Intro to SQL

Data Engineering

Why Learning Structured Query Language?

- The structured query language is the **standard query language used by most database management systems**
- Definitely required to be known by **anybody** who is going to be **using a database management system**

What is a Relational Database?

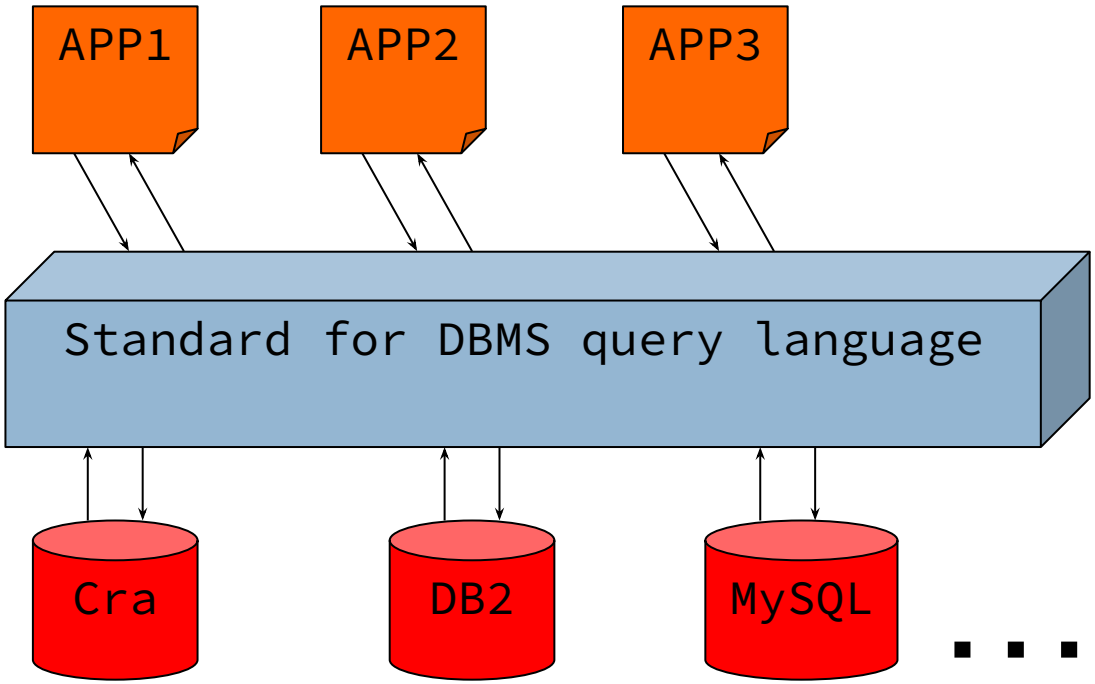
- A relational database stores data in tables, where each row is a record and each column is a field.
- Tables can be linked together using keys, allowing us to model relationships—like employees belonging to departments.
- Common relational databases include PostgreSQL, MySQL, Oracle, and Microsoft SQL Server.
- They use the Structured Query Language (SQL) for querying and managing data.

Why Relational Databases in Data Engineering?

- Relational databases are optimized for handling large datasets efficiently and safely.
- They’re the backbone for data pipelines, analytics, reporting, and business logic.
- They enforce data consistency and allow powerful querying with SQL.
- Nearly every company’s data stack involves SQL at some stage.

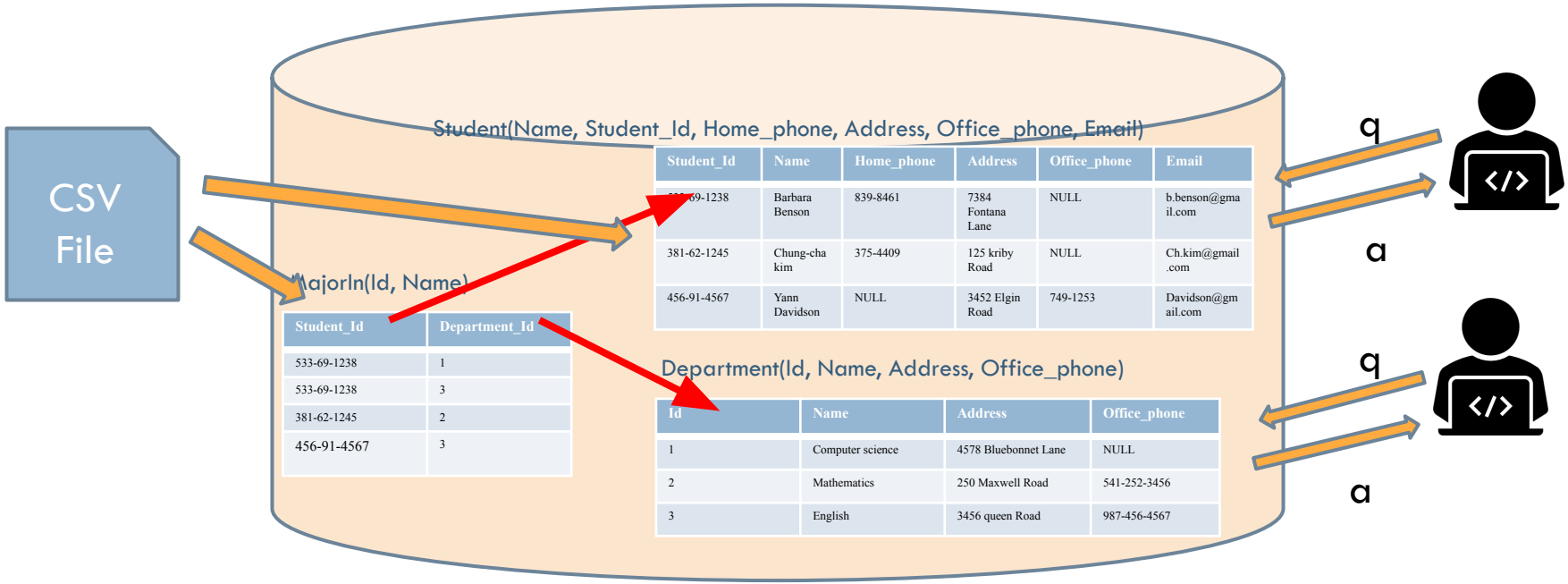
- **Question:**
  - What is meant by standard for query language(SQL)?
  - Why do we need a standard for query language?
- **Answer:**
  - SQL is **standard for relational databases** that all databases or **database users use in some form or the other.**
    - Exp: Microsoft sequel server, MySQL, Postgres
  - Large number of database management systems are available(commercially available, freely available , implemented in academic institution, some of them implemented in companies, industries and so on) with SQL
  - **All the application programs need to know SQL.** They **all speak SQL** and **use SQL**. we should be able to connect or use any database management system

Why Standard Query Language?



Steps in Creating and Using Database

1. Design schema, create using ddl
2. Once it is ready the next step is loading the initial data
3. Execute queries and modification



Querying Database

Result table

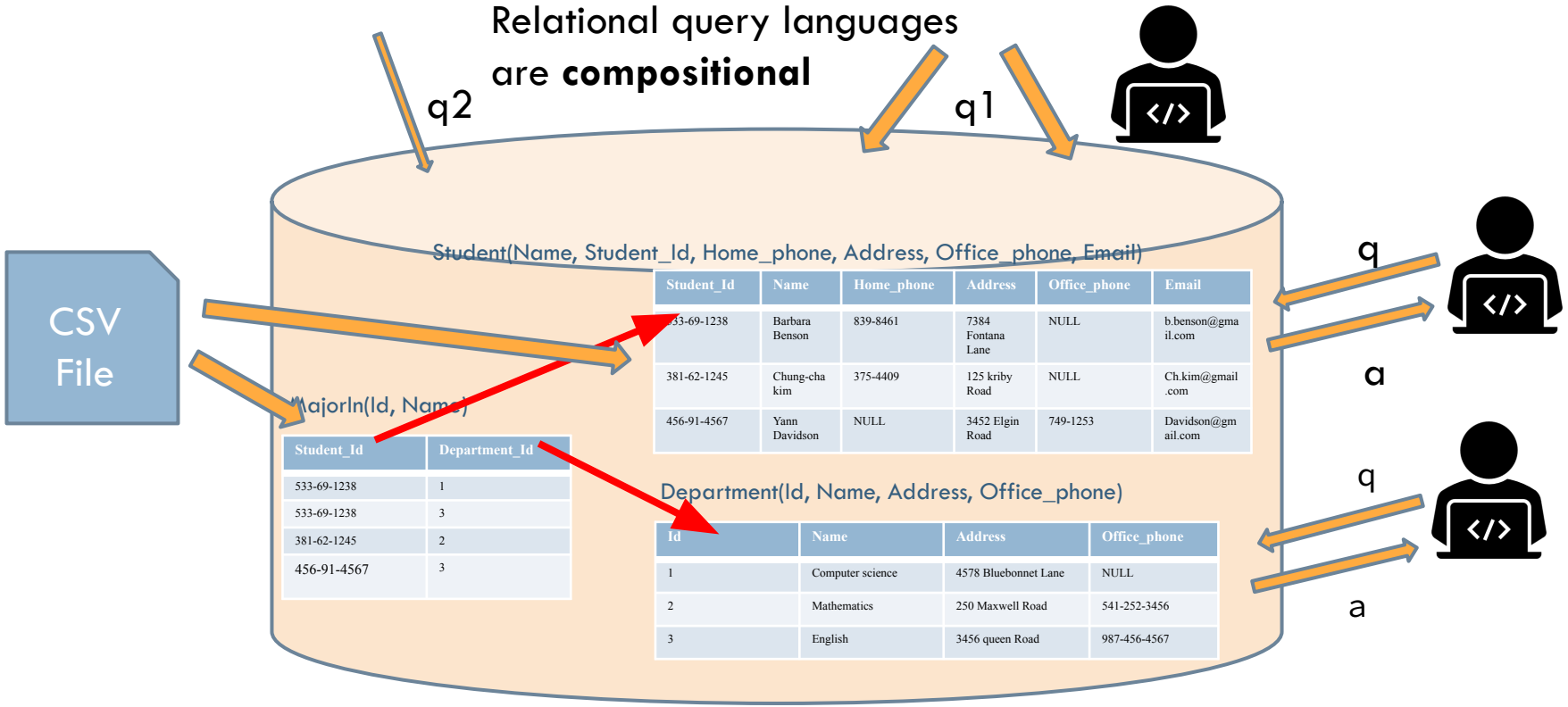
Student_Id	Name
533-69-1238	Barbara Benson
456-91-4567	Yann Davidson

Write query

List of student whose majoring in English

```
SELECT Student_Id, Student.Name
From Student, MajorIn, Department
Where Department.Name = 'English'
```

Relational query languages are **compositional**



Student_Id	Name	Home_phone	Address	Office_phone	Email
533-69-1238	Barbara Benson	839-8461	7384 Fontana Lane	NULL	b.benson@gmail.com
381-62-1245	Chung-cha kim	375-4409	125 kriby Road	NULL	Ch.kim@gmail.com
456-91-4567	Yann Davidson	NULL	3452 Elgin Road	749-1253	Davidson@gmail.com

Student_Id	Department_Id
533-69-1238	1
533-69-1238	3
381-62-1245	2
456-91-4567	3

Id	Name	Address	Office_phone
1	Computer science	4578 Bluebonnet Lane	NULL
2	Mathematics	250 Maxwell Road	541-252-3456
3	English	3456 queen Road	987-456-4567

SQL Syntax

```
SELECT column1, column2, ...
FROM table_name
[WHERE condition]
[GROUP BY column_list]
[HAVING group_condition]
[ORDER BY column [ASC|DESC]]
[LIMIT number]
```

Company

Employee

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
0	198	Donald	OConnell	DOCONNEL	650.507.9833	21-Jun-07	SH_CLERK	2600	-	124.0	50
1	199	Douglas	Grant	DGRANT	650.507.9844	13-Jan-08	SH_CLERK	2600	-	124.0	50
2	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-Sep-03	AD_ASST	4400	-	101.0	10
3	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-Feb-04	MK_MAN	13000	-	100.0	20
4	202	Pat	Fay	PFAY	603.123.6666	17-Aug-05	MK_REP	6000	-	201.0	20

Department

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
0	10	Administration	200	1700
1	20	Marketing	201	1800
2	30	Purchasing	114	1700
3	40	Human Resources	203	2400
4	50	Shipping	121	1500

Selecting Columns and Filtering Rows

Select names and salary of employees older than 30

Pandas:

```
employees[['name', 'salary']][employees['age'] > 30]
```

SQL:

```
SELECT name, salary
FROM employees
WHERE age > 30;
```

Sorting the results

Pandas:

```
employees.sort_values('salary', ascending=False)
```

SQL:

```
SELECT *
FROM employees
ORDER BY salary DESC;
```

Aggregation and GROUP BY

Pandas:

```
employee.groupby('deptartment_id')['salary'].sum()
```

SQL:

```
SELECT department_id, SUM(salary) as total_salary
FROM employee
GROUP BY department_id;
```

Filtering After Aggregation (HAVING)

Pandas:

```
total_salary =
employee.groupby('department_id')['salary'].sum()
total_salary[total_salary['salary'] > 150000]
```

SQL:

```
SELECT department_id, SUM(salary) as total_salary
FROM employee
GROUP BY department_id
HAVING SUM(salary) > 150000;
```

Multiple Aggregations

Pandas:

```
employee.groupby('department_id').agg({'salary': ['mean',
'max', 'min']})
```

SQL:

```
SELECT department_id, AVG(salary) AS avg_salary, MAX(salary)
AS max_salary, MIN(salary) AS min_salary
FROM employees
GROUP BY department_id;
```

Joining Tables

Pandas:

```
pd.merge(employee, department, on='dept_id', how='inner')
```

SQL:

```
SELECT e.*, d.department_name
FROM employee e
JOIN department d
ON e.department_id = d.department_id;
```

Filtering on Joined Data

Pandas:

```
merged = pd.merge(employee, department, on='department_id')
merged[(merged['department_name'] == 'Engineering') &
(merged['salary'] > 80000)]
```

SQL:

```
SELECT e.first_name, e.last_name e.salary, d.department_name
FROM employee e
JOIN department d ON e.department_id = d.department_id
WHERE d.department_name = 'Engineering' AND e.salary > 80000;
```

Calculated Columns

Pandas:

```
employee['annual_bonus'] = employees['salary'] * 0.10
```

SQL:

```
SELECT first_name, last_name,
    salary, salary * 0.10 AS annual_bonus
FROM employee;
```

Nested Queries (Subqueries)

Pandas:

```
avg_salary = employee['salary'].mean()
employee[employee['salary'] > avg_salary]
```

SQL:

```
SELECT *
FROM employee
WHERE salary > (
  SELECT AVG(salary) FROM employee
);
```

IN, NOT IN, and EXISTS

Pandas:

```
exclude_depts = department[(department['department_name'] == 'HR') |
(department['department_name'] == 'Marketing')
]['department_id']
employee[~employee['department_id'].isin(exclude_depts)]
```

SQL:

```
SELECT *
FROM employees
WHERE department_id NOT IN (SELECT department_id
  FROM department
  WHERE department_name = 'HR' OR department_name = 'Marketing');
```

Null Values and Missing Data

Pandas:

```
employee[employee['salary'].isnull()]
employee[employee['salary'].notnull()]
```

SQL:

```
SELECT * FROM employees WHERE salary IS NULL;
SELECT * FROM employees WHERE salary IS NOT NULL;
```