Welcome to CO 572: Advanced Databases

Holger Pirk

Slides as of 10/02/22 09:24

Today's Lecture

Purpose: Figuring stuff out

- Who I am and what I am doing here
- Why this class exists
  - Why do we need Data Management Systems
  - Why should you learn about them?
- What we're trying to achieve in this class
- What I will expect
- What you can expect
- Also: Some Admin
- Relational Algebra

Hi, I am Holger (not Dr. Pirk, please)

Three truths and one lie

- I am German
- I joined the department two years ago
- I am very into coffee
- I like data

What do Germans like above all other things – apologies for the stereotype?

Efficiency!

Incidentally, that is why I teach this course!

This class is called "Advanced Databases"

I would prefer "Advanced Data Management" or "Database Management Systems"

This raises a question:

What is a Database Management System?

Well, a System that manages a Database

So what are

- a Database
- Management
- a System

Database Management Systems

What is a Database?

Merriam Webster

Database *A usually large collection of organized data*

Data *Factual information*

In Context

- Pretty much any structured collection of data points/data items
  - A relational table
  - A set in your favorite programming language
  - A vector in your favorite programming language
  - A graph
  - A stack of index cards

What is Management?

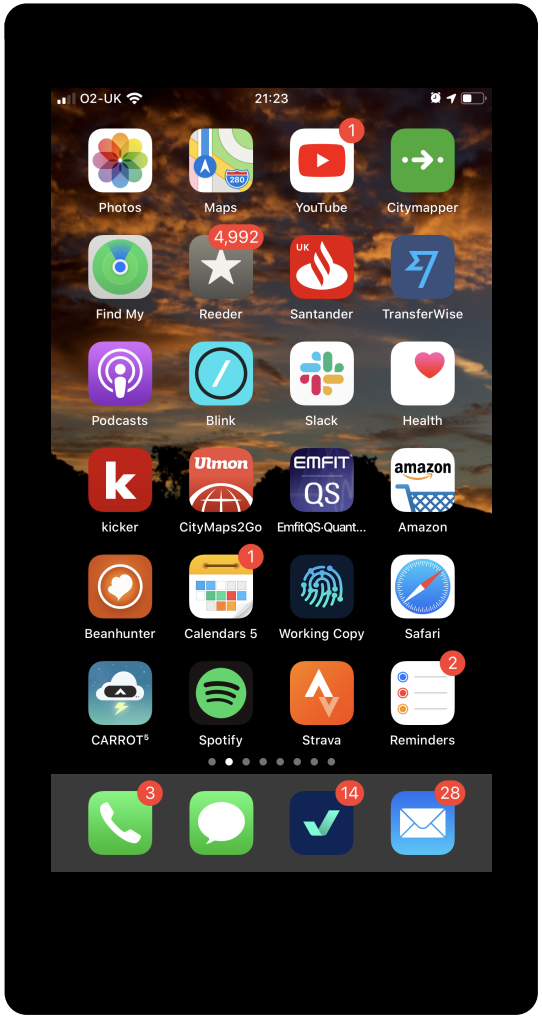
Merriam Webster

- *the conducting or supervising of something*
- *to handle or direct with a degree of skill*
- *to work upon or try to alter for a purpose*
- *judicious use of means to accomplish an end*

What is data management?

- Roughly, those parts of an application that "work with" data

Who needs data management?



Every single one of the Apps on my iPhone

let's call them data-intensive

What is a data-intensive application?

- One that acquires, stores and/or processes a significant amount of data

Claim: Data Management is hard!

Application 1

Szenario You work at a large hospital. At any given time, there are 800 in-patients each producing a continuous stream of data (one sample per second): heartrate, blood-pressure, temperature, etc. (five metrics). There are 200 doctors and nurses, each producing a textual report every 10 minutes and 80 lab technicians producing a structured dataset (containing say 10 metrics) every five minutes.

Requirements Every datapoint needs to be stored reliably. Once it is stored, no data shall be lost ($p < 0.001$).

Objective Build an application that fulfills the requirements using only technology that you understand (i.e., could build yourself).

Application 2

Szenario You are developing an interactive dashboard for a global retail company. The company has acquired and stored 500GBs of sales, inventory and customer records. The dashboard shall provide interactive access to calculated statistics.

Requirements The system shall allow filtering of the dataset using conjunctions of (in)equality predicates (e.g., `country = "UK" and sales > 500000`). The system shall support the calculation of sums of records as well as limiting the results to the top N records (e.g., the 15 customers with the highest sales volume). The response time for all queries shall be below 1 second.

Objective Build an application that fulfills the requirements using only technology that you understand (i.e., could build yourself).

Let's keep those in mind and discuss them in the Q&A

But one thing is clear: they are non-trivial

Fortunately, Data Management follows patterns!

Typical data-intensive application patterns

Online Transaction Processing

- Lots of small updates to a persistent database
- Focus is on throughput
- ACID is key

Online Analytical Processing

- Running a single data analysis task
- Focus is on latency
- Queries are ad-hoc

Reporting

- Running a many data analysis tasks given a fixed time budget
- Focus is on resource efficiency
- Queries are known in advance

Hybrid Transactional/Analytical Processing

- Small updates interwoven with larger analytics

Thus, data management is outsourced to "Data Management Systems"

What is a System?

Merriam Webster

- *a regularly interacting or interdependent group of items forming a unified whole*

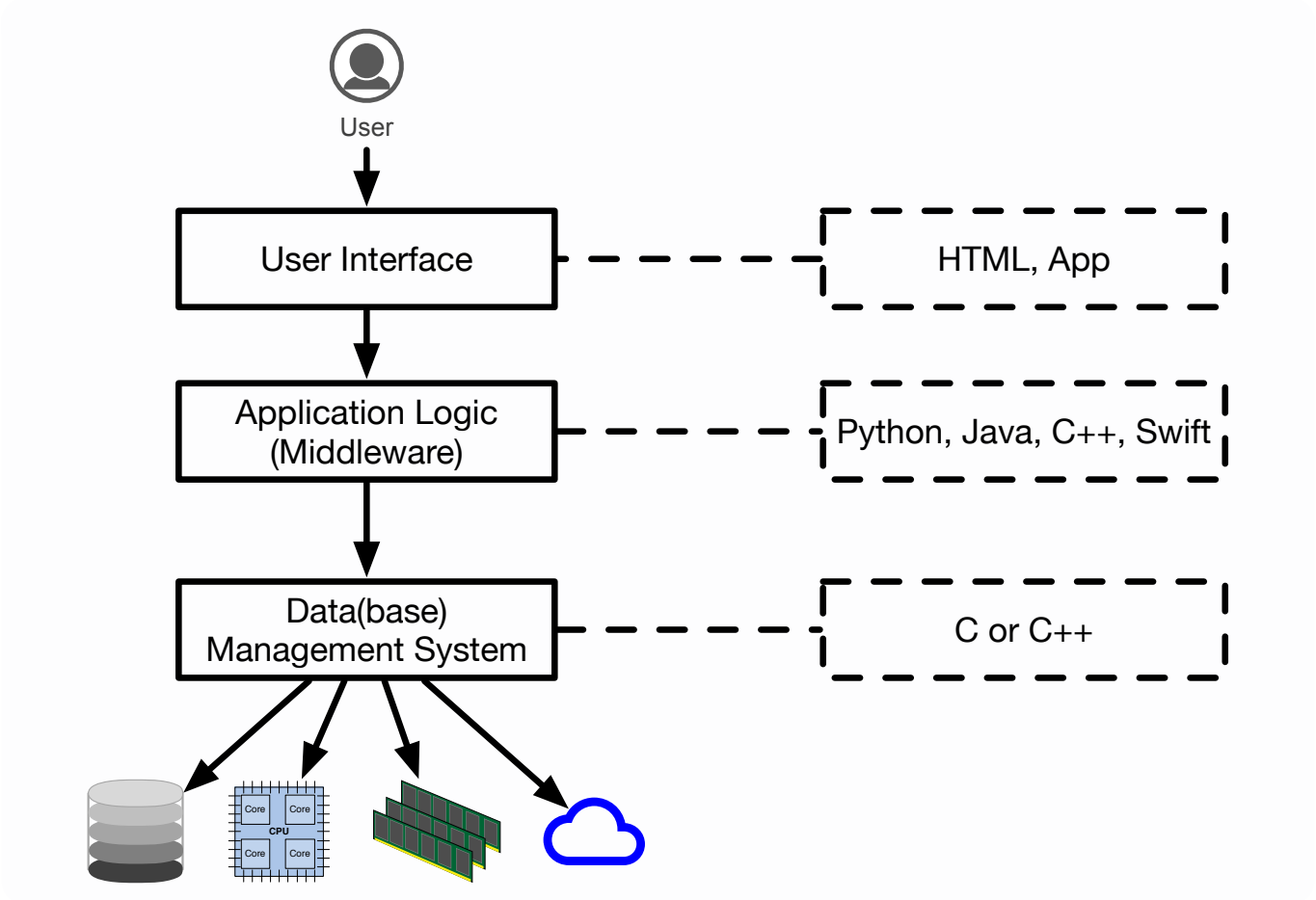
In Context

- Often made up from **components**,
  - that interact
  - to achieve a **greater goal**
- Usually applicable to many situations (i.e., generic)

Data Management System Components/Functionality

- Storage
- Data Ingestions
- Concurrency
- Data Analysis
- Standardized Programming Model
- User-Defined Functions
- Access Control
- Self-optimization
- (and counting)

Data-intensive application architecture



Data management systems are a big market

- We are talking about a 50 billion dollar market (in 2017)
  - That is only the pure sales volume of relational DBMSs
  - add administration, tuning, application development, ...
- Fun fact: SQLite is the second most-used piece software on the planet (the first is zlib)

Data-intensive Applications vs. Management Systems?

- The boundary is blurry
- Applications
  - Not generic
    - Domain-specific
    - Hard to generalize
    - Often contain domain-specific optimizations and restrictions
- Experience shows that the cost of application-specific data management outweighs the benefits in most cases

Data-intensive Applications vs. Management Systems?

Here is a proposed spectrum (from most to least "systemsy")

- Yelp
- A mobile app for geo-services
- A library to manage unordered collections of tagged coordinates
- A spatial data management library
- A relational database
- A block storage system

Non-functional requirements

Efficiency

Data Management Systems should not be (significantly) slower than hand-written applications

Resilience

Data Management Systems should recover from problems (power outage, hardware failure, software crashes, . . .)

Robustness

Data Management Systems should have predicatable performance: small changes in a query should not cause major changes in performance

Non-functional requirements

Scalability

Data Management Systems should make efficient use of the available resources. An increase of resources should cause an improvement of performance.

Concurrency

Data Management Systems should serve multiple simultaneous clients transparently (results should not be impacted).

And they actually do all that

But, if database do such a great job...

. . . why would I care about internals?

Why would I care about internals?

Why this class?

- If you ever happen to work on a DBMS (not very likely)
- To make you a kick-ass DBA (somewhat more likely)
- To apply data management techniques outside the field (extremely likely)
  - Bragging rights if you can implement a radix-partitioned in-memory hash-join
  - Some of this actually comes up if you interview at Google, Facebook, Microsoft, etc.
- If you would like to work with me

Plus:

Performance: Physical & logical Data Model Separation

- Users
  - don't care about file format
  - don't care (much) about storage devices
  - don't care (much) about portability
  - often send data in a fire and forget manner
- DBMSs
  - separate external from internal model
  - exploit degrees of freedom for performance

Transparent concurrency: Transactional Semantics

Isolated	Run like you were alone on the system
Atomic	Run completely or not at all
Consistent	The interesting thing here is that there may be inconsistency in between
Durable	After the transaction commits, even power outage won't undo the transaction

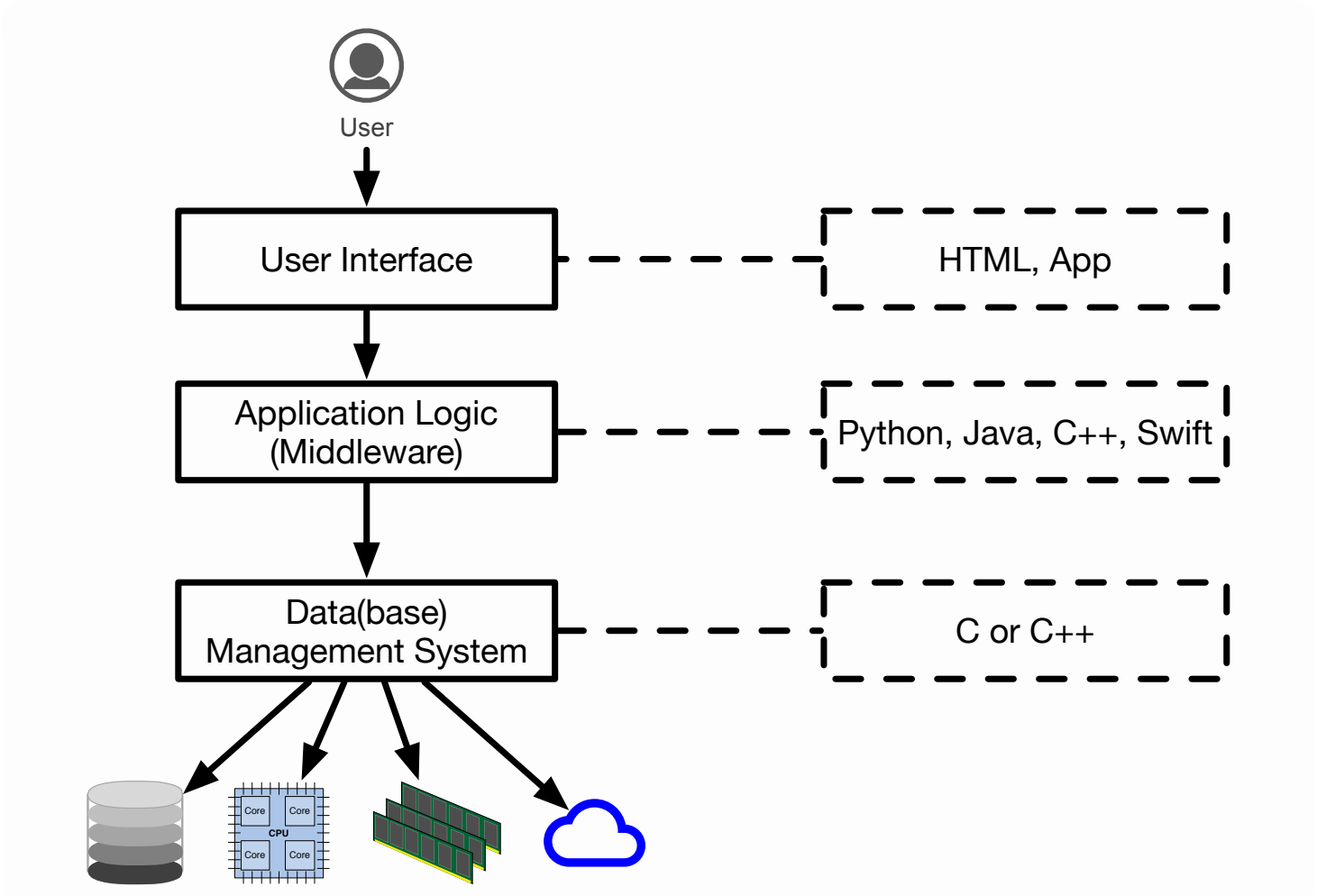
Ease of use: Declarative Data analysis

- Every so often, I want to retrieve information about my data, e.g.,
  - retrieve a single tuple
  - calculate some statistic like "what is the total sales volume for each country we do business in" right now
  - generate a detailed report of all our business characteristics (sales, hiring, procurement, . . .) over night
  - train a model to predict future data
- Just describe your result, the system will generate it

But these seem natural! This is obviously how you do it!

Well. . .

Recall the architecture



Technically, this is also where the filesystem sits!

Are data management systems filesystems?

Let's discuss this in the Q&A!

What a Database Management System is not?

- A filesystem
- A runtime for your applications (though people have tried)
  - some support user-defined functions
  - some even have embedded webservers, middleware, ...
    - a horrible idea
- A place to store intermediate state

Why are databases a bad place for intermediate state?

Let's discuss this in the Q&A!

What will this course look like

This course...

- ...is an advanced databases class
- ...is concerned with the internals and implementation of database systems
- ...contains useful knowledge whenever you have a data-intensive application
- ...will contain a fair number of coding examples (brush up you C++)
- ...is not an introduction course
- ...is not concerned with the use of data management systems

Things we will not cover (properly)

- Data Models
- Entity Relationship Modeling
- Schema Design
- SQL
- Database Management System (DBMS) Usage Models
  - OLTP, OLAP, Reporting, ...
- Map/Reduce as a programming model

Some Admin stuff

- Course is taught in two halves
  - First halve on single-node databases (taught by your's truly)
  - Second halve on distributed databases (taught by Peter McBrien)
- Register for this course (at level 2)!!!
  - Bad things will happen if you don't
  - Ask your cohort admin person how to do it

My Plan for this course

Plans are useless but planning is indispensable – Dwight Eisenhower

My Plan for the first half of this course

A "proper" flipped classroom approach

- I will release **a video every week on Monday** (might be earlier)
  - roughly one two-hour lecture block in length (ca. 100 minutes)
- We will have an **interactive session on Thursday**
  - I expect that you have watched the video by that time
  - We will have **50 minutes of Q&A**
  - and **50 minutes of tutorial**

Coursework

- There will be two assignments
  - Indexing & Query Processing in pairs of two (with me)
  - This will include our **programming competition** (sponsored by Snowflake)
  - Distributed data processing (with Peter McBrien)
- Exam in the end
  - Relevant is what is discussed in class

Things I assume you know

- CO 130, i.e., knowledge about the use of a database
- CO 120, knowledge about algorithms & data structures
  - Arrays, linked lists, trees, heaps, ...
  - Sorting, hashing, binary searching, graph/tree-traversal
- CO 112 & 113, i.e., knowledge about the workings of hardware components
  - Main memory, CPUs, disks, caches, multicore, ...
- CO 275, i.e., Programming C++
  - Programming for one of the two coursework assignments
- Honesty when preparing coursework
  - Don't try me!

Books

Fundamentals of Database Systems

Ramez Elmasri, Shamkant Navathe
Sixth edition, Pearson new international edition., Pearson,

Database Systems: The Complete Book

Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
2nd ed., Pearson Education

Database Systems: Practical Approach to Design, Implementation, and Management

Thomas M. Connolly, Carolyn E. Begg
Sixth edition, Global edition., Pearson Education Limited

Communication

- Zoom for remote participation in interactive sessions
- `https://co572.pages.doc.ic.ac.uk`
- Edstem for online discussion

Remember

- Register for this course
  - at Level 2!!!

Before we start

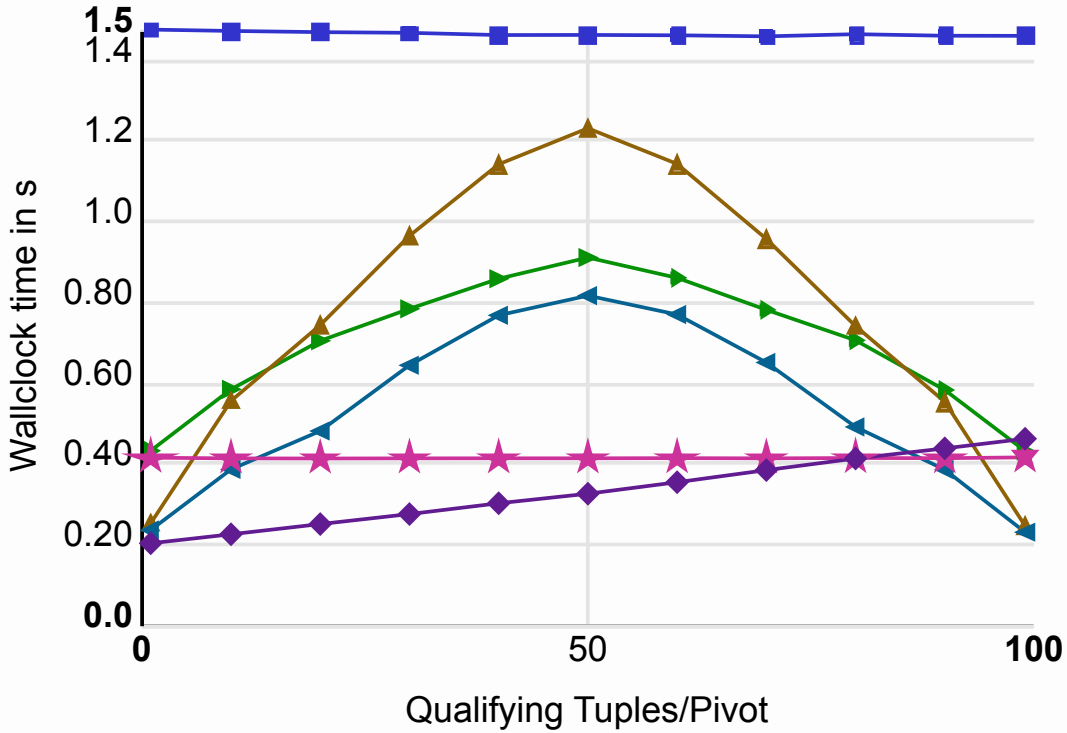
A note on performance

A note on performance

- Much of this course is concerned with performance: interpretation, prediction and optimization
- Why?
  - Efficiency is important in practice (think energy, money, time)
  - **Understanding performance** is even more important (you are here to learn)

Database performance internals: an example

What Resource bounds this query's performance



Database internals: an example

Option A

CPU!

Option B

Memory!

Option C

Disk!

Option D

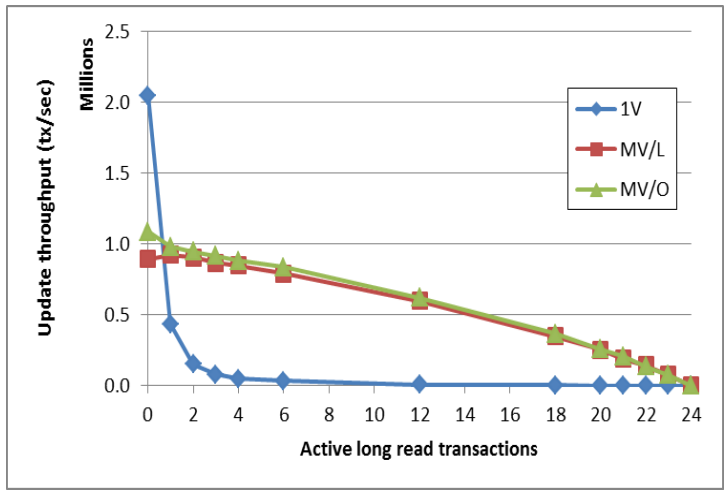
What?

Let's discuss this in the Q&A

Database internals: another example

What is this Workload's Isolation Level

Effect of long read-only transactions



Workload:

- Short txns 10R+ 2W
- Long txns: R 10% of rows

24 threads in total

- X threads running short tx
- 24-X threads running long txns

Database internals: another example

Option A

Repeatable Read!

Option C

Read Committed!

Option B

Serializable!

Option D

I Have no idea!

I have no idea :-)

Takeaways

- **Claim:** If you can explain (and predict) its performance, you have understood a system
- You (probably) know how to use a database system
- You (probably) don't know everything about how it works
- I certainly don't
- But I know a lot of people who do – **ASK!!!**
- If nobody knows, it is research (see me!)

Enough banter! Let's get started!

Preliminaries: Vectors, tuples, bags & sets

Vector	an ordered collection of objects of the same type
Tuple	an ordered collection of objects of different type
Bag	an unordered collection of objects of the same type
Set	an unordered collection of unique objects of the same type

Preliminaries: Schemas

The definition of the attributes of the tuples in your relations (duh)

```
create table review (stars int, comment varchar(1024), user int);
```

But also integrity contraints (uniqueness, keys, foreign-keys)

```
alter table review add foreign key(user) references user(name);
```

Some people distinguish external and internal schema

- The idea of internal schemas is misleadingly simplistic
  - there may not even be a schema

The Purpose of Relational Algebra

A logical representation of relational operations

- Used to define the semantics of operations
- Used for logical optimization
- Not actually all that useful for end-users
- Not actually how (most) systems evaluate queries

Relations

From the Codd paper

An array which represents an n-ary relation R has the following properties:

(1) Each row represents an n-tuple of R.
(2) The ordering of rows is immaterial.
(3) All rows are distinct.
(4) The ordering of columns is significant—it corresponds to the ordering $S_1, S_2, \cdots, S_n$ of the domains on which R is defined (see, however, remarks below on domain-ordered and domain-unordered relations).
(5) The significance of each column is partially conveyed by labeling it with the name of the corresponding domain.

Relations are almost sets of tuples

Let's get coding: Implementing Relations

A first shot

```
template <typename... types> //
struct Relation {
  set<tuple<types...>> data;
  array<string, sizeof...(types)> schema;

  Relation(array<string, sizeof...(types)> schema, set<tuple<types...>> data)
      : schema(schema), data(data) {}
};
```

Implementing Relations

Creating a relation

```
auto createCustomerTable() {
  Relation<int, string, string> customer( //
      {"ID", "Name", "ShippingAddress"},  //
      {{1, "holger", "180 Queens Gate"},
       {2, "Sam", "32 Vassar Street"},
       {3, "Peter", "180 Queens Gate"}});
  return customer;
}
```

Making relations composable

Exposing type information

```
template <typename... types> //
struct Relation {
  using OutputType = tuple<types...> ;
  set<tuple<types...>> data;
  array<string, sizeof...(types)> schema;
  Relation(){};
  Relation(array<string, sizeof...(types)> schema, set<tuple<types...>> data)
      : schema(schema), data(data) {}
};
```

Relational Expressions

Nomenclatures

- An **expression** in relational algebra is composed from **operators**
- I will often refer to an expression as a **(logical) plan**
- **Cardinality** is the number of tuples in a set

Handy properties

- Set-based
  - Order-invariant and duplicate eliminated
- Relational algebra is closed
  - Every operator produces a relation as output
  - Every operator accepts one or two relations as input
  - This simplifies the composition of operators into expressions
    - Note, that expressions can still be invalid

Relational Operators

Implementation

```
template <typename... types> struct Operator : public Relation<types...> {};
```

Relational Operators

A minimal set

- Project
- Select
- Cross (Carthesian) Product
- Union
- Difference

Not included

- Intersection

Project π

Intuitive semantics

- Extract one or multiple attributes from a relation
- Preserve relational semantics
- **Changes the schema**

Customer

ID	Name	ShippingAddress
1	Holger	180 Queens Gate
2	Sam	32 Vassar Street
3	Peter	180 Queens Gate

$\pi_{Name} Customer$

Name
Holger
Sam
Peter

$\pi_{ShippingAddress} Customer$

ShippingAddress
180 Queens Gate
32 Vassar Street

Project π

Quick Quiz: What is the cardinality of the output of a projection

- It can only be determined by evaluating it
- Cardinality of the input
- I don't know

Quick Quiz: What is the upper bound for the cardinality of the output of a project

- It can only be determined by evaluating it
- Cardinality of the input
- I don't know

Project π

Implementation

```
template <typename InputOperator, typename... types>
struct Project : public Operator<types...> {
        InputOperator input;
        set<pair<string, string>> projections; // attribute mappings
        Project(InputOperator input, set<pair<string, string>> projections)
                        : input(input), projections(projections) {}
};
```

Example

```
void example1() {
  auto customer = createCustomerTable();
  auto p = Project<decltype(customer), string>(customer, {{"Name", "customerName"}});
}
```

Project π, with functions

Intuitive semantics

- Extract one or multiple attributes from a relation and perform a scalar operation on them

Implementation

```
#include <variant>
template <typename InputOperator, typename... outputTypes>
struct Project : public Operator<outputTypes...> {
  InputOperator input;

  function<tuple<outputTypes...>(typename InputOperator::OutputType)> projectionFunction;
  Project(InputOperator input,
          function<tuple<outputTypes...>(typename InputOperator::OutputType)> //
              projectionFunction)
      : input(input), projectionFunction(projectionFunction) {}
};
```

Project π, with functions

Example

```
void example2() {
  auto customer = createCustomerTable();
  auto p = Project<decltype(customer), string>(customer, //
                                                []( auto input) { return get<1>(input); });
}
```

Project π, for real

Implementation

```
#include <variant>
template <typename InputOperator, typename... outputTypes>
struct Project : public Operator<outputTypes...> {
  InputOperator input;

  variant<function<tuple<outputTypes...>(typename InputOperator::OutputType)>,
          set<pair<string, string>>>
      projections; // attribute mappings
  Project(InputOperator input,
          function<tuple<outputTypes...>(typename InputOperator::OutputType)> projections)
    : input(input), projections(projections) {}

  Project(InputOperator input, set<pair<string, string>> projections)
    : input(input), projections(projections) {}
};
```

Project π, for real

Example

```
void example3() {
  auto customer = createCustomerTable();
  auto p1 = Project<decltype(customer), string>(customer, //
                                                []( auto input) { return get<1>(input); });
  auto p2 = Project<decltype(customer), string>(customer, {{"Name", "customerName"}});
}
```

Select σ

Intuitive Semantics

- Produce a new relation containing input tuples that satisfy a condition
- Does not change the schema
- Changes cardinality (i.e., the number of tuples in a relation)

OrderedItem

OrderId	BookID
1	1
1	2
2	1
3	3

$\sigma_{BookID=1} OrderedItem$

OrderId	BookID
1	1
2	1

Select σ

Quick Quiz: What is the cardinality of the output of a selection

- It can only be determined by evaluating it
- Cardinality of the input
- I don't know

Quick Quiz: What is the upper bound for the cardinality of the output of a select

- It can only be determined by evaluating it
- Cardinality of the input
- I don't know

Select σ

Implementation

```
enum class Comparator { less, lessEqual, equal, greaterEqual, greater };

struct Column {
  string name;
  Column(string name) : name(name) {}
};
using Value = variant<string, int, float>;

struct Condition {
  Column leftHandSide;
  Comparator compare;
  variant<Column, Value> rightHandSide;
  Condition(Column leftHandSide, Comparator compare, variant<Column, Value> rightHandSide)
      : leftHandSide(leftHandSide), compare(compare), rightHandSide(rightHandSide) {}
};
```

Select σ

Example

```
void example4() {
  auto c2c = Condition(Column("Name"), Comparator::equal, Column("ShippingAddress"));
  auto c2v = Condition(Column("Name"), Comparator::equal, Value("ShippingAddress"));
}
```

Select σ

Implementation

```
template <typename InputOperator>
struct Select : public Operator<typename InputOperator::OutputType> {
        InputOperator input;
        Condition condition;
        Select(InputOperator input, Condition condition)
                        : input(input), condition(condition){};
};
```

Select σ

Example

```
void example5() {
  auto customer = createCustomerTable();
  auto c2v = Condition(Column("Name"), Comparator::equal, Value("Holger"));
  auto p1 = Select<decltype(customer)>(customer, c2v);
}
```

Cross (Carthesian) Product $\times$

Intuitive Semantics

- **Takes two inputs**
- Produce a new relation by combining every tuple from the left with every tuple from the right
- **Changes the schema**

Cross (Carthesian) Product $\times$

Order

ID	CustomerID
1	1
2	2
3	3

OrderedItem

OrderId	BookID
1	1
1	2
2	1
3	3

$Order \times OrderedItem$

ID	CustomerID	OrderId	BookID
1	1	1	1
1	1	1	2
1	1	2	1
1	1	3	3
2	2	1	1
2	2	1	2
2	2	2	1
2	2	3	3
3	3	1	1
3	3	1	2
3	3	2	1
3	3	3	3

Cross (Carthesian) Product $\times$

Implementation

```
template <typename LeftInputOperator, typename RightInputOperator>
struct CrossProduct : public Operator<Concat<typename LeftInputOperator::OutputType,
                                             typename RightInputOperator::OutputType>> {
  LeftInputOperator leftInput;
  RightInputOperator rightInput;
  CrossProduct(LeftInputOperator leftInput, RightInputOperator rightInput)
      : leftInput(leftInput), rightInput(rightInput){};
};
```

Cross (Carthesian) Product $\times$

Quick Quiz: What is the cardinality of the output of a cross product

- It can only be determined by evaluating it
- Cardinality of left input plus cardinality of right input
- Cardinality of left input times cardinality of right input
- I don't know

Composing operators

Rules

- Since relational algebra is closed, operators can be combined as long as their signature is respected
  - Cross products take two inputs
  - Selections and Projections take one

Example

$\pi_{BookID}(\sigma_{Order.ID==OrderedItem.OrderID}(Order \times OrderedItem))$

Composing operators

Implementation

```
int main() {
  Relation<int, int> order({"ID", "CustomerID"}, {{1, 1}, {2, 2}, {3, 3}});
  Relation<int, int> orderedItem({"OrderID", "BookID"}, {{1, 1}, {1, 2}, {2, 1}, {3, 3}});
  auto plan =                                                    //
      Project(                                                   //
          Select(                                                //
              CrossProduct(order, orderedItem),                  //
              Condition(Column("Order.ID"), Comparator::equal,   //
                        Column("OrderedItem.OrderID"))),         //
          {{"OrderedItem.BookID", "OrderedItem.BookID"}});
}
```

Union $\cup$

Intuitive Semantics

- Produce a new relation from two relations containing any tuple that is present in one of the inputs
- Does not change the schema
  - Requires input schema compatibility
- Changes cardinality (i.e., the number of tuples in a relation)

$\sigma_{Name=Holger}Customer$

ID	Name	ShippingAddress
1	Holger	180 Queens Gate

$\sigma_{Name=Sam}Customer$

ID	Name	ShippingAddress
2	Sam	32 Vassar Street

$\sigma_{Name=Sam}Customer \cup \sigma_{Name=Holger}Customer$

ID	Name	ShippingAddress
1	Holger	180 Queens Gate
2	Sam	32 Vassar Street

Union ∪

Implementation

```
template <typename LeftInputOperator, typename RightInputOperator>
struct Union : public Operator<typename LeftInputOperator::outputType> {
        LeftInputOperator leftInput;
        RightInputOperator rightInput;

        Union(LeftInputOperator leftInput, RightInputOperator rightInput)
                        : leftInput(leftInput), rightInput(rightInput){};
};
```

Can we ensure schema compatibility here?

Quick quiz: Union $\cup$

Quick Quiz: What is the cardinality of the output of a union

- It can only be determined by evaluating it
- Cardinality of left input plus cardinality of right input
- Cardinality of left input times cardinality of right input
- I don't know

Quick Quiz: What is the upper bound for the cardinality of a union

- It can only be determined by evaluating it
- Cardinality of left input plus cardinality of right input
- Cardinality of left input times cardinality of right input
- I don't know

Difference –

Intuitive Semantics

- Produce a new relation from two relations containing any tuple that is present in the first but not the second input
- Does not change the schema
  - Requires input schema compatibility
- Changes cardinality (i.e., the number of tuples in a relation)

$Customer$

ID	Name	ShippingAddress
1	Holger	180 Queens Gate
2	Sam	32 Vassar Street
3	Peter	180 Queens Gate

$\sigma_{Name=Sam}Customer$

ID	Name	ShippingAddress
2	Sam	32 Vassar Street

$Customer - \sigma_{Name=Sam}Customer$

ID	Name	ShippingAddress
1	Holger	180 Queens Gate
3	Peter	180 Queens Gate

Difference –

Implementation

```
template <typename LeftInputOperator, typename RightInputOperator>
struct Difference : public Operator<typename LeftInputOperator::outputType> {
        LeftInputOperator leftInput;
        RightInputOperator rightInput;
        Difference(LeftInputOperator leftInput, RightInputOperator rightInput)
                        : leftInput(leftInput), rightInput(rightInput){};
};
```

Can we ensure schema compatibility here?

Grouped Aggregation Γ

Intuitive Semantics

- Produce a new relation from one input by grouping together tuples that have equal values in some attributes and aggregate others
  - The groups are defined by the set of *grouping attributes*
    - This set can be empty
  - The aggregates are defined by the set of *aggregations*, i.e., triples of
    - the *input attribute*
    - the *aggregation function*: min, max, avg, sum, count
    - the *output attribute*
- Changes schema & cardinality

$Customer$

ID	Name	ShippingAddress
1	Holger	180 Queens Gate
2	Sam	32 Vassar Street
3	Peter	180 Queens Gate

$\Gamma_{((ShippingAddress),((ID,count,c)))}Customer$

ShippingAddress	c
32 Vassar Street	1
180 Queens Gate	2

Grouped Aggregation Γ

Implementation

```
enum class AggregationFunction { min, max, sum, avg, count };

template <typename InputOperator, typename... Output>
struct GroupedAggregation : public Operator<Output...> {
  InputOperator input;
  set<string> groupAttributes;
  set<tuple<string, AggregationFunction, string>> aggregations;
  GroupedAggregation(InputOperator input, set<string> groupAttributes,
                     set<tuple<string, AggregationFunction, string>> aggregations)
      : input(input), groupAttributes(groupAttributes), aggregations(aggregations){};
};
```

Grouped Aggregation Γ

Example

```
void example6() {
  auto customer = createCustomerTable();
  GroupedAggregation<decltype(customer), int>(customer, {"ShippingAddress"},
                                              {{"ID", AggregationFunction::count, "c"}});
}
```

Top-N T

Intuitive Semantics

- Produce a new relation from one input selecting the tuples with the N greatest (w.l.o.g.) values with respect to an attribute
  - The *top-n predicate* is a single attribute
- Changes cardinality, maintains schema

$Customer$

ID	Name	ShippingAddress
1	Holger	180 Queens Gate
2	Sam	32 Vassar Street
3	Peter	180 Queens Gate

$T_{(2,ID)}Customer$

ID	Name	ShippingAddress
2	Sam	32 Vassar Street
3	Peter	180 Queens Gate

Top-N T

Implementation

```
template <typename InputOperator>
struct TopN : public Operator<typename InputOperator::OutputType> {
  InputOperator input;
  size_t N;
  string predicate;
  TopN(InputOperator input, size_t N, string predicate)
      : input(input), N(N), predicate(predicate){};
};
```

Top-N T

Example

```
void example7() {
  auto customer = createCustomerTable();
  TopN<decltype(customer)>(customer, 2, "id");
}
```

Thank You!

Provide feedback, please!

https://co572.pages.doc.ic.ac.uk/feedback/introduction

Get the slides online

https://co572.pages.doc.ic.ac.uk/decks/Introduction.pdf