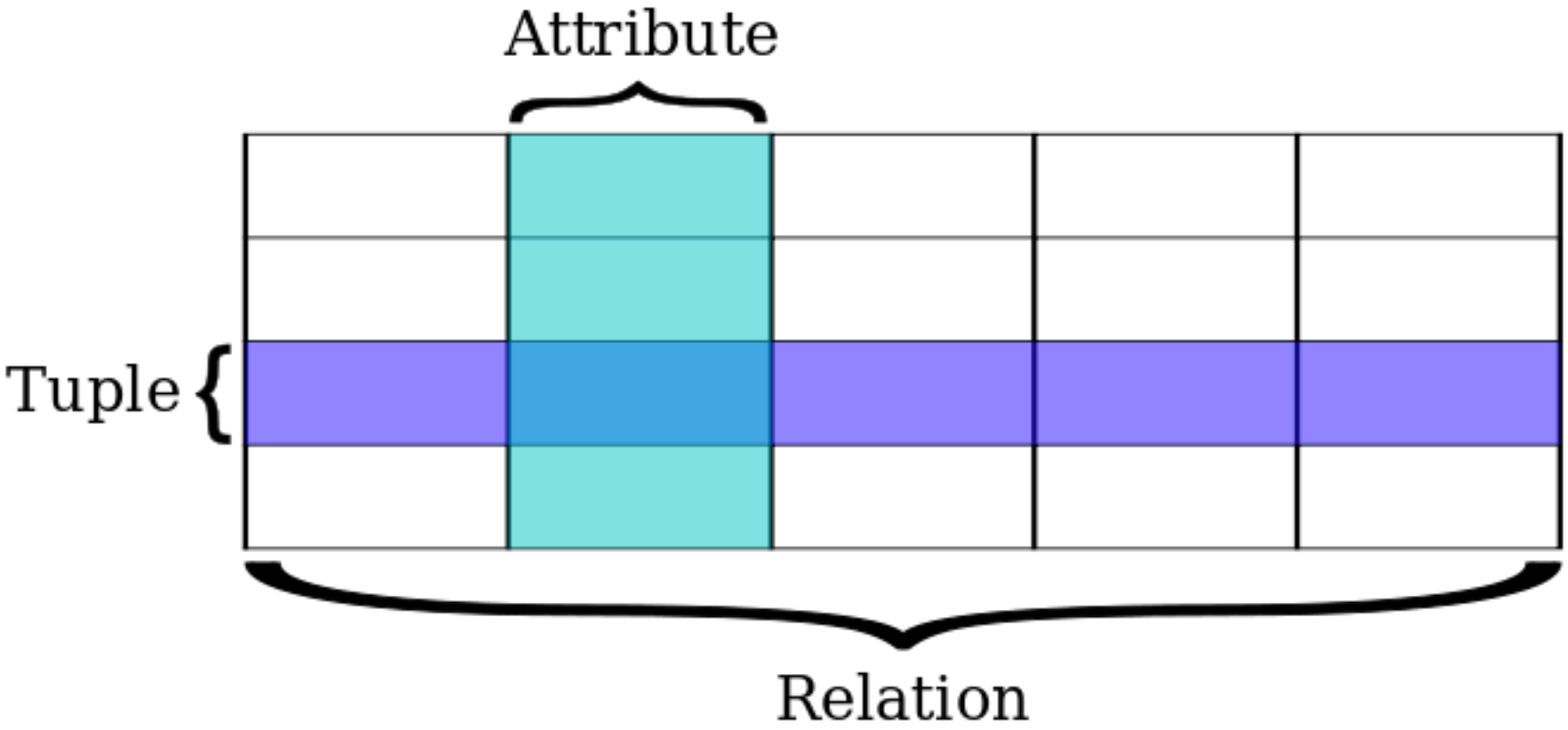
An overview of SQL

Dec 2017

Structured Query Language

- Pronounced “Sequel”
  - Originally called Sequel but changed for trademark reasons
- Dates to 1974
  - Written by IBM (Chamberlin and Boyce)
  - Based on “A Relational Model of Data for Large Shared Data Banks” by Edward Codd
  - First commercialised by Oracle
  - Standardised in 1986

Relational terminology (from Codd)



Relational Database

- Every row in a table has the same attributes (columns)
  - Relations are either tables or views on those tables
- A primary key for each row uniquely identifies it
- A foreign key points to another table's primary key

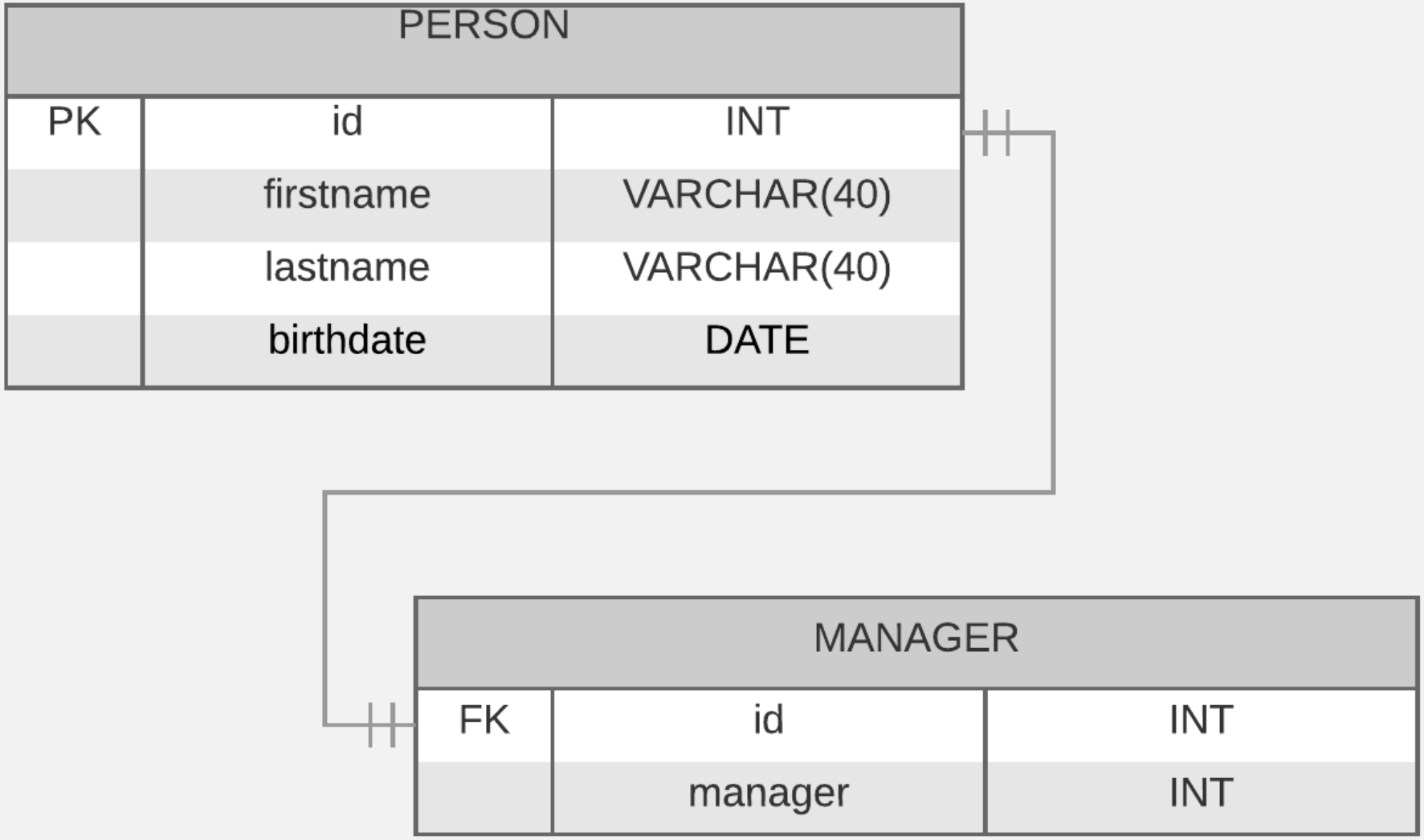
Relational database

Id	Firstname	Lastname	birthdate
2587	John	Hopkins	5/12/1973
7789	Henry	Gleeson	1/5/1985
22398	Eleanor	Richardson	10/6/1996

Why are we looking at SQL today?

- SQL and variations are widely used
  - Not just for relational databases
- Hive / SparkSQL
  - SQL over big data using map-reduce techniques
- Siddhi / KSQL / StreamingSQL
  - SQL queries over real-time streaming data
- Other SQL interfaces
  - e.g. SQL into Sloan Digital Sky Survey

Entity Relationship Diagram



SQL STATEMENTS

corresponding to the previous diagram

```
CREATE TABLE `PERSON` (
  `id` INT,
  `firstname` VARCHAR(40),
  `lastname` VARCHAR(40),
  `birthdate` DATE,
  PRIMARY KEY (`id`)
);

CREATE TABLE `MANAGER` (
  `id` INT,
  `manager` INT,
  KEY `FK` (`id`)
);
```

INSERT

```
INSERT INTO person
    (id, firstname, lastname, birthdate)
    values
    (564, "Henry", "Gleeson", "1968-12-5");
INSERT INTO person
    (id, firstname, lastname, birthdate)
    values
    (2343, "Eleanor", "Smith", "1995-1-9");
```

SELECT

SELECT * FROM person;

id	firstname	lastname	birthdate
564	Henry	Gleeson	1968-12-5
2343	Eleanor	Smith	1995-1-9

SELECT

SELECT * FROM person WHERE id = 564;

id	firstname	lastname	birthdate
564	Henry	Gleeson	1968-12-5

SELECT

SELECT * FROM person WHERE firstname = ”Eleanor";

id	firstname	lastname	birthdate
2343	Eleanor	Smith	1995-1-9

SELECT

```
SELECT firstname, lastname FROM person
    ORDER BY firstname;
```

```
firstname   lastname
----------  ----------
Eleanor     Smith
Henry       Gleeson
```

SELECT

```
SELECT firstname, lastname FROM person
   ORDER BY lastname LIMIT 1;
```

```
firstname   lastname
----------  ----------
Henry       Gleeson
```

SELECT

```
SELECT AVG(birthdate) FROM person;

AVG(birthdate)
--------------
1981.5
```

Functions

- MIN
- MAX
- AVG
- COUNT
- SUM

OTHER COMMANDS

- DELETE

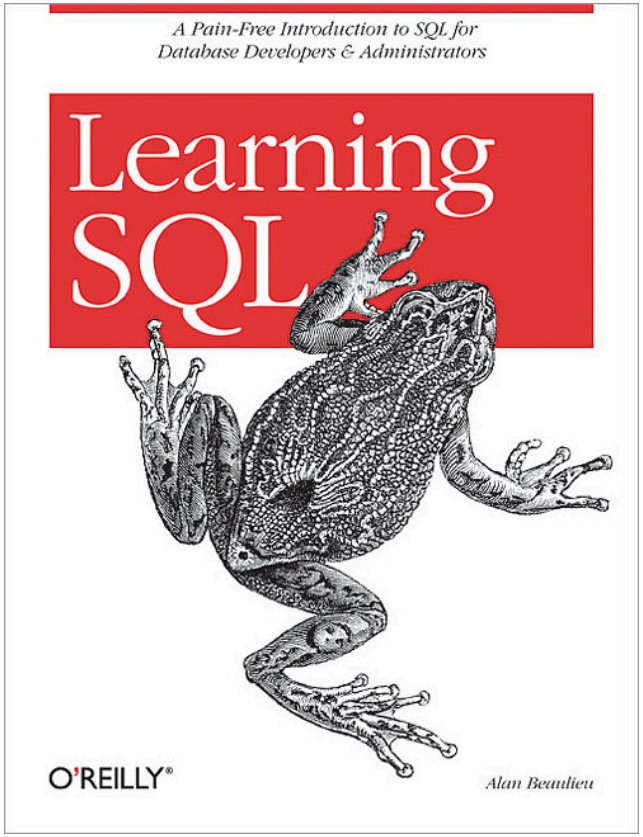
```
DELETE FROM person WHERE ID=564;
```

- UPDATE

```
UPDATE PERSON
   SET firstname = Henrietta WHERE ID=564;
```

This is a very brief introduction!

- We will learn more from the exercises
- There are lots of resources on the Web



Apache Hive

http://hive.apache.org

- Just like SQL except it generates Map Reduce jobs
- Works on Hadoop and Spark
  - Embedded into Spark as SparkSQL
- Includes DDL (Data Definition Language) as well as SQL
- Makes many processing tasks very simple

Hive example

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
                page_url STRING, referrer_url STRING,
                ip STRING COMMENT 'IP Address of the User')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
STORED AS SEQUENCEFILE;


LOAD DATA LOCAL INPATH /tmp/pv_2008-06-08_us.txt INTO TABLE page_view
PARTITION(date='2008-06-08', country='US')

INSERT OVERWRITE TABLE xyz_com_page_views
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01' AND page_views.date <=
'2008-03-31' AND
      page_views.referrer_url like '%xyz.com';
```

SparkSQL

- Integrates into existing Spark programs
  - Mixes SQL with Python, Scala or Java
- Integrates data from CSV, Avro, Parquet, JDBC, ODBC, JSON, etc
  - Including joins across them
- Fully supports Apache Hive
  - *If you build it with Hive support*
- Fits into the resilient scalable model of Spark

Spark SQL example

```
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)

lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))

schemaPeople = sqlContext.createDataFrame(people)
schemaPeople.registerTempTable("people")

teenagers = sqlContext.sql("SELECT name FROM people WHERE age >= 13
AND age <= 19")

teenNames = teenagers.map(lambda p: "Name: " + p.name)
for teenName in teenNames.collect():
  print(teenName)
```

DataFrame

Based on Python and R dataframes

- Column based object used by SQL
- Offers SQL like programming
- Supports algebraic optimisation and code gen
- E.g. in Scala:

```
means = users.where(users["age"] > 20)
              .groupBy("city")
              .avg("income")
```

And they run up to 2-5x faster than equivalent computations expressed via the functional API.

More SQL

```
df.
  select('postcode','id').
  withColumn('first_pc',
    split(df.postcode, '\s'[0]).
    where((col("first_pc") == 'SW11') or
                (col("first_pc") == 'OX1')).
     groupBy('first_pc').
     agg({"id": "count"}).show()
```

User Defined Functions

- In SQL a User Defined Function is an extension that helps

Questions?