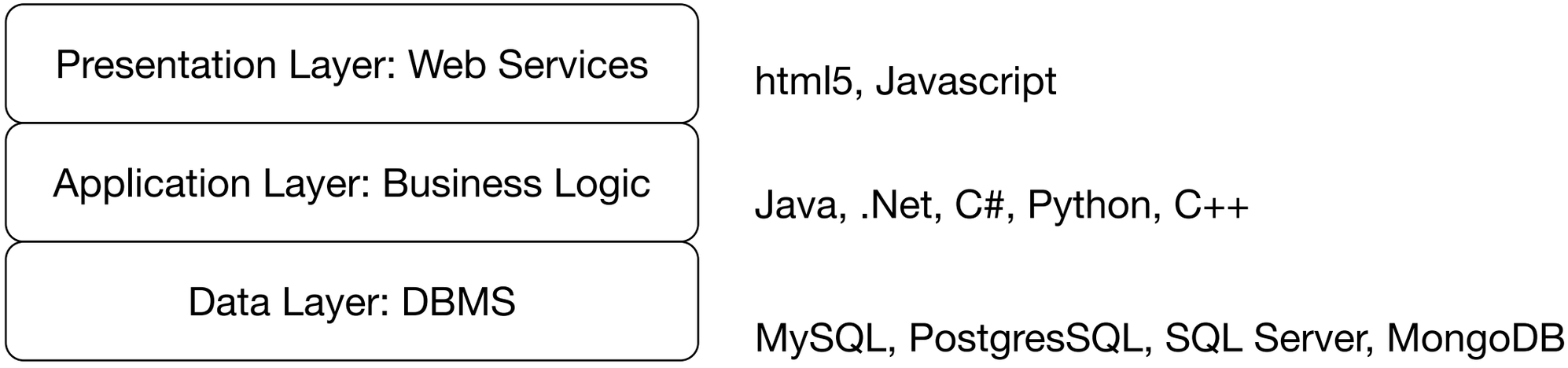
Database System Interactions

Thomas Schwarz, SJ

Three Tier Web Server Architecture

- Standard architecture for e-commerce sites
  - Tiered / layered architecture around since the THE operating system 1965



Three Tier Web Server Architecture

- Web Services Layer:
  - User interact with the site using a web browser
  - Forms, scripts, …
  - Requests are being routed to the application layer
    - Simple example: Embed PHP scripts into a web server
      - Download: LAMP / WAMP / XAMPP etc. With Apache, MySQL, PHP, Perl, …
      - Embed PHP script in HTML: <?php … ?>

Three Tier Web Server Architecture

- Application Tier
  - Simple system: Bypass application tier by directly translating web requests to database requests
  - Normally:
    - Integrate different databases
    - Implement business logic

Three Tier Web Server Architecture

- Database Tier:
  - Executes queries (including updates and inserts)

Integrating SQL with Application Layer

- Application layer uses languages like PHP, Python, Java, …
- Needs to interact with an application programming environment

SQL Environment

- SQL environment
  - Schemas: Tables, views, assertions, triggers, stored procedures, character sets, grant statements (for rights) maintained by a catalog
  - Servers / Clients
    - Clients need to connect to a server
    - Client/server connection is divided into Sessions
      - Each session selects a catalog and a schema

Integrating SQL with Application Layer

- Impedance mismatch problem
  - All languages / environment are Turing complete
  - Standard SQL is not:
    - Not everything that a computer can do can be done with SQL
      - E.g. cannot compute factorial with SQL
- Need to use both SQL (to interact with database) AND application level program

Integrating SQL with Application Layer

- Program sets up a connection to a database and closes it at the end
  - which might be automatic

Integrating SQL with Application Layer

- Central idea is the 'cursor'
  - Basically a pointer into the result table of an SQL query
  - Usually:
    - Can get result table row by row
    - Can get result table all at once
      - Could be hard on memory resources
    - Can get result table in tranches

Integrating Python with MySQL

- Solutions differ widely according to application tier environment and
  - Here: look at how to connect Python with MySQL
    - There are a variety of Python packages that will do that
      - I chose SQL-connector

Python 3 SQL connector

- Needed: Python 3
- Install MySQL Connector
    - Install with pip
    - Be careful for which Python you install
        - E.g. Mac has a Python 2.7 installed as part of the OS
        - `pip3.13 install mysql.connector`
- You will need to know your MySQL password
    - If necessary, just re-install MySQL

Python 3 MySQL Connector

- You can use
  - https://www.mysqltutorial.org/python-mysql/

Installing a database

- https://www.mysqltutorial.org/python-mysql/python-connecting-mysql-databases/
- Get the pub database and run it:
  - Download and move zip-file into a project directory
  - In the directory, invoke mysql

```
%mysql -u root -p
```

  - Create the database

```
mysql> source pub.sql
```

Setting up a connection

```
import mysql.connector as mc

PASSWORD = '1ignatius'

mydb = mysql.connector.connect(
    host = 'localhost',
    user = 'root',
    password = PASSWORD
    )
```

Using configparser

- Python module to read configuration files
  - Similar to .ini files in windows
  - Allows many scripts to use the same configuration
  - Configuration files are broken into section

Using configparser

- Create a file app.ini

```
[mysql]
host = localhost
port = 3306
database = pub
user = root
password = 1ignatius
```

Using configparser

```
def read_config(filename='app.ini',
section='mysql'):
    config = ConfigParser()
    config.read(filename)
    data = {}
    if config.has_section(section):
        items = config.items(section)
        for entry, value in items:
            data[entry] = value
    else:
        raise Exception(f'{section} section not
found')
    return data
```

Cursors

- After establishing a connection to the database, you use a ***cursor***
  - Cursors are also used in stored procedures
  - Cursors allow to pass through the result from a select row by row

```
cursor = conn.cursor()
```

Cursors

- We use the cursor to place queries and also to retrieve rows

```
cursor.execute("""SELECT first_name, last_name
                  FROM authors""")
```

- There are several ways to get the results
  - Using tuple assignment

```
for (fn, ln) in cursor:
        print(fn, ln)
```

Cursors

- Number of tuples:
  - Use cursor.rowcount

```
print('Total Row(s):', cursor.rowcount)
```

- `fetchall` fetches all (remaining) rows of a query result and returns a list of tuples
  -

```
rows = cursor.fetchall()
    for row in rows:
        print(row)
```

Cursors

- `fetchone` gets the next result

```
for _ in range(5):
        row = cursor.fetchone()
        print(row)
print(5*'\n')
rows = cursor.fetchall()
for row in rows:
    print(row)
```

Cursors

```
while row is not None:
    print(row)
    row = cursor.fetchone()
```

Cursors

- `fetchmany` gets a number of tuples

```
for _ in range(10):
rows = cursor.fetchmany(5)
for row in rows:
    print(row)
print(2*'\n')
```

Cursors

- There a variety of cursors
  - This cursor returns the results as a dictionary

```
cursor = conn.cursor(dictionary = True)
```

SQL for Data Analysis

Thomas Schwarz, SJ

2025

Data Cleaning and Data Wrangling

- Difficult to understand data
  - **Data dictionary**:
    - Repository describing:
      - fields
      - possible values
      - data collection modes
      - relation to other data

Data Cleaning and Data Wrangling

- Structured versus unstructured data
  - Unstructured data does not fit easily into databases
  - Often stored outside
  - Difficult to query
- Quantitative versus qualitative data
- First-, Second-, Third-Party Data
  - collected by organization itself, collected from vendors and other service providers, or collected by government or free sources

Data Cleaning and Data Wrangling

- Exploratory Data Analysis (EDA)
  - Collect summaries and visualizations
  - Fit distributions to numerical data
    - Use histograms and frequencies

Data Cleaning and Data Wrangling

- Example: Distribution of the number of payments in sakila
  - We get the number of payments a client has made

```
SELECT customer_id, count(payment_id) as payments
FROM payment
GROUP BY customer_id;
```

customer_id	payments
1	32
2	27
3	26
4	22
5	38
6	28
7	33

Data Cleaning and Data Wrangling

- However: we want to know the number of clients that made *x* payments
  - Use the previous result as a subquery

```
SELECT payments, count(*) as num_customers
FROM
(
    SELECT customer_id, count(payment_id) as payments
     FROM payment
     GROUP BY customer_id
) temp
GROUP BY payments
ORDER BY payments ASC;
```

Data Cleaning and Data Wrangling

- Explanation

```
SELECT payments, count(*) as num_customers
FROM
(
    SELECT customer_id, count(payment_id) as payments
     FROM payment
     GROUP BY customer_id
) temp
GROUP BY payments
ORDER BY payments ASC;
```

customer_id	payments
1	32
2	27
3	26
4	22
5	38
6	28
7	33

Data Cleaning and Data Wrangling

- Explanation:

```
SELECT payments, count(*) as num_customers
FROM
(
    SELECT customer_id, count(payment_id) as payments
     FROM payment
     GROUP BY customer_id
) temp
GROUP BY payments
ORDER BY payments ASC;
```

Outer query counts the number of times a certain number of payments has been made

payments	num_customers
32	35
27	41
26	53
22	35
38	4
28	41
33	19

Data Cleaning and Data Wrangling

- Explanation:

```
SELECT payments, count(*) as num_customers
FROM
(
    SELECT customer id, count(payment id) as payments
    FROM payment
    GROUP BY customer_id
) temp
GROUP BY payments
ORDER BY payments ASC;
```

And then we order

payments	num_customers
22	35
23	45
24	36
25	50
26	53
27	41
28	41

Data Cleaning and Data Wrangling

- Binning:
- Can use case statement
  - Create bins using a case statement

```
SELECT
CASE
WHEN payments < 10 THEN ',10'
WHEN 10 <= payments AND payments <20 THEN '10, 20'
WHEN 20 <= payments AND payments <30 THEN '20, 30'
WHEN 30 <= payments AND payments <40 THEN '30, 40'
WHEN 40 <= payments THEN '40, '
END AS bin,
num_customers FROM
   (SELECT
       payments, COUNT(*) AS num_customers
    FROM
      (SELECT
         customer_id, COUNT(payment_id) AS payments
      FROM
         payment
      GROUP BY customer_id) temp
GROUP BY payments);
```

```
SELECT
    CASE
        WHEN payments < 10 THEN ',10'
        WHEN 10 <= payments AND payments < 20 THEN '10, 20'
        WHEN 20 <= payments AND payments < 30 THEN '20, 30'
        WHEN 30 <= payments AND payments < 40 THEN '30, 40'
        WHEN 40 <= payments THEN '40, '
    END AS bin,
    SUM(num_customers)
FROM
    (SELECT
        payments, COUNT(*) AS num_customers
    FROM
        (SELECT
            customer_id, COUNT(payment_id) AS payments
         FROM
            payment
         GROUP BY customer_id) temp
    GROUP BY payments) temp2
GROUP BY bin
;
```

Data Cleaning and Data Wrangling

- Deduplication
  - Duplicates are normal
    - Find customers who made a payment last month (Feb. 2006)

```
SELECT customer_id
FROM payment
WHERE Month(last_update) = 2 AND YEAR(last_update) = 2006;
```

    - Send a coupon to these customers

Data Cleaning and Data Wrangling

- Deduplicate by using distinct

```
SELECT DISTINCT customer_id
FROM payment
WHERE Month(last_update) = 2 AND YEAR(last_update) = 2006;
```

Data Cleaning and Data Wrangling

- Deduplication with "Group By"

```
SELECT customer_id, first_name, last_name
FROM payment JOIN customer USING(customer_id)
WHERE Month(payment.last_update) = 2
      AND YEAR(payment.last_update) = 2006
GROUP BY customer_id, first_name, last_name;
```

Sampling

- Can use `RAND( )` to make a random selection

```
CREATE PROCEDURE get_sample()
BEGIN
    DROP TABLE  IF EXISTS sample;
    CREATE TEMPORARY TABLE sample
    SELECT
       title
    FROM
       books
    WHERE RAND() < 0.1;
END$$

DELIMITER ;

CALL get_sample();

SELECT * FROM sample;
```

To sample several times, we need to drop the temp table first

Sampling

- Can use `RAND( )` to make a random selection

```
CREATE PROCEDURE get_sample()
BEGIN
   DROP TABLE  IF EXISTS sample;
   CREATE TEMPORARY TABLE sample
   SELECT
      title
   FROM
      books
   WHERE RAND() < 0.1;
END$$

DELIMITER ;

CALL get_sample();

SELECT * FROM sample;
```

Creating the sample table

Sampling

- Can use `RAND( )` to make a random selection

```
CREATE PROCEDURE get_sample()
BEGIN
   DROP TABLE  IF EXISTS sample;
   CREATE TEMPORARY TABLE sample
   SELECT
      title
   FROM
      books
   WHERE RAND() < 0.1;
END$$

DELIMITER ;

CALL get_sample();
SELECT * FROM sample;
```

Selects a record with 10% probability

Sampling

- Can use `RAND( )` to make a random selection

```
CREATE PROCEDURE get_sample()
BEGIN
   DROP TABLE  IF EXISTS sample;
   CREATE TEMPORARY TABLE sample
   SELECT
      title
   FROM
      books
   WHERE RAND() < 0.1;
END$$

DELIMITER ;

CALL get_sample();
SELECT * FROM sample;
```

Repeat this several times to see that we get independent samples

```
DELIMITER $$
CREATE PROCEDURE get_sample(OUT sample_size INT)
BEGIN
   DROP TABLE  IF EXISTS sample;
   CREATE TEMPORARY TABLE sample
   SELECT
      title
   FROM
      books
   WHERE RAND() < 0.1;

    SELECT COUNT(*) INTO sample_size
    FROM sample;

END$$

DELIMITER
```