CSCI 334: Principles of Programming Languages

Lecture 14-2: SQL

Instructor: Dan Barowy

Williams

Outline

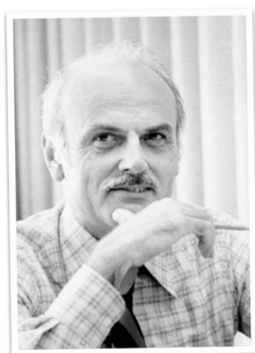
SQL History

Relational Algebra

SQL Language

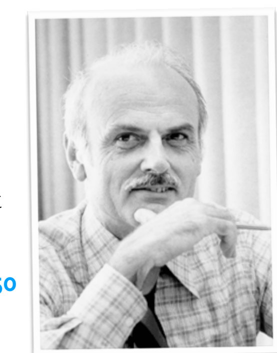
SQL

- SQL, or "structured query language," is a **DSL** for querying data, invented by E. F. Codd in 1970.
- **Limits** itself to certain kinds of queries.
- All valid queries can be answered **efficiently** (and they terminate).
- Based on a theory of data queries called the **relational algebra**.



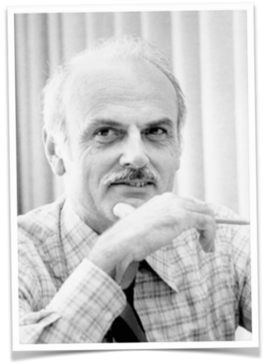
Importance of SQL

- IBM (Codd's employer) **never capitalized** on its invention.
- But Larry Ellison (**Oracle**) did, and as a result, became one of the richest people on earth.
- E.F., Codd won a **Turing Award** for his work on the relational algebra and relational database management systems.
- As of 2017, relational database systems alone were a **$50 billion market**.
- RDBMSs are a major area of **CS research**.
- SQL is one of the most **important** and **successful** languages ever invented.



Failures of SQL

- One of Codd's goals was to enable **non-programmers** to perform data querying tasks.
  ("Seven Steps to Rendezvous with the Casual User." E.F. Codd. IBM Research Labratory report RJ 1333 (#20842). 1974.")
- This goal was **not achieved**. Writing SQL is still considered a specialized task suited for programmers.



Relational Algebra

The relational algebra is a **calculus** defined over **set theory**.

Relational Algebra: Data

A **relation** is a **set** of **tuples**.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

- Recall: **sets** contain only **unique** elements.
- Also, the **order** of elements in a set does not matter.

Relational Algebra: Data

A **relation** is a **set** of **tuples**.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

- The members of a tuple are called **attributes**.
- The **order** of attributes in a tuple does not matter.

Relational Algebra: Data

A **relation** is a **set** of **tuples**.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

- An **instance of a relation** is a **table**.

Relational Algebra: Data

A **database** is a **set** of **tables (relation instances)**.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Relational Algebra: Data

A **schema** describes a database **independently of instances**.

A **relation** is described by its **attributes**.

```
Employee(Name, EmpId, DeptName)
Dept(DeptName, Manager)
```

schema : **instances** :: **class** : **object**

Relational Algebra

database

Employee (**instance**)

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept (**instance**)

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

schema

```
Employee(Name, EmpId, DeptName)
Dept(DeptName, Manager)
```

relation

relation

To be clear:

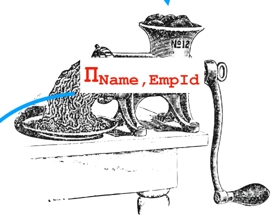
A **database** contains **instances** of **relations** described by a **schema**.

Relational Algebra: Operations

The relational algebra also defines **operations** over **relations**.

Such operations yield **new relations**.

Employee(Name, EmpId, DeptName)



$\Pi_{Name,EmpId}$(Employee)

Operations: Projection

Projection selects a subset of **attributes** $a_1, \ldots, a_n$ in a tuple.

$$\Pi_{a_1,\ldots,a_n}(\mathbf{R})$$

$$\downarrow$$

$$(a_1, \ldots, a_n)$$

Operations: Projection

Projection selects a subset of **attributes** $a_1, \ldots, a_n$ in a tuple.

Employee(Name, EmpId, DeptName)

$\Pi_{Name,EmpId}$(**Employee**) → (Name, DeptName)

example

$\Pi_{Name,EmpId}$ (*Employee*) →

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Name	EmpId
Harry	3415
Sally	2241
George	3401
Harriet	2202

Operations: Projection

Projection selects a subset of **attributes** $a_1, \ldots, a_n$ in a tuple.

Dept(DeptName, Manager)

$\Pi_{Manager}$(**Dept**) → (Manager)

example

$\Pi_{Manager}$ (*Dept*) →

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Manager
George
Harriet
Charles

Operations: Selection

Selection selects a subset of **tuples** matching a **predicate** φ.

$$\sigma_{\varphi}(\mathbf{R})$$

$$\downarrow$$

$$\{t \mid t \in \mathbf{R},\ a\ \theta\ v \text{ is true}\}$$

where φ has the form $a\theta v$:

- a is an attribute.
- θ is an operation like $<, \leq, >, \geq, =, \neq$.
- v is a value.

Operations: Selection

Selection selects a subset of **tuples** matching a predicate φ.

Employee(Name, EmpId, DeptName)

$\sigma_{EmpId>3000}(\mathbf{Employee}) \rightarrow$ (Name, EmpId, DeptName)

example

$\sigma_{EmpId>3000}($ *Employee*

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

$) \rightarrow$

Name	EmpId	DeptName
Harry	3415	Finance
George	3401	Finance

Operations: Selection

Selection selects a subset of **tuples** matching a predicate φ.

Dept(DeptName, Manager)

$\sigma_{Manager=Harriet}(\mathbf{Dept}) \rightarrow$ (DeptName,Manager)

example

$\sigma_{Manager=Harriet}($ *Dept*

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

$) \rightarrow$

DeptName	Manager
Sales	Harriet

Operations: Rename

Rename renames an **attribute** of a **tuple** according to a **substition rule** a/b.

$$\rho_{a/b}(\mathbf{R})$$

$$\downarrow$$

$$\{t \mid t[a/b] \in \mathbf{R}\}$$

where $t[a/b]$ is the tuple with attribute a renamed to b.

Operations: Rename

Rename renames an **attribute** of a **tuple** according to a **substition rule `a/b`**.

Employee(Name, EmpId, DeptName)

$\rho_{EmpId/Foo}$(**Employee**) → (Name, Id, DeptName)

example

$\rho_{EmpId/Foo}$ (Employee) → Employee

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Employee

Name	Foo	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Operations: Rename

Rename renames an **attribute** of a **tuple** according to a **substition rule `a/b`**.

Dept(DeptName, Manager)

$\rho_{Manager/Boss}$(**Dept**) → (DeptName,HeadHoncho)

example

$\rho_{Manager/Boss}$ (Dept) → Dept

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Dept

DeptName	Boss
Finance	George
Sales	Harriet
Production	Charles

Operations: Join

Join returns **a new relation** from relations R_1 and R_2 based on **join predicate θ**.

$$\mathbf{R_1} \bowtie_\theta \mathbf{R_2}$$

$$\downarrow$$

$$\sigma_\theta(\mathbf{R_1} \times \mathbf{R_2})$$

where **θ** is a predicate of the form **aθv**:

- **a** is an attribute.
- **θ** is an operation (most commonly =) .
- **v** is a value.

and where **×** is the Cartesian product.

Operations: Join

Join returns **a new relation** from relations R_1 and R_2 based on **join predicate θ**.

Employee(Name, EmpId, DeptName)

Dept(DeptName, Manager)

Employee$\bowtie_{DeptName=DeptName}$**Dept**

→ (Name, EmpId, DeptName, Manager)

example

Employee $\bowtie_\theta$ Dept →

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Name	EmpId	DeptName	Manager
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

where θ is DeptName=DeptName

Relational Algebra: Closure

All operations in the relational algebra are **closed**, meaning that **every operation on a relation yields a relation**.

As a programming language designer, closure is a **convenient** property.

E.g., once the **relation primitive** is defined in the language, no additional primitives are needed in order to define the language's **operations**.

Also, each operation's semantics can be considered in **isolation**. This **contains the potential explosion in complexity**, and it's what makes programming languages possible.

SQL

(example)

Optimizations

- Relational algebra **abstracts queries** from **data representation**.
- Modern SQL engines rewrite queries to make them **faster**.
- On-disk data layouts can be **automatically optimized** for specific queries.
- Efficient implementation is an active area of research.

Want to play with it?

MySQL™

You can download a copy for free at https://www.mysql.com/

Or on a Mac with Homebrew: "`brew install mysql`"

Recap & Next Class

This lecture:

SQL History

Relational Algebra

SQL Language

Next lecture:

Evaluation rules