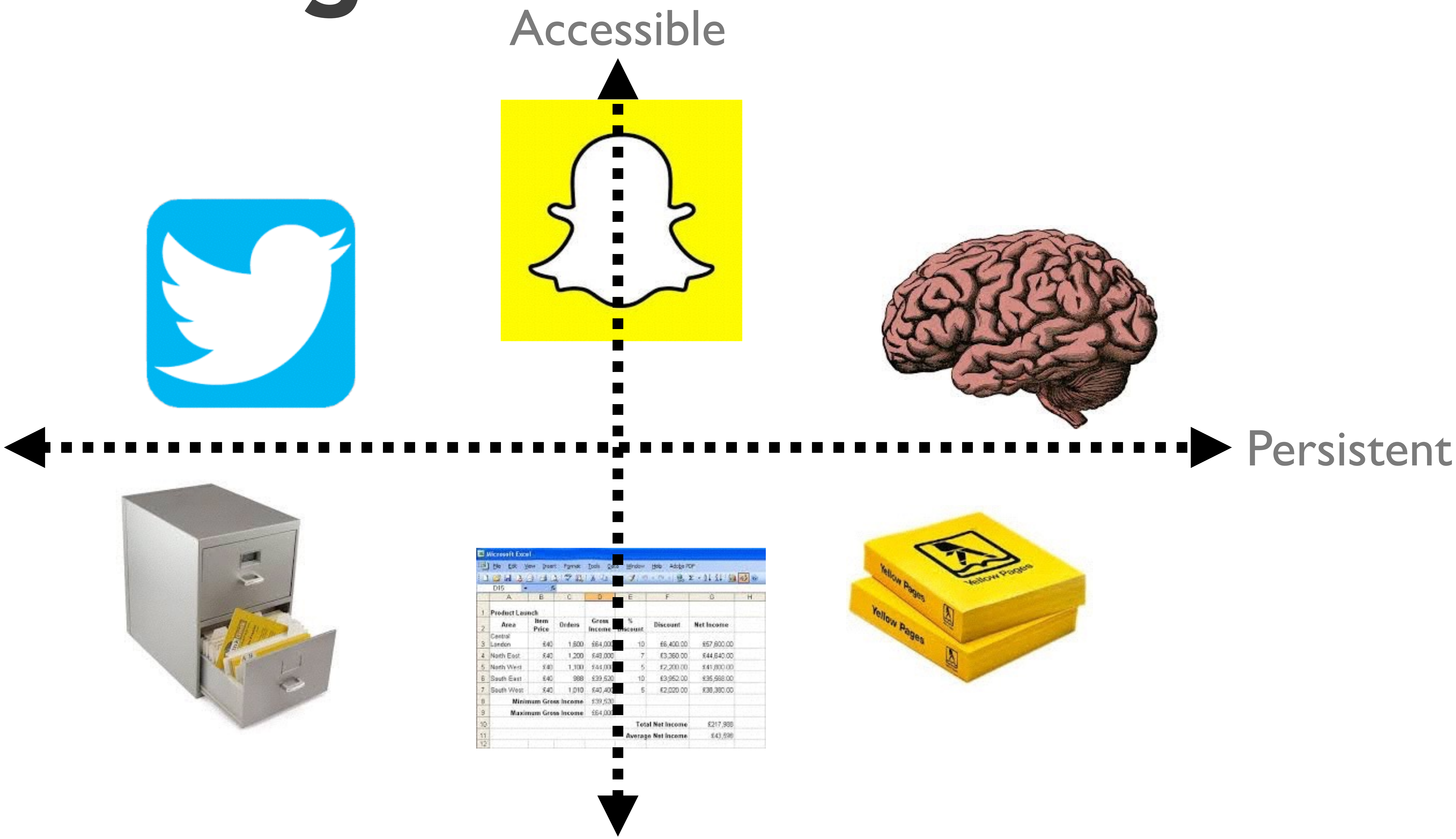
# Intro to Databases

*SQL*

# What is a database?

# Things that hold info



A database **persists** information and is **accessible** via code

**organized** **queryable** **manageable**

# Organized: Standard Storage Formatting

- Relational DBs are a collection of Tables (or *relations*)
- Tables have Columns (*attributes* / *fields*) that describe Rows (*instances* / *tuples*)
- Duplicate rows are not allowed
- Rows often have a primary key (unique identifier)

# Table / Relation

| | Column / Attribute / Field | Column / Attribute / Field | Column / Attribute / Field |
|---|---|---|---|
| | **ID** | **Name** | **Type** |
| Row / Tuple / Instance | 1 | Pikachu | lightning |
| Row / Tuple / Instance | 2 | Squirtle | water |
| Row / Tuple / Instance | 3 | Charmander | fire |
| Row / Tuple / Instance | 4 | Bulbasaur | grass |

# Queryable: via a Standard Language

- A simple, structured query language: SQL
- Declarative (vs. imperative)
- No more hand-rolled algorithms / data structures
- DBMS picks an efficient execution strategy based on indexes, data, workload etc.

# SQL

```sql
-- Pikachu, I choose you!
SELECT id, name
  FROM pokemon
  WHERE type = 'lightning'
  LIMIT 1;
```

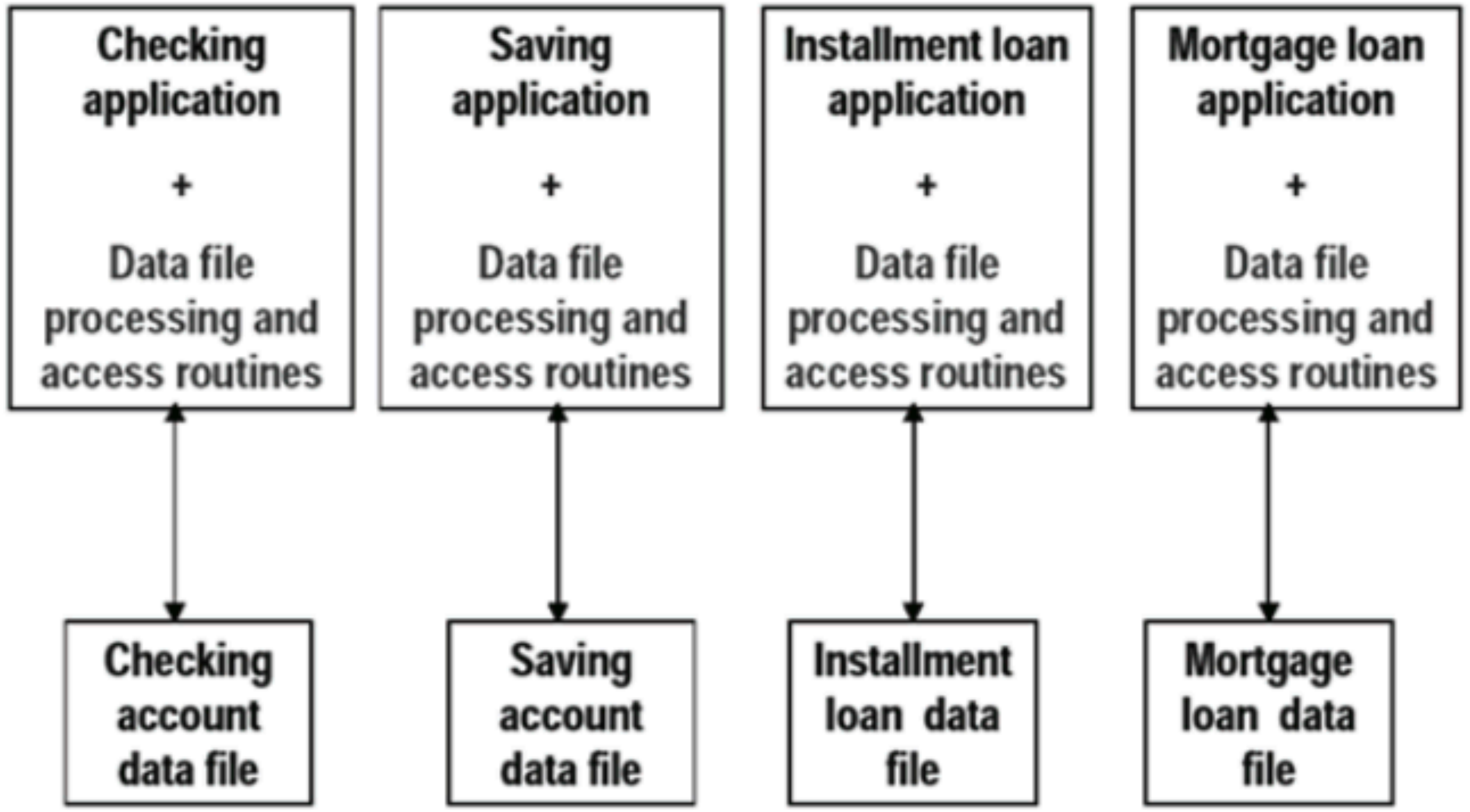
# Manageable: Easy, Safe, Performant

- Offloads work and requisite understanding of programming
- Knowledge is portable
- Abstraction
- Transfer data between systems
- DBMS (Database Management System) can make certain guarantees
  - prevent unsafe operations
  - built-in redundancies
  - handle multiple users, threads

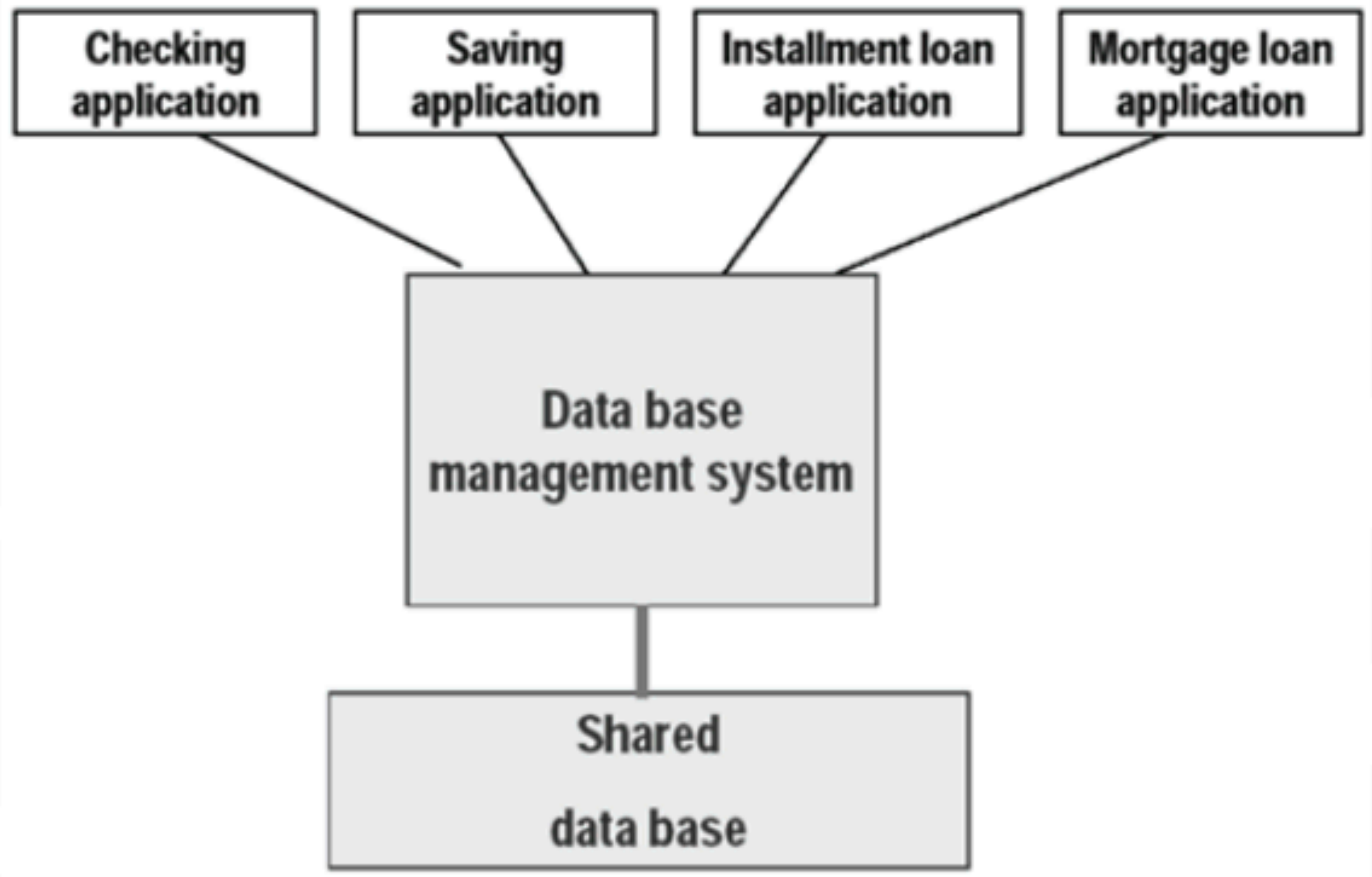
# How did we end up here?

# Before Relational DBs (ca. < 1970s)

- Data stored in custom "data files"
- Queried via application-specific code
- Advantages
  - Middle layer not needed
  - Solutions customized for each application
- Disadvantages
  - Hard to change the system
  - Data-transfer is difficult
  - Knowledge not compounding

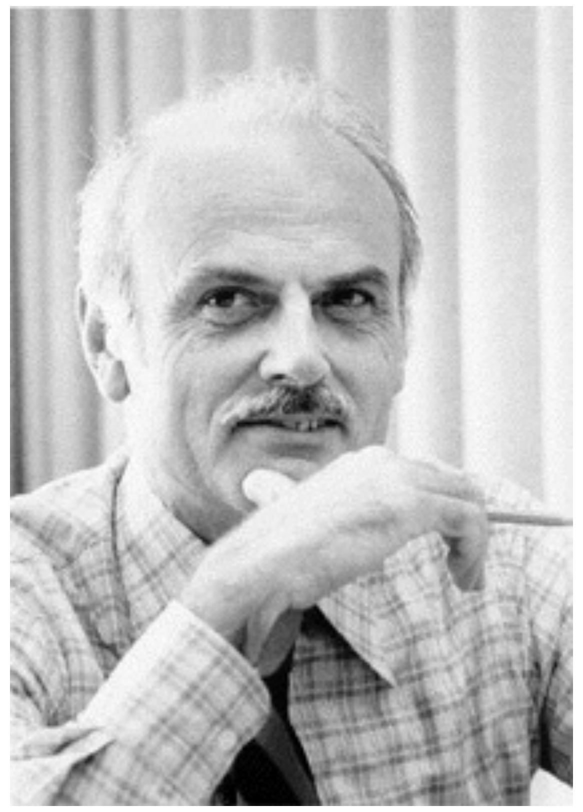


# Database Management Systems (DBMS)




- One layer and language to store and access data
- Sold as a way for “non-technical people” to manage data

# Relational Databases & Logic



- **1969: Edgar Frank "Ted" Codd outlines *relational model* of data**
- **Wrote Alpha (never implemented) as a *query language***
- **IBM slow to adopt his ideas**
  - Competitors started to do so
  - IBM team formed without Codd, created Structured English Query Lang
- **SEQUEL way better than what came before**
  - 1979: copied by Larry Ellison (from pre-launch papers / talks!) as "SQL"
- **SQL became the standard (ANSI 1986, ISO 1987)**
  - Codd continued to fault SQL compared to his theoretical model
  - The Third Manifesto: solve the *object-relational impedance mismatch*

# RDBMS vs NoSQL

- **A DBMS doesn't *have* to be relational**
  - Remember, DBMS is just an application that intelligently stores data and can answer requests to manage that data
- **Lately, many "NoSQL" or non-relational DBMSs have been gaining popularity**
  - Graph databases (e.g. Neo4j)
  - Document databases (e.g. MongoDB)
  - Hybrids (e.g. PostgreSQL)
- **RDBMSs still remain the #1 DB option for now**

# Pop Quiz

1. If we didn't have "databases" what could/would data persistence look like?
2. What is a "relation"? What does it mean if a database is non-relational?
3. What is SQL and why is it important?

# Appreciating Databases

- Ubiquitous
- Standardized
- Complex / deep
- Powerful: database admins are…
  - …feared by developers.
  - …but also taken for granted until things break. 😔
  - …befriended by those who want to mine (business people & governments!)

# ACID Guarantees

- Atomicity
- Consistency
- Isolation
- Durability

# What happens if the database crashes?

- **Suppose we have a banking system with client accounts**
- **Every debit must have a matching credit**
- **Imagine a crash results in only one table being updated**
  - Database inconsistency → unexpected data and software crashes/bugs
  - Financial risk for clients

# Transaction Code Example

```sql
BEGIN;
UPDATE accounts SET balance = balance - 100.00
  WHERE name = 'Alice';
UPDATE accounts SET balance = balance + 100.00
  WHERE name = 'Bob';
COMMIT;
```

` https://www.postgresql.org/docs/8.3/static/tutorial-transactions.html

# Atomic Transactions

- atomic transaction: A set of database operations that must occur together
  - i.e. A debit to one bank account, and a credit to another
- A transaction must either succeed or fail; it cannot partially complete.
- Every database query is represented by a transaction

# Consistency

- **Specify rules that columns need to follow**
  - Gender column can only contain M, F, or U.
  - Savings account must start with S or checking with C
  - Column cannot be null
- **Protect the database from inconsistencies and simplify software logic**
  - Allows software to make assumptions about underlying data

# Transaction Code Example

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
  WHERE name = 'Alice';
UPDATE accounts SET balance = balance + 100.00
  WHERE name = 'Bob';
COMMIT;
```

` https://www.postgresql.org/docs/8.3/static/tutorial-transactions.html

# Resource Management

- Processes can be readers and writers
- Files can have many readers
- If a process has a writer, no other process can read from it, and no other process can write to it

# Proposed File Scheme

- Suppose that we have decided not to use a database and instead store our data in a series of files.

- How might our setup fail to serve queries from multiple users?

# Deadlock

# Databases give us concurrency (Isolation)

- Multiple clients can make queries to read and update without the risk of deadlock or starvation.

# Persistence/Durability

- Files are also persistence (store information without power)

# Enough Theory. Examples!

# Example DB

Primary Key (PK)

Foreign Key (FK)

Students

| ID | Name | Age | Gender | Address |
|---|---|---|---|---|
| 1 | Nick D. | 20 | M | 2 |
| 2 | Andy D. | 28 | M | 2 |
| 3 | Beth M. | 23 | F | 1 |
| 4 | Lisa N. | 20 | F | 4 |

Addresses

| ID | Street | Zip | City | State |
|---|---|---|---|---|
| 1 | 423 Main St. | 60647 | Chicago | IL |
| 2 | 13 Main St | 60655 | Barrington | IL |
| 3 | 15 Main St | 60651 | Elsewhere | IL |
| 4 | 14 Main St | 60650 | Chicago | IL |

# All 20 Year Old Students

Students

| ID | Name | Age | Gender | Address |
|---|---|---|---|---|
| 1 | Nick D. | 20 | M | 2 |
| 2 | Andy D. | 28 | M | 2 |
| 3 | Beth M. | 23 | F | I |
| 4 | Lisa N. | 20 | F | 4 |

Result Set:

20 Year Old Students

| ID | Name | Age |
|---|---|---|
| 1 | Nick D. | 20 |
| 4 | Lisa N. | 20 |

```
SELECT ID, Name, Age
  FROM Students
  WHERE Age = 20;
```

## Students

| ID | Name | Age | Gender | Address |
|---|---|---|---|---|
| 1 | Nick D. | 20 | M | 2 |
| 2 | Andy D. | 28 | M | 2 |
| 3 | Beth M. | 23 | F | 1 |
| 4 | Lisa N. | 20 | F | 4 |

## Addresses

| ID | Street | Zip | City | State |
|---|---|---|---|---|
| 1 | 423 Main St. | 60647 | Chicago | IL |
| 2 | 13 Main St. | 60655 | Barrington | IL |
| 3 | 15 Main St. | 60651 | Elsewhere | IL |
| 4 | 14 Main St. | 60650 | Chicago | IL |

```
SELECT Students.ID, Name, Street, Zip, City
  FROM Students
  JOIN Addresses
    ON Students.Address = Addresses.ID;
```

## Result Set:

## Students with Addresses

| Student.ID | Name | Street | Zip | City |
|---|---|---|---|---|
| 1 | Nick D. | 13 Main St. | 60655 | Barrington |
| 2 | Andy D. | 13 Main St. | 60655 | Barrington |
| 3 | Beth M. | 423 Main St. | 60647 | Chicago |
| 4 | Lisa N. | 14 Main St. | 60650 | Chicago |

## Students

| ID | Name | Age | Gender | Address |
|---|---|---|---|---|
| 1 | Nick D. | 20 | M | 2 |
| 2 | Andy D. | 28 | M | 2 |
| 3 | Beth M. | 23 | F | 1 |
| 4 | Lisa N. | 20 | F | 4 |

## Addresses

| ID | Street | Zip | City | State |
|---|---|---|---|---|
| 1 | 423 Main St. | 60647 | Chicago | IL |
| 2 | 13 Main St. | 60655 | Barrington | IL |
| 3 | 15 Main St. | 60651 | Elsewhere | IL |
| 4 | 14 Main St. | 60650 | Chicago | IL |

```
SELECT Students.ID, Name, Street, Zip, City
  FROM Students
  JOIN Addresses
    ON Students.Address = Addresses.ID
 WHERE Addresses.City = 'Chicago';
```

## Result Set:

## Students with Addresses

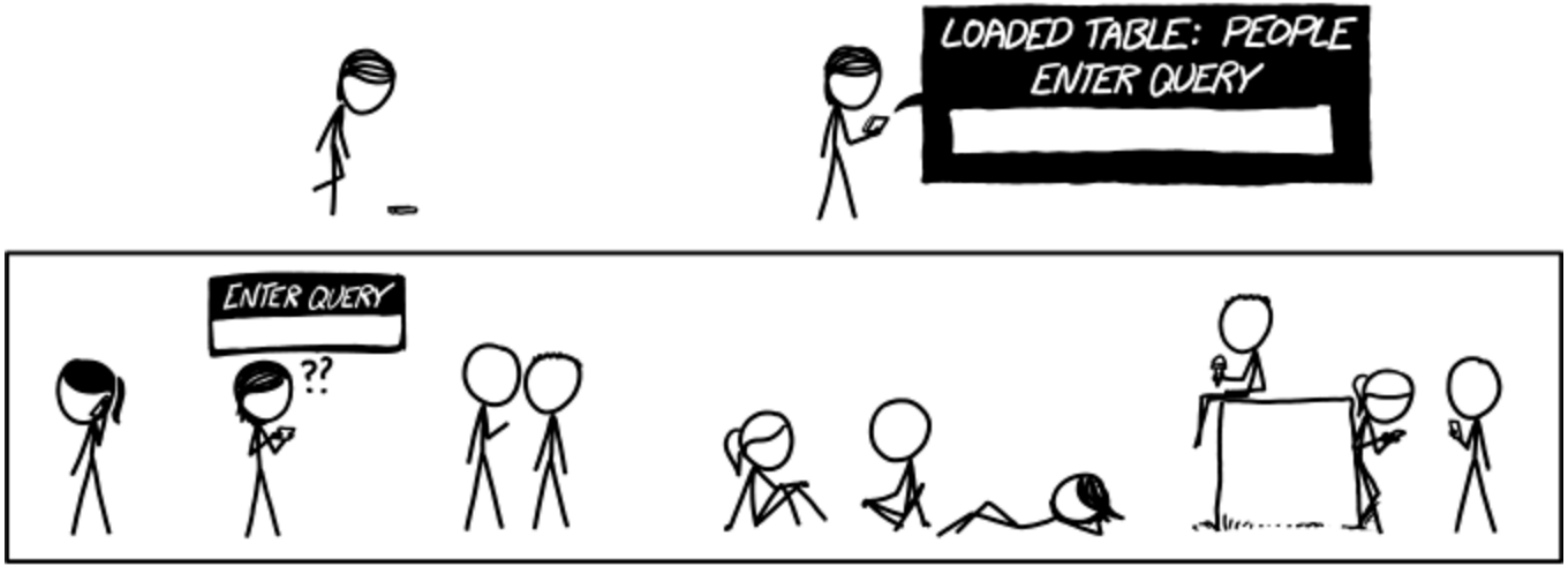
| Student.ID | Name | Street | Zip | City |
|---|---|---|---|---|
| 3 | Beth M. | 423 Main St. | 60647 | Chicago |
| 4 | Lisa N. | 14 Main St. | 60650 | Chicago |

# Some Common SQL Keywords

| Keyword | Action |
|---|---|
| SELECT | Which COLUMNS to include in output table (shrinks the result horizontally!) |
| FROM | Which TABLE to pull data from |
| JOIN | Another TABLE to glue / concatenate to the output |
| ON | What COLUMNS must match when joining two tables |
| WHERE | Which ROWS to include in the output table (shrinks the result vertically!) |

# Schema and Content

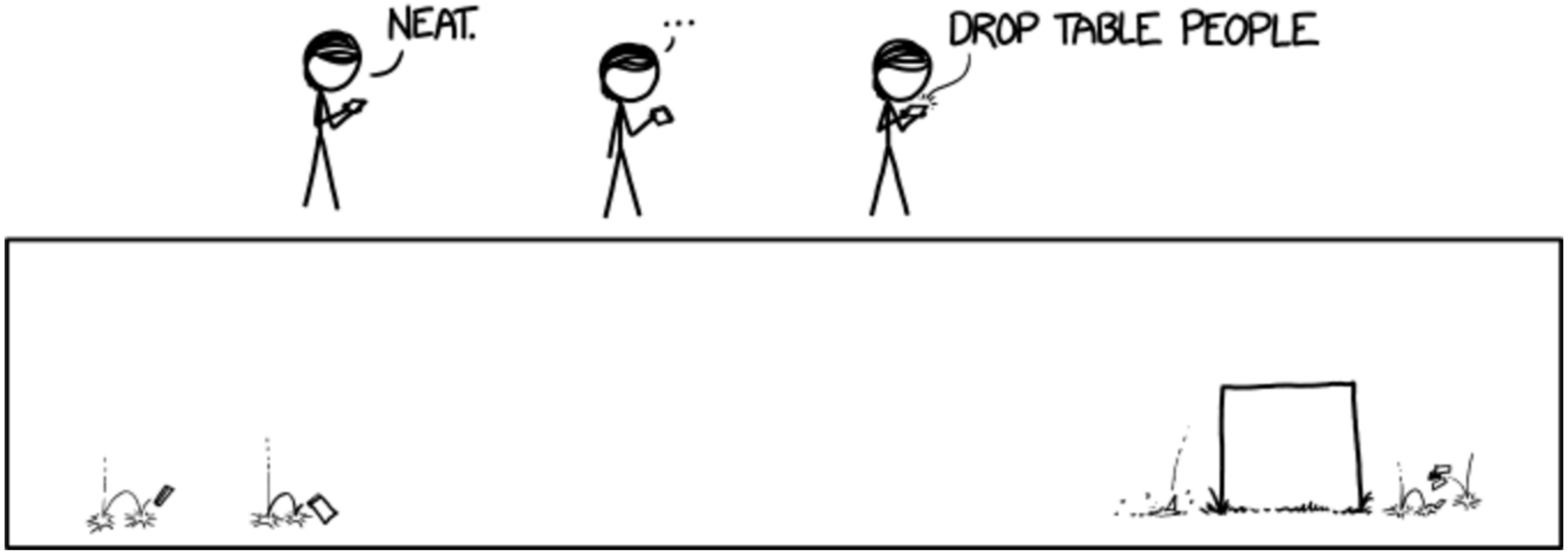
- Schema: table's blueprint for data shape/format
- Content: actual data (a row) e.g. {1, "Bart S.", 10, "M"}
- A schema is used to validate incoming content


LOADED TABLE: PEOPLE
ENTER QUERY
ENTER QUERY
??


SELECT * FROM PEOPLE WHERE AGE > 30


SELECT * FROM PEOPLE WHERE ANNUAL_INCOME > 100000


SELECT * FROM PEOPLE WHERE AFRAID_OF_FLYING = TRUE


NEAT.
...
DROP TABLE PEOPLE

# SQL

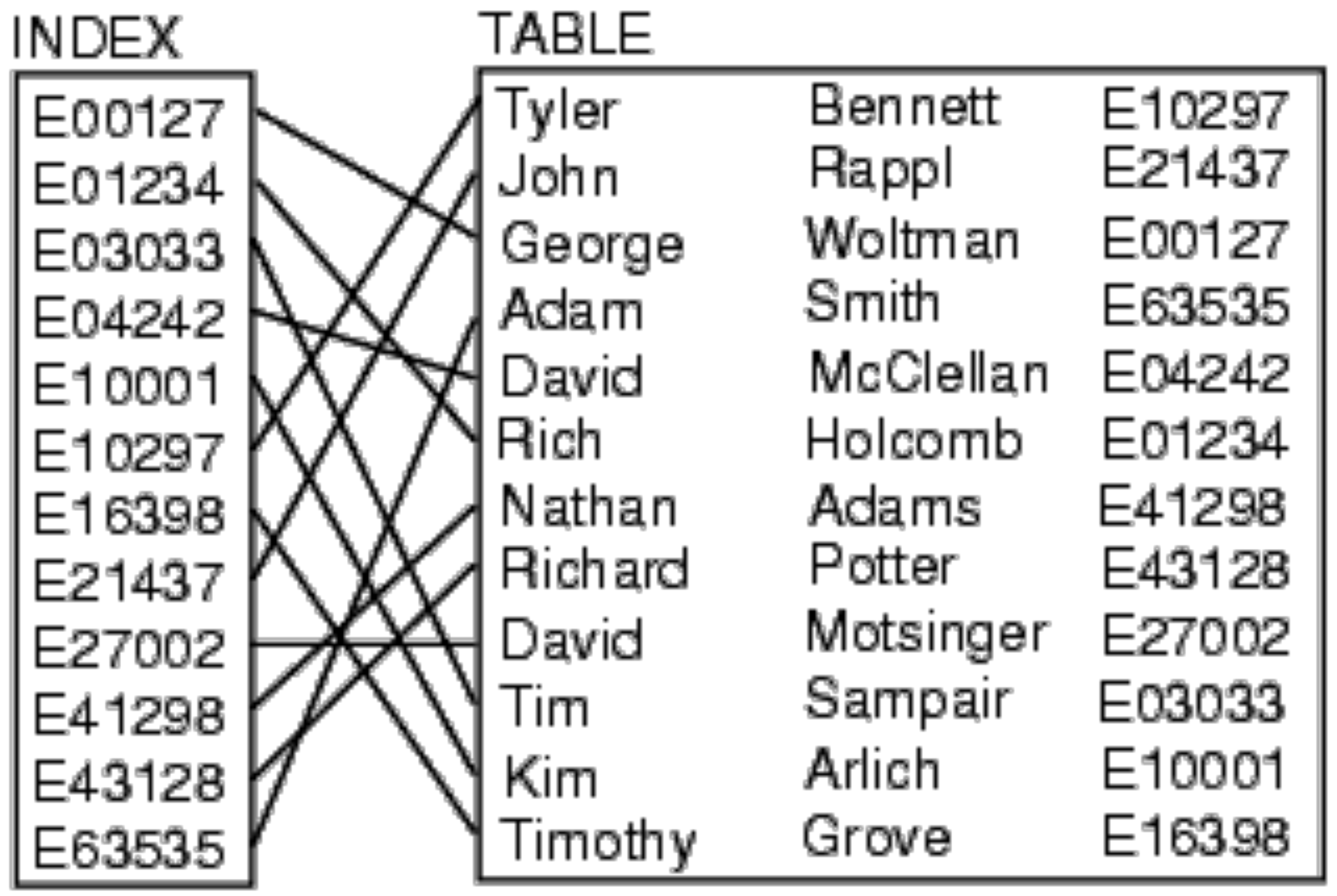
## SQL is used to create/read/update/delete (CRUD) data from a database

- INSERT: Insert new rows into a table
- SELECT: Get data from a database
- UPDATE: Update existing rows in a table
- DELETE: Delete rows from a table

- CREATE / DROP: Make / delete new dbs/tables/views/indexes

# Quick Notes on Indexing

- Creates another data structure, which holds
  - row's indexed field's value
  - a pointer to the record the row relates to
- Index on Employee_ID
  - Index points to record
  - Index is ordered




https://www.progress.com/tutorials/odbc/using-indexes

# Quick Notes on Indexing

- A database driver can use indexes to find records quickly
- Without an index, the driver's worst case is searching the entire database table to find a record for a specific Employee_ID
- Downside
  - Extra storage space requirements (though less than a copy of the table)
  - Redundant data means update/insert/delete takes more time


https://lol.browserling.com/tables.png

# Find all students from Springfield Elementary

Student

| ID | Name | Age |
|---|---|---|
| 1 | Bart S. | 10 |
| 2 | Lisa S. | 8 |
| 3 | Jim F. | 13 |
| 4 | Joan B. | 15 |

Enrollment

| StudentID | SchoolID |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |

School

| ID | Name | Level |
|---|---|---|
| 1 | Springfield Elementary | E |
| 2 | Brook Middle | M |
| 3 | Springbrook High | H |
| 4 | Springfield University | U |

```
SELECT *
  FROM Student
  INNER JOIN Enrollment
    ON Student.ID = Enrollment.StudentID
  INNER JOIN School
    ON Enrollment.SchoolID = School.ID;
```

```sql
SELECT *
FROM Student
INNER JOIN Enrollment
  ON Student.ID = Enrollment.StudentID
INNER JOIN School
  ON Enrollment.SchoolID = School.ID;
```

| ID | Name | Age |
|---|---|---|
| 1 | Bart S. | 10 |
| 2 | Lisa S. | 8 |

| StudentID | SchoolID |
|---|---|
| 1 | 1 |
| 2 | 1 |

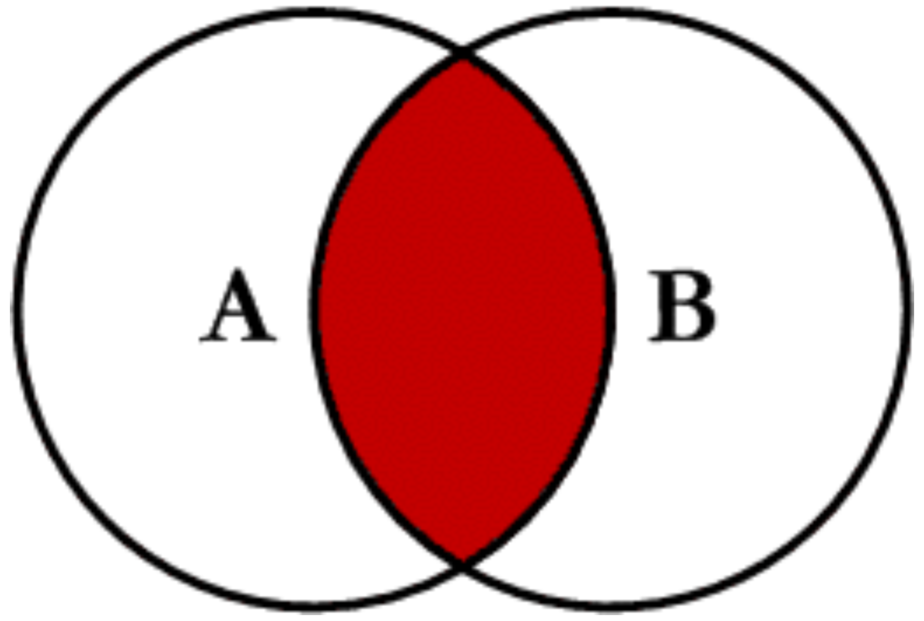
| ID | Name | Level |
|---|---|---|
| 1 | Springfield Elementary | E |
| 2 | Brook Middle | M |

| Student.ID | Student.Name | Age | StudentID | SchoolID | School.ID | School.Name | Level |
|---|---|---|---|---|---|---|---|
| 1 | Bart S. | 10 | 1 | 1 | 1 | Springfield Elementary | E |
| 2 | Lisa S. | 8 | 2 | 1 | 1 | Springfield Elementary | E |
| 3 | Jim F. | 13 | 3 | 2 | 2 | Brook Middle | M |
| 4 | Joan B. | 15 | 4 | 3 | 3 | Springbrook High | H |

```sql
SELECT *
  FROM Student
  INNER JOIN Enrollment
    ON Student.ID = Enrollment.StudentID
  INNER JOIN School
    ON Enrollment.SchoolID = School.ID;
  WHERE School.Name = 'Springfield Elementary';
```

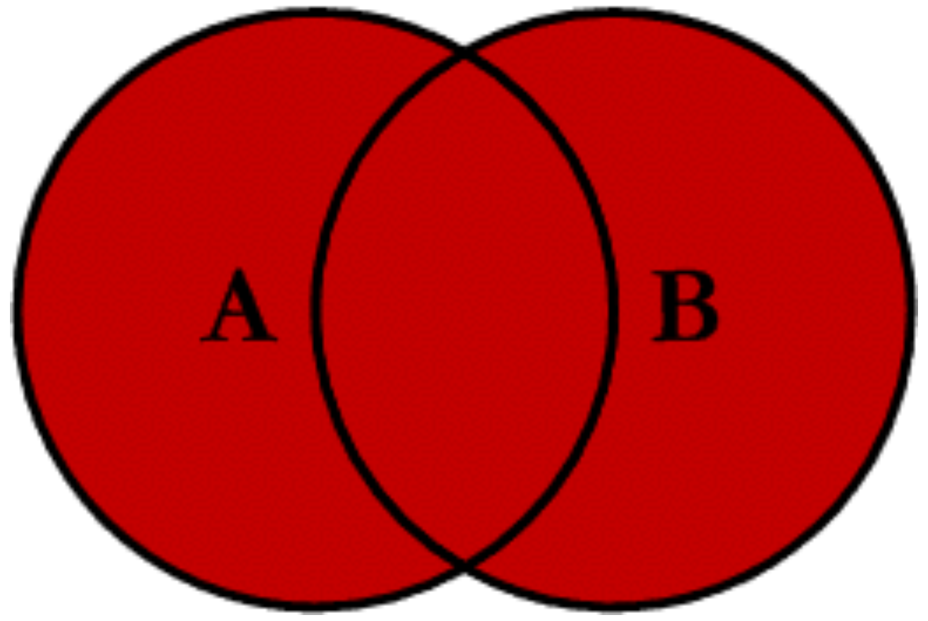
| Student.ID | Student.Name | Age | StudentID | SchoolID | School.ID | School.Name | Level |
|---|---|---|---|---|---|---|---|
| 1 | Bart S. | 10 | 1 | 1 | 1 | Springfield Elementary | E |
| 2 | Lisa S. | 8 | 2 | 1 | 1 | Springfield Elementary | E |

## Inner Join



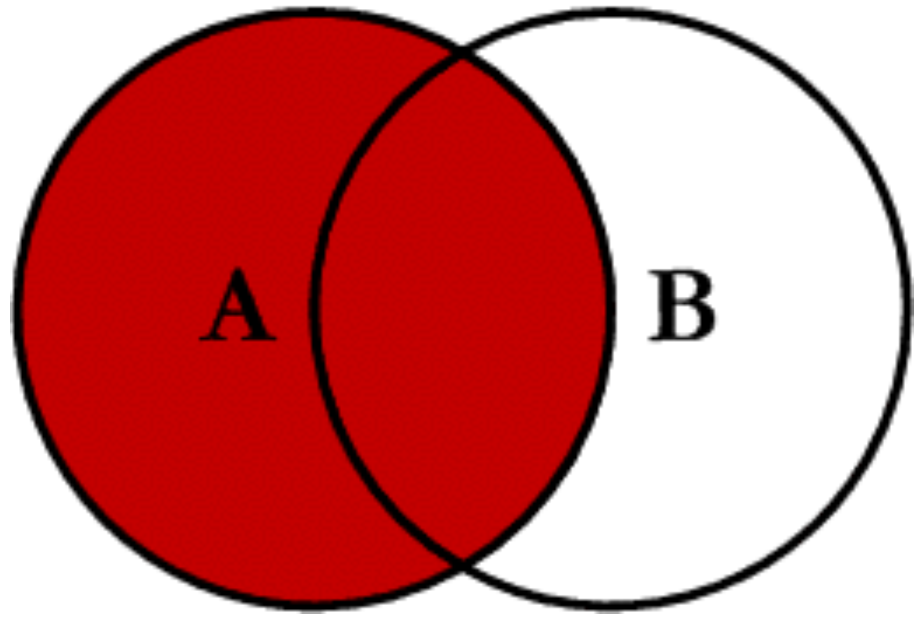

```
SELECT *
FROM A
INNER JOIN B
ON A.Key = B.Key
```

## Outer Join



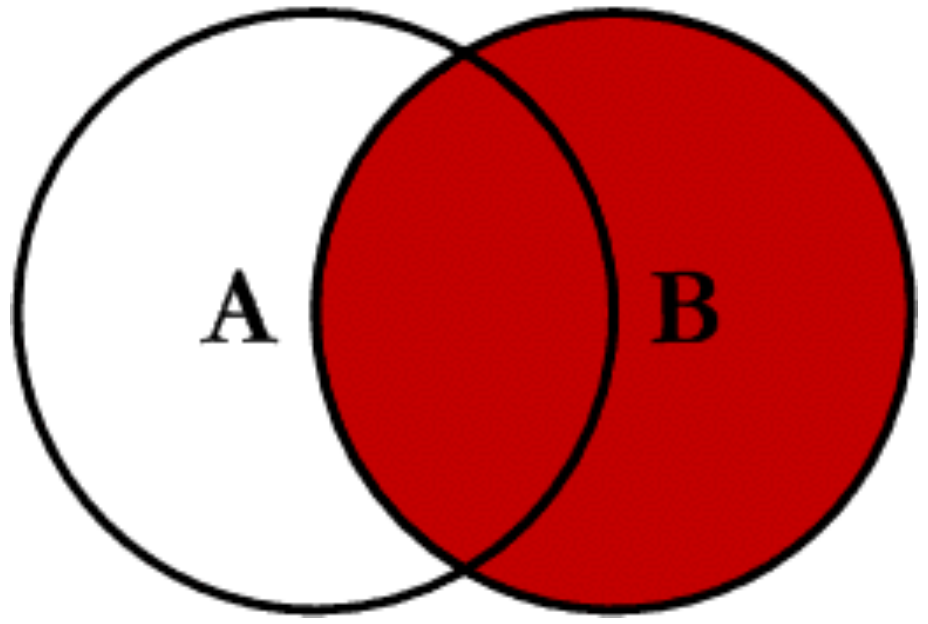

```
SELECT *
FROM A
FULL OUTER JOIN B
ON A.Key = B.Key
```

## Left Join




```
SELECT *
FROM A
LEFT JOIN B
ON A.Key = B.Key
```

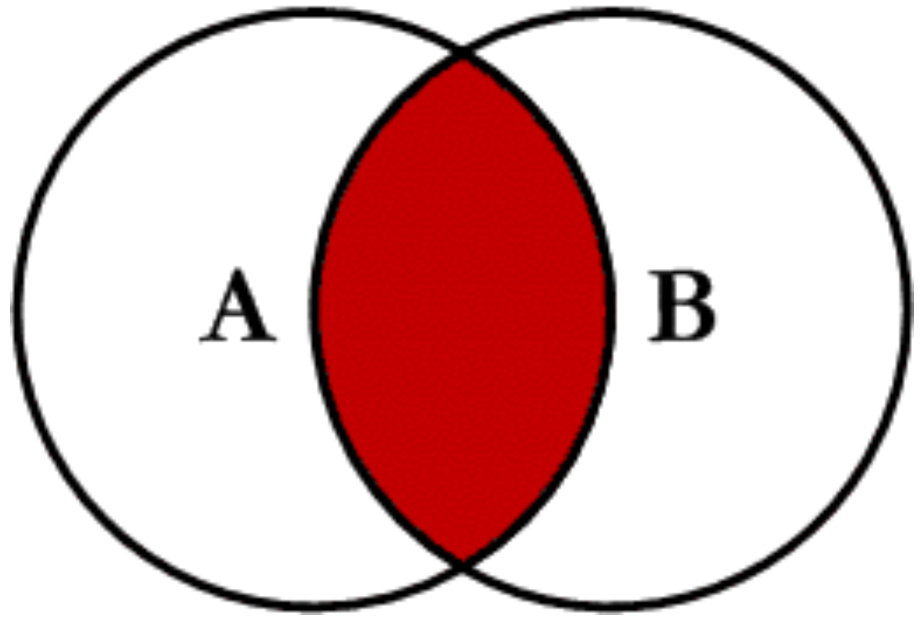
## Right Join




```
SELECT *
FROM A
RIGHT JOIN B
ON A.Key = B.Key
```

http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

# Inner Join




```sql
SELECT pets.name, owners.name
FROM owners
INNER JOIN pets
ON pets.ownerID = owners.ID
```

OWNERS

| ID | name |
|---|---|
| 1 | Geordi |
| 2 | Janeway |
| 3 | Data |
| 4 | Spok |

PETS

| ID | ownerID | type | name |
|---|---|---|---|
| 1 | 4 | Monkey | Mittens |
| 2 | null | Lizard | Carol |
| 3 | 1 | Dog | Rufus |
| 4 | 2 | Cat | Fireball |

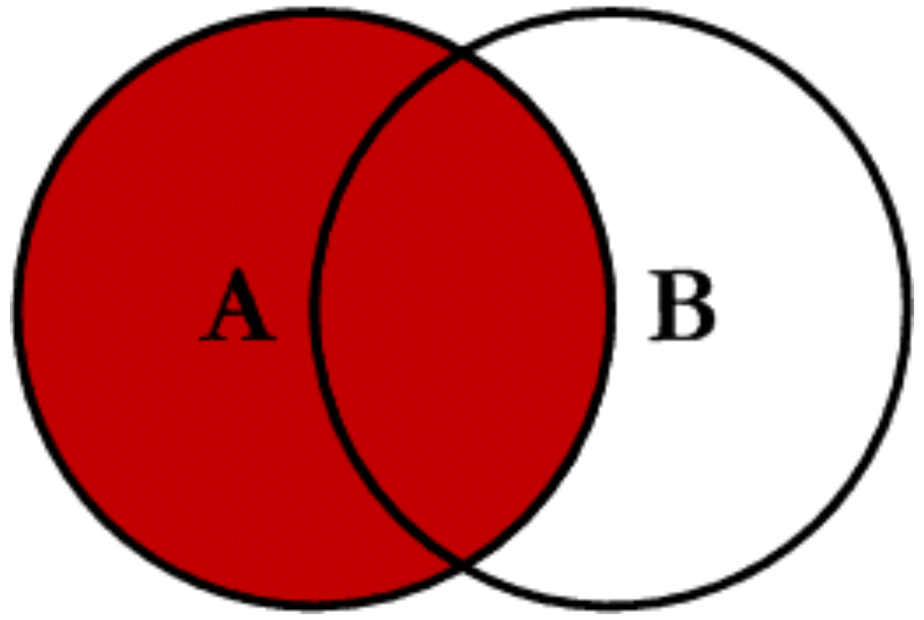
| pets.name | owners.name |
|---|---|
| Mittens | Spok |
| Rufus | Geordi |
| Fireball | Janeway |

## PETS

| ID | ownerID | type | name |
|---|---|---|---|
| 1 | 4 | Monkey | Mittens |
| 2 | null | Lizard | Carol |
| 3 | 1 | Dog | Rufus |
| 4 | 2 | Cat | Fireball |

| pets.name | owners.name |
|---|---|
| Mittens | Spok |
| Rufus | Geordi |
| Fireball | Janeway |
| null | Data |

# Left Join




```sql
SELECT pets.name, owners.name
FROM owners
LEFT JOIN pets
ON pets.ownerID = owners.ID
```

## OWNERS

| ID | name |
|---|---|
| 1 | Geordi |
| 2 | Janeway |
| 3 | Data |
| 4 | Spok |

| pets.name | owners.name |
|---|---|
| Mittens | Spok |
| Carol | null |
| Rufus | Geordi |
| Fireball | Janeway |

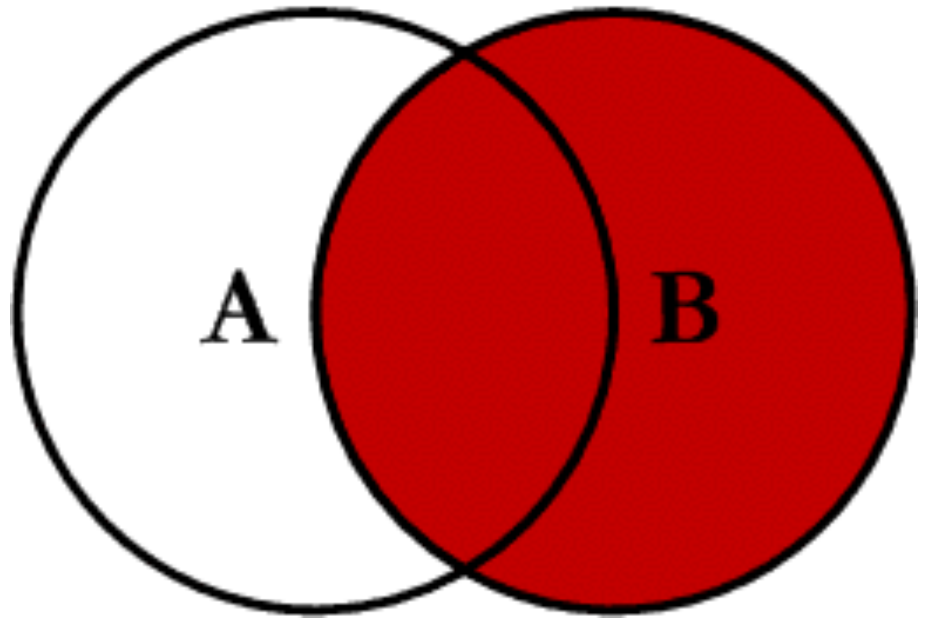
PETS

| ID | ownerID | type | name |
|---|---|---|---|
| 1 | 4 | Monkey | Mittens |
| 2 | null | Lizard | Carol |
| 3 | 1 | Dog | Rufus |
| 4 | 2 | Cat | Fireball |

OWNERS

| ID | name |
|---|---|
| 1 | Geordi |
| 2 | Janeway |
| 3 | Data |
| 4 | Spok |

# Right Join

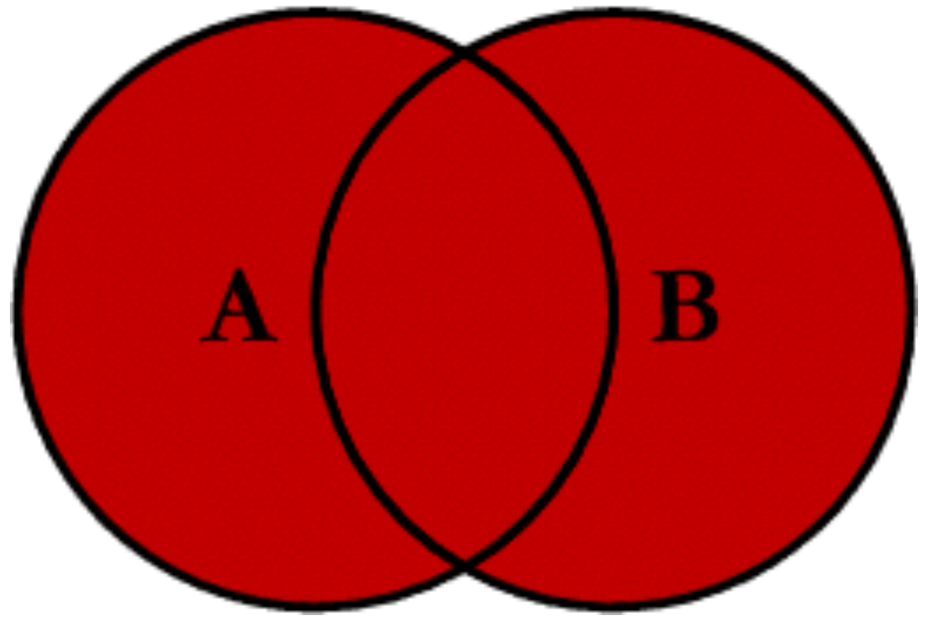



```
SELECT pets.name, owners.name
FROM owners
RIGHT JOIN pets
ON pets.ownerID = owners.ID
```

OWNERS

| ID | name |
|---|---|
| 1 | Geordi |
| 2 | Janeway |
| 3 | Data |
| 4 | Spok |

# Outer Join




```
SELECT pets.name, owners.name
FROM owners
FULL OUTER JOIN pets
ON pets.ownerID = owners.ID
```

PETS

| ID | ownerID | type | name |
|---|---|---|---|
| 1 | 4 | Monkey | Mittens |
| 2 | null | Lizard | Carol |
| 3 | 1 | Dog | Rufus |
| 4 | 2 | Cat | Fireball |

| pets.name | owners.name |
|---|---|
| Mittens | Spok |
| Carol | null |
| Rufus | Geordi |
| Fireball | Janeway |
| null | Data |

# Pop Quiz

1. What is ACID compliance and why is it important?
2. Give examples of how to do all CRUD operations in SQL.
3. Name the different types of joins and describe the differences between them.

# SQL POWER

# AS

| ID | Name | Age |
|---|---|---|
| 1 | Bart S. | 10 |
| 2 | Lisa S. | 8 |
| 3 | Jim F. | 13 |
| 4 | Joan B. | 15 |

| StudentID | SchoolID |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |

| ID | Name | Level |
|---|---|---|
| 1 | Springfield Elementary | E |
| 2 | Brook Middle | M |
| 3 | Springbrook High | H |
| 4 | Springfield University | U |

```sql
SELECT *
  FROM Student AS st
  INNER JOIN Enrollment AS e
    ON st.ID = e.StudentID
  INNER JOIN School as sc
    ON e.SchoolID = sc.ID;
```

| st.ID | st.Name | Age | StudentID | SchoolID | sc.ID | sc.Name | Level |
|---|---|---|---|---|---|---|---|
| 1 | Bart S. | 10 | 1 | 1 | 1 | Springfield Elementary | E |
| 2 | Lisa S. | 8 | 2 | 1 | 1 | Springfield Elementary | E |
| 3 | Jim F. | 13 | 3 | 2 | 2 | Brook Middle | M |
| 4 | Joan B. | 15 | 4 | 3 | 3 | Springbrook High | H |

# AS
# (without AS)

| ID | Name | Age |
|---|---|---|
| 1 | Bart S. | 10 |
| 2 | Lisa S. | 8 |
| 3 | Jim F. | 13 |
| 4 | Joan B. | 15 |

| StudentID | SchoolID |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |

| ID | Name | Level |
|---|---|---|
| 1 | Springfield Elementary | E |
| 2 | Brook Middle | M |
| 3 | Springbrook High | H |
| 4 | Springfield University | U |

```
SELECT *
  FROM Student st
  INNER JOIN Enrollment e
    ON st.ID = e.StudentID
  INNER JOIN School sc
    ON e.SchoolID = sc.ID;
```

| st.ID | st.Name | Age | StudentID | SchoolID | sc.ID | sc.Name | Level |
|---|---|---|---|---|---|---|---|
| 1 | Bart S. | 10 | 1 | 1 | 1 | Springfield Elementary | E |
| 2 | Lisa S. | 8 | 2 | 1 | 1 | Springfield Elementary | E |
| 3 | Jim F. | 13 | 3 | 2 | 2 | Brook Middle | M |
| 4 | Joan B. | 15 | 4 | 3 | 3 | Springbrook High | H |

# GROUP BY + COUNT

| ID | Name | Age |
|---|---|---|
| 1 | Bart S. | 10 |
| 2 | Lisa S. | 8 |
| 3 | Jim F. | 13 |
| 4 | Joan B. | 15 |

| StudentID | SchoolID |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |

| ID | Name | Level |
|---|---|---|
| 1 | Springfield Elementary | E |
| 2 | Brook Middle | M |
| 3 | Springbrook High | H |
| 4 | Springfield University | U |

```sql
SELECT Name, COUNT(*)
  FROM School
  INNER JOIN Enrollment
    ON School.ID = Enrollment.StudentID
  GROUP BY Name;
```

| Name | COUNT(*) |
|---|---|
| Springfield Elementary | 2 |
| Brook Middle | 1 |
| Springbrook High | 1 |
| Springfield University | 0 |

# ORDER BY

```sql
SELECT *
  FROM Student
  ORDER BY Age DESC;
```

| ID | Name | Age |
|---|---|---|
| 1 | Bart S. | 10 |
| 2 | Lisa S. | 8 |
| 3 | Jim F. | 13 |
| 4 | Joan B. | 15 |

| StudentID | SchoolID |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |

| ID | Name | Level |
|---|---|---|
| 1 | Springfield Elementary | E |
| 2 | Brook Middle | M |
| 3 | Springbrook High | H |
| 4 | Springfield University | U |

| ID | Name | Age |
|---|---|---|
| 4 | Joan B. | 15 |
| 3 | Jim F. | 13 |
| 1 | Bart S. | 10 |
| 2 | Lisa S. | 8 |

# SUB-QUERIES

```sql
SELECT ID, Name, Age
  FROM Student
  INNER JOIN Enrollment
    ON Student.ID = Enrollment.StudentID
  INNER JOIN (
    SELECT SchoolID
      FROM Student
      WHERE Student.Name = 'Lisa S.'
      INNER JOIN Enrollment
        ON Student.ID = Enrollment.StudentID
  ) AS LisaSchools
    ON LisaSchools.SchoolID = Enrollment.SchoolID
  WHERE Name != 'Lisa S.';
```

| ID | Name | Age |
|---|---|---|
| 1 | Bart S. | 10 |
| 2 | Lisa S. | 8 |
| 3 | Jim F. | 13 |
| 4 | Joan B. | 15 |

| StudentID | SchoolID |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |

| ID | Name | Level |
|---|---|---|
| 1 | Springfield Elementary | E |
| 2 | Brook Middle | M |
| 3 | Springbrook High | H |
| 4 | Springfield University | U |

| ID | Name | Age |
|---|---|---|
| 1 | Bart S. | 10 |

# WORKSHOP