SQL Fundamentals for Data

Thomas Nield
O'Reilly Media

About the Speaker

Thomas Nield

Business Consultant for Southwest Airlines in Schedule Initiatives

Author of *Getting Started with SQL* by O'Reilly and *Learning RxJava* by Packt

Teaches a few online trainings at O'Reilly

SQL Fundamentals for Data

Advanced SQL for Data Analysis

Reactive Python for Data Science


thomasnield9727


https://github.com/thomasnield

What to Expect in the Next Two Days

1. The role of databases and SQL in the IT/business landscape
2. Understand relational databases
3. Query and transform data with SQL
4. Database Design
5. Writing data in tables

Why Learn SQL?

Business and Technology professionals can both reap benefits from learning SQL.

SQL is a highly lucrative skill to have according to StackOverflow's Annual Survey.

It can be utilized and open up many career paths in both business and IT.

- Business Side - Analytical, managerial, strategic, research, and project roles
- Technology Side - Database design, database administration (DBA), systems engineering, IT project management, and software development

Section I

Introduction to Databases

What is a Database?

Broad definition: A *database* is anything that collects and organizes data

Examples:

- Excel spreadsheets
- Text files (CSV, XML, JSON)
- File cabinet with organized documents

When referred to professionally, a database is typically a Relational Database Management System (RDBMS)

Understanding Relational Databases

- A *Relational Database Management System* is simply a type of database holding tables that may have relationships
- A field in a table can point to another table for information

CUSTOMER_ORDER

ORDER_ID	ORDER_DATE	SHIP_DATE	CUSTOMER_ID	PRODUCT_ID	ORDER_QTY	SHIPPED
1	2015-05-15	2015-05-18	1	1	450	false
2	2015-05-18	2015-05-21	3	2	600	false
3	2015-05-20	2015-05-23	3	5	300	false
4	2015-05-18	2015-05-22	5	4	375	false
5	2015-05-17	2015-05-20	3	2	500	false

CUSTOMER

CUSTOMER_ID	NAME	REGION	STREET_ADDRESS	CITY	STATE	ZIP
1	LITE Industrial	Southwest	729 Ravine Way	Irving	TX	75014
2	Rex Tooling Inc	Southwest	6129 Collie Blvd	Dallas	TX	75201
3	Re-Barre Construction	Southwest	9043 Windy Dr	Irving	TX	75032
4	Prairie Construction	Southwest	264 Long Rd	Moore	OK	62104
5	Marsh Lane Metal Works	Southeast	9143 Marsh Ln	Avondale	LA	79782

Why Separate Tables?

- This idea of separating different types of data (e.g. CUSTOMER versus a CUSTOMER_ORDER) is known as **normalization**
- Putting both *CUSTOMER* and *CUSTOMER_ORDER* information in one table would be bloated, redundant and difficult to maintain
- Example of a non-normalized table:

NAME	REGION	STREET_ADDRESS	CITY	STATE	ZIP	ORDER_ID	ORDER_DATE	SHIP_DATE	ORDER_QTY	SHIPPED
LITE Industrial	Southwest	729 Ravine Way	Irving	TX	75014	1	2015-05-15	2015-05-18	450	false
Re-Barre Construction	Southwest	9043 Windy Dr	Irving	TX	75032	2	2015-05-18	2015-05-21	600	false
Re-Barre Construction	Southwest	9043 Windy Dr	Irving	TX	75032	3	2015-05-20	2015-05-23	300	false
Marsh Lane Metal Works	Southeast	9143 Marsh Ln	Avondale	LA	79782	4	2015-05-18	2015-05-22	375	false
Re-Barre Construction	Southwest	9043 Windy Dr	Irving	TX	75032	5	2015-05-17	2015-05-20	500	false

Imagine if we needed to change an address.
We would have to do it three times!

Why Separate Tables?

- This is why it is better to separate the CUSTOMER and CUSTOMER_ORDER information into separate tables
- You only need to update the address in one place

CUSTOMER_ORDER

ORDER_ID	ORDER_DATE	SHIP_DATE	CUSTOMER_ID	PRODUCT_ID	ORDER_QTY	SHIPPED
1	2015-05-15	2015-05-18	1	1	450	false
2	2015-05-18	2015-05-21	3	2	600	false
3	2015-05-20	2015-05-23	3	5	300	false
4	2015-05-18	2015-05-22	5	4	375	false
5	2015-05-17	2015-05-20	3	2	500	false

CUSTOMER

CUSTOMER_ID	NAME	REGION	**STREET_ADDRESS**	CITY	STATE	ZIP
1	LITE Industrial	Southwest	729 Ravine Way	Irving	TX	75014
2	Rex Tooling Inc	Southwest	6129 Collie Blvd	Dallas	TX	75201
3	Re-Barre Construction	Southwest	9043 Windy Dr	Irving	TX	75032
4	Prairie Construction	Southwest	264 Long Rd	Moore	OK	62104
5	Marsh Lane Metal Works	Southeast	9143 Marsh Ln	Avondale	LA	79782

Exercise 1.1

- Is this table normalized?

APPOINTMENT

APPOINTMENT_ID	PATIENT_FIRST_NAME	PATIENT_LAST_NAME	BIRTH_DATE	VISIT_DATE	CHECK_IN	CHECK_OUT
1	Jonathon	Reyes	5/1/1981	5/1/2016	8:00AM	9:00AM
2	Rebecca	Giles	1/8/1976	5/3/2016	10:00AM	11:00AM
3	Sam	Johnson	9/3/1985	5/4/2016	9:00AM	10:00AM
4	Jonathon	Reyes	5/1/1981	5/1/2016	1:00PM	2:00PM
5	Jonathon	Reyes	5/1/1981	6/18/2016	3:00PM	4:00PM
6	Rebecca	Giles	1/8/1976	5/3/2016	11:00AM	12:00PM

Exercise 1.1

- Is this table normalized? **It is not!**

APPOINTMENT

APPOINTMENT_ID	PATIENT_FIRST_NAME	PATIENT_LAST_NAME	BIRTH_DATE	VISIT_DATE	CHECK_IN	CHECK_OUT
1	Jonathon	Reyes	5/1/1981	5/1/2016	8:00AM	9:00AM
2	Rebecca	Giles	1/8/1976	5/3/2016	10:00AM	11:00AM
3	Sam	Johnson	9/3/1985	5/4/2016	9:00AM	10:00AM
4	Jonathon	Reyes	5/1/1981	5/1/2016	1:00PM	2:00PM
5	Jonathon	Reyes	5/1/1981	6/18/2016	3:00PM	4:00PM
6	Rebecca	Giles	1/8/1976	5/3/2016	11:00AM	12:00PM

Exercise 1.1

PATIENT and APPOINTMENT data should be in separate tables

PATIENT

PATIENT_ID	PATIENT_FIRST_NAME	PATIENT_LAST_NAME	BIRTH_DATE
1	Jonathon	Reyes	5/1/1981
2	Rebecca	Giles	1/8/1976
3	Sam	Johnson	9/3/1985

APPOINTMENT

APPOINTMENT_ID	PATIENT_ID	VISIT_DATE	CHECK_IN	CHECK_OUT
1	1	5/1/2016	8:00AM	9:00AM
2	2	5/3/2016	10:00AM	11:00AM
3	3	5/4/2016	9:00AM	10:00AM
4	1	5/1/2016	1:00PM	2:00PM
5	1	6/18/2016	3:00PM	4:00PM
6	2	5/3/2016	11:00AM	12:00PM

Exercise 1.1

PATIENT and APPOINTMENT data should be in separate tables

PATIENT

PATIENT_ID	PATIENT_FIRST_NAME	PATIENT_LAST_NAME	BIRTH_DATE
1	Jonathon	Reyes	5/1/1981
2	Rebecca	Giles	1/8/1976
3	Sam	Johnson	9/3/1985

APPOINTMENT

APPOINTMENT_ID	PATIENT_ID	VISIT_DATE	CHECK_IN	CHECK_OUT
1	1	5/1/2016	8:00AM	9:00AM
2	2	5/3/2016	10:00AM	11:00AM
3	3	5/4/2016	9:00AM	10:00AM
4	1	5/1/2016	1:00PM	2:00PM
5	1	6/18/2016	3:00PM	4:00PM
6	2	5/3/2016	11:00AM	12:00PM

Types of Databases

- Relational databases and SQL are not proprietary to one company or organization
- Many companies and organizations have created their own relational database software

MySQL	**Microsoft Access**	**SQLite**
Oracle	**Microsoft SQL Server**	**MariaDB**
IBM DB2	**PostgreSQL**	**SAP Sybase**

- Do not be confused by “SQL” being used to brand database software, like Microsoft SQL Server, MySQL, and SQLite. SQL is the universal language used on all RDBMS platforms

NoSQL and “Big Data”

- **NoSQL** stands for *not only SQL*, and is often used to describe “Big Data” platforms that may leverage SQL but are not relational.
  - NoSQL databases include MongoDB, Couchbase, Apache Cassandra, and Redis.
  - These platforms store massive amounts of data in a variety of raw and unstructured formats (e.g. *documents, key-value*).
  - Most of these solutions are **distributed** across multiple machines, which is difficult to do with relational databases.
- Other “Big Data” solutions such as Apache Hadoop and Apache Spark can be interacted with using SQL, but are not limited to relational databases.
- Therefore most of the knowledge in this course can be applied to “Big Data” solutions.
- Caution using NoSQL and Big Data: “When all you have is a hammer, everything starts to look like a nail.”
  - Do not fall into the trap of treating all data problems as Big Data problems, because most are not.
  - While Big Data will continue to grow, data will always come in all shapes and sizes.

SQL vs NoSQL

Feature	SQL	NoSQL	Winner
Integrity/Consistency	Data is enforced with logical relationships, minimized redundancy, and “Up-to-date” consistency.	Simple key-value and document storage does not enforce any rules or structure. Redundancy and write latency is common.	SQL
Design changes	Easy to “add” to database, but harder to modify.	NoSQL can quickly and arbitrarily change what data it stores.	NoSQL
Analysis	SQL is a universal language that makes accessing and analyzing data simple.	SQL support is sparse, and proprietary languages are esoteric and hardly universal.	SQL
Programming	Programmers of Java, Python, and .NET have to map entities to tables, which can be tedious. But data integrity is given.	Programming against a NoSQL database is quick and simple, but onus is on programmer to validate data.	*Draw*
Performance	Relational databases can store data for most use cases, but struggle with true “big data” cases. Integrity constraints also slow down performance.	NoSQL is capable of storing vast amounts of data with horizontal scaling. It also performs quickly due to horizontal scaling and no integrity constraints.	NoSQL

TL;DR

SQL = integrity and accuracy

NoSQL = speed and scalability

SQL should be a prerequisite before learning NoSQL and “Big data”.

If you are absolutely uncertain which to use, always start with SQL.

Lightweight vs Centralized Databases

Lightweight Databases

- When you want a simple solution for a small number of users, lightweight databases are a great place to start
- They store data in a file that can be shared, but can break down when edited simultaneously
- Common Lightweight Databases
    - Microsoft Access
    - SQLite
    - H2

Centralized Databases

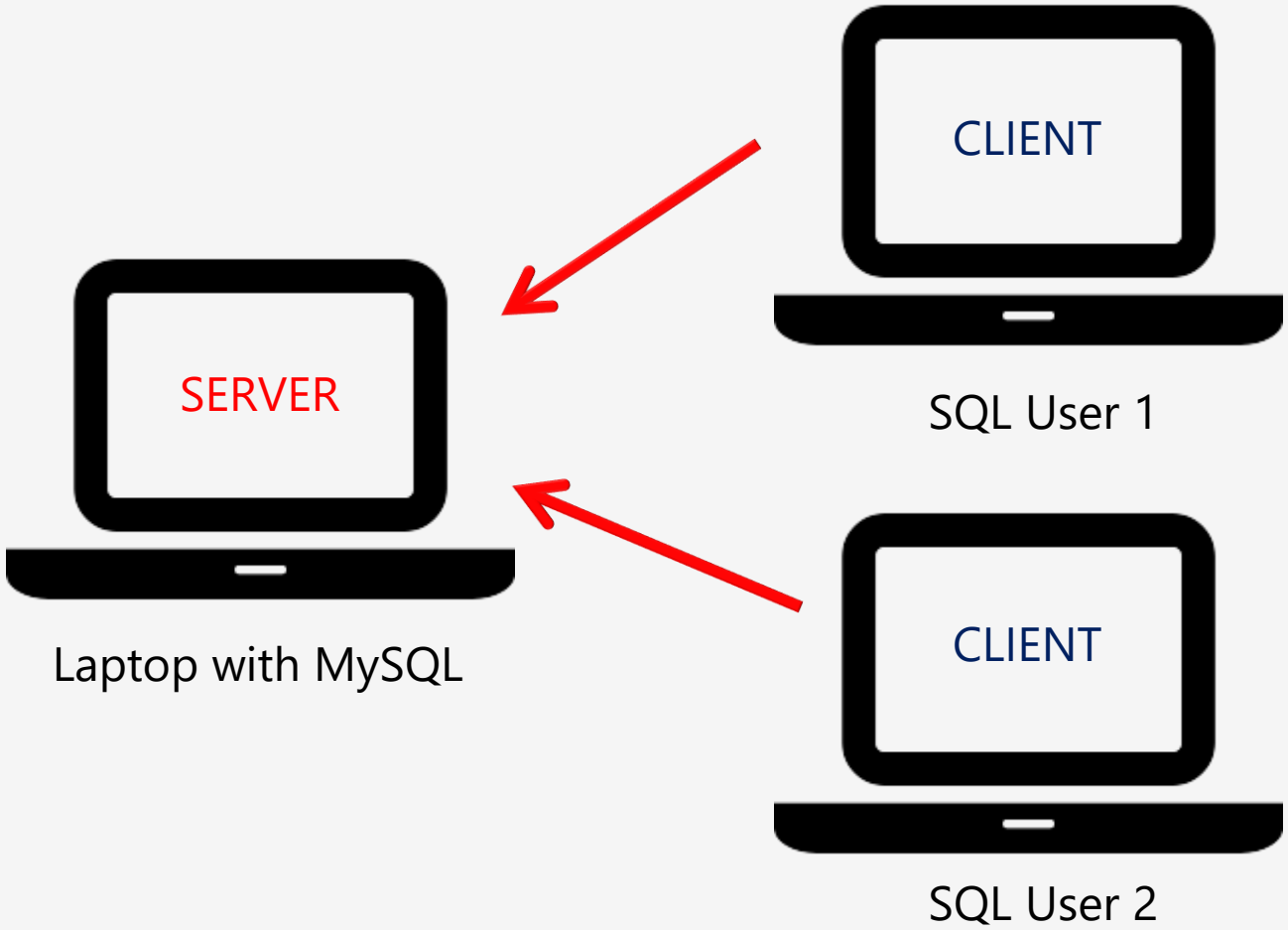
When you need to support tens, hundreds, or thousands of users and applications, you need a centralized database

These databases are designed to handle a high volume of traffic efficiently

Some examples of centralized database platforms

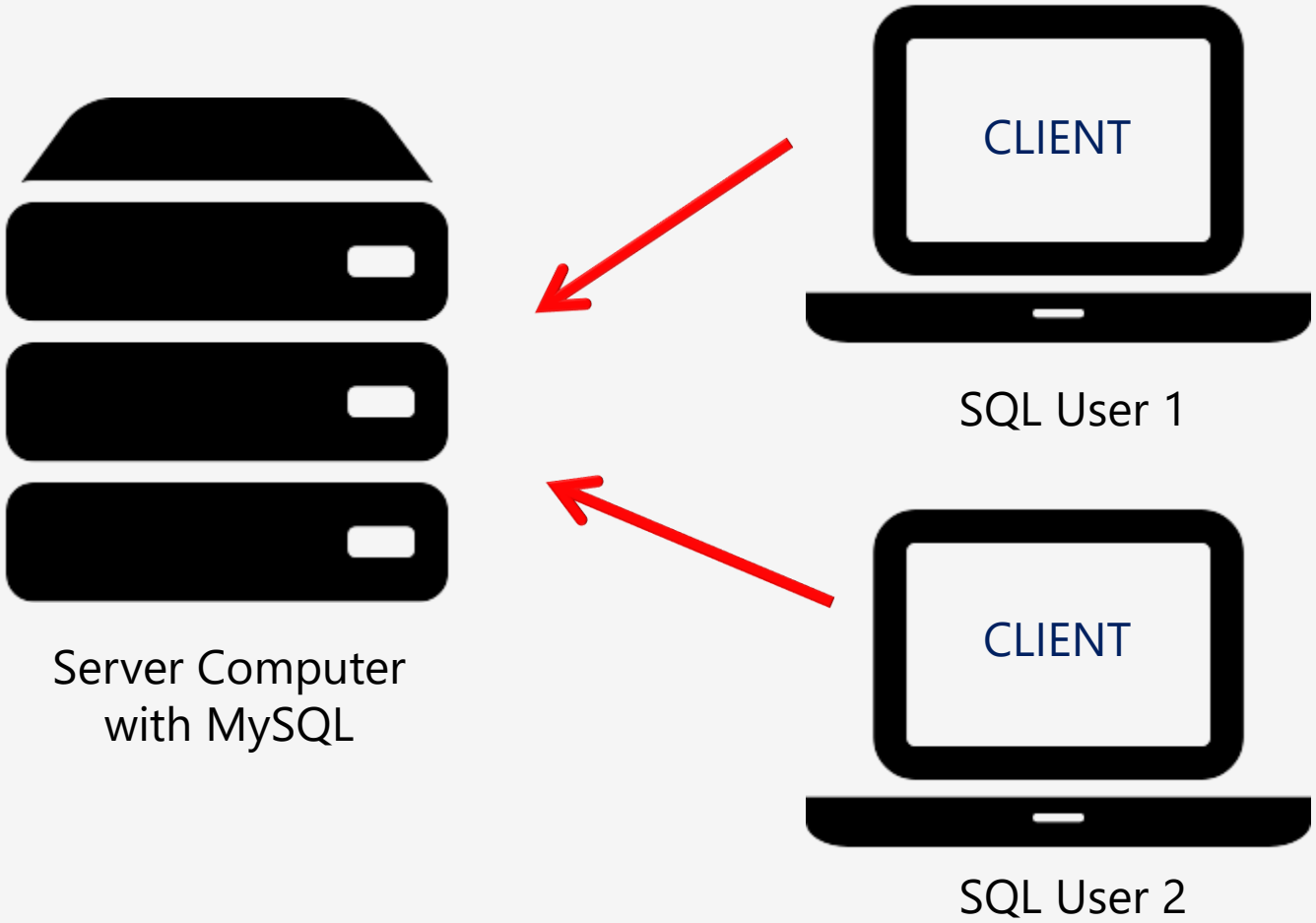
- Oracle
- Microsoft SQL Server
- MySQL
- PostgreSQL
- Teradata

Typical Centralized Database Setup



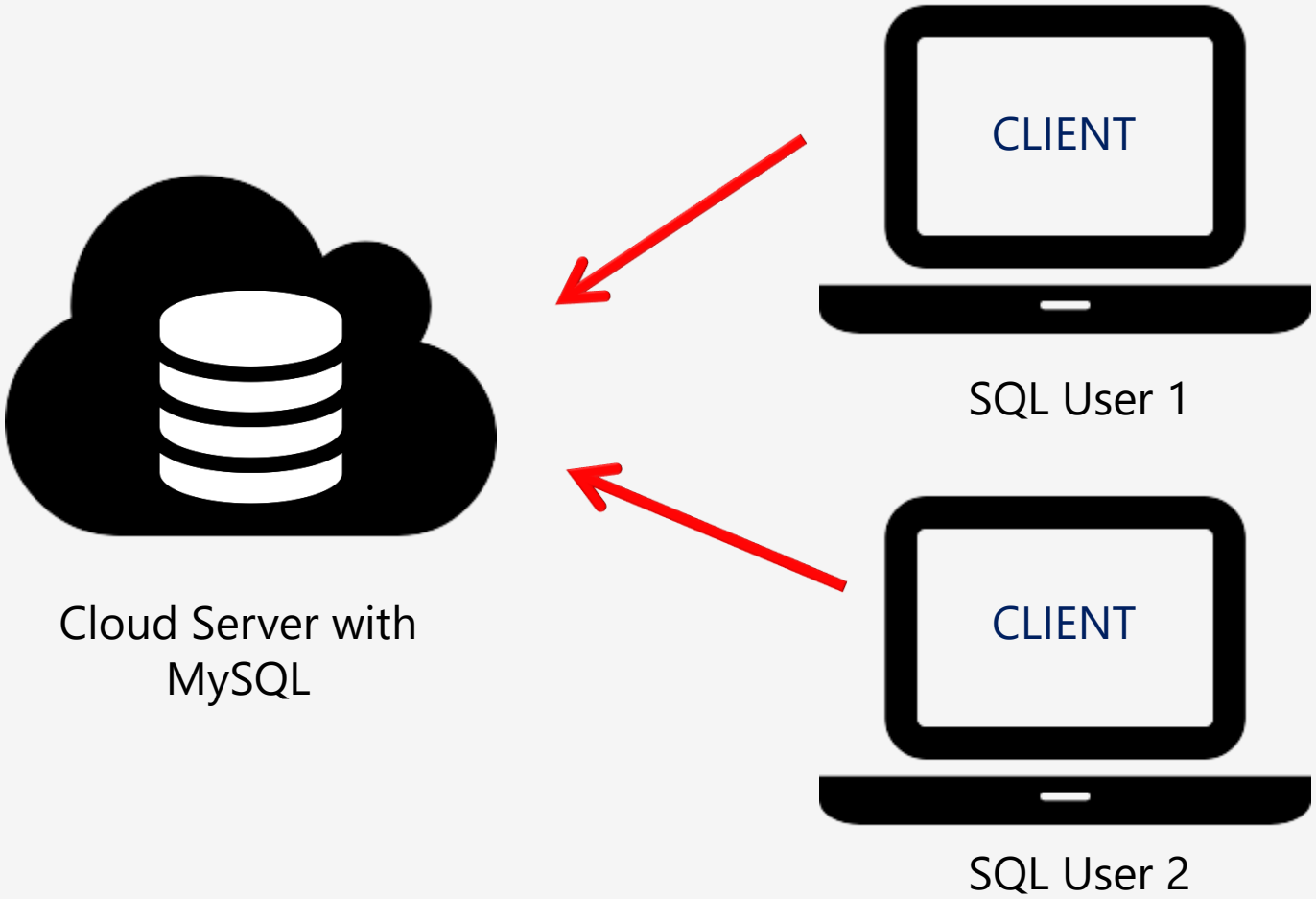
Centralized databases use a Client/Server Setup

Typical Centralized Database Setup



For production, you typically use a server computer to host the database rather than a laptop or desktop

Typical Centralized Database Setup



A popular architecture nowadays is to have cloud services from Amazon, Google, or Heroku host your database for you.

What We Will Use

Upon entering a workplace, there is a good chance you will need access to an existing centralized database

We will not be using centralized databases in this course, but we will be using SQLite

The experience between lightweight and centralized databases should largely be the same

Exercise 1.2

Identify the following as being a *lightweight* or *centralized* database:

1. Facebook's MySQL database holding all user data

   CENTRALIZED

2. A SQLite database holding an iPhone user's data *locally* on the hard drive

   LIGHTWEIGHT

3. An Oracle database with shopping data for an e-commerce site

   CENTRALIZED

Section II

SQLite

SQLite

- We will be using SQLite with SQLiteStudio in this course
- SQLite is a lightweight database and can be found on:
  - Android, iPhone, iPad, and Windows 10
  - Car consoles, thermostats, and other gadgets
  - Satellites and the Airbus A350 XWB
  - SQLite excels where simplicity and low overhead is needed

SQLiteStudio

- We will be using SQLiteStudio to work with SQLite database files
- SQLiteStudio can be downloaded at http://sqlitestudio.pl/
- On Windows and Linux, download and copy the folder contents to a location of your choice
- On Mac, you can either drag the DMG to the Applications folder or use Homebrew http://macappstore.org/sqlitestudio/

Database Files

- The database files can be found on GitHub https://github.com/thomasnield/oreilly_getting_started_with_sql
- Click the *Clone or Download* button in the top-right, and then *Download ZIP* to download all the database files at once
- Open the database files in SQLiteStudio

Break and Q&A

Section III

SELECT

Basic Math Operators

Operator	Description	Example
+	Adds two numbers	STOCK + NEW_SHIPMENT
-	Subtracts two numbers	STOCK - DEFECTS
*	Multiplies two numbers	PRICE * 1.07
/	Divides two numbers	STOCK / PALLET_SIZE
%	Divides two numbers, but returns the remainder	STOCK % PALLET_SIZE

Exercise 3.1

SELECT all records (with all fields) from the CUSTOMER_ORDER table

ANSWER:

```
SELECT * FROM CUSTOMER_ORDER;
```

Exercise 3.2

SELECT the ORDER_ID and SHIP_DATE fields from the CUSTOMER_ORDER table

ANSWER:

```
SELECT ORDER_ID, SHIP_DATE FROM CUSTOMER_ORDER;
```

Exercise 3.3

SELECT the PRODUCT_ID, DESCRIPTION, and a REDUCED_PRICE (which subtracts $1.10 from each PRICE) from the PRODUCT table

ANSWER:

```
SELECT PRODUCT_ID,

DESCRIPTION,

PRICE - 1.10 as REDUCED_PRICE

FROM PRODUCT
```

Section IV

WHERE

Exercise 4.1

SELECT all records where TEMPERATURE is between 30 and 50 degrees

```
SELECT * FROM station_data
WHERE temperature BETWEEN 30 AND 50;
```

OR

```
SELECT * FROM station_data
WHERE temperature >= 30 and temperature <= 50;
```

Exercise 4.1

SELECT all records where TEMPERATURE is between 30 and 50 degrees

```
SELECT * FROM station_data
WHERE temperature BETWEEN 30 AND 50;
```

OR

```
SELECT * FROM station_data
WHERE temperature >= 30 and temperature <= 50;
```

Exercise 4.2

SELECT all records where station_pressure is greater than 1000 and a tornado was present

```
SELECT * FROM STATION_DATA

WHERE station_pressure > 1000 AND tornado;
```

OR

```
SELECT * FROM STATION_DATA

WHERE station_pressure > 1000 AND tornado = 1;
```

Exercise 4.3

SELECT all records with report codes E6AED7, B950A1, 98DDAD

```
SELECT * FROM STATION_DATA
WHERE report_code IN ('E6AED7','B950A1','98DDAD')
```

OR

```
SELECT * FROM STATION_DATA
WHERE report_code = 'E6AED7‘
OR report_code = 'B950A1‘
OR report_code = '98DDAD'
```

Exercise 4.4

SELECT all records with report codes E6AED7, B950A1, 98DDAD

```
SELECT * FROM STATION_DATA
WHERE report_code IN ('E6AED7','B950A1','98DDAD');
```

OR

```
SELECT * FROM STATION_DATA
WHERE report_code = 'E6AED7'
OR report_code = 'B950A1'
OR report_code = '98DDAD';
```

Exercise 4.5

SELECT all records WHERE station_pressure is null

```
SELECT * FROM STATION_DATA

WHERE station_pressure IS NULL;
```

Section V

GROUP BY and ORDER BY

Exercise 5.1

Find the SUM of precipitation by year when a tornado was present, and sort by year descending.

ANSWER:

```
SELECT year,
SUM(precipitation) as tornado_precipitation
FROM station_data
WHERE tornado = 1
GROUP BY year
ORDER BY year DESC
```

Exercise 5.2

SELECT the year and max snow depth, but only years where the max snow depth is at least 50.

ANSWER:

```
SELECT year,
max(snow_depth) AS max_snow_depth
FROM STATION_DATA
GROUP BY year
HAVING max_snow_depth >= 50
```

Section VI

CASE

Exercise 6.1

SELECT the report_code, year, quarter, and temperature, where a "quarter" is "Q1", "Q2", "Q3", or "Q4" reflecting months 1-3, 4-6, 7-9, and 10-12 respectively.

```
SELECT

report_code,
year,

CASE
    WHEN month BETWEEN 1 and 3 THEN "Q1"
    WHEN month BETWEEN 4 and 6 THEN "Q2"
    WHEN month BETWEEN 7 and 9 THEN "Q3"
    WHEN month BETWEEN 10 and 12 THEN "Q4"
END as quarter,

temperature

FROM STATION_DATA
```

Exercise 6.2

Get the average temperature grouped by quarter and year, where a "quarter" is "Q1", "Q2", "Q3", or "Q4" reflecting months 1-3, 4-6, 7-9, and 10-12 respectively.

```
SELECT
year,

CASE
    WHEN month BETWEEN 1 and 3 THEN "Q1"
    WHEN month BETWEEN 4 and 6 THEN "Q2"
    WHEN month BETWEEN 7 and 9 THEN "Q3"
    WHEN month BETWEEN 10 and 12 THEN "Q4"
END as quarter,

AVG(temperature) as avg_temp

FROM STATION_DATA
GROUP BY 1,2
```

Section VII

JOIN

Revisiting Table Relationships

- Remember when we were talking about tables having relationships with each other?

CUSTOMER_ORDER

ORDER_ID	ORDER_DATE	SHIP_DATE	CUSTOMER_ID	PRODUCT_ID	ORDER_QTY	SHIPPED
1	2015-05-15	2015-05-18	1	1	450	false
2	2015-05-18	2015-05-21	3	2	600	false
3	2015-05-20	2015-05-23	3	5	300	false
4	2015-05-18	2015-05-22	5	4	375	false
5	2015-05-17	2015-05-20	3	2	500	false

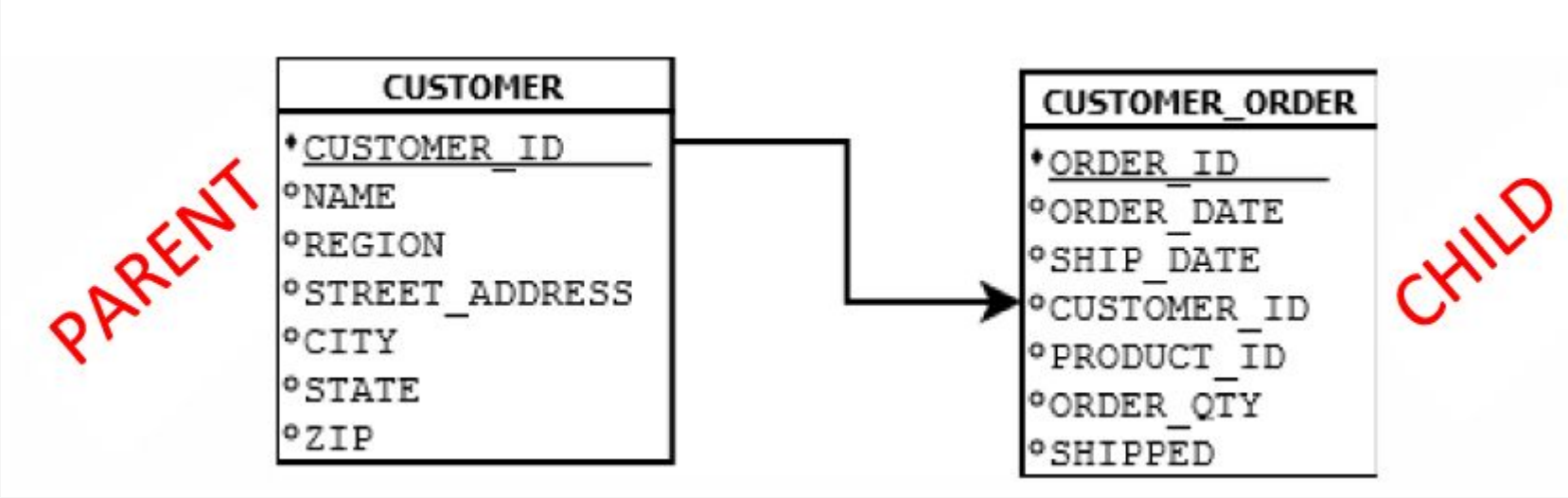
CUSTOMER

CUSTOMER_ID	NAME	REGION	STREET_ADDRESS	CITY	STATE	ZIP
1	LITE Industrial	Southwest	729 Ravine Way	Irving	TX	75014
2	Rex Tooling Inc	Southwest	6129 Collie Blvd	Dallas	TX	75201
3	Re-Barre Construction	Southwest	9043 Windy Dr	Irving	TX	75032
4	Prairie Construction	Southwest	264 Long Rd	Moore	OK	62104
5	Marsh Lane Metal Works	Southeast	9143 Marsh Ln	Avondale	LA	79782

- A table can supply data to another table, like CUSTOMER information for a CUSTOMER_ORDER

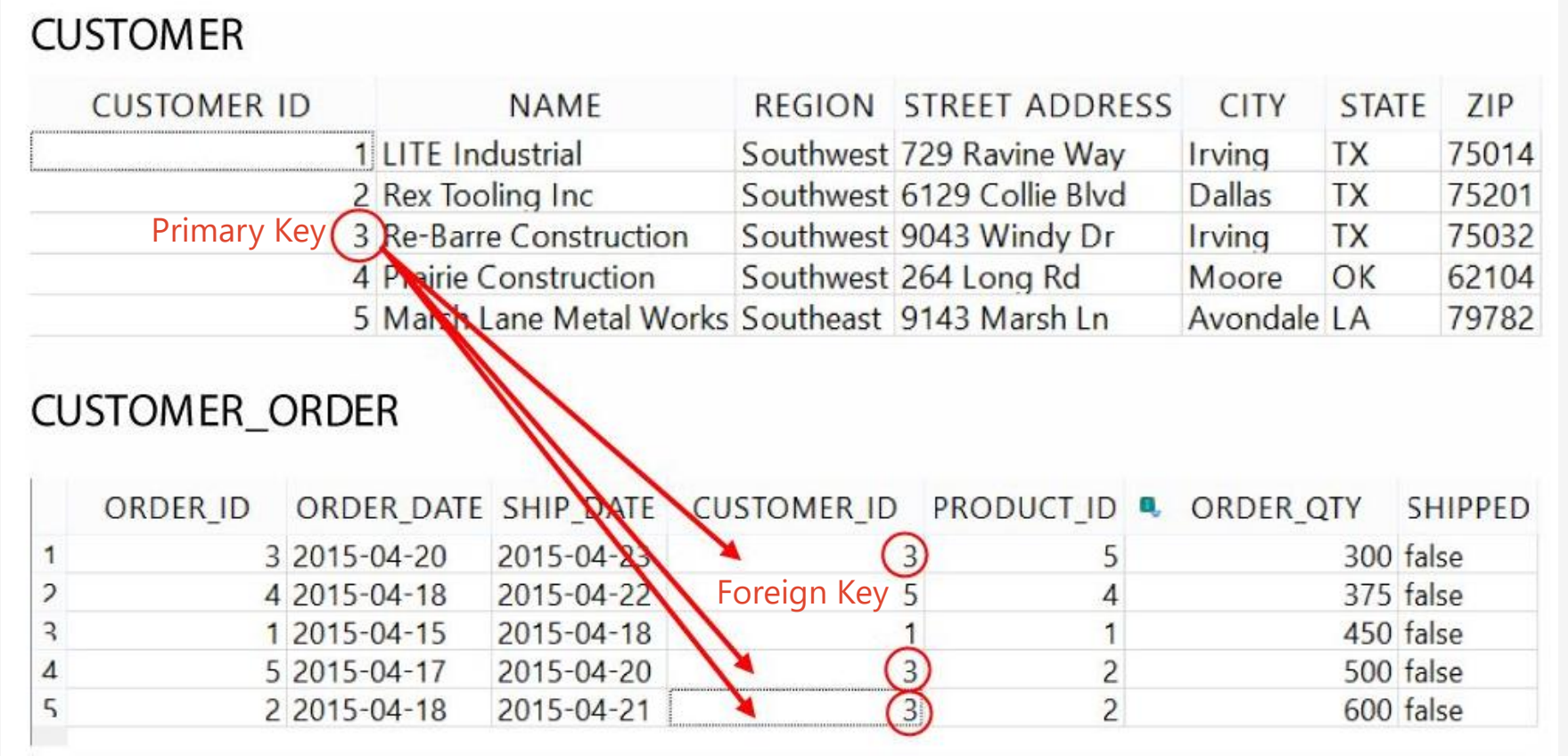
Parent/Child Tables

- Because the CUSTOMER table supplies data to CUSTOMER_ORDER, it is the **parent** table to CUSTOMER_ORDER
- Because the CUSTOMER_ORDER table receives data from CUSTOMER, it is the **child** table to CUSTOMER



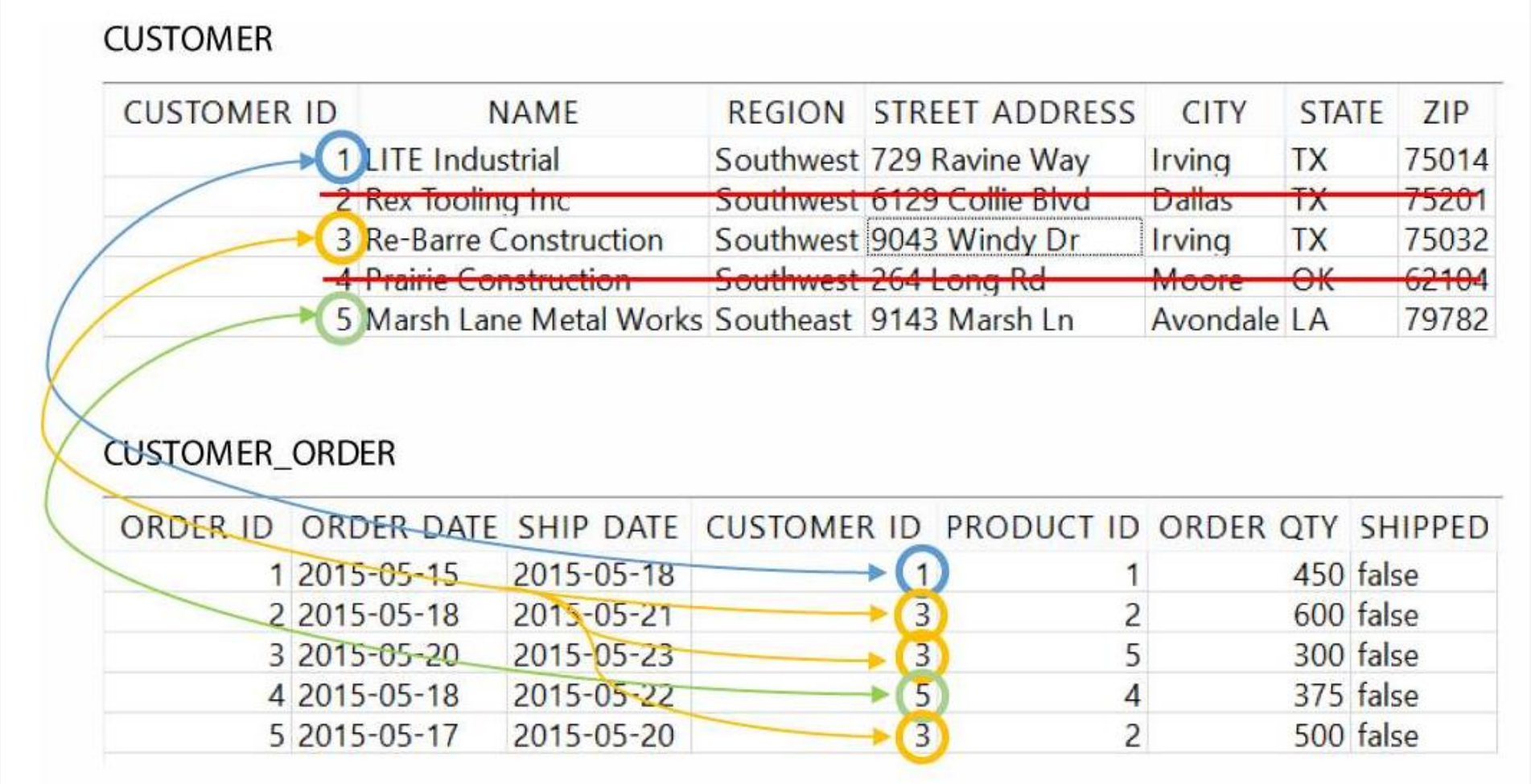
Primary/Foreign Keys

- Typically, a parent table will have a **primary key** and the child table will have a **foreign key**.

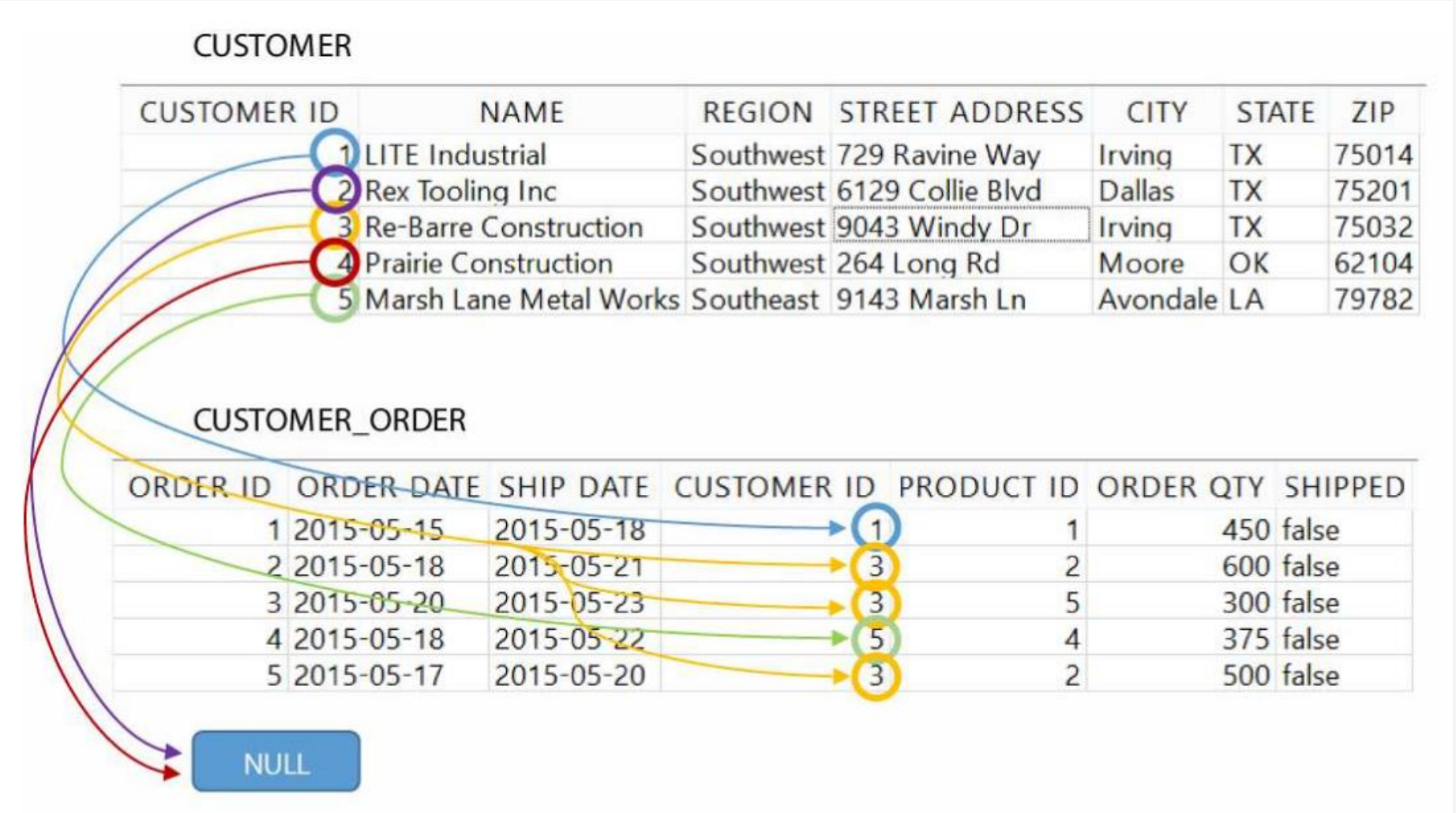


- The primary key is unique and can map to multiple foreign keys

INNER JOIN


CUSTOMER
CUSTOMER ID NAME REGION STREET ADDRESS CITY STATE ZIP
1 LITE Industrial Southwest 729 Ravine Way Irving TX 75014
2 Rex Tooling Inc Southwest 6129 Collie Blvd Dallas TX 75201
3 Re-Barre Construction Southwest 9043 Windy Dr Irving TX 75032
4 Prairie Construction Southwest 264 Long Rd Moore OK 62104
5 Marsh Lane Metal Works Southeast 9143 Marsh Ln Avondale LA 79782
CUSTOMER_ORDER
ORDER ID ORDER DATE SHIP DATE CUSTOMER ID PRODUCT ID ORDER QTY SHIPPED
1 2015-05-15 2015-05-18 1 1 450 false
2 2015-05-18 2015-05-21 3 2 600 false
3 2015-05-20 2015-05-23 3 5 300 false
4 2015-05-18 2015-05-22 5 4 375 false
5 2015-05-17 2015-05-20 3 2 500 false

LEFT OUTER JOIN

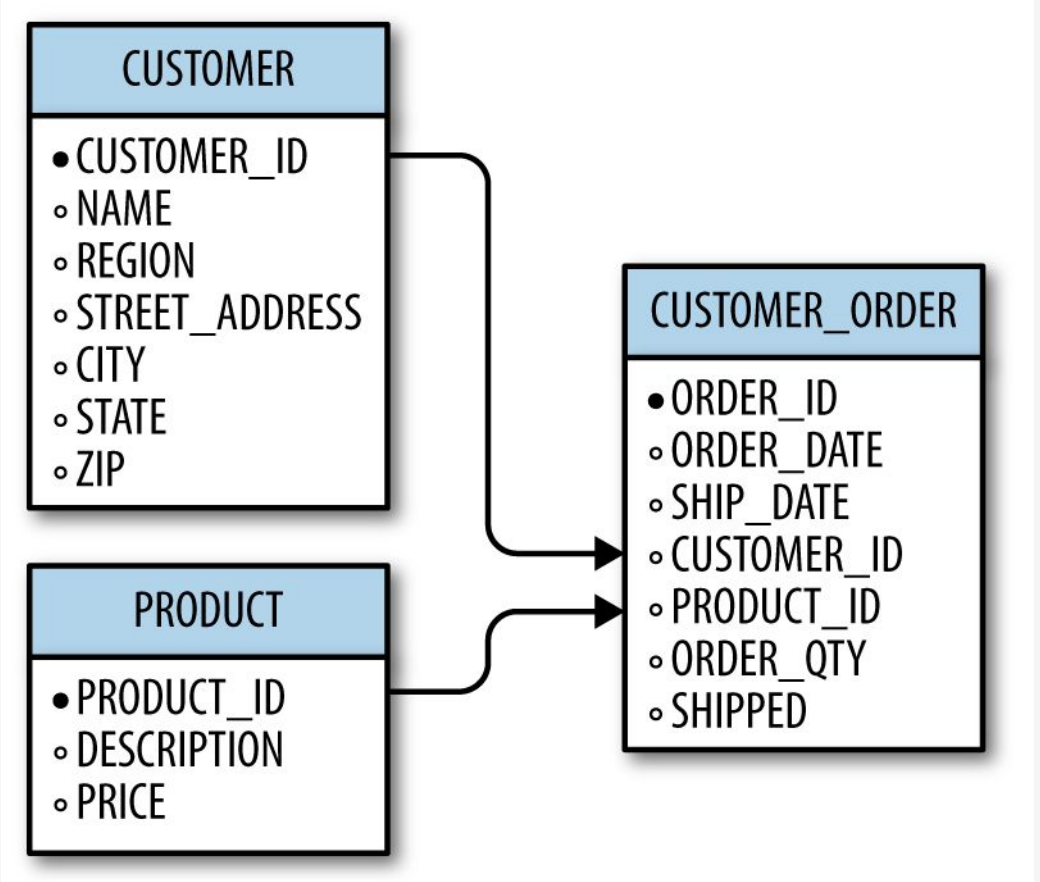


LEFT OUTER JOIN

```
SELECT CUSTOMER.CUSTOMER_ID,
NAME,
STREET_ADDRESS,
CITY,
STATE,
ZIP,
ORDER_DATE,
SHIP_DATE,
ORDER_ID,
PRODUCT_ID,
ORDER_QTY

FROM CUSTOMER LEFT JOIN CUSTOMER_ORDER
ON CUSTOMER.CUSTOMER_ID = CUSTOMER_ORDER.CUSTOMER_ID
```

Joining Multiple Tables

- It is not uncommon to have a table be a parent to one table, but a child to another
- A given table can also be a child to more than one table, so what does this look like?
- We can bring in a third table PRODUCT to supply product information to CUSTOMER_ORDER



Exercise 7.1

SELECT the ORDER_ID, ORDER_DATE, and DESCRIPTION (from PRODUCT)
(hint, you will need to INNER JOIN CUSTOMER_ORDER and PRODUCT)

ANSWER:

```
SELECT ORDER_ID, ORDER_DATE, DESCRIPTION

FROM CUSTOMER_ORDER INNER JOIN PRODUCT
ON CUSTOMER_ORDER.PRODUCT_ID = PRODUCT.PRODUCT_ID
```

Exercise 7.2

Find the total revenue by product. Include the fields PRODUCT_ID, DESCRIPTION, and then the TOTAL_REVENUE.

(Hint: you will need to join CUSTOMER_ORDER and PRODUCT. Then do a GROUP BY)

ANSWER:

```
SELECT PRODUCT_ID,
DESCRIPTION,
COALESCE(SUM (ORDER_QTY * PRICE), 0) AS TOTAL_REVENUE

FROM PRODUCT LEFT JOIN CUSTOMER_ORDER
ON PRODUCT.PRODUCT_ID = CUSTOMER_ORDER.PRODUCT_ID
GROUP BY 1, 2
```

Section VIII

Database Design

Planning a Database

Design Questions

- *What are the business requirements?*
- *What tables will I need to fulfill those requirements?*
- *What columns will each table contain?*
- *How will the tables be normalized?*
- *What will their parent/child relationships be?*

Planning a Database

Data Questions

- *How much data will be populated into these tables?*
- *Who/what will populate data into these tables?*
- *Where will the data come from?*
- *Do we need processes to automatically populate these tables?*

Planning a Database

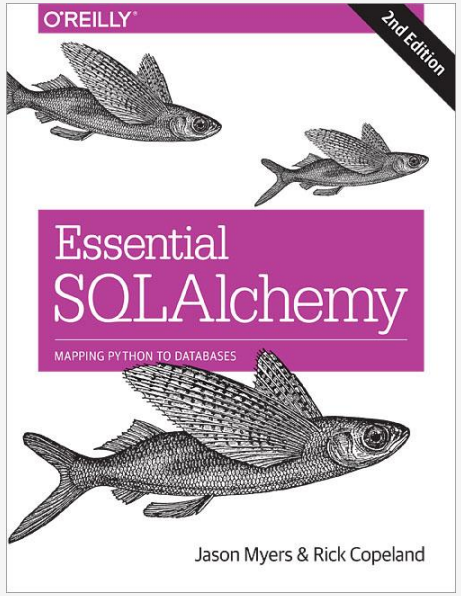
Security Questions

- *Who should have access to this database?*
- *Who should have access to which tables? Read-only access? Write access?*
- *Is this database critical to business operations?*
- *What backup plans do we have in the event of disaster/failure?*
- *Should changes to tables be logged?*
- *If the database is used for websites or web applications, is it secure?*

Preventing SQL Injection

- To prevent SQL injection, *never* concatenate a SQL string with parameters
- Instead, use the right tools and libraries to safely inject parameters for you
- *For Python, use SQLAlchemy*

```
from sqlalchemy import create_engine, text

engine = create_engine('sqlite:///C:\\Users\\thoma\\Dropbox\\rexon_metals.db')
conn = engine.connect()


def customer_for_id(customer_id):
    stmt = text("SELECT * FROM CUSTOMER WHERE CUSTOMER_ID = :id")
    return conn.execute(stmt, id=customer_id).fetchone()


print(customer_for_id(2))
```



More info at:
http://www.sqlalchemy.org/

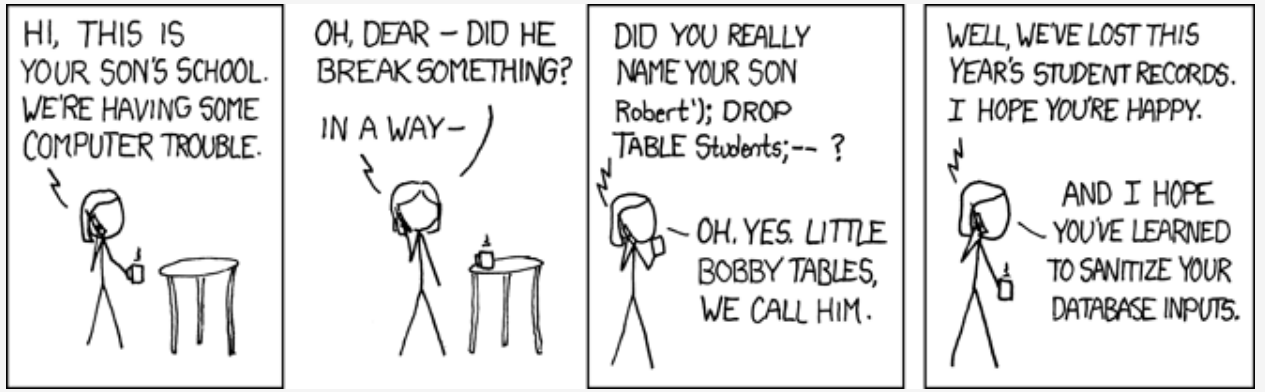
Preventing SQL Injection

For Java, Scala, Kotlin, and other JVM languages use JDBC's PreparedStatement

```
int customerId = 2;

Connection connection =
        DriverManager.getConnection("jdbc:sqlite:C:\\Users\\thoma\\Dropbox\\rexon_metals.db");

String sql = "SELECT * FROM CUSTOMER WHERE CUSTOMER_ID = ?";

PreparedStatement ps = connection.prepareStatement(sql);
ps.setInt(1,customerId);

ResultSet rs = ps.executeQuery();
rs.next();

System.out.println(rs.getInt("CUSTOMER_ID") + " " +  rs.getString("NAME"));

connection.close();
```

More info at:
http://tutorials.jenkov.com/jdbc/index.html
http://www.marcobehler.com/make-it-so-java-db-connections-and-transactions

SQL Injection Humor



Credit: https://xkcd.com/327/

SQL Injection in the News

- This couple cannot do the simplest things online because their last name is 'Null'

  https://thenextweb.com/insider/2016/03/27/last-name-null-is-tough-for-computers/

- Catholic financial services hacked, 130K accounts exposed

  http://www.twincities.com/2017/10/16/catholic-united-financial-data-breach-may-have-affected-nearly-130k-accounts/

- South Africa's massive data breach

  https://www.moneyweb.co.za/news/tech/revealed-the-real-source-of-sas-massive-data-breach/

- TalkTalk gets record £400K fine for failing to prevent October 2015 attack

  https://ico.org.uk/about-the-ico/news-and-events/news-and-blogs/2016/10/talktalk-gets-record-400-000-fine-for-failing-to-prevent-october-2015-attack/

The SurgeTech Conference

Let's design a database for a real-world scenario!

You are a staff member for the SurgeTech conference, a gathering of tech startup companies seeking publicity and investors. The organizer has tasked you with creating a database to manage the attendees, companies, presentations, rooms, and presentation attendance. How should this database be designed?

There are five entities here that can be turned into tables

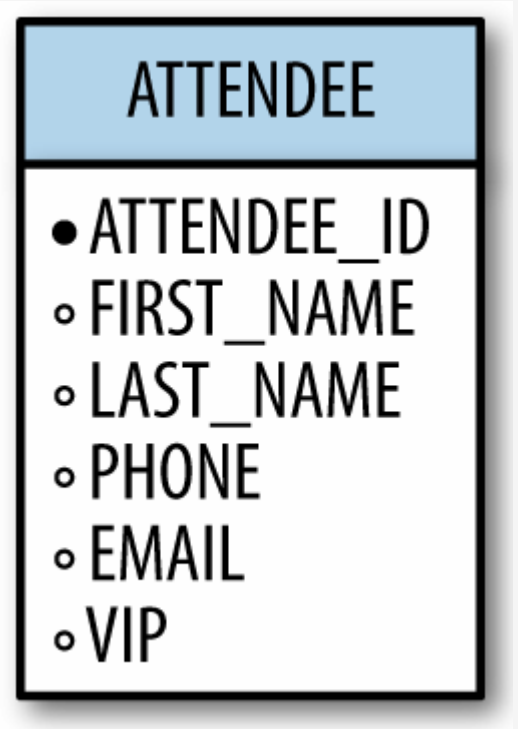
- ATTENDEE
- COMPANY
- PRESENTATION
- ROOM
- PRESENTATION_ATTENDANCE

ATTENDEE

The attendees are guests (including some VIP's) who have registered for the conference

Each attendee holds the following information:

- ID
- Name
- Phone Number
- Email
- VIP status



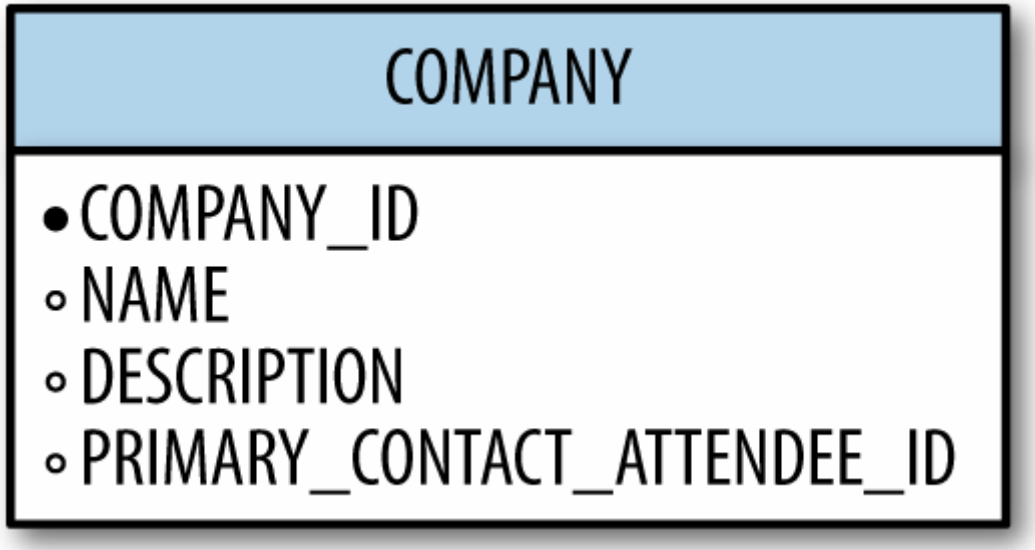
To the right is our design for the ATTENDEE table

COMPANY

The startup companies need to be tracked as well

Each company holds the following information:

- Company ID
- Name
- Description
- Primary contact attendee ID



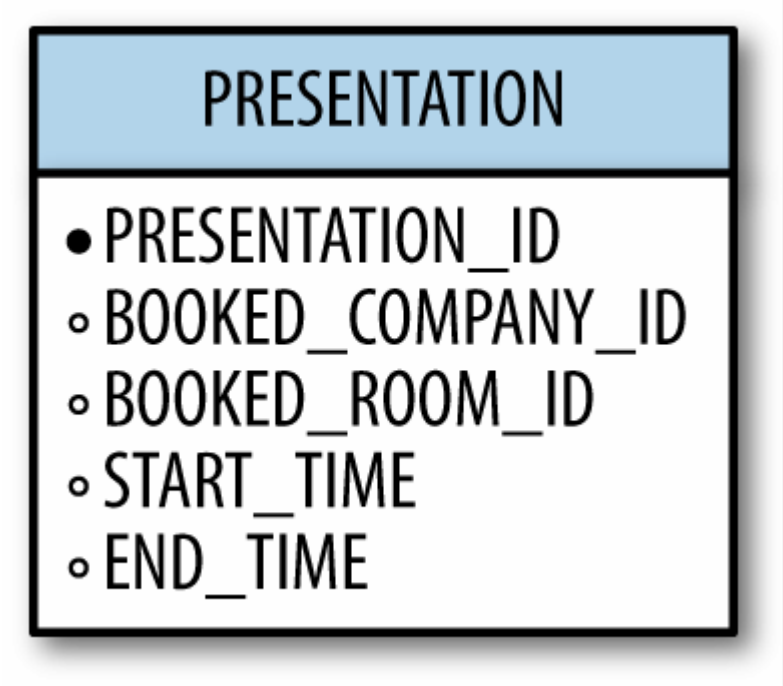
To the right is our design for the COMPANY table

PRESENTATION

Some companies will schedule a presentation for a specific slot of time

Each presentation is defined by:

- Presentation ID
- Booked company ID
- Booked room ID
- Start time
- End time



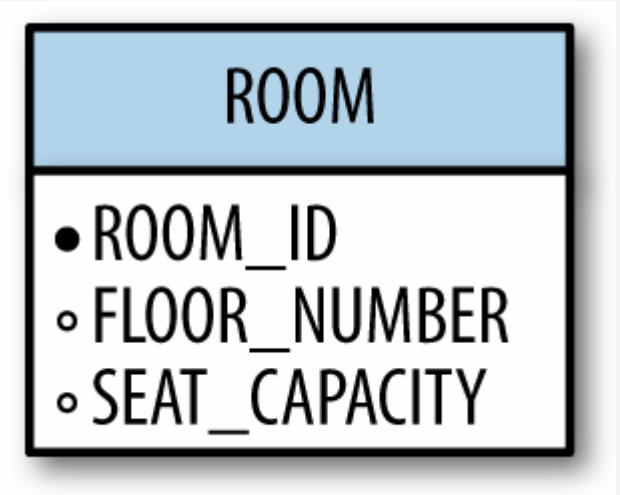
To the right is our design for the PRESENTATION table

ROOM

Rooms are available for presentations

Each room is defined with these attributes:

- Room ID
- Floor number
- Seat capacity



To the right is our design for the ROOM table

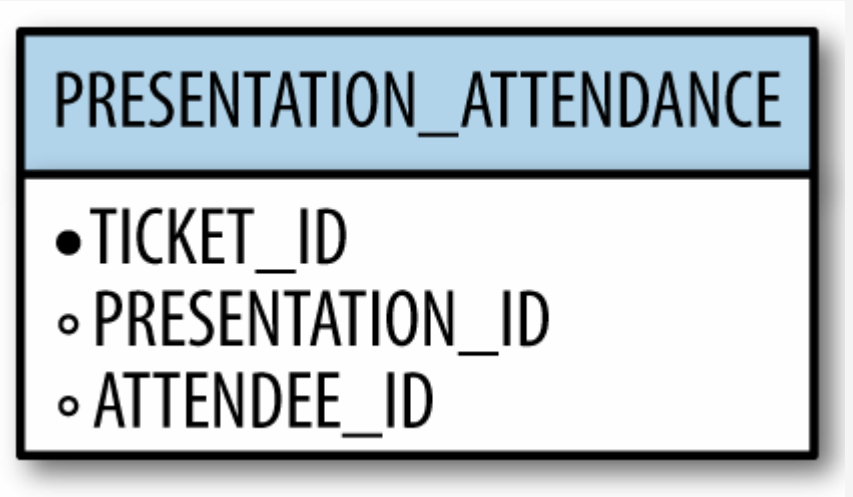
PRESENTATION_ATTENDANCE

When an ATTENDEE wants to attend a PRESENTATION, they can acquire a ticket with a ticket id

We can use these tickets to keep track of presentation attendance

Each presentation attendance is defined with these attributes:

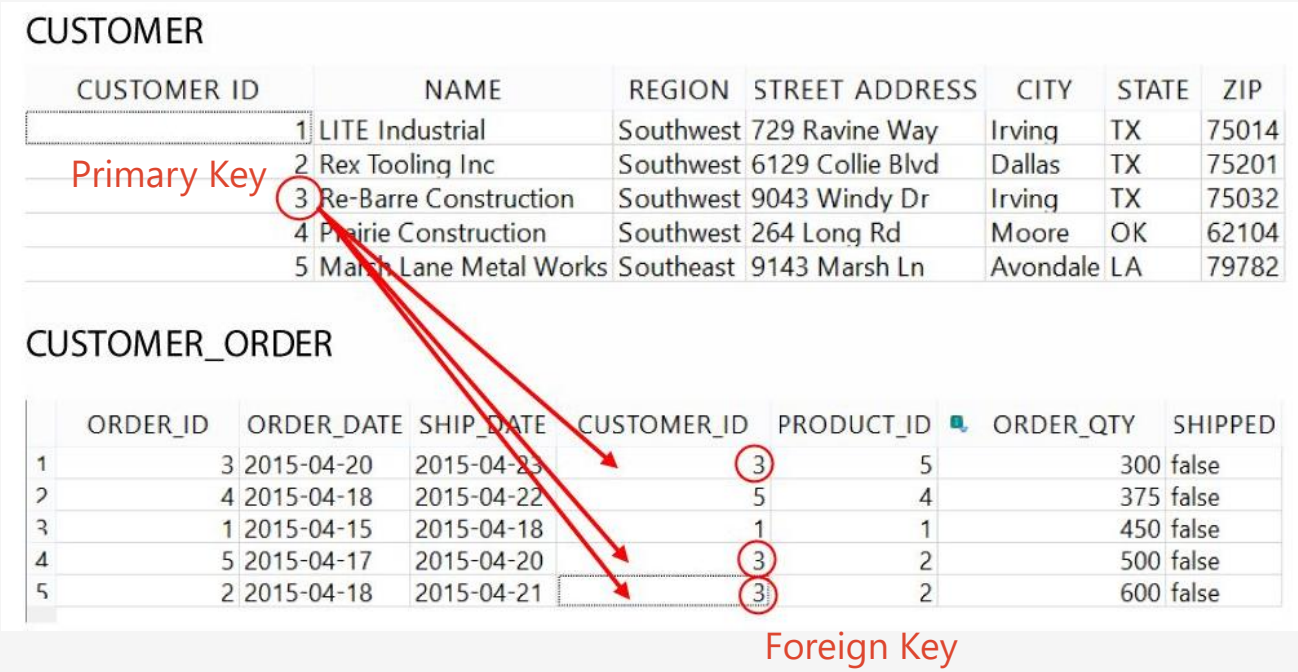
- Ticket ID
- Presentation ID
- Attendee ID



To the right is our design for the PRESENTATION_ATTENDANCE table

Revisiting Primary/Foreign Keys

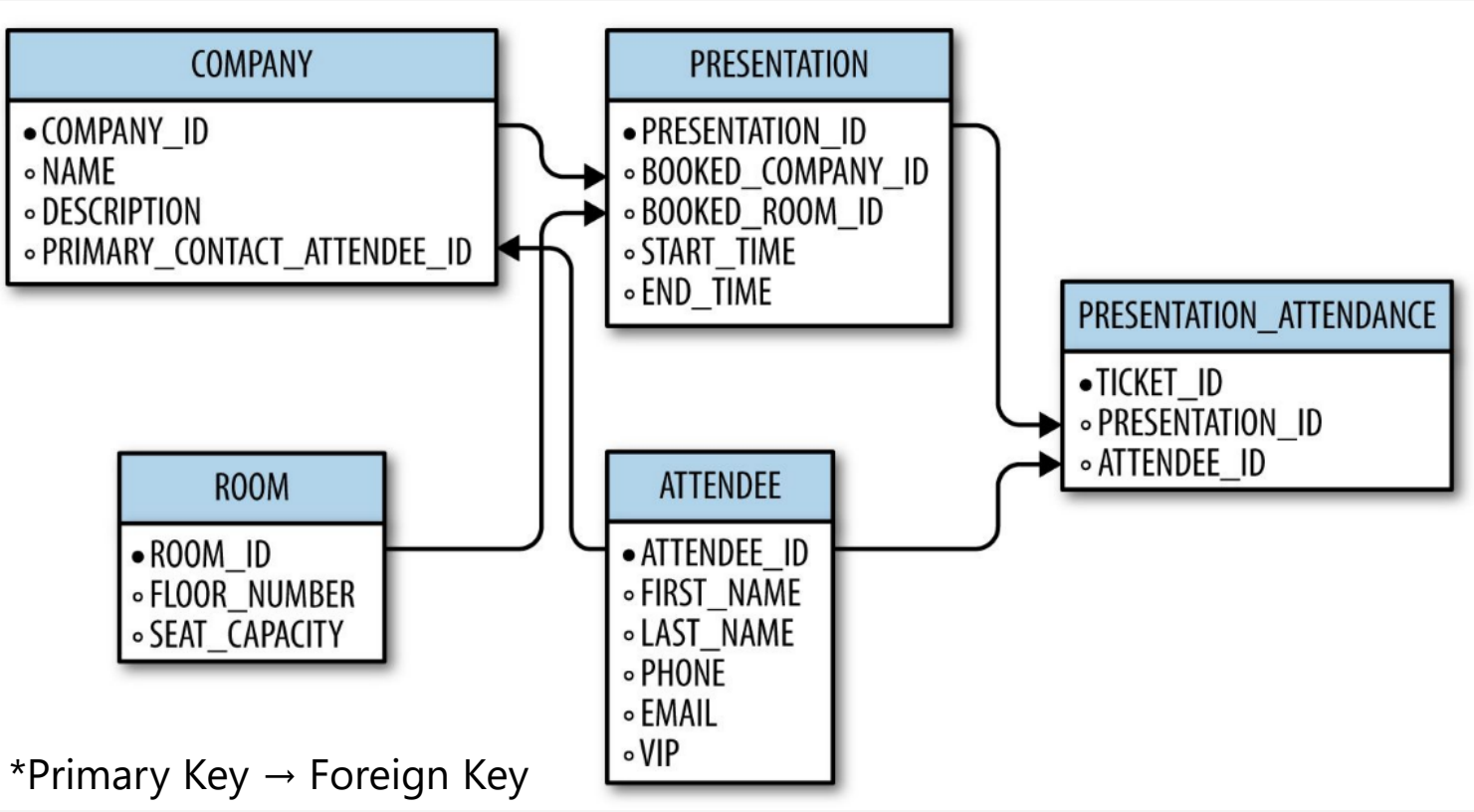
- With table relationships it is important to distinguish the primary key from the foreign key



- The field that *supplies* data to other tables is the **primary key**, and a field that receives data from another table is a **foreign key**.

The Database Schema

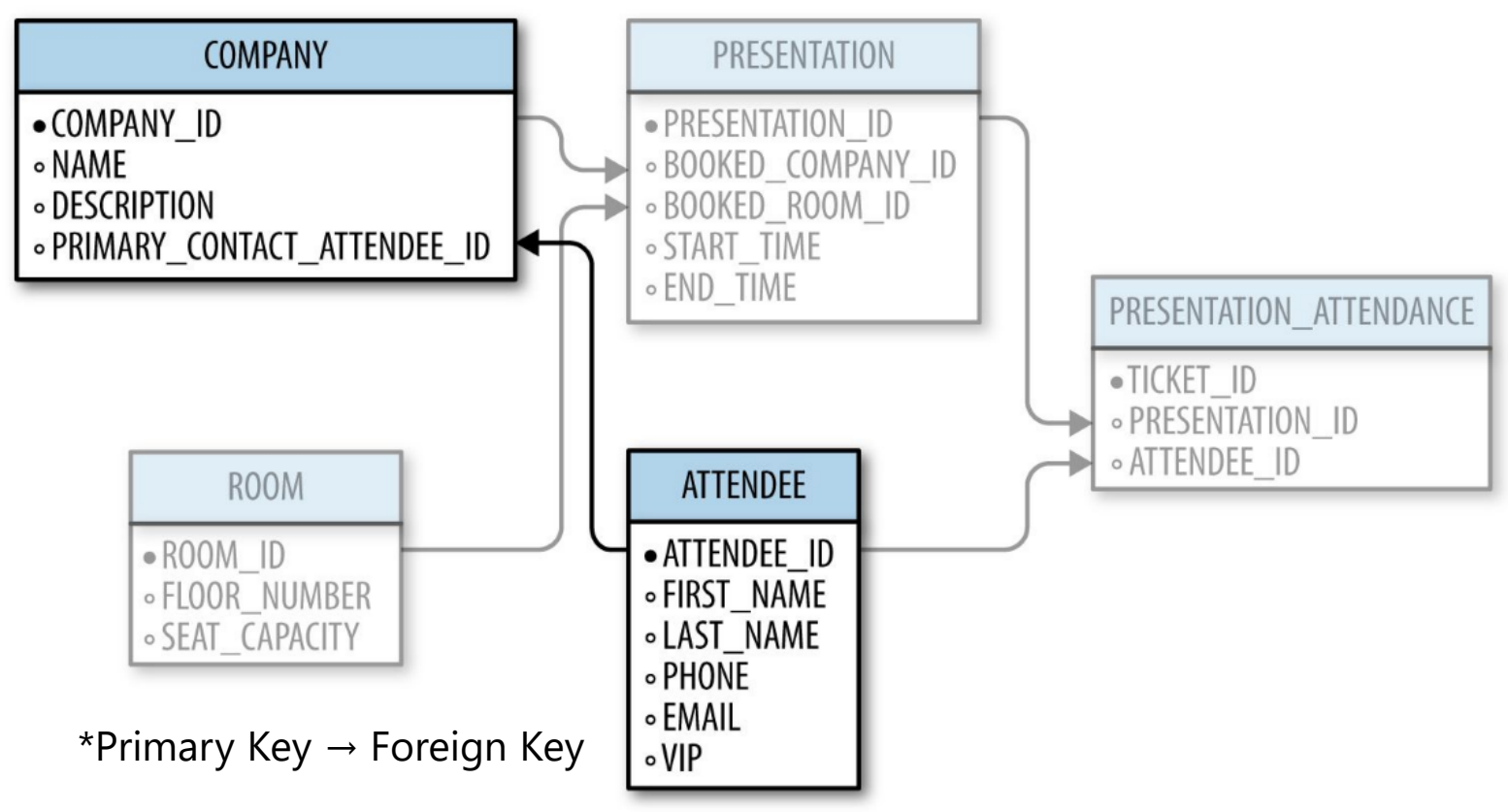
With our knowledge of primary and foreign keys, we can create a **database schema** of all tables and their relationships for the SurgeTech conference



*Primary Key → Foreign Key

The Database Schema

It can be overwhelming seeing all tables and their relationships at once, so the secret to reviewing a database schema is to focus on 2-3 tables at a time



Common Column Types

Type	Description
INTEGER	A simple, whole number
DOUBLE/DECIMAL	Supports non-whole, decimal numbers
BOOLEAN	A True/False value represented by 1 or 0
CHAR	A fixed number of text characters
VARCHAR	Any number of text characters, with an optional maximum
DATE	A calendar date value
TIME	A time value
DATETIME	A date and time value
TEXT	A longer piece of text (such as memos, articles, books, emails)

Common Column Modifiers

Modifier	Behavior
PRIMARY KEY	Makes the column a PRIMARY KEY
FOREIGN KEY	Makes the column a FOREIGN KEY
NOT NULL	Enforces that values can never be null in that column
DEFAULT	Allows you to specify a default value for a column rather than it default to NULL

Section IX

Writing Data

Exercise 9.1

Insert a new record into the COMPANY table for a company named “Pied Piper”, and provide a DESCRIPTION of “Compression platform for mobile and desktop” and a PRIMARY_CONTACT_ATTENDEE_ID of 1.

```
INSERT INTO COMPANY(NAME,DESCRIPTION, PRIMARY_CONTACT_ATTENDEE_ID)

VALUES ('Pied Piper','Compression platform for mobile and desktop', 1)
```

Exercise 9.2

Create a new ATTENDEE named Richard Hendricks, with an EMAIL of *richard.hendricks@piedpiper.com* and a VIP true value

```
INSERT INTO ATTENDEE (FIRST_NAME, LAST_NAME, EMAIL, VIP)

VALUES ('Richard', 'Hendricks', 'richard.hendricks@piedpiper.com',1)
```

Exercise 9.3

Make Richard Hendricks' ATTENDEE_ID the PRIMARY_CONTACT_ATTENDEE_ID for the COMPANY "Pied Piper"

```
UPDATE COMPANY SET PRIMARY_CONTACT_ATTENDEE_ID = 5

WHERE COMPANY_ID = 2
```

Section X

Going Forward

What Now?

- You now have the fundamentals of SQL in your tool belt
  - Get comfortable with consistent use and practice
  - If your job uses a specific database platform (e.g. MySQL, Oracle), apply this knowledge to learn that platform
  - Keep practicing with SQLite!
- There are SQL features you can advance into:
  - **Subqueries** – query off of other queries just like they were tables
  - **Indexing** – Configure large tables to perform better with SELECT operations
  - **Transactions** – Perform multiple update commands into a single, fail-safe batch
  - **Triggers** – Configure databases to react to UPDATE/DELETE/INSERT commands
  - **Database Administration** – Fine-tune production databases for large corporate environments
  - **Advanced Business Analysis** – Use advanced SQL features to perform deeper business analysis

What Now?

SQL Resources

- Getting Started with SQL (O'Reilly) by Thomas Nield
- Learning SQL (O'Reilly) by Alan Beaulieu
- Using SQLite (O'Reilly) by Jay A. Kreibich

It can be lucrative to combine SQL with another technical skill

- Python - versatile scripting language
- R – statistical scripting language and environment
- Java – Build full software solutions

Other Online Trainings by Thomas Nield

LIVE ONLINE TRAINING

Advanced SQL for Data Analysis (with Python, R, and Java)

Unleashing relational database analytics

LIVE ONLINE TRAINING

Reactive Python for Data Science: Production-Ready, Scalable Code for Real-Time Data

Learn the basics of reactive programming for more resilient, event-driven code models