DEALING WITH DATA

SPRING 2019: INFO-GB.2346.30

PROFESSOR GUTHRIE COLLIN

TEACHING FELLOW AJINKYA WALIMBE

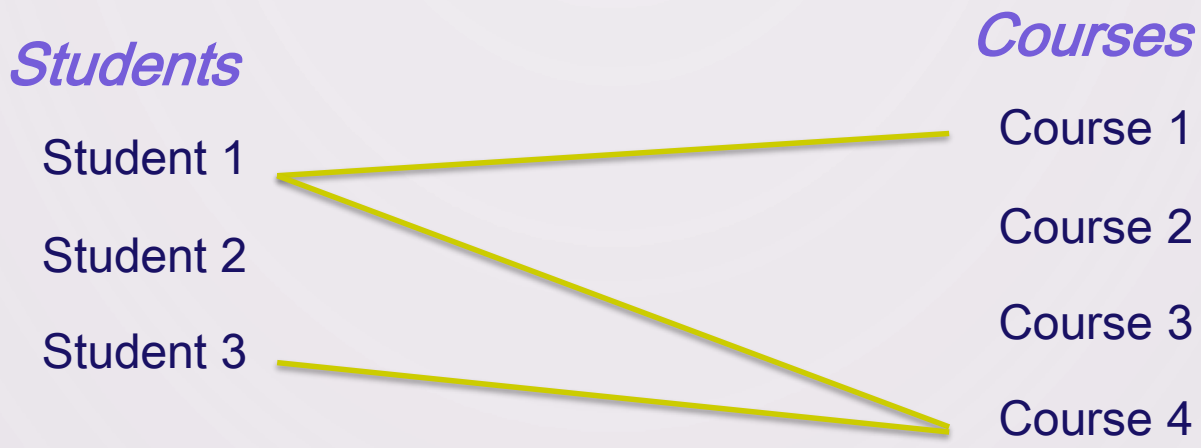
CLASS 5: SQL

MARCH 14, 2019

CONTENTS

Relationship Cardinalities

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

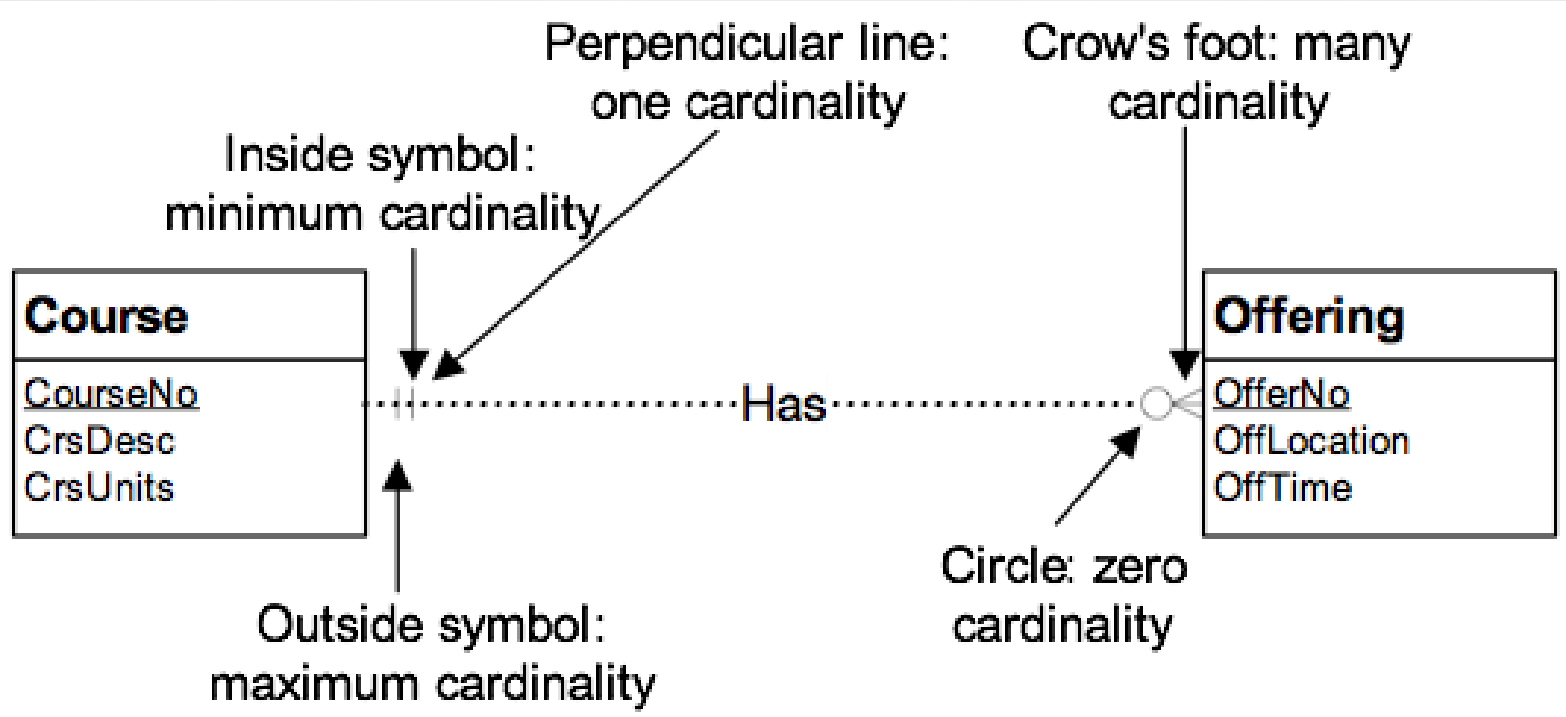


A student can take 0,1, or more courses (many). A course can be taken by 0,1, or more students (many). This example, the relationship has a **many to many** cardinality.

Cardinality Notation

Symbol	*Meaning*
	One - mandatory
	Many - mandatory
	One - optional
	Many - optional

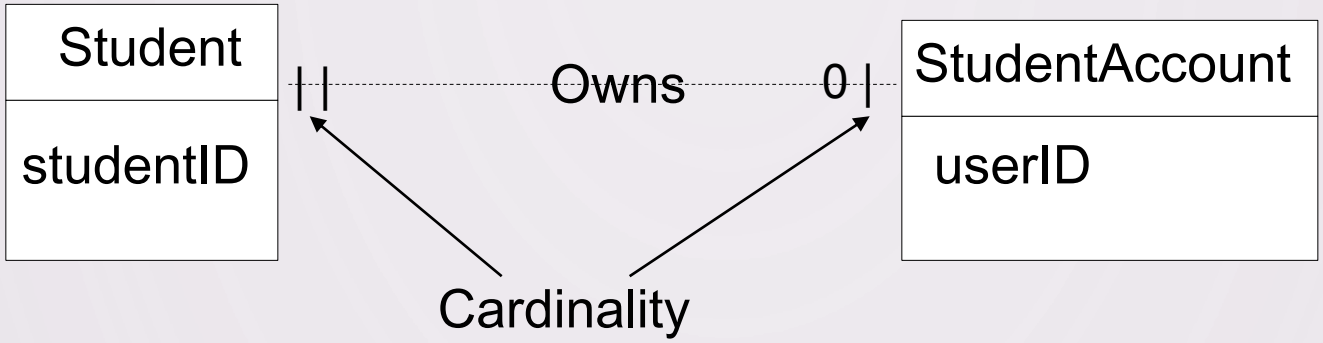
Cardinality Notation Example



Notice that

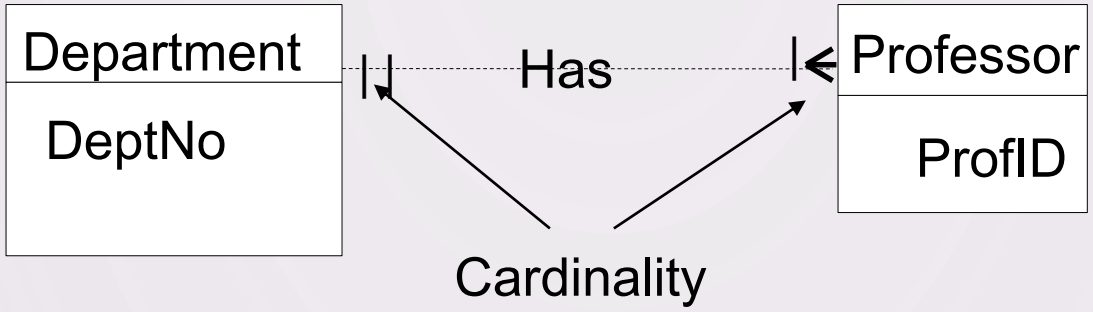
- The “1-1” cardinality next to the Course means that “Each Offering has at least one and at most one Course”
- The “0-many” cardinality next to the Offering means that “Each Course may have zero offerings and may have many offerings”

One-to-one Relationships



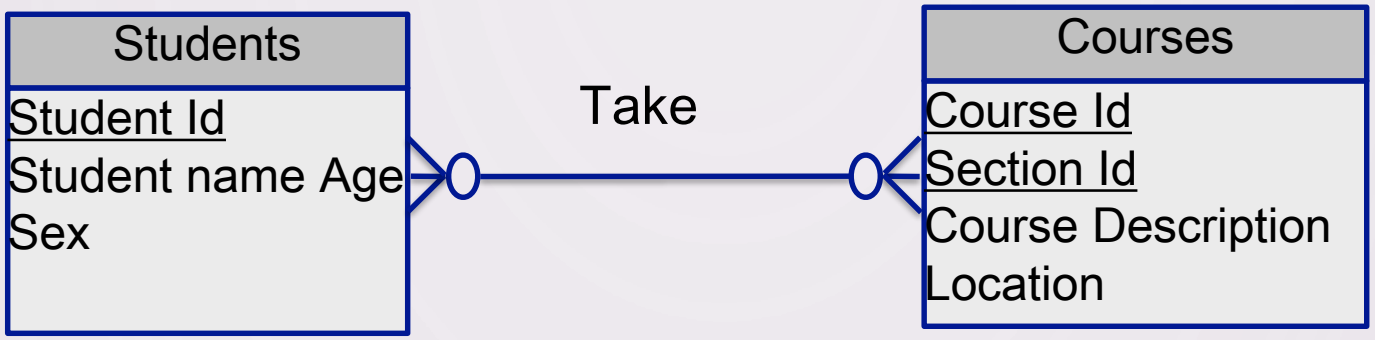
- The maximum number involved in a 1 to 1 relationship is one
  - Each student can own zero or one (expressed as “0 |”) account
  - Each account is owned by exactly one (expressed as “||”) student

One-to-many Relationships



- One-to-many , e.g., each department has many professors:
  - Each department can have one or more (expressed as “ |<”) professors
  - Each professor is affiliated with one and only one (expressed as “| |”) department.

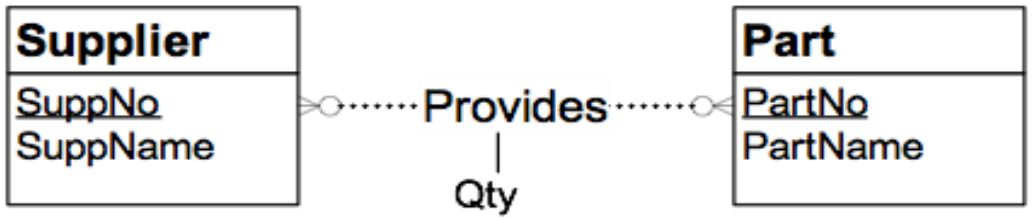
Many-to-Many Relationships



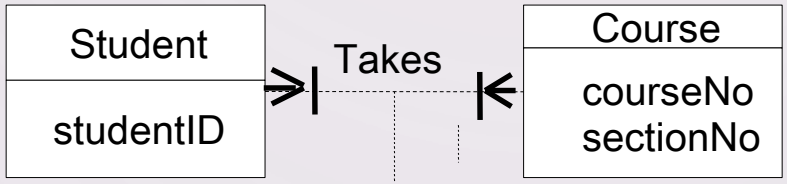
- Many-to-many, e.g., many students take many courses:
  - Each student can take 0 or more courses
  - Each course can enroll 0 to many students.

Relationships can also have attributes

a) *Provides* relationship



b) *Writes* relationship



registrationDate

Review

- An entity is a collection of objects with **the same properties (attributes)**:
  - Students (student name, student id, age, sex, etc.)
  - Courses (course id, section id, course description, location, etc..)

- A primary key is an attribute whose value is unique in each instance:
  - **Student id** in the students entity

- A relationship is an association among entities:
  - Students **take** courses (take is a relationship)

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

From a business narrative to an Entity - Relationship Model to a Relational Database

Questions

- How can we create an ER diagram from scratch?
- How can we go from an ER diagram to a design for a database?
- How can we use SQL to create the database in a relational database?

Outline

- Step-by-step procedure for converting narrative data into Entity-Relationship Diagram
- Application of the procedure for designing a DB for water-utility company
- Common problems during ER Design

From Narrative to ER diagram

- The procedure for analysis:
  - Identify entities and attributes
  - Determine primary keys
  - Identify relationships
  - Determine relationship cardinalities
  - Refine the ERD

Example: Water-Utility Database

- Customer database for a municipal water utility
- Business Narrative:
  - Customer has number, name, billing address, type, applicable rate, collection of meters.
  - Available meter data is number, address, size, model.
  - Employee periodically reads each meter. Reading has meter reading number, timestamp, consumption level, and employee number.
  - Bills are based on most recent meter readings and applicable rates.
  - Rate has rate number, description, fixed and variable dollar amounts, consumption threshold.

Step 1: Identify Entities and Attributes

- For entities, find nouns that describe people, places, things, and events
- For attributes, look for details about the entities
- Difficult decision: Attributes vs. Entities
  - Simplicity principal: consider as an attribute unless other details are presented.
- *Example : Should an address be an attribute or a separate entity? Advantages and disadvantages?*

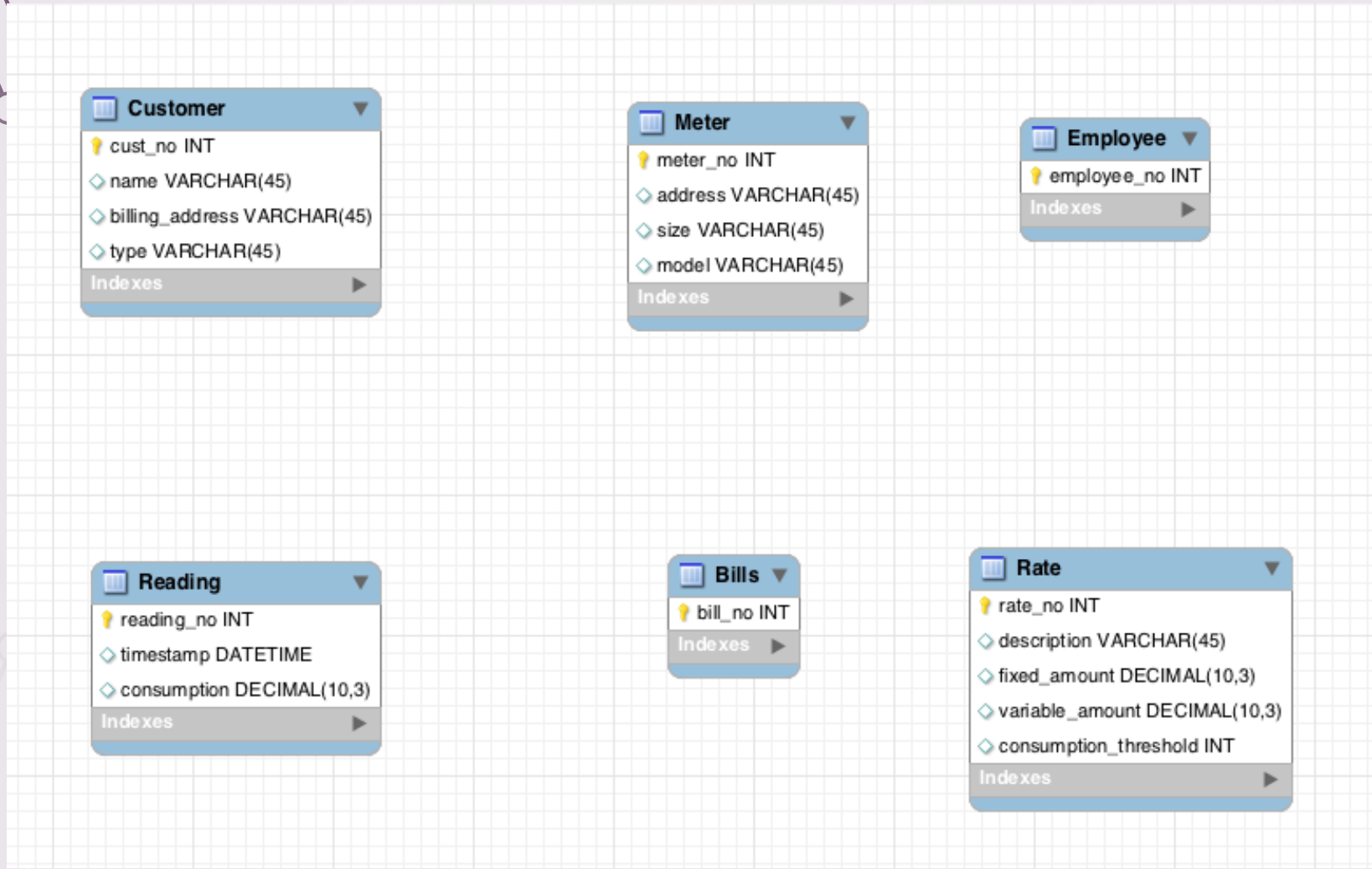
Step 2: Determine primary keys

- Stable: never change after assigned
- Each entity should have one and only one (good choice: automatically generated values, eg StudentId, CourseId)
- What about the following PKs for a person?
- *Credit card*
- *Phone number*
- *PassportID*
- *SSN*

Water utility DB: Entities & attributes

- Customer has number, name, billing address, type, applicable rate, collection of meters.
- Available meter data is number, address, size, model.
- Employee periodically reads each meter. Reading has meter reading number, timestamp, consumption level, and employee number.
- Bills are based on most recent meter readings and applicable rates.
- Rate has rate number, description, fixed and variable dollar amounts, consumption threshold.

Entities & Attributes



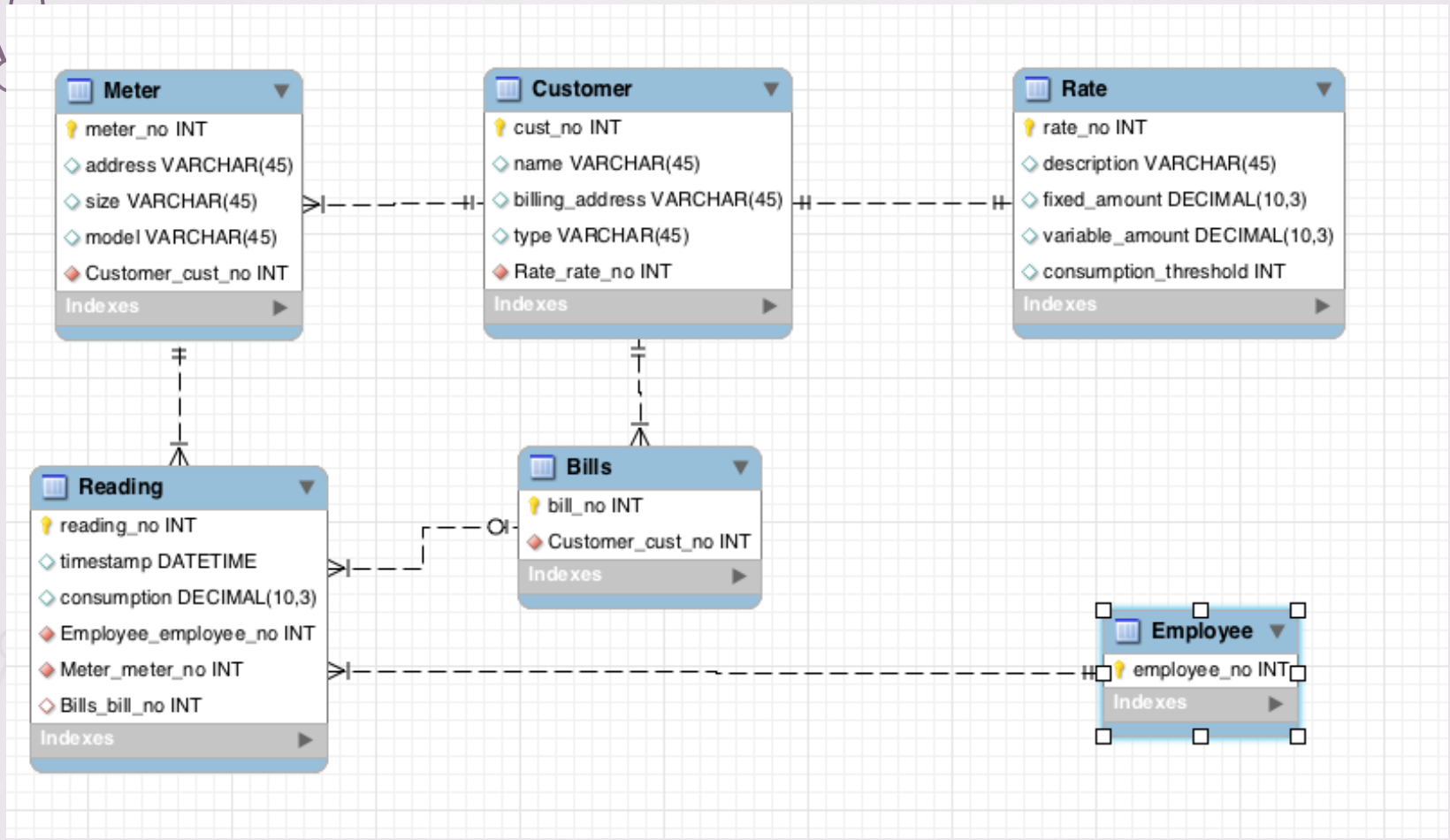
Step 3: Identify Relationships and determine cardinalities

- Identify relationships connecting previously identified entity types
  - Relationships = associations among nouns representing entity types
- Identify maximum cardinalities and minimum cardinalities

Identify Relationships

- Customer has number, name, billing address, type, applicable rate, collection of meters.
- Available meter data is number, address, size, model.
- Employee periodically reads each meter. Reading has meter reading number, timestamp, consumption level, and employee number.
- Bills are based on most recent meter readings and applicable rates.
- Rate has rate number, description, fixed and variable dollar amounts, consumption threshold.

Water Utility: Relationships & Cardinalities

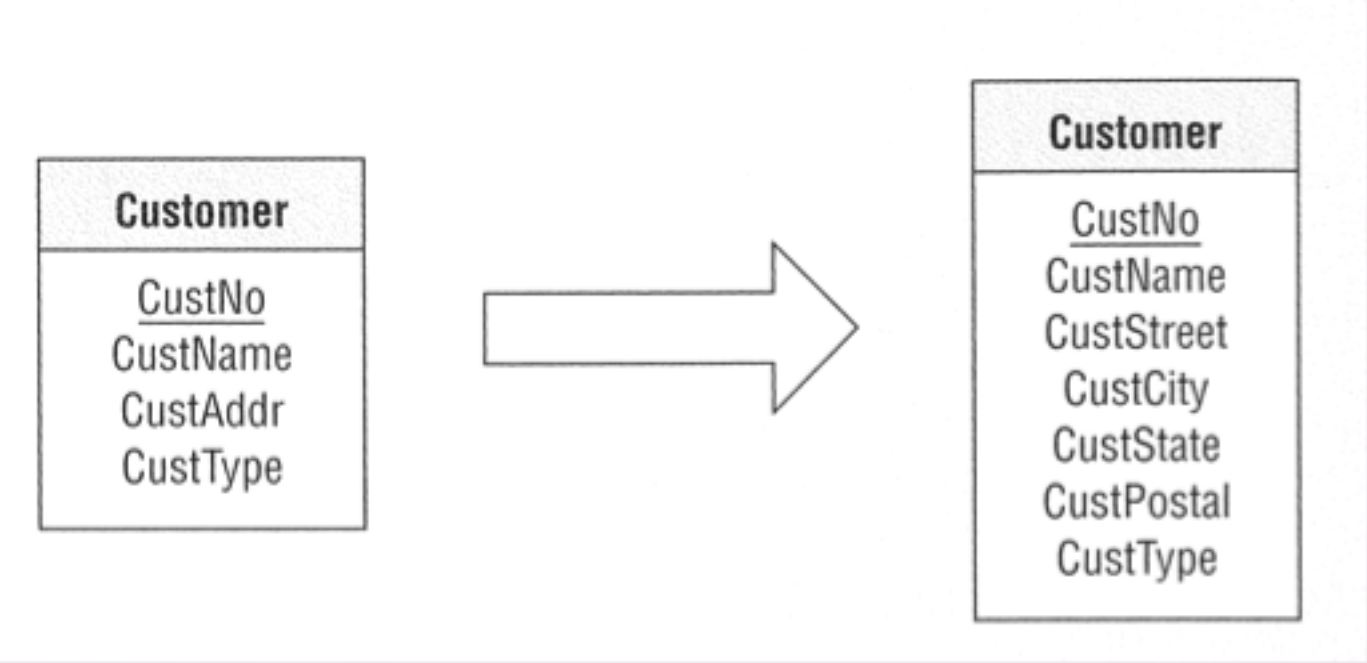


ERD refinements

- We constructed initial ERD
- However, several refinements can be made
- Typical list of refinements:
  - Attributes -> Entities
  - Splitting compound attributes

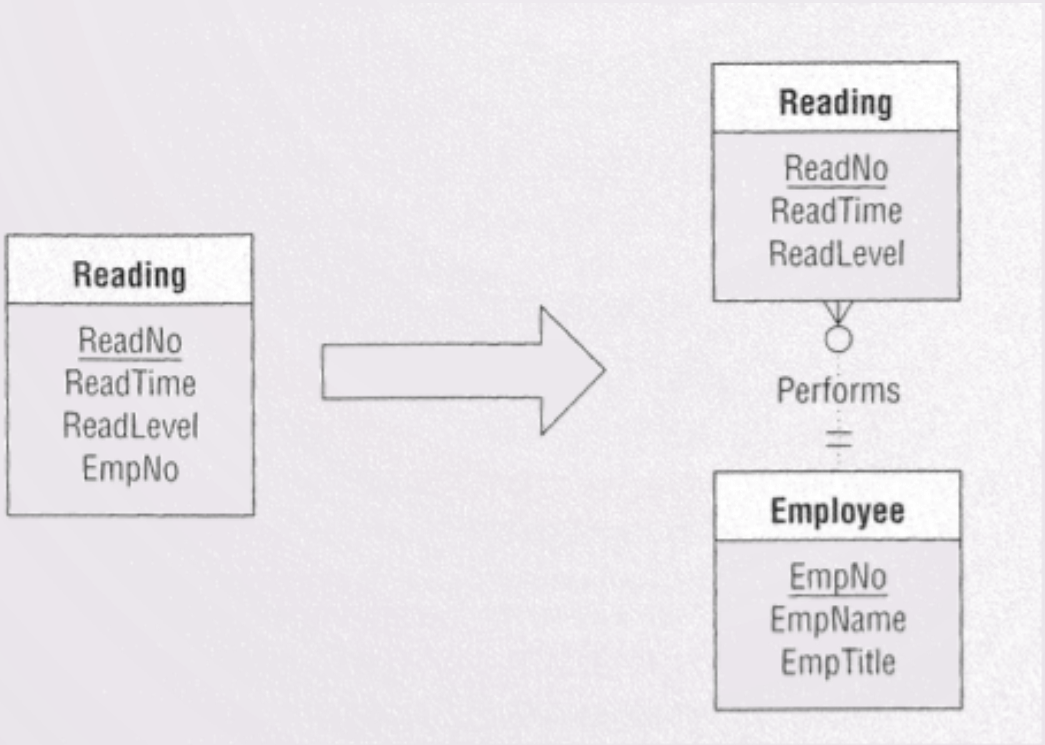
Splitting attributes

- Customer Address is a compound attribute.
  Maybe we need to search by city?

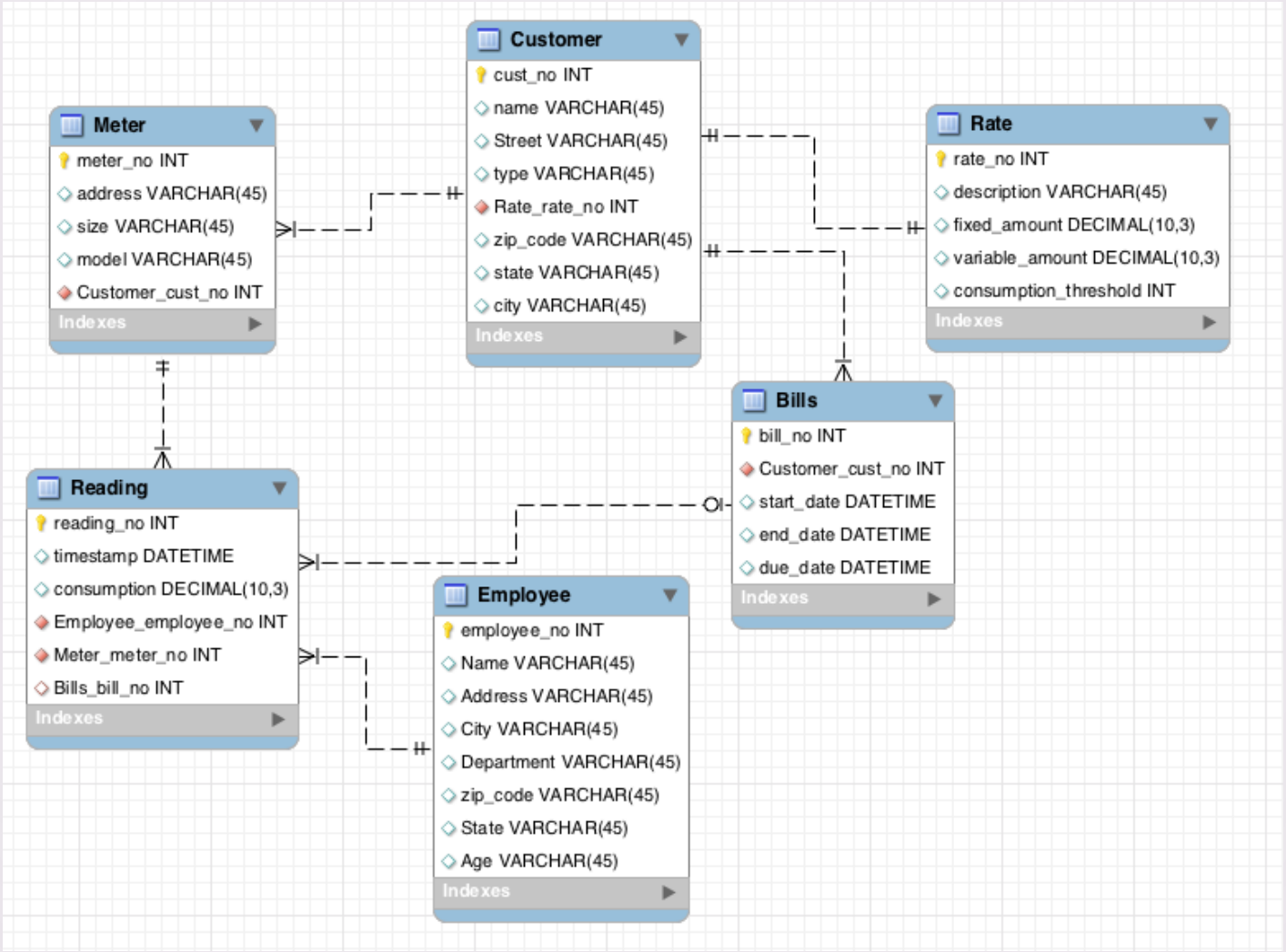


Attributes -> Entities

- Readings are done by employees. What if we know something about them (e.g. name, title)?



Final ERD

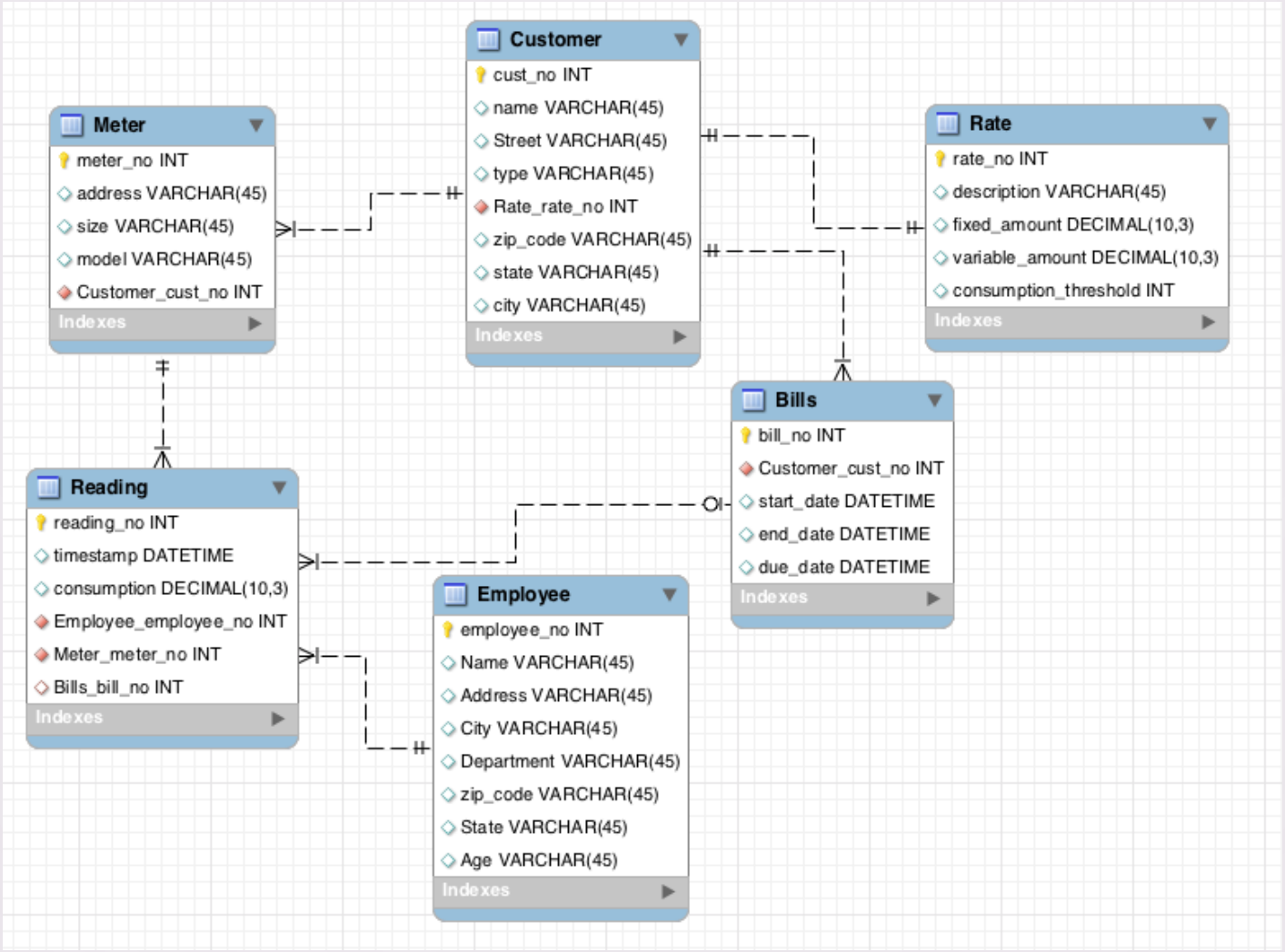

Customer
cust_no INT
name VARCHAR(45)
Street VARCHAR(45)
type VARCHAR(45)
Rate_rate_no INT
zip_code VARCHAR(45)
state VARCHAR(45)
city VARCHAR(45)
Indexes
Meter
meter_no INT
address VARCHAR(45)
size VARCHAR(45)
model VARCHAR(45)
Customer_cust_no INT
Indexes
Rate
rate_no INT
description VARCHAR(45)
fixed_amount DECIMAL(10,3)
variable_amount DECIMAL(10,3)
consumption_threshold INT
Indexes
Bills
bill_no INT
Customer_cust_no INT
start_date DATETIME
end_date DATETIME
due_date DATETIME
Indexes
Reading
reading_no INT
timestamp DATETIME
consumption DECIMAL(10,3)
Employee_employee_no INT
Meter_meter_no INT
Bills_bill_no INT
Indexes
Employee
employee_no INT
Name VARCHAR(45)
Address VARCHAR(45)
City VARCHAR(45)
Department VARCHAR(45)
zip_code VARCHAR(45)
State VARCHAR(45)
Age VARCHAR(45)
Indexes

FROM ER DIAGRAMS TO TABLES

ER Diagram → Relational Model

- Step 0: Identify:
  - Entities
  - Relationships (many-to-many, one-to-many, one-to-one)
  - Special attributes (primary key)
- Step 1: Entities
- Step 2: Many-to-many relationships
- Step 3: One-to-many relationships
- Step 4: One-to-one relationships

Example - Water Utility

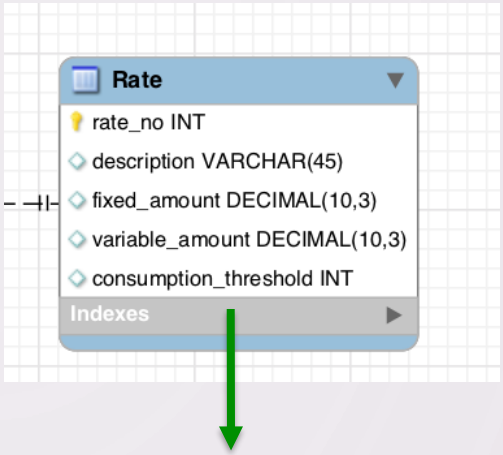


Step 0: Entities map to Tables

- Each entity maps 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
- Primary keys are underlined

Step 1: Water Utility Entities

- Rate
- Customer
- Meter
- Reading
- Employee
- Bill



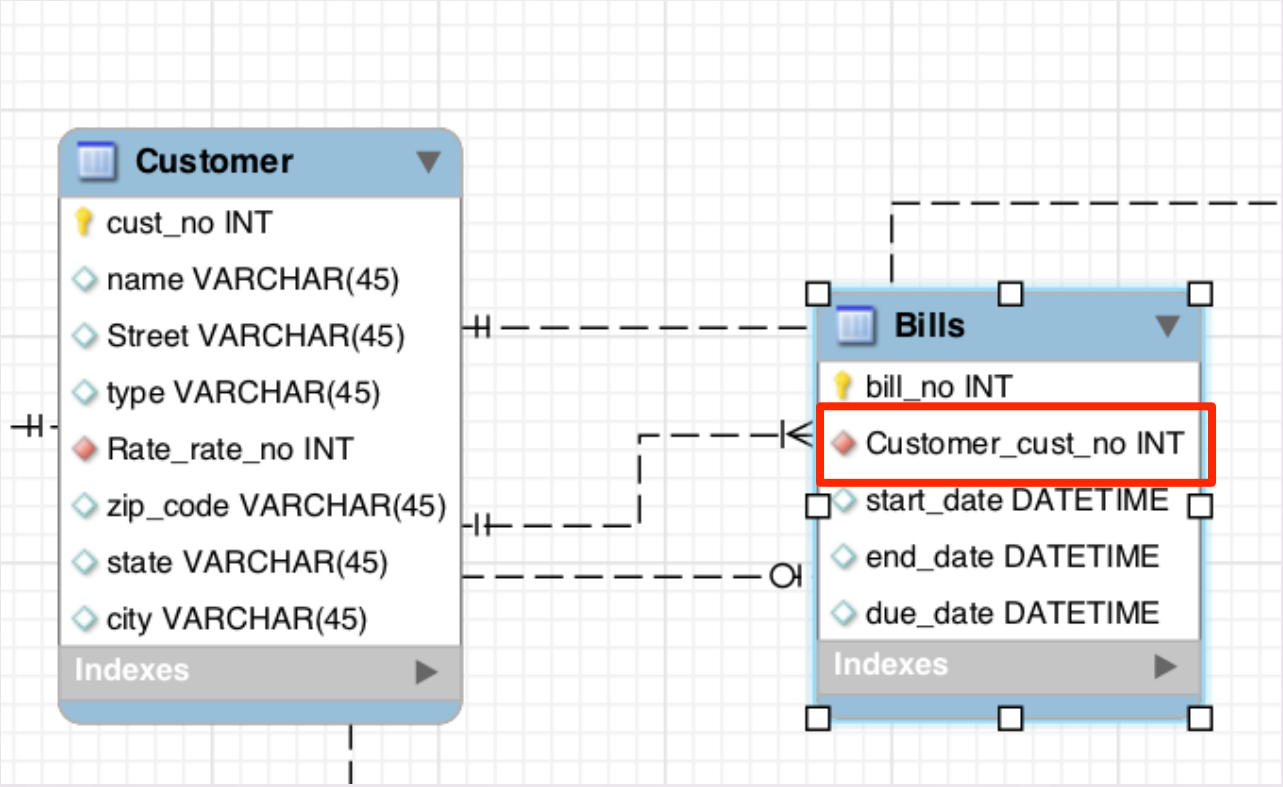
Rate

Rate_no	*Description*	*Fixed amount*	*Variable amount*	*Consumption Threshold*
R1278	Rate descr 1	40	10	0.7
R982	Rate descr 3	70	20	0.7
R1908	Rate descr 7	120	Null	Null

Step 2: One to many relationships

- For each One-to-Many relationship
  - Add a foreign key (FK) to the table corresponding to the “many” entity

Step 2: Example

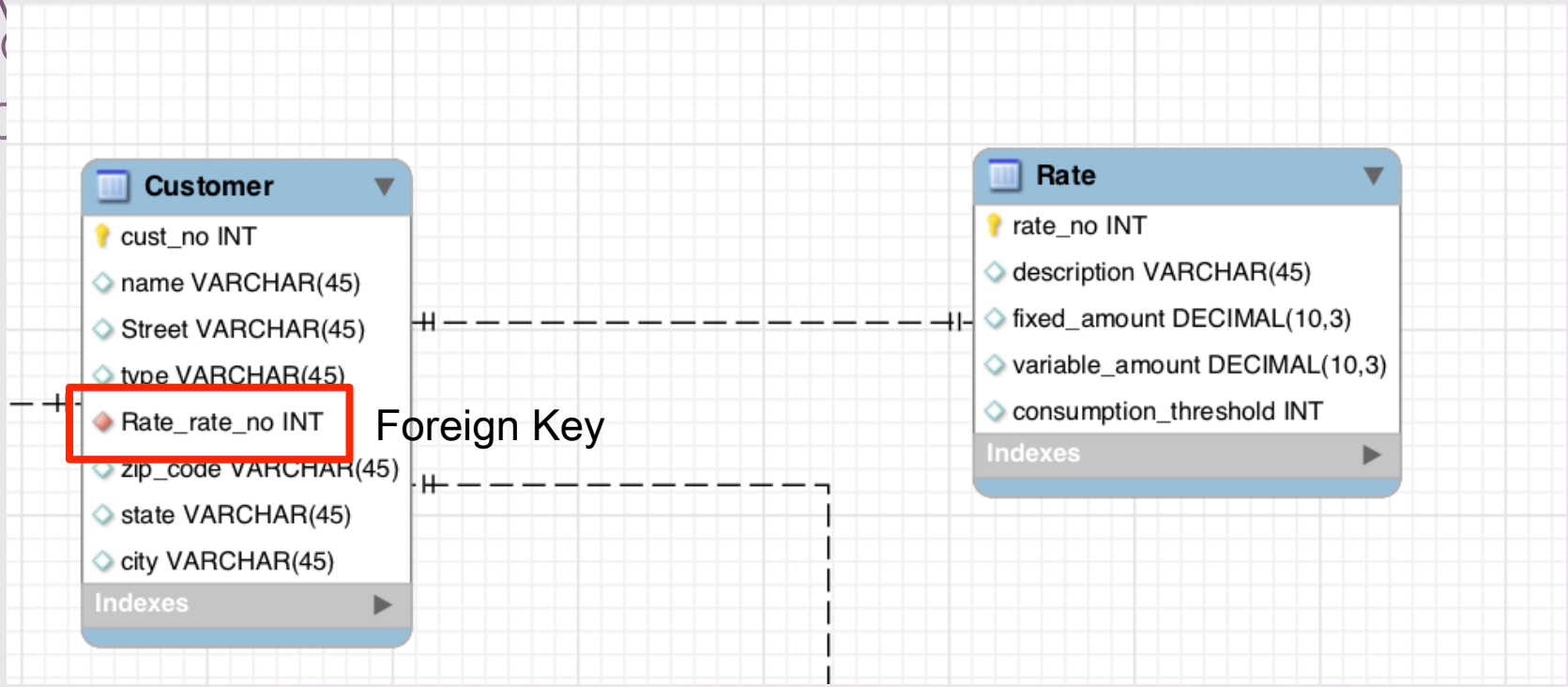


Foreign Key

Step 3: One to one Relationships

- We treat One-to-One relationship similarly as we treat One-to-Many relationship
- Option 1: Add a Foreign Key to either of the two tables
- Option 2: Merge the two tables (sometimes ok, sometimes bad style)

Step 3: Example

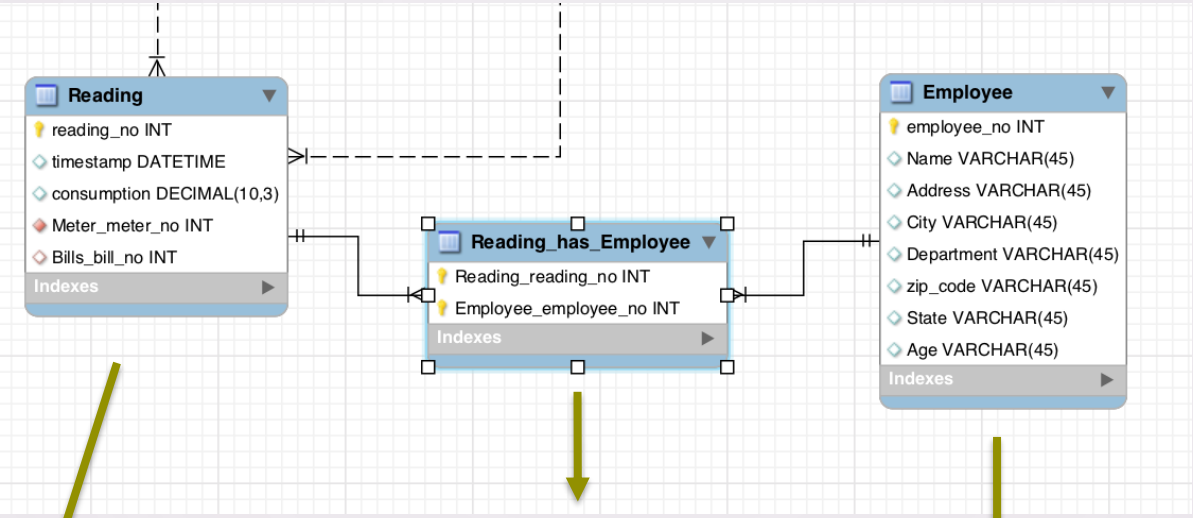


Step 4: Many to many relationships

- Need to introduce **bridge table** (Each Many-to-Many relationship becomes a separate 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

Step 4: Example

- Lets assume that a reading can be done by many employees



Reading No *Employee No*

Reading No *Timestamp* *…* *Employee No* *Name* *…*

Final Results

Meter (meter_no, address, size, model, *cust_no*)

Customer (cust_no, name, street, type, *rate_no*, zip_code, state, city)

Rate (rate_no, description, fixed_amount, variable_amount, consumption_threshold)

Reading (reading_no,timestamp,consumption, *meter_no*, *bill_no*)

Reading_has_Employee (*reading_no*, *employee_no*)

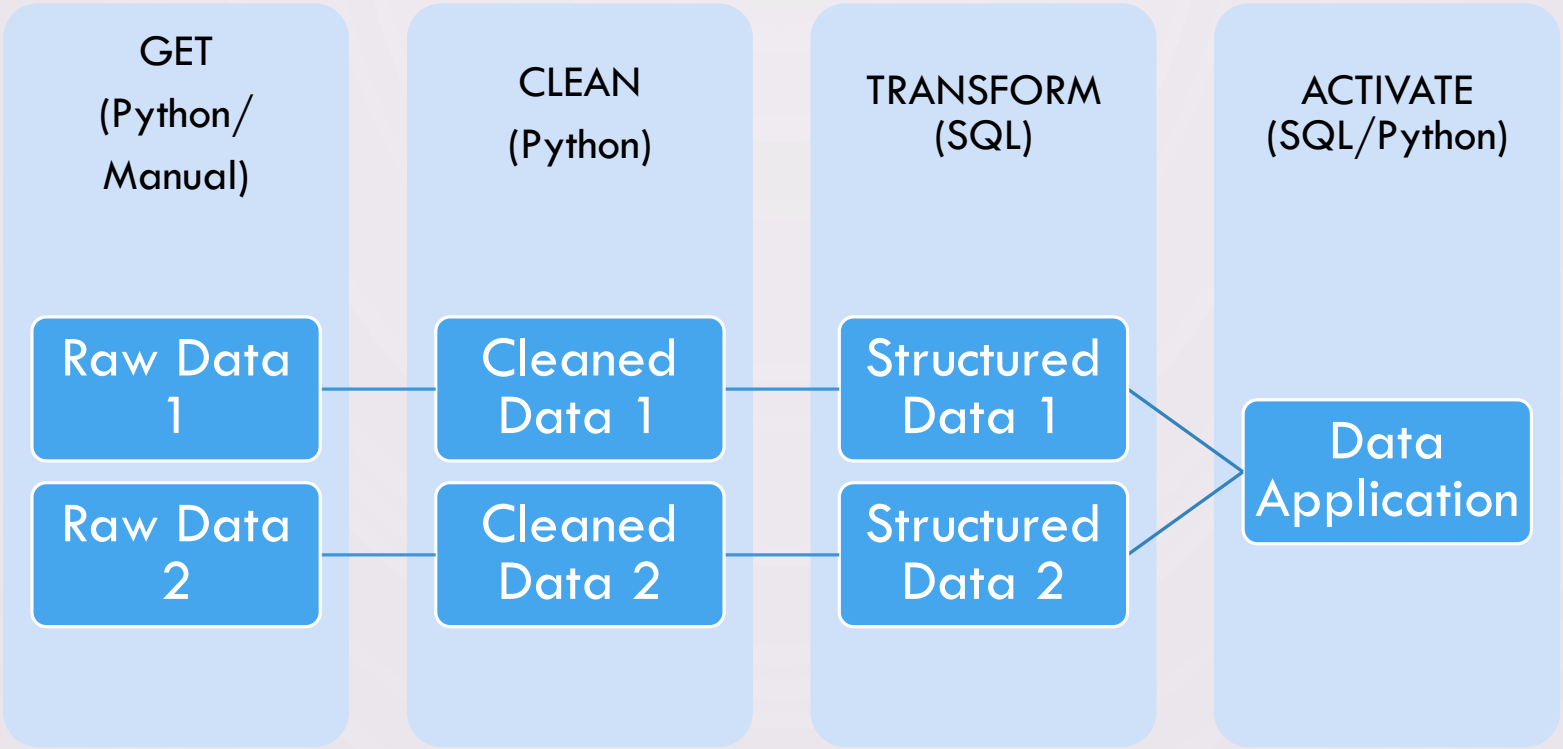
Employee (employee_no, name, address, city, dept, zip_code, state, age)

Bills (bill_no, *cust_no*, start_date, end_date, due_date)

Summary of ER modeling

- ER model is popular for conceptual design
  - Constructs are expressive, intuitive and graphical.
- Basic constructs: entity types, relationships (cardinalities), and attributes
- ER modeling is subjective!
  - There are often many ways to model a given scenario!
  - Analyzing alternatives is key
- ER modeling is iterative!
  - Resulting diagram should be analyzed and refined further.

CLASS PROJECT: BUILDING A DATA PIPELINE



CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION
➔ WORKING BACKWARDS FROM USER

- **Users:** Commissioner for NYC's Media & Entertainment Agency: https://www1.nyc.gov/site/mome/index.page
- **Decision Problem:** Daily decisions reviewing film permit requests for locations around NYC now need to consider how to create a balanced media representation of NYC's diversity.
- **Data Assistance Required:** Describe neighborhood characteristics of past film permits to inform future decisions

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ ACTIVATE'S DATA APPLICATION

- **Application Type = Descriptive** – the agency needs to start with an understanding of the neighborhoods shown in previously issued permits
- **Descriptive App Style = TBD**
  - Good data tables are adequate
  - Charts/visualizations could also be useful, but these are built on data tables so we'll focus on data tables to start

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION
➔ TRANSFORM

- **Data We Need?**
- **Map/Combine Data?**

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM

- **Data We Need?**
  - Neighborhood Film Permits ➔ NYC.gov
    - https://data.cityofnewyork.us/City-Government/Film-Permits/tg4x-b46p
  - Neighborhood resident income data ➔ IRS
    - https://www.irs.gov/statistics/soi-tax-stats-individual-income-tax-statistics-zip-code-data-soi
  - Neighborhood resident demographics ➔ US Census (2010)
    - https://factfinder.census.gov/faces/affhelp/jsf/pages/metadata.xhtml?lang=en&type=dataset&id=dataset.en.DEC_10_SF1
    - https://factfinder.census.gov/faces/nav/jsf/pages/searchresults.xhtml?refresh=t
  - What is needed to create the data application (dimensions/grains, facts/metrics)?
- **Map/Combine Data?**
  - Boroughs? Community Boards? Precincts? Zip Codes?

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE

- The IDEALIZED procedure for analysis & design of a NEW database for NEW data:
  - Identify entities and attributes
  - Determine primary keys
  - Identify relationships
  - Determine relationship cardinalities
  - Refine the ERD

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE

- The PRAGMATIC procedure for analysis & design of a NEW database for EXISTING data:
  - Inspect data to identify entities and attributes – and find gaps in available data
  - Determine IF we can create primary keys
  - Identify IDEAL relationships & cardinalities and MAP into REAL data
  - Sketch out the ERD

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE ➔ ENTITIES & ATTRIBUTES REAL/IDEAL

- Neighborhood Tax Returns
- Filming projects
- Neighborhood Resident Demographics

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE ➔ ENTITIES & ATTRIBUTES IDEAL

- Neighborhoods are within a Borough, can be located by Zip Code, contain 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's buildings and residents are shown in the result.
- Neighborhood Resident Demographics describe their diverse families and living accommodations
- Neighborhood Resident Tax Returns describe their incomes

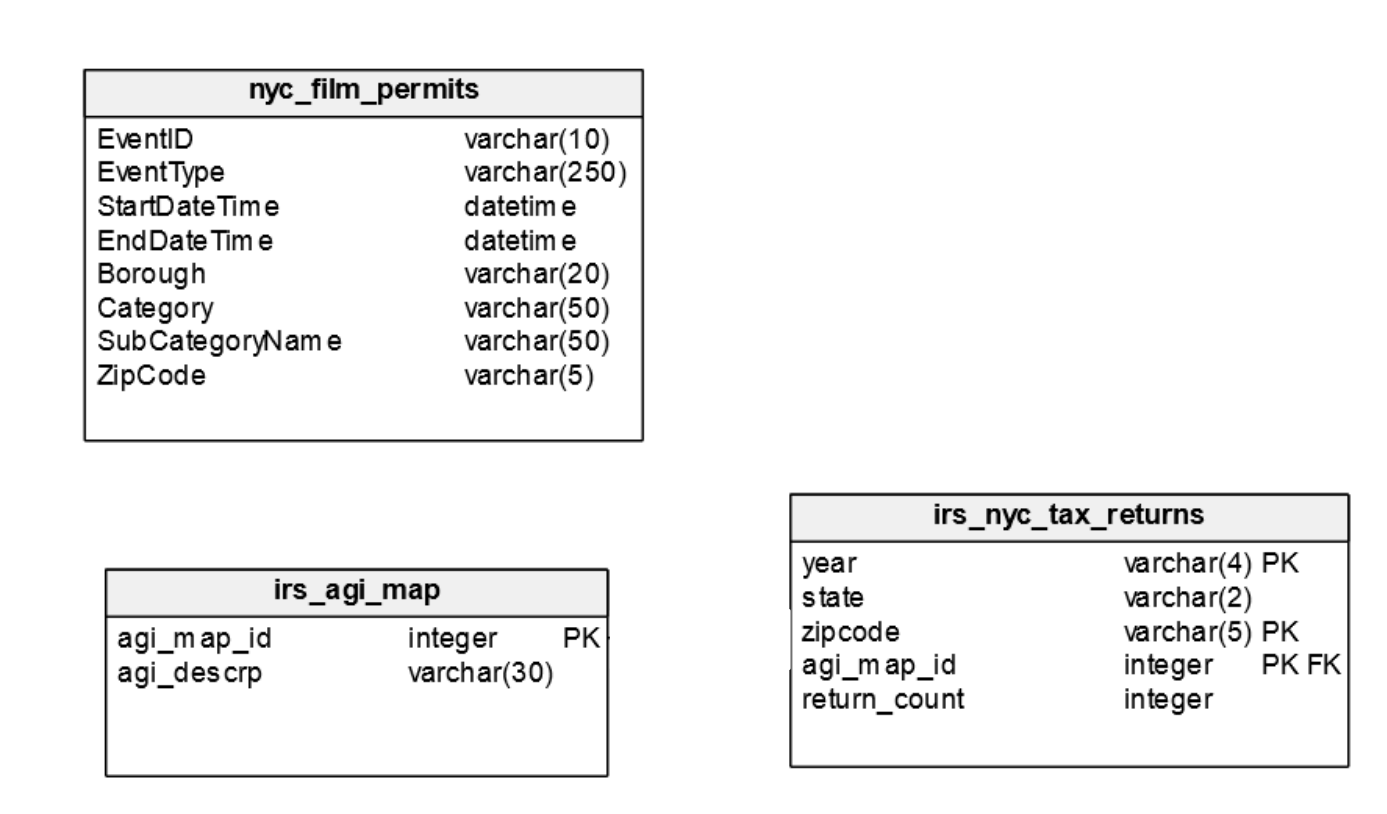
CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE ➔ ENTITIES & ATTRIBUTES

- Neighborhoods are within a Borough, can be located by Zip Code, contain 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's buildings and residents are shown in the result.
- Neighborhood Resident Demographics describe their diverse families and living accommodations
- Neighborhood Resident Tax Returns describe their incomes

GAP

CLASS PROJECT:

BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE ➔ ENTITIES & ATTRIBUTES FROM OUR DATA



CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE

- The PRAGMATIC procedure for analysis & design of a NEW database for EXISTING data:
  - ~~Inspect data to identify entities and attributes – find gaps in available data~~
  - Determine IF we can create primary keys
  - Identify IDEAL relationships & cardinalities and MAP into REAL data
  - Sketch out the ERD

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE ➔ PRIMARY KEYS

- Idealized world is designing database from scratch ➔ that's not our world
- Instead – we will inspect the existing data to identify where primary keys are available (or not)

CLASS PROJECT: BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE ➔ PRIMARY KEYS

- Real data findings (so far):
  - IRS Adjusted Gross Income Buckets have a Primary Key
  - IRS Tax Returns by Zip code could have a composed Primary Key of Year, Zip Code, AGI Bucket
- Open questions:
  - Film Permits? ➔ Homework
  - Resident Demographics? ➔ Next Class

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE

- The PRAGMATIC procedure for analysis & design of a NEW database for EXISTING data:
  - ~~Inspect data to identify entities and attributes – find gaps in available data~~
  - ~~Determine IF we can create primary keys~~
  - Identify IDEAL relationships & cardinalities and MAP into REAL data
  - Sketch out the ERD

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE ➔ IDENTIFY RELATIONSHIPS & CARDINALITIES

IDEALIZED

- ONE Neighborhood is identifiable 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

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE ➔ IDENTIFY RELATIONSHIPS & CARDINALITIES

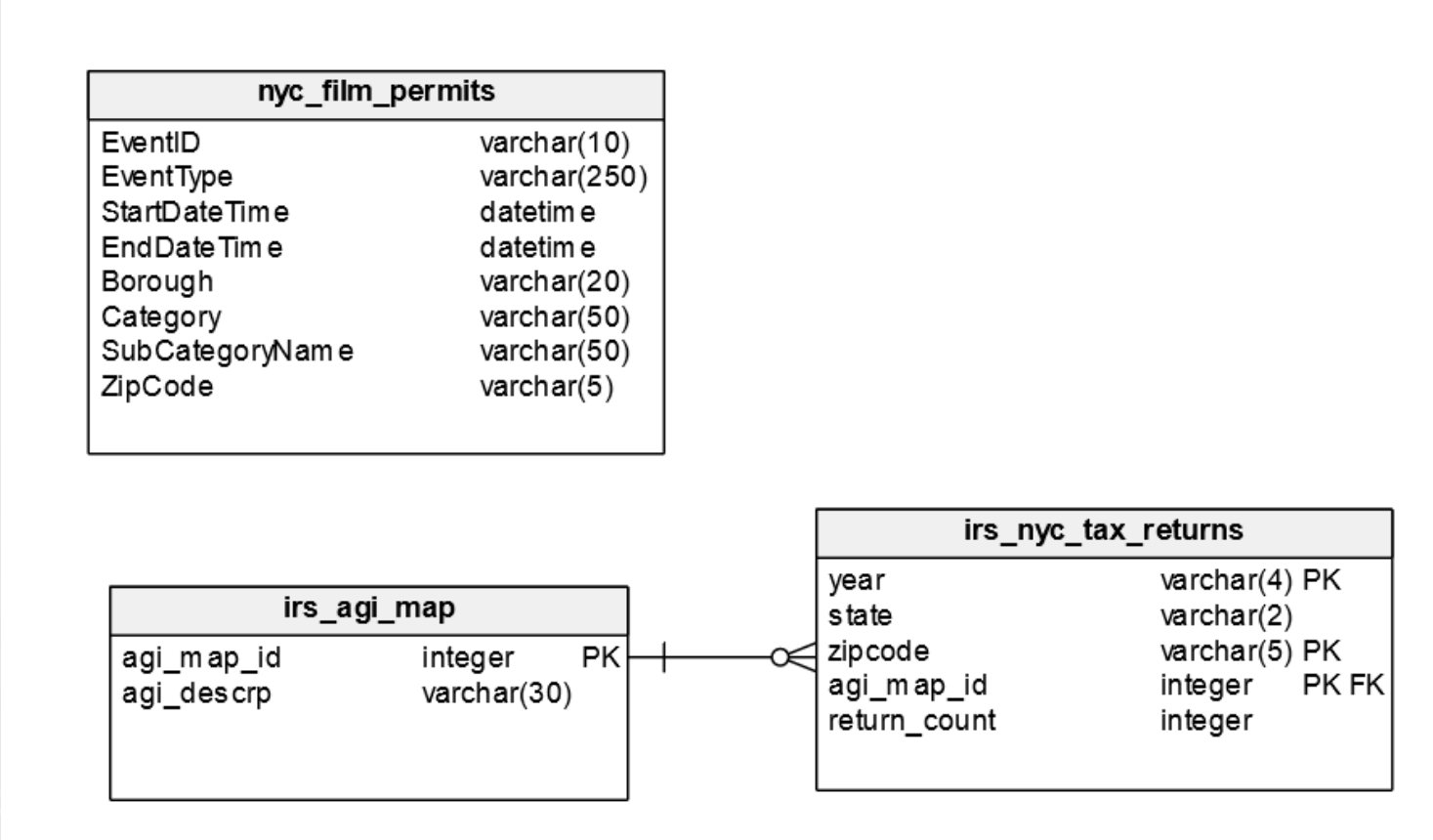
PRAGMATIC

- TODAY:
  - ONE Resident Income Level Bucket can map to MANY Neighborhood Tax Returns
- HOMEWORK:
  - ? Film Projects can map to ? Tax Returns
- NEXT CLASS:
  - ? Film Projects can map to? Neighborhood Demographic Attributes
  - ? Neighborhood Tax Returns can map to? Neighborhood Demographic Attributes

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE

- The PRAGMATIC procedure for analysis & design of a NEW database for EXISTING data:
  - ~~Inspect data to identify entities and attributes – find gaps in available data~~
  - ~~Determine IF we can create primary keys~~
  - ~~Identify IDEAL relationships & cardinalities and MAP into REAL data~~
  - Sketch out the ERD

CLASS PROJECT:
BALANCED NYC MEDIA REPRESENTATION ➔ TRANSFORM ➔ DESIGN DATABASE ➔ SKETCH ERD SO FAR…



CREATING TABLES IN SQL

SQL Statements to create tables

```
CREATE TABLE table_name
(
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
....
);
```

SQL Statements to create tables

```
CREATE TABLE Rate
(
id INT,
name VARCHAR(20),
price FLOAT,
startdate DATE
);
```

Define primary key

```
CREATE TABLE Rate
(
id INT,
name VARCHAR(20),
price FLOAT,
startdate DATE,
PRIMARY KEY(id)
);
```

Define FOREIGN KEY

```
CREATE TABLE Rate
(
id INT,
customerid INT,
price FLOAT,
startdate DATE,
PRIMARY KEY(id)
FOREIGN KEY (customerid) REFERENCES Customer(id)
);
```

“Enforcing participation” in a relationship

To specify that a relationship has a minimum cardinality of one, we specify that the attribute cannot be NULL (ie empty)

In this case, the rate needs to be associated with a customer

```
CREATE TABLE Rate
(
id INT,
customerid INT NOT NULL,
price FLOAT,
startdate DATE,
PRIMARY KEY(id),
FOREIGN KEY (customerid) REFERENCES Customer(id)
);
```

“Enforcing 1-1” in a relationship

To specify that a relationship has a maximum cardinality of one, we specify that the foreign key is UNIQUE

In this case, the rate needs to be associated with a customer, and a customer can be assigned with only one rate (due to the UNIQUE)

```
CREATE TABLE Rate
(
id INT,
customerid INT NOT NULL,
price FLOAT,
startdate DATE,
PRIMARY KEY(id),
UNIQUE (customerid),
FOREIGN KEY (customerid) REFERENCES Customer(id)
);
```

SQL

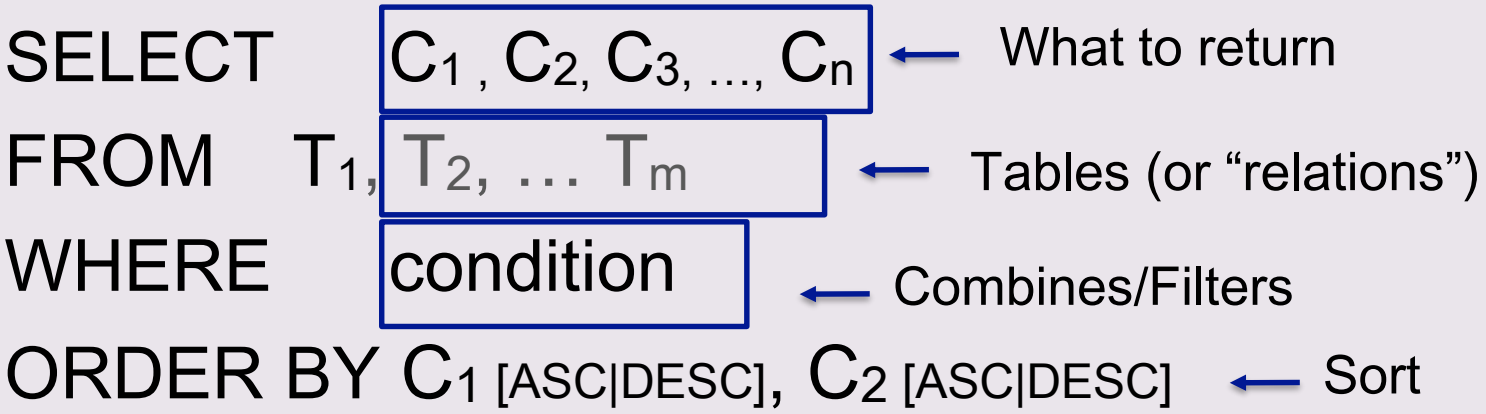
Intro

- “S.Q.L.” or “sequel”
- Supported by all major commercial DBMS
- Standardized
- Interactive via GUI or command line, or embedded in programs (e.g., in Python programs)
- Declarative

Terminology

- Data Definition Language (DDL)
  - Create Table
  - Drop Table
  - Indexes
- Data Manipulation language (DML)
  - Select
  - Insert
  - Delete
  - Update
- Other Commands
  - constraints, views, triggers etc.

The SELECT statement



The result of a query is a **relation**.

Note that a table is always a relation, but not vice versa.

The SELECT statement

```
SELECT *
FROM    TableName
```

Practice queries

```
SELECT

*

FROM irs_agi_map;
```

```
SELECT

*

FROM irs_nyc_tax_returns;
```

The SELECT statement

SELECT A_1 , A_2, … ← *Columns to return*

FROM *TableName*

Practice queries

```
SELECT state,zipcode
FROM irs_nyc_tax_returns
```

```
SELECT year,state,zipcode,return_count
FROM irs_nyc_tax_returns
```

The SELECT statement: The AS clause

Sometimes we want to rename a column to have a more descriptive name in the results.

SELECT A_1 **AS** *Name1*, AS_2, …**AS** *Name2*, …

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;
```

The SELECT DISTINCT statement

Used to eliminate duplicates in the results.

SELECT **DISTINCT** A_1 , A_2, A_3, ..., A_n
FROM TableName

Practice queries

```
SELECT

    DISTINCT year

FROM irs_nyc_tax_returns;
```

```
SELECT

    DISTINCT year,

            zipcode

FROM irs_nyc_tax_returns;
```

```
SELECT

    DISTINCT year,

            zipcode as 'zip code'

FROM irs_nyc_tax_returns;
```

The ORDER BY and LIMIT clause

- **ORDER BY**: Used to sort the result rows based on attribute values
  Potentially useful tip: the "order by" attributes do not need to appear in the results
- **LIMIT**: Limits the number of rows in the results
  - *MySQL Workbench limits all queries to 1000 rows by default*
  - *Different databases use different ways to say "LIMIT"*

SELECT A_1 , A_2, A_3, …, A_n
FROM TableName
ORDER BY A_1 [ASC|DESC], A_2 [ASC|DESC],…
LIMIT N

Practice queries

```
SELECT DISTINCT year,zipcode AS 'zip code'
FROM irs_nyc_tax_returns
LIMIT 5;

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

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

The WHERE clause

The WHERE clause defines which **rows** will appear in the results.

SELECT A_1 , A_2, A_3, …, A_n
FROM T_1
WHERE condition
ORDER BY A_1 [ASC|DESC], A_2 [ASC|DESC]

Conditions for WHERE clause

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");

SELECT *
FROM irs_nyc_tax_returns
WHERE zipcode NOT IN (10001,10002,10128);

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

The NULL value

- When columns do not have a value, they are assigned a “NULL” value, which is a special way that SQL handles the “empty value”
  - *Notice: NULL is **not** identical to “” (empty string). In practice, you may see both, although NULL is always a superior choice*

- To check if something is NULL you use the expression: “**attr IS NULL**”
  - Example: Find all students that have not listed their birthday

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

- Similarly, you use “attr IS NOT NULL” is you want only results that have non-NULL values

The NULL value

- When columns do not have a value, they are assigned a “NULL” value, which is a special way that SQL handles the “empty value”
  - *Notice: NULL is **not** identical to “” (empty string). In practice, you may see both, although NULL is always a superior choice*

- To check if something is NULL you use the expression: “**attr IS NULL**”
  - Example: Find all students that have not listed their birthday

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

- Similarly, you use “attr IS NOT NULL” is you want only results that have non-NULL values

Incorrect approaches when using NULL

*NULL is **not** equal to empty string. The query below will NOT work*

```
SELECT *
FROM Profiles
WHERE Birthday = '';
```

We do not use = to compare with NULL. The query below will NOT work

```
SELECT *
FROM Profiles
WHERE Birthday = NULL;
```

Using COMMENTS

SQL permits two commenting styles: -- and /* */

Examples:

-- permits one line of comment as shown here

SELECT *

FROM your_table;

/ permits several lines of comments here*

And here

*and you stop comments after */*

SELECT *

FROM your_table;

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