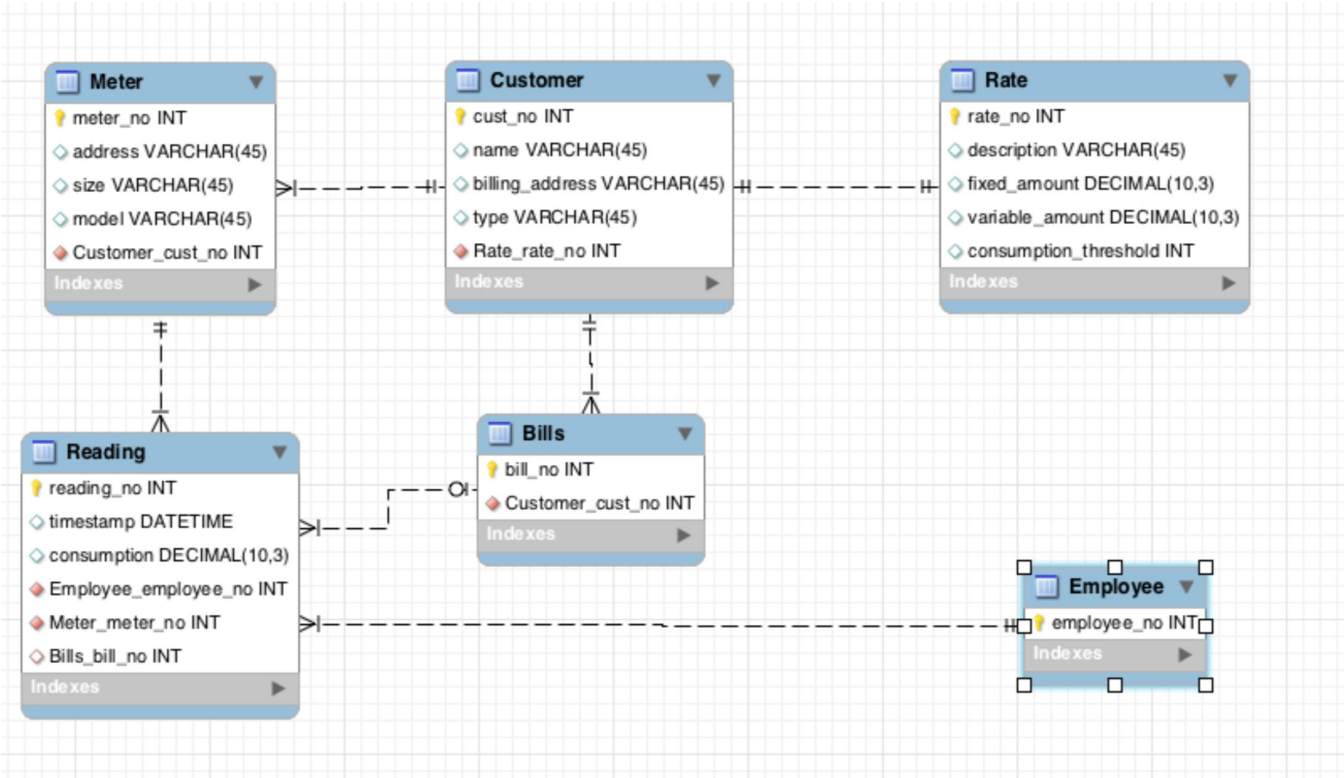
Determine Primary Keys:

- PK’s should be stable, and each entity should only have one PK

From Narrative to ERD to Tables



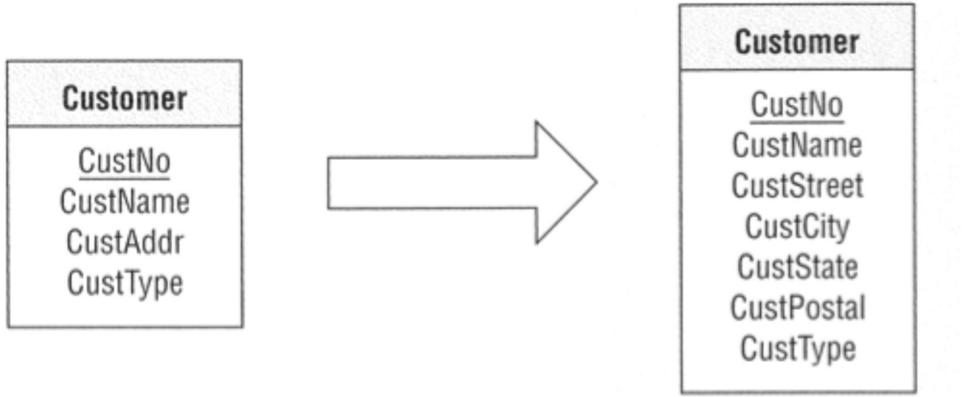
From Narrative to ERD to Tables



A Note on Splitting Attributes

Often times you will end up with **compound attributes**, and need to decide if they are better off as individual entities.

For instance, **Customer Address** is a compound attribute. What happens if we need to search by city?

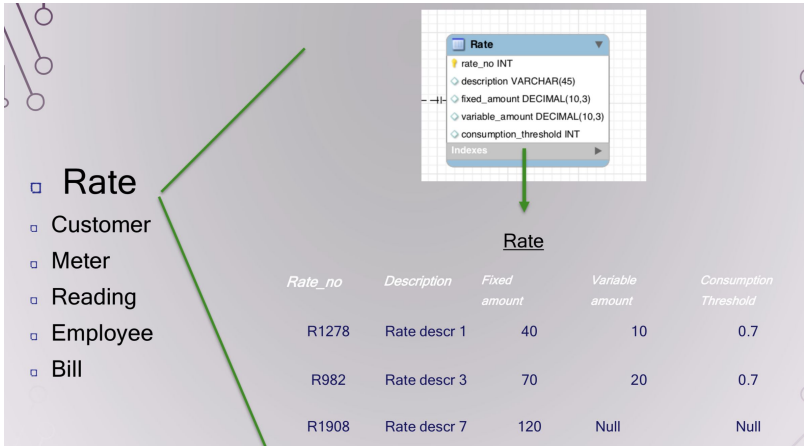


From Narrative to ERD to Tables

Each **entity** matches to a table

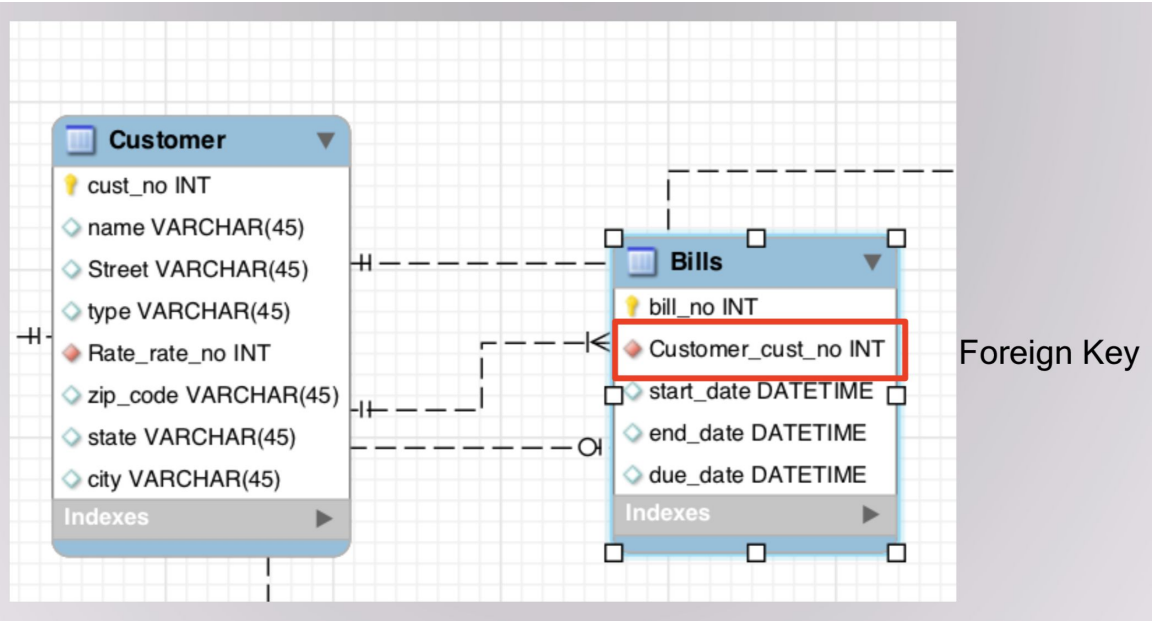
Each **attribute** maps to a column in the table

The **primary key** of the entity maps to the **primary key** of the table



From Narrative to ERD to Tables

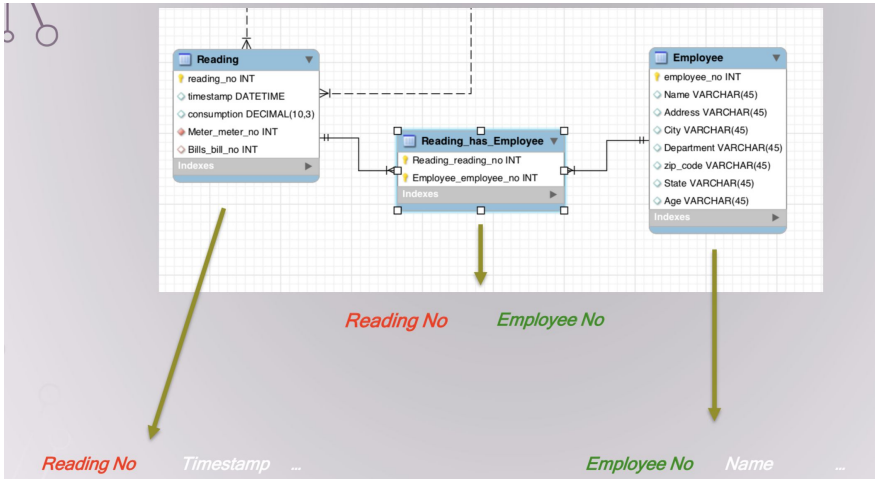
For each One-to-Many relationship, we add a **Foreign Key** to the table corresponding to the “many” entity



From Narrative to ERD to Tables

With Many-to-Many relationships, we need to introduce a **Bridge Table** (each many-to-many relationship becomes its own table). The **primary key** of the **bridge table** is the combination of the primary keys of the entity types participating in the relationship. Each of the primary keys stored in the bridge table is a foreign key, pointing to the original entity table.

E.g., a reading can be done by many employees:



ER, Summarized

- Popular Framework for Conceptual Design
  - Constructs are expressive, intuitive, and graphical

- ER Modeling is subjective
  - There are often multiple ways to model a given scenario
  - Analyzing alternatives is key

- ER Modeling is iterative
  - Any resulting diagram should be further analyzed and refined

Exploring Balanced Media Representation

- Users: Commissioner for NYC's Media and Entertainment Agency
- Decision Problem: Daily decisions reviewing film permit requests for locations around NYC now need to consider how they can create a balanced media representation of NYC's diversity
- Data Assistance Required: Describe neighborhood characteristics of past film permits to inform future decisions

This is a **descriptive** application; the agency needs to start with an understanding of the neighborhoods shown in previously issued permits

App Style: TBD (Charts? Graphs? Tables?)

Database Design

Ideal:

1. Identify **entities** and **attributes**
2. Determine **primary keys**
3. Identify relationships
4. Determine relationship cardinalities
5. Refine the ERD

Pragmatic:

1. Inspect data to identify entities and attributes, and find gaps in available data
2. Determine if we can create primary keys
3. Identify ideal relationships and cardinalities, and map into real data
4. Sketch out the ERD

Our Data

- Neighborhood Film Permits: https://data.cityofnewyork.us/City-Government/Film-Permits/tg4x-b46p

- Neighborhood Resident Income: https://www.irs.gov/statistics/soi-tax-stats-individual-income-tax-statistics-zip-code-data-soi

- Neighborhood Resident Demographic Data: https://factfinder.census.gov/faces/affhelp/jsf/pages/metadata.xhtml?lang=en&type=dataset&id=dataset.en.DEC_10_SF1

Identifying Data Gaps

- Neighborhoods are within a Borough and can be located by Zip Code, containing buildings and their resident businesses and people.
- Filming projects are for a specific medium (TV, movies), take place in a Neighborhood for a specific time period, and require a permit from the city. Neighborhood buildings and residents are shown in the results.
- Neighborhood Resident Demographics describe their diverse families and living accommodations.
- Neighborhood Resident Tax Returns describe their incomes.

Identifying Primary Keys

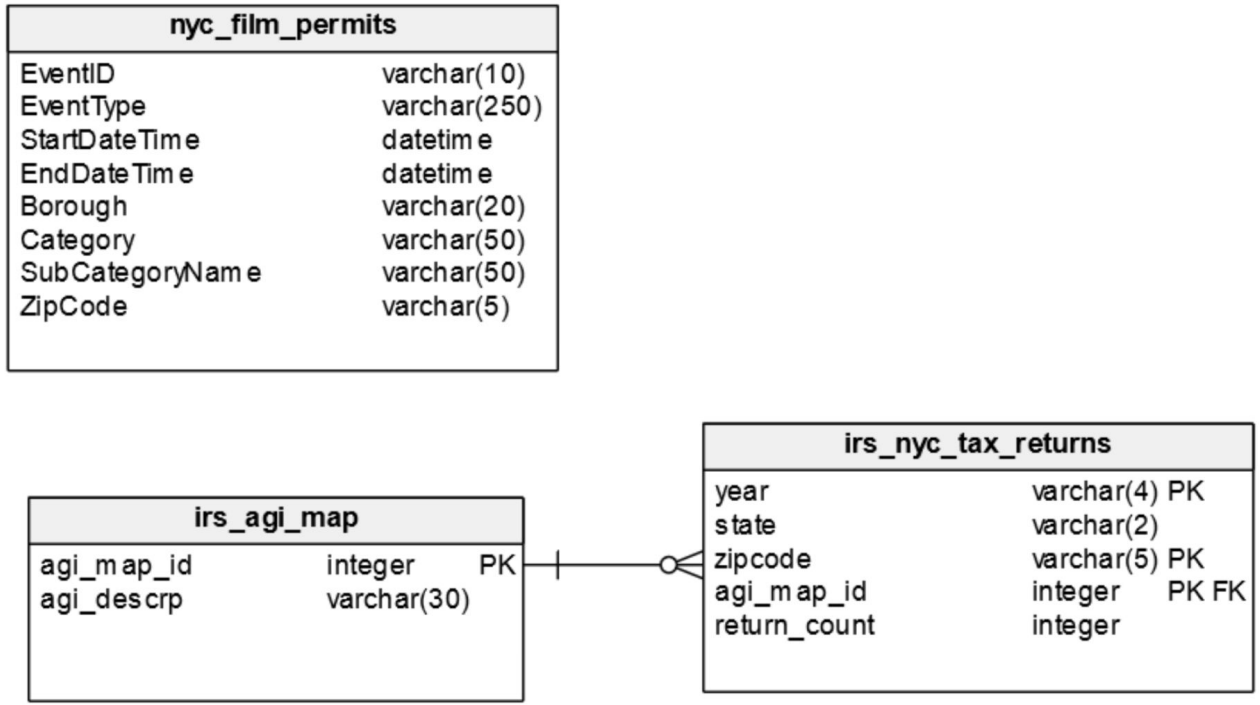
- IRS Adjusted Gross Income Buckets already have a **Primary Key**
- IRS Tax Returns have a few potential, **Composed Primary Keys** (Year, Zip Code, AGI Bucket)

- What about file permits? Resident demographics? (More on that in a bit)

Relationships and Cardinalities

- ONE neighborhood is identified by ONE zip code
- ONE film project can span MANY neighborhoods
- ONE neighborhood can host MANY film projects
- ONE neighborhood has MANY resident income levels and MANY demographic attributes
- ONE income level can be found in MANY neighborhoods
- ONE demographic attribute can be found in MANY neighborhoods

ERD Sketch

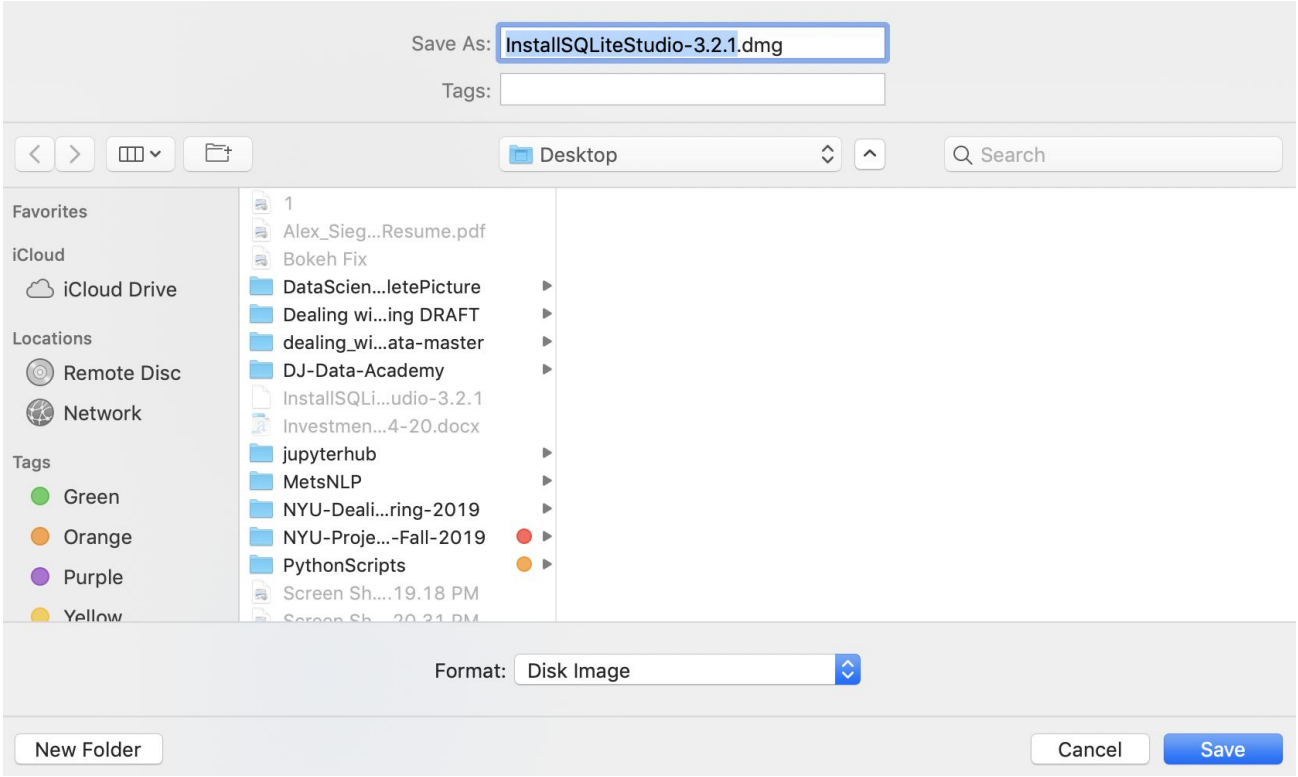

nyc_film_permits
EventID varchar(10)
EventType varchar(250)
StartDateTime datetime
EndDateTime datetime
Borough varchar(20)
Category varchar(50)
SubCategoryName varchar(50)
ZipCode varchar(5)
irs_agi_map
agi_map_id integer PK
agi_descrp varchar(30)
irs_nyc_tax_returns
year varchar(4) PK
state varchar(2)
zipcode varchar(5) PK
agi_map_id integer PK FK
return_count integer

SQLite Studio Download

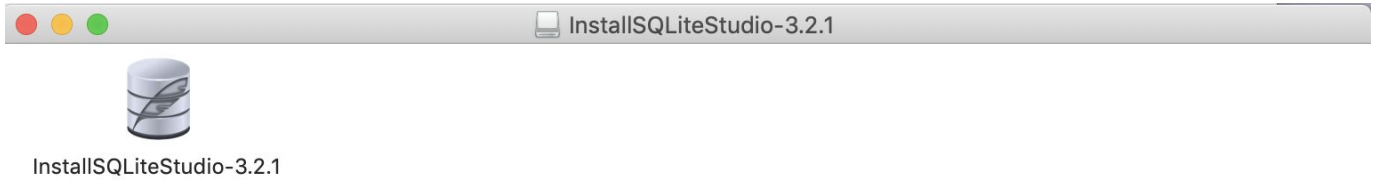
https://github.com/pawelsalawa/sqlitestudio/releases

**Windows: Select "Standard Installer for 32-bit"*

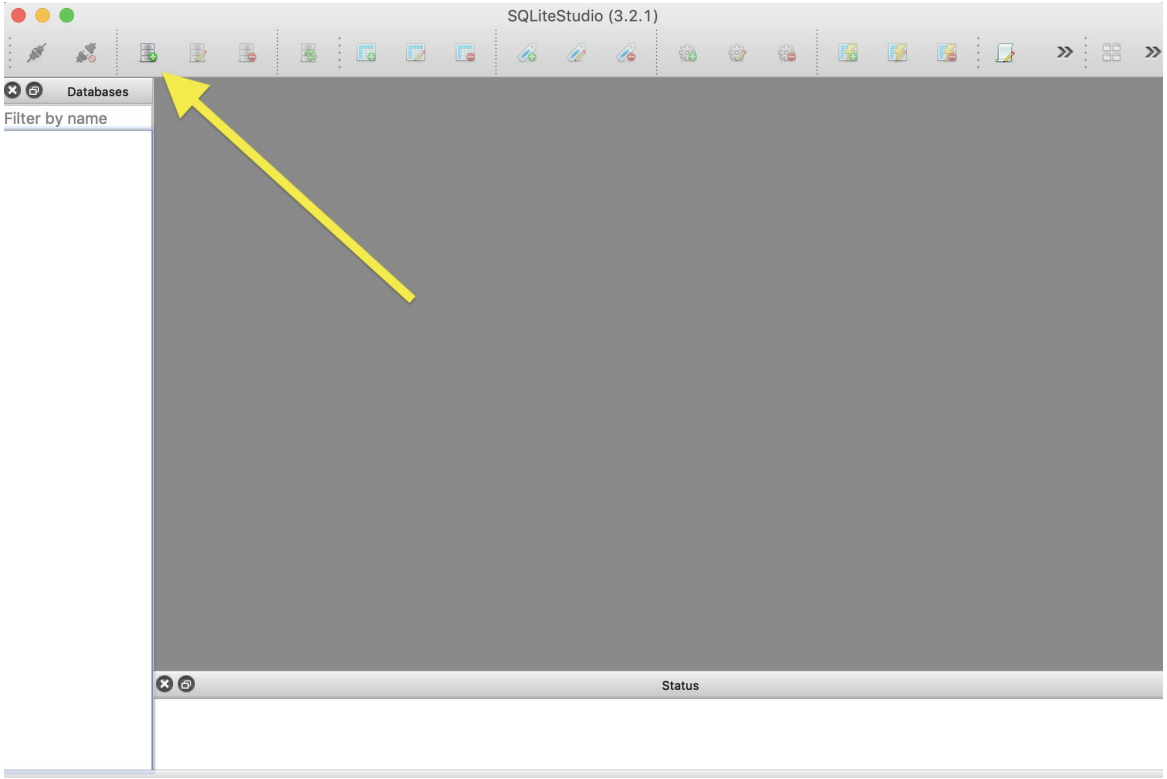
SQLite Download



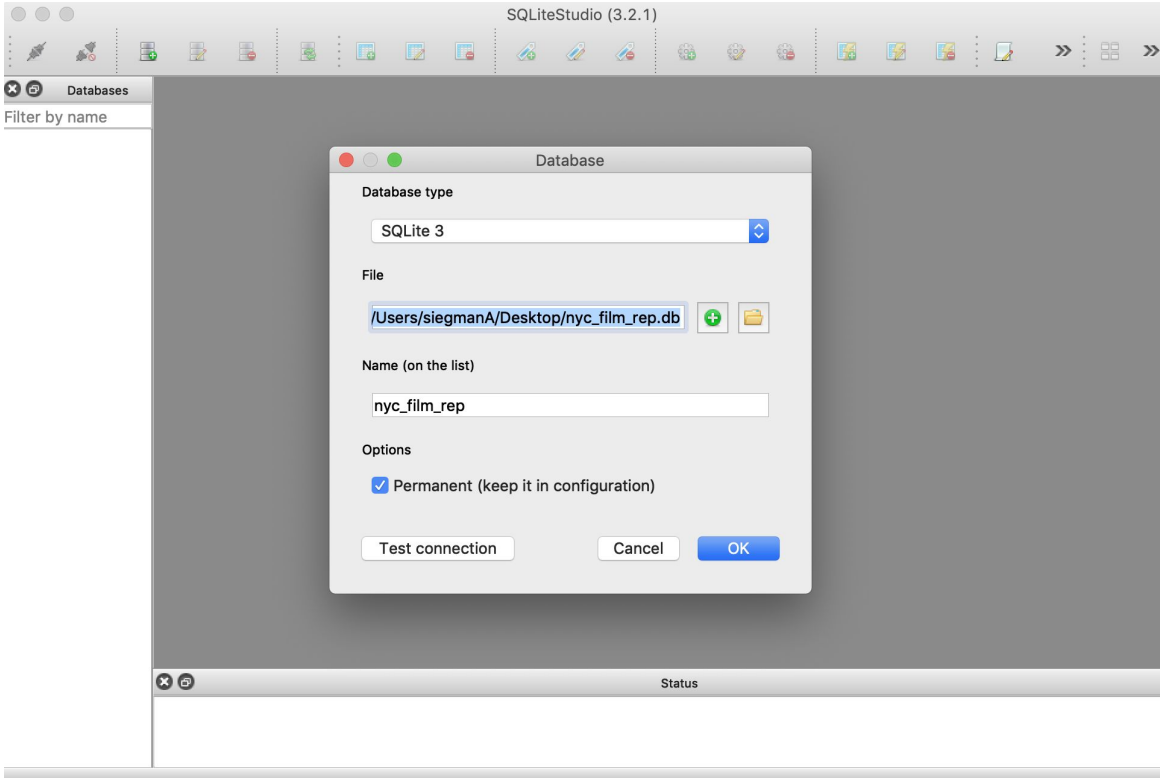
SQLite Download



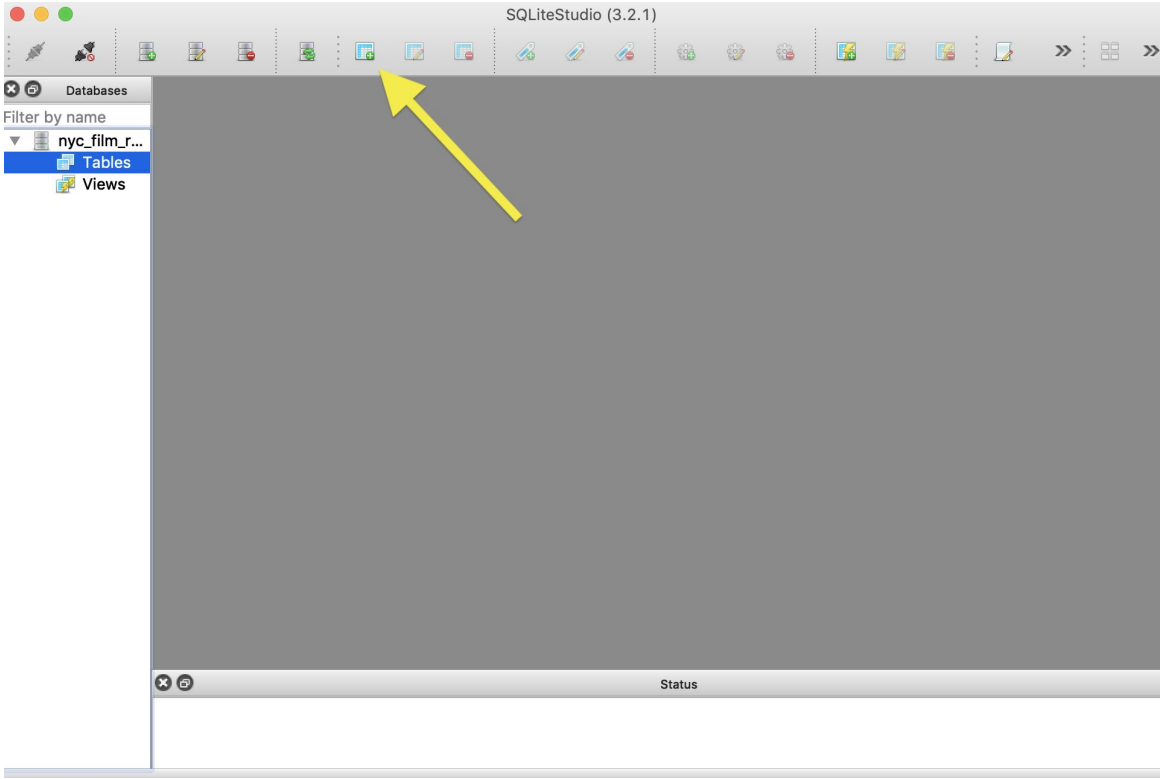
Creating a Database



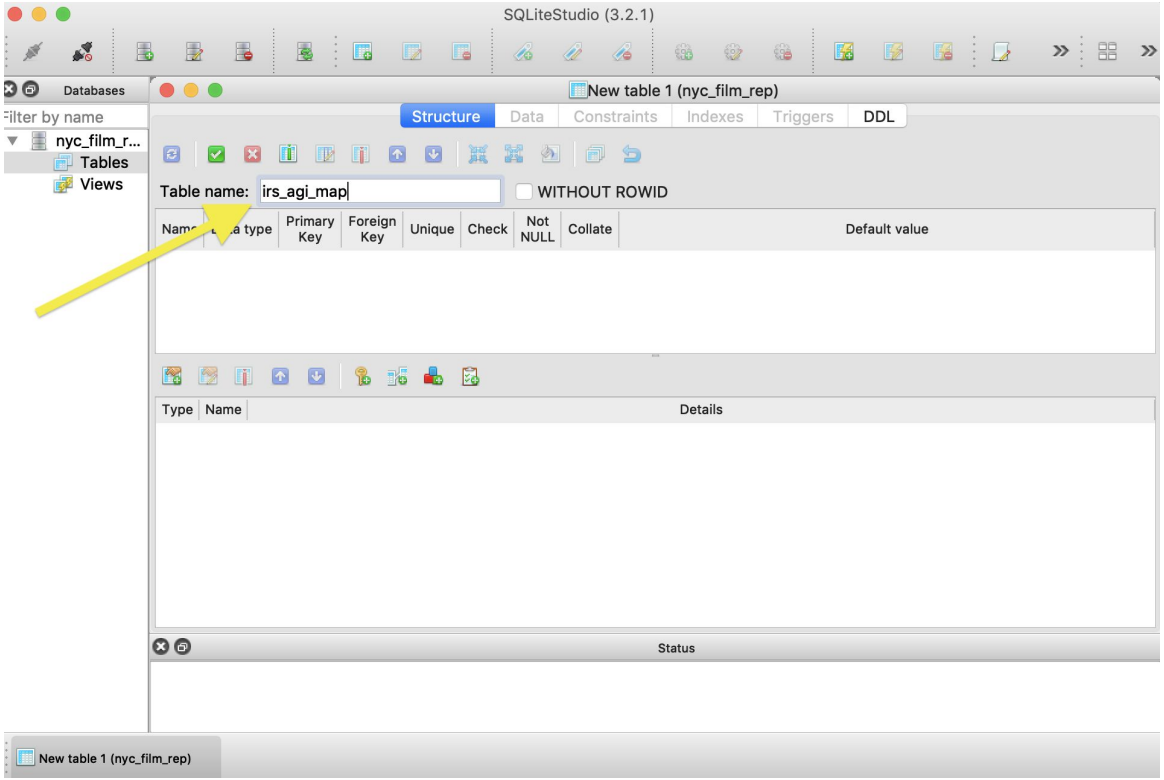
Creating a Database



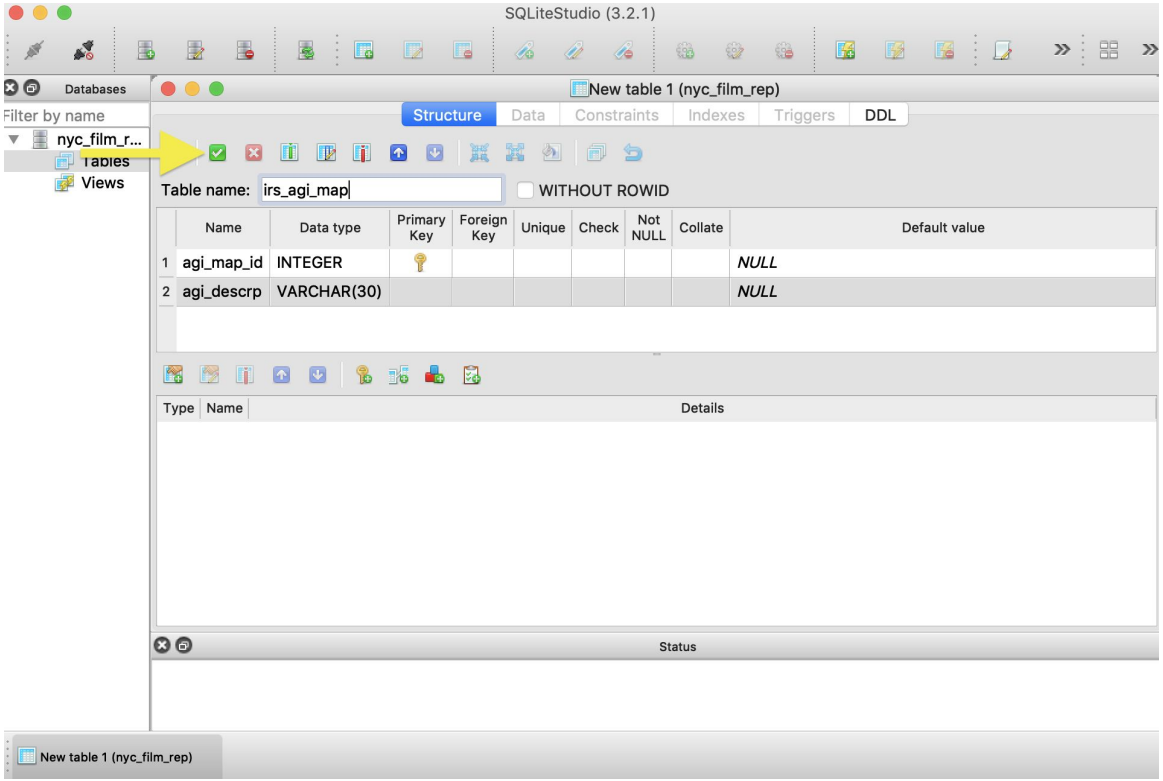
Creating a Table



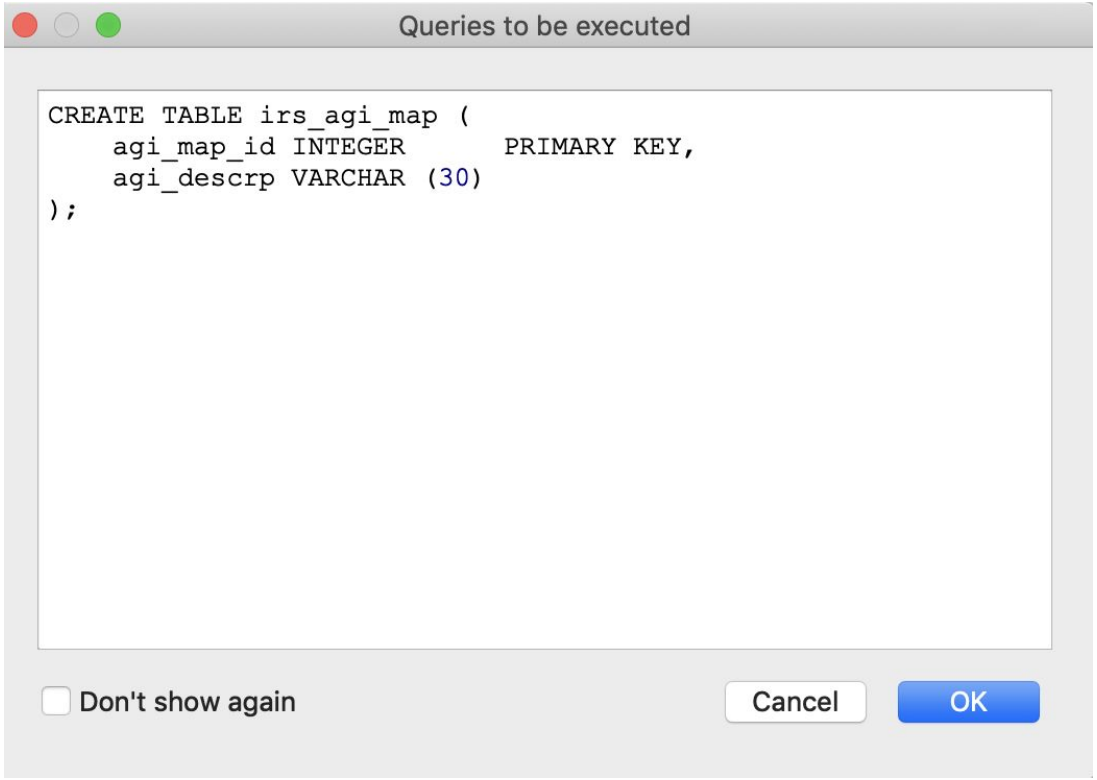
Creating a Table



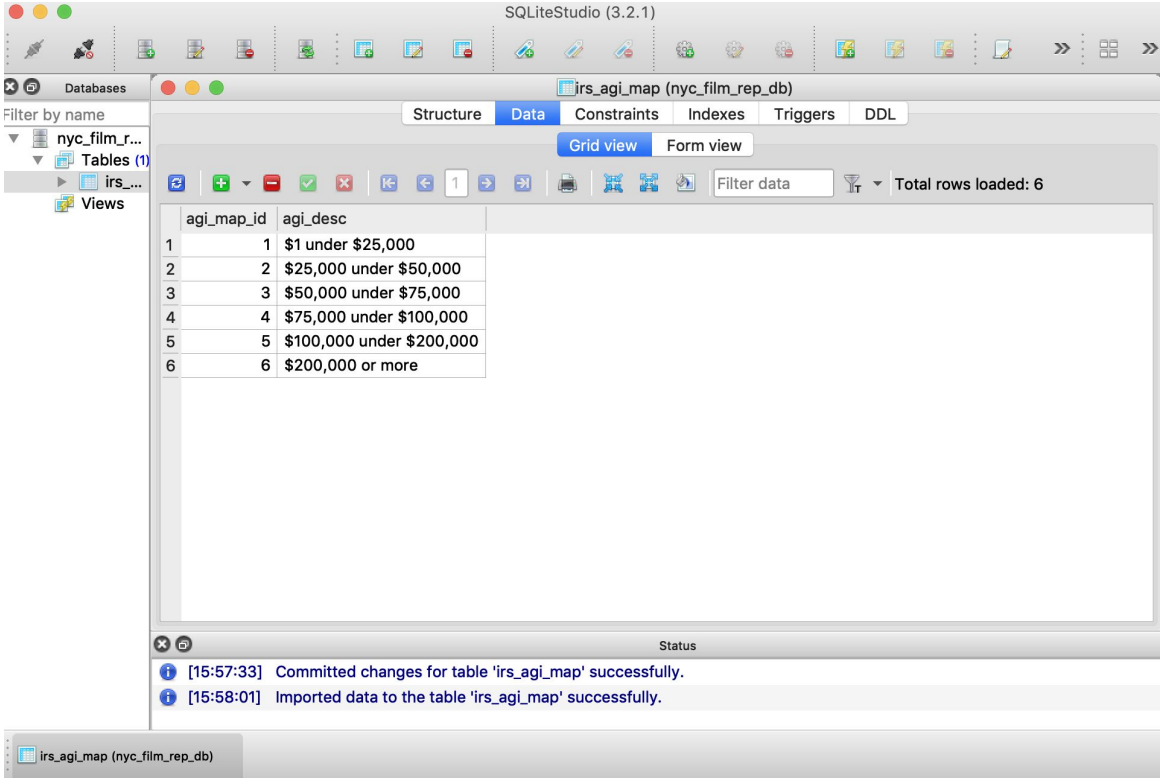
Creating a Table



Creating a Table



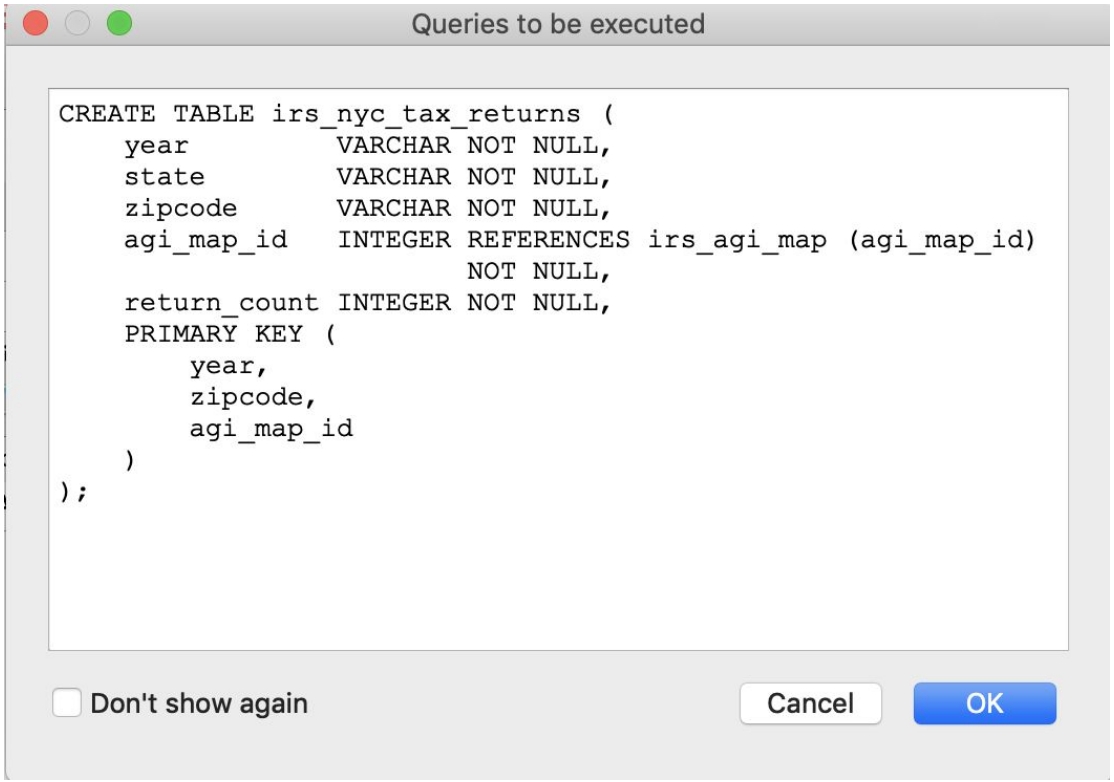
Creating a Table



Creating a Table – IRS Tax Data Exercise

...

Creating a Table – IRS Tax Data Exercise



Executing SQL Queries

(tools > open SQL editor)

Buttons on toolbar

- 1st button ( ) executes query the you typed in the query text field below.
- 2nd button ( ) executes `EXPLAIN` statement for the query below.
- 3rd button ( ) uses currently configured SqlFormatterPlugin to format queries typed below.
- 4th button ( ) clears query execution history (available in the last tab of the SQL Editor window).
- 5th button ( ) opens Export_dialog for exporting results from query typed below.
- 6th button ( ) creates view from the `SELECT` query typed below.
- 7th button ( ) saves contents of the query text field below into the file.
- 8th button ( ) loads contents of selected file into the query edit field below.
- 9th position is a combo box, where you can pick current working database for the SQL Editor window. All queries are executed on database selected in this combobox.
- 10th and 11th buttons ( ) are a shortcut to configure SQL Editor results presentation mode - in separate tab, or below the query field.

SELECT Queries

SELECT A1, A2 ← columns to return
FROM *TableName*

SELECT Queries - ‘AS’

- Sometimes we want to rename a column to provide a more descriptive name in the results

SELECT A1 AS *Name1*, A2 AS *Name2* ← columns to return with new naming conventions
FROM *TableName*

Practice Queries

```
SELECT year,state,zipcode,return_count as tax_returns
FROM irs_nyc_tax_returns;
```

```
SELECT year,state,return_count as ‘tax returns’
FROM irs_nyc_tax_returns;
```

SELECT Queries - ‘DISTINCT’

- Used to eliminate duplicates in the results

SELECT DISTINCT A1, A2, A3,...,A100
FROM *TableName*

Practice Queries

```
SELECT DISTINCT year
FROM irs_nyc_tax_returns;
```

```
SELECT DISTINCT year, zipcode as ‘zip code’
FROM irs_nyc_tax_returns;
```

SELECT Queries - ‘ORDER BY’ and ‘LIMIT’

- **ORDER BY** is used to sort the result rows based on attribute values
- **LIMIT** limits the number of rows in the result

SELECT DISTINCT A1, A2,A3
FROM *TableName*
ORDER BY A1[ASC|DESC],A2[ASC|DESC]
LIMIT N;

Practice Queries

```
SELECT DISTINCT year,zipcode AS ‘zip code’
FROM irs_nyc_tax_returns
ORDER BY year DESC
LIMIT 5;
```

SELECT Queries - ‘WHERE’

- ‘WHERE” defines which rows will appear in the results

SELECT A1, A2, A3,...,A100
FROM *T1*
WHERE condition
ORDER BY A1[ASC|DESC],A2[ASC|DECS];

Conditions for WHERE clauses

Condition	Description	Example
attr = 'text'	Equality comparison for a textual attribute	gender = 'Male'
attr = number	Equality comparison for a numeric attribute	year = 2006
attr <> value attr != value	Attribute is not equal to value	genre <> 'Drama' genre != 'Drama'
attr > value	Attribute is greater than value	rating > 7.8
attr < value	Attribute is smaller than value	year < 1900
attr >= value	Attribute is equal or greater than value	year >= 1900
attr <= value	Attribute is equal or smaller than value	year <=1900
attr IN (x1, x2, x3, …)	Attribute value is either x1, or x2 or x3, or …	genre IN ('Drama', 'Comedy')
attr NOT IN (x1, x2, x3, …)	Attribute value is not x1, nor x2, nor x3, …	genre NOT IN ('Adult', 'War')
cond1 AND cond2	Both conditions should hold	
cond2 OR cond2	At least one of the conditions should hold	

Practice Queries

```
SELECT *
FROM irs_nyc_tax_returns WHERE (zipcode = "10001") ;

SELECT *
FROM irs_nyc_tax_returns
WHERE (zipcode = 10001 AND return_count > 4000) OR (diff_agi_map_id = 2 and year = 14);

SELECT return_count
FROM irs_nyc_tax_returns
WHERE zipcode IN ("10128") AND year IN ("12");
```

The NULL Value

- When a column has no value, it's assigned a NULL value, a special way SQL handles the "empty" value

- To check if something is null, you use **"attr IS NULL"**, e.g.:

```
SELECT ProfileID
FROM Profiles
WHERE Birthday IS NULL
```

Comments

- SQL supports two commenting styles:
  - --
  - /* */

-- permits one line of comment

/ permits several lines of comments */*

Comparison Operators

Operator	*Description*
=	equals
<>	is not equal to
!=	is not equal to
<	less than
>	greater than
AND	logical and
OR	logical or
NOT	logical not

Other Operators

SQL	*Description*
as	used to change the name of a column in the result
distinct	no duplicate rows
order by column(s)	sorts by column(s) in ascending order
order by .. desc	sorts by column(s) in descending order
*	select all columns
like ‘%pattern_’	$: any sequence of characters _: any single character
attribute is null	rows that have null values for the specific attribute
is not null	rows that have not null values for the specific attribute
between this and that	between **this** value and **that** value
in	set membership
limit n	fetches only the top n rows from the database