Rotman

Master of Management Analytics

INTRO TO SQL

Bootcamp (https://tdmdal.github.io/mma-sql-2023/)

August 9, 2023 Prepared by Jay / TDMDAL

Rotman School of Management
UNIVERSITY OF TORONTO

What's SQL (Structured Query Language)

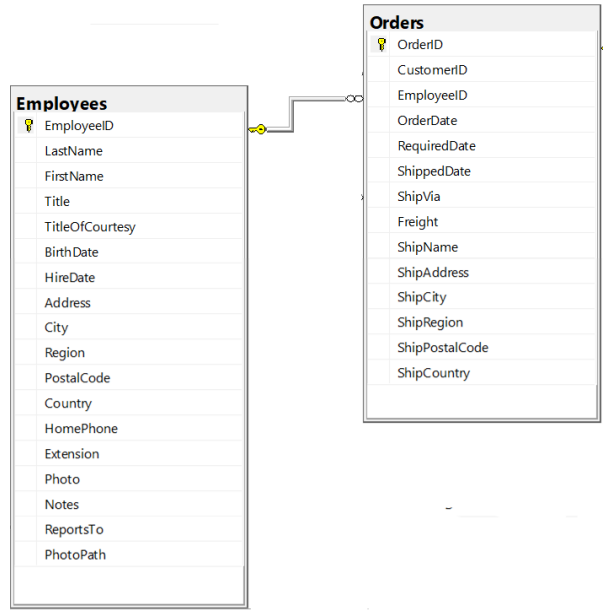
- Most widely used database (DB) language
  - a domain specific language: managing data stored in *relational DBs*
- Not a proprietary language
  - Open specifications/standards (ANSI & ISO)
  - All major DBMS (DB Mgmt. System) vendors implement Standard SQL
  - However, SQL Extensions are usually DB specific (SQL dialects)
- Powerful despite simplicity

ANSI - American National Standards Institute; ISO – International Organization for Standardization

What's a DB and a Relational DB

- What's a database: A collection of data in an organized way

- *Relational DB (RDB)*
  - tables
    - columns/fields/variables, and a datatype per column
    - rows/records/observations
  - primary key, foreign key, constraints and relationships
  - other objects: indices, views, triggers and many more



What is a DB Management System

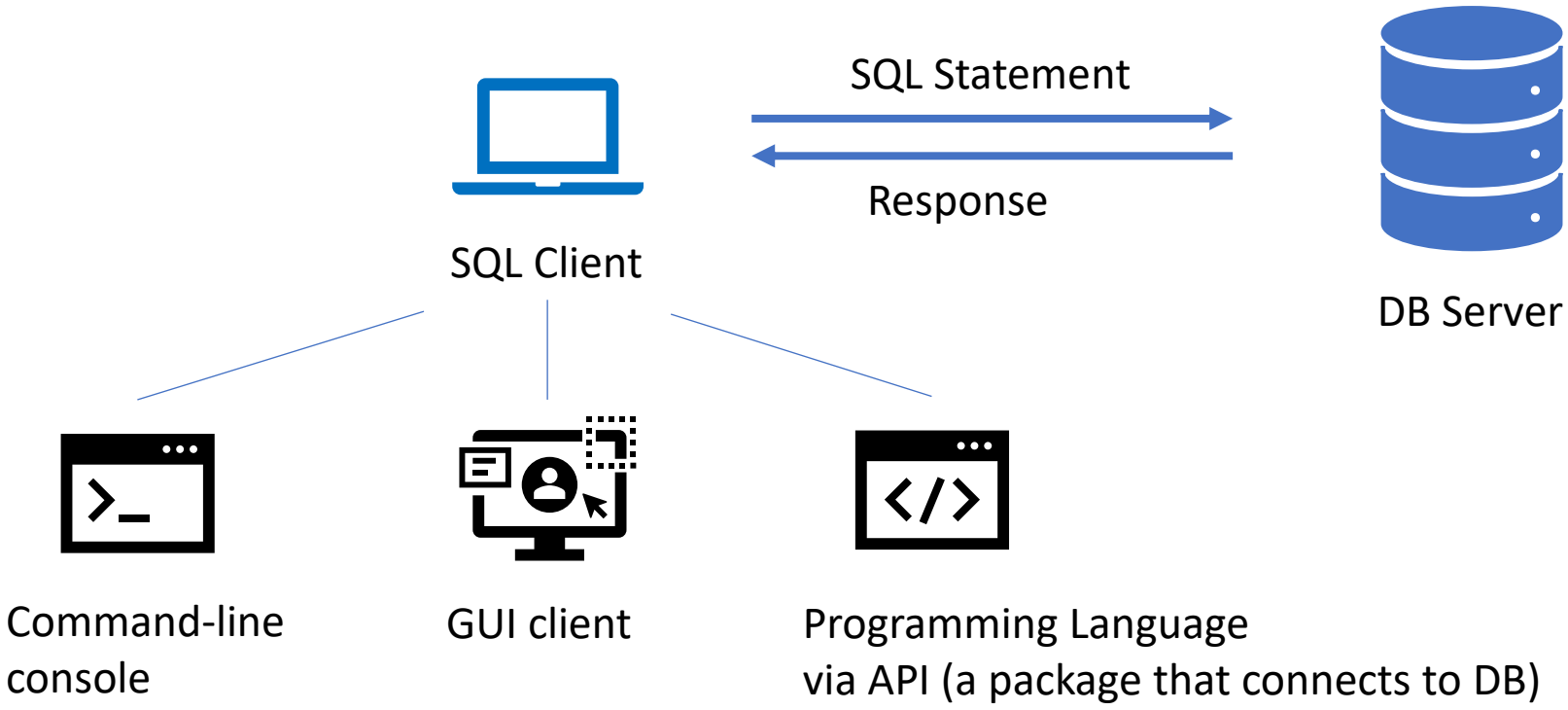
- A software system that manages/maintains DBs

- A few examples of Relational DBMS (RDBMS)
  - Open source: SQLite, DuckDB, MariaDB, PostgreSQL
  - Commercial: MySQL, Microsoft SQL Server, Oracle, etc.

Our choice for this bootcamp

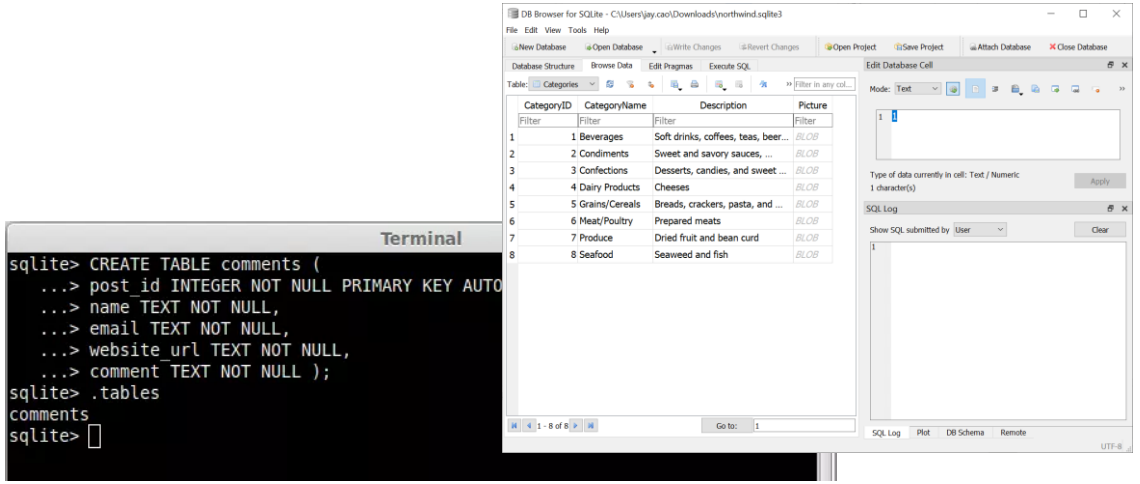
RSM8411's choice

How do I use SQL – Logical Architecture



Note: 1) SQL client and DB server can be on the same computer;
2) our DB choice for this bootcamp, SQLite, isn't a client-server DB engine. It's "server-less".

SQL Clients



- DB specific management client
  - Command-line console
  - GUI (Graphic User Interface) client
    - e.g., DB Browser for SQLite, MySQL Workbench, pgAdmin for PostgreSQL, MS SSMS

- Generic DB client can connect to different DBs through connectors
  - GUI client (e.g. DBeaver, Beekeeper Studio, Navicat)
  - Programming language
    - e.g., Python + SQLAlchemy + DBAPI (e.g. SQLite, MySQL, PostgreSQL, etc.), R + dbplyr
    - **In this workshop**: Python notebook with ipython-sql (depends on SQLAlchemy) + SQLite (bundled with Python)

Beyond a relational DB language

- SAS's PROC SQL
- Spark's SparkSQL
  - Apache Spark is a big data computing framework
- Hive's HiveQL, an SQL-like query language
  - Apache Hive is a distributed data warehouse (data warehouse?)
- **Google BigQuery's SQL** (a great first step to big data analysis)
  - BigQuery is Google's data warehouse (analyze petabytes of data at ease)

ref. 1) Database vs data warehouse; 2) Data warehouse vs data lake; 3) NoSQL DB vs SQL DB

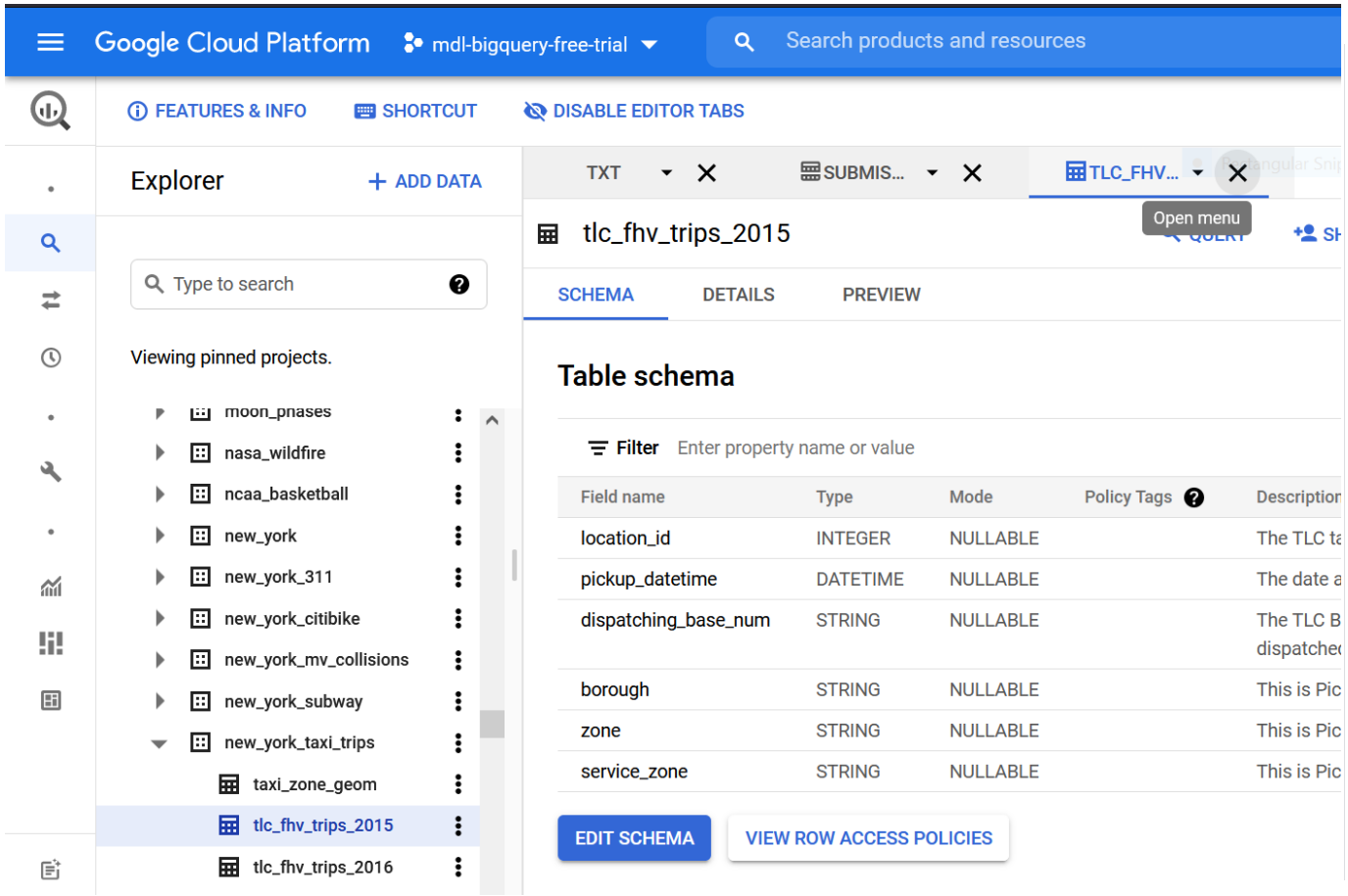
You Can Go Big with SQL

- Todd W. Schneider's original analysis on NYC Taxi trips (2015)
  - 267 GB of raw data
  - 1.1 billion rows
  - **PostgreSQL** and R
  - Macbook Pro then (2015), but now Macbook Air should be able to handle

- Many big data DB systems (not necessarily RDBs) in the cloud support SQL (SQL-like) query
  - E.g. Google BigQuery

Ref. Analyzing 1.1 Billion NYC Taxi and Uber Trips, with a Vengeance - Todd W. Schneider (toddwschneider.com)

Big Data ML with SQL (e.g. Google BigQuery)




```
CREATE OR REPLACE MODEL taxi.taxifare_model_2
OPTIONS
  (model_type='linear_reg', labels=['total_fare']) AS
WITH params AS (
    SELECT
    1 AS TRAIN,
    2 AS EVAL
    ),
  daynames AS
    (SELECT ['Sun', 'Mon', 'Tues', 'Wed', 'Thurs', 'Fri',
'Sat'] AS daysofweek),
  taxitrips AS (
  SELECT
    (tolls_amount + fare_amount) AS total_fare,
    daysofweek[ORDINAL(EXTRACT(DAYOFWEEK FROM
pickup_datetime))] AS dayofweek,
    EXTRACT(HOUR FROM pickup_datetime) AS hourofday,
    SQRT(POW((pickup_longitude - dropoff_longitude),2) +
POW(( pickup_latitude - dropoff_latitude), 2)) as dist,
#Euclidean distance between pickup and drop off
    SQRT(POW((pickup_longitude - dropoff_longitude),2))
as longitude, #Euclidean distance between pickup and drop
off in longitude
    SQRT(POW((pickup_latitude - dropoff_latitude), 2)) as
...
```

Ref. 1) [Using BigQuery ML and BigQuery GIS together to predict NYC taxi trip cost | Google Cloud Blog]

2) [Analyzing 1.1 Billion NYC Taxi and Uber Trips, with a Vengeance - Todd W. Schneider (toddwschneider.com)]

SQL Hands-on Exercises (Learning-by-doing)

- Course website: https://tdmdal.github.io/mma-sql-2023/

- Google Colab
  - Google's Jupyter Notebook
  - A notebook can contain live code, equations, visualizations and narrative text

- Why SQLite?
  - a small, fast, self-contained, high-reliability, full-featured, SQL DB engine
  - perfect for learning SQL

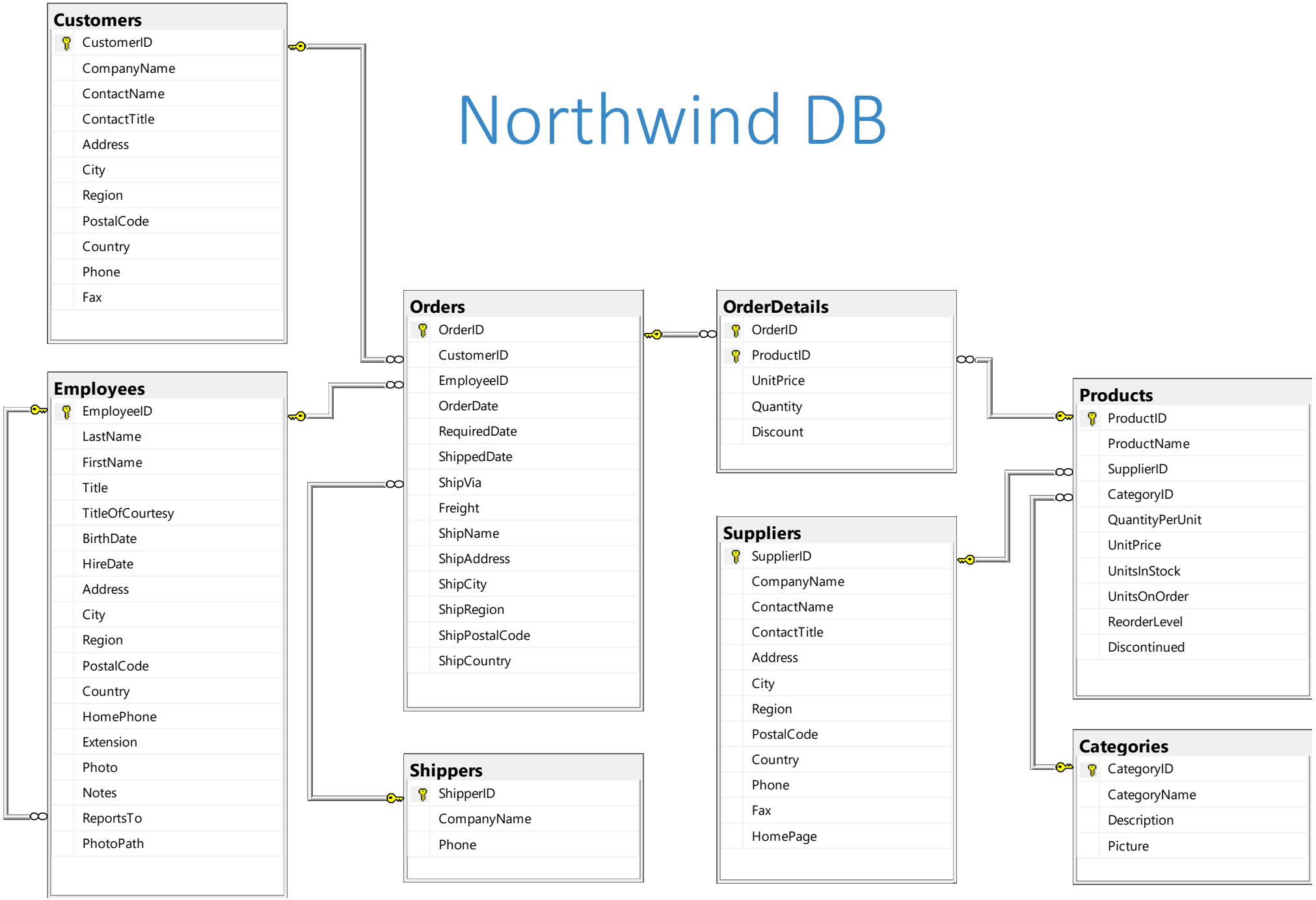
Preparation For RSM8411 (MMA, Fall 2023)

- A different setup (a more advanced/powerful DBMS)
  - PostgreSQL, an open-source DBMS
  - pgAdmin, a GUI client for managing PostgreSQL
  - Installation guide and get-started resources: see our bootcamp website

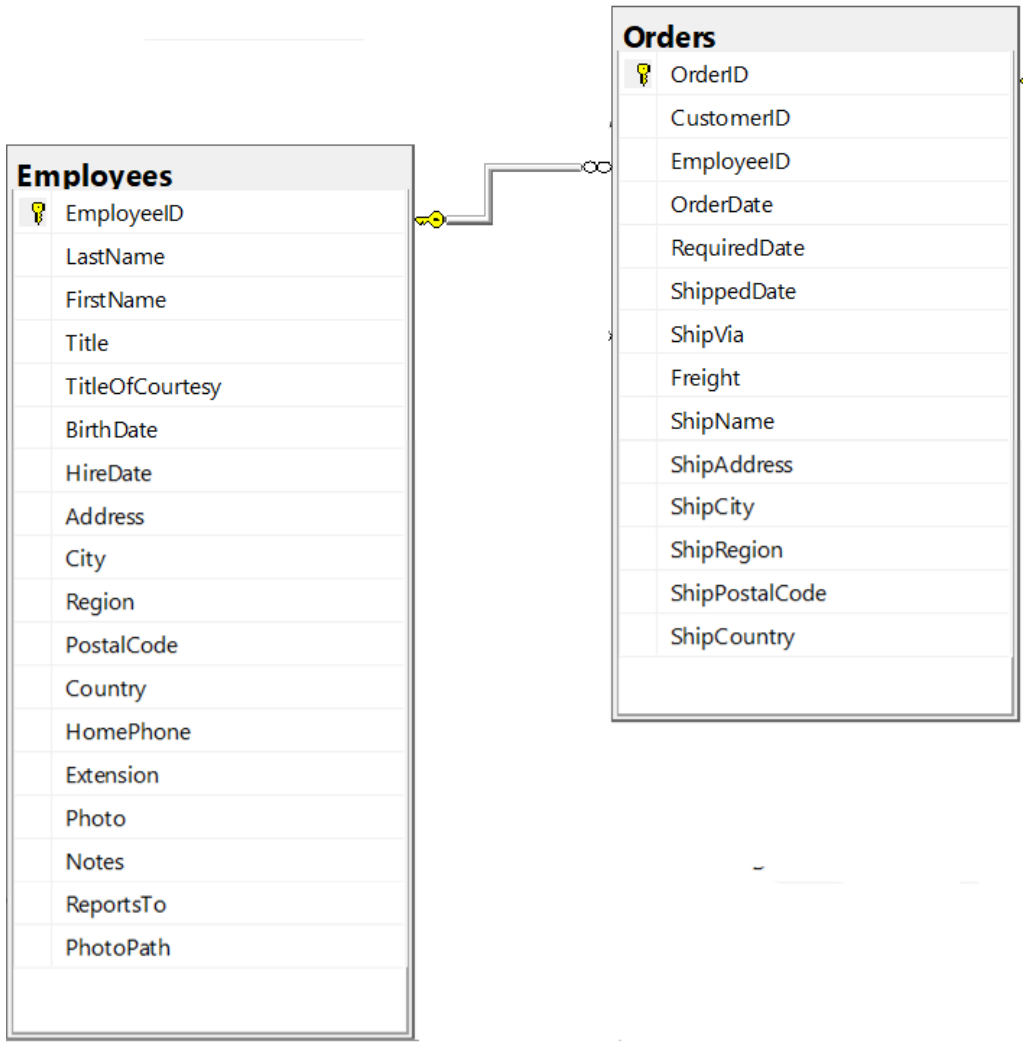
- Please make sure you have the above setup installed
  - Set it up before the end of this bootcamp
  - Email me if you have trouble with installation

- SQL syntax difference between SQLite and PostgreSQL
  - For 99% of what we will learn in this bootcamp, they are the same

Northwind DB



Primary key (PK), foreign key (FK), constraints & relationships - 1

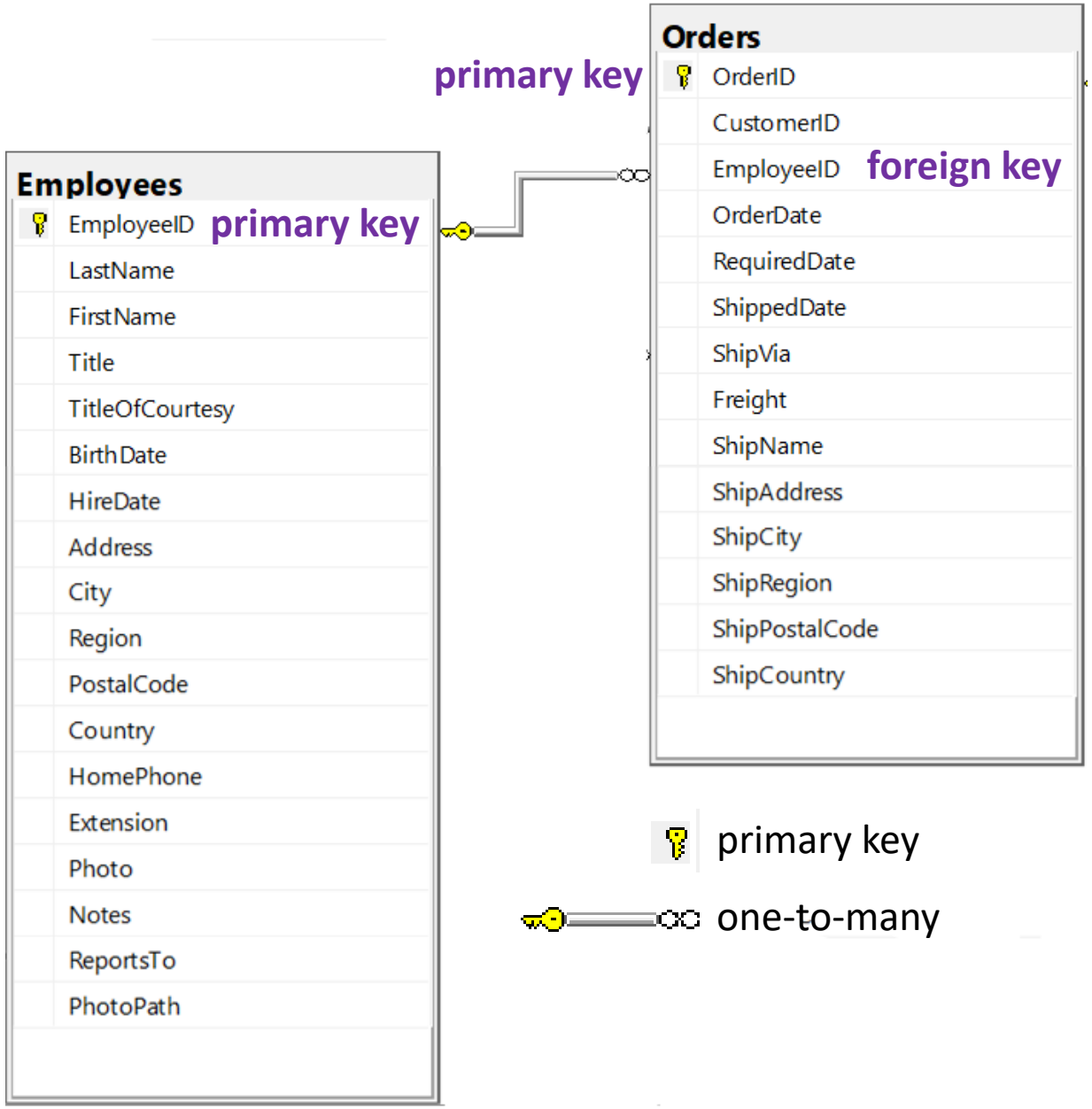


EmployeeID	LastName	FirstName	Title	…
1	Davolio	Nancy	Sales Representative	…
2	Fuller	Andrew	Vice President, Sales	…
3	Leverling	Janet	Sales Representative	…
4	Peacock	Margaret	Sales Representative	…
…	…	…	…	…

OrderID	CustomerID	EmployeeID	…
10248	VINET	5	…
10249	TOMSP	6	…
10250	HANAR	4	…
…	…	…	…

PK, FK, constraints & relationships - 2

- Two keys
  - **primary key**: uniquely identifies an observation in its own table
  - **foreign key**: uniquely identifies an observation in another table
- Relationship between tables
  - one-to-one
  - **one-to-many**
  - many-to-many
- FK constraints



Hands-on Part 1: Warm up

- Retrieve data: `SELECT...FROM...`
- Sort retrieved data: `SELECT...FROM...ORDER BY...`
- Filter data: `SELECT...FROM...WHERE...`
  - `IN`, `NOT`, `LIKE` and `%` wildcard
- Create calculated fields
  - mathematical calculations (e.g. `+`, `-`, `*`, `/`)
  - data manipulation functions (e.g. `DATE()`, `||`)

Hands-on Part 2: Summarize and Group Data

- Summarize data using aggregate functions (e.g. `COUNT()`, `MIN()`, `MAX()`, and `AVG()`).
- Group data and filter groups: `SELECT...FROM...GROUP BY...HAVING...`
- SELECT clause ordering: `SELECT...FROM...WHERE...GROUP BY...HAVING...ORDER BY...`
- Filter data by subquery: `SELECT...FROM...WHERE...(SELECT...FROM...)`

Works in SQLite, But… (1)

- Refer to column alias in WHERE and HAVING keywords
  - `SELECT column_name/col_expression AS alias_name`
- SQL is a *declarative* language, not a *procedural* language
  - You tell the system what you want to compute, not how to compute it
  - Behind the scenes, a query planner figures out the best way to compute it
- SQL syntax order
  - `SELECT… FROM… [INNER/LEFT] JOIN… ON… WHERE… GROUP BY… HAVING… ORDER BY… LIMIT…`
- SQL execution order
  - `FROM, JOIN…ON…, WHERE, GROUP BY, HAVING, SELECT, ORDER BY, LIMIT`

Works in SQLite, But… (2)

- group by and aggregation functions

```
SELECT col1, col2, another_col, agg_fun1(col3), agg_fun2(col4)
FROM table1
GROUP BY col1, col2
```

Hands-on Part 3: Join Tables

- Inner join: `SELECT...FROM...INNER JOIN...ON...`
- Left join: `SELECT...FROM...LEFT JOIN...ON...`
- Other join variations.

Join – Inner Join

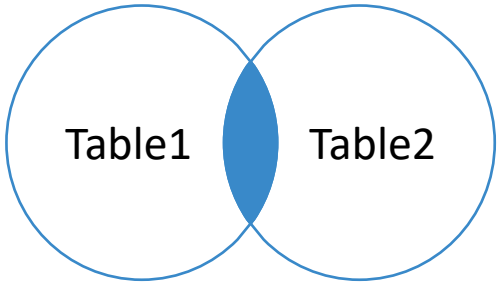


Table1

pk	t1c1
1	a
2	b

Table2

fk	t2c1
1	c
1	d
3	e

```
SELECT *
FROM Table1
  INNER JOIN Table2
  ON Table1.pk = Table2.fk;
```

pk	t1c1	fk	t2c1
1	a	1	c
1	a	1	d

Join – Left (Outer) Join

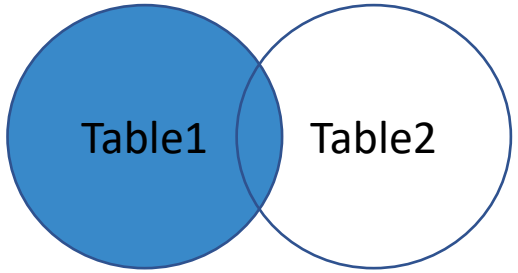


Table1

pk	t1c1
1	a
2	b

Table2

fk	t2c1
1	c
1	d
3	e

```
SELECT *
FROM Table1
  LEFT JOIN Table2
  ON Table1.pk = Table2.fk;
```

pk	t1c1	fk	t2c1
1	a	1	c
1	a	1	d
2	b	null	null

Join - Left (Outer) Join With Exclusion

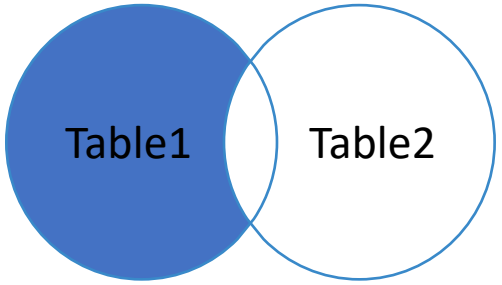


Table1

pk	t1c1
1	a
2	b

Table2

fk	t2c1
1	c
1	d
3	e

```
SELECT *
FROM Table1
  LEFT JOIN Table2
  ON Table1.pk = Table2.fk
WHERE Table2.fk is NULL;
```

pk	t1c1	fk	t2c1
2	b	null	null

Join – Right Outer Join*

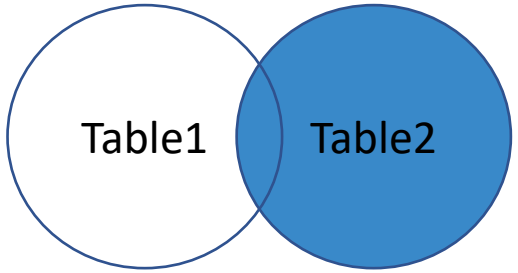


Table1

pk	t1c1
1	a
2	b

Table2

fk	t2c1
1	c
1	d
3	e

```
SELECT *
FROM Table2
  LEFT JOIN Table1
  ON Table2.fk = Table1.pk
---------------------------
SELECT *
FROM Table1
  RIGHT JOIN Table2
  ON Table1.pk = Table2.fk;
```

SQLite doesn't support this RIGHT JOIN key word, but some DBMSs do (e.g. MySQL).

pk	t1c1	fk	t2c1
1	a	1	c
1	a	1	d
null	null	3	e

Join - Right Outer Join With Exclusion*

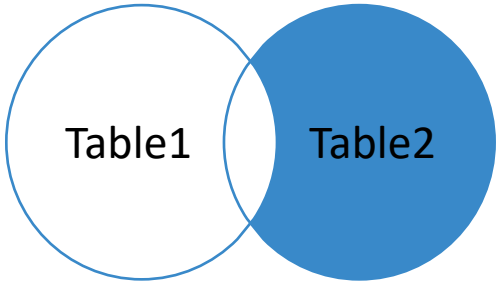


Table1

pk	t1c1
1	a
2	b

Table2

fk	t2c1
1	c
1	d
3	e

```
SELECT *
FROM Table2
  LEFT JOIN Table1
  ON Table2.fk = Table1.pk
WHERE Table1.pk is NULL;
---------------------------
SELECT *
FROM Table1
  RIGHT JOIN Table2
  ON Table1.pk = Table2.fk
WHERE Table1.pk is NULL;
```

pk	t1c1	fk	t2c1
null	null	3	e

SQLite doesn't support this RIGHT JOIN key word, but some DBMSs do (e.g. MySQL).

Join – Full Outer Join

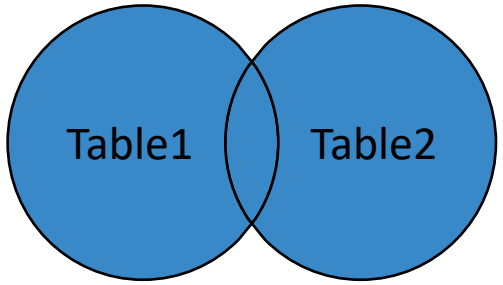


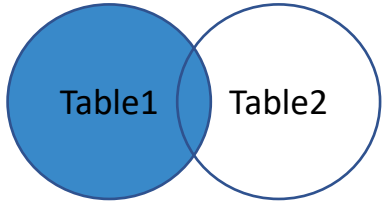
Table1

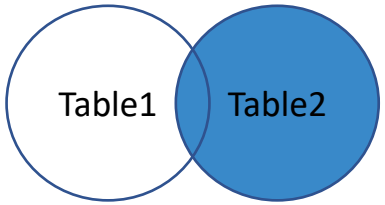
pk	t1c1
1	a
2	b

Table2

fk	t2c1
1	c
1	d
3	e

```
SELECT pk, t1c1, fk, t2c1
FROM Table1
  LEFT JOIN Table2
  ON Table1.pk = Table2.fk
UNION
SELECT pk, t1c1, fk, t2c1
FROM Table2
  LEFT JOIN Table1
  ON Table2.fk = Table1.pk;
```





pk	t1c1	fk	t2c1
1	a	1	c
1	a	1	d
2	b	null	null
null	null	3	e

Note: Some DBMS support FULL OUTER JOIN keyword (e.g. MS SQL) so you don't need to do it the above way.

Join – Full Outer Join With Exclusion*

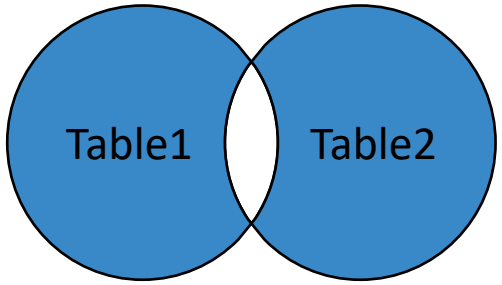


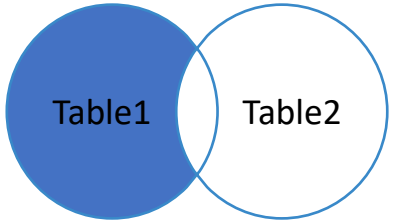
Table1

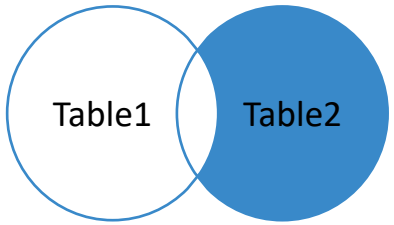
pk	t1c1
1	a
2	b

Table2

fk	t2c1
1	c
1	d
3	e

```
SELECT pk, t1c1, fk, t2c1
FROM Table1
  LEFT JOIN Table2
  ON Table1.pk = Table2.fk
WHERE Table2.fk is NULL
UNION
SELECT pk, t1c1, fk, t2c1
FROM Table2
  LEFT JOIN Table1
  ON Table2.fk = Table1.pk
WHERE Table1.pk is NULL;
```





pk	t1c1	fk	t2c1
2	b	null	null
null	null	3	e

UNION vs UNION All

- Syntax: Query 1 UNION (ALL) Query 2

- Both combine rows
- UNION removes duplicated rows, but UNION All doesn't

- Rules
  - # of cols in both queries must be the same
  - Corresponding cols must have compatible data types
  - Col names of the first query determine the col names of the combined result

Ref. https://www.sqlitetutorial.net/sqlite-union/

UNION vs UNION All – An Example (Part 1)

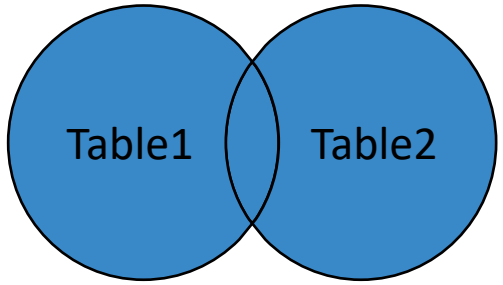


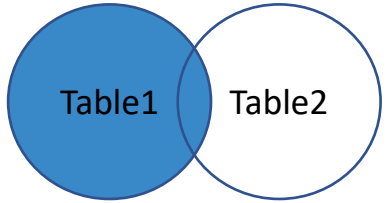
Table1

pk	t1c1
1	a
2	b

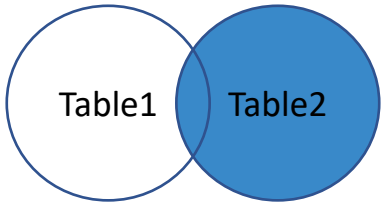
Table2

fk	t2c1
1	c
1	d
3	e

```
SELECT pk, t1c1, fk, t2c1
FROM Table1
  LEFT JOIN Table2
  ON Table1.pk = Table2.fk
UNION
SELECT pk, t1c1, fk, t2c1
FROM Table2
  LEFT JOIN Table1
  ON Table2.fk = Table1.pk;
```



pk	t1c1	fk	t2c1
1	a	1	c
1	a	1	d
2	b	null	null



pk	t1c1	fk	t2c1
1	a	1	c
1	a	1	d
null	null	3	e

pk	t1c1	fk	t2c1
1	a	1	c
1	a	1	d
2	b	null	null
null	null	3	e

UNION vs UNION All – An Example (Part 2)

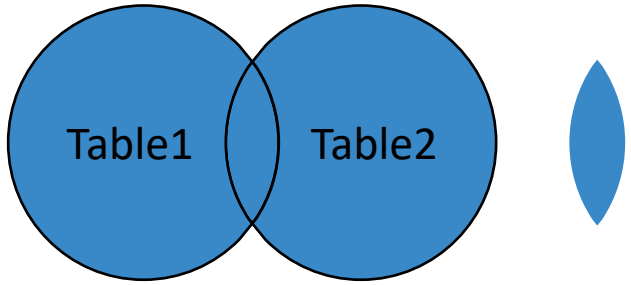


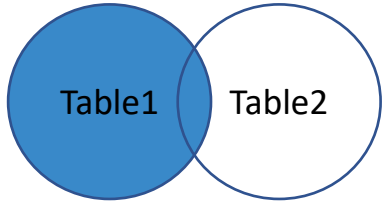
Table1

pk	t1c1
1	a
2	b

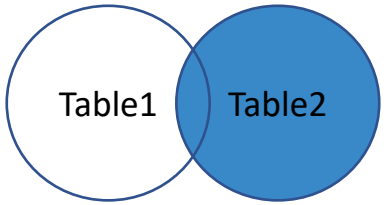
Table2

fk	t2c1
1	c
1	d
3	e

```
SELECT pk, t1c1, fk, t2c1
FROM Table1
  LEFT JOIN Table2
  ON Table1.pk = Table2.fk
UNION ALL
SELECT pk, t1c1, fk, t2c1
FROM Table2
  LEFT JOIN Table1
  ON Table2.fk = Table1.pk;
```



pk	t1c1	fk	t2c1
1	a	1	c
1	a	1	d
2	b	null	null



pk	t1c1	fk	t2c1
1	a	1	c
1	a	1	d
null	null	3	e

pk	t1c1	fk	t2c1
1	a	1	c
1	a	1	d
2	b	null	null
1	a	1	c
1	a	1	d
null	null	3	e

Others

- CTE and temporary table
- Self-join
- CASE keyword
- UNION keyword

Many things we didn't cover

- Insert data (`INSERT INTO…VALUES…; INSERT INTO…SELECT…FROM…`)
- Update data (`UPDATE…SET…WHERE…`)
- Delete data (`DELETE FROM…WHERE…`)
- Manipulate tables (`CREATE TABLE…; ALTER TABLE…; DROP TABLE…`)
- Views (`CREATE VIEW…AS…`)

Ref. SQLite Tutorial - An Easy Way to Master SQLite Fast

The list goes on and on

- Stored procedures (not supported in SQLite)
- User-defined Functions (not supported in SQLite)
- Transaction processing
- Cursors (going through table row by row)
- WINDOW functions (SQLite >= version 3.25)
- Query optimization
- DB permissions & security
- …

Ref. A stack overflow discussion on What is “Advanced” SQL.