

Course overview and background

Lecture 1
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

22 April 2015

Welcome to the Database Technology Course!

Today

- ▶ Introduction to database systems
- ▶ Syllabus
- ▶ Review of relational data model and its query languages

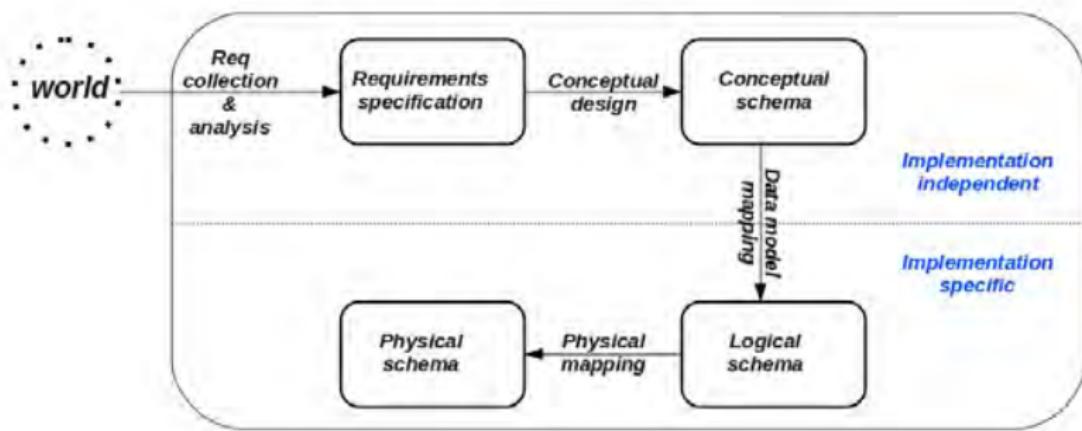
Course overview

- ▶ general topic: Big data; engineering for of data science
- ▶ Data is everywhere, outlasting code (which is also data
...)

Course overview

- ▶ **general topic:** Big data; engineering for of data science
- ▶ Data is everywhere, outlasting code (which is also data ...)
- ▶ **our focus:** core technologies behind principled management of massive collections of data
- ▶ Database management systems (DBMS) as microcosm of CS

Course overview



DB design process

Course overview

“Goodness” in DB design

- ▶ **Conceptual.** Accurately reflect the semantics of use in the modeled domain.

Course overview

“Goodness” in DB design

- ▶ **Conceptual.** Accurately reflect the semantics of use in the modeled domain.
- ▶ **Logical.** Accurately reflect conceptual model and disallow redundancies and update anomalies, as best possible.

Course overview

“Goodness” in DB design

- ▶ **Conceptual.** Accurately reflect the semantics of use in the modeled domain.
- ▶ **Logical.** Accurately reflect conceptual model and disallow redundancies and update anomalies, as best possible.
- ▶ **Physical.** Accurately reflect logical model and efficiently and reliably support use of data.
 - ▶ *Our focus in this course*

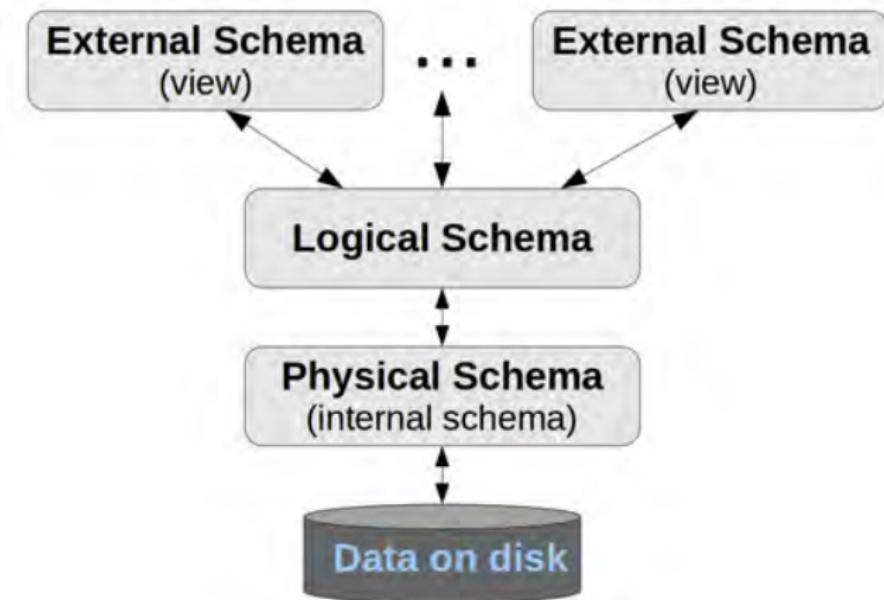
Course overview

- ▶ Why not filesystem and/or virtual memory (i.e., use the OS)?

Course overview

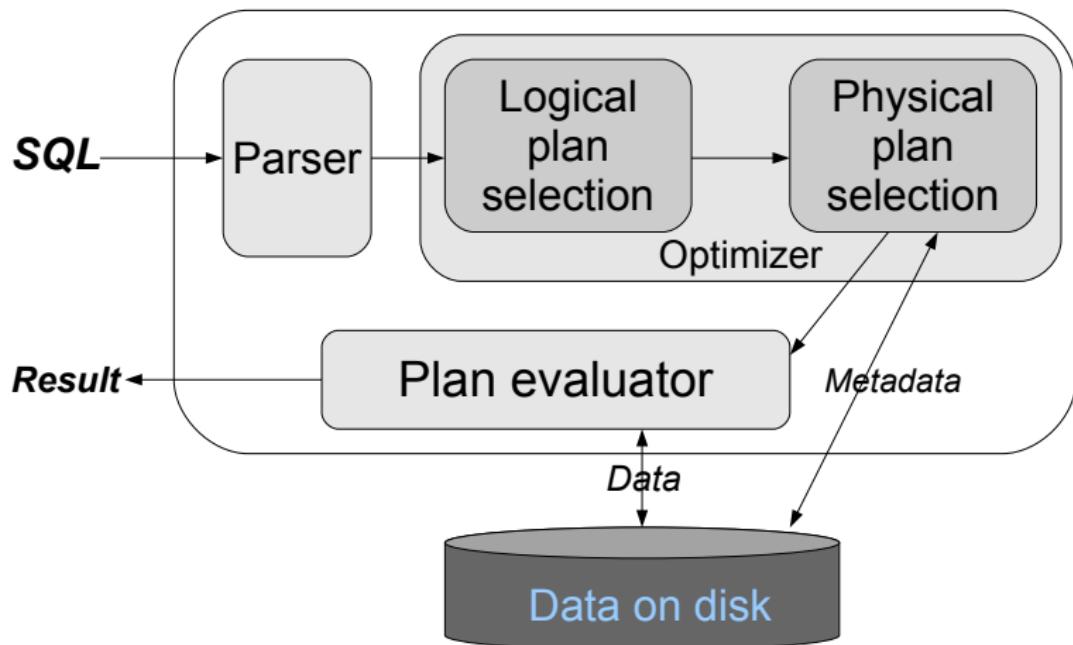
- ▶ Why not filesystem and/or virtual memory (i.e., use the OS)?
- ▶ Key idea: data independence
 - ▶ insulation from changes in the way data is structured and stored
- ▶ physical vs. logical data independence

Course overview



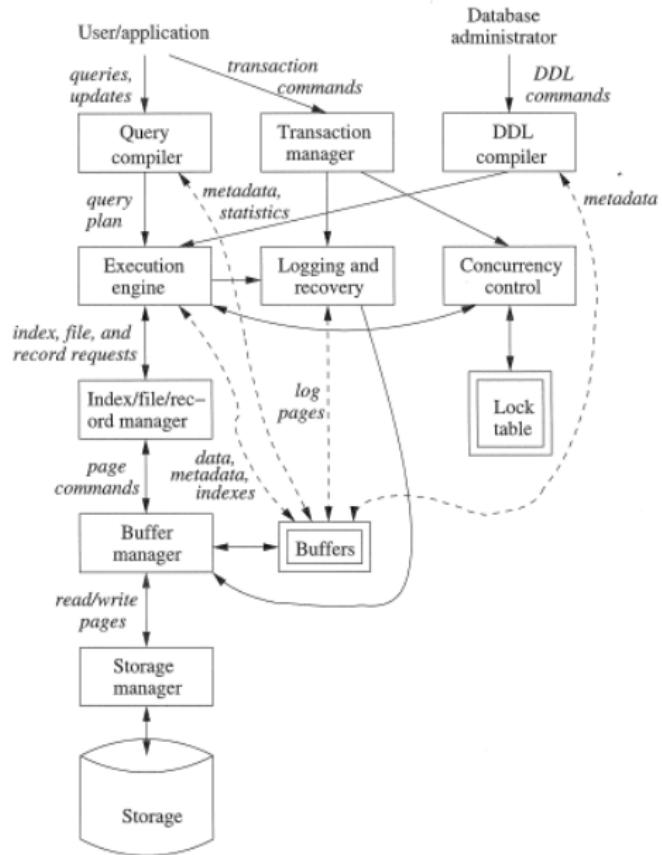
Layers of Abstraction

Course overview



The life of a query

Course overview



Admin: staff

Staff

- ▶ dr. George Fletcher
 - web:* www.win.tue.nl/~gfletche/
 - office:* MF7.063

Admin: website

Course website

Google “George Fletcher TUE teaching”

or, go to

<http://wwwis.win.tue.nl/2ID35/>

Admin: textbook

Textbook

A. Silberschatz, H. Korth, and S. Sudarshan. *Database System Concepts*, 6th Edition, McGraw-Hill, 2011.

Admin: responsibilities

Student responsibilities

- ▶ One multi-phase team project,
- ▶ One written individual assignment, and
- ▶ One final individual exam.

Admin: responsibilities

Student responsibilities

- ▶ One multi-phase team project,
- ▶ One written individual assignment, and
- ▶ One final individual exam.

Grading criteria

- ▶ 50% project
- ▶ 10% written assignment(s)
- ▶ 40% final exam

Admin: responsibilities

Student responsibilities

- ▶ One multi-phase team project,
- ▶ One written individual assignment, and
- ▶ One final individual exam.

Grading criteria

- ▶ 50% project
- ▶ 10% written assignment(s)
- ▶ 40% final exam

Note: Late work will lose $\frac{1}{3}$ of its original point-value for each day it is late, and once solutions are posted or discussed, late submissions will not be accepted. Full details and other policies can be found in the syllabus, on the course webpage.

Admin: Course project

Carefully study a recent DB research paper

- ▶ As a team, implement main idea and repeat experiments from the paper
- ▶ Goal: validation (or refutation) and extension (and/or correction) of claimed results

Admin: Course project

Carefully study a recent DB research paper

- ▶ As a team, implement main idea and repeat experiments from the paper
- ▶ Goal: validation (or refutation) and extension (and/or correction) of claimed results
- ▶ **Your first assignment:** go to the course webpage, and read details (first item due before the end of this Sunday, 26 April)

Admin: Syllabus

- ▶ (22 April) Course introduction and background review (relational model, data independence).
- ▶ (24 April) Storage, the I/O computational model, & external sorting
 - ▶ Sunday 26 April: project part 1 due (paper selection)

Admin: Syllabus

- ▶ (22 April) Course introduction and background review (relational model, data independence).
- ▶ (24 April) Storage, the I/O computational model, & external sorting
 - ▶ Sunday 26 April: project part 1 due (paper selection)
- ▶ (29 April) Indexing: B-trees, R-trees, and GiST
- ▶ (1 May) Indexing: hashing and evaluation of indexes

Admin: Syllabus

- ▶ (22 April) Course introduction and background review (relational model, data independence).
- ▶ (24 April) Storage, the I/O computational model, & external sorting
 - ▶ Sunday 26 April: project part 1 due (paper selection)
- ▶ (29 April) Indexing: B-trees, R-trees, and GiST
- ▶ (1 May) Indexing: hashing and evaluation of indexes
- ▶ (6 May) Query processing
- ▶ (8 May) Data statistics & Answering queries using views
 - ▶ Sunday 10 May: project part 2 due (first report)

Admin: Syllabus

- ▶ (13 May) Query optimization
 - ▶ written assignment posted
- ▶ (15 May) *Hemelvaart – no lecture*

Admin: Syllabus

- ▶ (13 May) Query optimization
 - ▶ written assignment posted
- ▶ (15 May) *Hemelvaart – no lecture*
- ▶ (20 May) *no lecture – work on team project*
- ▶ (22 May) Distributed query processing
 - ▶ Sunday 24 May: project part 3 due (second report)

Admin: Syllabus

- ▶ (13 May) Query optimization
 - ▶ written assignment posted
- ▶ (15 May) *Hemelvaart – no lecture*
- ▶ (20 May) *no lecture – work on team project*
- ▶ (22 May) Distributed query processing
 - ▶ Sunday 24 May: project part 3 due (second report)
- ▶ (27 May) *no lecture – team project meetings with instructor*
- ▶ (29 May) *no lecture – team project meetings with instructor*

Admin: Syllabus

- ▶ (3 June) Transaction management
- ▶ (5 June) Graph and tree (XML) query processing

Admin: Syllabus

- ▶ (3 June) Transaction management
- ▶ (5 June) Graph and tree (XML) query processing
- ▶ (10 June) Course summary and exam review
- ▶ (12 June) Project presentations I
 - ▶ in class: written assignment due

Admin: Syllabus

- ▶ (3 June) Transaction management
- ▶ (5 June) Graph and tree (XML) query processing
- ▶ (10 June) Course summary and exam review
- ▶ (12 June) Project presentations I
 - ▶ in class: written assignment due
- ▶ (17 June) Project presentations II
- ▶ (19 June) Project presentations III
 - ▶ Sunday 21 June: project part 4 (final submission) due

Admin: Syllabus

- ▶ (3 June) Transaction management
- ▶ (5 June) Graph and tree (XML) query processing
- ▶ (10 June) Course summary and exam review
- ▶ (12 June) Project presentations I
 - ▶ in class: written assignment due
- ▶ (17 June) Project presentations II
- ▶ (19 June) Project presentations III
 - ▶ Sunday 21 June: project part 4 (final submission) due
- ▶ (26 June, 13:30-16:30) Final exam

After the break ...

Quick tutorial on what we are actually implementing this quarter:

- ▶ Review of the relational model
- ▶ Review of relational algebra, relational calculus, datalog, and SQL

But first a 5 minute paper

But first a 5 minute paper

What are the top 2 things you hope to take away from this course?

a quick refresher on first order logic

First order logic:

$\exists, \forall, \neg, \wedge, \vee, R(\bar{x}), x = y, \varphi \rightarrow \psi$

The query languages we will consider are fragments of first order logic (FO)

that is, the queries (i.e., computable mappings from database instances to database instances) expressible in these languages are equivalently expressible as formulas in FO

we next review some basics of FO

FO review: data

Fix some universe U of atomic values.

- ▶ A **relation schema** consists of a name R and finite set of attribute names $\text{attributes}(R) = \{A_1, \dots, A_k\}$. The arity of R is $\text{arity}(R) = k$.

FO review: data

Fix some universe U of atomic values.

- ▶ A **relation schema** consists of a name R and finite set of attribute names $\text{attributes}(R) = \{A_1, \dots, A_k\}$. The arity of R is $\text{arity}(R) = k$.
- ▶ A **fact** over relation schema R of arity k is a term of the form $R(a_1, \dots, a_k)$, where $a_1, \dots, a_k \in U$.
 - ▶ alternatively, a **tuple** over R is a function from $\text{attributes}(R)$ to U .

FO review: data

Fix some universe U of atomic values.

- ▶ A **relation schema** consists of a name R and finite set of attribute names $\text{attributes}(R) = \{A_1, \dots, A_k\}$. The arity of R is $\text{arity}(R) = k$.
- ▶ A **fact** over relation schema R of arity k is a term of the form $R(a_1, \dots, a_k)$, where $a_1, \dots, a_k \in U$.
 - ▶ alternatively, a **tuple** over R is a function from $\text{attributes}(R)$ to U .
- ▶ An **instance** of relation schema R is a finite set of facts/tuples over R .

FO review: data

Fix some universe U of atomic values.

- ▶ A **relation schema** consists of a name R and finite set of attribute names $\text{attributes}(R) = \{A_1, \dots, A_k\}$. The arity of R is $\text{arity}(R) = k$.
- ▶ A **fact** over relation schema R of arity k is a term of the form $R(a_1, \dots, a_k)$, where $a_1, \dots, a_k \in U$.
 - ▶ alternatively, a **tuple** over R is a function from $\text{attributes}(R)$ to U .
- ▶ An **instance** of relation schema R is a finite set of facts/tuples over R .
- ▶ A **database schema** (aka “signature”) is a finite set D of relation schemas.

FO review: data

Fix some universe U of atomic values.

- ▶ A **relation schema** consists of a name R and finite set of attribute names $\text{attributes}(R) = \{A_1, \dots, A_k\}$. The arity of R is $\text{arity}(R) = k$.
- ▶ A **fact** over relation schema R of arity k is a term of the form $R(a_1, \dots, a_k)$, where $a_1, \dots, a_k \in U$.
 - ▶ alternatively, a **tuple** over R is a function from $\text{attributes}(R)$ to U .
- ▶ An **instance** of relation schema R is a finite set of facts/tuples over R .
- ▶ A **database schema** (aka “signature”) is a finite set D of relation schemas.
- ▶ An **instance** of database schema D is a set of relation instances, one for each $R \in D$.

FO review: syntax

Fix a database schema D and let $R_1, \dots, R_m \in D$.

SQL

```
SELECT A1, ..., Ak  
FROM R1, ..., Rm  
WHERE Cond
```

where Cond is a well-formed selection condition over \mathcal{A} , and $A_1, \dots, A_k \in \mathcal{A}$, for $\mathcal{A} = \bigcup_{R \in \{R_1, \dots, R_m\}} \text{attributes}(R)$.

FO review: syntax

Fix a database schema D and let $R_1, \dots, R_m \in D$.

SQL

```
SELECT A1, ..., Ak  
FROM R1, ..., Rm  
WHERE Cond
```

where Cond is a well-formed selection condition over \mathcal{A} , and $A_1, \dots, A_k \in \mathcal{A}$, for $\mathcal{A} = \bigcup_{R \in \{R_1, \dots, R_m\}} \text{attributes}(R)$.

Relational Algebra (RA)

$$\pi_{A_1, \dots, A_k}(\sigma_{Cond}(R_1 \times \dots \times R_m))$$

FO review: syntax

Datalog

$$\text{result}(A_1, \dots, A_k) \leftarrow R_1(\overline{A^1}), \dots, R_m(\overline{A^m}), C_1, \dots, C_j$$

where

- ▶ $\overline{A^i}$ is a list of variables of length $\text{arity}(S_i)$ (for $1 \leq i \leq m$),
- ▶ C_i is a well-formed selection condition over \mathcal{A} (for $1 \leq i \leq j$), and
- ▶ $A_1, \dots, A_k \in \mathcal{A}$

for the set \mathcal{A} of all variables appearing in $\overline{A^1}, \dots, \overline{A^m}$.

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

List the titles of all books written by Italian speakers.

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

In SQL

```
SELECT book.title  
FROM author, book  
WHERE book.authorID = author.authorID  
      AND  
      author.language = 'Italian'
```

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

In RA

$$\pi_{book.title}(\sigma_C(author \times book))$$

where C is

$$book.authorID = author.authorID \wedge author.language = \text{'Italian'}$$

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

In Datalog

$result(T) \leftarrow author(A, N, D, LA), book(B, T, A, P, LB, Y),$
 $LA = \text{'Italian'}$

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

List the IDs of all authors writing in Italian or born in 1985.

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

```
SELECT A.authorID  
FROM Author A  
WHERE A.Language = 'Italian'  
UNION  
SELECT A.authorID  
FROM Author A  
WHERE A.Birthdate = 1985
```

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

```
SELECT A.authorID  
FROM Author A  
WHERE A.Language = 'Italian'  
UNION  
SELECT A.authorID  
FROM Author A  
WHERE A.Birthdate = 1985
```

$$\pi_{authorID}(\sigma_{language='I...'}(author)) \cup \pi_{authorID}(\sigma_{birthdate=1985}(author))$$

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

```
SELECT A.authorID  
FROM Author A  
WHERE A.Language = 'Italian'  
UNION  
SELECT A.authorID  
FROM Author A  
WHERE A.Birthdate = 1985
```

$$\pi_{authorID}(\sigma_{language='I...'}(author)) \cup \pi_{authorID}(\sigma_{birthdate=1985}(author))$$

result(A) ← author(A, N, B, L), L = 'Italian'

result(A) ← author(A, N, B, L), B = 1985

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

List the IDs of all authors writing in Italian and born in 1985.

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

```
SELECT A.authorID  
FROM Author A  
WHERE A.Language = 'Italian'
```

INTERSECT

```
SELECT A.authorID  
FROM Author A  
WHERE A.Birthdate = 1985
```

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

```
SELECT A.authorID  
FROM Author A  
WHERE A.Language = 'Italian'  
INTERSECT
```

```
SELECT A.authorID  
FROM Author A  
WHERE A.Birthdate = 1985
```

$$\pi_{authorID}(\sigma_{language='I...'}(author)) \cap \pi_{authorID}(\sigma_{birthdate=1985}(author))$$

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

```
SELECT A.authorID  
FROM Author A  
WHERE A.Language = 'Italian'
```

INTERSECT

```
SELECT A.authorID  
FROM Author A  
WHERE A.Birthdate = 1985
```

$$\pi_{authorID}(\sigma_{language='I...'}(author)) \cap \pi_{authorID}(\sigma_{birthdate=1985}(author))$$

$I(A) \leftarrow author(A, N, B, L), L = 'Italian'$

$B(A) \leftarrow author(A, N, B, L), B = 1985$

$result(A) \leftarrow I(A), B(A)$

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

List the IDs of all authors writing in Italian not born in 1985.

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

```
SELECT A.authorID  
FROM Author A  
WHERE A.Language = 'Italian'  
EXCEPT
```

```
SELECT A.authorID  
FROM Author A  
WHERE A.Birthdate = 1985
```

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

```
SELECT A.authorID  
FROM Author A  
WHERE A.Language = 'Italian'  
EXCEPT  
SELECT A.authorID  
FROM Author A  
WHERE A.Birthdate = 1985
```

$$\pi_{authorID}(\sigma_{language='I...'}(author)) - \pi_{authorID}(\sigma_{birthdate=1985}(author))$$

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

```
SELECT A.authorID  
FROM Author A  
WHERE A.Language = 'Italian'  
EXCEPT  
SELECT A.authorID  
FROM Author A  
WHERE A.Birthdate = 1985
```

$$\pi_{authorID}(\sigma_{language='I...'}(author)) - \pi_{authorID}(\sigma_{birthdate=1985}(author))$$

$I(A) \leftarrow author(A, N, B, L), L = 'Italian'$

$B(A) \leftarrow author(A, N, B, L), B = 1985$

$result(A) \leftarrow I(A), not B(A)$

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

List the IDs of all books which are sold in every store.

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

In RA

$$\pi_{bookID}(book) - \pi_{bookID}((\pi_{storeID}(store) \times \pi_{bookID}(book)) - sells)$$

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

In SQL

```
SELECT B.bookID
FROM book B
WHERE NOT EXISTS
(SELECT bookID, storeID
 FROM book, store
 WHERE bookID=B.bookID
 EXCEPT
 sells)
```

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeId, address, phone)

sells(storeId, bookID)

In Datalog

all(S, B) \leftarrow *book*(B, T, A, P, L, Y), *store*(S, Ad, Ph)

missing(B) \leftarrow *all*(S, B), *not sells*(S, B)

result(B) \leftarrow *book*(B, T, A, P, L, Y), *not missing*(B)

FO review: syntax

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

in TRC, the Tuple Relational Calculus (i.e., essentially straight FO)

$$\{t \mid \exists b \in book(t.bookID = b.bookID \wedge \\ \forall s \in store \exists \ell \in sell(\ell.storeID = s.storeID \\ \wedge \ell.bookID = b.bookID))\}$$

FO review: complexity of query evaluation

Fact. SQL (without aggregation and other bells-and-whistles), RA, TRC, and (safe non-recursive) Datalog are all equivalent in expressive power.

FO review: complexity of query evaluation

Fact. SQL (without aggregation and other bells-and-whistles), RA, TRC, and (safe non-recursive) Datalog are all equivalent in expressive power.

Let

- ▶ FO denote the full TRC
 - ▶ i.e., any of the above languages

FO review: complexity of query evaluation

Fact. SQL (without aggregation and other bells-and-whistles), RA, TRC, and (safe non-recursive) Datalog are all equivalent in expressive power.

Let

- ▶ FO denote the full TRC
 - ▶ i.e., any of the above languages
- ▶ $Conj$ denote the TRC using only \exists and \wedge
 - ▶ i.e., the “conjunctive” queries
 - ▶ corresponds in SQL to basic SELECT-FROM-WHERE blocks
 - ▶ corresponds to the $\{\sigma, \pi, \times\}$ fragment of RA
 - ▶ corresponds to single positive datalog rules

FO review: complexity of query evaluation

Fact. SQL (without aggregation and other bells-and-whistles), RA, TRC, and (safe non-recursive) Datalog are all equivalent in expressive power.

Let

- ▶ FO denote the full TRC
 - ▶ i.e., any of the above languages
- ▶ $Conj$ denote the TRC using only \exists and \wedge
 - ▶ i.e., the “conjunctive” queries
 - ▶ corresponds in SQL to basic SELECT-FROM-WHERE blocks
 - ▶ corresponds to the $\{\sigma, \pi, \times\}$ fragment of RA
 - ▶ corresponds to single positive datalog rules
- ▶ $AConj$ denote the “acyclic” conjunctive queries
 - ▶ queries with join trees
 - ▶ full def in a later lecture

FO review: complexity of query evaluation

Fact. The complexity of query evaluation is as follows

	<i>FO</i>	<i>Conj</i>	<i>AConj</i>
<i>combined data</i>	PSPACE-complete Logspace	NP-complete Logspace	LOGCFL-complete Linear time

where *combined* means in the size of the query and the database and *data* means in the size of the database (i.e., for some fixed query).

Books schema

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeId, address, phone)

sells(storeId, bookID)

Books schema

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeId, address, phone)

sells(storeId, bookID)

1. List (the bookIDs of) all books authored by a native speaker.

Books schema

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeId, address, phone)

sells(storeId, bookID)

1. List (the bookIDs of) all books authored by a native speaker.
2. List (the authorIDs of) all authors who only have books appearing in their native language.

Books schema

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeId, address, phone)

sells(storeId, bookID)

1. List (the bookIDs of) all books authored by a native speaker.
2. List (the authorIDs of) all authors who only have books appearing in their native language.
3. List all books which are not available in stores.

Books schema

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeId, address, phone)

sells(storeId, bookID)

1. List (the bookIDs of) all books authored by a native speaker.
2. List (the authorIDs of) all authors who only have books appearing in their native language.
3. List all books which are not available in stores.
4. List all books sold in more than one store.

Books schema

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeId, address, phone)

sells(storeId, bookID)

1. List (the bookIDs of) all books authored by a native speaker.
2. List (the authorIDs of) all authors who only have books appearing in their native language.
3. List all books which are not available in stores.
4. List all books sold in more than one store.
5. List all books sold in exactly one store.

Books schema

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeId, address, phone)

sells(storeId, bookID)

1. List (the bookIDs of) all books authored by a native speaker.
2. List (the authorIDs of) all authors who only have books appearing in their native language.
3. List all books which are not available in stores.
4. List all books sold in more than one store.
5. List all books sold in exactly one store.
6. List all (authorID, language) pairs where the given author has not published a book in the given language.

Books schema

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeId, address, phone)

sells(storeId, bookID)

1. List (the bookIDs of) all books authored by a native speaker.
2. List (the authorIDs of) all authors who only have books appearing in their native language.
3. List all books which are not available in stores.
4. List all books sold in more than one store.
5. List all books sold in exactly one store.
6. List all (authorID, language) pairs where the given author has not published a book in the given language.
7. List all authors which have a book in every known language (i.e., every language occurring in the author or book tables). Note that this can be a different book for each language.

Storage, I/O complexity, & external sorting

Lecture 2
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

24 April 2015

Agenda

Last time: course overview and background

Agenda

Last time: course overview and background

Today

- ▶ Admin: results of 5-minute paper
- ▶ Admin: project update
- ▶ Finish background review: FO queries

Agenda

Last time: course overview and background

Today

- ▶ Admin: results of 5-minute paper
- ▶ Admin: project update
- ▶ Finish background review: FO queries
- ▶ Topic 1: Storage on disk & I/O model of computing
- ▶ Topic 2: External sorting
- ▶ Topic 3: Indexing, part 1 – ordered indexes

Admin: results of 5-minute papers

Interests range from foundations to applications

- ▶ “Realising an efficient database implementation.”
- ▶ “How research in database technology is developing.”
- ▶ “I would like to see some real-world examples, not only the theoretical viewpoint.”

Admin: results of 5-minute papers

Interests range from foundations to applications

- ▶ “Realising an efficient database implementation.”
- ▶ “How research in database technology is developing.”
- ▶ “I would like to see some real-world examples, not only the theoretical viewpoint.”

many papers: NoSQL technologies (JSON stores, Graph DBs)

- ▶ many non-relational technologies are explored in the team projects
- ▶ let's also shift lecture of 5 June to further highlight these topics

Admin: results of 5-minute papers, cont.

Also, several papers: obtain knowledge to be able to make effective data management decisions in practice

- ▶ "How to make use of in-depth knowledge about the inner workings of current existing popular DB systems, while working in industry."
- ▶ "How are the technologies discussed used in a business setting?"
- ▶ "To know more about the different technologies used in DBs so I can make informed decisions on which existing DB software to use in my applications."

Admin: results of 5-minute papers, cont.

Also, several papers: obtain knowledge to be able to make effective data management decisions in practice

- ▶ “How to make use of in-depth knowledge about the inner workings of current existing popular DB systems, while working in industry.”
- ▶ “How are the technologies discussed used in a business setting?”
- ▶ “To know more about the different technologies used in DBs so I can make informed decisions on which existing DB software to use in my applications.”

core goal of 2ID35: developing the solid technical foundations for making such decisions, for both state-of-the-art and emerging technologies

Funny Dilbert comic strip removed. See
<http://search.dilbert.com/search?w=mauve+database>

Admin

Project part 1 update:

- ▶ Please finish your submissions ASAP, if you haven't already
 - ▶ I will give contact via email
 - ▶ ~5 members per team
 - ▶ each team focuses on one paper
- ▶ Next step: carefully read and understand team paper. Full details to follow on course website.

FO review

author(authorID, name, birthdate, language)

book(bookID, title, authorID, publisher, language, year)

store(storeID, address, phone)

sells(storeID, bookID)

- ▶ List all (authorID, language) pairs where the given author has not published a book in the given language.

Agenda

- ▶ Topic 1: Storage on disk & I/O model of computing
- ▶ Topic 2: External sorting
- ▶ Topic 3: Indexing, part 1 – ordered indexes

Topic 1: Storage

Storage

Assumption: database may grow too large to maintain in volatile memory and/or we'd like to park it on stable storage

Storage

Assumption: database may grow too large to maintain in volatile memory and/or we'd like to park it on stable storage

- ▶ the memory hierarchy
- ▶ mapping relations to disk
- ▶ the structure of tuples/relations on disk
- ▶ the buffer manager
- ▶ I/O computing

Storage: the memory hierarchy

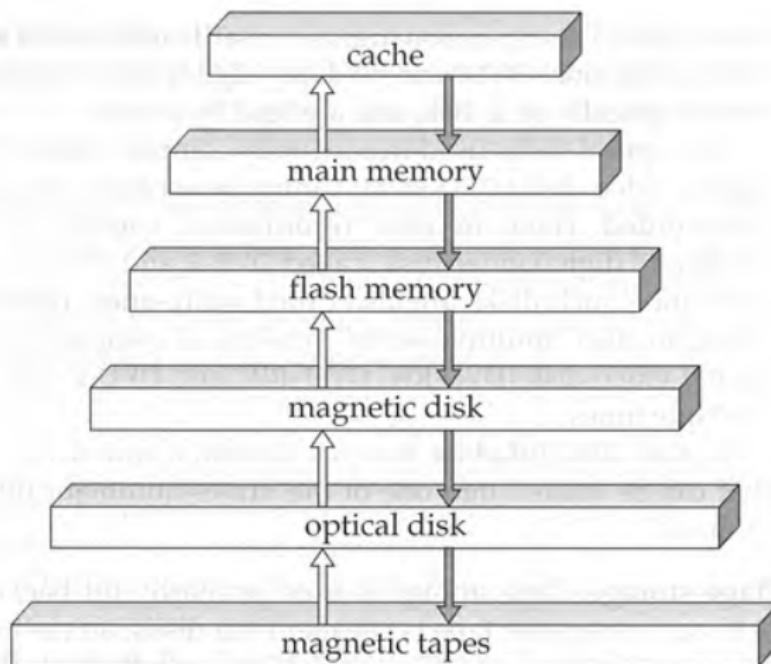
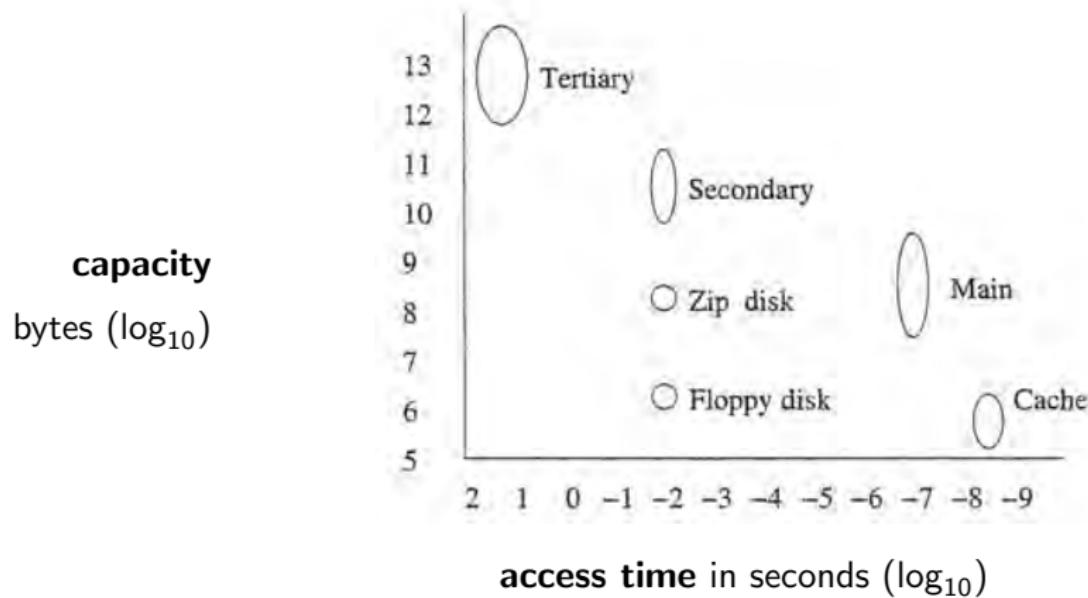
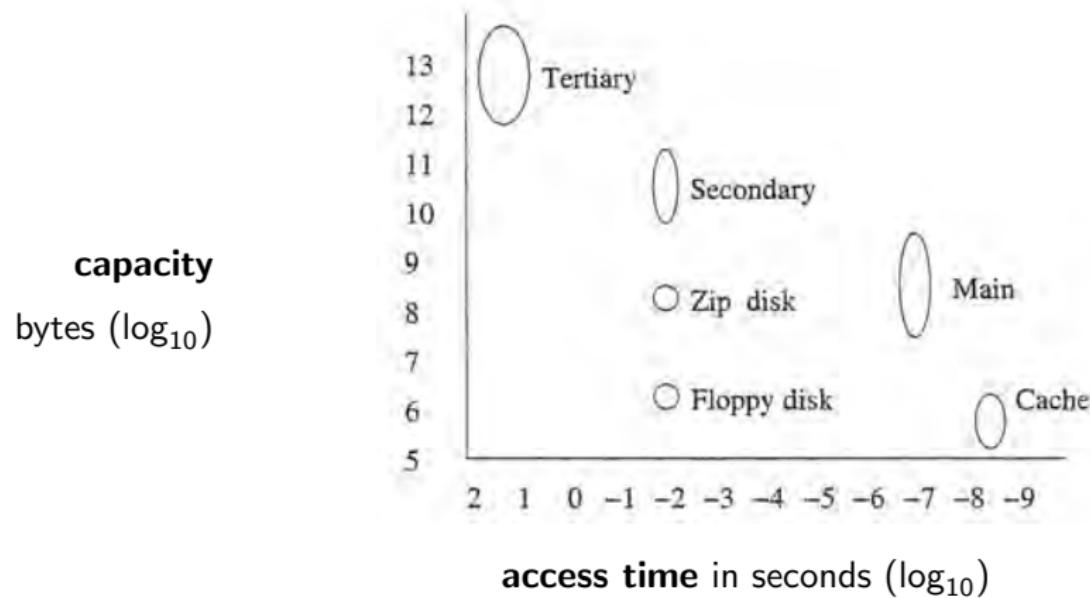


Figure 11.1 Storage device hierarchy.

Storage: the memory hierarchy



Storage: the memory hierarchy



- ⇒ main memory is $\approx 10^5$ times faster and $\frac{1}{100}$ the size of secondary memory
- ▶ SSDs narrow the speed gap by $\approx 10^2$

Storage: the memory hierarchy

As a comparison, if a book in your hand represents primary memory and a book in the library is like secondary memory, and it takes 20 **seconds** to locate an item in the book at hand....

Storage: the memory hierarchy

As a comparison, if a book in your hand represents primary memory and a book in the library is like secondary memory, and it takes 20 **seconds** to locate an item in the book at hand....

... it would take over 23 **days** to locate an item in a book in the library ...

Storage: the memory hierarchy

As a comparison, if a book in your hand represents primary memory and a book in the library is like secondary memory, and it takes 20 **seconds** to locate an item in the book at hand....

... it would take over 23 **days** to locate an item in a book in the library ...

... not a very good library ...

Storage: the memory hierarchy

As another comparison, if retrieving something from main memory is like retrieving an object from Amsterdam (126 km), then retrieving something from disk is like ...

Storage: the memory hierarchy

As another comparison, if retrieving something from main memory is like retrieving an object from Amsterdam (126 km), then retrieving something from disk is like ...
traveling to New York city?

Storage: the memory hierarchy

As another comparison, if retrieving something from main memory is like retrieving an object from Amsterdam (126 km), then retrieving something from disk is like ...

traveling to New York city?

no, that is only 6,000 km away

Storage: the memory hierarchy

As another comparison, if retrieving something from main memory is like retrieving an object from Amsterdam (126 km), then retrieving something from disk is like ...

traveling to New York city?

no, that is only 6,000 km away

traveling to the moon?

Storage: the memory hierarchy

As another comparison, if retrieving something from main memory is like retrieving an object from Amsterdam (126 km), then retrieving something from disk is like ...

traveling to New York city?

no, that is only 6,000 km away

traveling to the moon?

no, that is only 385,000 km away

Storage: the memory hierarchy

As another comparison, if retrieving something from main memory is like retrieving an object from Amsterdam (126 km), then retrieving something from disk is like ...

traveling to New York city?

no, that is only 6,000 km away

traveling to the moon?

no, that is only 385,000 km away

it would be like traveling to New York **2100 times**
(or, around the earth **315 times**, or only 33 times to
the moon)!

Storage: the memory hierarchy

moral: we don't want to travel to the moon, unless we absolutely must

Storage: secondary memory

- ▶ tuple → record
- ▶ relation (collection of records) → file

Storage: secondary memory

- ▶ tuple → record
- ▶ relation (collection of records) → file
- ▶ file → collection of pages (disk blocks)
- ▶ block size typically in the range 4K to 56K

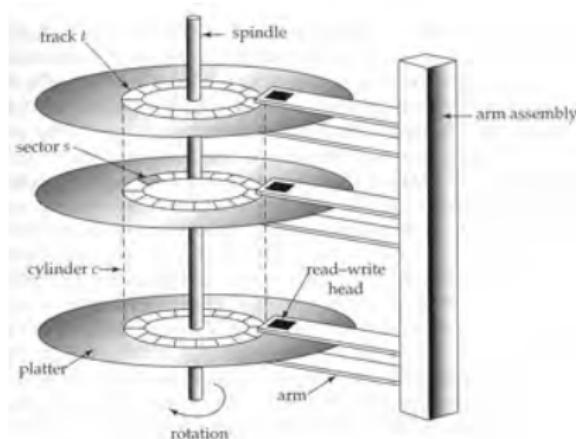
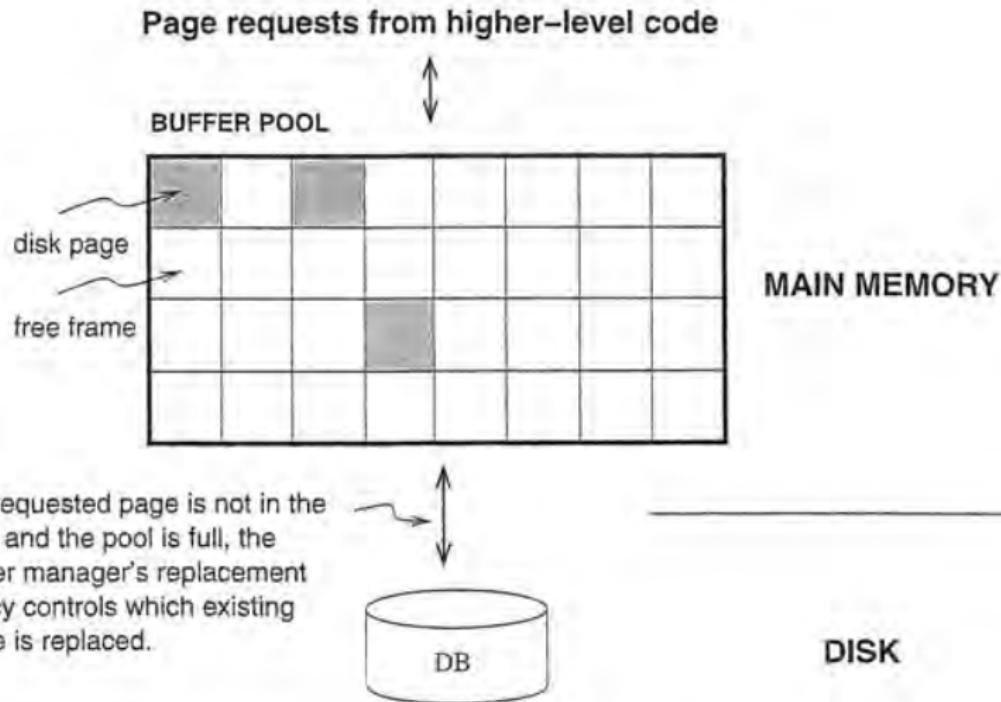


Figure 11.2 Moving head disk mechanism.

Storage: the buffer manager

We must bring data into lowest memory level (i.e., CPU registers) to perform work.

Storage: the buffer manager



I/O model of computation

- ▶ disk block is unit of I/O
- ▶ model is defined by dominance of I/O cost over CPU cost
- ▶ parameters:
 - ▶ B = number of blocks in a file
 - ▶ D = average time to read a block
 - ▶ R = number of records per block
 - ▶ C = average time to process a record

I/O model of computation

- ▶ disk block is unit of I/O
- ▶ model is defined by dominance of I/O cost over CPU cost
- ▶ parameters:
 - ▶ B = number of blocks in a file
 - ▶ D = average time to read a block
 - ▶ R = number of records per block
 - ▶ C = average time to process a record

Typically $D = 15$ milliseconds ($10^{-3}s$) and $C = 100$ nanoseconds ($10^{-9}s$)

⇒ 150,000 record ops per block access!!

I/O model of computation

- ▶ disk block is unit of I/O
- ▶ model is defined by dominance of I/O cost over CPU cost
- ▶ parameters:
 - ▶ B = number of blocks in a file
 - ▶ D = average time to read a block
 - ▶ R = number of records per block
 - ▶ C = average time to process a record

Suppose 16KB blocks and 160B records

$\Rightarrow R = 100$ records per block

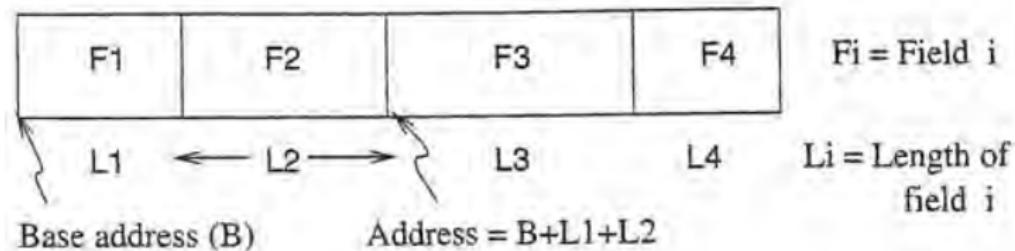
\Rightarrow even a linear scan is negligible wrt block access!

Storage: record structure

- ▶ a file is organized logically as a sequence of records, and these records are mapped onto disk blocks
- ▶ two basic record types: **fixed length** vs. variable length

Storage: record structure

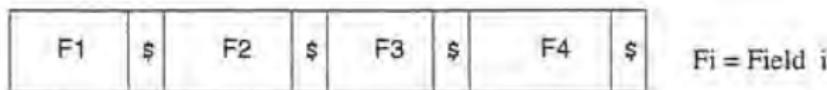
- ▶ a file is organized logically as a sequence of records, and these records are mapped onto disk blocks
- ▶ two basic record types: **fixed length** vs. variable length



Organization of Records with Fixed-Length Fields

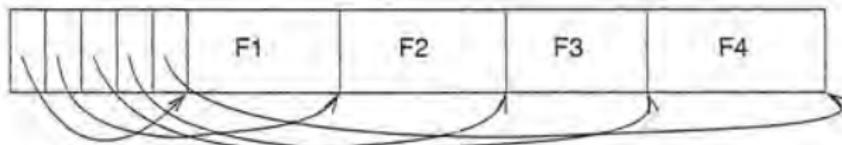
Storage: record structure

- ▶ a file is organized logically as a sequence of records, and these records are mapped onto disk blocks
- ▶ two basic record types: fixed length vs. **variable length**



$F_i = \text{Field } i$

Fields delimited by special symbol \$



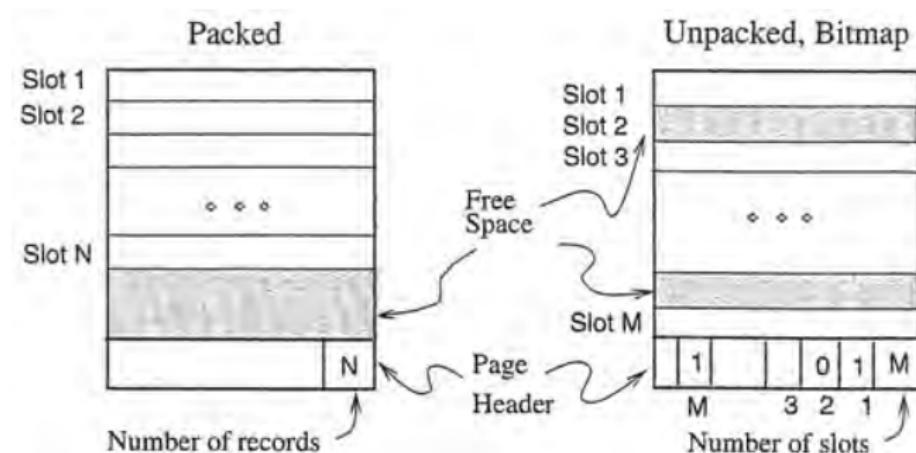
Array of field offsets

Storage: page structure

Hence, we have two basic types of page structure

Storage: page structure

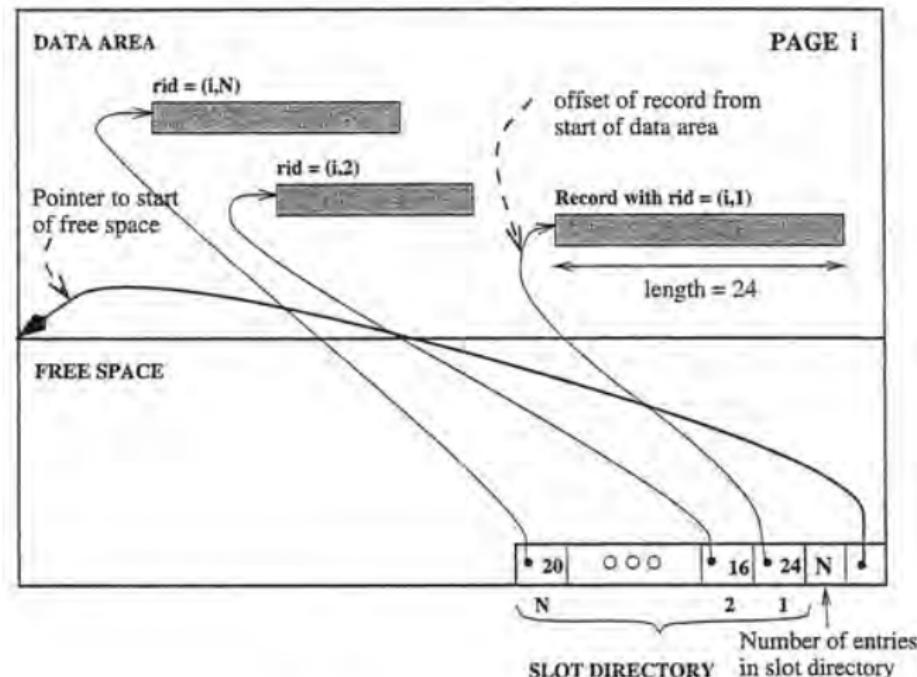
Hence, we have two basic types of page structure



Alternative Page Organizations for Fixed-Length Records

Storage: page structure

Hence, we have two basic types of page structure



Storage: file structure

Within a file, records can be maintained

- ▶ *sequentially*, i.e., ordered on some field(s); as a

Storage: file structure

Within a file, records can be maintained

- ▶ *sequentially*, i.e., ordered on some field(s); as a
- ▶ *heap*, i.e., unordered; or, as a

Storage: file structure

Within a file, records can be maintained

- ▶ *sequentially*, i.e., ordered on some field(s); as a
- ▶ *heap*, i.e., unordered; or, as a
- ▶ *hash* file (which we'll study in some detail in a few lectures).

Storage: file structure

Within a file, records can be maintained

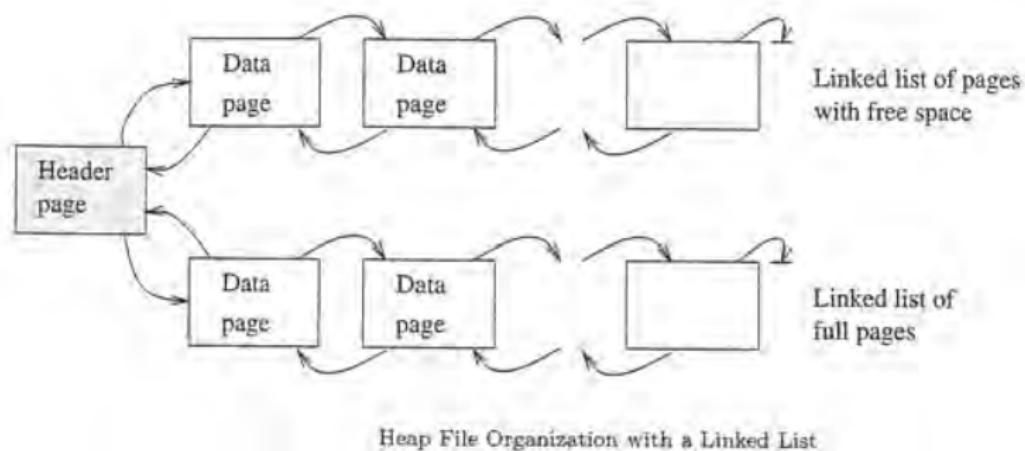
- ▶ *sequentially*, i.e., ordered on some field(s); as a
- ▶ *heap*, i.e., unordered; or, as a
- ▶ *hash* file (which we'll study in some detail in a few lectures).

Sorted data is very valuable, for a variety of reasons as we'll see, yet requires some work to maintain.

We'll investigate *indexing* as a means for maintaining a sorted file, in the next lecture.

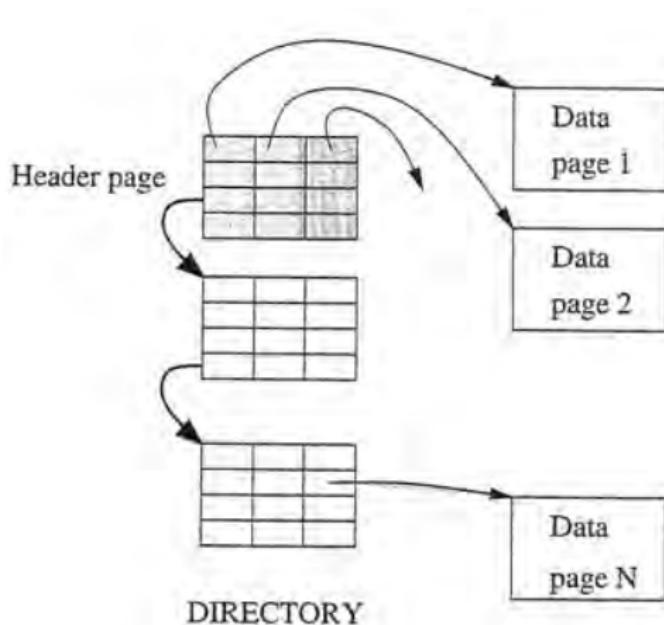
Storage: heap file structure

There are several natural ways to maintain a heap: as a linked list.



Storage: heap file structure

There are several natural ways to maintain a heap:
using a directory.



Heap File Organization with a Directory

Searching in a heap

Recall our scenario of 16KB blocks and 160B records (i.e., 100 records/block).

Suppose our relation has 10,000,000 tuples, and therefore occupies 100,000 blocks.

Searching in a heap

Recall our scenario of 16KB blocks and 160B records (i.e., 100 records/block).

Suppose our relation has 10,000,000 tuples, and therefore occupies 100,000 blocks.

Now, suppose we are looking for records with a given value in a field (say, authors named Jan).

Since searching in a heap is exhaustive, the delay for this search is

$$\begin{aligned} B \times D &= 100,000 \times 0.015s \\ &= 1500s \\ &= 25 \text{ minutes} \end{aligned}$$

Searching in a heap

Clearly, this is unacceptable a sequential file, sorted on author names would be nice ...

Searching in a heap

Clearly, this is unacceptable a sequential file, sorted on author names would be nice ...

some other uses of sorted data:

- ▶ presentation of results,
- ▶ computing aggregates,
- ▶ duplicate elimination,
- ▶ algorithms for RA operations (e.g., join and projection)

Searching in a heap

Clearly, this is unacceptable a sequential file, sorted on author names would be nice ...

some other uses of sorted data:

- ▶ presentation of results,
- ▶ computing aggregates,
- ▶ duplicate elimination,
- ▶ algorithms for RA operations (e.g., join and projection)

So, let's consider how to efficiently sort in external memory

Topic 2: External sorting

Sorting: two-way external merge sort

Suppose we have 3 buffer frames available, and 2^k pages in the file to be sorted.

Sorting: two-way external merge sort

Suppose we have 3 buffer frames available, and 2^k pages in the file to be sorted.

- (0) read each page in, sort in main-memory, and write it out. result: 2^k sorted runs of length 1.

Sorting: two-way external merge sort

Suppose we have 3 buffer frames available, and 2^k pages in the file to be sorted.

- (0) read each page in, sort in main-memory, and write it out. result: 2^k sorted runs of length 1.
- (1) read in two runs, merge sort, and write out. result: 2^{k-1} sorted runs of length 2.

Sorting: two-way external merge sort

Suppose we have 3 buffer frames available, and 2^k pages in the file to be sorted.

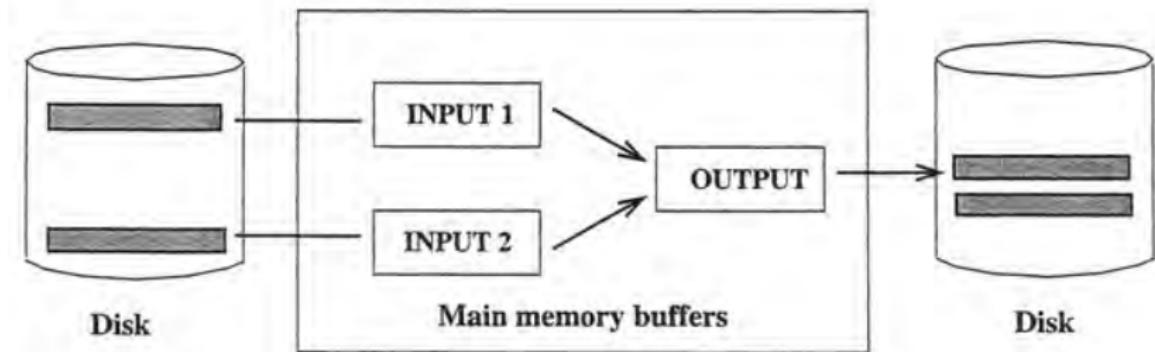
- (0) read each page in, sort in main-memory, and write it out. result: 2^k sorted runs of length 1.
- (1) read in two runs, merge sort, and write out. result: 2^{k-1} sorted runs of length 2.
- (2) read in two runs, merge sort, and write out. result: 2^{k-2} sorted runs of length 4.

Sorting: two-way external merge sort

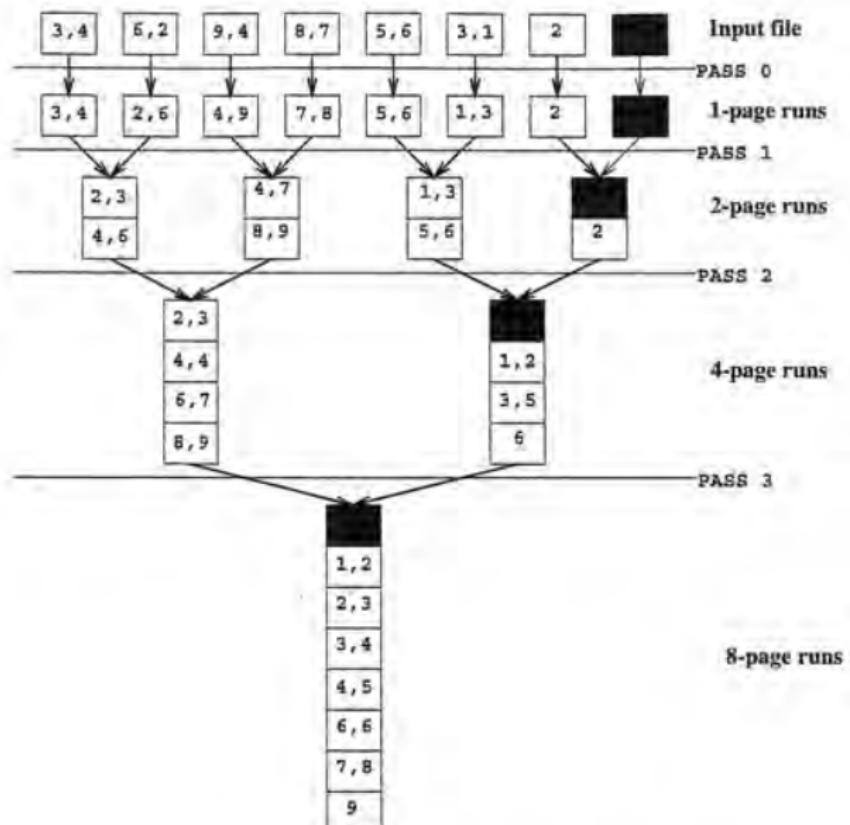
Suppose we have 3 buffer frames available, and 2^k pages in the file to be sorted.

- (0) read each page in, sort in main-memory, and write it out. result: 2^k sorted runs of length 1.
- (1) read in two runs, merge sort, and write out. result: 2^{k-1} sorted runs of length 2.
- (2) read in two runs, merge sort, and write out. result: 2^{k-2} sorted runs of length 4.
- ⋮
- (k) read in two runs, merge sort, and write out. result: 2^0 sorted runs of length 2^k .

Sorting: two-way external merge sort



Sorting: two-way external merge sort



Sorting: two-way external merge sort

So, with 2 I/Os per page, per pass, we have

$$2B(\lceil \log B \rceil + 1)$$

I/Os for two-way merge sort

Sorting: two-way external merge sort

So, with 2 I/Os per page, per pass, we have

$$2B(\lceil \log B \rceil + 1)$$

I/Os for two-way merge sort

In our example, one scan was 25 minutes, giving us 50 minutes per pass, and therefore

$$\begin{aligned} 50 \times (\lceil \log 100,000 \rceil + 1) &= 50 \times 18 \\ &= 900 \text{ minutes} \\ &= 15 \text{ hours} \end{aligned}$$

Sorting: two-way external merge sort

So, with 2 I/Os per page, per pass, we have

$$2B(\lceil \log B \rceil + 1)$$

I/Os for two-way merge sort

In our example, one scan was 25 minutes, giving us 50 minutes per pass, and therefore

$$\begin{aligned} 50 \times (\lceil \log 100,000 \rceil + 1) &= 50 \times 18 \\ &= 900 \text{ minutes} \\ &= 15 \text{ hours} \end{aligned}$$

a disaster.

Sorting

A disaster indeed

But what if we have N buffer pages available (for some $N > 3$)?

We can do an $(N - 1)$ -way merge sort. Perhaps this will help?

Sorting

A disaster indeed

But what if we have N buffer pages available (for some $N > 3$)?

We can do an $(N - 1)$ -way merge sort. Perhaps this will help?

Why $N - 1$?

Sorting: external merge sort

Suppose we have N buffer frames available.

- ▶ (pass 0) read in N pages at a time, sort in main-memory, and write out. result: $\lceil \frac{B}{N} \rceil$ sorted runs of length N .

Sorting: external merge sort

Suppose we have N buffer frames available.

- ▶ (pass 0) read in N pages at a time, sort in main-memory, and write out. result: $\lceil \frac{B}{N} \rceil$ sorted runs of length N .
- ▶ (pass 1, 2, ...) Use $N - 1$ buffer frames to do an $N - 1$ way merge sort

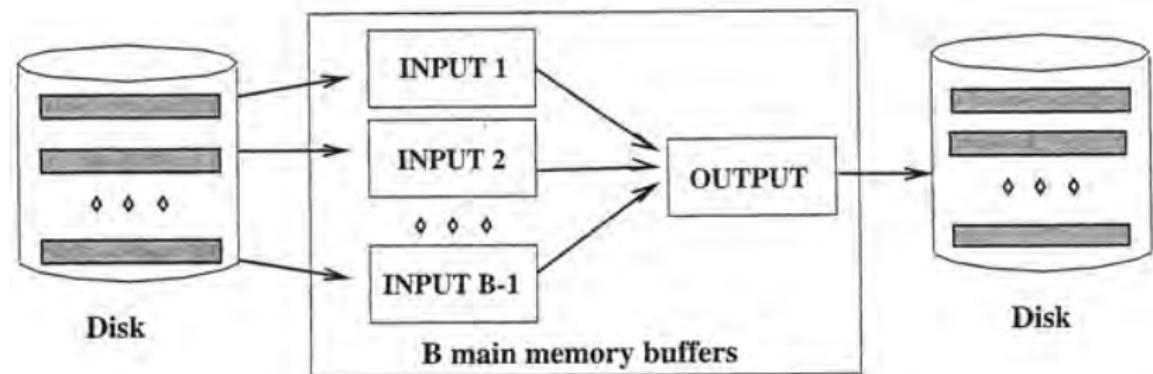
Sorting: external merge sort

Suppose we have N buffer frames available.

- ▶ (pass 0) read in N pages at a time, sort in main-memory, and write out. result: $\lceil \frac{B}{N} \rceil$ sorted runs of length N .
- ▶ (pass 1, 2, ...) Use $N - 1$ buffer frames to do an $N - 1$ way merge sort

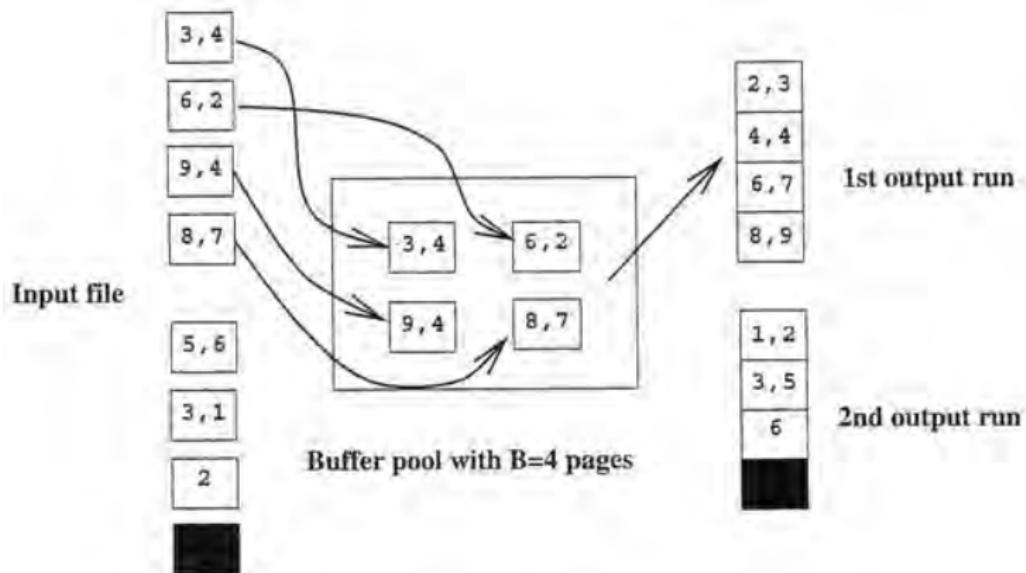
Number of passes: $\lceil \log_{N-1} \lceil \frac{B}{N} \rceil \rceil + 1$

Sorting: external merge sort



External Merge Sort with B Buffer Pages: Pass $i > 0$

Sorting: external merge sort



External Merge Sort with B Buffer Pages: Pass 0

Sorting: external merge sort

Suppose $N = 101$, and let's revisit our example.

$$\begin{aligned} 50 \times (\lceil \log_{100} \left\lceil \frac{100,000}{101} \right\rceil \rceil + 1) &= 50 \times (\lceil 1.5 \rceil + 1) \\ &= 150 \text{ minutes} \\ &= 2.5 \text{ hours} \end{aligned}$$

Not too shabby

Sorting: external merge sort

B	$N = 3$	$N = 5$	$N = 9$	$N = 17$	$N = 129$	$N = 257$
100	7	4	3	2	1	1
1000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

Number of passes of external merge sort

Exercise

Suppose we have $N = 3$ buffer frames available and at most two records fit on a block. Given the file

$\langle [9, 8], [2, 1], [7, 6], [4, 3], [5, 10], [5, 1], [4, 11], [12] \rangle$

show each step of performing an external merge sort on the file.

Topic 3: Ordered indexes

Indexing

- ▶ **key**: a collection of attributes (of a relation)

Indexing

- ▶ **key**: a collection of attributes (of a relation)
- ▶ **index**:
 - ▶ data structure for searching (i.e., mapping key values to data)

Indexing

- ▶ **key**: a collection of attributes (of a relation)
- ▶ **index**:
 - ▶ data structure for searching (i.e., mapping key values to data)
 - ▶ i.e., collection of data entries $k*$ for search key k

Indexing

we are of course primarily interested in *external* search

key value → index

→ blocks holding records

→ matching records

Indexing

we are of course primarily interested in *external* search

key value → index
→ blocks holding records
→ matching records

Example. Find all bookstore tuples with
“*City* = Eindhoven”

Indexing

- ▶ *basic operations:*
 - ▶ $\text{search}(k)$
 - ▶ $\text{insert}(k)$
 - ▶ $\text{delete}(k)$

Indexing

- ▶ *basic operations:*
 - ▶ $\text{search}(k)$
 - ▶ $\text{insert}(k)$
 - ▶ $\text{delete}(k)$
- ▶ also known as: dictionaries or symbol tables
 - ▶ ex: linked-list, binary search tree, AVL tree, skip list, hash table

Index entries

- ▶ index = collection of data entries k^* for search key k
- ▶ three main alternatives for k^*

Index entries

- ▶ index = collection of data entries k^* for search key k
- ▶ three main alternatives for k^*
 1. k^* is actual data record for k

Index entries

- ▶ index = collection of data entries k^* for search key k
- ▶ three main alternatives for k^*
 1. k^* is actual data record for k
 2. $k^* = \langle k, rid \rangle$, where rid is a record-id of a data record with key k

Index entries

- ▶ index = collection of data entries k^* for search key k
- ▶ three main alternatives for k^*
 1. k^* is actual data record for k
 2. $k^* = \langle k, rid \rangle$, where rid is a record-id of a data record with key k
 3. $k^* = \langle k, ridList \rangle$, where $ridList$ is a list of record-id's of data records with key k

Index entries

- ▶ index = collection of data entries $k*$ for search key k
- ▶ three main alternatives for $k*$
 1. $k*$ is actual data record for k
 2. $k* = \langle k, rid \rangle$, where rid is a record-id of a data record with key k
 3. $k* = \langle k, ridList \rangle$, where $ridList$ is a list of record-id's of data records with key k
- ▶ alternative (1) is called an indexed file organization
- ▶ alternatives (2) and (3) are independent of the underlying file's organization

Indexing

Evaluation criteria

- ▶ construction cost

Indexing

Evaluation criteria

- ▶ construction cost
- ▶ update cost: insertion/deletion

Indexing

Evaluation criteria

- ▶ construction cost
- ▶ update cost: insertion/deletion
- ▶ space overhead

Indexing

Evaluation criteria

- ▶ construction cost
- ▶ update cost: insertion/deletion
- ▶ space overhead
- ▶ access types

Indexing

Evaluation criteria

- ▶ construction cost
- ▶ update cost: insertion/deletion
- ▶ space overhead
- ▶ access types
- ▶ access cost

Indexing

Two basic types of indexes

- ▶ **Ordered indexes.** based on a sorted ordering of the values

Indexing

Two basic types of indexes

- ▶ **Ordered indexes.** based on a sorted ordering of the values
- ▶ **Hash indexes.** based on a uniform distribution of values across a range of “buckets.” The bucket to which a value is assigned is determined by a hash function. (e.g., integer value of key, modulo index size)

Ordered indexes

Based on a sorted ordering of the values, just like the index of a book

Ordered indexes

Based on a sorted ordering of the values, just like the index of a book

- ▶ **clustering** (clustered, primary): search key also defines the sequential order of the file
 - ▶ i.e., data records in file are sorted in the same order as in the index (on the search key fields)

Ordered indexes

Based on a sorted ordering of the values, just like the index of a book

- ▶ **clustering** (clustered, primary): search key also defines the sequential order of the file
 - ▶ i.e., data records in file are sorted in the same order as in the index (on the search key fields)
- ▶ **nonclustering** (nonclustered, secondary): search key specifies an order different from the sequential order of the file

Ordered indexes

two types:

Ordered indexes

two types:

- ▶ **dense**: an index record appears for *every* search-key value in the file

Ordered indexes

two types:

- ▶ **dense**: an index record appears for *every* search-key value in the file
- ▶ **sparse**: an index record appears for only *some* of the search-key values in the file

Ordered indexes

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

Figure 12.1 Sequential file for *account* records.

Ordered indexes: dense clustering index

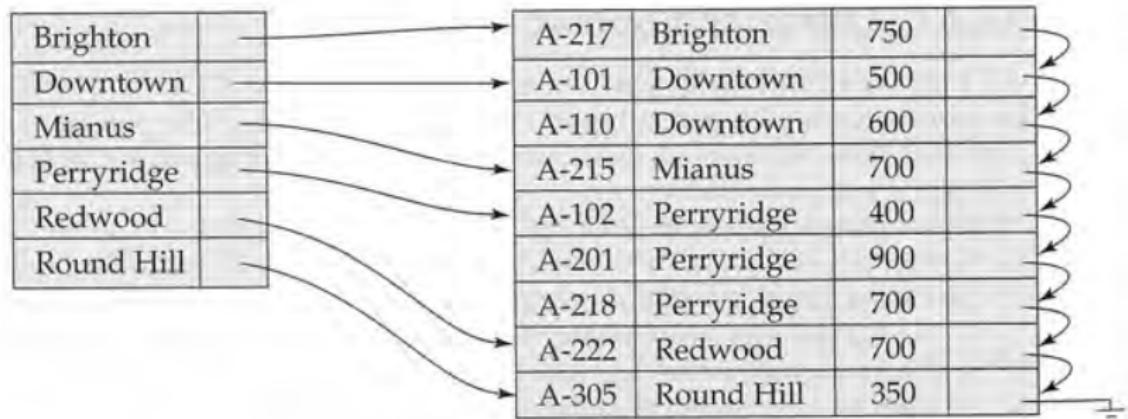


Figure 12.2 Dense index.

Ordered indexes: sparse clustering index

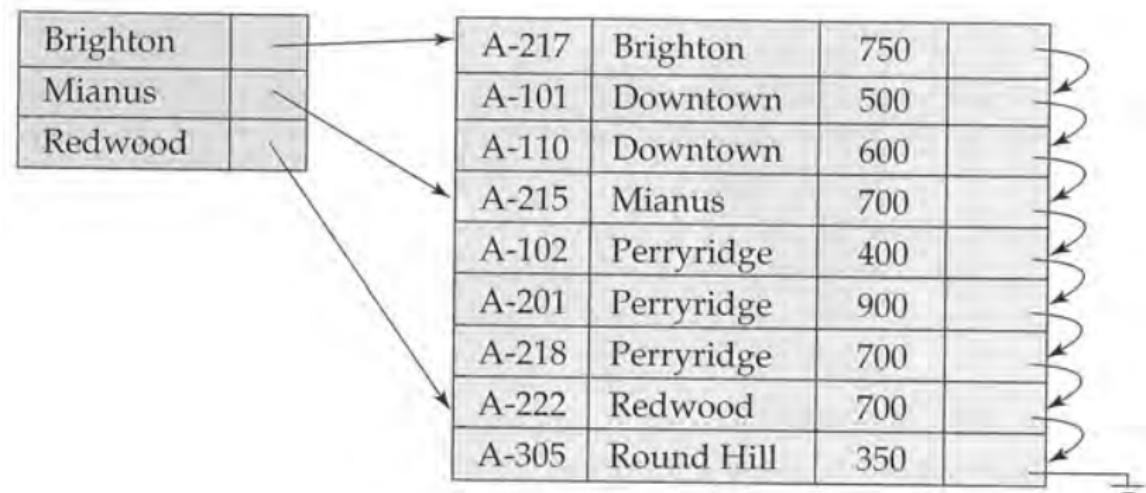


Figure 12.3 Sparse index.

Nonclustering index

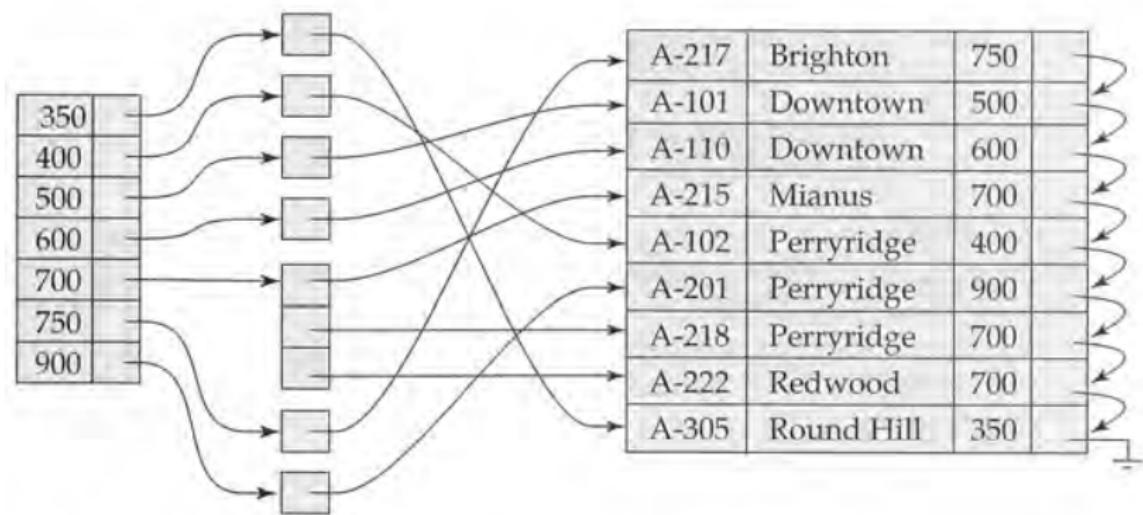


Figure 12.5 Secondary index on *account* file, on noncandidate key *balance*.

Nonclustering index

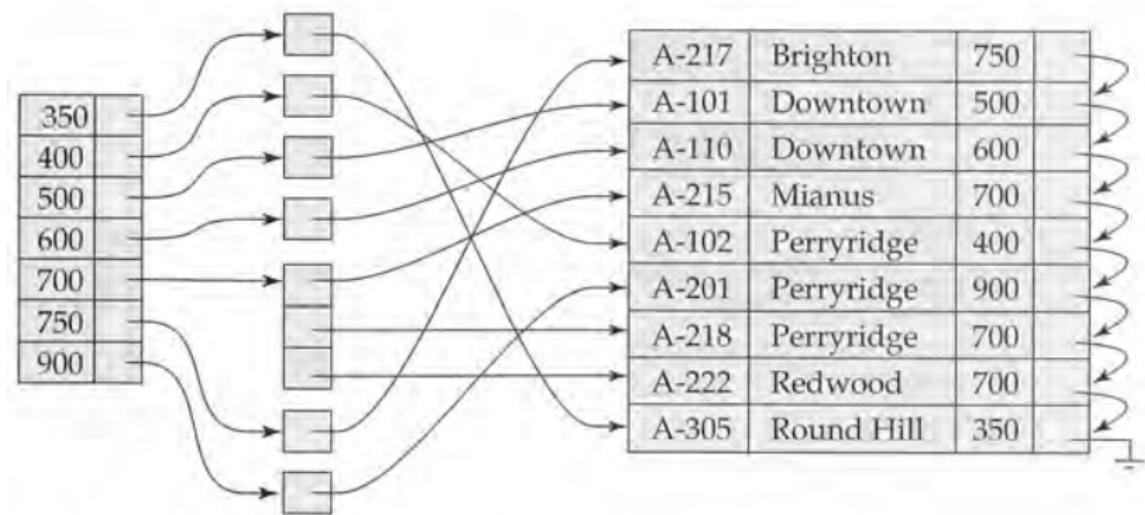


Figure 12.5 Secondary index on *account* file, on noncandidate key *balance*.

Must be dense (why?)

Two-level sparse index

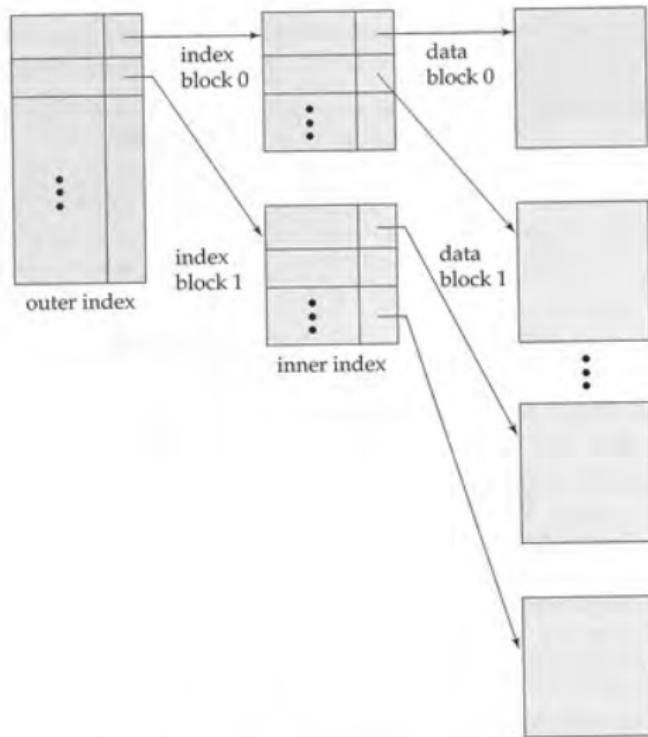


Figure 12.4 Two-level sparse index.

Ordered indexing

Evaluation criteria

- ▶ access types
- ▶ access cost
- ▶ update cost: insertion/deletion
- ▶ space overhead

Ordered indexing

- ▶ performance of index-sequential file organization degrades as the file grows, necessitating reorganization

Ordered indexing

- ▶ performance of index-sequential file organization degrades as the file grows, necessitating reorganization
- ▶ multi-level indexing as indexing an index
- ▶ isn't this a search tree?

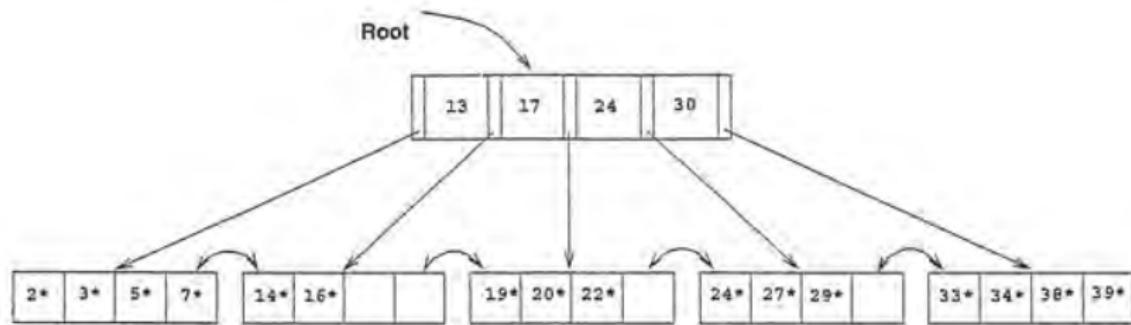
Ordered indexing: B+ trees

- ▶ height-balanced, dynamic search tree
- ▶ all $k*$ at the leaf level
- ▶ generalizes multi-level sparse index structure

Ordered indexing: B+ trees

- ▶ height-balanced, dynamic search tree
- ▶ all k^* at the leaf level
- ▶ generalizes multi-level sparse index structure
- ▶ leaf level can be sparse or dense
- ▶ designed to minimize I/O
- ▶ efficient equality and range search

Ordered indexing: B+ trees



B+ trees: structure

- ▶ node structure:
 $[P_1, K_1, P_2, K_2, \dots, P_{n-1}, K_{n-1}, P_n]$
- ▶ $n - 1$ search-key values K_i
- ▶ n pointers P_i
- ▶ if $i < j$, then $K_i < K_j$

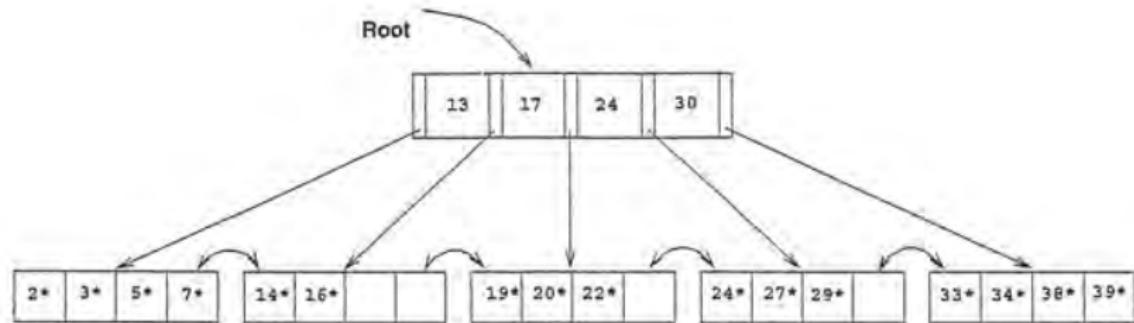
B+ trees: structure

- ▶ node structure:
 $[P_1, K_1, P_2, K_2, \dots, P_{n-1}, K_{n-1}, P_n]$
- ▶ $n - 1$ search-key values K_i
- ▶ n pointers P_i
- ▶ if $i < j$, then $K_i < K_j$
- ▶ leaf nodes:
 - ▶ P_i points to records with key value K_i
 - ▶ P_n points to next sibling leaf, forming *sequence set*, supporting sequential search
 - ▶ must contain at least $\lceil (n - 1)/2 \rceil$ search-key values

B+ trees: structure

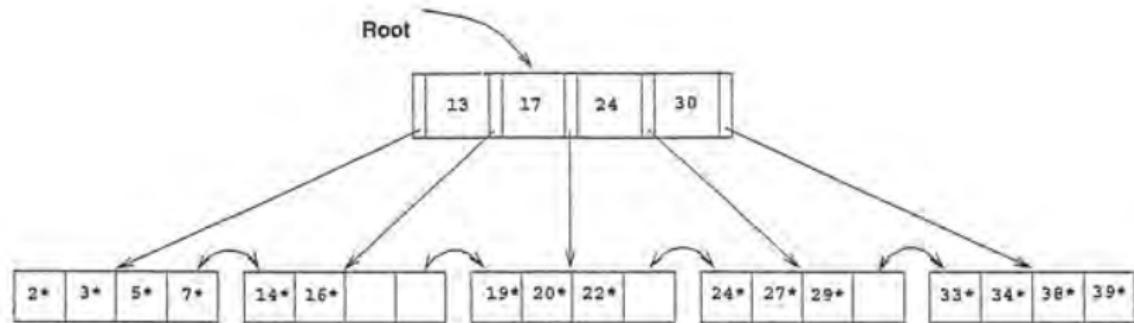
- ▶ node structure:
 $[P_1, K_1, P_2, K_2, \dots, P_{n-1}, K_{n-1}, P_n]$
- ▶ $n - 1$ search-key values K_i
- ▶ n pointers P_i
- ▶ if $i < j$, then $K_i < K_j$
- ▶ internal nodes:
 - ▶ P_i points to records with key value less than K_i
 - ▶ P_{i+1} points to records with key value greater than or equal to K_i
 - ▶ must contain at least $\lceil n/2 \rceil$ pointers (i.e., fanout)
 - ▶ except the root, which can have fewer, with a minimum of two pointers

B+ trees: structure



order $n = 5$

B+ trees: structure



order $n = 5$

- Internal nodes support random access
- Sequence set supports ordered access

B+ trees: search

Search(key k , node N)

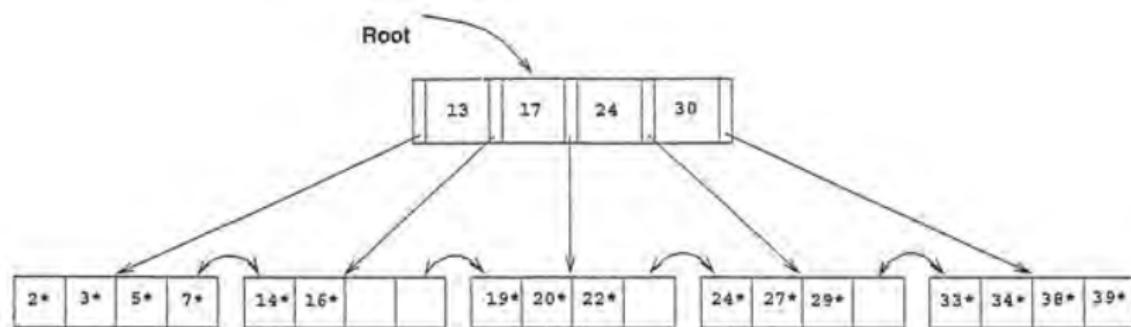
- ▶ if $k = K_i \in N$
 - ▶ if N is a leaf, return P_i
 - ▶ else return Search(k , P_{i+1})

B+ trees: search

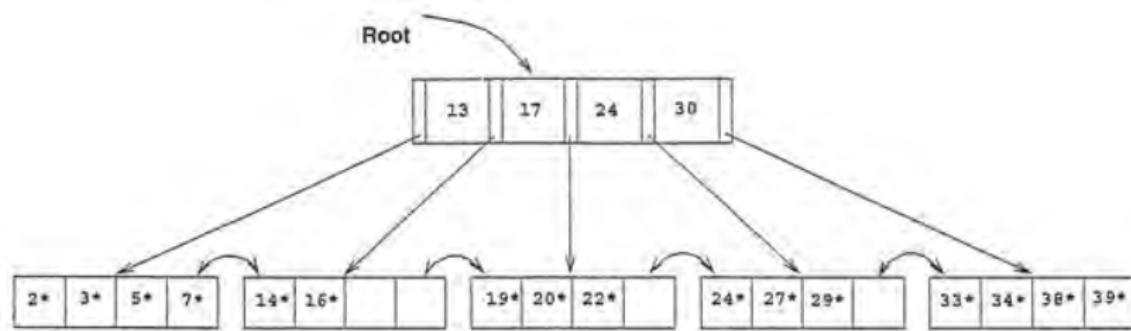
Search(key k , node N)

- ▶ if $k = K_i \in N$
 - ▶ if N is a leaf, return P_i
 - ▶ else return Search(k , P_{i+1})
- ▶ if $k \notin N$
 - ▶ if N is a leaf, return FAIL
 - ▶ else, find $K_i \in N$ such that K_i is minimal in node satisfying $k < K_i$, and return Search(k , P_i)

B+ trees: search

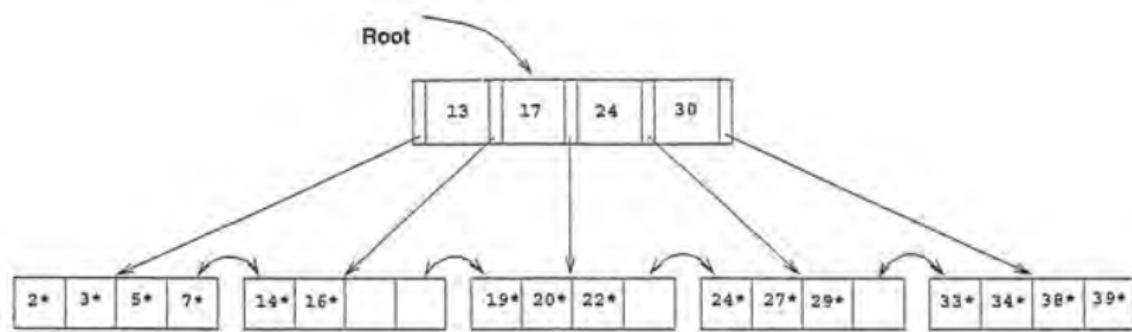


B+ trees: search



Search in $\mathcal{O}(\log_{\lceil n/2 \rceil} N)$, for tree of N search-key values.

B+ trees: search



Search in $\mathcal{O}(\log_{\lceil n/2 \rceil} N)$, for tree of N search-key values.

So, if $n = 100$ (a conservative estimate in practice), then locating an item in an index of $N = 1,000,000$ entries requires $\lceil \log_{50} 1,000,000 \rceil = 4$ I/Os.

B+ trees: insertion

Insert(key k , node N , record r)

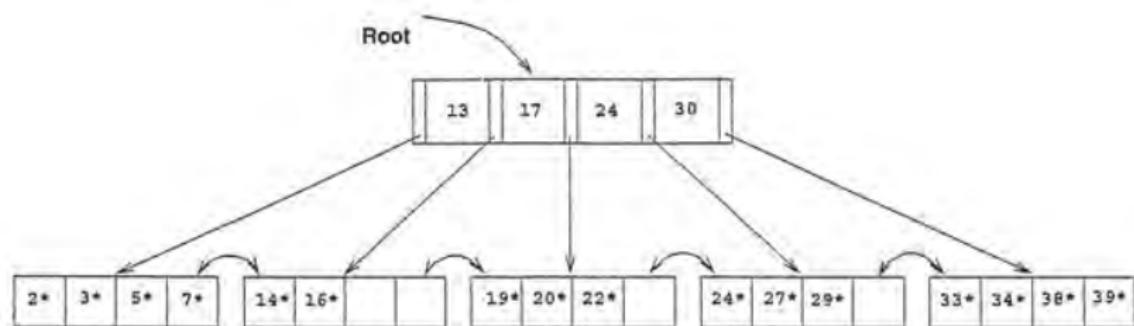
- ▶ if $k = K_i \in N$
 - ▶ if N is a leaf, return FAIL (no duplicates allowed)
 - ▶ else return Insert(k , P_{i+1} , r)

B+ trees: insertion

Insert(key k , node N , record r)

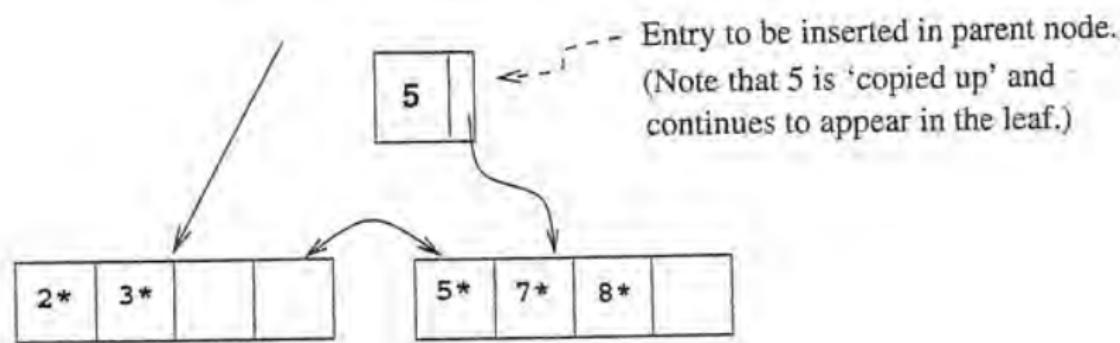
- ▶ if $k = K_i \in N$
 - ▶ if N is a leaf, return FAIL (no duplicates allowed)
 - ▶ else return Insert(k, P_{i+1}, r)
- ▶ if $k \notin N$
 - ▶ if N is a leaf
 - ▶ if there is space, insert k and r
 - ▶ else, split N into N and N' , redistribute entries evenly between them, and insert index entry for N' into parent of N
 - ▶ else, find $K_i \in N$ such that K_i is minimal in node satisfying $k < K_i$, and return Insert(k, P_i, r)

B+ trees: insertion



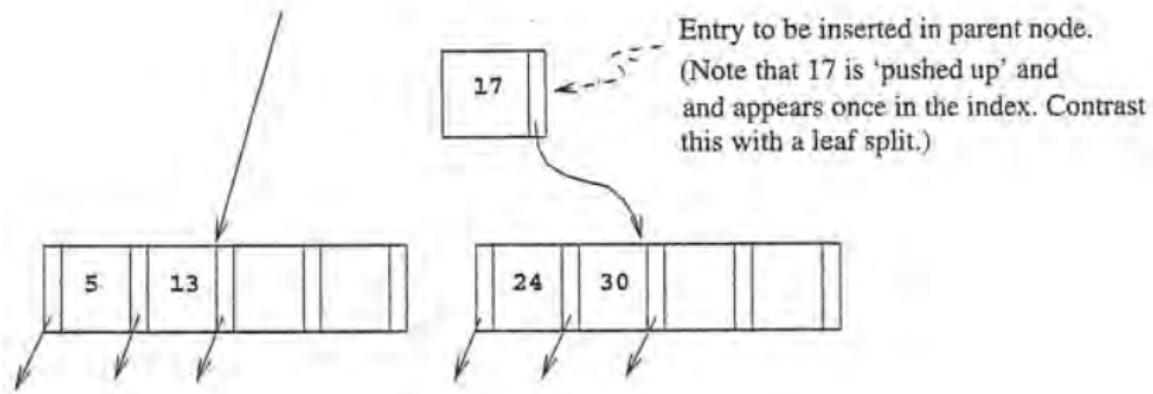
lets insert 8*

B+ trees: insertion



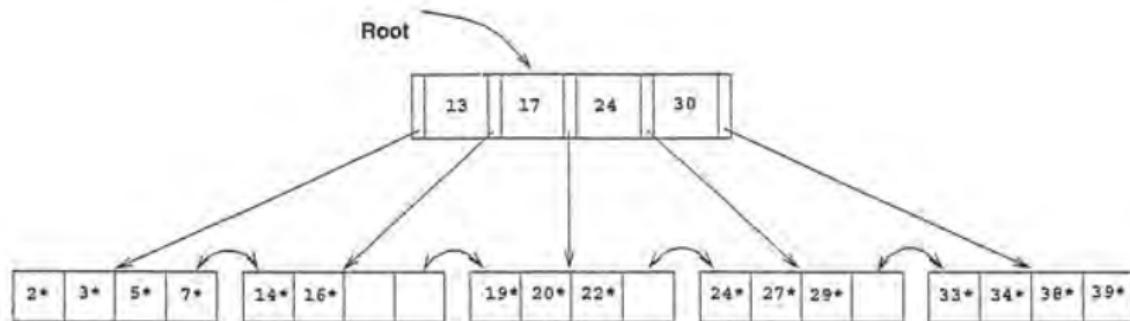
Split leaf pages during insertion of 8*

B+ trees: insertion



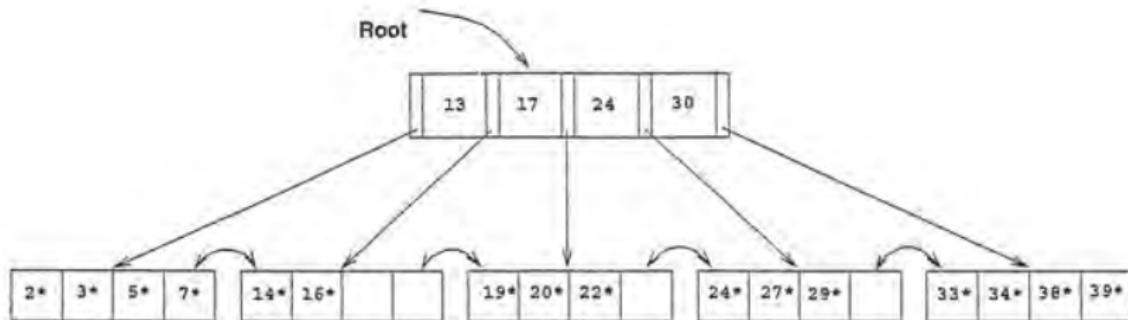
Split index pages during insertion of 8*

B+ trees: insertion

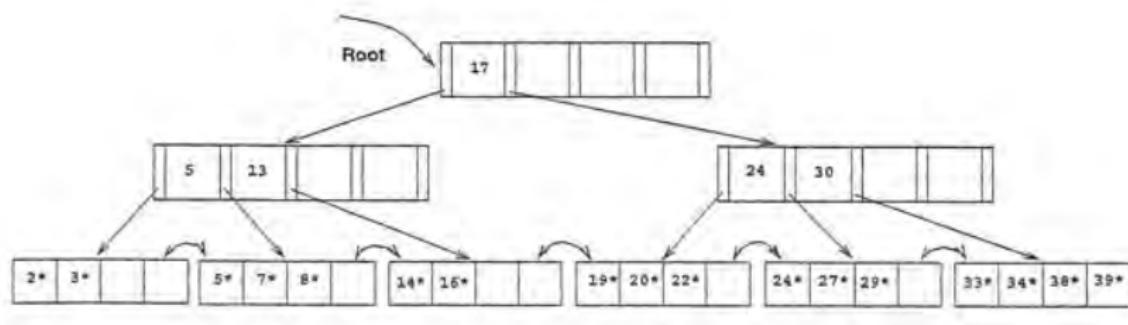


before ...

B+ trees: insertion



before ...

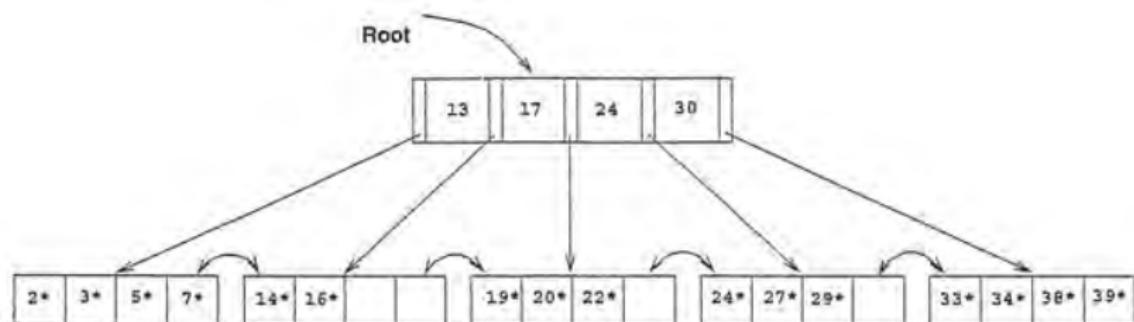


... and after inserting 8*

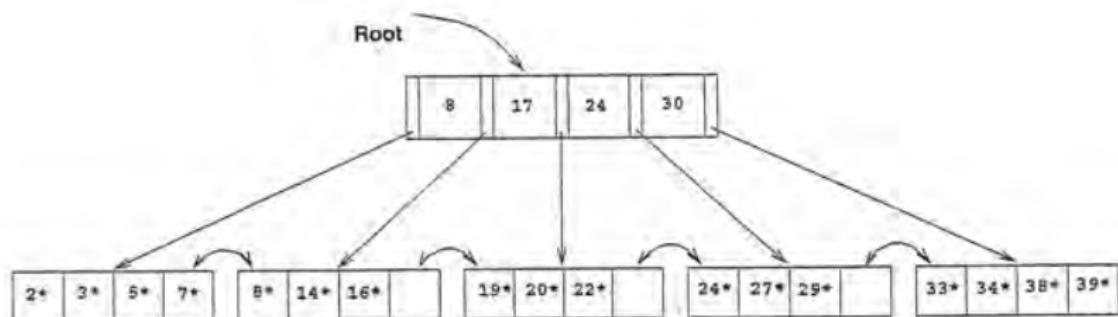
B+ trees: insertion

If space is available, we can also redistribute overflow to siblings, delaying (recursive) splitting

B+ trees: insertion



before ...



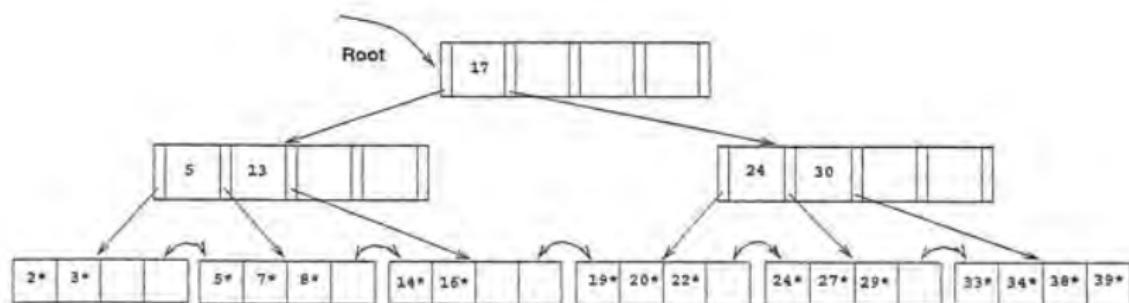
... and after inserting 8* using redistribution

B+ trees: deletion

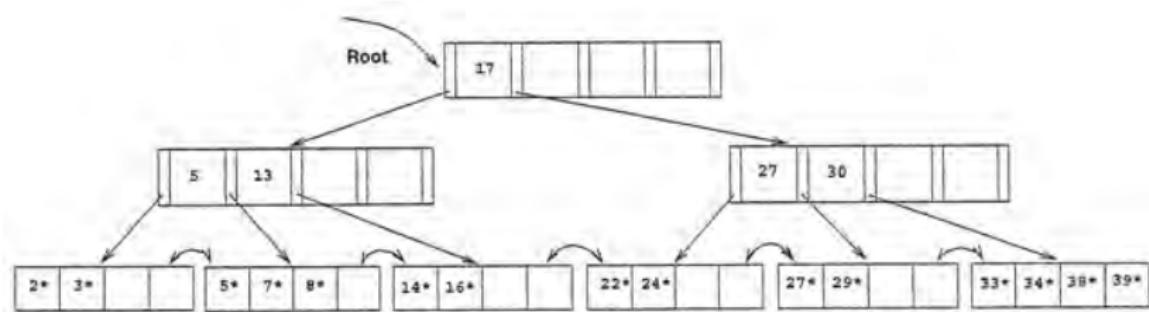
Delete(key k , node N)

- ▶ find the leaf node L where k belongs
 - ▶ remove entry
 - ▶ if L is still at least half full, return SUCCESS
 - ▶ else if possible, borrow entry from adjacent sibling
 - ▶ else, merge L and sibling, and delete entry for sibling from parent

B+ trees: deletion

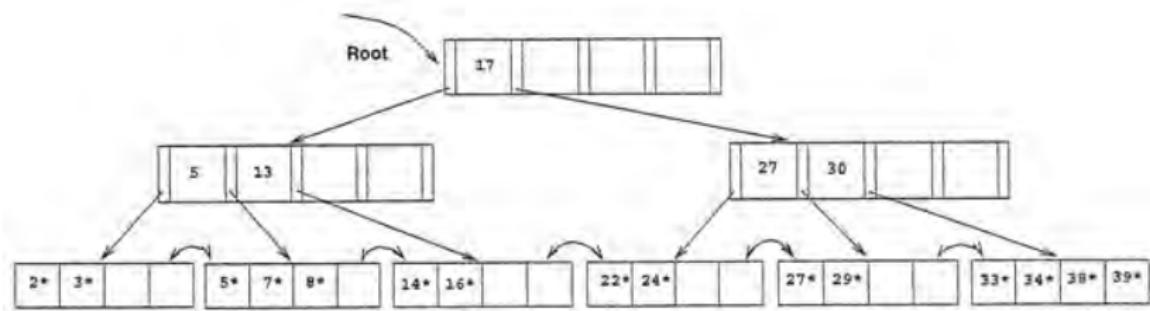


Before



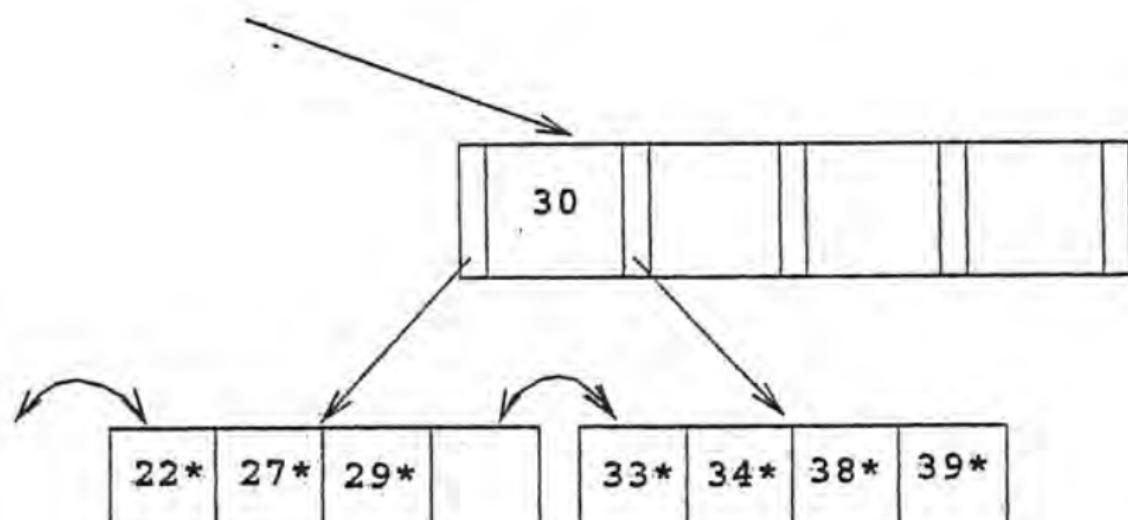
... and after deleting 19* and 20*

B+ trees: deletion



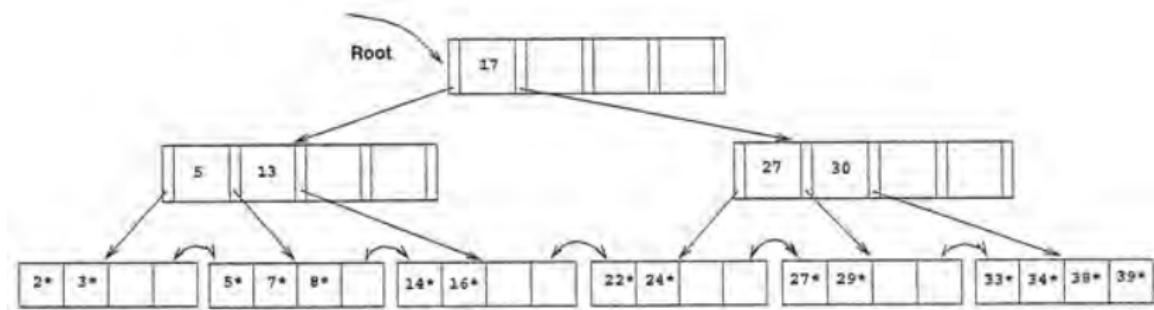
suppose we next delete 24*

B+ trees: deletion

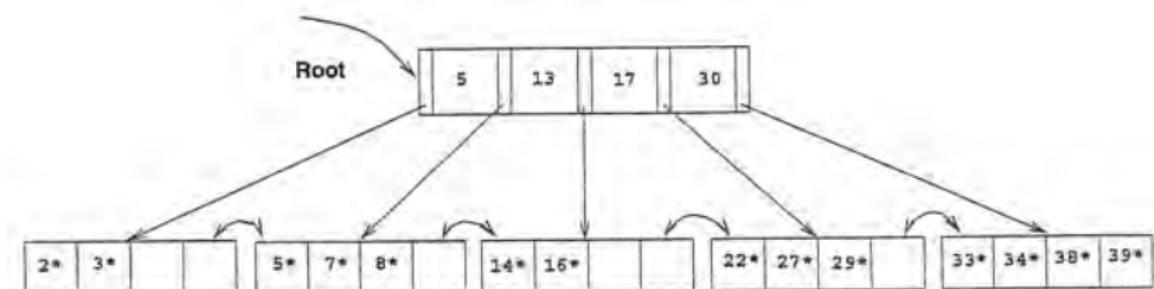


During deletion of 24*

B+ trees: deletion

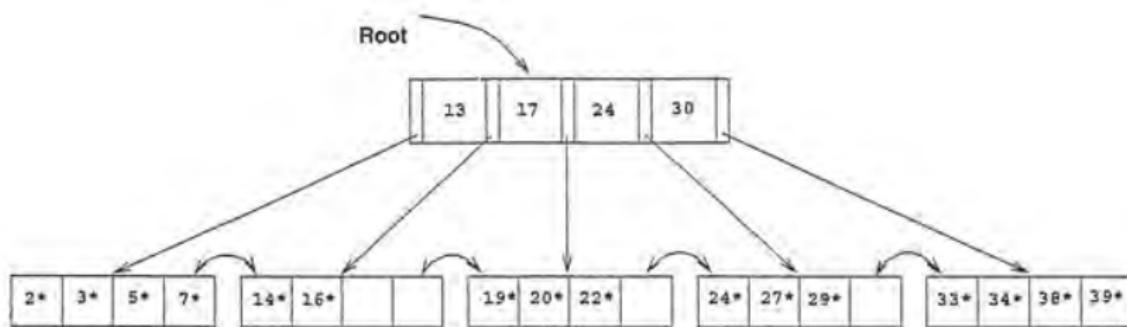


before ...



and after deletion of 24*

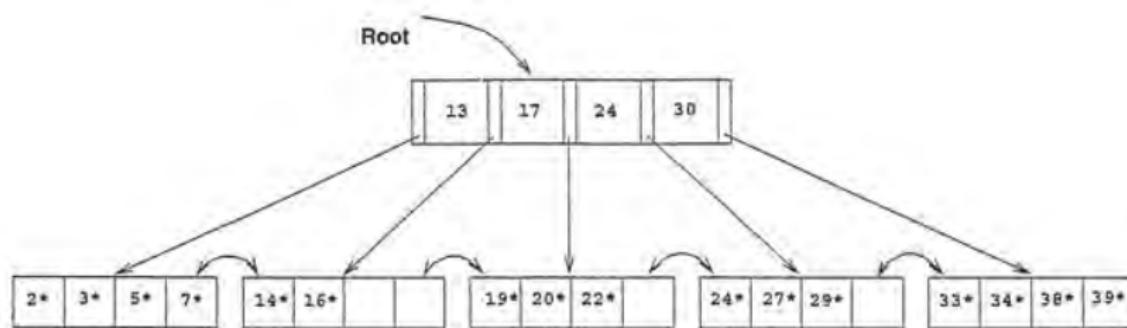
B+ trees



order $n = 5$

Exercise: Show the full tree that would result from inserting an item with key 6, with sibling redistribution.

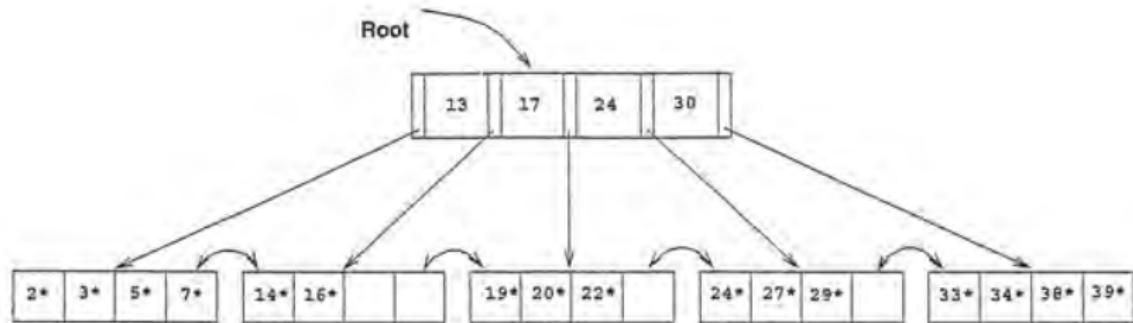
B+ trees



order $n = 5$

Exercise: Show the full tree that would result from inserting an item with key 6, **without** sibling redistribution.

B+ trees



order $n = 5$

Exercise: Show the full tree that would result from deleting item with key 14.

Recap

- ▶ Topic 1: Storage on disk & I/O model of computing
- ▶ Topic 2: External sorting
- ▶ Topic 3: Indexing, part 1 – ordered indexes

Recap

- ▶ Topic 1: Storage on disk & I/O model of computing
- ▶ Topic 2: External sorting
- ▶ Topic 3: Indexing, part 1 – ordered indexes

Next Wednesday: Ordered indexes, cont.

Next Friday: hash-based index data structures.

Please submit your paper selections **now**, if you haven't already done so.

Figure credits

- ▶ Our textbook (Silberschatz *et al.*, 2011)
- ▶ Ramakrishnan & Gehrke, 2003
- ▶ Garcia-Molina *et al.*, 2002

Indexing: ordered indexes, continued

Lecture 3
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

29 April 2015

Admin

Project part 2 will be posted later today. Once posted:

- ▶ find your team mates
- ▶ study your paper
- ▶ determine your research problem
- ▶ first report due on or before 10 May

Where we've been

Last time

- ▶ File structures

Where we've been

Last time

- ▶ File structures
- ▶ I/O model of computation

Where we've been

Last time

- ▶ File structures
- ▶ I/O model of computation
- ▶ external sorting

Where we've been

Last time

- ▶ File structures
- ▶ I/O model of computation
- ▶ external sorting
- ▶ B+ trees

Today's agenda

Ordered indexes, continued

- ▶ R trees
- ▶ GiST: generalized search trees

Ordered indexing: spatial data



Why spatial/multi-dimensional data?

- ▶ GIS, CAD design, multimedia, ...
- ▶ composite search keys are not enough

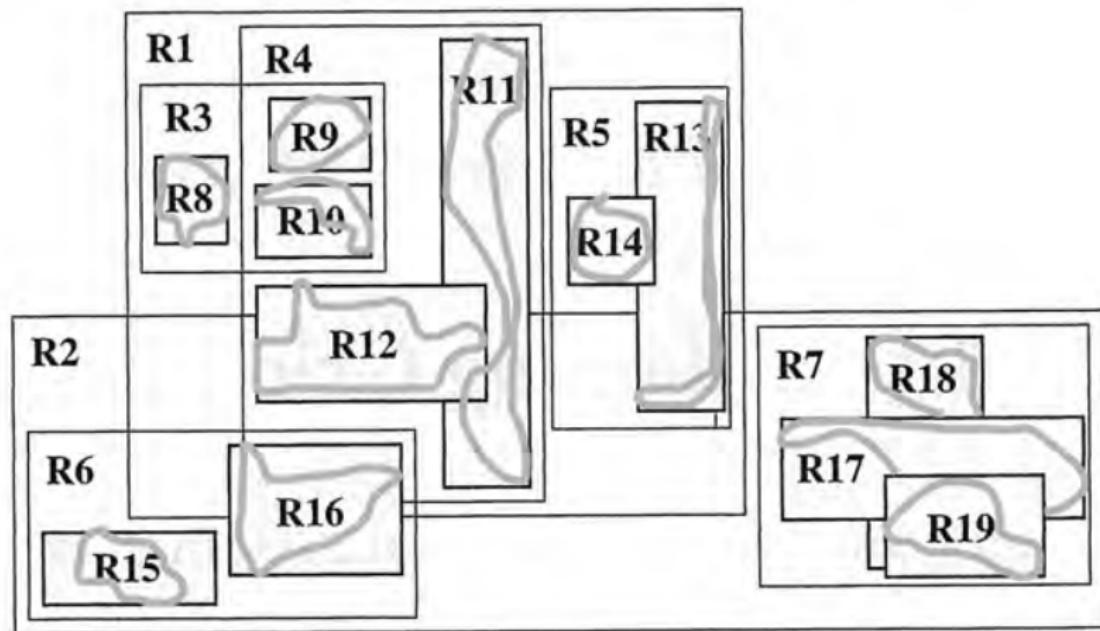
Ordered indexing: R-trees

- ▶ height-balanced, dynamic search tree
- ▶ designed to minimize I/O

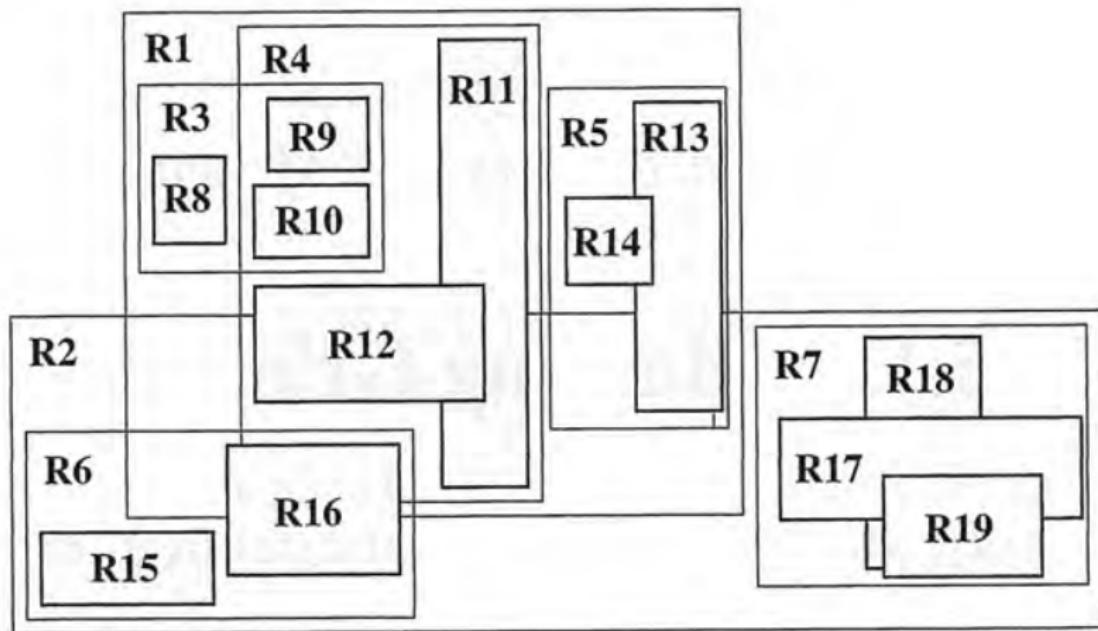
Ordered indexing: R-trees

- ▶ height-balanced, dynamic search tree
- ▶ designed to minimize I/O
- ▶ instead of range of values, a rectangular **bounding box** is associated with each node
- ▶ efficient spatial search

Ordered indexing: R-trees



Ordered indexing: R-trees



R-trees: structure

Properties

- ▶ leaf entry:
[n-dimensional bbox, pointer to record]
bounding box is tightest bounding box for data object

R-trees: structure

Properties

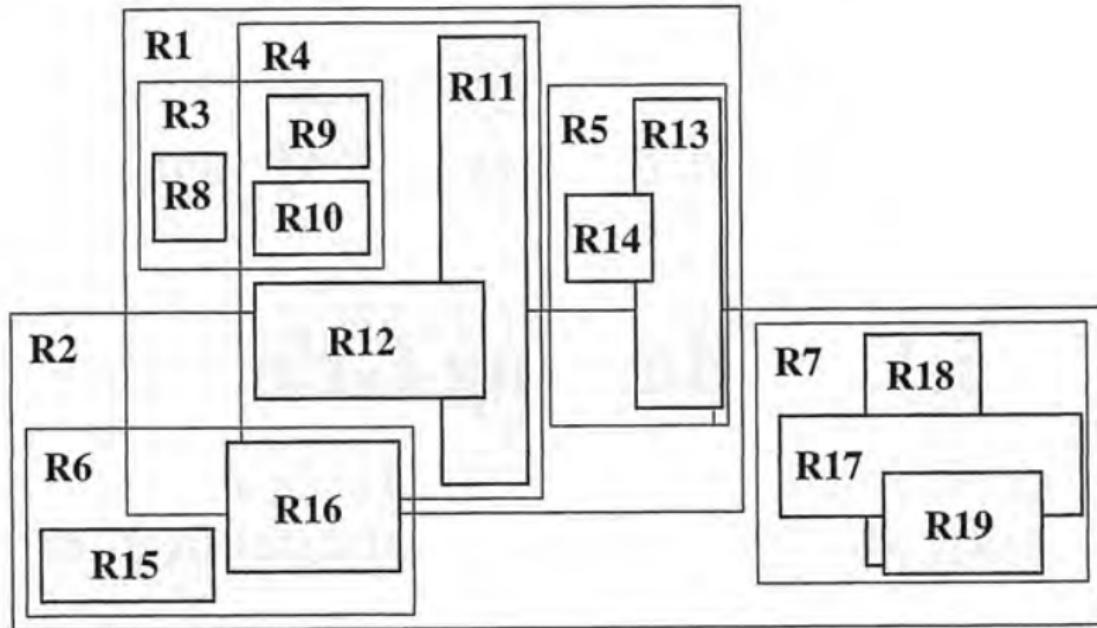
- ▶ leaf entry:
 $[n\text{-dimensional bbox}, \text{pointer to record}]$
bounding box is tightest bounding box for data object
- ▶ internal entry:
 $[n\text{-dimensional bbox}, \text{pointer to child node}]$
the box covers all boxes in subtree rooted at child

R-trees: structure

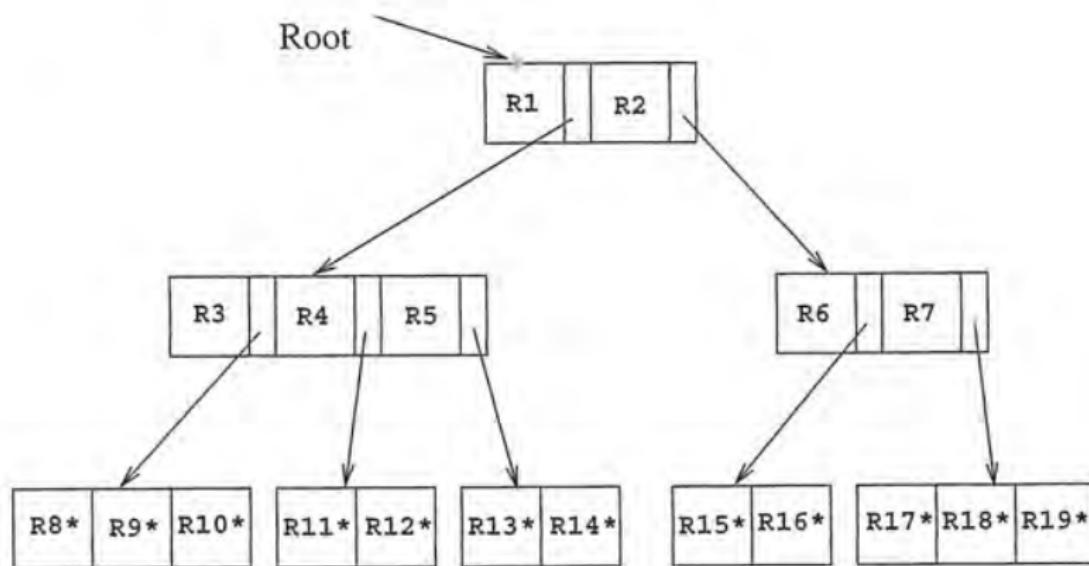
Properties

- ▶ leaf entry:
[n-dimensional bbox, pointer to record]
bounding box is tightest bounding box for data object
- ▶ internal entry:
[n-dimensional bbox, pointer to child node]
the box covers all boxes in subtree rooted at child
- ▶ same minimal occupancy rules as for B+ trees

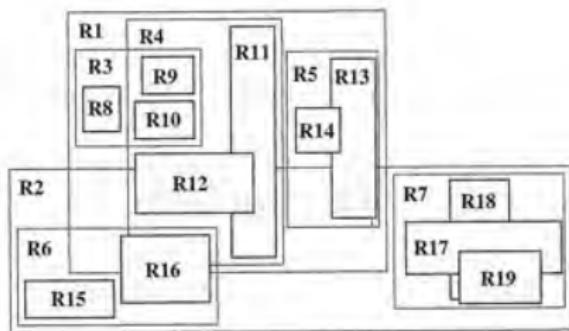
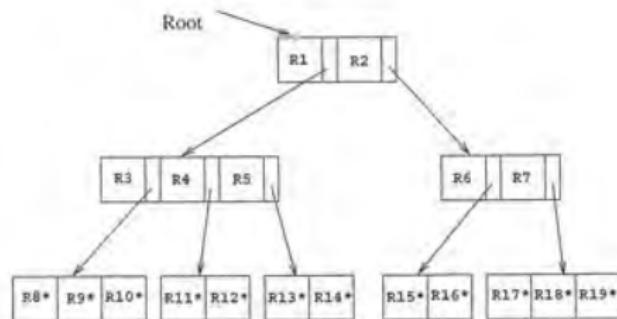
R-trees: structure



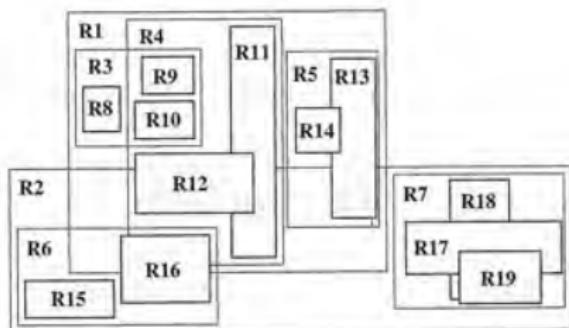
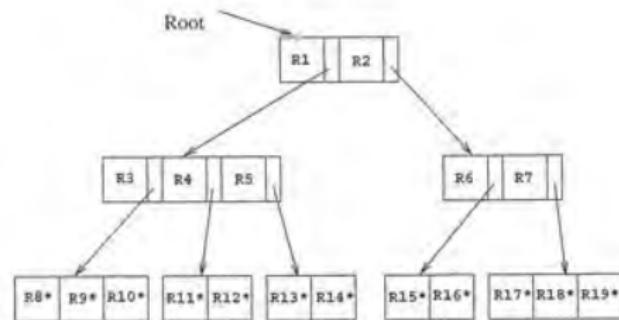
R-trees: structure



R-trees: structure



R-trees: structure



Note that, unlike B+ trees, we have no ordering on objects (and hence no ordering on tree nodes, siblings, etc.)

R-trees: search

Search for objects overlapping box B

- ▶ if current node is internal
 - ▶ for each entry $[E, \text{pointer}]$, if E overlaps B , search pointer

R-trees: search

Search for objects overlapping box B

- ▶ if current node is internal
 - ▶ for each entry $[E, \text{pointer}]$, if E overlaps B , search pointer
- ▶ if current node is leaf
 - ▶ for each entry $[E, \text{rid}]$, if E overlaps B , rid identifies an object which might overlap B

R-trees: search

Search for objects overlapping box B

- ▶ if current node is internal
 - ▶ for each entry $[E, \text{pointer}]$, if E overlaps B , search pointer
- ▶ if current node is leaf
 - ▶ for each entry $[E, \text{rid}]$, if E overlaps B , rid identifies an object which might overlap B

Note that, unlike B+ tree, we may have to search multiple subtrees

R-trees: insertion

Insert $[B, rid]$ in node N

- ▶ if N is internal
 - ▶ Insert $[B, rid]$ in child node C , where the bounding box associated with C needs the least enlargement to cover B

R-trees: insertion

Insert $[B, rid]$ in node N

- ▶ if N is internal
 - ▶ Insert $[B, rid]$ in child node C , where the bounding box associated with C needs the least enlargement to cover B
- ▶ if N is leaf
 - ▶ if N has space, insert
 - ▶ else, split N into N_1 and N_2 , adjust N 's parent to only cover new N_1 , and add entry for N_2

R-trees: deletion

Delete is handled as expected, except that under-full nodes are just deleted, and the remaining entries re-inserted.

R-trees

R-trees

- ▶ Very popular spatial indexing mechanism
(MySQL, Oracle, Postgres, ...)
- ▶ Intensively studied, leading to many variants
such as R* and R+ trees
 - ▶ **example:** Hilbert R-trees

R-trees: Hilbert variant

Hilbert R-trees

- ▶ lack of ordering has negative impact on clustering (i.e., spatial locality) of objects
 - ▶ leads to poor space utilization (~70%)

R-trees: Hilbert variant

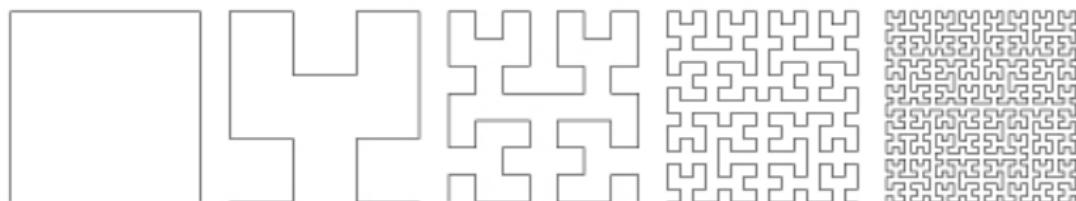
Hilbert R-trees

- ▶ lack of ordering has negative impact on clustering (i.e., spatial locality) of objects
 - ▶ leads to poor space utilization (~70%)
- ▶ *key idea of Hilbert R-trees:* add an ordering to objects (and hence on nodes in the tree)
 - ▶ leads to excellent space utilization (~100%) and search/update costs (up to ~30% fewer page accesses)

R-trees: Hilbert variant

Hilbert curves

- ▶ example of a “space filling” curve, proposed by Hilbert in 1891
- ▶ imposes a linear ordering on the points in the unit n-cube
- ▶ has very good spatial locality



R-trees: Hilbert variant

Ordering bounding boxes with Hilbert curves

- ▶ identify each bounding box with the Hilbert curve value of the center point of the box
 - ▶ i.e., its position on the curve

R-trees: Hilbert variant

Ordering bounding boxes with Hilbert curves

- ▶ identify each bounding box with the Hilbert curve value of the center point of the box
 - ▶ i.e., its position on the curve
- ▶ extend each entry in non-leaf nodes to also include the largest Hilbert value (LHV) among the boxes contained in the subtree rooted at the entry

[LHV, n-dimensional bbox, pointer to child node]

R-trees: Hilbert variant

Ordering bounding boxes with Hilbert curves

- ▶ identify each bounding box with the Hilbert curve value of the center point of the box
 - ▶ i.e., its position on the curve
- ▶ extend each entry in non-leaf nodes to also include the largest Hilbert value (LHV) among the boxes contained in the subtree rooted at the entry
 - [*LHV, n-dimensional bbox, pointer to child node*]
- ▶ Search as usual; insertion/deletion are guided by Hilbert value of the inserted/deleted box
 - ▶ also, Hilbert ordering can be effectively used for bulk-loading/building the index

Search trees, in general

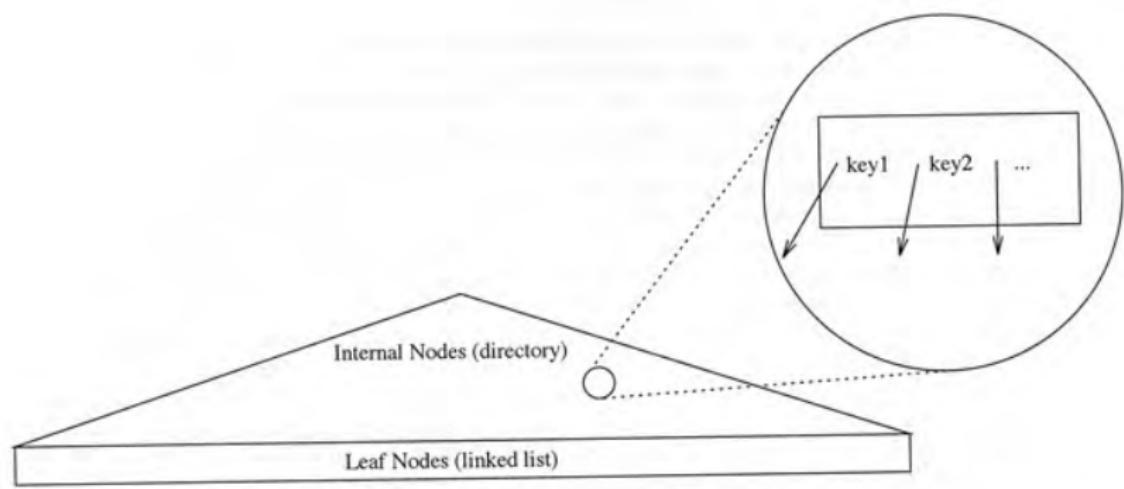
There has been an explosion in types of data:
geographic data, multimedia, CAD, document
libraries, sequence data, fingerprint data,
biochemical and bioinformatics data,

Search trees, in general

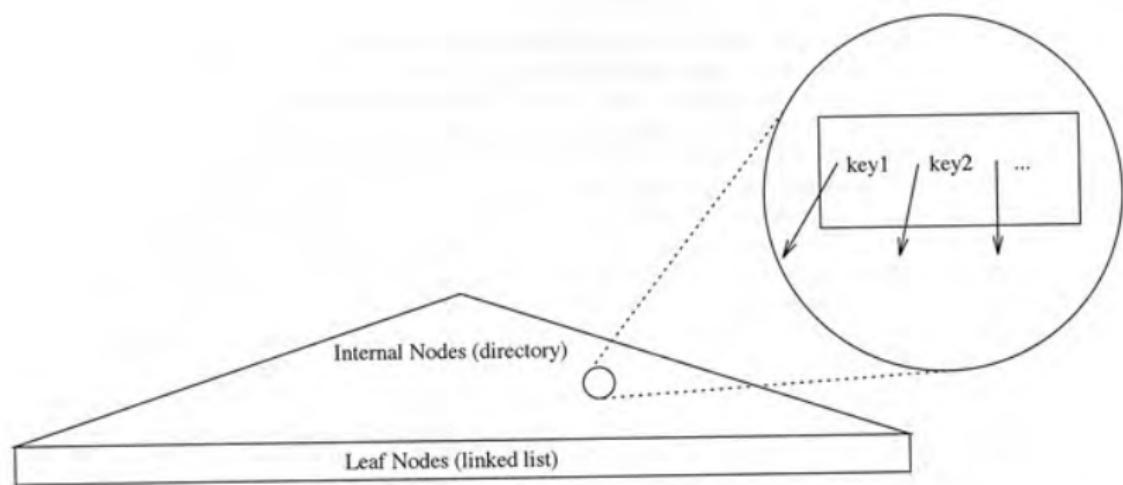
Consequently, there has been an explosion of tree-based indexing techniques ...

often based on specialized search trees, or search trees over extensible data types.

Search trees, in general



Search trees, in general



What can we say in general about tree-based ordered indexes? How far can we generalize this structure?

GiST: generalized search trees

Proposal for generic tree-index infrastructure

- ▶ a *search key* may be any arbitrary predicate q that holds for each datum below the key

GiST: generalized search trees

Proposal for generic tree-index infrastructure

- ▶ a *search key* may be any arbitrary predicate q that holds for each datum below the key
- ▶ a *search tree* is a hierarchy of partitions of a dataset, in which each partition has a categorization that holds for all data in the partition.

GiST: generalized search trees

Proposal for generic tree-index infrastructure

- ▶ a *search key* may be any arbitrary predicate q that holds for each datum below the key
- ▶ a *search tree* is a hierarchy of partitions of a dataset, in which each partition has a categorization that holds for all data in the partition.

Keys are user-defined (e.g., integers, bounding boxes, etc.).

GiST: generalized search trees

Proposal for generic tree-index infrastructure

- ▶ a *search key* may be any arbitrary predicate q that holds for each datum below the key
- ▶ a *search tree* is a hierarchy of partitions of a dataset, in which each partition has a categorization that holds for all data in the partition.

Keys are user-defined (e.g., integers, bounding boxes, etc.).

Available as part of PostgreSQL

GiST: generalized search trees

Properties

- ▶ every node has a minimal occupancy

GiST: generalized search trees

Properties

- ▶ every node has a minimal occupancy
- ▶ for each index entry (p, ptr) in a leaf node, p is true when instantiated with the values from the tuple indicated by ptr

GiST: generalized search trees

Properties

- ▶ every node has a minimal occupancy
- ▶ for each index entry (p, ptr) in a leaf node, p is true when instantiated with the values from the tuple indicated by ptr
- ▶ for each index entry (p, ptr) in an internal node, p is true when instantiated with the values from any tuple reachable from ptr

GiST: generalized search trees

Properties

- ▶ every node has a minimal occupancy
- ▶ for each index entry (p, ptr) in a leaf node, p is true when instantiated with the values from the tuple indicated by ptr
- ▶ for each index entry (p, ptr) in an internal node, p is true when instantiated with the values from any tuple reachable from ptr
- ▶ the root has at least two children unless it is a leaf
- ▶ all leaves appear on the same level

GiST: generalized search trees

Key designer must define the following methods

- ▶ **Consistent(E, q)**, for entry $E = (p, \text{ptr})$ and search predicate q

GiST: generalized search trees

Key designer must define the following methods

- ▶ **Consistent**(E, q), for entry $E = (p, \text{ptr})$ and search predicate q
- ▶ **Union**(P), for list of entries P , return a new predicate satisfied by all members of P

GiST: generalized search trees

Key designer must define the following methods

- ▶ **Consistent**(E, q), for entry $E = (p, \text{ptr})$ and search predicate q
- ▶ **Union**(P), for list of entries P , return a new predicate satisfied by all members of P
- ▶ **Compress**(E) and **Decompress**(E),
(de)compressed representation for p , for insertion/reading of E into/from a node

GiST: generalized search trees

Key designer must define the following methods

- ▶ **Consistent**(E, q), for entry $E = (p, \text{ptr})$ and search predicate q
- ▶ **Union**(P), for list of entries P , return a new predicate satisfied by all members of P
- ▶ **Compress**(E) and **Decompress**(E),
(de)compressed representation for p , for insertion/reading of E into/from a node
- ▶ **Penalty**(E_1, E_2), for inserting E_2 into subtree at E_1

GiST: generalized search trees

Key designer must define the following methods

- ▶ **Consistent**(E, q), for entry $E = (p, \text{ptr})$ and search predicate q
- ▶ **Union**(P), for list of entries P , return a new predicate satisfied by all members of P
- ▶ **Compress**(E) and **Decompress**(E),
(de)compressed representation for p , for insertion/reading of E into/from a node
- ▶ **Penalty**(E_1, E_2), for inserting E_2 into subtree at E_1
- ▶ **PickSplit**(P), into two new lists P_1 and P_2 , maintaining occupancy requirements

GiST: generalized search trees

Basic infrastructure:

- ▶ Search
- ▶ Search on linearly ordered domains
 - ▶ using FindMin and Next

GiST: generalized search trees

Basic infrastructure:

- ▶ Search
- ▶ Search on linearly ordered domains
 - ▶ using FindMin and Next
- ▶ Insert
 - ▶ using ChooseSubTree, Split, and AdjustKeys

GiST: generalized search trees

Basic infrastructure:

- ▶ Search
- ▶ Search on linearly ordered domains
 - ▶ using FindMin and Next
- ▶ Insert
 - ▶ using ChooseSubTree, Split, and AdjustKeys
- ▶ Delete

GiST: search

Algorithm Search(R, q)

Input: GiST rooted at R , predicate q

Output: all tuples that satisfy q

Sketch: Recursively descend all paths in tree whose keys are consistent with q .

S1: [Search subtrees] If R is not a leaf, check each entry E on R to determine whether Consistent(E, q). For all entries that are Consistent, invoke Search on the subtree whose root node is referenced by $E.\text{ptr}$.

S2: [Search leaf node] If R is a leaf, check each entry E on R to determine whether Consistent(E, q). If E is Consistent, it is a qualifying entry. At this point $E.\text{ptr}$ could be fetched to check q accurately, or this check could be left to the calling process.

GiST: search on linearly ordered domains

Algorithm $\text{FindMin}(R, q)$

Input: GiST rooted at R , predicate q

Output: minimum tuple in linear order that satisfies q

Sketch: descend leftmost branch of tree whose keys are Consistent with q . When a leaf node is reached, return the first key that is Consistent with q .

FM1: [Search subtrees] If R is not a leaf, find the first entry E in order such that $\text{Consistent}(E, q)^1$. If such an E can be found, invoke FindMin on the subtree whose root node is referenced by $E.\text{ptr}$. If no such entry is found, return **NULL**.

FM2: [Search leaf node] If R is a leaf, find the first entry E on R such that $\text{Consistent}(E, q)$, and return E . If no such entry exists, return **NULL**.

GiST: search on linearly ordered domains

Algorithm $\text{Next}(R, q, E)$

Input: GiST rooted at R , predicate q , current entry E

Output: next entry in linear order that satisfies q

Sketch: return next entry on the same level of the tree if it satisfies q . Else return NULL.

N1: [next on node] If E is not the rightmost entry on its node, and N is the next entry to the right of E in order, and $\text{Consistent}(N, q)$, then return N . If $\neg\text{Consistent}(N, q)$, return NULL.

N2: [next on neighboring node] If E is the rightmost entry on its node, let P be the next node to the right of R on the same level of the tree (this can be found via tree traversal, or via sideways pointers in the tree, when available [LY81].) If P is non-existent, return NULL. Otherwise, let N be the leftmost entry on P . If $\text{Consistent}(N, q)$, then return N , else return NULL.

GiST: insertion

Algorithm $\text{Insert}(R, E, l)$

Input: GiST rooted at R , entry $E = (p, \text{ptr})$, and level l , where p is a predicate such that p holds for all tuples reachable from ptr .

Output: new GiST resulting from insert of E at level l .

Sketch: find where E should go, and add it there, splitting if necessary to make room.

11. [invoke ChooseSubtree to find where E should go] Let $L = \text{ChooseSubtree}(R, E, l)$
12. If there is room for E on L , install E on L (in order according to Compare , if $\text{IsOrdered}.$)
Otherwise invoke $\text{Split}(R, L, E)$.
13. [propagate changes upward]
 $\text{AdjustKeys}(R, L)$.

GiST: insertion

Algorithm ChooseSubtree(R, E, l)

Input: subtree rooted at R , entry $E = (p, \text{ptr})$, level l

Output: node at level l best suited to hold entry with characteristic predicate $E.p$

Sketch: Recursively descend tree minimizing Penalty

CS1. If R is at level l , return R ;

CS2. Else among all entries $F = (q, \text{ptr}')$ on R find the one such that $\text{Penalty}(F, E)$ is minimal. Return $\text{ChooseSubtree}(F.\text{ptr}', E, l)$.

GiST: insertion

Algorithm Split(R, N, E)

Input: GiST R with node N , and a new entry $E = (p, \text{ptr})$.

Output: the GiST with N split in two and E inserted.

Sketch: split keys of N along with E into two groups according to PickSplit. Put one group onto a new node, and Insert the new node into the parent of N .

SP1: Invoke PickSplit on the union of the elements of N and $\{E\}$, put one of the two partitions on node N , and put the remaining partition on a new node N' .

SP2: [Insert entry for N' in parent] Let $E_{N'} = (q, \text{ptr}')$, where q is the Union of all entries on N' , and ptr' is a pointer to N' . If there is room for $E_{N'}$ on Parent(N), install $E_{N'}$ on Parent(N) (in order if IsOrdered.) Otherwise invoke Split($R, \text{Parent}(N), E_{N'}$)².

SP3: Modify the entry F which points to N , so that $F.p$ is the Union of all entries on N .

GiST: insertion

Algorithm $\text{AdjustKeys}(R, N)$

Input: GiST rooted at R , tree node N

Output: the GiST with ancestors of N containing correct and specific keys

Sketch: ascend parents from N in the tree, making the predicates be accurate characterizations of the subtrees. Stop after root, or when a predicate is found that is already accurate.

PR1: If N is the root, or the entry which points to N has an already-accurate representation of the Union of the entries on N , then return.

PR2: Otherwise, modify the entry E which points to N so that $E.p$ is the Union of all entries on N . Then $\text{AdjustKeys}(R, \text{Parent}(N))$.

GiST: deletion

Deletion (1) maintains balance of tree, and (2) keeps keys as specific as possible.

Follows B+tree style on linearly ordered domains, and otherwise R-tree style.

GiST

Generalizes a wide variety of indexing proposals.

GiST

Generalizes a wide variety of indexing proposals.

Permits DB engine designers to focus on core extensible infrastructure.

Promotes deeper study of indexing

Wrap up

- ▶ Indexing, part 1: ordered indexes
 - ▶ B+ trees
 - ▶ R trees
 - ▶ GiST: generalized search trees

Wrap up

- ▶ Indexing, part 1: ordered indexes
 - ▶ B+ trees
 - ▶ R trees
 - ▶ GiST: generalized search trees
- ▶ next time: Indexing, part 2: hash indexes and indexability

Credits

- ▶ Our textbook (Silberschatz *et al.*, 2011)
- ▶ Ramakrishnan & Gehrke, 2003
- ▶ Hellerstein *et al.*, 1995

Indexing 2: hash indexes, join indexes, and models of indexability

Lecture 4
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

1 May 2015

Where we've been

Last time

- ▶ basics of indexing

Where we've been

Last time

- ▶ basics of indexing
 - ▶ clustering vs. nonclustering
 - ▶ dense vs. sparse
 - ▶ ordered vs. hashed

Where we've been

Last time

- ▶ basics of indexing
 - ▶ clustering vs. nonclustering
 - ▶ dense vs. sparse
 - ▶ ordered vs. hashed
- ▶ ordered indexes: B+ trees
 - ▶ supports range search, with only logarithmic overhead

Where we've been

Last time

- ▶ basics of indexing
 - ▶ clustering vs. nonclustering
 - ▶ dense vs. sparse
 - ▶ ordered vs. hashed
- ▶ ordered indexes: B+ trees
 - ▶ supports range search, with only logarithmic overhead
- ▶ ordered indexes: R trees
 - ▶ spatial search, with only logarithmic overhead

Where we've been

Last time

- ▶ basics of indexing
 - ▶ clustering vs. nonclustering
 - ▶ dense vs. sparse
 - ▶ ordered vs. hashed
- ▶ ordered indexes: B+ trees
 - ▶ supports range search, with only logarithmic overhead
- ▶ ordered indexes: R trees
 - ▶ spatial search, with only logarithmic overhead
- ▶ ordered indexes: GiST
 - ▶ extensible generalized search-tree infrastructure

Where we're going

Today's agenda

- ▶ Indexing, part 2
 - ▶ hash-based indexing

Where we're going

Today's agenda

- ▶ Indexing, part 2
 - ▶ hash-based indexing
 - ▶ join indexing

Where we're going

Today's agenda

- ▶ Indexing, part 2
 - ▶ hash-based indexing
 - ▶ join indexing
 - ▶ models of indexability

Indexing

Two basic types of indexes

- ▶ Ordered indexes. based on a sorted ordering of the key values
- ▶ **Hash indexes.** based on a uniform distribution of key values across a finite number B of buckets. The bucket to which a value is assigned is determined by a hash function.
 - ▶ i.e., a function h which assigns each element k of the key space to an integer $h(k) \in \{0, \dots, B - 1\}$

Indexing

Two basic types of indexes

- ▶ Ordered indexes. based on a sorted ordering of the key values
- ▶ Hash indexes. based on a uniform distribution of key values across a finite number B of buckets. The bucket to which a value is assigned is determined by a hash function.
 - ▶ i.e., a function h which assigns each element k of the key space to an integer $h(k) \in \{0, \dots, B - 1\}$
 - ▶ e.g., for strings, the sum over integer representations of each character, modulo B

Static hashing

Static hashing: hash file

A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Static hashing: hash file

A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

bucket 0



bucket 1



bucket 2



bucket 3

A-217	Brighton	750
A-305	Round Hill	350

bucket 4

A-222	Redwood	700

bucket 5

A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700

bucket 6



bucket 7

A-215	Mianus	700

bucket 8

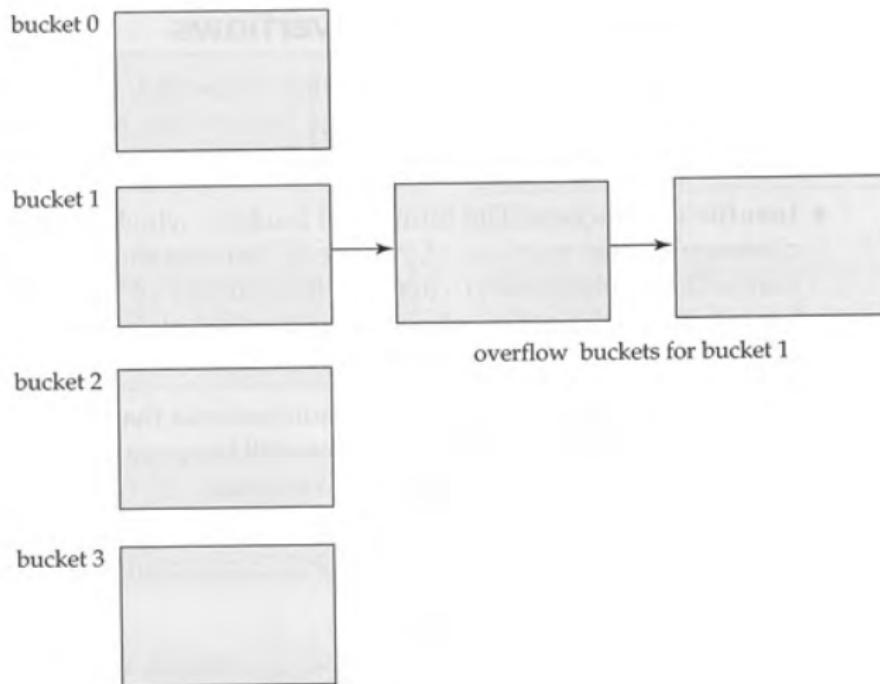
A-101	Downtown	500
A-110	Downtown	600

bucket 9

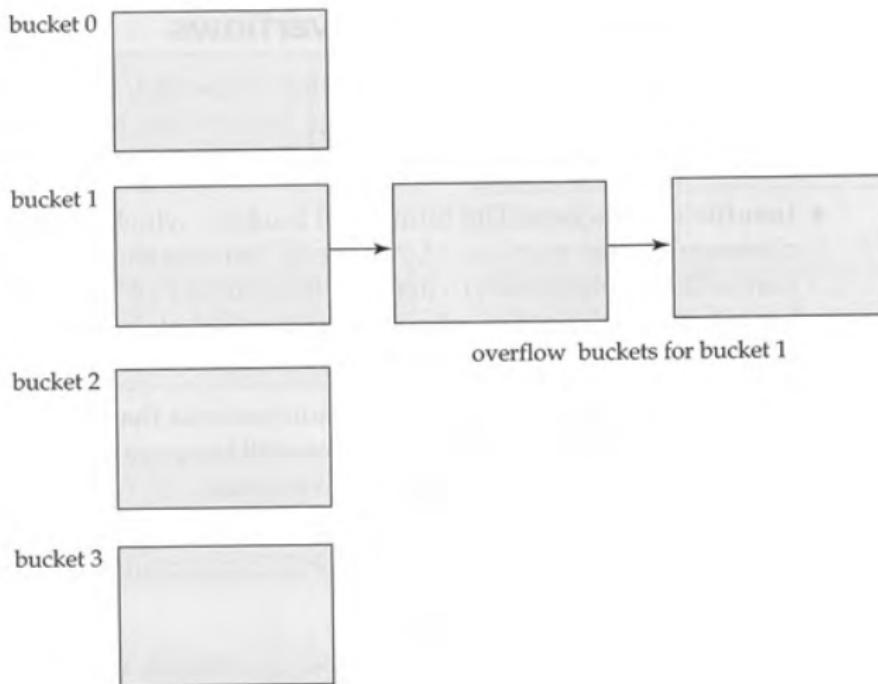


with branch name as key

Static hashing: overflow chaining (closed hashing)

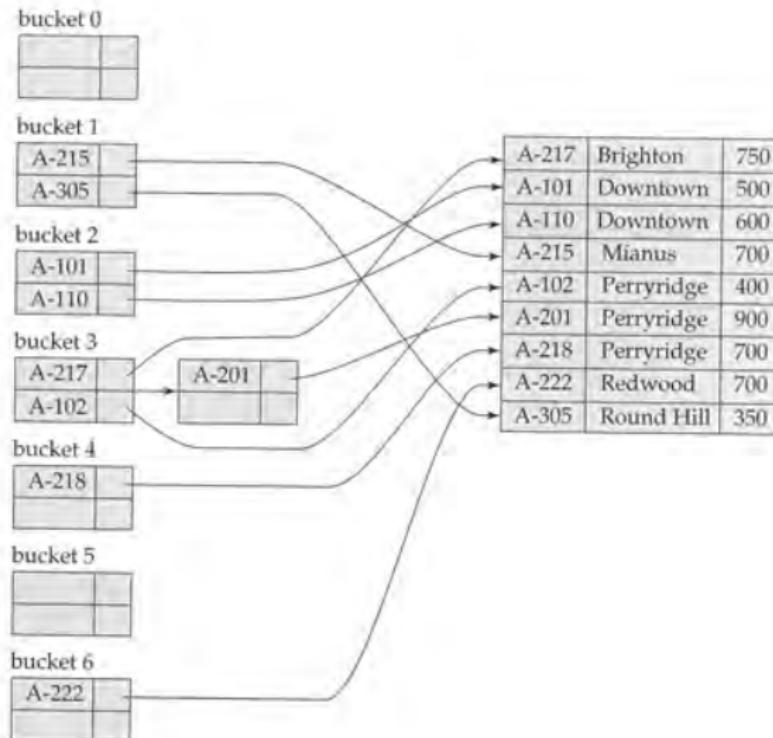


Static hashing: overflow chaining (closed hashing)



vs. open hashing (e.g., linear probing)

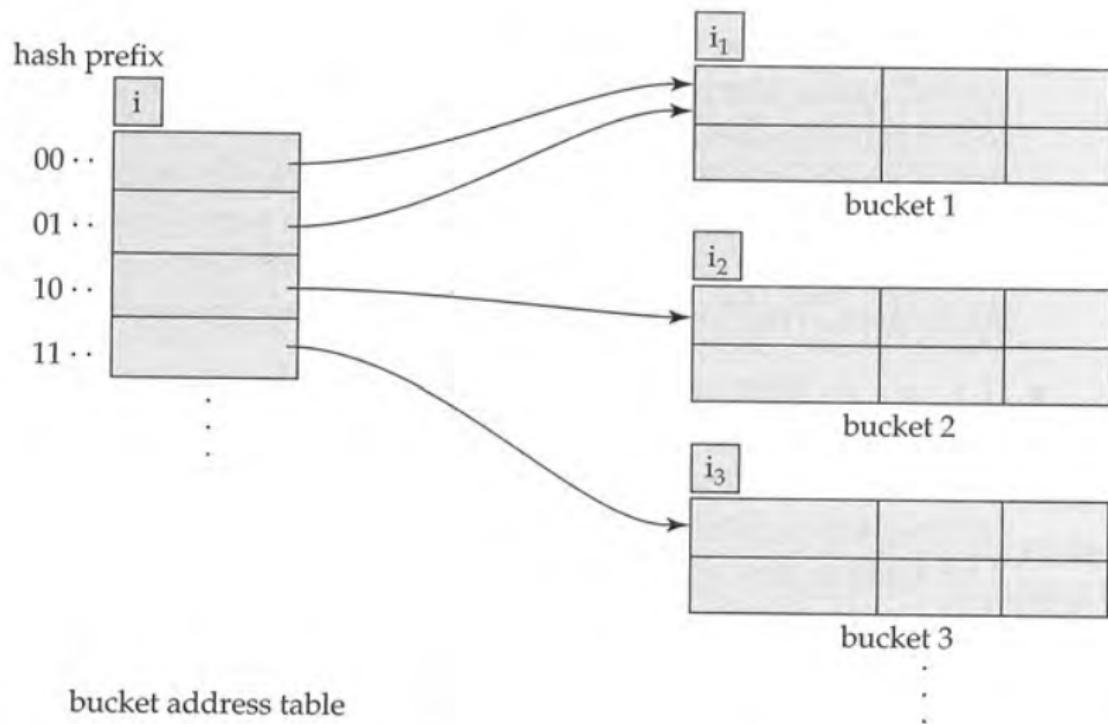
Static hashing: hash-based indexing



secondary index on account number (hash value: sum of digits modulo 7)

Dynamic hashing

Dynamic hashing: extendable hash index



Dynamic hashing: extendable hash index

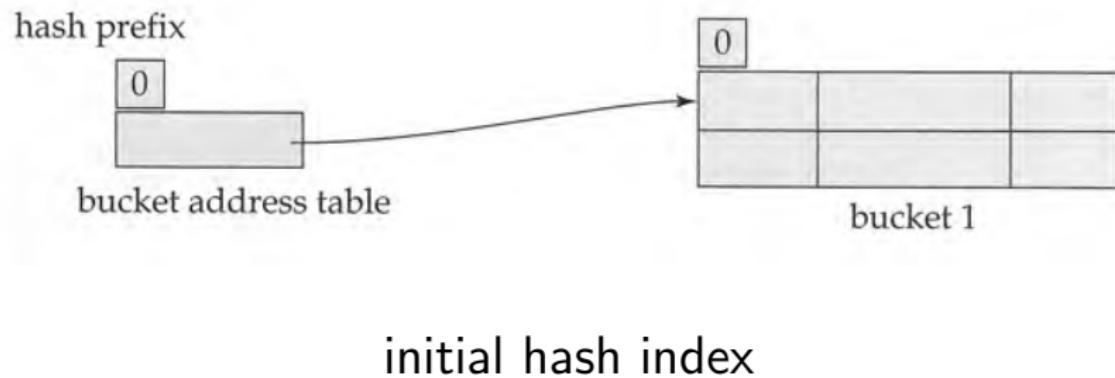
A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Dynamic hashing: extendable hash index

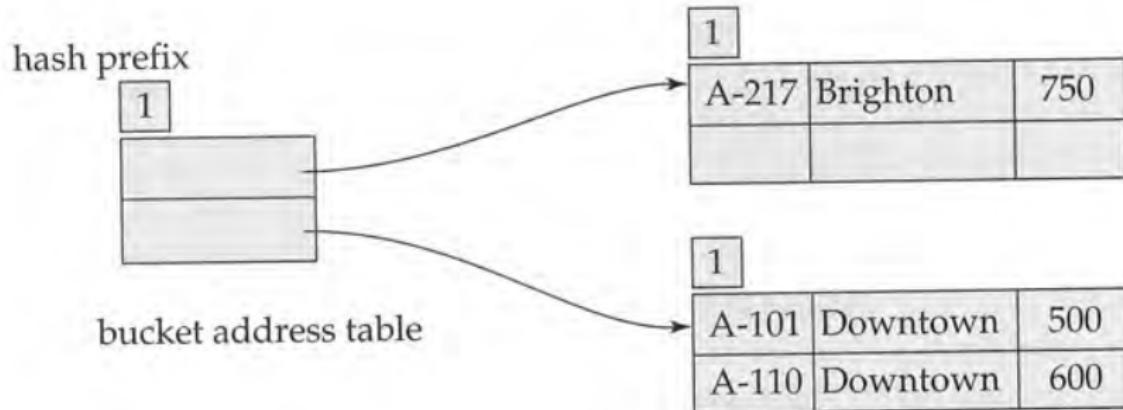
A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

<i>branch_name</i>	$h(branch_name)$
Brighton	0010 1101 1111 1011 0010 1100 0011 0000
Downtown	1010 0011 1010 0000 1100 0110 1001 1111
Mianus	1100 0111 1110 1101 1011 1111 0011 1010
Perryridge	1111 0001 0010 0100 1001 0011 0110 1101
Redwood	0011 0101 1010 0110 1100 1001 1110 1011
Round Hill	1101 1000 0011 1111 1001 1100 0000 0001

Dynamic hashing: extendable hash index

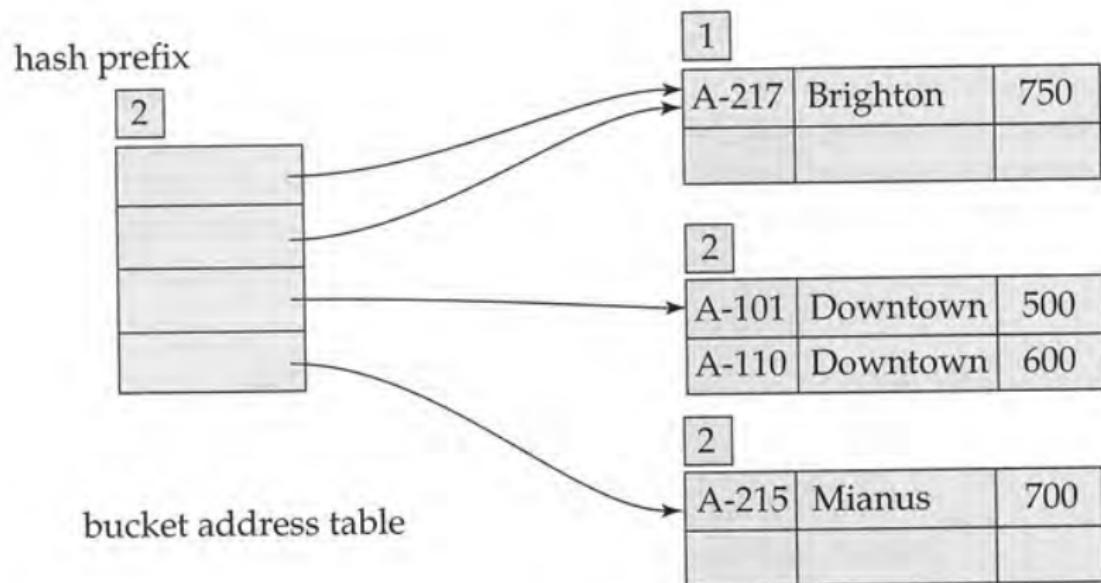


Dynamic hashing: extendable hash index



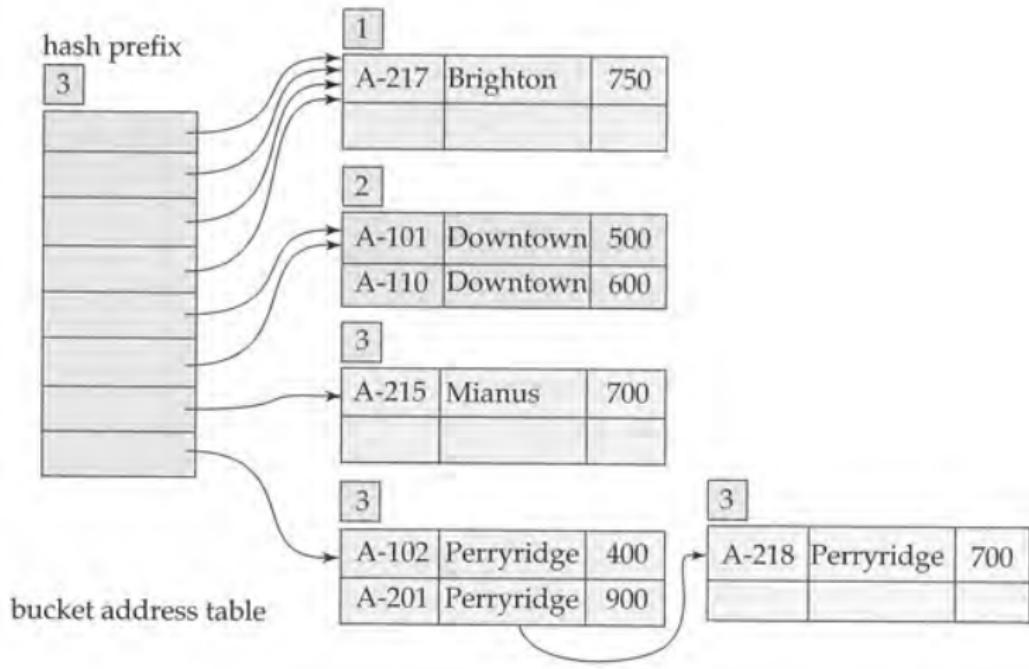
after insertion of:
(A-217, Brighton, 750):0010,
(A-101, Downtown, 500):1010,
(A-110, Downtown, 600):1010

Dynamic hashing: extendable hash index



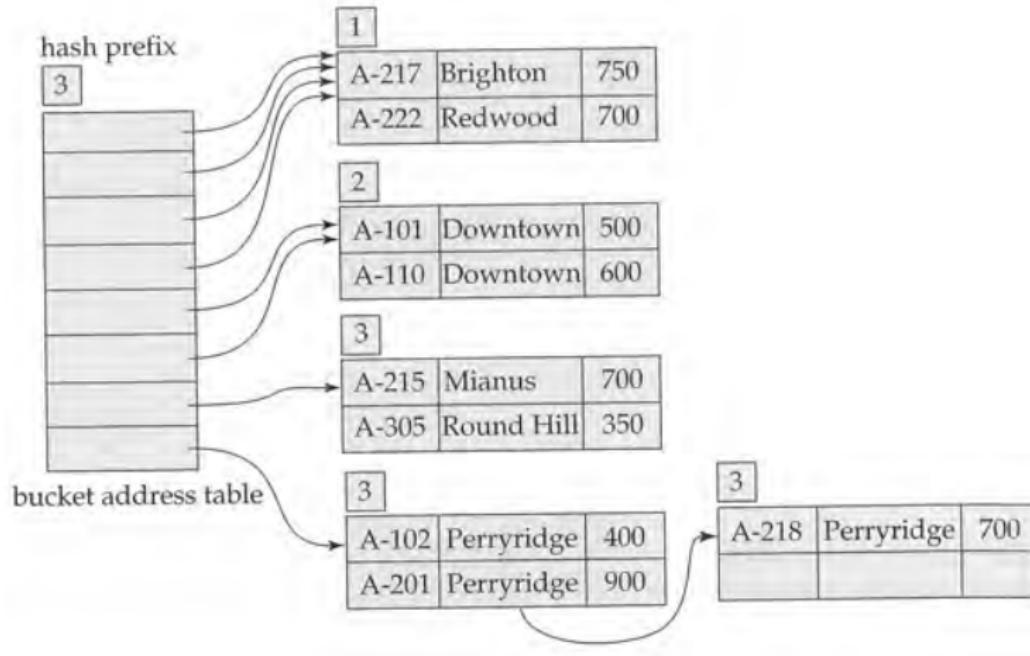
after insertion of:
(A-215, Mianus, 700):1100

Dynamic hashing: extendable hash index



after insertion of: (A-102, Perryridge, 400):1111,
(A-201, Perryridge, 900):1111, (A-218, Perryridge, 700):1111

Dynamic hashing: extendable hash index



after insertion of: (A-222, Redwood, 700):0011,
(A-305, Round Hill, 350):1101

Dynamic hashing: extendable hash index

Deletions

- ▶ essentially the reverse of insertions
- ▶ merge locally when buckets are empty
- ▶ shrink bucket address table when all local hash prefixes fall below the global hash prefix

Dynamic hashing: extendable hash index

- ▶ extendable hashing requires two or three I/Os per search
- ▶ shortcomings of extendable hashing
 - ▶ extra overhead of directory, both for storage and look-up

Dynamic hashing: extendable hash index

- ▶ extendable hashing requires two or three I/Os per search
- ▶ shortcomings of extendable hashing
 - ▶ extra overhead of directory, both for storage and look-up
- ▶ can we eliminate the directory?

Dynamic hashing: extendable hash index

- ▶ extendable hashing requires two or three I/Os per search
- ▶ shortcomings of extendable hashing
 - ▶ extra overhead of directory, both for storage and look-up
- ▶ can we eliminate the directory?
- ▶ yes, use a family of hash functions to simulate the directory
 - ▶ linear hash index

Dynamic hashing: linear hash index

Idea:

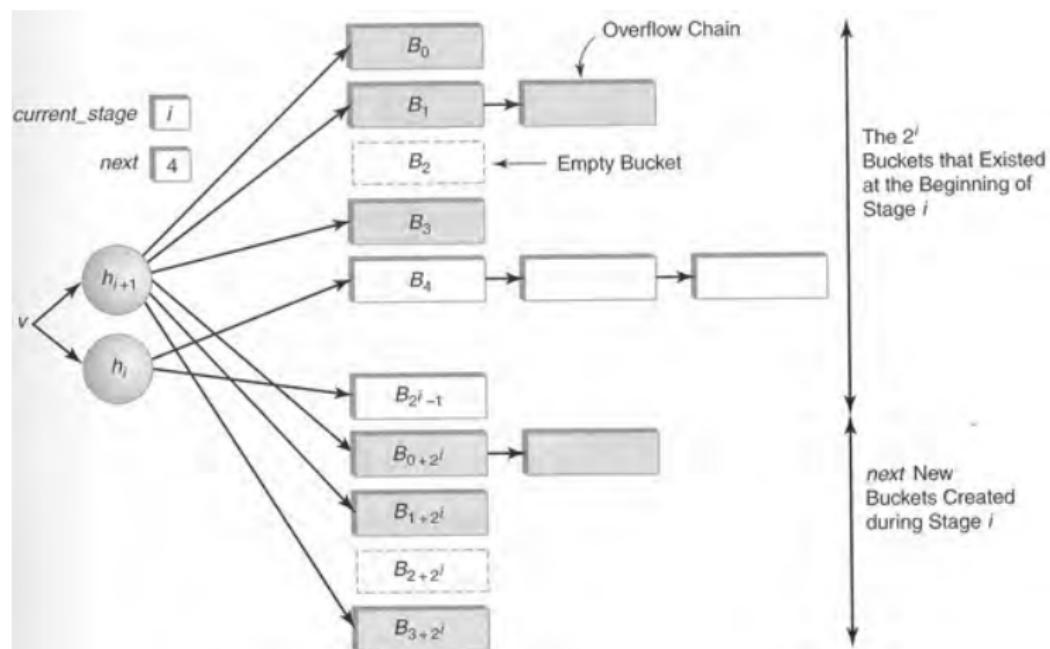
- ▶ use a family of hash functions h_0, \dots, h_n , where h_i uses i (least significant) bits to determine the appropriate bucket
- ▶ at stage i , there are initially 2^i buckets, and we use h_i for search

Dynamic hashing: linear hash index

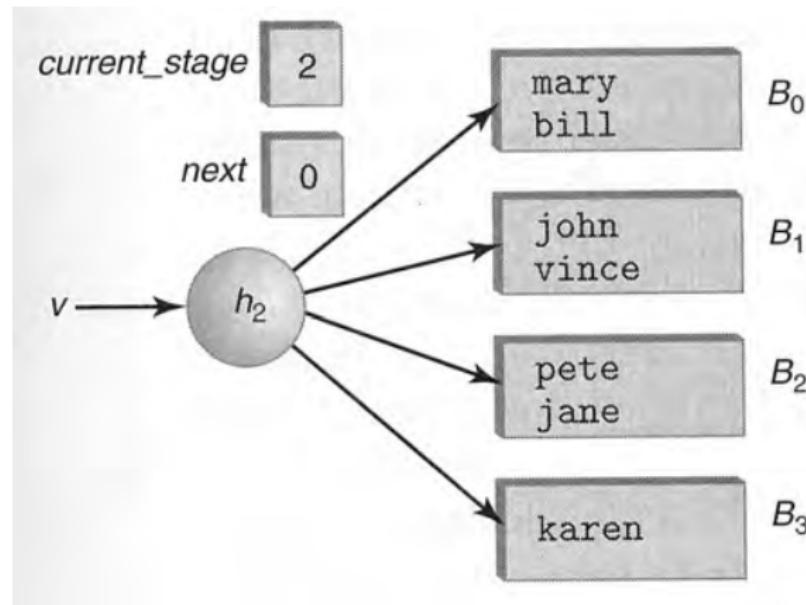
Idea:

- ▶ use a family of hash functions h_0, \dots, h_n , where h_i uses i (least significant) bits to determine the appropriate bucket
- ▶ at stage i , there are initially 2^i buckets, and we use h_i for search
- ▶ to grow, systematically split and rehash buckets from 0 to 2^i , as needed
 - ▶ keep pointer to *next* bucket to split
 - ▶ split upon overflow (of any bucket)
 - ▶ after split, $next = next + 1$
- ▶ for those buckets which have already been split, use h_{i+1} for search

Dynamic hashing: linear hash index

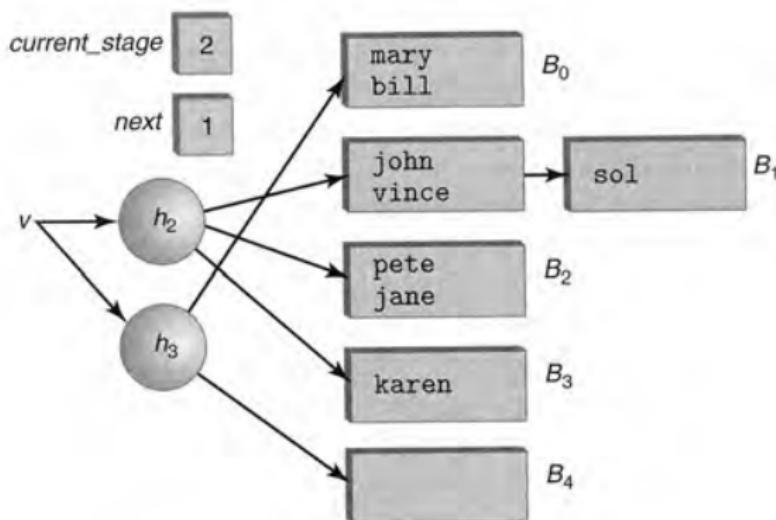


Dynamic hashing: linear hash index



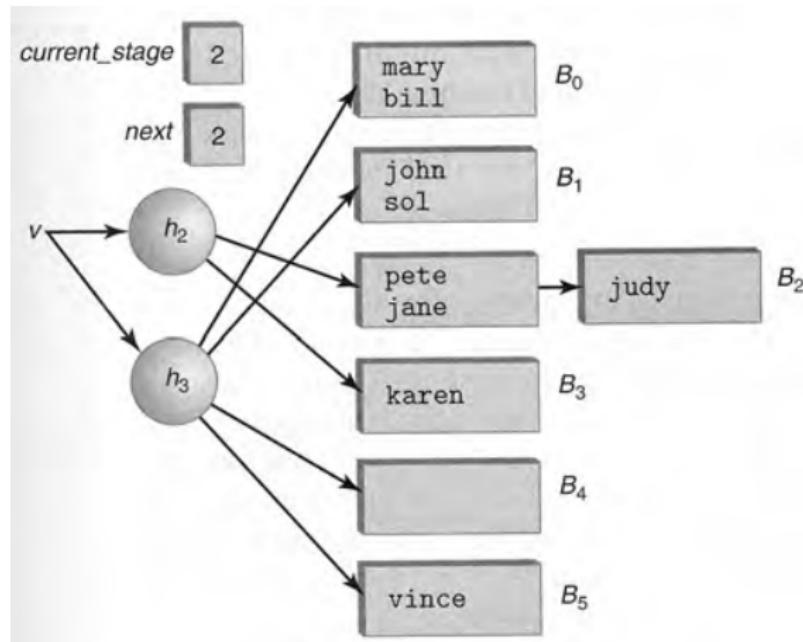
initial table at stage 2

Dynamic hashing: linear hash index



after insertion of “sol”

Dynamic hashing: linear hash index



after insertion of “judy”

Exercise: linear hash index

A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

<i>branch_name</i>	$h(branch_name)$
Brighton	0010 1101 1111 1011 0010 1100 0011 0000
Downtown	1010 0011 1010 0000 1100 0110 1001 1111
Mianus	1100 0111 1110 1101 1011 1111 0011 1010
Perryridge	1111 0001 0010 0100 1001 0011 0110 1101
Redwood	0011 0101 1010 0110 1100 1001 1110 1011
Round Hill	1101 1000 0011 1111 1001 1100 0000 0001

Suppose L is an empty linear hash table, where each page holds two records.

Show L after inserting all account records (hashed on branch name, as in the examples above), in order of their appearance.

Ordered vs. Hash-based Indexing

Ordered vs. hash-based index

- ▶ space overhead
- ▶ supported query types
- ▶ I/O costs
 - ▶ search
 - ▶ maintenance

Bitmap indexes

Bitmap indexes

Bitmap index

- ▶ implemented as one or more bit vectors
- ▶ ideal for selections on attributes of small domain cardinality
 - ▶ job category { entry level, lead, management }
 - ▶ academic degree { BSc, MSc PhD }
 - ▶ sex { female, male }
 - ▶ smoker { yes, no }
- ▶ the i th bit in the “male” vector is 1 if, in the i th row of the Person table, the Sex attribute has value Male.

Bitmap indexes

record number	<i>name</i>	<i>gender</i>	<i>address</i>	<i>income_level</i>
0	John	m	Perryridge	L1
1	Diana	f	Brooklyn	L2
2	Mary	f	Jonestown	L1
3	Peter	m	Brooklyn	L4
4	Kathy	f	Perryridge	L3

Bitmaps for *gender*

m	1 0 0 1 0
f	0 1 1 0 1

Bitmaps for *income_level*

L1	1 0 1 0 0
L2	0 1 0 0 0
L3	0 0 0 0 1
L4	0 0 0 1 0
L5	0 0 0 0 0

Bitmap indexes

Bitmap index

- ▶ only one bit per row, per domain value
- ▶ bitmaps are very small, compared to actual relation size, usually less than 1%
- ▶ also, highly compressible

Bitmap indexes

Bitmap index

- ▶ only one bit per row, per domain value
- ▶ bitmaps are very small, compared to actual relation size, usually less than 1%
- ▶ also, highly compressible
- ▶ complex boolean selection conditions over bitmaps can be performed efficiently using bitwise AND, OR, NOT
- ▶ very successful in read-intensive applications in data mining and data warehousing

Join indexes

Join indexes

CUSTOMER

csur	cname	city	age	job
1	Smith	Boston	21	clerk
2	Collins	Austin	26	secretary
3	Ross	Austin	36	manager
4	Jones	Paris	29	engineer

CP

cpsur	cname	pname	qty	date
2	Smith	jeans	2	052585
3	Smith	shirt	4	052585
1	Ross	jacket	3	072386

JI

csur	cpsur
1	2
1	3
3	1

JI on attribute cname

Join indexes: implementation

CUSTOMER

csur	cname	city	age	job
1	Smith	Boston	21	clerk
2	Collins	Austin	26	secretary
3	Ross	Austin	36	manager
4	Jones	Paris	29	engineer

index
on csur

Π_{csur}

csur	cpsur
1	2
1	3
3	1

index
on csur

CP

cpsur	cname	pname	qty	date
2	Smith	jeans	2	052585
3	Smith	shirt	4	052585
1	Ross	jacket	3	072386

index
on cpsur

Π_{cpsur}

cpsur	csur
1	3
2	1
3	1

index
on cpsur

indexing on surrogates

Join indexes: implementation

can also be implemented as “bitmapped” join index

- ▶ instead of

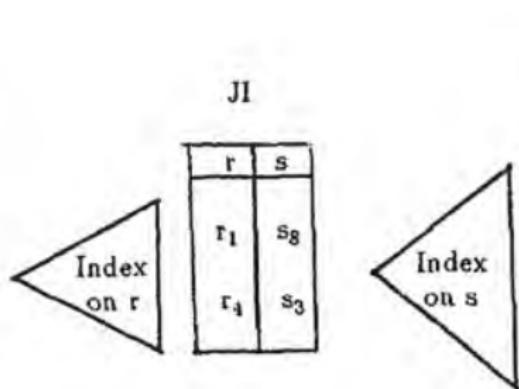
$$(rid, sid)$$

we have

$$(rid, \text{bitmap for matching tuples in } S)$$

Join indexes: use

R	A	B	r
	b		r_1
	b		r_4



S	s	C	D
	s_3		
	s_8		

$$R.B = 'b' \text{ and } R.A = S.D$$

Join indexes: use

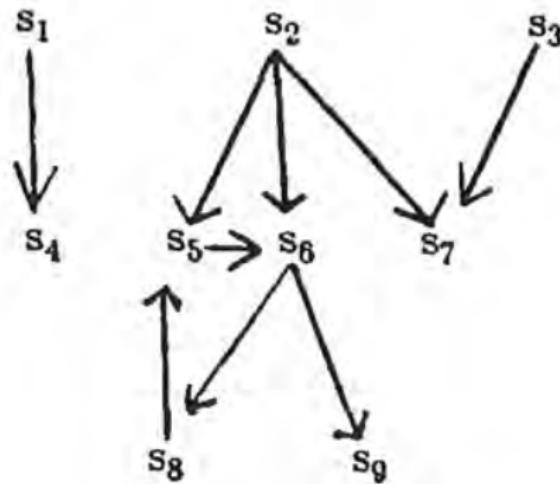
Ph.D					JI	
sur	advisee	advisor	university	year	sur	sur
1	Doe		Harvard	1970	1	2
2	Smith	Doe	Harvard	1976	2	3
3	Ross	Smith	MIT	1980	4	5
4	Hayes		Stanford	1978		
5	James	Hayes	Princeton	1984		

coding a self-join

Join indexes: use

JI

s_1	s_4
s_2	s_5
s_2	s_6
s_2	s_7
s_3	s_7
s_5	s_6
s_6	s_8
s_8	s_9
s_8	s_5



coding a directed graph

Join indexes: use

Many variations in practice, e.g., extensions for OO hierarchies

Models of indexing

Indexing

- ▶ **key**: a collection of attributes (of a relation)

Indexing

- ▶ **key**: a collection of attributes (of a relation)
- ▶ **index**: data structure for external searching
 - ▶ i.e., mapping key values to data
 - ▶ i.e., locating a collection of data entries k^* for search key k
 - ▶ i.e.,
 - value \mapsto index
 - \mapsto blocks holding records
 - \mapsto matching records

Indexing

- ▶ **key**: a collection of attributes (of a relation)
- ▶ **index**: data structure for external searching
 - ▶ i.e., mapping key values to data
 - ▶ i.e., locating a collection of data entries k^* for search key k
 - ▶ i.e.,
 - value \mapsto index
 - \mapsto blocks holding records
 - \mapsto matching records
- ▶ **basic operations**: $\text{search}(k)$, $\text{insert}(k)$, $\text{delete}(k)$

Models of Indexing

1. workload model (Hellerstein et al)
2. GMAP model (Tsatalos et al)
3. query language model (Fletcher et al)

Models of Indexing: workload model

(1) Workload model

- ▶ proposed and studied by Hellerstein et al,
JACM 2002
- ▶ studies indexing schemes with respect to
“workloads” of queries

Models of Indexing: workload model

A **workload** is modeled as a finite subset of some domain, together with a set of queries

$$W = (D, I, \mathcal{Q})$$

where

- ▶ D is a **domain** (e.g., \mathbb{R} with \leq)

Models of Indexing: workload model

A **workload** is modeled as a finite subset of some domain, together with a set of queries

$$W = (D, I, \mathcal{Q})$$

where

- ▶ D is a **domain** (e.g., \mathbb{R} with \leq)
- ▶ I is a finite subset of D , called the **instance**

Models of Indexing: workload model

A **workload** is modeled as a finite subset of some domain, together with a set of queries

$$W = (D, I, \mathcal{Q})$$

where

- ▶ D is a **domain** (e.g., \mathbb{R} with \leq)
- ▶ I is a finite subset of D , called the **instance**
- ▶ $\mathcal{Q} = \{Q_1, \dots, Q_q\}$ is a finite set of **queries**, which are

$$Q_i \subseteq I$$

for each $1 \leq i \leq q$

Models of Indexing: workload model

A **workload** is modeled as a finite subset of some domain, together with a set of queries

$$W = (D, I, \mathcal{Q})$$

For example, the one-dimensional range queries

- ▶ $D = \mathbb{N}$, with natural ordering \leq
- ▶ $I = \{i \mid 1 \leq i \leq n\}$, for some fixed $n \in \mathbb{N}$
- ▶ $\mathcal{Q} = \{Q[a, b] \mid 1 \leq a \leq b \leq n\}$, where
 $Q[a, b] = \{i \mid a \leq i \leq b\}$

Models of Indexing: workload model

An indexing scheme \mathcal{S} for workload $W = (D, I, \mathcal{Q})$ is a collection of blocks b_1, \dots, b_k where each block is a B sized subset of I , such that $\bigcup b_i = I$, for some fixed positive integer B .

- ▶ in practice, B is on the order of 100 to 1,000

Models of Indexing: workload model

Two performance measures: storage and access cost

- ▶ Storage redundancy of \mathcal{S} is the maximum number of blocks that contain an element of I .

Models of Indexing: workload model

Two performance measures: storage and access cost

- ▶ Storage redundancy of \mathcal{S} is the maximum number of blocks that contain an element of I .
- ▶ The average redundancy is then the average number of blocks that contain an element of I , i.e., $\frac{kB}{|I|}$
 - ▶ in the range of 1 (best) to k (worst)
 - ▶ best when $|I| = kB$
 - ▶ worst when $|I| = B$

Models of Indexing: workload model

Two performance measures: storage and access cost

- Given query $Q \in \mathcal{Q}$, the ideal access cost would be

$$\left\lceil \frac{|Q|}{B} \right\rceil$$

blocks. If we let C_Q be a minimum-sized set of blocks which actually covers Q , then the **access overhead** of Q in \mathcal{S} is

$$A(Q) = \frac{|C_Q|}{\left\lceil \frac{|Q|}{B} \right\rceil}$$

Models of Indexing: workload model

Then, the access overhead of \mathcal{S} is

$$A(\mathcal{S}) = \max_{Q \in \mathcal{Q}} A(Q)$$

Models of Indexing: workload model

Then, the access overhead of \mathcal{S} is

$$A(\mathcal{S}) = \max_{Q \in \mathcal{Q}} A(Q)$$

Note that $A(\mathcal{S})$ is in the range of 1 (best) to B (worst)

- ▶ best when $|C_Q| = \left\lceil \frac{|Q|}{B} \right\rceil$, for all $Q \in \mathcal{Q}$
- ▶ worst when $|C_Q| = |Q|$, for some $Q \in \mathcal{Q}$

Models of Indexing: workload model

Example. In one-dimensional range query workloads, any access method that partitions data items along the associated linear order \leq achieves optimality, both in access overhead and average redundancy.

- ▶ B+trees, with an additive constant of two or three in access overhead

Models of Indexing: workload model

The model suppresses

- ▶ how to map I into S

Models of Indexing: workload model

The model suppresses

- ▶ how to map I into \mathcal{S}
- ▶ how to map $Q \in \mathcal{Q}$ into \mathcal{S}

Models of Indexing: workload model

The model suppresses

- ▶ how to map I into \mathcal{S}
- ▶ how to map $Q \in \mathcal{Q}$ into \mathcal{S}
- ▶ storage and access costs associated with these two (e.g., auxiliary information such as directories and internal nodes)

Models of Indexing: workload model

The model suppresses

- ▶ how to map I into \mathcal{S}
- ▶ how to map $Q \in \mathcal{Q}$ into \mathcal{S}
- ▶ storage and access costs associated with these two (e.g., auxiliary information such as directories and internal nodes)

rationale

- ▶ the model is only interested in lower bounds
- ▶ suppressed costs don't seem to be the primary source of difficulty in practice
- ▶ secondary storage techniques, such as buffer management, absorb many of these auxiliary costs

Models of Indexing: workload model

Benefits include:

- ▶ focus on main balance in indexing: the tradeoff between redundancy and access costs

Models of Indexing: workload model

Benefits include:

- ▶ focus on main balance in indexing: the tradeoff between redundancy and access costs
- ▶ can analyze new workloads by showing them to be isomorphic to well-understood workloads

Models of Indexing: GMAP model

(2) GMAP model

- ▶ introduced by Tsatalos et al, DEXA 1994,
VLDBJ 1996
- ▶ focuses on decoupling logical schema from physical schema
 - ▶ how to map \mathcal{I} into \mathcal{S}

Models of Indexing: GMAP model

- ▶ logical schema is modeled as entities, their attributes, and relationships between entities
 - ▶ as a logical collection of binary relations

Models of Indexing: GMAP model

- ▶ logical schema is modeled as entities, their attributes, and relationships between entities
 - ▶ as a logical collection of binary relations
- ▶ presents an extended conjunctive query language to define physical structures on the logical schema

Models of Indexing: GMAP model

Example. Suppose we have Faculty, Students, Departments, and Courses, and we'd like to build a b+-tree on CS faculty, by their research area

Models of Indexing: GMAP model

Example. Suppose we have Faculty, Students, Departments, and Courses, and we'd like to build a b+tree on CS faculty, by their research area

```
def_gmap cs_faculty_by_area as btree by
given Faculty.area
select Faculty
where Faculty worksIn Dept
      and Dept = csOID
```

Models of Indexing: GMAP model

Example. Suppose we have Faculty, Students, Departments, and Courses, and we'd like to build a b+tree on CS faculty, by their research area

$$\pi_{F, F.area}(\sigma_{D=csOID}(F.area \bowtie worksIn))$$

Models of Indexing: GMAP model

Then, given query/update Q on the logical schema, translate Q into query/update over physical structure

Models of Indexing: GMAP model

Then, given query/update Q on the logical schema, translate Q into query/update over physical structure

Examples

- ▶ relations (primary heap)
- ▶ secondary indexes (B+trees)
- ▶ join indexes
- ▶ access support relations
- ▶ ...

Models of Indexing: query language model

(3) Query language model

- ▶ presented in Fletcher et al, *Inf Sys* 2009
- ▶ focuses on finding ideal object partitioning for a given query language
 - ▶ how to map I into \mathcal{S}
 - ▶ how to map $Q \in \mathcal{Q}$ into \mathcal{S}

Models of Indexing: query language model

- ▶ a query language \mathcal{L} is typically an infinite object
- ▶ for an instance I , $\mathcal{L}(I)$ is also typically infinite

Models of Indexing: query language model

- ▶ a query language \mathcal{L} is typically an infinite object
- ▶ for an instance I , $\mathcal{L}(I)$ is also typically infinite
- ▶ however, the partition of I with respect to distinguishability of objects by queries in \mathcal{L} , denoted I/\mathcal{L} , is finite if I is finite
- ▶ so, if we can efficiently build I/\mathcal{L} , we have an ideal basis for indexing I with respect to \mathcal{L}

Models of Indexing: query language model

Examples

- ▶ keyword search and inverted indexes
 - ▶ language equivalence is captured by keyword equivalence

Models of Indexing: query language model

Examples

- ▶ keyword search and inverted indexes
 - ▶ language equivalence is captured by keyword equivalence
- ▶ range queries and B+trees
 - ▶ language equivalence is captured by value equivalence

Models of Indexing: query language model

Examples

- ▶ keyword search and inverted indexes
 - ▶ language equivalence is captured by keyword equivalence
- ▶ range queries and B+trees
 - ▶ language equivalence is captured by value equivalence
- ▶ XML path queries and “structural” indexes
 - ▶ language equivalence is captured by bisimulation equivalence

Models of Indexing: query language model

Examples

- ▶ keyword search and inverted indexes
 - ▶ language equivalence is captured by keyword equivalence
- ▶ range queries and B+trees
 - ▶ language equivalence is captured by value equivalence
- ▶ XML path queries and “structural” indexes
 - ▶ language equivalence is captured by bisimulation equivalence

All of these “structural” characterizations of language equivalence are efficiently computable

Models of Indexing: query language model

Questions

How to go from \mathcal{L} to I/\mathcal{L} to $\mathcal{L}(I)$?

- ▶ That is, how to map a query $q \in \mathcal{L}$ to $q(I)$, via I/\mathcal{L} ?
 - ▶ rewrite q on reduced space I/\mathcal{L}

Models of Indexing: query language model

Questions

How to go from \mathcal{L} to I/\mathcal{L} to $\mathcal{L}(I)$?

- ▶ That is, how to map a query $q \in \mathcal{L}$ to $q(I)$, via I/\mathcal{L} ?
 - ▶ rewrite q on reduced space I/\mathcal{L}

How can we use I/\mathcal{L} to query in a superset of \mathcal{L} ?

Models of Indexing: query language model

Questions

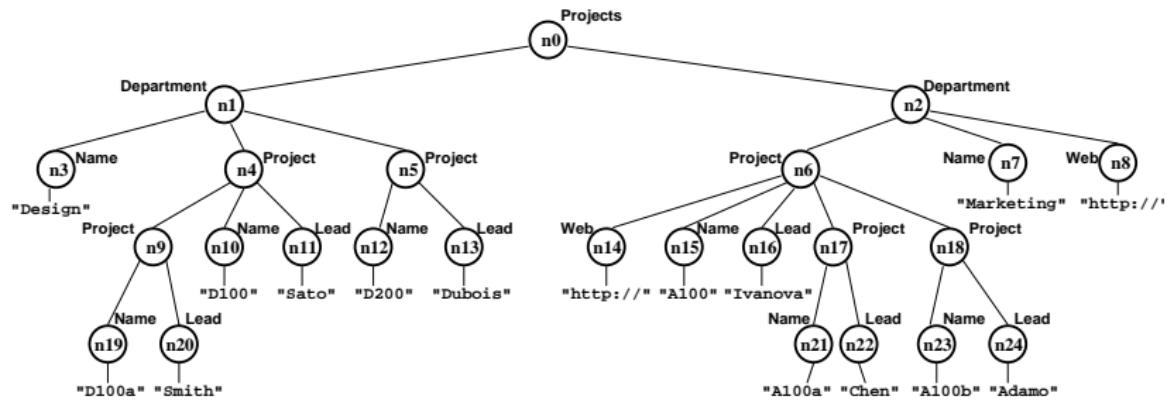
How to go from \mathcal{L} to I/\mathcal{L} to $\mathcal{L}(I)$?

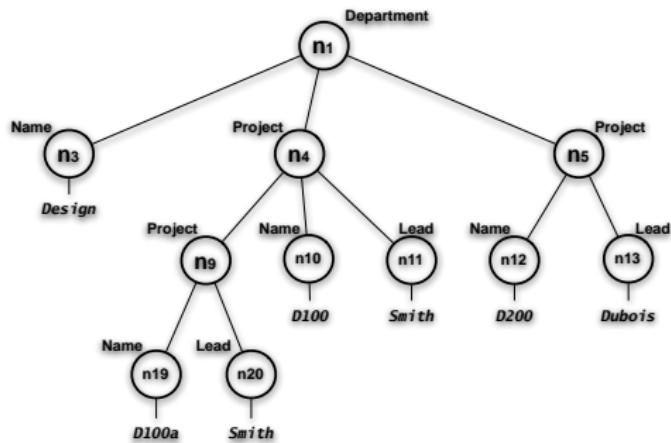
- ▶ That is, how to map a query $q \in \mathcal{L}$ to $q(I)$, via I/\mathcal{L} ?
 - ▶ rewrite q on reduced space I/\mathcal{L}

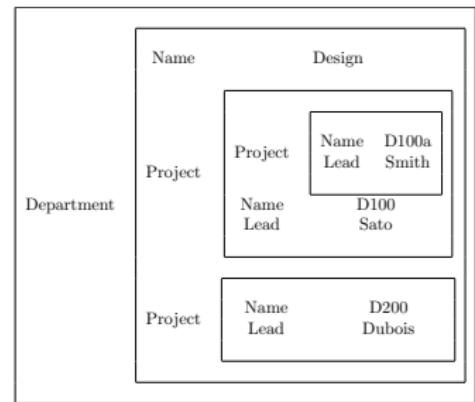
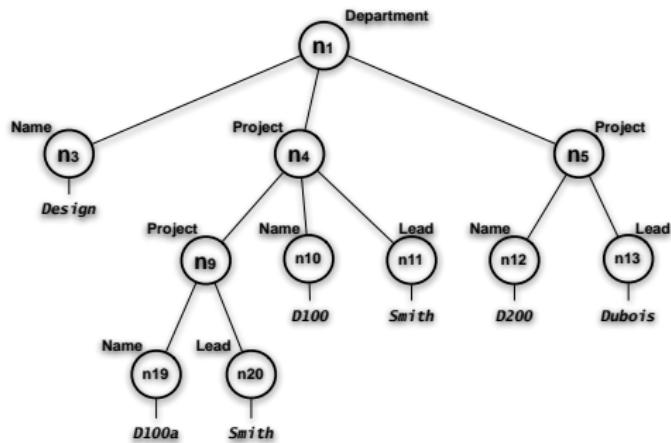
How can we use I/\mathcal{L} to query in a superset of \mathcal{L} ?

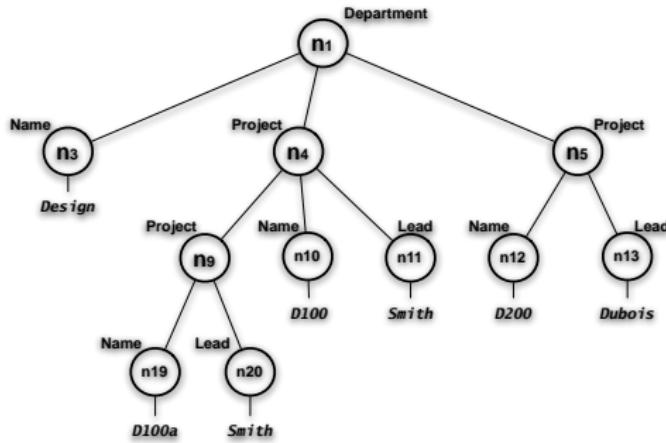
- ▶ query decomposition, mapping subexpressions to blocks, and then result reconstruction
- ▶ example: branching queries and $P(k)$ indexes for XML

Case study: XPath and XML









```

<Department>
  <Name>Design</Name>
  <Project>
    <Project>
      <Name>D100a</Name>
      <Lead>Smith</Name>
    </Project>
    <Name>D100</Name>
    <Lead>Sato</Lead>
  </Project>
  <Project>
    <Name>D200</Name>
    <Lead>Dubois</Lead>
  </Project>
</Department>

```

XML Data Model

Documents $D = (V, Ed, r, \lambda)$ are finite unordered node-labeled trees:

- ▶ nodes V
- ▶ edges $Ed \subseteq V \times V$
- ▶ root r
- ▶ labels $\lambda : V \rightarrow L$

XPath

Expressions specify tree patterns

XPath

Expressions specify tree patterns

example: “Retrieve leaders of subprojects.”

XPath

Expressions specify tree patterns

example: “Retrieve leaders of subprojects.”

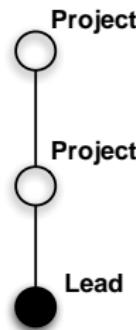
```
//Project/Project/Lead
```

XPath

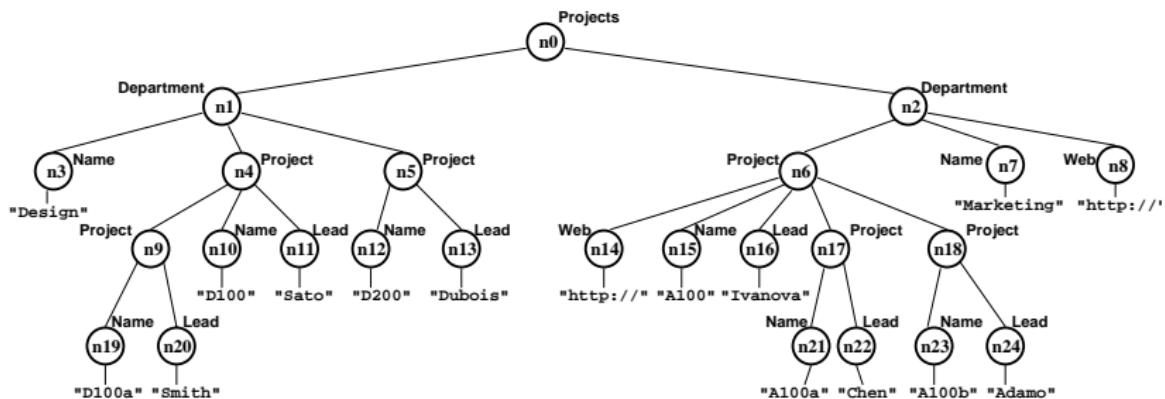
Expressions specify tree patterns

example: “Retrieve leaders of subprojects.”

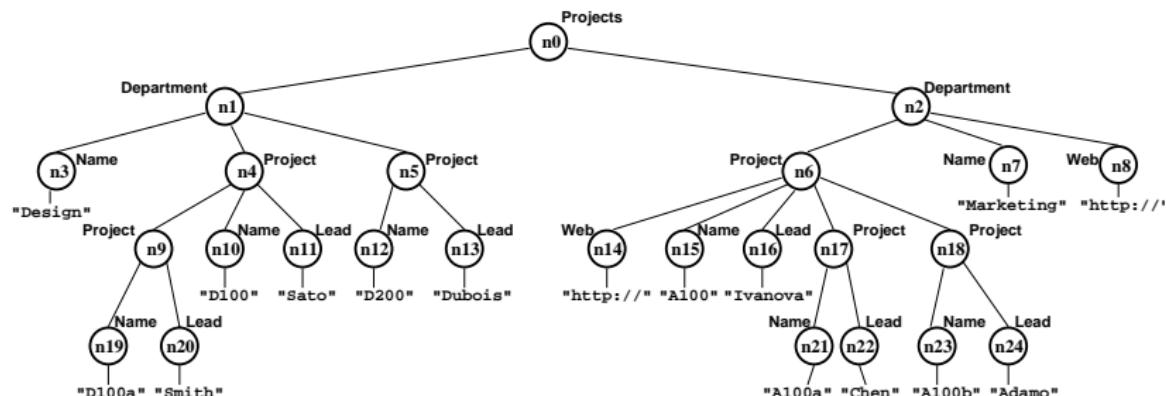
`//Project/Project/Lead`



XPath



XPath



→

$$\{n_{20}, n_{22}, n_{24}\}$$

XPath

Expressions specify tree patterns

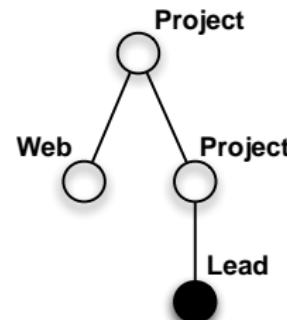
example: “Retrieve leaders of a subproject of a project having a website.”

XPath

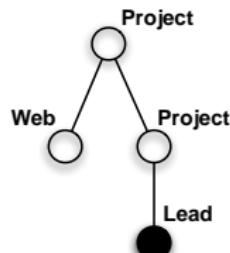
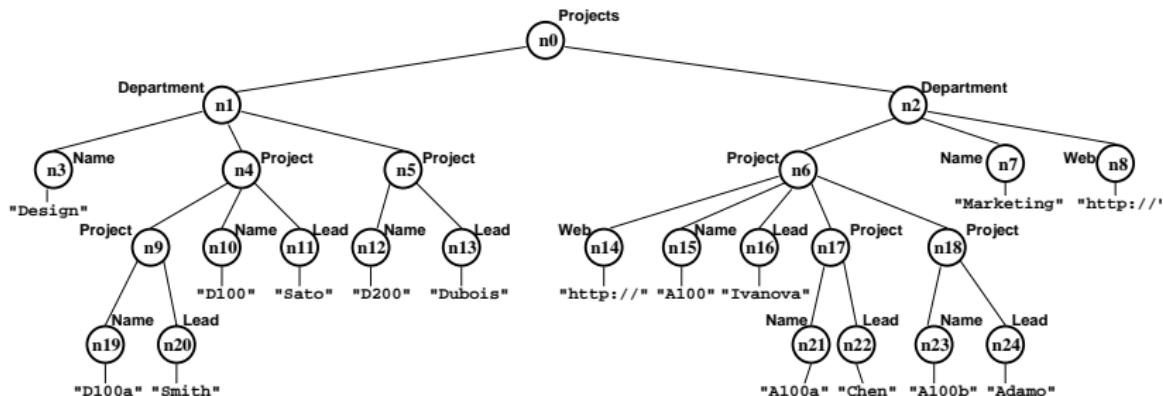
Expressions specify tree patterns

example: “Retrieve leaders of a subproject of a project having a website.”

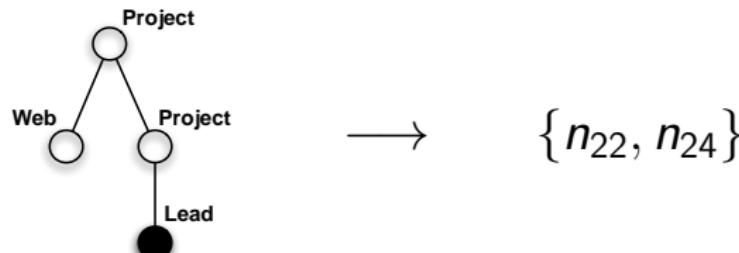
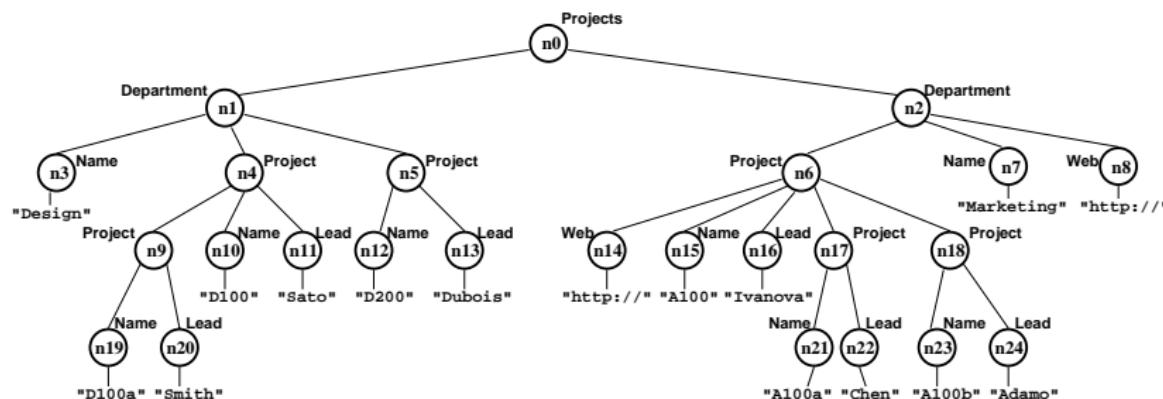
```
//Project[/Web]/Project/Lead
```



XPath



XPath



The XPath algebra

Given document

$$D = (V, Ed, r, \lambda)$$

The XPath algebra

Given document

$$D = (V, Ed, r, \lambda)$$

$$\varepsilon(D) = \{(n, n) \mid n \in V\}$$

$$\emptyset(D) = \emptyset$$

$$\ell(D) = \{(n, n) \mid m \in V \text{ and } \lambda(n) = \ell\}$$

$$\downarrow(D) = Ed$$

$$\uparrow(D) = Ed^{-1}$$

The XPath algebra

Given document

$$D = (V, Ed, r, \lambda)$$

$$\varepsilon(D) = \{(n, n) \mid n \in V\}$$

$$\emptyset(D) = \emptyset$$

$$\ell(D) = \{(n, n) \mid m \in V \text{ and } \lambda(n) = \ell\}$$

$$\downarrow(D) = Ed$$

$$\uparrow(D) = Ed^{-1}$$

$$E \cup F(D) = E(D) \cup F(D)$$

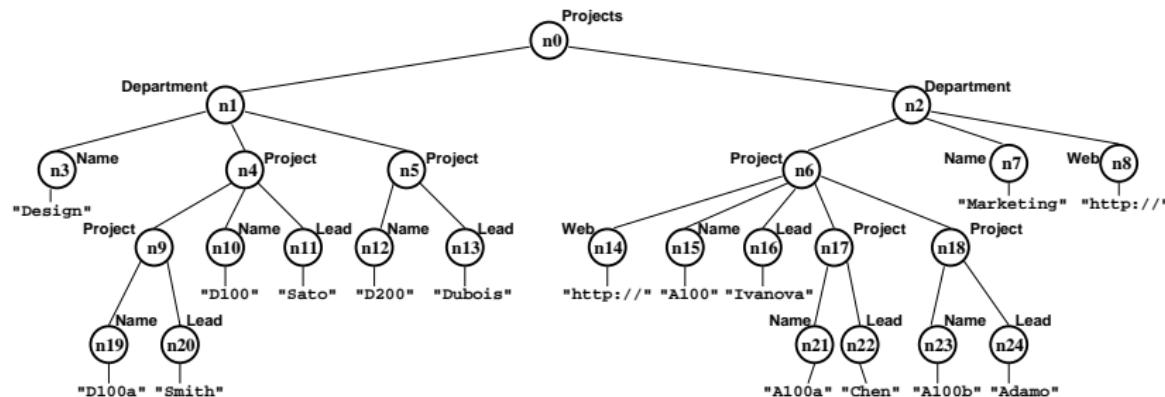
$$E \cap F(D) = E(D) \cap F(D)$$

$$E - F(D) = E(D) - F(D)$$

$$E \circ F(D) = \{(n, m) \mid \exists w: (n, w) \in E(D) \& (w, m) \in F(D)\}$$

$$E[F](D) = \{(n, m) \in E(D) \mid \exists w: (m, w) \in F(D)\}$$

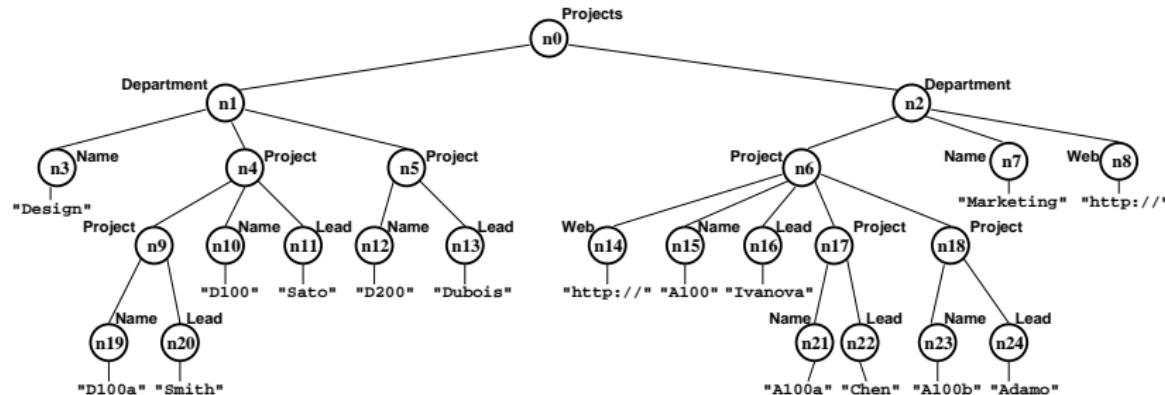
The XPath algebra



“Retrieve department names.”

$$E = \text{Projects} \circ \downarrow \circ \text{Department} \circ \downarrow \circ \text{Name}$$

The XPath algebra



“Retrieve department names.”

$$E = \text{Projects} \circ \downarrow \circ \text{Department} \circ \downarrow \circ \text{Name}$$
$$E(D) = \{(n_0, n_3), (n_0, n_7)\}$$

The $A(k)$ Partition of a Document

Towards structural indexing for efficient XPath evaluation ...

For nodes n and m , we have that they are $A(k)$ -equivalent (denoted $n \equiv_{A(k)} m$) if

- ▶ they have the same label, and
- ▶ for $k > 0$, if one has a parent, so does the other and, furthermore, their parents are $A(k - 1)$ -equivalent

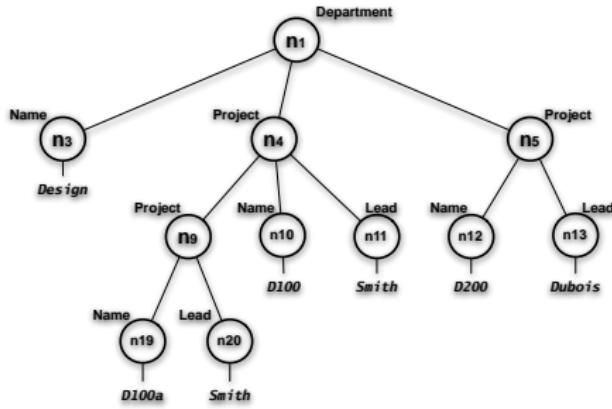
The $A(k)$ Partition of a Document

Towards structural indexing for efficient XPath evaluation ...

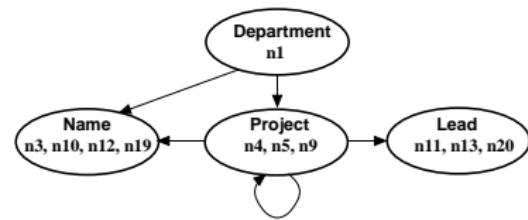
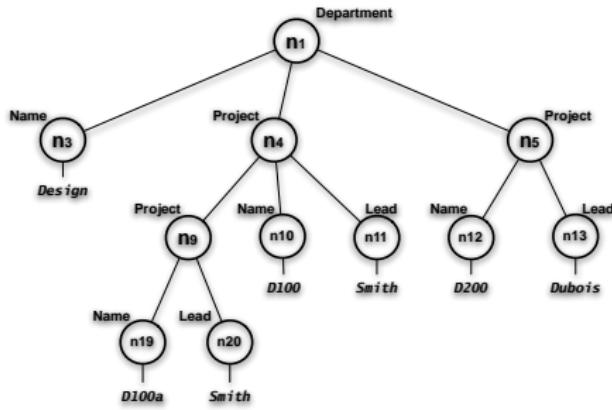
For nodes n and m , we have that they are $A(k)$ -equivalent (denoted $n \equiv_{A(k)} m$) if

- ▶ they have the same label, and
- ▶ for $k > 0$, if one has a parent, so does the other and, furthermore, their parents are $A(k - 1)$ -equivalent

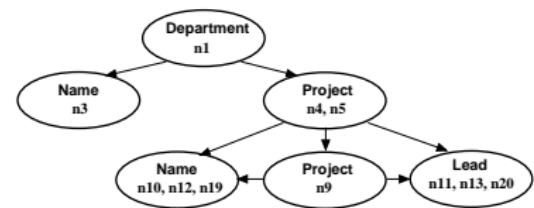
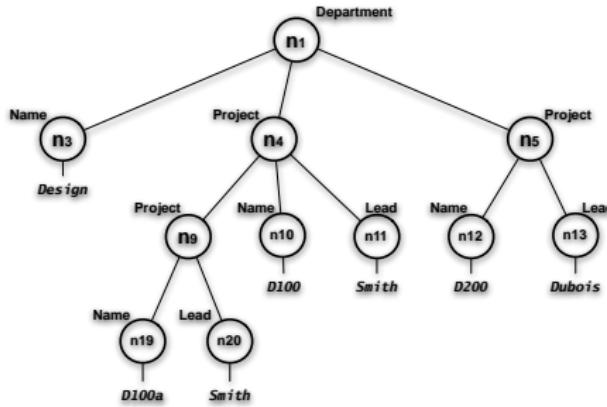
The partition of V induced by this (equivalence) relation on nodes is called the $A(k)$ partition of the document



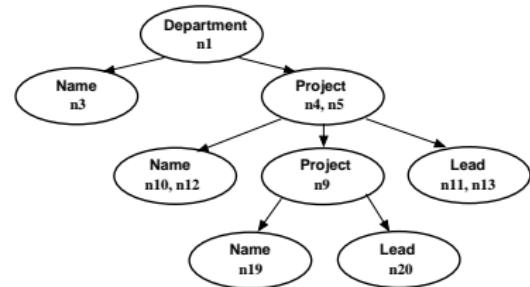
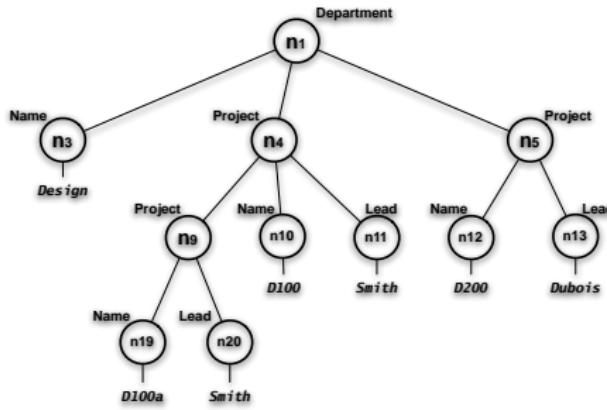
Consider $A(k)$ indexes
on the “Design”
department subtree



A(0) index



A(1) index



A(2) index

The $P(k)$ Partition of a Document

For nodes $n_1, n_2, m_1, m_2 \in V$, we have that (n_1, m_1) and (n_2, m_2) are $P(k)$ -equivalent (denoted $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$) if

- ▶ (n_1, m_1) and (n_2, m_2) are in $UpPaths(D, k)$
- ▶ the distance from n_1 to m_2 in the document is the same as that from n_2 to m_2 , and
- ▶ $n_1 \equiv_{A(k)} n_2$

The $P(k)$ Partition of a Document

For nodes $n_1, n_2, m_1, m_2 \in V$, we have that (n_1, m_1) and (n_2, m_2) are $P(k)$ -equivalent (denoted $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$) if

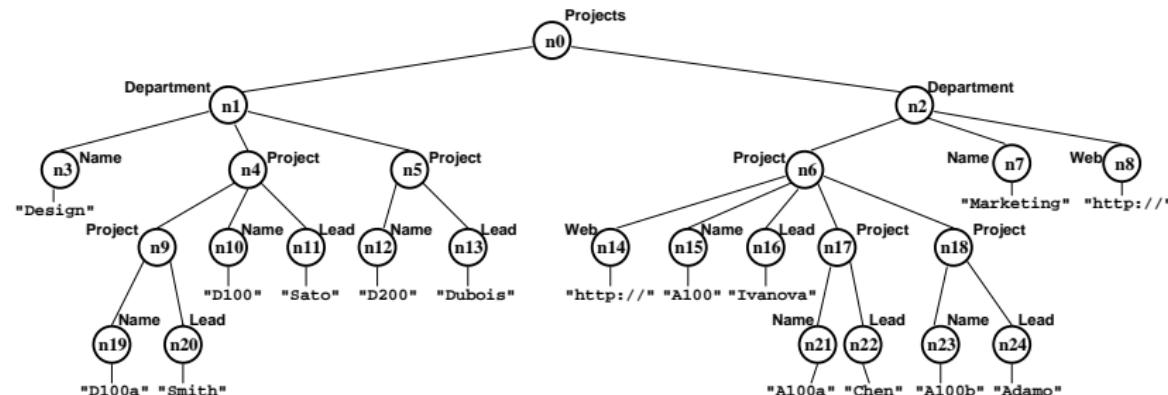
- ▶ (n_1, m_1) and (n_2, m_2) are in $UpPaths(D, k)$
- ▶ the distance from n_1 to m_2 in the document is the same as that from n_2 to m_2 , and
- ▶ $n_1 \equiv_{A(k)} n_2$

The partition induced by this (equivalence) relation on node pairs in $UpPaths(D, k)$ is called the $P(k)$ partition of the document

Upward- k Algebras

For $k \geq 0$, $U(k)$ is the fragment of the XPath-Algebra with expressions that do not use the \downarrow primitive and have at most k uses of the \uparrow primitive in a “path”

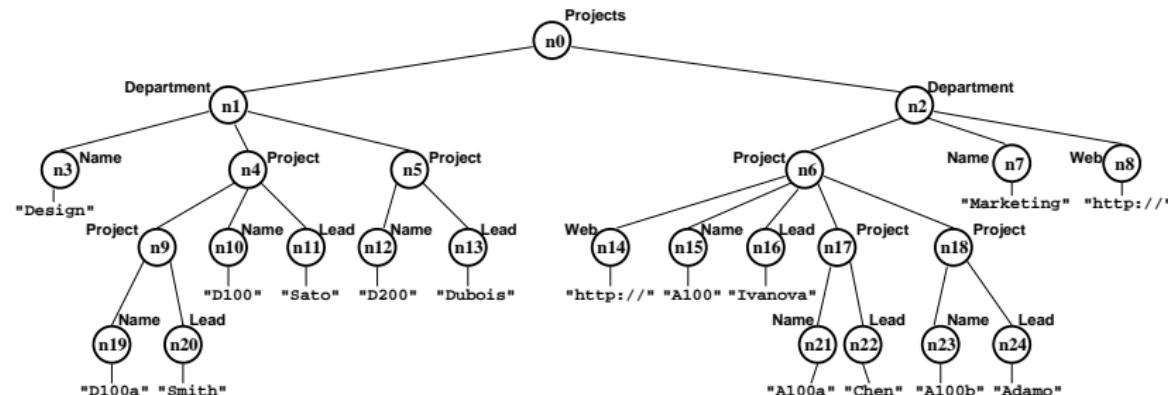
Upward- k Algebras



“Retrieve sub-project leaders.”

$$E = \text{Lead}[\uparrow \circ \text{Project} \circ \uparrow \circ \text{Project}]$$

Upward- k Algebras

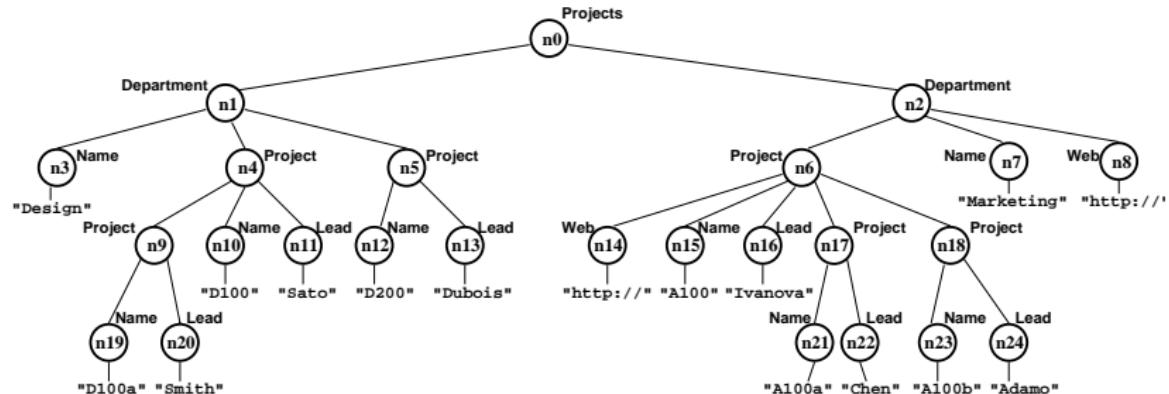


“Retrieve sub-project leaders.”

$$E = \text{Lead}[\uparrow \circ \text{Project} \circ \uparrow \circ \text{Project}]$$

In $U(2)$ but not $U(1)$

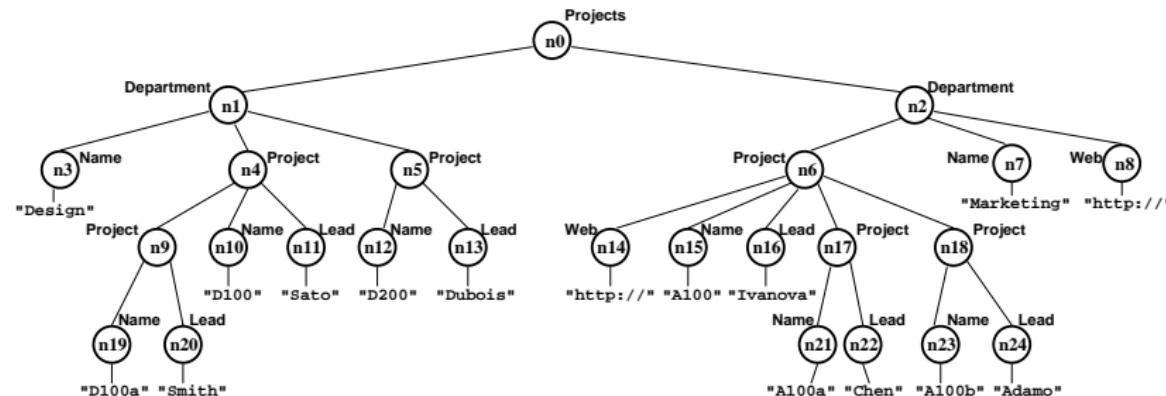
Upward- k Algebras



“Retrieve all projects which are sub-projects of projects with a website.”

$$E = \text{Project}[\uparrow \circ \text{Project} \circ \downarrow \circ \text{Web}]$$

Upward- k Algebras



“Retrieve all projects which are sub-projects of projects with a website.”

$$E = \text{Project}[\uparrow \circ \text{Project} \circ \downarrow \circ \text{Web}]$$

Not in $U(k)$, for any k

Language Indistinguishability

For fragment \mathcal{L} of the XPath algebra, we say node pairs (n_1, m_1) and (n_2, m_2) are **indistinguishable by \mathcal{L}** if for any expression in $e \in \mathcal{L}$, it is the case that $(n_1, m_1) \in e(D)$ if and only if $(n_2, m_2) \in e(D)$.

Language Indistinguishability

For fragment \mathcal{L} of the XPath algebra, we say node pairs (n_1, m_1) and (n_2, m_2) are **indistinguishable by \mathcal{L}** if for any expression in $e \in \mathcal{L}$, it is the case that $(n_1, m_1) \in e(D)$ if and only if $(n_2, m_2) \in e(D)$.

The partition induced by $\equiv_{\mathcal{L}}$ on $Paths(D)$ is called the **\mathcal{L} -partition of D** .

Coupling $P(k)$ and $U(k)$

Theorem (Coupling)

Let D be a document and $k \in \mathbb{N}$. The $P(k)$ -partition of D and the $U(k)$ -partition of D are the same.

Coupling $P(k)$ and $U(k)$

Theorem (Coupling)

Let D be a document and $k \in \mathbb{N}$. The $P(k)$ -partition of D and the $U(k)$ -partition of D are the same.

Theorem (Block-Union)

Let D be a document, $k \in \mathbb{N}$, and $e \in U(k)$. Then there exists a class \mathfrak{B}_e of blocks of the $P(k)$ -partition of D such that $e(D) = \bigcup_{B \in \mathfrak{B}_e} B$.

Coupling $P(k)$ and $U(k)$

The Coupling Theorem provides a precise linguistic characterization of the $P(k)$ partition (alternatively, a precise structural characterization of $U(k)$).

Coupling $P(k)$ and $U(k)$

The Coupling Theorem provides a precise linguistic characterization of the $P(k)$ partition (alternatively, a precise structural characterization of $U(k)$).

The Block-Union Theorem provides us with the evaluation strategy for expressions in $U(k)$. In particular, expression evaluation in $U(k)$ amounts to unions of direct index lookups.

Expression evaluation in Upward Algebra

For expressions in $U(l)$, for $l > k$, then evaluation is via

- ▶ decomposition into $U(k)$ sub-expressions
- ▶ and then joining intermediate results.

Expression evaluation in XPath Algebra

In richer fragments of the algebra, we take a two step rewriting process:

1. predicate elimination
2. invert remaining “downward” subexpressions
into $U(k)$ expressions

Expression evaluation in XPath Algebra

In richer fragments of the algebra, we take a two step rewriting process:

1. predicate elimination
 2. invert remaining “downward” subexpressions into $U(k)$ expressions
- ... and then proceed as before with $U(k)$ evaluation, to make optimal use of available indices.

Expression evaluation in XPath Algebra

Example. Suppose we have document D , the $P(2)$ partition of D , and the query $\downarrow [\downarrow]$.

Expression evaluation in XPath Algebra

Example. Suppose we have document D , the $P(2)$ partition of D , and the query $\downarrow [\downarrow]$.

Then ...

$$\downarrow [\downarrow](D)$$

Expression evaluation in XPath Algebra

Example. Suppose we have document D , the $P(2)$ partition of D , and the query $\downarrow [\downarrow]$.

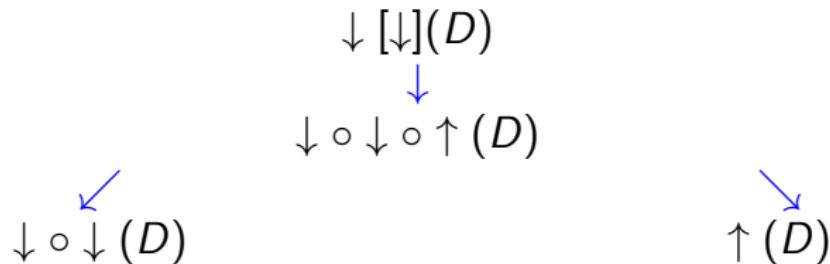
Then ...

$$\begin{array}{c} \downarrow [\downarrow](D) \\ \downarrow \\ \downarrow \circ \downarrow \circ \uparrow (D) \end{array}$$

Expression evaluation in XPath Algebra

Example. Suppose we have document D , the $P(2)$ partition of D , and the query $\downarrow [\downarrow]$.

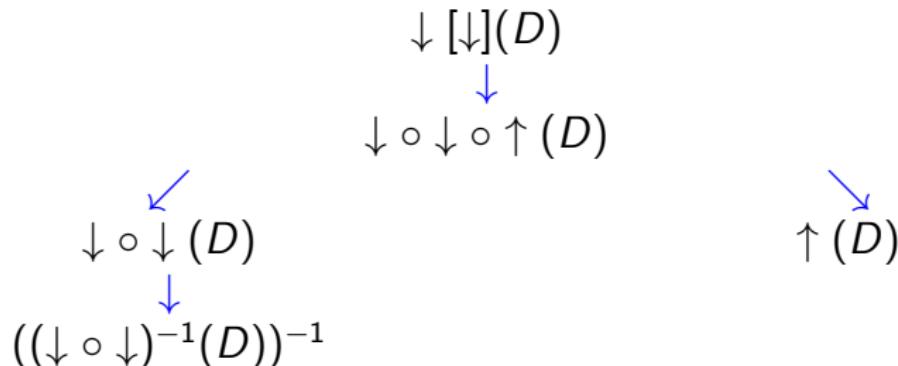
Then ...



Expression evaluation in XPath Algebra

Example. Suppose we have document D , the $P(2)$ partition of D , and the query $\downarrow [\downarrow]$.

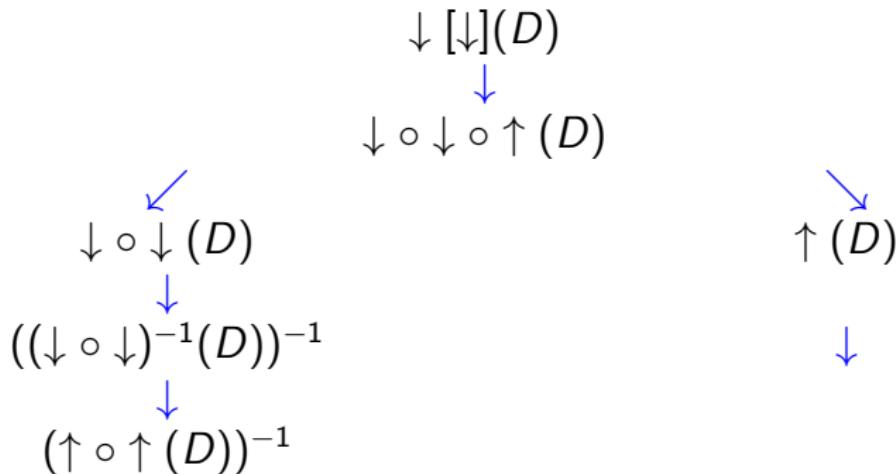
Then ...



Expression evaluation in XPath Algebra

Example. Suppose we have document D , the $P(2)$ partition of D , and the query $\downarrow [\downarrow]$.

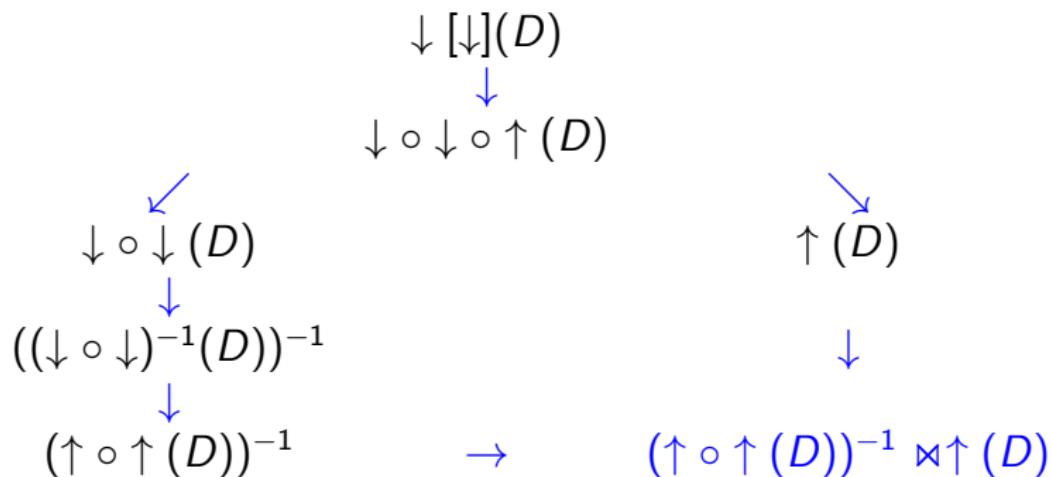
Then ...



Expression evaluation in XPath Algebra

Example. Suppose we have document D , the $P(2)$ partition of D , and the query $\downarrow [\downarrow]$.

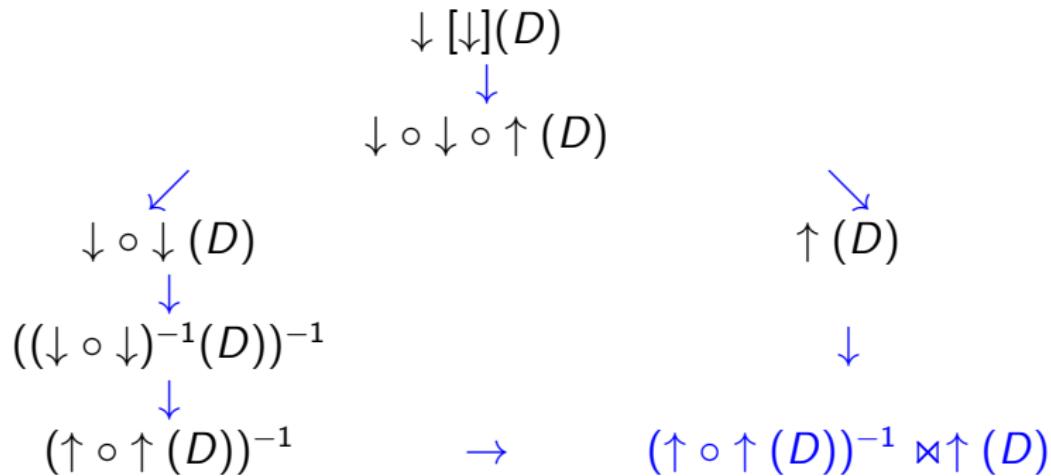
Then ...



Expression evaluation in XPath Algebra

Example. Suppose we have document D , the $P(2)$ partition of D , and the query $\downarrow [\downarrow]$.

Then ...



... and this can be evaluated directly over the $P(2)$ partition.

end of case study

Models of Indexing: query language model

Focuses on the issues set aside by the workload and GMAP models:

What is the relationship of a particular query language (and its syntax) to indexing?

Models of Indexing: query language model

Focuses on the issues set aside by the workload and GMAP models:

What is the relationship of a particular query language (and its syntax) to indexing?

Novel and powerful perspective, still relatively unexplored

Wrap up

Wrap up

- ▶ Indexing, part 2:
 - ▶ hash-based indexing
 - ▶ bitmap indexes
 - ▶ join indexing
 - ▶ models of indexability

Wrap up

- ▶ Indexing, part 2:
 - ▶ hash-based indexing
 - ▶ bitmap indexes
 - ▶ join indexing
 - ▶ models of indexability
- ▶ Teams and project part 2 have been posted.
Find your teammates and start studying your paper.

Wrap up

- ▶ Indexing, part 2:
 - ▶ hash-based indexing
 - ▶ bitmap indexes
 - ▶ join indexing
 - ▶ models of indexability
- ▶ Teams and project part 2 have been posted.
Find your teammates and start studying your paper.
- ▶ [Next time \(Wednesday 6 May\)](#): Query processing (i.e., evaluation of relational operators)

Credits

- ▶ Our textbook (Silberschatz *et al.*, 2011)
- ▶ Kifer *et al.*, 2006
- ▶ Valduriez, 1987

Query processing

Lecture 5
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

6 May 2015

Where we've been

Last time

- ▶ hash-based indexing
- ▶ join indexes
- ▶ models of indexing

Where we're headed

Today's agenda

- ▶ Evaluation of relational operators

The life of a query

- ▶ Employee(EID, EName, ECity)
- ▶ Company(CID, CName, CCity)
- ▶ WorksFor(EID, CID, Salary)

The life of a query

- ▶ Employee(EID, EName, ECity)
- ▶ Company(CID, CName, CCity)
- ▶ WorksFor(EID, CID, Salary)

```
SELECT E.EName  
FROM Employee E, WorksFor W  
WHERE E.EID = W.EID AND W.Salary > 5000
```

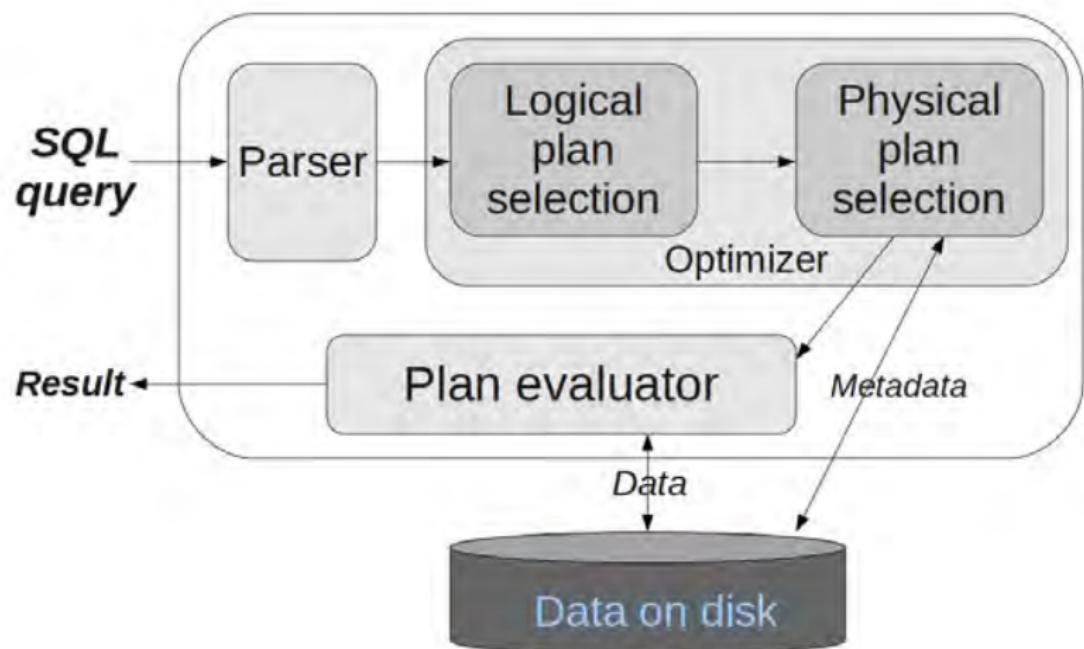
The life of a query

- ▶ Employee(EID, EName, ECity)
- ▶ Company(CID, CName, CCity)
- ▶ WorksFor(EID, CID, Salary)

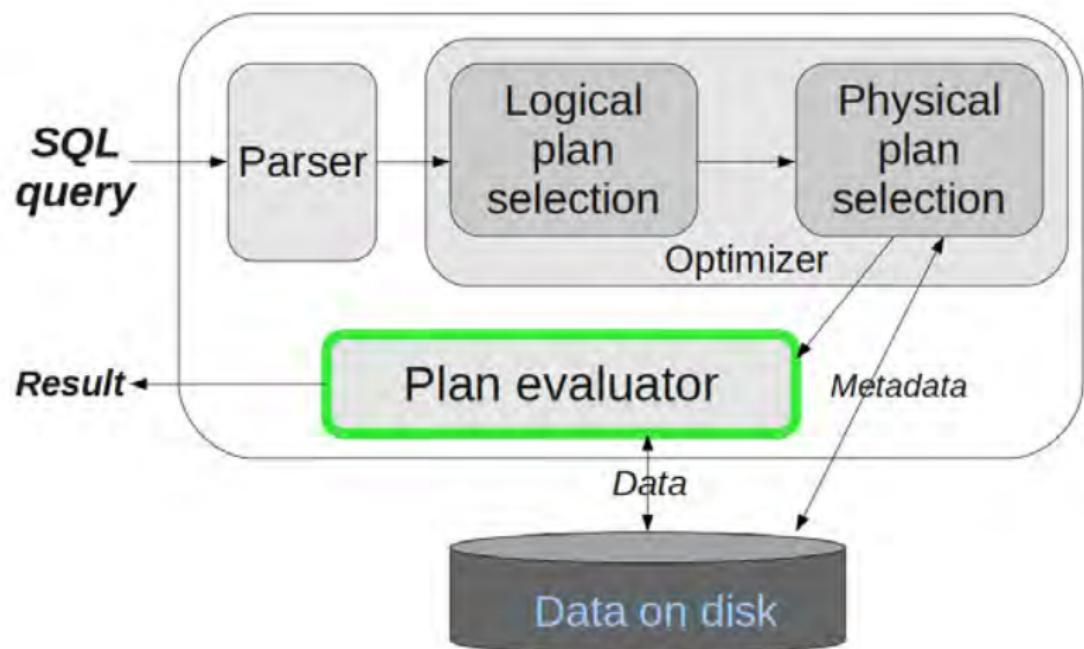
```
SELECT E.EName  
FROM Employee E, WorksFor W  
WHERE E.EID = W.EID AND W.Salary > 5000
```

$$\pi_{E.ename}(\sigma_{W.salary>5000}(E \bowtie_{E.eid=W.eid} W))$$

The life of a query



The life of a query: evaluation



The life of a query: plans & physical operators

- ▶ A physical query **plan** is what the evaluation engine executes (i.e., interprets)

The life of a query: plans & physical operators

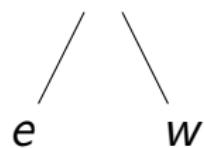
- ▶ A physical query **plan** is what the evaluation engine executes (i.e., interprets)
- ▶ A plan is a tree of physical operators
 - ▶ i.e., operators which access and manipulate physical data
- ▶ Each physical operator consumes a relation and outputs a relation

The life of a query: plans & physical operators

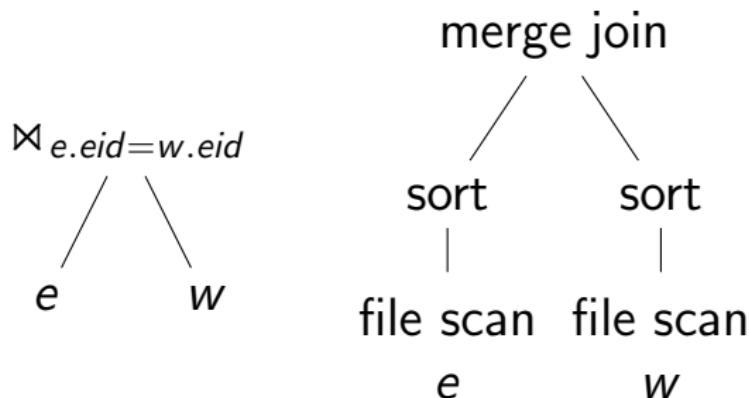
- ▶ A physical query **plan** is what the evaluation engine executes (i.e., interprets)
- ▶ A plan is a tree of physical operators
 - ▶ i.e., operators which access and manipulate physical data
- ▶ Each physical operator consumes a relation and outputs a relation
- ▶ (logical) RA operations may be mapped to multiple (physical) operators
 - ▶ and, there are often multiple mappings to choose from

The life of a query: plans & physical operators

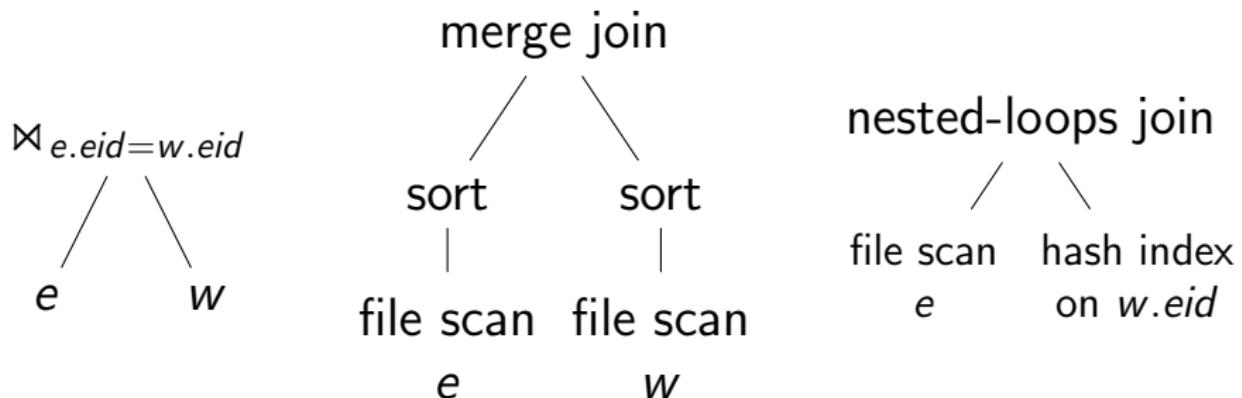
$\bowtie_{e.eid=w.eid}$



The life of a query: plans & physical operators



The life of a query: plans & physical operators



The life of a query: pipelining

- ▶ **pipelining**: read, process, propagate
 - ▶ the opposite is to materialize intermediate results
 - ▶ in practice, materialization becomes necessary (due to “blocking” operators, such as sorting)

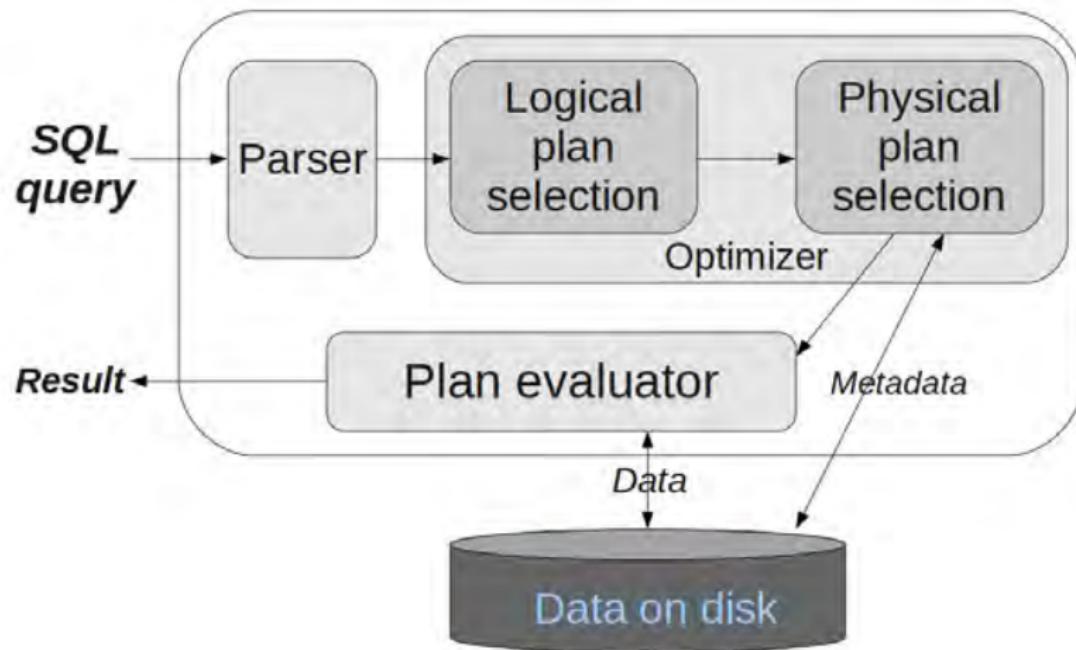
The life of a query: pipelining

- ▶ **pipelining**: read, process, propagate
 - ▶ the opposite is to materialize intermediate results
 - ▶ in practice, materialization becomes necessary (due to “blocking” operators, such as sorting)
- ▶ benefits of pipelining
 - ▶ no buffering
 - ▶ faster evaluation (since no I/O)
 - ▶ better resource utilization (more in-memory ops)

The life of a query: pipelining

- ▶ **pipelining**: read, process, propagate
 - ▶ the opposite is to materialize intermediate results
 - ▶ in practice, materialization becomes necessary (due to “blocking” operators, such as sorting)
- ▶ benefits of pipelining
 - ▶ no buffering
 - ▶ faster evaluation (since no I/O)
 - ▶ better resource utilization (more in-memory ops)
- ▶ hence, pipeline is simulated through the operator interface: `open()`, `getNext()`, `close()`
 - ▶ push model (buffering in calling operator)
 - ▶ pull model (buffering in called operator)
 - ▶ streams model (buffering in the connections)
- ▶ pull (demand-driven) model is common

The life of a query



- ▶ Optimizer is responsible for coming up with good physical plan

The life of a query: evaluation

Today: For each relational operation, how can we efficiently implement it?

- ▶ i.e., for each RA operation, what are some possible physical operators?

$$\sigma_{\theta}(R) \qquad \pi_{A_1, \dots, A_n}(R)$$

$$R \cup S \qquad R - S$$

$$R \cap S \qquad R \times S$$

$$R \bowtie S$$

Query evaluation

Three common techniques

- ▶ **iteration (scan)**: examine all tuples in a table or index

Query evaluation

Three common techniques

- ▶ **iteration (scan)**: examine all tuples in a table or index
- ▶ **partitioning**: of tuples on a sort key, and applying operations on buckets.
 - ▶ Sorting and hashing are commonly used partitioning techniques

Query evaluation

Three common techniques

- ▶ **iteration (scan)**: examine all tuples in a table or index
- ▶ **partitioning**: of tuples on a sort key, and applying operations on buckets.
 - ▶ Sorting and hashing are commonly used partitioning techniques
- ▶ **indexing**: if a selection or join condition is specified, use available index to inspect just those tuples satisfying the condition

Query evaluation: access paths

A method of retrieving tuples.

Query evaluation: access paths

A method of retrieving tuples. Either:

- ▶ a file scan, or

Query evaluation: access paths

A method of retrieving tuples. Either:

- ▶ a file scan, or
- ▶ an index plus a matching selection condition C
 - ▶ a hash index matches C if there is a term “att = val” in C for each attribute in the index’s search key
 - ▶ e.g., hash index on $(e.ecity, e.eid)$ can be used for
$$\sigma_{E.ecity=Delft \wedge E.eid=1234}(E)$$
 but not for
$$\sigma_{E.ecity=Delft}(E)$$

Query evaluation: access paths

A method of retrieving tuples. Either:

- ▶ a file scan, or
- ▶ an index plus a matching selection condition C
 - ▶ a hash index matches C if there is a term “att = val” in C for each attribute in the index’s search key
 - ▶ e.g., hash index on $(e.ecity, e.eid)$ can be used for $\sigma_{E.ecity=Delft \wedge E.eid=1234}(E)$ but not for $\sigma_{E.ecity=Delft}(E)$
 - ▶ a tree index matches C if there is a term “att θ val” for each attribute in a prefix of the index’s search key ($\theta \in \{<, \leq, =, \geq, >, \neq\}$)
 - ▶ e.g., B+tree on $(e.ecity, e.eid)$ can be used for $\sigma_{E.ecity=Delft}(E)$ and $\sigma_{E.ecity=Delft \wedge E.eid=1234}(E)$, but not for $\sigma_{E.eid=1234}(E)$

Query evaluation: access paths

Selectivity of an access path

- ▶ number of (index and data) pages retrieved
- ▶ “most selective” means retrieves fewest pages
 - ▶ cf. the notion of *access overhead* from last lecture

The selection operation σ

Suppose the employee relation E occupies $B_E = 1000$ pages and has $T_E = 10,000$ tuples.

The selection operation σ

Suppose the employee relation E occupies $B_E = 1000$ pages and has $T_E = 10,000$ tuples.

To evaluate $\sigma_{\text{ecity}=\text{Eindhoven}}(E)$, we can

- ▶ scan E (i.e., 1000 I/Os), or

The selection operation σ

Suppose the employee relation E occupies $B_E = 1000$ pages and has $T_E = 10,000$ tuples.

To evaluate $\sigma_{\text{ecity}=\text{Eindhoven}}(E)$, we can

- ▶ scan E (i.e., 1000 I/Os), or
- ▶ use index if available
 - ▶ if clustering, then great

The selection operation σ

Suppose the employee relation E occupies $B_E = 1000$ pages and has $T_E = 10,000$ tuples.

To evaluate $\sigma_{\text{ecity}=\text{Eindhoven}}(E)$, we can

- ▶ scan E (i.e., 1000 I/Os), or
- ▶ use index if available
 - ▶ if clustering, then great
 - ▶ if non-clustering, then potentially worse than scan (i.e., worst case is 10,000 I/Os)

The selection operation σ

$$\sigma_{att \ \theta \ val}(R)$$

The selection operation σ

$\sigma_{att \ \theta \ val}(R)$

- ▶ no index, unsorted data
 - ▶ iteration (i.e., full file scan)
 - ▶ $\mathcal{O}(B_R)$

The selection operation σ

$\sigma_{att \ \theta \ val}(R)$

- ▶ no index, unsorted data
 - ▶ iteration (i.e., full file scan)
 - ▶ $\mathcal{O}(B_R)$
- ▶ no index, sorted data
 - ▶ binary search
 - ▶ $\mathcal{O}(\log B_R)$

The selection operation σ

$\sigma_{att \ \theta \ val}(R)$

- ▶ no index, unsorted data
 - ▶ iteration (i.e., full file scan)
 - ▶ $\mathcal{O}(B_R)$
- ▶ no index, sorted data
 - ▶ binary search
 - ▶ $\mathcal{O}(\log B_R)$
- ▶ B+tree
 - ▶ clustered, and θ is not $=$, then best choice
 - ▶ unclustered, then need an estimate of selectivity of $att \ \theta \ val$

The selection operation σ

$\sigma_{att \ \theta \ val}(R)$

- ▶ no index, unsorted data
 - ▶ iteration (i.e., full file scan)
 - ▶ $\mathcal{O}(B_R)$
- ▶ no index, sorted data
 - ▶ binary search
 - ▶ $\mathcal{O}(\log B_R)$
- ▶ B+tree
 - ▶ clustered, and θ is not $=$, then best choice
 - ▶ unclustered, then need an estimate of selectivity of $att \ \theta \ val$
- ▶ Hash index, with equality selection (θ is $=$)
 - ▶ best choice

The projection operation π

```
SELECT DISTINCT Salary  
FROM WorksFor  
WHERE CID = 1234
```

$$\pi_{\text{Salary}}(\sigma_{cid=1234}(W))$$

The projection operation π

$\pi_{A_1, \dots, A_n}(R)$

- ▶ without duplicate elimination, scan or index
(clustered/unclustered doesn't matter)

The projection operation π

$\pi_{A_1, \dots, A_n}(R)$

- ▶ without duplicate elimination, scan or index
(clustered/unclustered doesn't matter)
- ▶ with duplicate elimination
 - ▶ remove unwanted attributes
 - ▶ eliminate any duplicates produced

The projection operation π

Two basic approaches to duplicate elimination

- ▶ sort-based: scan R producing projected tuples, sort result, and scan and eliminate adjacent duplicates
 - ▶ $\mathcal{O}(B_R \log B_R)$

The projection operation π

Two basic approaches to duplicate elimination

- ▶ **sort-based:** scan R producing projected tuples, sort result, and scan and eliminate adjacent duplicates
 - ▶ $\mathcal{O}(B_R \log B_R)$
- ▶ **hash-based:** take N buffer pages
 - ▶ partition each page of R at a time into $N - 1$ buckets, writing them out as they fill up. (diff buckets implies not duplicates)

The projection operation π

Two basic approaches to duplicate elimination

- ▶ **sort-based:** scan R producing projected tuples, sort result, and scan and eliminate adjacent duplicates
 - ▶ $\mathcal{O}(B_R \log B_R)$
- ▶ **hash-based:** take N buffer pages
 - ▶ partition each page of R at a time into $N - 1$ buckets, writing them out as they fill up. (diff buckets implies not duplicates)
 - ▶ read in each bucket and rehash in-memory with new hash function (collision implies duplicate)
 - ▶ write out resulting hash table after reading whole bucket

requires $N > B_R/N$

The union and difference operations \cup , $-$

```
SELECT CID
FROM Company
WHERE CCity = Eindhoven
UNION
SELECT CID
FROM WorksFor
WHERE Salary > 5000
```

The union and difference operations \cup , $-$

$$R \cup S$$

The union and difference operations \cup , $-$

$R \cup S$

- ▶ sort based
 - ▶ Sort R and S on all fields
 - ▶ scan results in parallel and merge, eliminating duplicates

The union and difference operations \cup , $-$

$R \cup S$

- ▶ sort based
 - ▶ Sort R and S on all fields
 - ▶ scan results in parallel and merge, eliminating duplicates
- ▶ hash based
 - ▶ Partition both R and S , using h_1
 - ▶ for each partition block P
 - ▶ build in-memory hash table for S_P using new h_2
 - ▶ scan R_P . For each tuple, probe hash table for S_P . If tuple is already in table, discard, otherwise insert.
 - ▶ write out table and clear for next partition

The union and difference operations \cup , $-$

```
SELECT CID
FROM Company
WHERE CCity = Eindhoven
EXCEPT
SELECT CID
FROM WorksFor
WHERE Salary > 5000
```

The union and difference operations \cup , $-$

$R - S$

The union and difference operations \cup , $-$

$R - S$

- ▶ sort based
 - ▶ Sort R and S on all fields
 - ▶ scan results in parallel, eliminating duplicates and tuples appearing in S

The union and difference operations \cup , $-$

$R - S$

- ▶ sort based
 - ▶ Sort R and S on all fields
 - ▶ scan results in parallel, eliminating duplicates and tuples appearing in S
- ▶ hash based
 - ▶ Partition both R and S , using h_1
 - ▶ for each partition block P
 - ▶ build in-memory hash table for S_P using new h_2
 - ▶ scan R_P . For each tuple, probe hash table for S_P . If tuple is not in table, write it out.
 - ▶ clear for next partition

The join operation \bowtie

```
SELECT E.EName  
FROM Employee E, WorksFor W  
WHERE E.EID = W.EID AND W.Salary > 5000
```

$$\pi_{E.ename}(\sigma_{W.salary > 5000}(E \bowtie_{E.eid=W.eid} W))$$

The join operation \bowtie

```
SELECT E.EName  
FROM Employee E, WorksFor W  
WHERE E.EID = W.EID AND W.Salary > 5000
```

$$\pi_{E.ename}(\sigma_{W.salary > 5000}(E \bowtie_{E.eid=W.eid} W))$$

The join operation \bowtie

- ▶ quite expensive, yet very common
 - ▶ e.g., due to table normalization
- ▶ intensively studied

The join operation \bowtie

- ▶ quite expensive, yet very common
 - ▶ e.g., due to table normalization
- ▶ intensively studied
- ▶ also, note that \cap (i.e., the INTERSECT operation in SQL) and \times are special cases of \bowtie

The join operation \bowtie

- ▶ *n.b.* choosing physical plan for a single join is different from choosing the order in which joins should be evaluated in the overall plan
 - ▶ in fact, the order in which joins are evaluated affects the choice of join algorithm
 - ▶ these two issues are very interrelated
- ▶ semantically, $R \bowtie S = S \bowtie R$
 - ▶ however, for physical join
 $cost(R \bowtie S) \neq cost(S \bowtie R)$

The join operation \bowtie

- ▶ *n.b.* choosing physical plan for a single join is different from choosing the order in which joins should be evaluated in the overall plan
 - ▶ in fact, the order in which joins are evaluated affects the choice of join algorithm
 - ▶ these two issues are very interrelated
- ▶ semantically, $R \bowtie S = S \bowtie R$
 - ▶ however, for physical join
$$\text{cost}(R \bowtie S) \neq \text{cost}(S \bowtie R)$$
- ▶ three main factors in determining cost:
 - ▶ input cardinalities T_R, T_S and number of pages B_R, B_S
 - ▶ selectivity factor of the join predicate
 - ▶ i.e., the ratio $\frac{|R \bowtie S|}{|R \times S|}$
 - ▶ available memory in buffer

The join operation \bowtie

four classes of join algorithms:

- ▶ iteration-based
- ▶ order-based
- ▶ partition-based
- ▶ special index-based

Iteration-based \bowtie

Nested-loops join

- ▶ simple, matching the semantics of \bowtie
- ▶ most flexible, for non-equi joins

Iteration-based \bowtie

Simple nested-loops join, $R \bowtie S$

- ▶ for each tuple $r \in R$
 - ▶ for each tuple $s \in S$
 - ▶ if $r \bowtie s$, then add (r, s) to result

Iteration-based \bowtie

Simple nested-loops join, $R \bowtie S$

- ▶ Call R the **outer** relation and S the **inner** relation
- ▶ One scan over the outer relation
- ▶ For each tuple in the outer relation, one scan over the inner relation

Iteration-based \bowtie

Simple nested-loops join, $R \bowtie S$

- ▶ Call R the **outer** relation and S the **inner** relation
- ▶ One scan over the outer relation
- ▶ For each tuple in the outer relation, one scan over the inner relation
- ▶ if relations **are not** clustered, then
$$\text{cost}(R \bowtie S) = \mathcal{O}(T_R + T_R \cdot T_S)$$

Iteration-based \bowtie

Simple nested-loops join, $R \bowtie S$

- ▶ Call R the **outer** relation and S the **inner** relation
- ▶ One scan over the outer relation
- ▶ For each tuple in the outer relation, one scan over the inner relation
- ▶ if relations **are not** clustered, then
$$\text{cost}(R \bowtie S) = \mathcal{O}(T_R + T_R \cdot T_S)$$
 - ▶ Suppose $T_R = T_S = 10,000$. Then $\text{cost}(R \bowtie S) = 10000 + 100,000,000 = 100,010,000$ I/Os!
 - ▶ if 15ms per I/O, then this is 417 hours (i.e., over 17 days)!

Iteration-based \bowtie

Simple nested-loops join, $R \bowtie S$

- ▶ if relations **are** clustered, then
 $cost(R \bowtie S) = \mathcal{O}(B_R + B_R \cdot B_S)$

Iteration-based \bowtie

Simple nested-loops join, $R \bowtie S$

- ▶ if relations **are** clustered, then

$$\text{cost}(R \bowtie S) = \mathcal{O}(B_R + B_R \cdot B_S)$$

- ▶ Suppose $B_R = B_S = 1000$. Then

$$\text{cost}(R \bowtie S) = 1000 + 1,000,000 = 1,001,000 \\ \text{I/Os!}$$

- ▶ this is still 4.17 hours!

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ We didn't make use of our buffer pool in simple nested-loop join
- ▶ as we saw with sorting, extra working space can have a non-trivial impact on cost

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ We didn't make use of our buffer pool in simple nested-loop join
- ▶ as we saw with sorting, extra working space can have a non-trivial impact on cost
- ▶ suppose the outer relation fits in the buffer, with at least two free pages left
 - ▶ then, we can put the outer relation in the buffer, and read the inner relation one page at a time

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ We didn't make use of our buffer pool in simple nested-loop join
- ▶ as we saw with sorting, extra working space can have a non-trivial impact on cost
- ▶ suppose the outer relation fits in the buffer, with at least two free pages left
 - ▶ then, we can put the outer relation in the buffer, and read the inner relation one page at a time
 - ▶ this costs us $\mathcal{O}(B_R + B_S)$, very nice!

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ We didn't make use of our buffer pool in simple nested-loop join
- ▶ as we saw with sorting, extra working space can have a non-trivial impact on cost
- ▶ suppose the outer relation fits in the buffer, with at least two free pages left
 - ▶ then, we can put the outer relation in the buffer, and read the inner relation one page at a time
 - ▶ this costs us $\mathcal{O}(B_R + B_S)$, very nice!
 - ▶ for our running example,
 $cost(R \bowtie S) = 1000 + 1000 = 2000$ I/Os

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ We didn't make use of our buffer pool in simple nested-loop join
- ▶ as we saw with sorting, extra working space can have a non-trivial impact on cost
- ▶ suppose the outer relation fits in the buffer, with at least two free pages left
 - ▶ then, we can put the outer relation in the buffer, and read the inner relation one page at a time
 - ▶ this costs us $\mathcal{O}(B_R + B_S)$, very nice!
 - ▶ for our running example,
 $cost(R \bowtie S) = 1000 + 1000 = 2000$ I/Os
 - ▶ and at 15ms per I/O, this takes 30 seconds!

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ now, what if we have N free buffer pages, and $B_R > N$?

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ now, what if we have N free buffer pages, and $B_R > N$?
- ▶ we can scan in $\lceil \frac{B_R}{N-2} \rceil$ blocks of size $N - 2$ of R , and compare S against each block

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ for each block of $N - 2$ pages of R
 - ▶ for each page of S
 - ▶ for all r in the current R -block and s in the current S -page such that $r \bowtie s$, add (r, s) to result

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ for each block of $N - 2$ pages of R
 - ▶ for each page of S
 - ▶ for all r in the current R -block and s in the current S -page such that $r \bowtie s$, add (r, s) to result

$$\text{cost}(R \bowtie S) = \mathcal{O}(B_R + \lceil \frac{B_R}{N-2} \rceil \cdot B_S) \text{ I/Os}$$

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ for each block of $N - 2$ pages of R
 - ▶ for each page of S
 - ▶ for all r in the current R -block and s in the current S -page such that $r \bowtie s$, add (r, s) to result

$$\text{cost}(R \bowtie S) = \mathcal{O}(B_R + \lceil \frac{B_R}{N-2} \rceil \cdot B_S) \text{ I/Os}$$

- ▶ so, for our running example, if $N = 102$, then
$$\text{cost}(R \bowtie S) = 1000 + 1000 \cdot \lceil \frac{1000}{100} \rceil = 11000 \text{ I/Os}$$

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ for each block of $N - 2$ pages of R
 - ▶ for each page of S
 - ▶ for all r in the current R -block and s in the current S -page such that $r \bowtie s$, add (r, s) to result

$$\text{cost}(R \bowtie S) = \mathcal{O}(B_R + \lceil \frac{B_R}{N-2} \rceil \cdot B_S) \text{ I/Os}$$

- ▶ so, for our running example, if $N = 102$, then
$$\text{cost}(R \bowtie S) = 1000 + 1000 \cdot \lceil \frac{1000}{100} \rceil = 11000 \text{ I/Os}$$
- ▶ and at 15ms per I/O, this takes 2.75 minutes!

Iteration-based \bowtie

Block nested-loops join, $R \bowtie S$

- ▶ the inner relation is scanned a number of times which is dependent on the size of the outer relation
- ▶ so, the outer should be chosen to be the smaller of the two!

Iteration-based \bowtie

Index nested-loops join, $R \bowtie S$

- ▶ what if we have an index available on the inner relation, on the join attribute?
- ▶ then, we can proceed just as simple nested-loops, except we use the index in the inner loop to perform predicate eval

Iteration-based \bowtie

Index nested-loops join, $R \bowtie S$

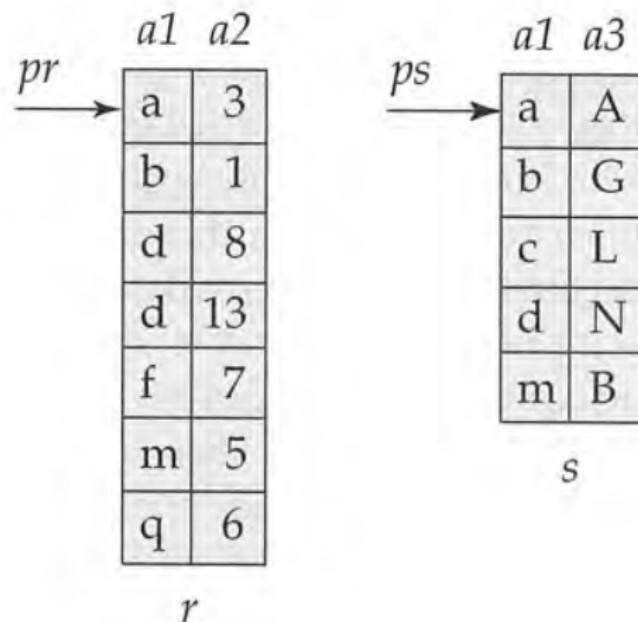
- ▶ what if we have an index available on the inner relation, on the join attribute?
- ▶ then, we can proceed just as simple nested-loops, except we use the index in the inner loop to perform predicate eval
- ▶ in the worst case, this costs $B_R + T_R \cdot I_S$, for average index access cost I_S on S
 - ▶ i.e., 3 or 4 I/Os for a B+tree, and 2 or 3 I/Os for a hash index
- ▶ if the outer relation is small, then this can lead to significant I/O savings

Order-based \bowtie

Sort-merge join $R \bowtie S$

- ▶ clean, simple idea:
 - ▶ sort R and S
 - ▶ scan together, and merge results
- ▶ key idea: there are groups in the sorted relations with the same value for the join attribute

Order-based \bowtie



Merge-join

Order-based \bowtie

Sort-merge join $R \bowtie S$

- ▶ Cost?

- ▶ sort R costs $2B_R \log B_R$
- ▶ sort S costs $2B_S \log B_S$
- ▶ merge is linear scan: $B_R + B_S$

Order-based \bowtie

Sort-merge join $R \bowtie S$

- ▶ Cost?
 - ▶ sort R costs $2B_R \log B_R$
 - ▶ sort S costs $2B_S \log B_S$
 - ▶ merge is linear scan: $B_R + B_S$
- ▶ so $\text{cost}(R \bowtie S)$ is the sum of these costs

Order-based \bowtie

Sort-merge join $R \bowtie S$

- ▶ Cost?
 - ▶ sort R costs $2B_R \log B_R$
 - ▶ sort S costs $2B_S \log B_S$
 - ▶ merge is linear scan: $B_R + B_S$
- ▶ so $\text{cost}(R \bowtie S)$ is the sum of these costs
- ▶ in our running example, we have 10,000 I/Os, which takes 2.5 minutes (about the same as block nested loops)

Partition-based \bowtie

Hash join $R \bowtie S$

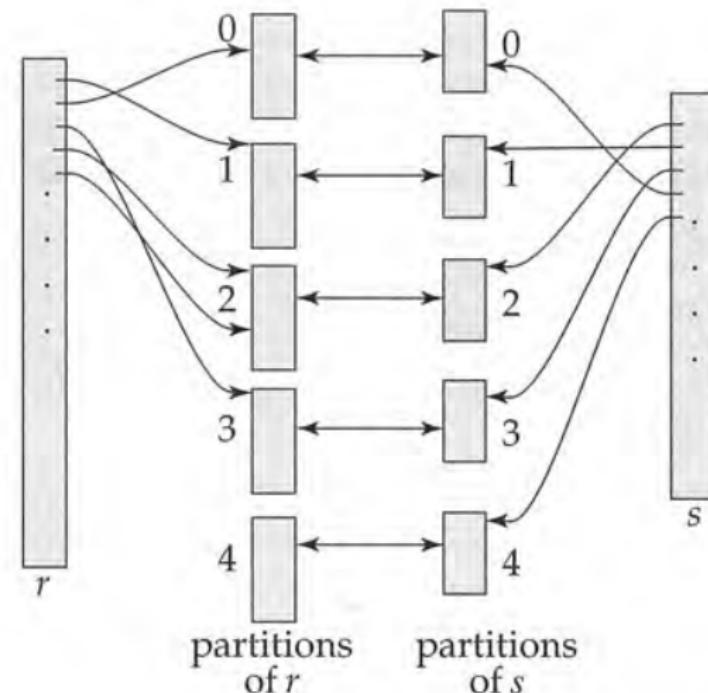
- ▶ partition-based
- ▶ key idea: using the same hash function,
 - ▶ partition R and S into m blocks $\{R_1, \dots, R_m\}$ $\{S_1, \dots, S_m\}$ such that each partition block fits in memory
 - ▶ tuples in R_i will only join with tuples in S_i

Partition-based \bowtie

Hash join $R \bowtie S$

- ▶ partition-based
- ▶ key idea: using the same hash function,
 - ▶ partition R and S into m blocks $\{R_1, \dots, R_m\}$ $\{S_1, \dots, S_m\}$ such that each partition block fits in memory
 - ▶ tuples in R_i will only join with tuples in S_i
 - ▶ then, for every R_i , load it in memory, scan S_i , and produce join results (just like block nested loops)

Partition-based \bowtie



Hash partitioning of R and S

Partition-based \bowtie

Hash join $R \bowtie S$: using the same hash function,

- ▶ partition R and S into m blocks $\{R_1, \dots, R_m\}$ $\{S_1, \dots, S_m\}$ such that each partition block fits in memory
 - ▶ tuples in R_i will only join with tuples in S_i
- ▶ then, for every R_i , load it in memory, scan S_i , and produce join results (just like block nested loops)

Exercise. Suppose

$$R = \{(14, x), (135, y), (40, x), (10, z)\},$$

$$S = \{(3, p), (10, q), (14, r), (10, s)\},$$

$$h(x) = x \% 5,$$

and you have four buffer slots each of which can hold two tuples. Illustrate the steps of computing $R \bowtie_{R.1=S.1} S$ with hash join using h .

Partition-based \bowtie

Hash join $R \bowtie S$: using the same hash function,

- ▶ partition R and S into m blocks $\{R_1, \dots, R_m\}$ $\{S_1, \dots, S_m\}$ such that each partition block fits in memory
 - ▶ tuples in R_i will only join with tuples in S_i
- ▶ then, for every R_i , load it in memory, scan S_i , and produce join results (just like block nested loops)

Exercise. Suppose

$$R = \{(14, x), (135, y), (40, x), (10, z)\},$$

$$S = \{(3, p), (10, q), (14, r), (10, s)\},$$

$$h(x) = x \% 5,$$

and you have four buffer slots each of which can hold two tuples. Illustrate the steps of computing $R \bowtie_{R.1=S.1} S$ with hash join using h .

What would happen if you had only three slots?

Partition-based \bowtie

Hash join $R \bowtie S$

- ▶ Cost?
 - ▶ $2(B_R + B_S)$ I/Os to build partitions
 - ▶ $(B_R + B_S)$ I/Os for probing and matching

Partition-based \bowtie

Hash join $R \bowtie S$

- ▶ Cost?
 - ▶ $2(B_R + B_S)$ I/Os to build partitions
 - ▶ $(B_R + B_S)$ I/Os for probing and matching
- ▶ $cost(R \bowtie S)$ is the sum of these costs

Partition-based \bowtie

Hash join $R \bowtie S$

- ▶ Cost?
 - ▶ $2(B_R + B_S)$ I/Os to build partitions
 - ▶ $(B_R + B_S)$ I/Os for probing and matching
- ▶ $\text{cost}(R \bowtie S)$ is the sum of these costs
- ▶ in our running example, we have 6,000 I/Os, which takes 1.5 minutes (about the same as block nested loops and sort-merge algorithms)

Using special datastructures for \bowtie

Recall the join index from last lecture

- ▶ Binary relation $\{(r_i, s_j), \dots\}$ over tuple surrogates in R and S , such that $r_i \bowtie s_j$

Using special datastructures for \bowtie

Recall the join index from last lecture

- ▶ Binary relation $\{(r_i, s_j), \dots\}$ over tuple surrogates in R and S , such that $r_i \bowtie s_j$

Algorithm:

- ▶ Scan join index, to find matching tuples (r_i, s_j)
- ▶ retrieve matching tuples
- ▶ add tuples to result

Using special datastructures for \bowtie

Cost of $R \bowtie S$ with join index:

- ▶ if R and S are clustered on join attributes,
then at worst we have
$$\text{AccessCost} + B_R + T_R \cdot \log B_S \text{ I/Os}$$
- ▶ if R and S are not clustered on join attributes,
then at worst we have
$$\text{AccessCost} + T_R + T_R \cdot \log B_S \text{ I/Os}$$

Using special datastructures for \bowtie

Cost of $R \bowtie S$ with join index:

- ▶ if R and S are clustered on join attributes,
then at worst we have
$$\text{AccessCost} + B_R + T_R \cdot \log B_S \text{ I/Os}$$
- ▶ if R and S are not clustered on join attributes,
then at worst we have
$$\text{AccessCost} + T_R + T_R \cdot \log B_S \text{ I/Os}$$

In our running example, we have

$$1000 + 10000 \cdot 10 = 101,000 \text{ I/Os}$$

Using special datastructures for \bowtie

Cost of $R \bowtie S$ with join index:

- ▶ if R and S are clustered on join attributes,
then at worst we have
$$\text{AccessCost} + B_R + T_R \cdot \log B_S \text{ I/Os}$$
- ▶ if R and S are not clustered on join attributes,
then at worst we have
$$\text{AccessCost} + T_R + T_R \cdot \log B_S \text{ I/Os}$$

In our running example, we have

$$1000 + 10000 \cdot 10 = 101,000 \text{ I/Os}$$

Best suited for highly selective predicates ...

The join operation \bowtie

- ▶ All join algorithms work on equi-join predicates
 - ▶ only nested loops and join-index algorithms work for non equi-join predicates
 - ▶ fortunately, equi-join is the most common join type

The join operation \bowtie

- ▶ All join algorithms work on equi-join predicates
 - ▶ only nested loops and join-index algorithms work for non equi-join predicates
 - ▶ fortunately, equi-join is the most common join type
- ▶ Join is most optimized physical operator
 - ▶ Four classes: iteration, order, partition, special datastructures

The join operation ✕

- ▶ All join algorithms work on equi-join predicates
 - ▶ only nested loops and join-index algorithms work for non equi-join predicates
 - ▶ fortunately, equi-join is the most common join type
- ▶ Join is most optimized physical operator
 - ▶ Four classes: iteration, order, partition, special datastructures
- ▶ Figuring out the best join algorithm for a particular pair of relations (in the context of a larger query plan) is the job of the query optimizer
 - ▶ important choice, since we are talking about seconds vs. days!

Wrap up

- ▶ Query processing
 - ▶ Evaluating selections, projections, and binary ops
 - ▶ Evaluating joins

Wrap up

- ▶ Query processing
 - ▶ Evaluating selections, projections, and binary ops
 - ▶ Evaluating joins
- ▶ Next lecture: Data statistics & Views

Image credits

- ▶ Our textbook (Silberschatz *et al.*, 2006)

Data statistics for query optimization

&

Views: maintenance and use

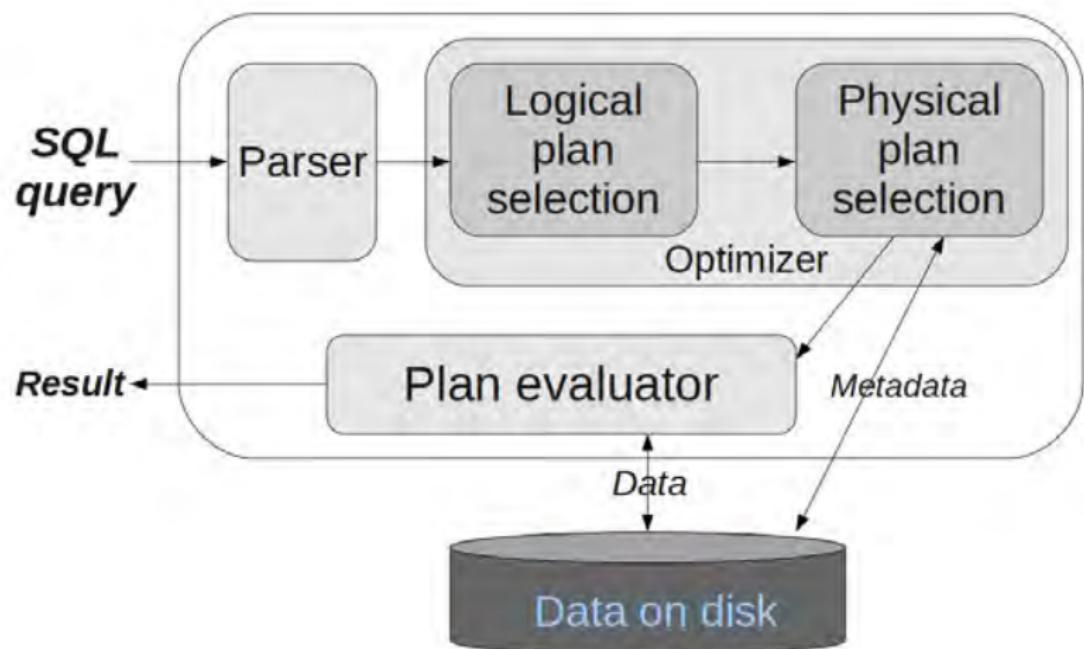
Lecture 6
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

8 May 2015

The life of a query

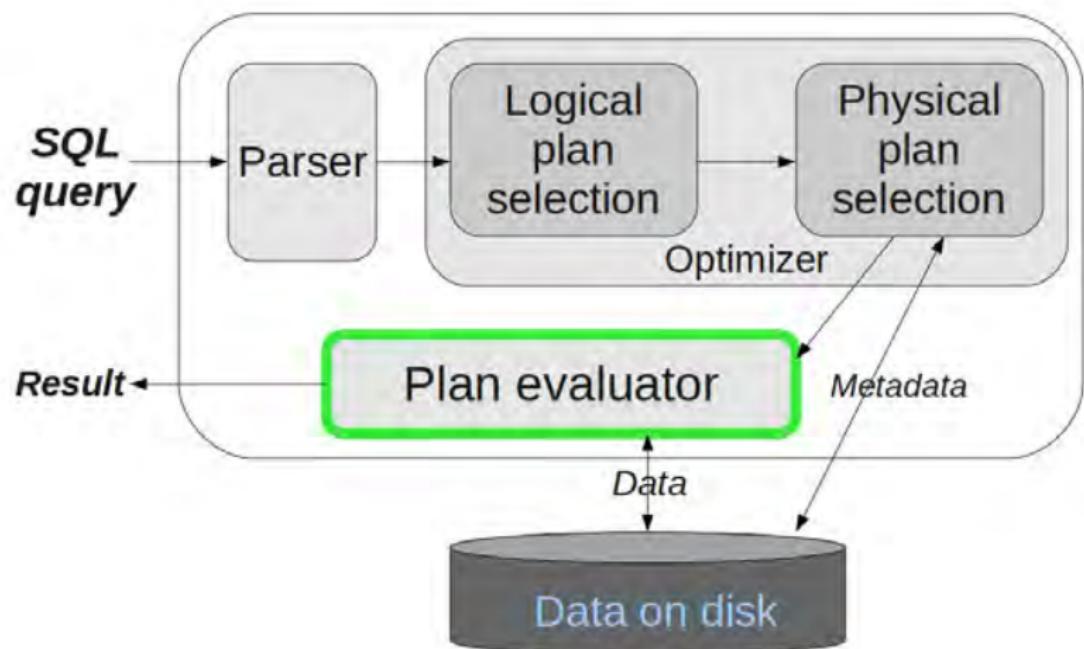


Where we've been

query evaluation

- ▶ physical processing of relational operators

The life of a query: evaluation



Where we're headed

query optimization

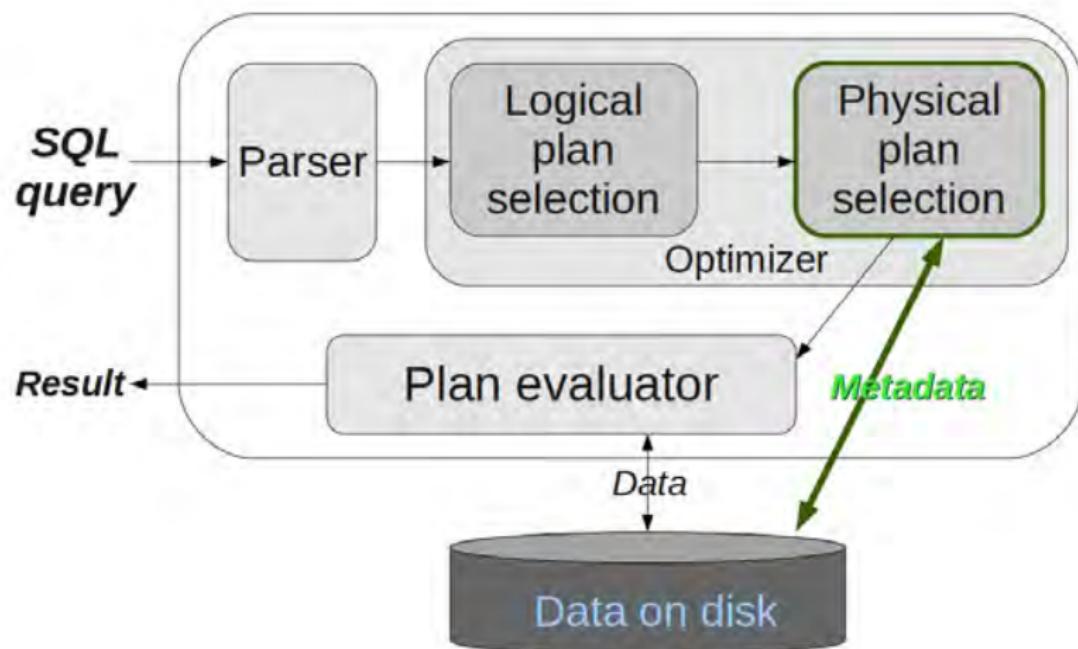
- ▶ logical optimization
- ▶ physical optimization

Where we're headed

query optimization

- ▶ logical optimization
- ▶ physical optimization
 - ▶ today: cost estimation using data statistics

The life of a query: cost estimation



The life of a query: cost estimation

- ▶ A plan is a tree of physical operators
 - ▶ i.e., operators which access and manipulate physical data

The life of a query: cost estimation

- ▶ A plan is a tree of physical operators
 - ▶ i.e., operators which access and manipulate physical data
- ▶ physical plan indicates
 - ▶ algorithm for each node
 - ▶ the way stored data is obtained (i.e., access paths)
 - ▶ the way in which data is passed between nodes
 - ▶ the order in which nodes are performed

The life of a query: cost estimation

- ▶ two parts to estimating the cost of a plan
 - ▶ for each node, estimate the cost of performing the operation
 - ▶ for each node, estimate the result size, and any properties it might have (e.g., sorted)

The life of a query: cost estimation

- ▶ two parts to estimating the cost of a plan
 - ▶ for each node, estimate the cost of performing the operation
 - ▶ for each node, estimate the result size, and any properties it might have (e.g., sorted)
- ▶ the overall estimate is obtained by combining these local estimates
 - ▶ note, however, that errors in estimates propagate exponentially ...

The life of a query: cost estimation

- ▶ two parts to estimating the cost of a plan
 - ▶ for each node, estimate the cost of performing the operation
 - ▶ for each node, estimate the result size, and any properties it might have (e.g., sorted)
- ▶ the overall estimate is obtained by combining these local estimates
 - ▶ note, however, that errors in estimates propagate exponentially ...
- ▶ keep in mind:
 - ▶ cost estimates are truly approximations
 - ▶ goal is really to just avoid the *worst* plans

The life of a query: cost estimation

reduction factor:

- ▶ the fraction of tuples which satisfy a condition C is called the reduction factor of C

The life of a query: cost estimation

reduction factor:

- ▶ the fraction of tuples which satisfy a condition C is called the reduction factor of C
- ▶ collectively, the reduction factor of $C_1 \wedge \cdots \wedge C_m$ is $rf_1 \times \cdots \times rf_m$ (assuming statistical independence)

The life of a query: cost estimation

```
SELECT A1, ..., Ak  
FROM R1, ..., Rn  
WHERE C1 AND ... AND Cm
```

The life of a query: cost estimation

```
SELECT A1, ..., Ak  
FROM R1, ..., Rn  
WHERE C1 AND ... AND Cm
```

max size:

$$T_{R_1} \times \cdots \times T_{R_n}$$

The life of a query: cost estimation

```
SELECT A1, ..., Ak  
FROM R1, ..., Rn  
WHERE C1 AND ... AND Cm
```

max size:

$$T_{R_1} \times \cdots \times T_{R_n}$$

estimate of actual size:

$$(rf_1 \times \cdots \times rf_m) \times (T_{R_1} \times \cdots \times T_{R_n})$$

The life of a query: cost estimation

- ▶ for each RA implementation, we discussed cost

The life of a query: cost estimation

- ▶ for each RA implementation, we discussed cost
- ▶ the parameters to cost were
 - ▶ input relation size (pages and/or tuples), and
 - ▶ available buffer space

The life of a query: cost estimation

- ▶ for each RA implementation, we discussed cost
- ▶ the parameters to cost were
 - ▶ input relation size (pages and/or tuples), and
 - ▶ available buffer space
- ▶ let's consider “quick-and-dirty” heuristics for estimating result size (under the assumption of uniform distribution of values)

Result size estimation: π

For **projection**, actually computable

Result size estimation: π

For **projection**, actually computable

- ▶ reduction factor is $\frac{\text{new tuple size}}{\text{old tuple size}}$

Result size estimation: π

For **projection**, actually computable

- ▶ reduction factor is $\frac{\text{new tuple size}}{\text{old tuple size}}$
- ▶ impacts the number of pages in output

Result size estimation: σ

selection

- ▶ let $V(R, A)$ denote the number of distinct values appearing in attribute A of relation R

Result size estimation: σ

selection

- ▶ let $V(R, A)$ denote the number of distinct values appearing in attribute A of relation R
 - ▶ by default, we choose $V(R, A) = 10$
 - ▶ some systems keep actual counts
 - ▶ if there is an index on $R.A$, then $V(R, A)$ is equal to the number of keys

Result size estimation: σ

selection

- ▶ let $V(R, A)$ denote the number of distinct values appearing in attribute A of relation R
 - ▶ by default, we choose $V(R, A) = 10$
 - ▶ some systems keep actual counts
 - ▶ if there is an index on $R.A$, then $V(R, A)$ is equal to the number of keys
- ▶ then, the size estimate of $\sigma_{A=c}(R)$ is

$$\frac{1}{V(R, A)} T_R$$

Result size estimation: σ

For the size estimate of $\sigma_{A>c}(R)$

- ▶ if A is not an arithmetic type, then

$$\frac{1}{3} T_R$$

Result size estimation: σ

For the size estimate of $\sigma_{A>c}(R)$

- ▶ if A is not an arithmetic type, then

$$\frac{1}{3} T_R$$

- ▶ else

$$\frac{\text{highVal}(A) - c}{\text{highVal}(A) - \text{lowVal}(A)} T_R$$

Result size estimation: σ

- ▶ for the size estimate of $\sigma_{A \neq c}(R)$

$$T_R$$

Result size estimation: σ

- ▶ for the size estimate of $\sigma_{A \neq c}(R)$

$$T_R$$

- ▶ for the size estimate of $\sigma_{C_1 \vee C_2}(R)$

$$\min \{ T_r, \text{size}(\sigma_{C_1}(R)) + \text{size}(\sigma_{C_2}(R)) \}$$

Result size estimation: σ

- ▶ for the size estimate of $\sigma_{A \neq c}(R)$

$$T_R$$

- ▶ for the size estimate of $\sigma_{C_1 \vee C_2}(R)$

$$\min \{ T_r, \text{size}(\sigma_{C_1}(R)) + \text{size}(\sigma_{C_2}(R)) \}$$

- ▶ for the size estimate of $\sigma_{C_1 \wedge C_2}(R)$

$$rf_{C_1} \times rf_{C_2} \times T_r$$

Result size estimation: \cup

$$|R \cup S| \approx \frac{(T_R + T_S) + \max\{T_R, T_S\}}{2}$$

Result size estimation: \cup

$$\begin{aligned}|R \cup S| &\approx \frac{(T_R + T_S) + \max\{T_R, T_S\}}{2} \\&= \frac{(2 \times \text{larger}) + \text{smaller}}{2}\end{aligned}$$

Result size estimation: \cup

$$\begin{aligned}|R \cup S| &\approx \frac{(T_R + T_S) + \max\{T_R, T_S\}}{2} \\&= \frac{(2 \times \text{larger}) + \text{smaller}}{2} \\&= \text{larger} + \frac{\text{smaller}}{2}\end{aligned}$$

Result size estimation: \cap

For intersection $R \cap S$, the minimum result size is 0, and the max size is $\min\{T_R, T_S\}$.

Result size estimation: \cap

For intersection $R \cap S$, the minimum result size is 0, and the max size is $\min\{T_R, T_S\}$.

Hence,

$$|R \cap S| \approx \frac{0 + \min\{T_R, T_S\}}{2}$$

Result size estimation: \cap

For intersection $R \cap S$, the minimum result size is 0, and the max size is $\min\{T_R, T_S\}$.

Hence,

$$\begin{aligned}|R \cap S| &\approx \frac{0 + \min\{T_R, T_S\}}{2} \\&= \frac{\text{smaller}}{2}\end{aligned}$$

Result size estimation: —

For difference $R - S$, the minimum result size is $T_R - T_S$, and the max size is T_R .

Result size estimation: —

For difference $R - S$, the minimum result size is $T_R - T_S$, and the max size is T_R .

Hence,

$$|R - S| \approx \frac{T_R + (T_R - T_S)}{2}$$

Result size estimation: —

For difference $R - S$, the minimum result size is $T_R - T_S$, and the max size is T_R .

Hence,

$$\begin{aligned}|R - S| &\approx \frac{T_R + (T_R - T_S)}{2} \\&= T_R - \frac{1}{2} T_S\end{aligned}$$

Result size estimation: \bowtie

For natural join $R \bowtie S$ on attribute Y :

$$|R \bowtie S| \approx \min \left\{ T_R \frac{T_S}{V(S, Y)}, T_S \frac{T_R}{V(R, Y)} \right\}$$

Result size estimation: \bowtie

For natural join $R \bowtie S$ on attribute Y :

$$|R \bowtie S| \approx \min \left\{ T_R \frac{T_S}{V(S, Y)}, T_S \frac{T_R}{V(R, Y)} \right\}$$

since,

- ▶ if $V(R, Y) = V(S, Y)$, then each tuple of R joins with approximately $\frac{T_S}{V(S, Y)}$ tuples of R
 - ▶ and, each tuple of S joins with approximately $\frac{T_R}{V(R, Y)}$ tuples of S

Result size estimation: \bowtie

For natural join $R \bowtie S$ on attribute Y :

$$|R \bowtie S| \approx \min \left\{ T_R \frac{T_S}{V(S, Y)}, T_S \frac{T_R}{V(R, Y)} \right\}$$

since,

- ▶ if $V(R, Y) = V(S, Y)$, then each tuple of R joins with approximately $\frac{T_S}{V(S, Y)}$ tuples of R
 - ▶ and, each tuple of S joins with approximately $\frac{T_R}{V(R, Y)}$ tuples of S
- ▶ and if not, then we minimize the contribution of dangling tuples

Result size estimation: assumptions made

We have made a few assumptions in these estimates:

Result size estimation: assumptions made

We have made a few assumptions in these estimates:

- ▶ values across columns are not correlated
 - ▶ this assumption is hard to lift (active area of research)

Result size estimation: assumptions made

We have made a few assumptions in these estimates:

- ▶ values across columns are not correlated
 - ▶ this assumption is hard to lift (active area of research)
- ▶ values in a single column are uniformly distributed
 - ▶ this assumption can be lifted with better statistics

Result size estimation: histograms

Histograms: simple data structures for more refined computation of reduction factors

Result size estimation: histograms

Histograms: simple data structures for more refined computation of reduction factors

Provides approximation of value distribution of an attribute in a relation instance

- ▶ *small*: typically fit on one disk page
- ▶ *accurate*: typically, less than 5% error

Result size estimation: histograms

Histograms: simple data structures for more refined computation of reduction factors

Provides approximation of value distribution of an attribute in a relation instance

- ▶ *small*: typically fit on one disk page
- ▶ *accurate*: typically, less than 5% error

two basic types:

- ▶ equi-width
- ▶ equi-depth

Result size estimation: histograms

Equi-width, on column A

- ▶ divide range of values appearing in A into equal sized sub-ranges
- ▶ compute and store total number of tuples falling into each of these “buckets”

Result size estimation: histograms

Equi-width, on column A

- ▶ divide range of values appearing in A into equal sized sub-ranges
- ▶ compute and store total number of tuples falling into each of these “buckets”
- ▶ to estimate the output cardinality of a range query on A , find starting bucket, and scan forward until ending bucket is identified
- ▶ sum number of tuples seen, assuming uniform distribution of values within buckets

Result size estimation: histograms

Equi-depth, on column A

- ▶ divide range of values appearing in A into sub-ranges, such that each bucket contains the same number of tuples

Result size estimation: histograms

Equi-depth, on column A

- ▶ divide range of values appearing in A into sub-ranges, such that each bucket contains the same number of tuples
- ▶ use the same algorithm to approximate the number of tuples falling in a range query over A

Result size estimation: histograms

Example. Consider a “Sales” attribute with the following actual value distribution:

Number of tuples	Sales value
10	0.5 mil
10	1 mil
10	2 mil
5	5 mil
5	7 mil
5	15 mil
2	40 mil
1	50 mil
1	70 mil
1	100 mil

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

equi-width		equi-depth	
value range	tuple count	value range	tuple count
0-20	45	0-0.5	10
20-40	2	0.5-1	10
40-60	1	1-2	10
60-80	1	2-7	10
80-100	1	7-100	10

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

equi-width		equi-depth	
value range	tuple count	value range	tuple count
0-20	45	0-0.5	10
20-40	2	0.5-1	10
40-60	1	1-2	10
60-80	1	2-7	10
80-100	1	7-100	10

The selectivity estimate of “sales ≤ 10 million” for equi-width is $\frac{23}{50} = 46\%$ and for equi-depth is $\frac{40}{50} = 80\%$

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

equi-width		equi-depth	
value range	tuple count	value range	tuple count
0-20	45	0-0.5	10
20-40	2	0.5-1	10
40-60	1	1-2	10
60-80	1	2-7	10
80-100	1	7-100	10

The selectivity estimate of “sales ≤ 10 million” for equi-width is $\frac{23}{50} = 46\%$ and for equi-depth is $\frac{40}{50} = 80\%$ which is the actual selectivity!

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

equi-width		equi-depth	
value range	tuple count	value range	tuple count
0-20	45	0-0.5	10
20-40	2	0.5-1	10
40-60	1	1-2	10
60-80	1	2-7	10
80-100	1	7-100	10

The selectivity estimate of “sales ≤ 5 million” for equi-width is $\frac{12}{50} = 24\%$ and for equi-depth is $\frac{35}{50} = 70\%$

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

equi-width		equi-depth	
value range	tuple count	value range	tuple count
0-20	45	0-0.5	10
20-40	2	0.5-1	10
40-60	1	1-2	10
60-80	1	2-7	10
80-100	1	7-100	10

The selectivity estimate of “sales ≤ 5 million” for equi-width is $\frac{12}{50} = 24\%$ and for equi-depth is $\frac{35}{50} = 70\%$ which is again the actual selectivity!

Result size estimation: histograms

Equi-depth:

- ▶ more accurate than equi-width, since buckets with frequently occurring values correspond to smaller ranges, hence giving finer approximations
- ▶ effective, simple approach to selectivity estimation, and hence quite common

Result size estimation: histograms

Exercise. Consider a “Friends” attribute with the following actual value distribution:

Friends value	Number of tuples
0	1
1	3
2	6
3	10
4	3
5	2
6	2
7	2
8	1

Construct equi-depth and equi-width histograms over this attribute, using three buckets. What estimates do they give for the count of tuples with “friends ≥ 4 ”?

Wrap up

- ▶ Cost estimation, towards choosing a good physical plan

Wrap up

- ▶ Cost estimation, towards choosing a good physical plan
- ▶ **After the break**
 - ▶ Answering queries using views

Views

Where we've been

cost estimation using data statistics

Where we're headed

query optimization:

- ▶ logical optimization
- ▶ physical optimization

Where we're headed

query optimization:

- ▶ logical optimization
- ▶ physical optimization
- ▶ next: the creation, maintenance, and use of “views”

Views

Virtual/derived relations, providing alternative logical schemata

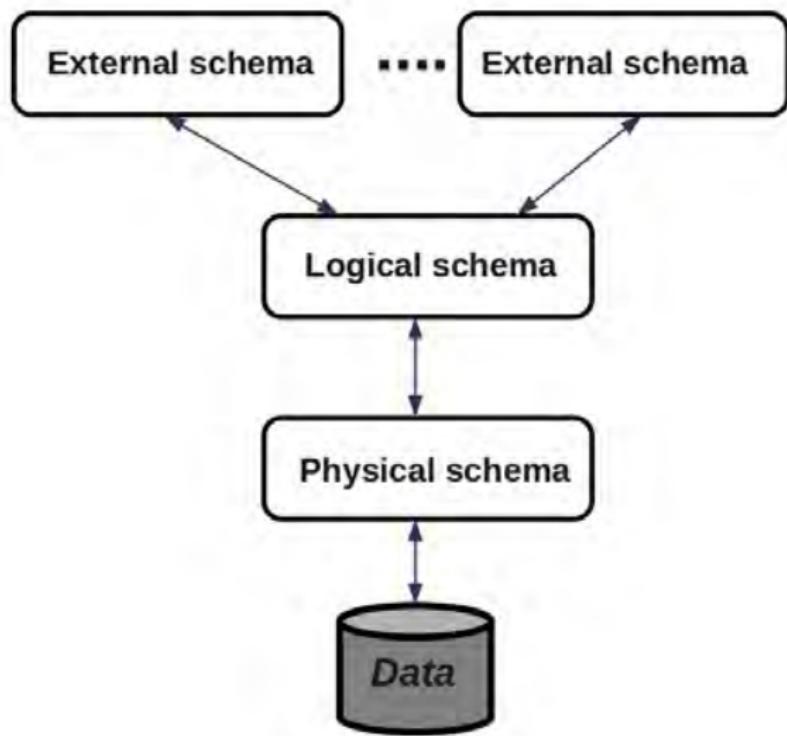
Views

Virtual/derived relations, providing alternative logical schemata

Often desirable to provide

- ▶ security
- ▶ efficiency
- ▶ logical data independence

Data independence



Views

We've already studied a few special cases:

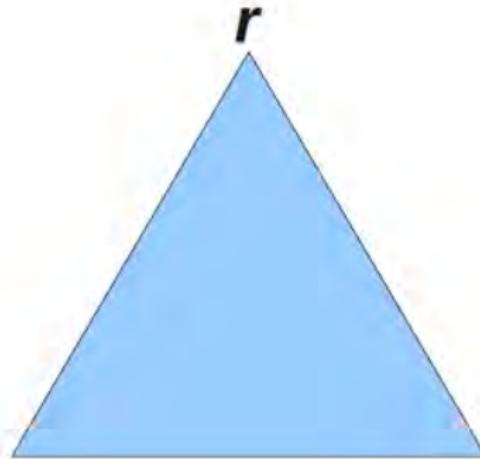
- ▶ Histograms/statistics as views

Views

We've already studied a few special cases:

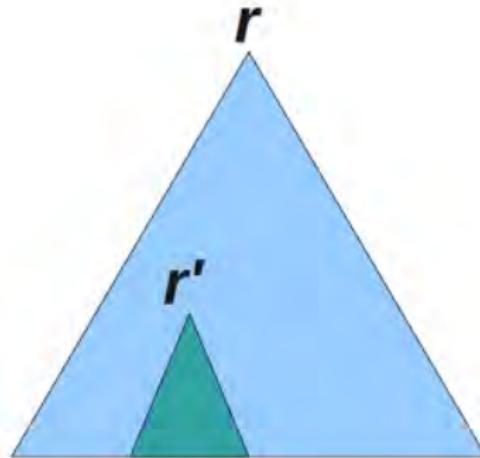
- ▶ Histograms/statistics as views
- ▶ Indexes as (family of) views

Indexes as views



predicate(r)

Indexes as views



$\textit{predicate}(r)$ vs. $\textit{predicate}(r')$

Views

We've already studied a few special cases:

- ▶ Histograms/statistics as views
- ▶ Indexes as (family of) views
 - ▶ trees as hierarchical histograms: annotate nodes with actual or approximate counts of items in subtrees

Views

We've already studied a few special cases:

- ▶ Histograms/statistics as views
- ▶ Indexes as (family of) views
 - ▶ trees as hierarchical histograms: annotate nodes with actual or approximate counts of items in subtrees
- ▶ GMAPs

Views

In SQL, views are created with the CREATE VIEW statement

- ▶ `CREATE VIEW view_name AS expression`

Views

In SQL, views are created with the CREATE VIEW statement

- ▶ CREATE VIEW *view_name* AS *expression*

- ▶ for example

```
CREATE VIEW mng_view AS  
SELECT name, address, phone  
FROM emp  
WHERE title='manager'
```

Views

Basic issues with views:

Views

Basic issues with views:

- ▶ creation and implementation
- ▶ maintenance under updates
- ▶ answering queries using views: query containment

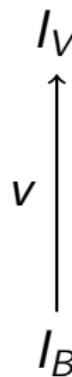
Updating views

Views

I_B

“Base” instance I_B and view v

Views



“Base” instance I_B and view v

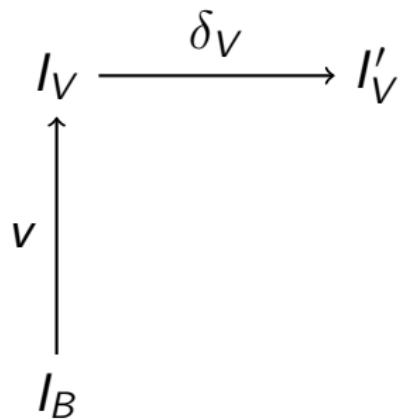
Views

- ▶ We'd like to allow users to treat the view instance I_V just like any other (base) relation.

Views

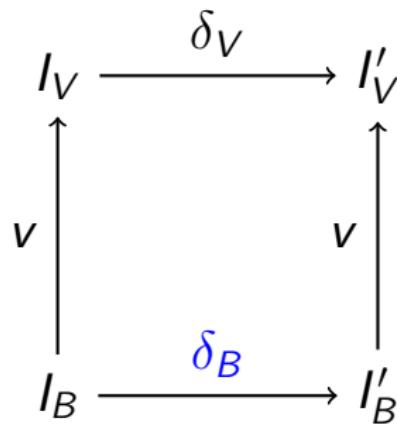
- ▶ We'd like to allow users to treat the view instance I_V just like any other (base) relation.
- ▶ in particular, it would be nice to support both queries and updates on I_V

Updating a view



Given update δ_V on view instance I_V , find appropriate update δ_B on the base data I_B

Updating a view



Given update δ_V on view instance I_V , find appropriate update δ_B on the base data I_B

Updating a view

Unfortunately, not all views are directly updateable

- ▶ `INSERT INTO mng_view
VALUES (Fred, Eindhoven, 1234567)`

Updating a view

Unfortunately, not all views are directly updateable

- ▶

```
INSERT INTO mng_view
VALUES (Fred, Eindhoven, 1234567)
```
- ▶ how to update the EMP table, if it has a salary field? What should Fred's salary be?

Updating a view

Another example

Suppose we have base table $\text{edge}(x, y)$, defining a directed graph, and the view

$$\text{hop}(x, y) = \text{edge} \bowtie \text{edge}$$

(i.e., paths of length 2)

Updating a view

Another example

Suppose we have base table $\text{edge}(x, y)$, defining a directed graph, and the view

$$\text{hop}(x, y) = \text{edge} \bowtie \text{edge}$$

(i.e., paths of length 2)

How should we handle an insert on hop ? How should we materialize this in edge ?

Updating a view

this is an active research topic!

- ▶ Can couple all view definitions with an appropriate “update policy”, using some formalism (e.g., so-called lenses)

Updating a view

this is an active research topic!

- ▶ Can couple all view definitions with an appropriate “update policy”, using some formalism (e.g., so-called lenses)
- ▶ DBA can build trigger on view, to enforce “reasonable” behavior

Updating a view

this is an active research topic!

- ▶ Can couple all view definitions with an appropriate “update policy”, using some formalism (e.g., so-called lenses)
- ▶ DBA can build trigger on view, to enforce “reasonable” behavior

```
CREATE TRIGGER mng_trigger
INSTEAD OF INSERT ON mng_view
BEGIN
    INSERT INTO emp VALUES
    (NEW.name, NEW.address, NEW.phone,
     'manager', $0);
END;
```

Updating a view

Can proceed with simple restrictions

Updating a view

Can proceed with simple restrictions

A view is *updateable* if

- ▶ FROM clause has only one relation,

Updating a view

Can proceed with simple restrictions

A view is *updateable* if

- ▶ FROM clause has only one relation,
- ▶ SELECT clause contains only attributes (no expressions, etc.),

Updating a view

Can proceed with simple restrictions

A view is *updateable* if

- ▶ FROM clause has only one relation,
- ▶ SELECT clause contains only attributes (no expressions, etc.),
- ▶ any attribute not listed in SELECT can be set to NULL, and

Updating a view

Can proceed with simple restrictions

A view is *updateable* if

- ▶ FROM clause has only one relation,
- ▶ SELECT clause contains only attributes (no expressions, etc.),
- ▶ any attribute not listed in SELECT can be set to NULL, and
- ▶ no GROUP BY or HAVING clauses.

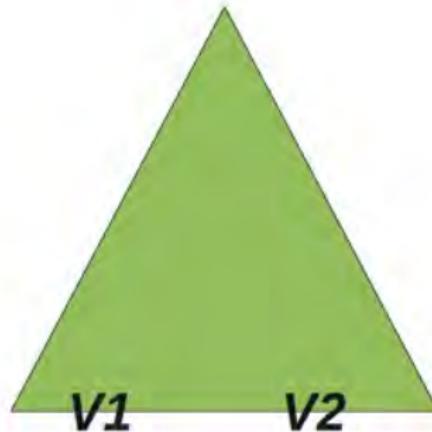
Implementing views

Implementing views

Two alternatives for view implementation

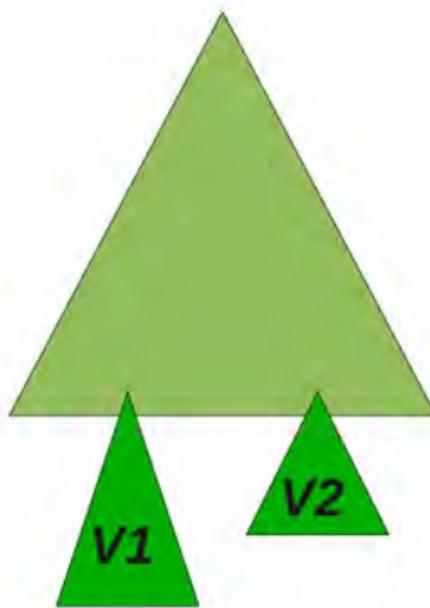
- ▶ virtual views: unfold any use of views in a query

Virtual view unfolding



expression parse tree, using views V_1 and V_2

Virtual view unfolding



expression parse tree, with V_1 and V_2 replaced with their definitions

Virtual view unfolding

For example

$$\sigma_{address=Eindhoven}(MngView)$$

Virtual view unfolding

For example

$$\sigma_{address=Eindhoven}(MngView)$$

becomes

$$\sigma_{address=Eindhoven}(\pi_{address, name, phone}(\sigma_{title=Manager}(Emp)))$$

Views

Two alternatives for implementation

- ▶ virtual views: unfold any use of views in a query

Views

Two alternatives for implementation

- ▶ virtual views: unfold any use of views in a query
- ▶ materialization

Materialized views

- ▶ precompute and store view results

Materialized views

- ▶ precompute and store view results
- ▶ more efficient for frequently used and/or expensive views
 - ▶ ex: the ATM view of your bank account

Materialized views

- ▶ precompute and store view results
- ▶ more efficient for frequently used and/or expensive views
 - ▶ ex: the ATM view of your bank account
 - ▶ ex: view which denormalizes (i.e., incurs costly joins)

Materialized views

Apps

- ▶ query optimization

Materialized views

Apps

- ▶ query optimization
- ▶ data warehousing
 - ▶ integration of data
 - ▶ OLAP

Materialized views

Apps

- ▶ query optimization
- ▶ data warehousing
 - ▶ integration of data
 - ▶ OLAP
- ▶ data replication/archiving

Materialized views

Apps

- ▶ query optimization
- ▶ data warehousing
 - ▶ integration of data
 - ▶ OLAP
- ▶ data replication/archiving
- ▶ data visualization

Materialized views

Apps

- ▶ query optimization
- ▶ data warehousing
 - ▶ integration of data
 - ▶ OLAP
- ▶ data replication/archiving
- ▶ data visualization
- ▶ caching in networked devices

Materialized views

Apps

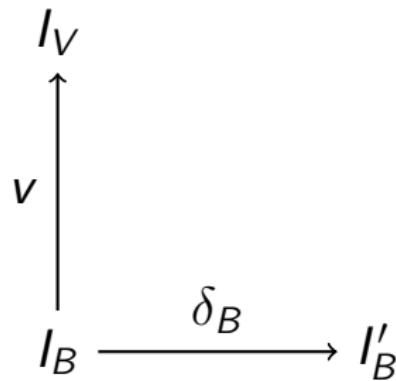
- ▶ query optimization
- ▶ data warehousing
 - ▶ integration of data
 - ▶ OLAP
- ▶ data replication/archiving
- ▶ data visualization
- ▶ caching in networked devices
- ▶

Maintaining materialized views

Materialized views

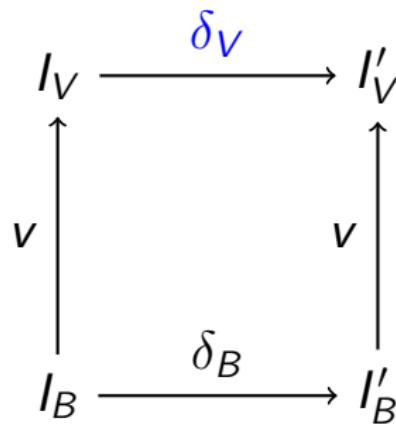
Issue: we must keep the view up-to-date, as base data evolves ...

View maintenance



Given update δ_B on base instance I_B , find appropriate update δ_V on materialized view I_V

View maintenance



Given update δ_B on base instance I_B , find appropriate update δ_V on materialized view I_V

Materialized views: maintenance

- ▶ incremental vs. complete
- ▶ immediate vs. deferred

Materialized views: maintenance

- ▶ manual code

Materialized views: maintenance

- ▶ manual code
- ▶ triggers on base relations

Materialized views: maintenance

- ▶ manual code
- ▶ triggers on base relations

```
CREATE TRIGGER mng_update
INSERT ON emp
BEGIN
    INSERT INTO mng_view VALUES
        (NEW.name, NEW.address, NEW.phone);
END;
```

Materialized views: maintenance

The Counting Algorithm for incremental
maintenance

Materialized views: maintenance

The Counting Algorithm for incremental maintenance

- ▶ keep track of the number of derivations of a tuple in the view
 - ▶ essentially, this is the number of duplicates of the tuple in bag-evaluation of the view query

Materialized views: maintenance

The Counting Algorithm for incremental maintenance

- ▶ keep track of the number of derivations of a tuple in the view
 - ▶ essentially, this is the number of duplicates of the tuple in bag-evaluation of the view query
- ▶ calculate update differential δ_V for the view
 - ▶ e.g., for view $V = R \bowtie S$, and update δ_R^+ on R , we have $\delta_V = \delta_R^+ \bowtie S$, and

$$V' = V \cup \delta_V$$

Materialized views: maintenance

The Counting Algorithm for incremental maintenance

- ▶ keep track of the number of derivations of a tuple in the view
 - ▶ essentially, this is the number of duplicates of the tuple in bag-evaluation of the view query
- ▶ calculate update differential δ_V for the view
 - ▶ e.g., for view $V = R \bowtie S$, and update δ_R^+ on R , we have $\delta_V = \delta_R^+ \bowtie S$, and

$$V' = V \cup \delta_V$$

- ▶ rules for other algebra operators given in our textbook

Materialized views: maintenance

The Counting Algorithm

- ▶ example: for
 $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we
have $hop = \{(a, c), (a, e)\}$

Materialized views: maintenance

The Counting Algorithm

- ▶ example: for
 $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we
have $hop = \{(a, c), (a, e)\}$
 - ▶ how many ways is $hop(a, c)$ derivable?

Materialized views: maintenance

The Counting Algorithm

- ▶ example: for
 $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we
have $hop = \{(a, c), (a, e)\}$
 - ▶ how many ways is $hop(a, c)$ derivable?
 - ▶ how many ways is $hop(a, e)$ derivable?

Materialized views: maintenance

The Counting Algorithm

- ▶ example: for
 $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we
have $hop = \{(a, c), (a, e)\}$
 - ▶ how many ways is $hop(a, c)$ derivable?
 - ▶ how many ways is $hop(a, e)$ derivable?
 - ▶ now, suppose $\delta_{edge}^- = \{(a, b)\}$.
 - ▶ then, $\delta_{hop}^- = \{(a, c), (a, e)\}$.

Materialized views: maintenance

The Counting Algorithm

- ▶ example: for
 $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we have $hop = \{(a, c), (a, e)\}$
 - ▶ how many ways is $hop(a, c)$ derivable?
 - ▶ how many ways is $hop(a, e)$ derivable?
 - ▶ now, suppose $\delta_{edge}^- = \{(a, b)\}$.
 - ▶ then, $\delta_{hop}^- = \{(a, c), (a, e)\}$.
- ▶ we know that after the update, $hop = \{(a, c)\}$.
- ▶ how to incrementally apply δ_{hop} to get the correct hop ?

Materialized views: maintenance

The Counting Algorithm

- ▶ example: for
 $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we have $hop = \{(a, c), (a, e)\}$
 - ▶ how many ways is $hop(a, c)$ derivable?
 - ▶ how many ways is $hop(a, e)$ derivable?
 - ▶ now, suppose $\delta_{edge}^- = \{(a, b)\}$.
 - ▶ then, $\delta_{hop}^- = \{(a, c), (a, e)\}$.
- ▶ we know that after the update, $hop = \{(a, c)\}$.
- ▶ how to incrementally apply δ_{hop} to get the correct hop ?
- ▶ check the counts of the elements of the view!

Materialized views: maintenance

The Counting Algorithm

- ▶ keep track of the number of derivations of a tuple in the view
- ▶ calculate update differential δ_V for the view

Materialized views: maintenance

The Counting Algorithm

- ▶ keep track of the number of derivations of a tuple in the view
- ▶ calculate update differential δ_V for the view
- ▶ for insertions (i.e., δ^+), increment counts; for deletions (i.e., δ^-), decrement counts
- ▶ if the count of a tuple goes to zero, remove it from the view

Materialized views: maintenance

The Counting Algorithm

- ▶ keep track of the number of derivations of a tuple in the view
- ▶ calculate update differential δ_V for the view
- ▶ for insertions (i.e., δ^+), increment counts; for deletions (i.e., δ^-), decrement counts
- ▶ if the count of a tuple goes to zero, remove it from the view

in our example $hop = \{(a, c) : 2, (a, e) : 1\}$, and
 $\delta_{hop}(hop) = \{(a, c) : 1\}$

Using views

Query containment & equivalence

When is a view useful for a given query?

Query containment & equivalence

When is a view useful for a given query?

- ▶ more generally, given two queries Q_1 and Q_2 , is it the case that $Q_1(I) \subseteq Q_2(I)$, for any database instance I ?

Query containment & equivalence

When is a view useful for a given query?

- ▶ more generally, given two queries Q_1 and Q_2 , is it the case that $Q_1(I) \subseteq Q_2(I)$, for any database instance I ?
- ▶ in this case we say Q_1 is **contained** in Q_2 , denoted $Q_1 \subseteq Q_2$

Query containment & equivalence

When is a view useful for a given query?

- ▶ more generally, given two queries Q_1 and Q_2 , is it the case that $Q_1(I) \subseteq Q_2(I)$, for any database instance I ?
- ▶ in this case we say Q_1 is **contained** in Q_2 , denoted $Q_1 \subseteq Q_2$
- ▶ if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$, then we say Q_1 and Q_2 are **equivalent**

Query containment & equivalence

- ▶ recall the *hop* view, which we now write as

$$\textit{hop}(x, y) \leftarrow \textit{edge}(x, z), \textit{edge}(z, y)$$

in familiar datalog notation

Query containment & equivalence

- ▶ recall the *hop* view, which we now write as

$$\textit{hop}(x, y) \leftarrow \textit{edge}(x, z), \textit{edge}(z, y)$$

in familiar datalog notation

- ▶ next, consider

$$\textit{hop}'(v, w) \leftarrow \textit{edge}(v, u), \textit{edge}(u, w), \textit{edge}(w, w)$$

Query containment & equivalence

- ▶ recall the *hop* view, which we now write as

$$\textit{hop}(x, y) \leftarrow \textit{edge}(x, z), \textit{edge}(z, y)$$

in familiar datalog notation

- ▶ next, consider

$$\textit{hop}'(v, w) \leftarrow \textit{edge}(v, u), \textit{edge}(u, w), \textit{edge}(w, w)$$

- ▶ is it the case that $\textit{hop} \subseteq \textit{hop}'$?
- ▶ is it the case that $\textit{hop}' \subseteq \textit{hop}$?

Query containment & equivalence

- ▶ recall the *hop* view, which we now write as

$$\textit{hop}(x, y) \leftarrow \textit{edge}(x, z), \textit{edge}(z, y)$$

in familiar datalog notation

- ▶ next, consider

$$\textit{hop}'(v, w) \leftarrow \textit{edge}(v, u), \textit{edge}(u, w), \textit{edge}(w, w)$$

- ▶ is it the case that $\textit{hop} \subseteq \textit{hop}'$?
- ▶ is it the case that $\textit{hop}' \subseteq \textit{hop}$?
- ▶ how can we prove this?
 - ▶ we will restrict our discussion now to conjunctive queries (containment is undecidable for FO ...)

Query containment & equivalence

Homomorphisms

- ▶ a mapping from the variables of Q_2 to the variables of Q_1 , such that
 - ▶ the head of Q_2 becomes the head of Q_1
 - ▶ each subgoal of Q_2 becomes some subgoal of Q_1

Query containment & equivalence

Homomorphisms

- ▶ a mapping from the variables of Q_2 to the variables of Q_1 , such that
 - ▶ the head of Q_2 becomes the head of Q_1
 - ▶ each subgoal of Q_2 becomes some subgoal of Q_1
- ▶ it isn't necessary for every subgoal of Q_1 to be mapped onto

Query containment & equivalence

for example, the homomorphism φ defined as

$$x \rightarrow v,$$

$$y \rightarrow w,$$

$$z \rightarrow u$$

Query containment & equivalence

for example, the homomorphism φ defined as

$$x \rightarrow v,$$

$$y \rightarrow w,$$

$$z \rightarrow u$$

maps

$$\text{hop}(x, y) \leftarrow \text{edge}(x, z), \text{edge}(z, y)$$

onto

$$\text{hop}'(v, w) \leftarrow \text{edge}(v, u), \text{edge}(u, w), \text{edge}(w, w)$$

Query containment & equivalence

Theorem

$Q_1 \subseteq Q_2$ if and only if there exists a homomorphism from Q_2 to Q_1

Query containment & equivalence

Theorem

$Q_1 \subseteq Q_2$ if and only if there exists a homomorphism from Q_2 to Q_1

proof: **(if)** Let $\varphi : Q_2 \rightarrow Q_1$ be a homomorphism, and let I be a database. Every tuple $t \in Q_1(I)$ is produced by some substitution ψ_t on the variables of Q_1 that make all of Q_1 's subgoals true in I . Then $\psi_t \circ \varphi$ is a substitution for variables of Q_2 such that $t \in Q_2(I)$. Hence, $Q_1 \subseteq Q_2$.

Query containment & equivalence

Theorem

$Q_1 \subseteq Q_2$ if and only if there exists a homomorphism from Q_2 to Q_1

proof, continued: **(only if)** Create a fresh unique atom for each variable of Q_1 , and let I_{Q_1} be the database instance consisting of all the subgoals of Q_1 , with the chosen atoms substituted for variables. Now, note that $Q_1(I_{Q_1})$ contains the “atom-head” t of Q_1 . Since $Q_1 \subseteq Q_2$, it must also be that $t \in Q_2(I_{Q_1})$.

Query containment & equivalence

Theorem

$Q_1 \subseteq Q_2$ if and only if there exists a homomorphism from Q_2 to Q_1

proof, continued: Let ψ_t be the substitution of constants from I_{Q_1} for the variables of Q_2 that makes each subgoal of Q_2 a tuple of the instance I_{Q_1} and yields t as the head; and, let φ be the substitution that maps constants of I_{Q_1} to their unique corresponding variable of Q_1 . Then $\varphi \circ \psi_t$ is a homomorphism from Q_2 to Q_1 . □

Query containment & equivalence

unfortunately, checking containment for conjunctive queries is an NP-complete problem

Query containment & equivalence

unfortunately, checking containment for conjunctive queries is an NP-complete problem

1. given a mapping m from Q_2 to Q_1 , we can check if m is indeed a homomorphism in polynomial time

Query containment & equivalence

unfortunately, checking containment for conjunctive queries is an NP-complete problem

1. given a mapping m from Q_2 to Q_1 , we can check if m is indeed a homomorphism in polynomial time
2. Let $G = (V, E)$ be a graph and k be an integer. Consider, for set C of k new distinct variables,

$$Q_2 = \text{out}() \leftarrow \bigwedge_{(u,v) \in E} E(u, v)$$

$$Q_1 = \text{out}() \leftarrow \bigwedge_{u, v \in C, u \neq v} E(u, v)$$

Then $Q_1 \subseteq Q_2$ iff G has a k -coloring.



Query containment & equivalence

Fortunately, queries are often quite small, especially with respect to the size of data

Furthermore, checking query containment for acyclic conjunctive queries is *tractable* (i.e., computable in polynomial time). More on this in a later lecture ...

Exercise: answering queries with views

Consider the following conjunctive query.

$$Q : \text{result}(A) \leftarrow r(A, B), r(A, C), s(B, D, E), s(B, F, F)$$

Minimize Q . In other words, give a query Q' that (i) has the smallest possible body and (ii) is equivalent to Q . Demonstrate that your query satisfies both of these properties.

Wrap Up

Wrap up

- ▶ Cost estimation, towards choosing a good physical plan
- ▶ the construction, maintenance, and use of views

Wrap up

- ▶ Cost estimation, towards choosing a good physical plan
- ▶ the construction, maintenance, and use of views
- ▶ **Next time**
 - ▶ putting everything together for query optimization

Wrap up

- ▶ Cost estimation, towards choosing a good physical plan
- ▶ the construction, maintenance, and use of views
- ▶ **Next time**
 - ▶ putting everything together for query optimization
- ▶ **Reminder:** team project report due by Wednesday 13 May

Credits

Ullman 1999

Query optimization

Lecture 7
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

13 May 2015

Admin

- ▶ Project part 2 due today
 - ▶ Part 3 has been posted

Admin

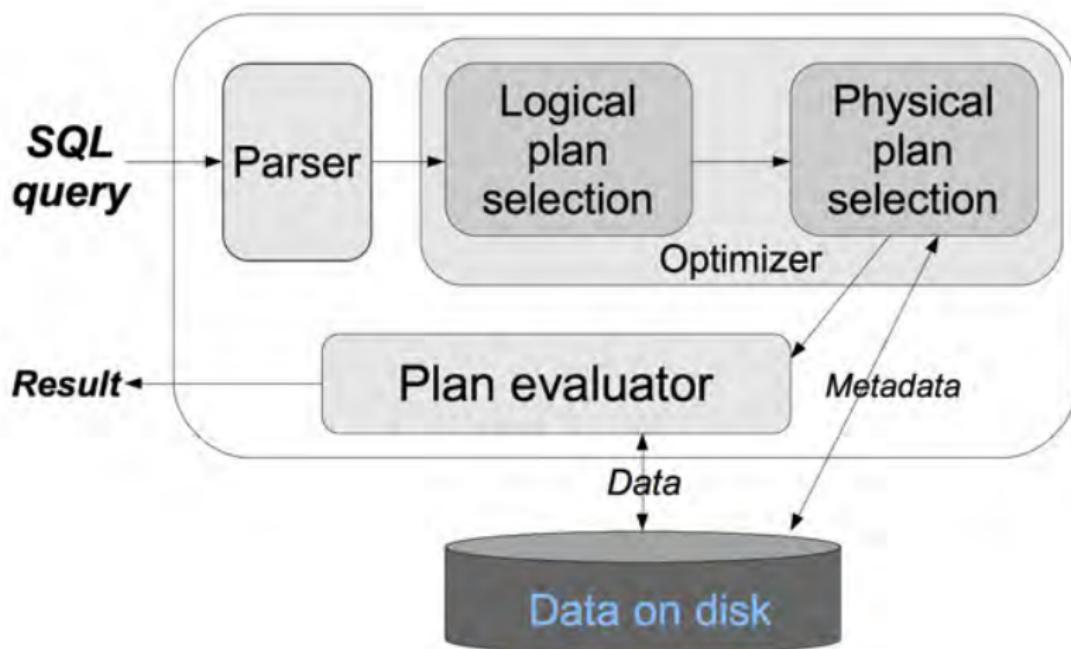
- ▶ Project part 2 due today
 - ▶ Part 3 has been posted
- ▶ No lecture this Friday or next Wednesday
- ▶ Written individual assignment will be posted this week

Last time

Size estimation (heuristics, histograms)

View creation, maintenance, and use

The life of a query

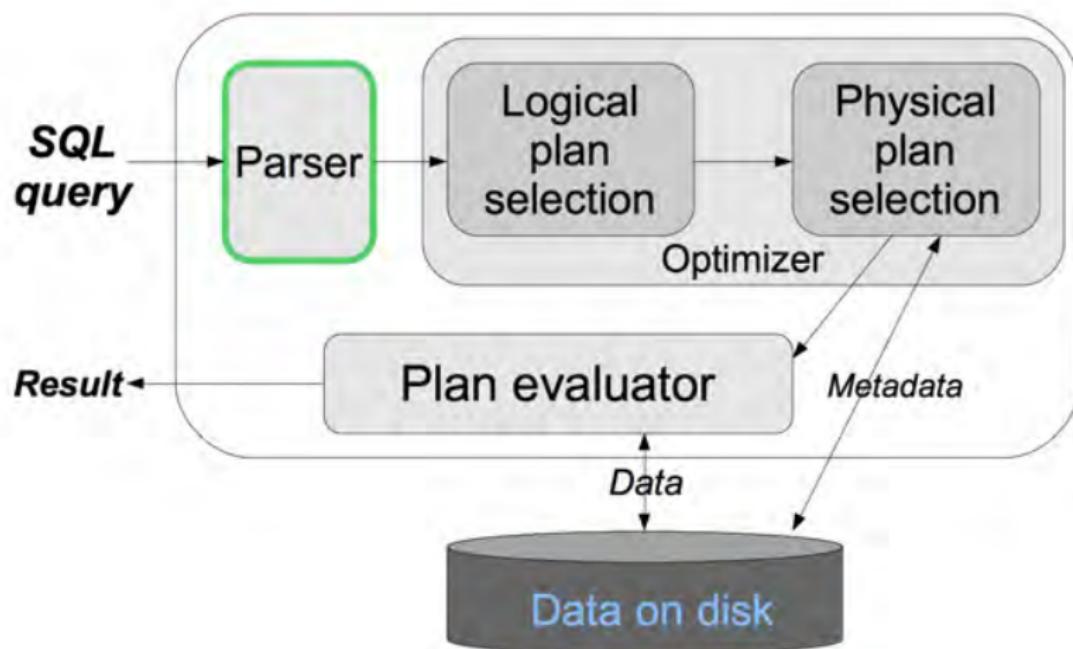


Today

Query compilation

- ▶ parsing
- ▶ logical optimization
- ▶ physical optimization

The life of a query: parsing



The life of a query: parsing

- ▶ Employee(EID, Name, ECity)
- ▶ Company(CID, Name, CCity)
- ▶ WorksFor(EID, CID, Salary)

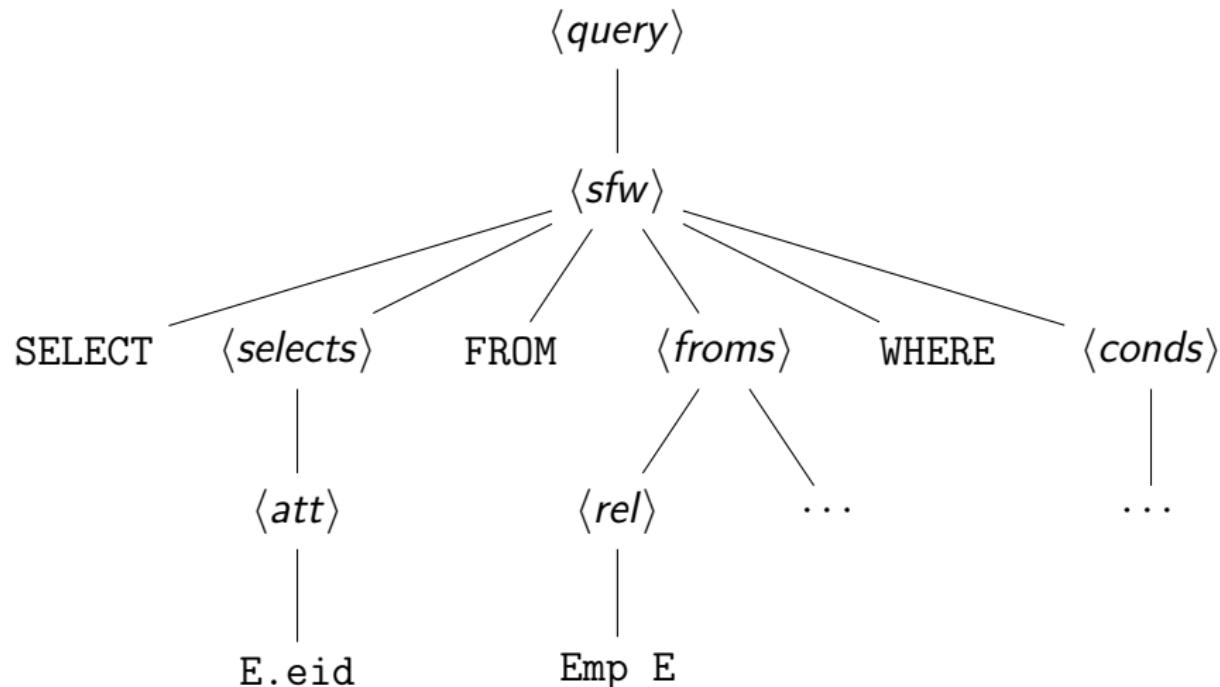
What are the ID's of employees living and working in Best with above-average salaries?

The life of a query: parsing

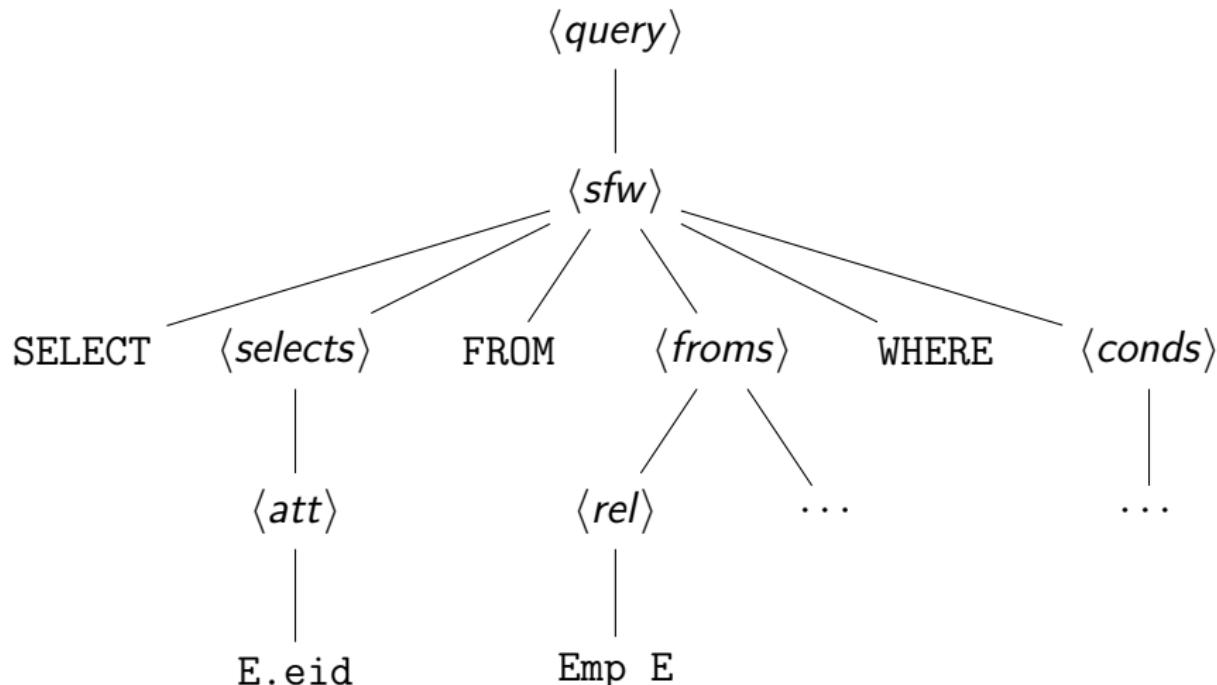
- ▶ Employee(EID, Name, ECity)
- ▶ Company(CID, Name, CCity)
- ▶ WorksFor(EID, CID, Salary)

```
SELECT E.EID
FROM Employee E, WorksFor W, Company C
WHERE E.EID = W.EID AND W.CID = C.CID
      AND E.ECity = 'Best'
      AND C.CCity = 'Best'
      AND W.Salary > (SELECT AVG(SALARY)
                        FROM WorksFor)
```

The life of a query: parsing



The life of a query: parsing



... and preprocess for semantics (e.g., typing)

The life of a query: parsing

Parser generates a collection of query *blocks*

- ▶ block = a S-F-W query with no nesting
 - ▶ essentially, a conjunctive query

The life of a query: parsing

Parser generates a collection of query *blocks*

- ▶ block = a S-F-W query with no nesting
 - ▶ essentially, a conjunctive query
- ▶ typically focus on optimizing one block at a time

The life of a query: parsing

- ▶ Employee(EID, Name, ECity)
- ▶ Company(CID, Name, CCity)
- ▶ WorksFor(EID, CID, Salary)

```
SELECT E.EID
FROM Employee E, WorksFor W, Company C
WHERE E.EID = W.EID AND W.CID = C.CID
      AND E.ECity = 'Best'
      AND C.CCity = 'Best'
      AND W.Salary >
          (SELECT AVG(SALARY)
           FROM WorksFor)
```

nested block

The life of a query: parsing

- ▶ Employee(EID, Name, ECity)
- ▶ Company(CID, Name, CCity)
- ▶ WorksFor(EID, CID, Salary)

```
SELECT E.EID
FROM Employee E, WorksFor W, Company C
WHERE E.EID = W.EID AND W.CID = C.CID
      AND E.ECity = 'Best'
      AND C.CCity = 'Best'
      AND W.Salary >
            (SELECT AVG(SALARY)
             FROM WorksFor)
```

outer block

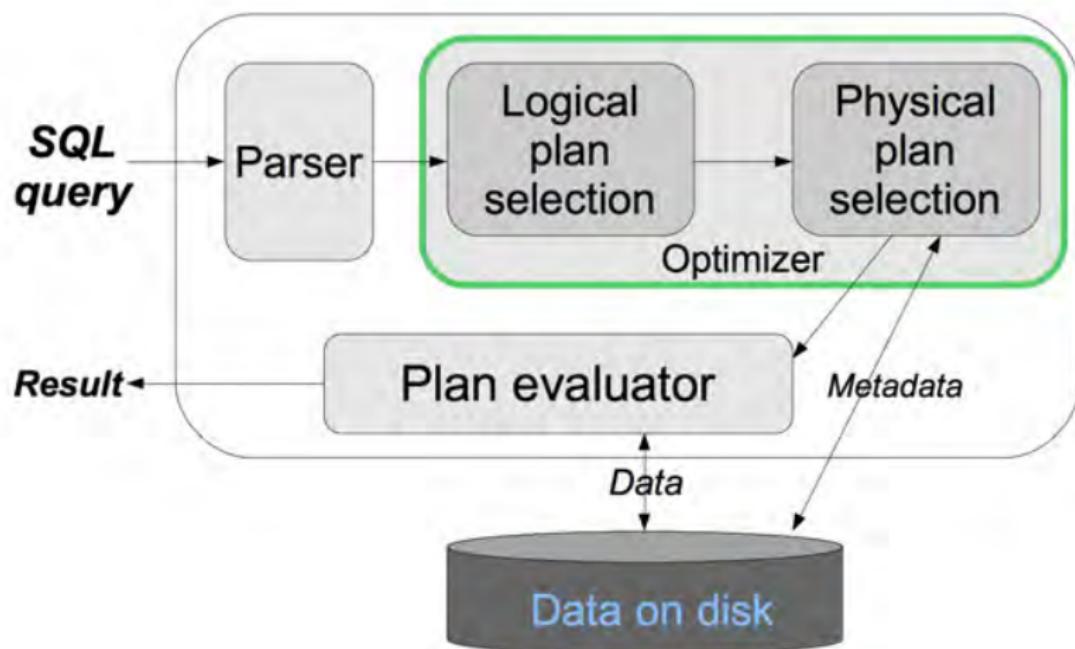
The life of a query: parsing

- ▶ Employee(EID, Name, ECity)
- ▶ Company(CID, Name, CCity)
- ▶ WorksFor(EID, CID, Salary)

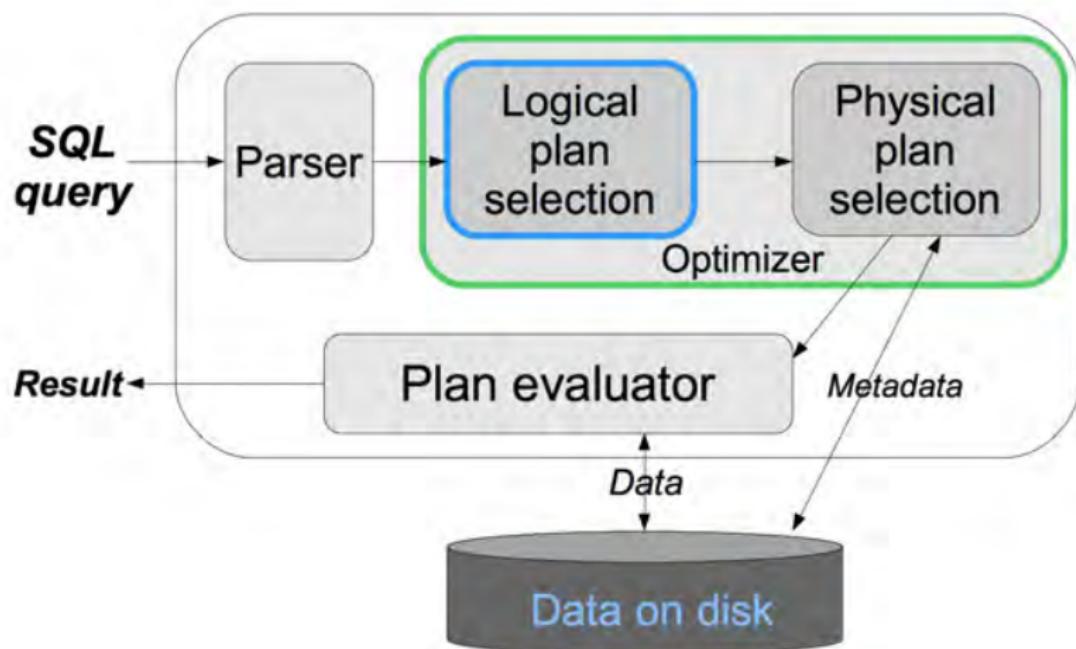
```
SELECT E.EID  
FROM Employee E, WorksFor W, Company C  
WHERE E.EID = W.EID AND W.CID = C.CID  
      AND E.ECity = 'Best'  
      AND C.CCity = 'Best'  
      AND W.Salary >  
          (SELECT AVG(SALARY)  
           FROM WorksFor)
```

focus on optimizing each block separately

The life of a query: optimization



The life of a query: logical optimization



The life of a query: logical optimization

Goal: map a query-block to a “preferred” logical query plan

The life of a query: logical optimization

Goal: map a query-block to a “preferred” logical query plan

Step A: map the block’s parse tree to equivalent RA expression, working bottom-up

1. map FROM list to cartesian product of all relations

The life of a query: logical optimization

Goal: map a query-block to a “preferred” logical query plan

Step A: map the block’s parse tree to equivalent RA expression, working bottom-up

1. map FROM list to cartesian product of all relations
2. form single σ for WHERE conditions

The life of a query: logical optimization

Goal: map a query-block to a “preferred” logical query plan

Step A: map the block’s parse tree to equivalent RA expression, working bottom-up

1. map FROM list to cartesian product of all relations
2. form single σ for WHERE conditions
3. form π list from the SELECT clause

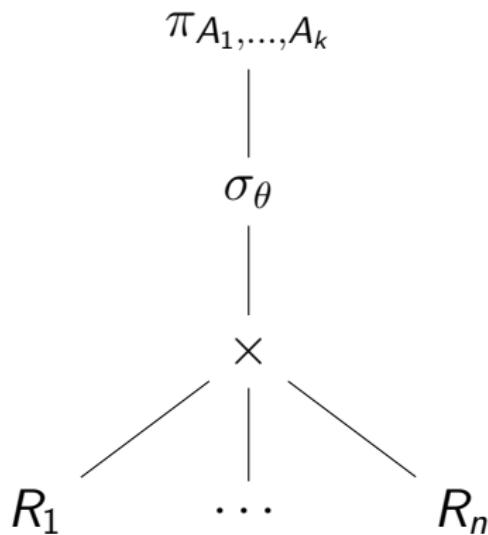
The life of a query: logical optimization

Goal: map a query-block to a “preferred” logical query plan

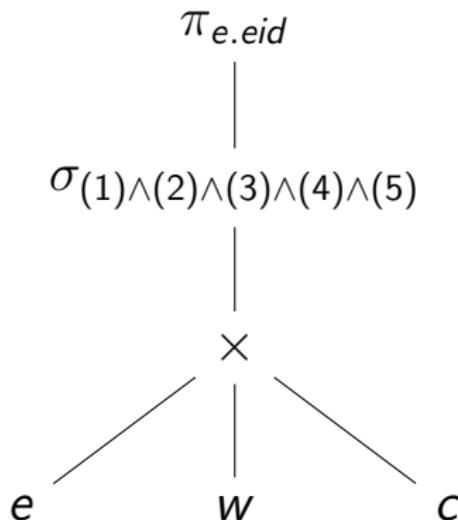
Step A: map the block’s parse tree to equivalent RA expression, working bottom-up

1. map FROM list to cartesian product of all relations
 2. form single σ for WHERE conditions
 3. form π list from the SELECT clause
- any aggregation can be applied after this $\pi \cdot \sigma \cdot \times$

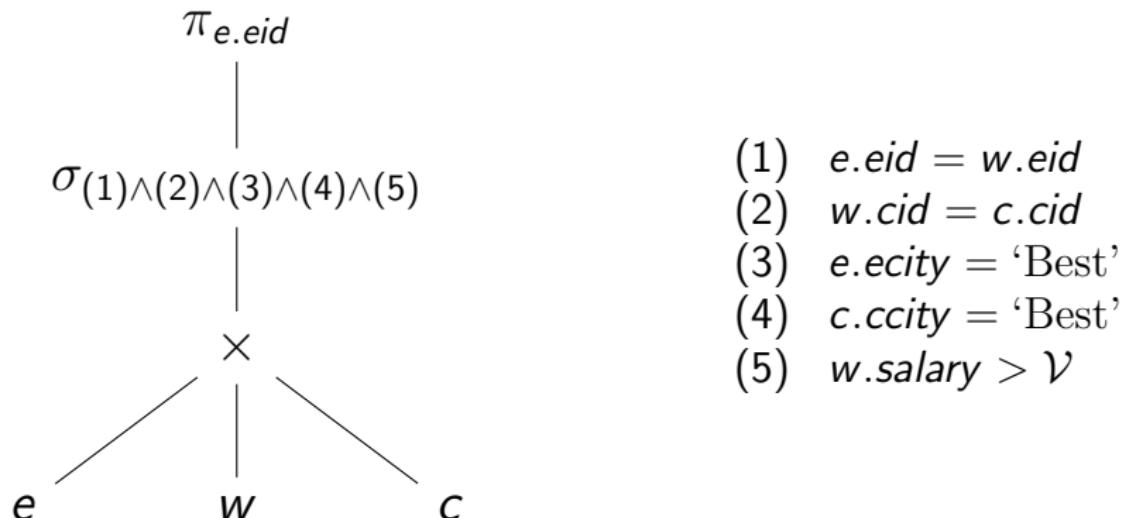
The life of a query: logical optimization



The life of a query: logical optimization



The life of a query: logical optimization



The life of a query: logical optimization

Step B: improve this initial logical query plan (i.e., generate a preferred logical query plan)

The life of a query: logical optimization

Step B: improve this initial logical query plan (i.e., generate a preferred logical query plan)

based on equivalence rules, used to identify different ways of formulating the query

The life of a query: logical optimization

Step B: improve this initial logical query plan (i.e., generate a preferred logical query plan)

based on equivalence rules, used to identify different ways of formulating the query

this defines a search space of alternative plans, to be considered by the optimizer

RA equivalence rules: commutativity & associativity

$$R \star S = S \star R$$

$$(R \star S) \star T = R \star (S \star T)$$

for $\star \in \{\times, \bowtie, \cup, \cap\}$

RA equivalence rules: commutativity & associativity

So, we have for example

$$R \cap S = S \cap R$$

$$(R \cap S) \cap T = R \cap (S \cap T)$$

RA equivalence rules: commutativity & associativity

So, we have for example

$$R \cap S = S \cap R$$

$$(R \cap S) \cap T = R \cap (S \cap T)$$

example application.

$$\begin{aligned}\sigma_\theta(\pi_A(R)) \cap (\pi_A(S) \cap \pi_A(T)) \\ = (\sigma_\theta(\pi_A(R)) \cap \pi_A(S)) \cap \pi_A(T)\end{aligned}$$

RA equivalence rules: selection laws

splitting

$$\sigma_{C_1 \wedge \dots \wedge C_n}(R) = \sigma_{C_1}(\dots \sigma_{C_n}(R) \dots)$$

RA equivalence rules: selection laws

splitting

$$\sigma_{C_1 \wedge \dots \wedge C_n}(R) = \sigma_{C_1}(\dots \sigma_{C_n}(R) \dots)$$

$$\sigma_{C_1 \vee C_2}(R) = \sigma_{C_1}(R) \cup \sigma_{C_2}(R)$$

RA equivalence rules: selection laws

splitting

$$\begin{aligned}\sigma_{C_1 \wedge \dots \wedge C_n}(R) &= \sigma_{C_1}(\dots \sigma_{C_n}(R) \dots) \\ \sigma_{C_1 \vee C_2}(R) &= \sigma_{C_1}(R) \cup \sigma_{C_2}(R)\end{aligned}$$

example application.

$$\sigma_{a=1}(\sigma_{b < c}(R)) \cup \sigma_{a=3}(\sigma_{b < c}(R))$$

RA equivalence rules: selection laws

splitting

$$\begin{aligned}\sigma_{C_1 \wedge \dots \wedge C_n}(R) &= \sigma_{C_1}(\dots \sigma_{C_n}(R) \dots) \\ \sigma_{C_1 \vee C_2}(R) &= \sigma_{C_1}(R) \cup \sigma_{C_2}(R)\end{aligned}$$

example application.

$$\sigma_{a=1}(\sigma_{b < c}(R)) \cup \sigma_{a=3}(\sigma_{b < c}(R)) = \sigma_{a=1 \vee a=3}(\sigma_{b < c}(R))$$

RA equivalence rules: selection laws

splitting

$$\begin{aligned}\sigma_{C_1 \wedge \dots \wedge C_n}(R) &= \sigma_{C_1}(\dots \sigma_{C_n}(R) \dots) \\ \sigma_{C_1 \vee C_2}(R) &= \sigma_{C_1}(R) \cup \sigma_{C_2}(R)\end{aligned}$$

example application.

$$\begin{aligned}\sigma_{a=1}(\sigma_{b < c}(R)) \cup \sigma_{a=3}(\sigma_{b < c}(R)) &= \sigma_{a=1 \vee a=3}(\sigma_{b < c}(R)) \\ &= \sigma_{(a=1 \vee a=3) \wedge b < c}(R)\end{aligned}$$

RA equivalence rules: selection laws

union

$$\sigma_\theta(R \cup S) = \sigma_\theta(R) \cup \sigma_\theta(S)$$

RA equivalence rules: selection laws

union

$$\sigma_\theta(R \cup S) = \sigma_\theta(R) \cup \sigma_\theta(S)$$

difference

$$\sigma_\theta(R - S) = \sigma_\theta(R) - \sigma_\theta(S)$$

RA equivalence rules: selection laws

union

$$\sigma_\theta(R \cup S) = \sigma_\theta(R) \cup \sigma_\theta(S)$$

difference

$$\begin{aligned}\sigma_\theta(R - S) &= \sigma_\theta(R) - \sigma_\theta(S) \\ &= \sigma_\theta(R) - S\end{aligned}$$

RA equivalence rules: selection laws

union

$$\sigma_\theta(R \cup S) = \sigma_\theta(R) \cup \sigma_\theta(S)$$

difference

$$\begin{aligned}\sigma_\theta(R - S) &= \sigma_\theta(R) - \sigma_\theta(S) \\ &= \sigma_\theta(R) - S\end{aligned}$$

and similarly for \times, \bowtie, \cap

RA equivalence rules: selection laws

union

$$\sigma_\theta(R \cup S) = \sigma_\theta(R) \cup \sigma_\theta(S)$$

difference

$$\begin{aligned}\sigma_\theta(R - S) &= \sigma_\theta(R) - \sigma_\theta(S) \\ &= \sigma_\theta(R) - S\end{aligned}$$

and similarly for \times, \bowtie, \cap

proof of the difference rule

RA equivalence rules: selection laws

example application. for $R(a, b)$ and $S(b, c)$

$$\sigma_{(a=1 \vee a=3) \wedge b < c}(R \bowtie S)$$

RA equivalence rules: selection laws

example application. for $R(a, b)$ and $S(b, c)$

$$\sigma_{(a=1 \vee a=3) \wedge b < c}(R \bowtie S) = \sigma_{a=1 \vee a=3}(\sigma_{b < c}(R \bowtie S))$$

RA equivalence rules: selection laws

example application. for $R(a, b)$ and $S(b, c)$

$$\begin{aligned}\sigma_{(a=1 \vee a=3) \wedge b < c}(R \bowtie S) &= \sigma_{a=1 \vee a=3}(\sigma_{b < c}(R \bowtie S)) \\ &= \sigma_{a=1 \vee a=3}(R \bowtie \sigma_{b < c}(S))\end{aligned}$$

RA equivalence rules: selection laws

example application. for $R(a, b)$ and $S(b, c)$

$$\begin{aligned}\sigma_{(a=1 \vee a=3) \wedge b < c}(R \bowtie S) &= \sigma_{a=1 \vee a=3}(\sigma_{b < c}(R \bowtie S)) \\ &= \sigma_{a=1 \vee a=3}(R \bowtie \sigma_{b < c}(S)) \\ &= \sigma_{a=1 \vee a=3}(R) \bowtie \sigma_{b < c}(S).\end{aligned}$$

RA equivalence rules: selection laws

example application. for $R(a, b)$ and $S(b, c)$

$$\begin{aligned}\sigma_{(a=1 \vee a=3) \wedge b < c}(R \bowtie S) &= \sigma_{a=1 \vee a=3}(\sigma_{b < c}(R \bowtie S)) \\ &= \sigma_{a=1 \vee a=3}(R \bowtie \sigma_{b < c}(S)) \\ &= \sigma_{a=1 \vee a=3}(R) \bowtie \sigma_{b < c}(S).\end{aligned}$$

note how this brings the selection conditions closer to the input relations ...

RA equivalence rules: projection law

if, for each $1 \leq i \leq n$, we have $a_i \subseteq a_{i+1}$, for subsets a_1, \dots, a_n of $\text{atts}(R)$, then

$$\pi_{a_1}(R) = \pi_{a_1}(\pi_{a_2}(\cdots(\pi_{a_n}(R))\cdots)).$$

RA equivalence rules: projection law

if, for each $1 \leq i \leq n$, we have $a_i \subseteq a_{i+1}$, for subsets a_1, \dots, a_n of $\text{atts}(R)$, then

$$\pi_{a_1}(R) = \pi_{a_1}(\pi_{a_2}(\cdots(\pi_{a_n}(R))\cdots)).$$

example application. for $R(a, b, c)$

$$\pi_a(R) = \pi_a(\pi_{a,b}(R)).$$

RA equivalence rules

Prove or disprove: $\pi_a(R - S) = \pi_a(R) - \pi_a(S)$,
where a is a nonempty set of attributes in R and S .

RA equivalence rules

Prove or disprove: $\pi_a(R - S) = \pi_a(R) - \pi_a(S)$,
where a is a nonempty set of attributes in R and S .

counterexample. Consider $R(a, b)$ and $S(a, b)$, with
respective instances $r = \{(1, 2)\}$ and $s = \{(1, 3)\}$.
Then $\pi_a(r - s) = \{(1)\}$, but $\pi_a(r) - \pi_a(s) = \{\}$.

RA equivalence rules

Prove or disprove: $\pi_x(R \bowtie S) = \pi_x(\pi_{xy}(R) \bowtie S)$,
where $y = \text{atts}(R) \cap \text{atts}(S)$ and $x \subseteq \text{atts}(R)$.

RA equivalence rules

Prove or disprove: $\pi_x(R \bowtie S) = \pi_x(\pi_{xy}(R) \bowtie S)$,
where $y = \text{atts}(R) \cap \text{atts}(S)$ and $x \subseteq \text{atts}(R)$.

proof. We show that $\pi_x(R \bowtie S) \subseteq \pi_x(\pi_{xy}(R) \bowtie S)$. The other direction is similar.

Suppose that $t \in \pi_x(R \bowtie S)$. Then we have that (1) there exists $t' \in R \bowtie S$ such that $t = t'[x]$ (i.e., t' projected on x).

It then follows that there exists $r \in R$ and $s \in S$ such that (2) $t'[\text{atts}(R)] = r$, (3) $t'[\text{atts}(S)] = s$, and (4) $r[y] = s[y]$.

From (2) we have that (5) $t'[xy] \in \pi_{xy}(R)$.

From (2), (4), and (5) we have that (6) $t'[xy] \bowtie s \in \pi_{xy}(R) \bowtie S$.

From (1) and (6), we have that $t = t'[x] \in \pi_x(\pi_{xy}(R) \bowtie S)$, as desired.

RA equivalence rules: SPJ laws

1. if $atts(\theta) \subseteq \{a_1, \dots, a_n\}$, then

$$\pi_{\{a_1, \dots, a_n\}}(\sigma_\theta(R)) = \sigma_\theta(\pi_{\{a_1, \dots, a_n\}}(R))$$

RA equivalence rules: SPJ laws

1. if $atts(\theta) \subseteq \{a_1, \dots, a_n\}$, then

$$\pi_{\{a_1, \dots, a_n\}}(\sigma_\theta(R)) = \sigma_\theta(\pi_{\{a_1, \dots, a_n\}}(R))$$

2. $R \bowtie_\theta S = \sigma_\theta(R \times S)$

RA equivalence rules: SPJ laws

3. if $a_1 \cup a_2 = a$, $a_1 \subseteq atts(R)$, and $a_2 \subseteq atts(S)$,
then

$$\pi_a(R \times S) = \pi_{a_1}(R) \times \pi_{a_2}(S)$$

RA equivalence rules: SPJ laws

3. if $a_1 \cup a_2 = a$, $a_1 \subseteq atts(R)$, and $a_2 \subseteq atts(S)$, then

$$\pi_a(R \times S) = \pi_{a_1}(R) \times \pi_{a_2}(S)$$

4. if $a_1 \cup a_2 = a$, $a_1 \subseteq atts(R)$, $a_2 \subseteq atts(S)$, and $atts(\theta) \subseteq a$, then

$$\pi_a(R \bowtie_\theta S) = \pi_{a_1}(R) \bowtie_\theta \pi_{a_2}(S)$$

RA equivalence rules: SPJ laws

3. if $a_1 \cup a_2 = a$, $a_1 \subseteq atts(R)$, and $a_2 \subseteq atts(S)$, then

$$\pi_a(R \times S) = \pi_{a_1}(R) \times \pi_{a_2}(S)$$

4. if $a_1 \cup a_2 = a$, $a_1 \subseteq atts(R)$, $a_2 \subseteq atts(S)$, and $atts(\theta) \subseteq a$, then

$$\pi_a(R \bowtie_\theta S) = \pi_{a_1}(R) \bowtie_\theta \pi_{a_2}(S)$$

exercise. prove 4

RA equivalence rules

Some heuristics

1. push down selections (σ) as far as they can go, splitting conjunctions in conditions (\wedge)

RA equivalence rules

Some heuristics

1. push down selections (σ) as far as they can go, splitting conjunctions in conditions (\wedge)
 - ▶ single most important strategy

RA equivalence rules

Some heuristics

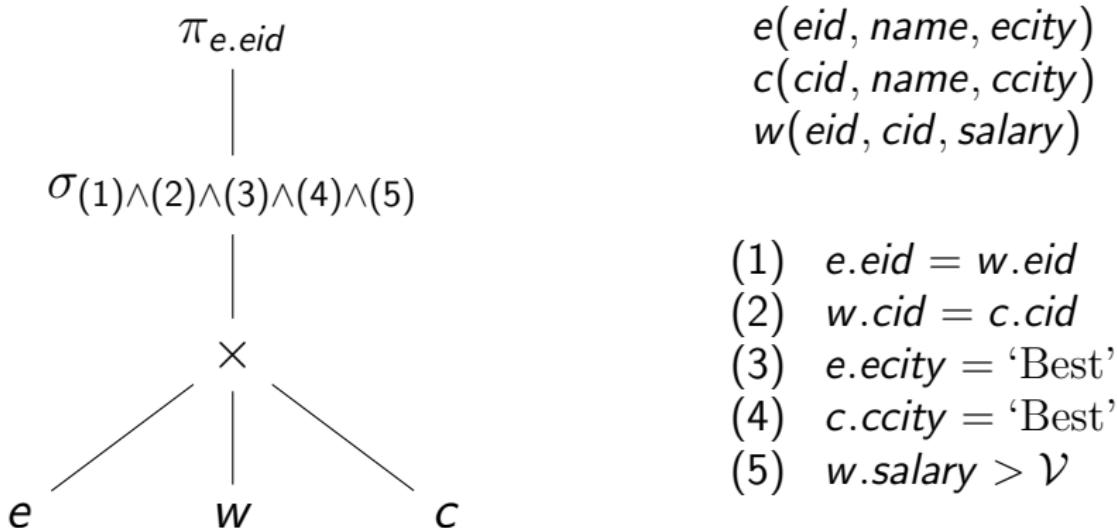
1. push down selections (σ) as far as they can go,
splitting conjunctions in conditions (\wedge)
 - ▶ single most important strategy
2. push down projections (π)

RA equivalence rules

Some heuristics

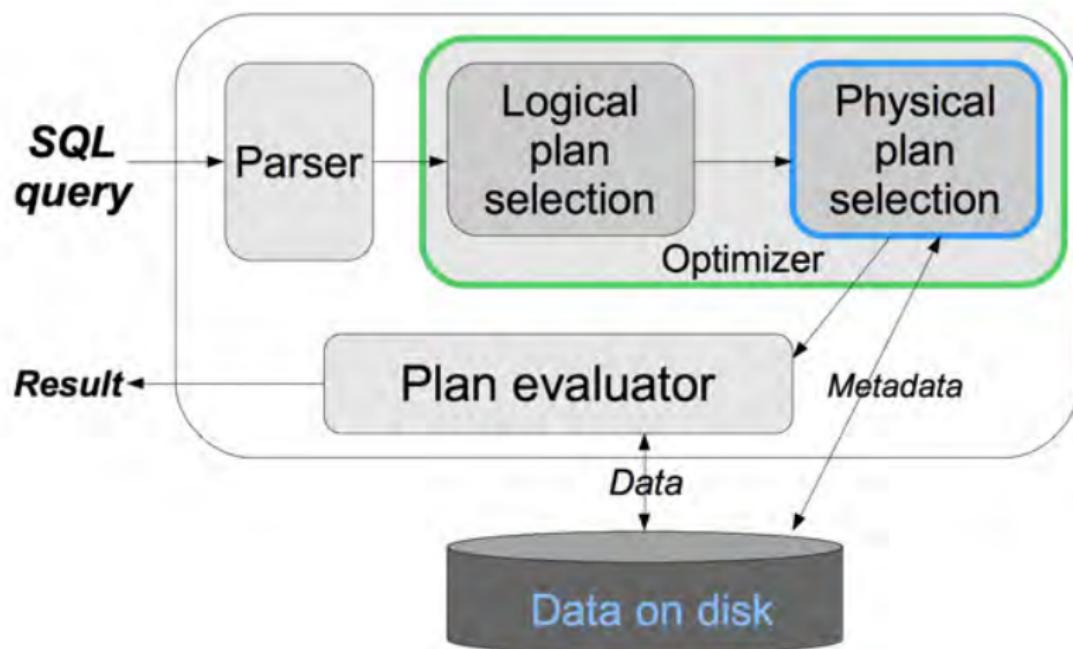
1. push down selections (σ) as far as they can go, splitting conjunctions in conditions (\wedge)
 - ▶ single most important strategy
2. push down projections (π)
3. if possible, turn $\sigma + \times$ into \bowtie , which is generally much cheaper to evaluate than σ and \times evaluated separately

RA equivalence rules



let's logically optimize our query, under these heuristics

The life of a query: physical optimization



The life of a query: physical optimization

Decisions

- ▶ two crucial decisions the optimizer makes
 - ▶ the order in which the physical operators are applied on the input relations
 - ▶ the choice of physical algorithms for each node of the plan
- ▶ of course, these are not independent

The life of a query: physical optimization

cost-based query optimization

- ▶ enumerate alternative plans, estimate the cost of each plan, pick the plan with minimum cost
- ▶ access methods and join algorithms define a search space
 - ▶ this space can be huge!
 - ▶ plan enumeration is the exploration of this search space

The life of a query: physical optimization

For example, suppose we have five relations, only one access method, only one join algorithm, and we only consider “linear” joins

The life of a query: physical optimization

For example, suppose we have five relations, only one access method, only one join algorithm, and we only consider “linear” joins

- ▶ there are then $5! = 120$ possible plans

The life of a query: physical optimization

For example, suppose we have five relations, only one access method, only one join algorithm, and we only consider “linear” joins

- ▶ there are then $5! = 120$ possible plans
- ▶ if we add one additional access method, we then have $2^5 \cdot 5! = 3840$ possible plans

The life of a query: physical optimization

For example, suppose we have five relations, only one access method, only one join algorithm, and we only consider “linear” joins

- ▶ there are then $5! = 120$ possible plans
- ▶ if we add one additional access method, we then have $2^5 \cdot 5! = 3840$ possible plans
- ▶ if we add one additional join algorithm, we then have $2^4 \cdot 2^5 \cdot 5! = 61440$ possible plans

The life of a query: physical optimization

For example, suppose we have five relations, only one access method, only one join algorithm, and we only consider “linear” joins

- ▶ there are then $5! = 120$ possible plans
- ▶ if we add one additional access method, we then have $2^5 \cdot 5! = 3840$ possible plans
- ▶ if we add one additional join algorithm, we then have $2^4 \cdot 2^5 \cdot 5! = 61440$ possible plans

In general, with n relations, there are $\frac{(2(n-1))!}{(n-1)!}$ different join orders. For $n = 5$, this gives us 860,160 possible plans with two access methods and join algorithms. For $n = 10$, this gives us roughly 17.6 billion plans ...

The life of a query: physical optimization

cost-based query optimization

- ▶ exhaustive search is certainly out of the question
- ▶ it could be that exploring the search space might take longer than actually evaluating the query

The life of a query: physical optimization

cost-based query optimization

- ▶ exhaustive search is certainly out of the question
- ▶ it could be that exploring the search space might take longer than actually evaluating the query
- ▶ the manner in which the plan space is explored describes a (physical) query optimization method
 - ▶ dynamic programming, rule-based optimization, randomized search, ...

The life of a query: physical optimization

cost-based query optimization

- ▶ costing plans and exploring a plan space is nontrivial
- ▶ now, consider that the DBMS must do this for 1000's of queries, simultaneously!
- ▶ hence, all of this must be done quickly, without looking back

The life of a query: physical optimization

cost-based query optimization

- ▶ costing plans and exploring a plan space is nontrivial
- ▶ now, consider that the DBMS must do this for 1000's of queries, simultaneously!
- ▶ hence, all of this must be done quickly, without looking back
- ▶ query optimization is very much still an active area of research
- ▶ indeed, rarely will an optimizer find the optimal plan
 - ▶ it must, however, not pick a bad plan – just an OK one

Query optimization: dynamic-programming algorithm

```
procedure FindBestPlan(S)
    if (bestplan[S].cost ≠ ∞) /* bestplan[S] already computed */
        return bestplan[S]
    if (S contains only 1 relation)
        set bestplan[S].plan and bestplan[S].cost based on best way of accessing S
    else for each non-empty subset S1 of S such that S1 ≠ S
        P1 = FindBestPlan(S1)
        P2 = FindBestPlan(S – S1)
        A = best algorithm for joining results of P1 and P2
        cost = P1.cost + P2.cost + cost of A
        if cost < bestplan[S].cost
            bestplan[S].cost = cost
            bestplan[S].plan = “execute P1.plan; execute P2.plan;
                                join results of P1 and P2 using A”
    return bestplan[S]
```

Query optimization: dynamic-programming algorithm

```
procedure FindBestPlan(S)
    if (bestplan[S].cost ≠ ∞) /* bestplan[S] already computed */
        return bestplan[S]
    if (S contains only 1 relation)
        set bestplan[S].plan and bestplan[S].cost based on best way of accessing S
    else for each non-empty subset S1 of S such that S1 ≠ S
        P1 = FindBestPlan(S1)
        P2 = FindBestPlan(S – S1)
        A = best algorithm for joining results of P1 and P2
        cost = P1.cost + P2.cost + cost of A
        if cost < bestplan[S].cost
            bestplan[S].cost = cost
            bestplan[S].plan = “execute P1.plan; execute P2.plan;
                                join results of P1 and P2 using A”
    return bestplan[S]
```

Running time is $\mathcal{O}(3^n)$, which gives us roughly 59,000 plans for $n = 10$ (compare this with 17.6 billion plans)

Query optimization: dynamic-programming algorithm

```
procedure FindBestPlan(S)
    if (bestplan[S].cost ≠ ∞) /* bestplan[S] already computed */
        return bestplan[S]
    if (S contains only 1 relation)
        set bestplan[S].plan and bestplan[S].cost based on best way of accessing S
    else for each non-empty subset S1 of S such that S1 ≠ S
        P1 = FindBestPlan(S1)
        P2 = FindBestPlan(S – S1)
        A = best algorithm for joining results of P1 and P2
        cost = P1.cost + P2.cost + cost of A
        if cost < bestplan[S].cost
            bestplan[S].cost = cost
            bestplan[S].plan = “execute P1.plan; execute P2.plan;
                                join results of P1 and P2 using A”
    return bestplan[S]
```

- ▶ most widely used strategy
- ▶ works well for queries with fewer than 10 to 15 joins

Query optimization in System R

- ▶ follows the classic dynamic programming approach

Query optimization in System R

- ▶ follows the classic dynamic programming approach
- ▶ heuristics: use the equivalence rules to push down selections and projections, and delay cartesian products

Query optimization in System R

- ▶ follows the classic dynamic programming approach
- ▶ heuristics: use the equivalence rules to push down selections and projections, and delay cartesian products
- ▶ constraints: left-deep plans, nested-loops and sort-merge join only

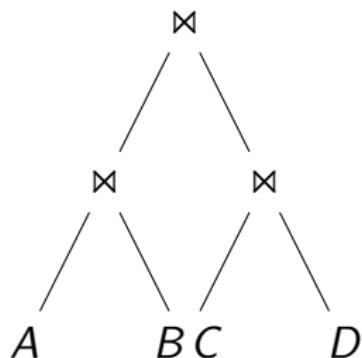
Query optimization in System R

- ▶ follows the classic dynamic programming approach
- ▶ heuristics: use the equivalence rules to push down selections and projections, and delay cartesian products
- ▶ constraints: left-deep plans, nested-loops and sort-merge join only
 - ▶ left-deep plans facilitate pipelining of output of each operator into the next operator, without materializing intermediate result

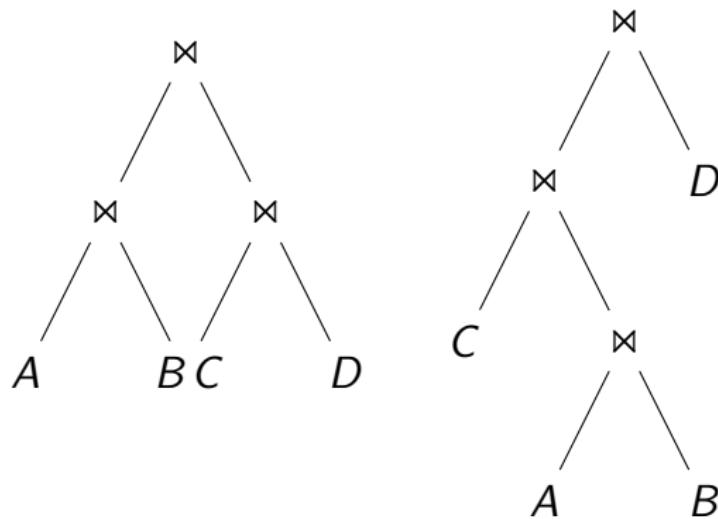
Query optimization in System R

$A \bowtie B \bowtie C \bowtie D$

Query optimization in System R

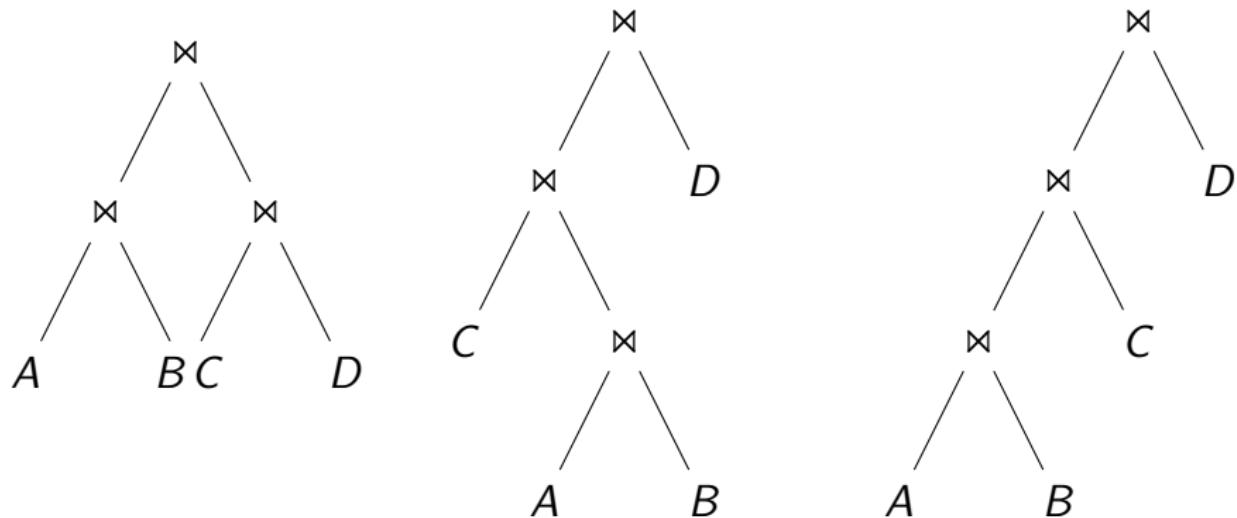
$$A \bowtie B \bowtie C \bowtie D$$


Query optimization in System R

$$A \bowtie B \bowtie C \bowtie D$$


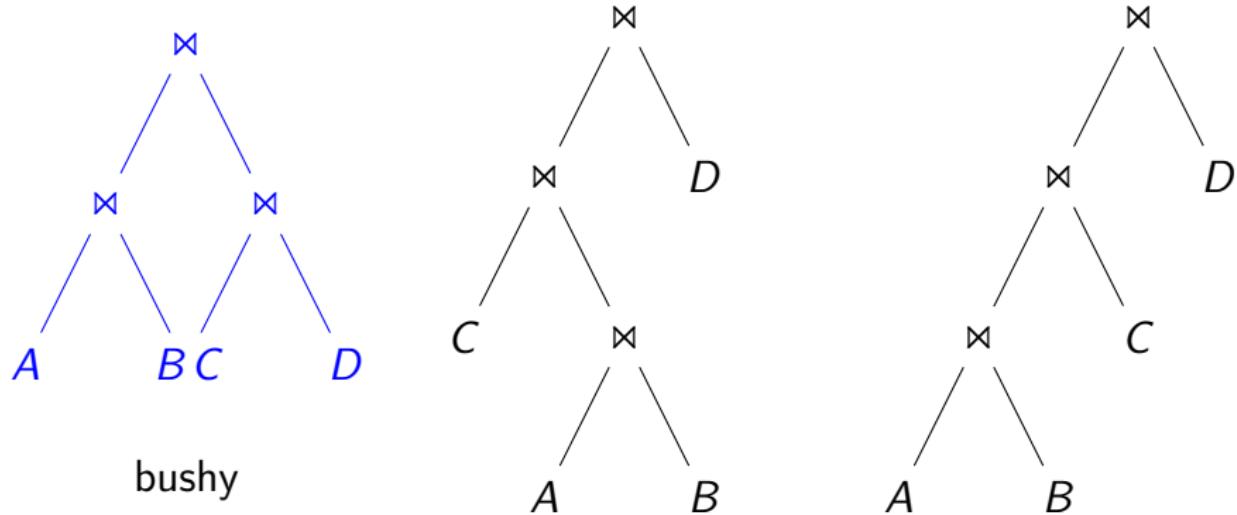
Query optimization in System R

$$A \bowtie B \bowtie C \bowtie D$$



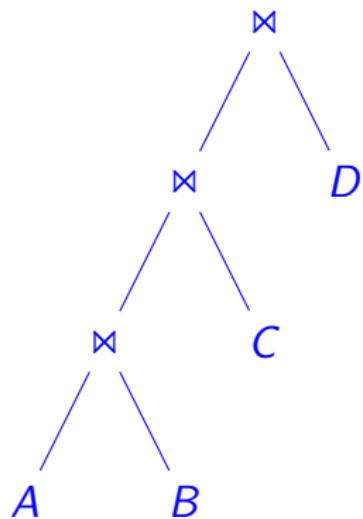
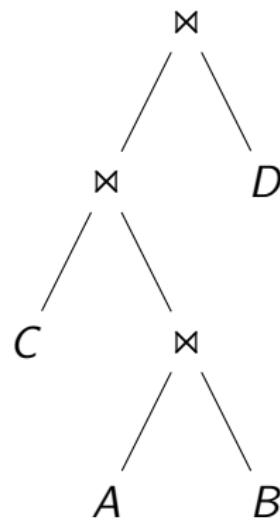
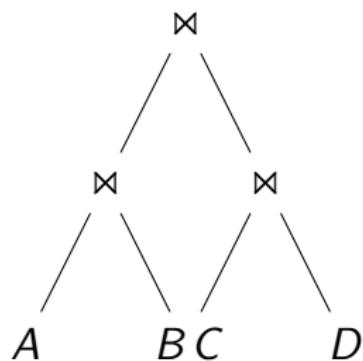
Query optimization in System R

$$A \bowtie B \bowtie C \bowtie D$$



Query optimization in System R

$$A \bowtie B \bowtie C \bowtie D$$



left-deep

Query optimization in System R

- ▶ left-deep plans differ only in the order of relations, the access method (file-scan or index) for each relation, and the join method for each join

Query optimization in System R

- ▶ left-deep plans differ only in the order of relations, the access method (file-scan or index) for each relation, and the join method for each join
- ▶ in spite of pruning, the resulting search space is still exponential in the number of relations (as we saw earlier)
 - ▶ actually is $\mathcal{O}(n2^n)$ in practice
 - ▶ with $n = 10$, this is roughly 10,000 plans (compared with 59,000 and 17,600,000,000)

Query optimization in System R

enumeration of left-deep plans: basic idea

- ▶ identify cheapest way to access each single relation in the query

Query optimization in System R

enumeration of left-deep plans: basic idea

- ▶ identify cheapest way to access each single relation in the query
- ▶ for every access method and join predicate, find the cheapest way to join in a second relation

Query optimization in System R

enumeration of left-deep plans: basic idea

- ▶ identify cheapest way to access each single relation in the query
- ▶ for every access method and join predicate, find the cheapest way to join in a second relation
- ▶ ...

Query optimization in System R

enumeration of left-deep plans, for N relations

- ▶ Pass 1: find cheapest 1-relation plan for each relation
- ▶ Pass 2: find cheapest way to join result of each 1-relation plan (as outer) to another relation (output: all 2-relation plans)
- ▶ ...
- ▶ Pass N : find cheapest way to join result of a $(N - 1)$ -relation plan (as outer) to the N th relation (output: all N -relation plans)

for each subset of relations, retain only cheapest plan overall, using $\mathcal{O}(2^N)$ space

Query optimization in System R

Suppose in our running example that we have 102 buffer pages available, and

- ▶ The Emp table has 10,000 tuples, on 1000 pages, with a clustered B-tree index on EID.
- ▶ The Works table has 20,000 tuples, on 2000 pages, sorted on $\langle \text{EID}, \text{CID} \rangle$.
- ▶ The Company table has 10,000 tuples, on 1000 pages, with a clustered B-tree index on CID.

Query optimization in System R

Pass 1: finding cheapest 1-relation plans

- ▶ for Emp, we have the local predicate $e.ecity = \text{'Best'}$. However, with only an index on EID, the cheapest plan is a file-scan (with on-the fly application of the selection predicate)

Query optimization in System R

Pass 1: finding cheapest 1-relation plans

- ▶ for Emp, we have the local predicate $e.ecity = \text{'Best'}$. However, with only an index on EID, the cheapest plan is a file-scan (with on-the fly application of the selection predicate)
- ▶ likewise, we have file-scans as cheapest access method for Works, and Company

Query optimization in System R

Pass 2: finding cheapest 2-relation plans

- ▶ we don't consider $\{E, C\}$ since this is just a cartesian product

Query optimization in System R

Pass 2: finding cheapest 2-relation plans

- ▶ we don't consider $\{E, C\}$ since this is just a cartesian product
- ▶ for EW (or WE), we can perform merge-join at cost $1000 + 2000 = 3000$
- ▶ for WE, we can perform index nested-loop-join at cost $2000 + 3 \cdot 20,000 = 62,000$
- ▶ for EW, we can perform block-nested-loops-join at cost $1000 + 2000 \cdot (1000/100) = 21,000$

Query optimization in System R

Pass 2: finding cheapest 2-relation plans

- ▶ for WC, we can perform index nested-loop-join at cost $2000 + 3 \cdot 20,000 = 62,000$
- ▶ for WC (or CW), we can perform sort-merge-join at cost $4000 + 2000 + 1000 = 7000$
- ▶ for CW, we can perform block-nested-loops-join at cost $1000 + 2000 \cdot 10 = 21,000$

Query optimization in System R

Pass 2: finding cheapest 2-relation plans

- ▶ for WC, we can perform index nested-loop-join at cost $2000 + 3 \cdot 20,000 = 62,000$
- ▶ for WC (or CW), we can perform sort-merge-join at cost $4000 + 2000 + 1000 = 7000$
- ▶ for CW, we can perform block-nested-loops-join at cost $1000 + 2000 \cdot 10 = 21,000$

So, best cost for $\{E, W\}$ is 3000, and for $\{W, C\}$ is 7000

Query optimization in System R

Pass 3: finding cheapest 3-relation plans

- ▶ for EW joined with C, assuming 2000 pages in result of EW, we can perform sort merge-join at cost $2000 + 4000 + 1000 = 7000$

Query optimization in System R

Pass 3: finding cheapest 3-relation plans

- ▶ for EW joined with C, assuming 2000 pages in result of EW, we can perform sort merge-join at cost $2000 + 4000 + 1000 = 7000$
- ▶ for WC joined with E, assuming 2000 pages in result of WC, we can perform sort merge-join at cost $2000 + 4000 + 1000 = 7000$

The life of a query: logical optimization

So, best overall plan is

- ▶ a merge-join of Emp and Works (with local predicates applied)
- ▶ followed by a sort-merge-join with Company,
- ▶ followed by the final projection

at cost $3000 + 7000 = 10000$ I/Os

Summary

- ▶ Query optimizer is at the heart of the query engine
- ▶ the paradigm typically followed is cost-based optimization
- ▶ System R follows a dynamic programming approach to plan space search
 - ▶ other practical approaches include randomized (e.g., simulated annealing) and genetic algorithms

Summary

- ▶ Query optimizer is at the heart of the query engine
- ▶ the paradigm typically followed is cost-based optimization
- ▶ System R follows a dynamic programming approach to plan space search
 - ▶ other practical approaches include randomized (e.g., simulated annealing) and genetic algorithms

Next time (Friday 22 May): distributed data management

Distributed data management

Lecture 8
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

22 May 2015

Admin

- ▶ project update (project part 3) due by the end of Monday
- ▶ team meetings with instructor next week
 - ▶ sign up your team in Doodle (choose exactly one of the unclaimed time slots)

Agenda

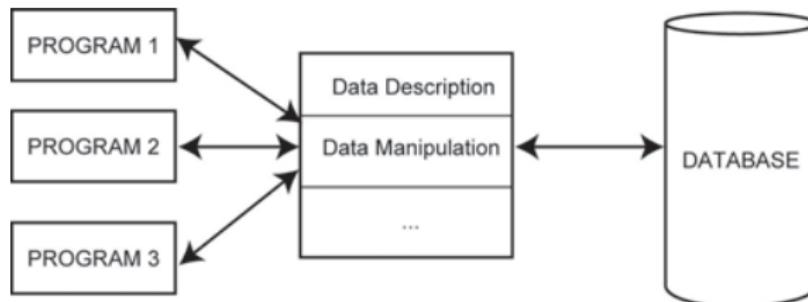
Today:

- ▶ overview of distributed DBMSs
- ▶ distributed query processing

The story so far ...

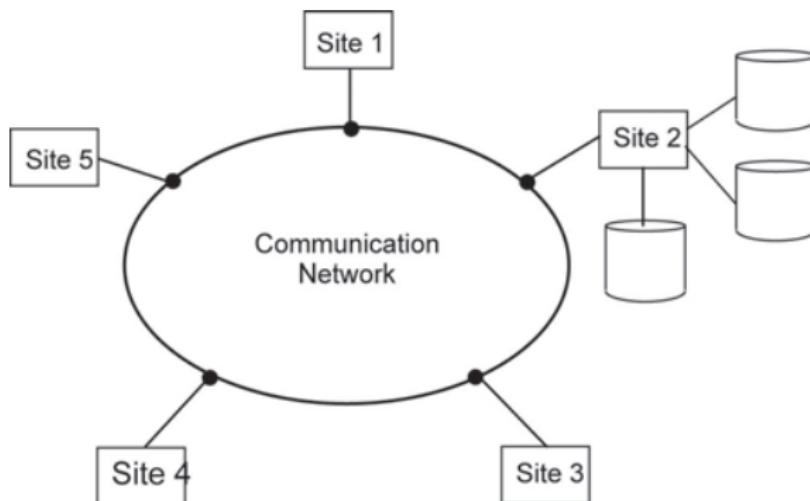
Up to this point, we've positioned a DBMS as:

1. protector of a precious commodity
2. centralized
3. directly interacting with clients



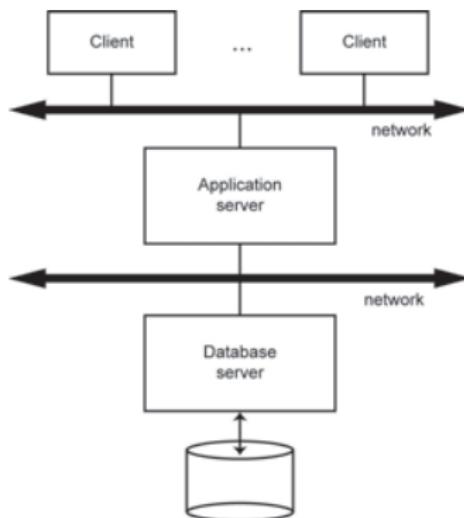
The story so far ...

- ▶ of course, we no longer live in the age of the mainframe
- ▶ in particular, a centralized DB is now typically served to many clients over a network (i.e., client-server/multi-tier architecture)



The story so far ...

- ▶ of course, we no longer live in the age of the mainframe
- ▶ in particular, a centralized DB is now typically served to many clients over a network (i.e., client-server/multi-tier architecture)

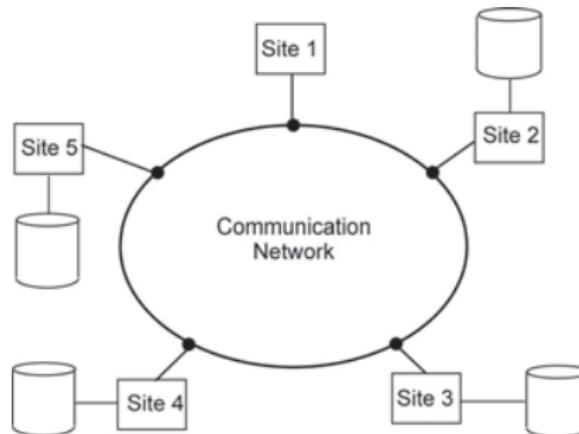


Distribution of resources

In fact, distribution is now often the norm

- ▶ data/storage
- ▶ CPUs
- ▶ clients

and it is desirable to both deal with and take advantage of this distribution



Distributed DBMSs

Today, we survey the broader landscape introduced by distribution

Distributed database

- ▶ a collection of multiple, logically interrelated databases distributed over a computer network

Distributed DBMSs

Today, we survey the broader landscape introduced by distribution

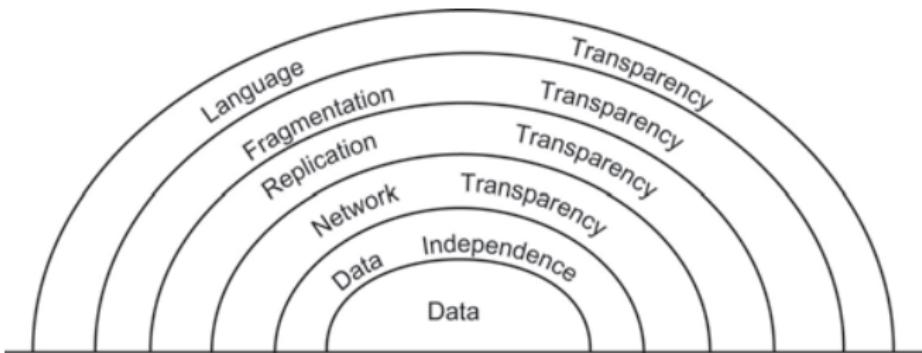
Distributed database

- ▶ a collection of multiple, logically interrelated databases distributed over a computer network

Distributed DBMS

- ▶ a software system that permits the management of a distributed database which makes the distribution transparent to its clients

Distributed DBMSs: transparency



Distributed DBMSs: hope & reality

Promises of distributed DBMSs

- ▶ transparent management of distributed and replicated data
- ▶ increased reliability through replication
- ▶ improved performance through data localization (fragmentation) and parallelism in query processing
- ▶ easier system expansion and language transparency

Distributed DBMSs: hope & reality

Promises of distributed DBMSs

- ▶ transparent management of distributed and replicated data
- ▶ increased reliability through replication
- ▶ improved performance through data localization (fragmentation) and parallelism in query processing
- ▶ easier system expansion and language transparency

Complications introduced by distributed DBMSs

- ▶ maintenance of replicated data
- ▶ dealing with communication and site failures
- ▶ synchronization of distributed transactions

Distributed DBMSs: architectures

centralized systems

- ▶ client-server/multi-tier

Distributed DBMSs: architectures

centralized systems

- ▶ client-server/multi-tier

parallel systems: multiple CPUs & disks in parallel,
centrally administered

- ▶ (shared memory or/and disk)
- ▶ shared nothing

Distributed DBMSs: architectures

centralized systems

- ▶ client-server/multi-tier

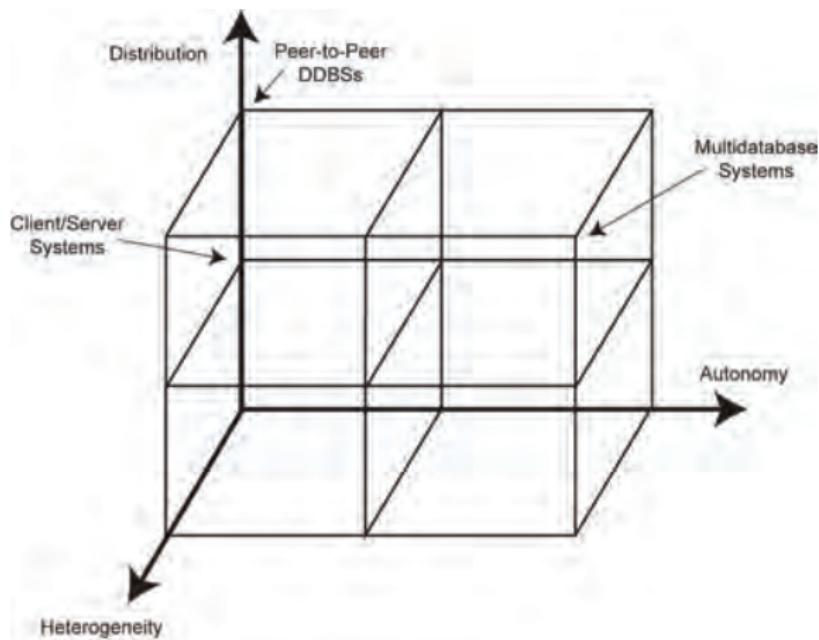
parallel systems: multiple CPUs & disks in parallel,
centrally administered

- ▶ (shared memory or/and disk)
- ▶ shared nothing

distributed systems: greater separation in
administration and communication

- ▶ homogeneous sites: peer-to-peer
- ▶ heterogeneous sites: multidatabase, federated,
mediator systems

Distributed DBMSs: architectures



Parallel DBMSs

Parallel DBMSs

Suppose we have a terabyte of data, which we'd like to scan

- ▶ with one “pipe” of 10 MB/s, this will take a little over one day

Parallel DBMSs

Suppose we have a terabyte of data, which we'd like to scan

- ▶ with one “pipe” of 10 MB/s, this will take a little over one day
- ▶ with 1,000 such pipes scanning separate partitions (i.e., disjoint subsets) in parallel, this will take about 90s

Parallel DBMSs

Parallelism is very natural for relational data processing

- ▶ bulk processing over many partitions in parallel
- ▶ natural pipelining in parallel
- ▶ inexpensive hardware is often sufficient
- ▶ clients are not forced to think in parallel (just SQL)

Parallel DBMSs

Parallelism is very natural for relational data processing

- ▶ bulk processing over many partitions in parallel
- ▶ natural pipelining in parallel
- ▶ inexpensive hardware is often sufficient
- ▶ clients are not forced to think in parallel (just SQL)

Extremely successful in practice: every major vendor has a parallel server product (e.g., OLTP, OLAP)

Parallel DBMSs: query processing

Forms of query parallelism

- ▶ **intra-operator** parallelism
 - ▶ multiple sites working to compute a single operator
(e.g., selection, sort, join)

Parallel DBMSs: query processing

Forms of query parallelism

- ▶ **intra-operator** parallelism
 - ▶ multiple sites working to compute a single operator (e.g., selection, sort, join)
- ▶ **inter-operator** parallelism
 - ▶ each operator running concurrently on different sites, with pipelining of results

Parallel DBMSs: query processing

Forms of query parallelism

- ▶ **intra-operator** parallelism
 - ▶ multiple sites working to compute a single operator (e.g., selection, sort, join)
- ▶ **inter-operator** parallelism
 - ▶ each operator running concurrently on different sites, with pipelining of results
- ▶ **inter-query** parallelism
 - ▶ different queries running concurrently on different sites

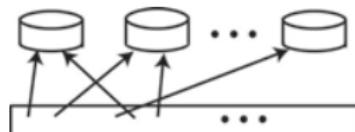
Parallel DBMSs: query processing

Forms of query parallelism

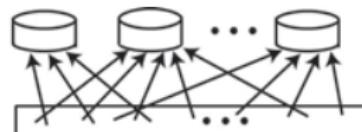
- ▶ **intra-operator** parallelism
 - ▶ multiple sites working to compute a single operator (e.g., selection, sort, join)
- ▶ **inter-operator** parallelism
 - ▶ each operator running concurrently on different sites, with pipelining of results
- ▶ **inter-query** parallelism
 - ▶ different queries running concurrently on different sites

We illustrate **intra-operator** parallelism in selection, sort, and join processing

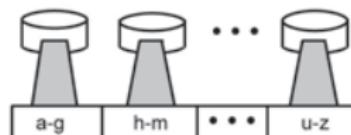
Parallel DBMSs: data placement



(a) Round-Robin



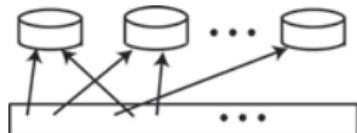
(b) Hashing



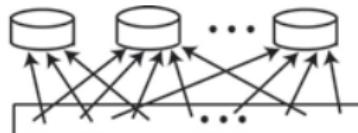
(c) Range

- (a) round-robin:** given n partition blocks, i th tuple is assigned to block $(i \bmod n)$
- ▶ load-balanced
 - ▶ selections and range queries require full scan

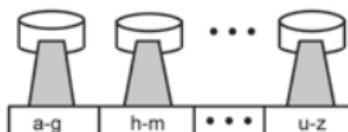
Parallel DBMSs: data placement



(a) Round-Robin



(b) Hashing

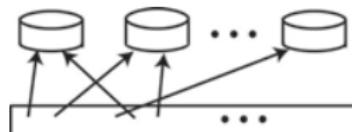


(c) Range

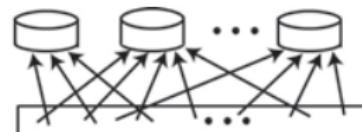
(b) hash: tuples assigned to partition blocks based on hash value

- ▶ problems with data skew
- ▶ range queries require full scan
- ▶ excellent for point-selections

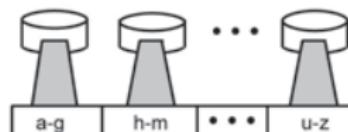
Parallel DBMSs: data placement



(a) Round-Robin



(b) Hashing



(c) Range

(c) range: tuples assigned to partition blocks in intervals

- ▶ problems with data skew
- ▶ excellent for point-selections and range queries

Parallel DBMSs: selections

parallel point-selections

- ▶ round-robin partitioning ☺
- ▶ hash partitioning ☺
- ▶ range partitioning ☺

Parallel DBMSs: selections

parallel point-selections

- ▶ round-robin partitioning ☹
- ▶ hash partitioning ☺
- ▶ range partitioning ☺

parallel range-selections

- ▶ round-robin partitioning ☹
- ▶ hash partitioning ☹
- ▶ range partitioning ☺

Parallel DBMSs: selections

parallel point-selections

- ▶ round-robin partitioning ☹
- ▶ hash partitioning ☺
- ▶ range partitioning ☺

parallel range-selections

- ▶ round-robin partitioning ☹
- ▶ hash partitioning ☹
- ▶ range partitioning ☺

... but good behavior is often offset by the costs introduced by data skew.

Parallel DBMSs: sorting

parallel sorting: sorting phases are intrinsically parallel

- ▶ scan in parallel, range-partition as you go
- ▶ use standard algorithm for local sorting
- ▶ resulting data is sorted and range-partitioned

Parallel DBMSs: sorting

parallel sorting: sorting phases are intrinsically parallel

- ▶ scan in parallel, range-partition as you go
- ▶ use standard algorithm for local sorting
- ▶ resulting data is sorted and range-partitioned



Jim Gray

Parallel DBMSs: joins

$R \bowtie S$: nested loop join

- ▶ **input.** R_1, \dots, R_m : fragments of R
- ▶ **input.** S_1, \dots, S_n : fragments of S

Parallel DBMSs: joins

$R \bowtie S$: nested loop join

- ▶ **input.** R_1, \dots, R_m : fragments of R
- ▶ **input.** S_1, \dots, S_n : fragments of S
- ▶ In the 1st phase, all m sites containing fragments of R ship their fragment to all n sites containing a fragment of S
- ▶ In the 2nd phase, joins are performed locally at each S site

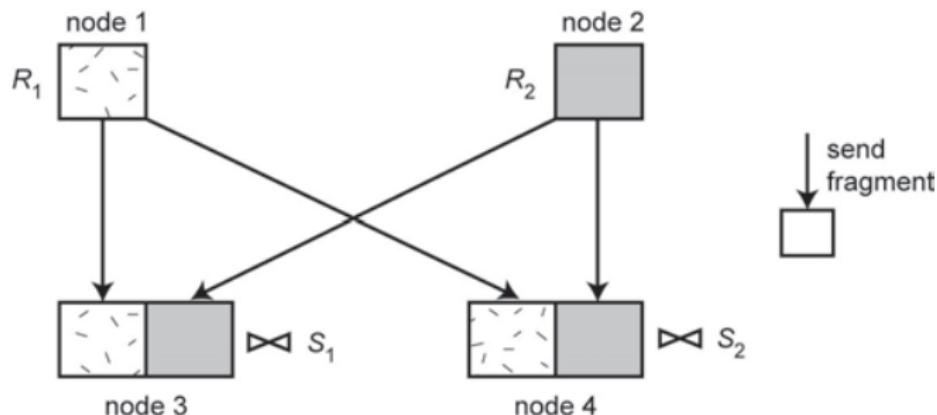
Parallel DBMSs: joins

$R \bowtie S$: nested loop join

- ▶ **input.** R_1, \dots, R_m : fragments of R
- ▶ **input.** S_1, \dots, S_n : fragments of S
- ▶ In the 1st phase, all m sites containing fragments of R ship their fragment to all n sites containing a fragment of S
- ▶ In the 2nd phase, joins are performed locally at each S site
- ▶ **output.** T_1, \dots, T_n : result fragments

So, $R \bowtie S$ is computed as $\bigcup_{1 \leq i \leq n} (R \bowtie S_i)$

Parallel DBMSs: joins



Parallel nested loop with $m = n = 2$

Parallel DBMSs: joins

$R \bowtie S$: (sort-) merge join

- ▶ **input.** R_1, \dots, R_m : fragments of R
- ▶ **input.** S_1, \dots, S_n : fragments of S

Parallel DBMSs: joins

$R \bowtie S$: (sort-) merge join

- ▶ **input.** R_1, \dots, R_m : fragments of R
- ▶ **input.** S_1, \dots, S_n : fragments of S
- ▶ In the 1st phase, range partition R and S across p sites
- ▶ The 2nd sort and merge phase is performed locally at each site

Parallel DBMSs: joins

$R \bowtie S$: (sort-) merge join

- ▶ **input.** R_1, \dots, R_m : fragments of R
- ▶ **input.** S_1, \dots, S_n : fragments of S
- ▶ In the 1st phase, range partition R and S across p sites
- ▶ The 2nd sort and merge phase is performed locally at each site
- ▶ **output.** T_1, \dots, T_p : result fragments

Parallel DBMSs: joins

$R \bowtie S$: hash join

- ▶ **input.** R_1, \dots, R_m : fragments of R
- ▶ **input.** S_1, \dots, S_n : fragments of S

Parallel DBMSs: joins

$R \bowtie S$: hash join

- ▶ **input.** R_1, \dots, R_m : fragments of R
- ▶ **input.** S_1, \dots, S_n : fragments of S
- ▶ In the 1st phase, each of the m sites containing a fragment of R hashes and ship its tuples to p sites.
- ▶ In the 2nd phase, the S fragments are also hashed and shipped, and joins are performed locally at each of the p sites

Parallel DBMSs: joins

$R \bowtie S$: hash join

- ▶ **input.** R_1, \dots, R_m : fragments of R
- ▶ **input.** S_1, \dots, S_n : fragments of S
- ▶ In the 1st phase, each of the m sites containing a fragment of R hashes and ship its tuples to p sites.
- ▶ In the 2nd phase, the S fragments are also hashed and shipped, and joins are performed locally at each of the p sites
- ▶ **output.** T_1, \dots, T_p : result fragments

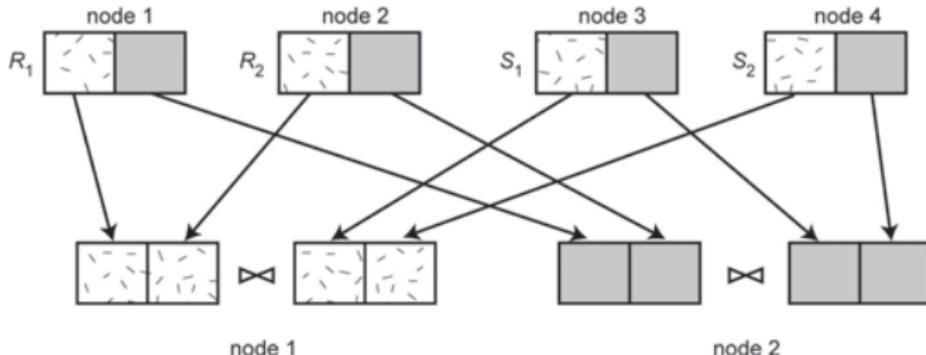
Parallel DBMSs: joins

$R \bowtie S$: hash join

So, $R \bowtie S$ is computed as

$$\bigcup_{1 \leq i \leq p} \left[\left(\bigcup_{1 \leq j \leq m} R_{ji} \right) \bowtie \left(\bigcup_{1 \leq j \leq n} S_{ji} \right) \right]$$

Parallel DBMSs: joins



Parallel hash join with $m = n = p = 2$, where results are produced at sites 1 and 2 (i.e., some transfers are local).

Parallel DBMSs

some other major issues we haven't touched on

- ▶ query optimization
- ▶ transaction management
- ▶ load balancing

Parallel DBMSs

some other major issues we haven't touched on

- ▶ query optimization
- ▶ transaction management
- ▶ load balancing

to wrap up this topic

- ▶ parallel processing is very natural for relational query processing
- ▶ many new challenges and opportunities for data management
- ▶ a major success story in practice, with new models/applications regularly emerging

Distributed DBMSs

Distributed DBMSs

two extremes of distributed DB systems

- ▶ **homogeneous** systems: same schema, tight cooperation, essentially identical sites
 - ▶ peer-to-peer

Distributed DBMSs

two extremes of distributed DB systems

- ▶ **homogeneous** systems: same schema, tight cooperation, essentially identical sites
 - ▶ peer-to-peer
- ▶ **heterogeneous** systems: varied schema and capabilities, loose cooperation
 - ▶ multi-database
 - ▶ federated databases
 - ▶ mediator systems

Distributed DBMSs: data storage

often have *replication and fragmentation* of data at multiple sites

- ▶ advantage: availability (localization)
- ▶ advantage: increased parallelism
- ▶ disadvantage: increased update cost

Distributed DBMSs: data storage

vertical fragmentation of a relation R

- ▶ $R = R_1 \bowtie \dots \bowtie R_n$
- ▶ i.e., the R_i 's are subsets of the attributes of R (including a key)

Distributed DBMSs: data storage

vertical fragmentation of a relation R

- ▶ $R = R_1 \bowtie \dots \bowtie R_n$
- ▶ i.e., the R_i 's are subsets of the attributes of R (including a key)

horizontal fragmentation of a relation R

- ▶ $R = R_1 \cup \dots \cup R_n$
- ▶ i.e., the R_i 's are subsets of tuples of R
- ▶ each fragment R_i has a guard condition g_i which defines it

$$R_i = \sigma_{g_i}(R)$$

Distributed DBMSs: data storage

Hence, at a high level, query processing involves *unfolding* the logical view definition, and then *optimizing*

Distributed DBMSs: data storage

example. suppose we have the following database schema

accounts(AID, balance, branch)

loans(LID, amount, branch)

holds(CID, AID, branch)

customers(CID, name, address)

where *holds* and *customers* are stored at the main bank office site, and (horizontal fragments of) *holds*, *accounts*, and *loans* are stored at the corresponding branch office sites (i.e., the guard at each branch b is defined as “ $branch = b$ ”)

Distributed DBMSs: data storage

if we wish to find the balances of all accounts, then we must unfold the logical *accounts* schema in terms of its fragmentation, i.e.,

$$\pi_{balance}(\text{accounts}) = \bigcup_{b \in B} \pi_{balance}(\text{accounts}_b)$$

for the set of branch names B and

$$\text{accounts}_b = \sigma_{branch=b}(\text{accounts}).$$

Distributed DBMSs: data storage

and, *updating* also faces the issues we saw for views

...

to update R with *insertion* of tuple t

- ▶ if $R = R_1 \bowtie \dots \bowtie R_n$, then insert $t[R_i]$ into fragment i

Distributed DBMSs: data storage

and, *updating* also faces the issues we saw for views

...

to update R with *insertion* of tuple t

- ▶ if $R = R_1 \bowtie \dots \bowtie R_n$, then insert $t[R_i]$ into fragment i
- ▶ if $R = R_1 \cup \dots \cup R_n$, then find one i such that $g_i(t)$ is true, and insert t into fragment i
 - ▶ if we have a choice, prefer a local fragment

Distributed DBMSs: data storage

to update R with **deletion** of tuple t

- ▶ if $R = R_1 \bowtie \dots \bowtie R_n$, then we can't simply delete $t[R_i]$ at each site i , since this impacts all tuples t' where $t'[R_i] = t[R_i]$
 - ▶ use TIDs to reconstruct and delete all and only those tuples t' where $t' = t$

Distributed DBMSs: data storage

to update R with **deletion** of tuple t

- ▶ if $R = R_1 \bowtie \dots \bowtie R_n$, then we can't simply delete $t[R_i]$ at each site i , since this impacts all tuples t' where $t'[R_i] = t[R_i]$
 - ▶ use TIDs to reconstruct and delete all and only those tuples t' where $t' = t$
- ▶ if $R = R_1 \cup \dots \cup R_n$, then delete t from each i where $g_i(t)$ is true

Distributed DBMSs: homogeneous case

assumptions

- ▶ tight site cooperation
 - ▶ we have a globally known [directory](#)
- ▶ site independence
- ▶ communication cost is significant
 - ▶ the cost of transmitting n bytes is modeled as

$$\text{cost}(n) = c_0 + c_1 n$$

for some setup cost c_0 and transfer cost c_1

Distributed DBMSs: homogeneous case

assumptions

- ▶ tight site cooperation
 - ▶ we have a globally known [directory](#)
- ▶ site independence
- ▶ communication cost is significant
 - ▶ the cost of transmitting n bytes is modeled as

$$\text{cost}(n) = c_0 + c_1 n$$

for some setup cost c_0 and transfer cost c_1

for example, suppose we have relations R_1 and R_2 distributed over sites s_1 and s_2 , with directory

	s_1	s_2
R_1	F_{11}	F_{12}
R_2	F_{21}	F_{22}

Distributed DBMSs: homogeneous case

let's consider query processing under these assumptions

Distributed DBMSs: homogeneous case

let's consider query processing under these assumptions

- ▶ $\sigma_c(R)$ and $\pi_{\bar{A}}(R)$ are straightforward

Distributed DBMSs: homogeneous case

let's consider query processing under these assumptions

- ▶ $\sigma_c(R)$ and $\pi_{\bar{A}}(R)$ are straightforward
- ▶ for $R_1 \bowtie R_2$, it would be nice if we could compute this as

$$R_1 \bowtie R_2 = (F_{11} \bowtie F_{21}) \cup (F_{12} \bowtie F_{22})$$

since this would mean no network communication

Distributed DBMSs: homogeneous case

of course, if we have $t_1 \in F_{11}$ and $t_2 \in F_{22}$ where $t_1 \bowtie t_2$, then this tuple won't appear in the result

however, it isn't unusual to have horizontal fragmentation where this strategy does work, for example

$$accounts_b \bowtie holds_b$$

and

$$employee_i \bowtie dependents_i$$

Distributed DBMSs: homogeneous case

of course, if we have $t_1 \in F_{11}$ and $t_2 \in F_{22}$ where $t_1 \bowtie t_2$, then this tuple won't appear in the result

however, it isn't unusual to have horizontal fragmentation where this strategy does work, for example

$$\textit{accounts}_b \bowtie \textit{holds}_b$$

and

$$\textit{employee}_l \bowtie \textit{dependents}_l$$

it would be nice to have the tools to determine when it is indeed possible to avoid communication ...

Distributed DBMSs: homogeneous case

definition. Two fragmented relations R_i and R_j have a placement dependency on attribute A if and only if $F_{is} \bowtie_A F_{jt} = \emptyset$ for distinct sites s and t . In other words,

$$R_i \bowtie_A R_j = \bigcup_{s \in S} (F_{is} \bowtie_A F_{js})$$

for sites S containing fragments of R_i and R_j .

Distributed DBMSs: homogeneous case

observation 1. If R_i and R_j have a placement dependency (pd) on attribute A , then they have a pd on any set of attributes B containing A

- ▶ since, if $F_{is} \bowtie_A F_{jt} = \emptyset$ (for $s \neq t$), then these fragments have no common value on A , and hence also on B .

Distributed DBMSs: homogeneous case

observation 1. If R_i and R_j have a placement dependency (pd) on attribute A , then they have a pd on any set of attributes B containing A

- ▶ since, if $F_{is} \bowtie_A F_{jt} = \emptyset$ (for $s \neq t$), then these fragments have no common value on A , and hence also on B .

Similarly, if $C \rightarrow A$ (i.e., C functionally determines A), then R_i and R_j have a pd on C

- ▶ since, if there is no pd on C , then there exist distinct fragments with tuples sharing a value on C , and hence on A , a contradiction.

Distributed DBMSs: homogeneous case

observation 2. If R_i and R_j have a pd on A and R_j and R_k have a pd on B , then

$$R_i \bowtie_A R_j \bowtie_B R_k = \bigcup_{s \in S} (F_{is} \bowtie_A F_{js} \bowtie_B F_{ks})$$

for sites S containing fragments of R_i , R_j , and R_k (i.e., can be processed without data transfer).

Distributed DBMSs: homogeneous case

observation 2. If R_i and R_j have a pd on A and R_j and R_k have a pd on B , then

$$R_i \bowtie_A R_j \bowtie_B R_k = \bigcup_{s \in S} (F_{is} \bowtie_A F_{js} \bowtie_B F_{ks})$$

for sites S containing fragments of R_i , R_j , and R_k (i.e., can be processed without data transfer).

This can be generalized to queries $Q = R_i \bowtie_B Q'$ where

1. the relation R_j to which R_i joins in Q' has a pd on (a subset of) B with R_i , and
2. Q' can be processed without data transfer.

Then, Q can be processed without data transfer.

Distributed DBMSs: homogeneous case

These observations give us the following simple algorithm to determine if a query Q can be processed without data transfer.

Distributed DBMSs: homogeneous case

let $R = \{R_1, \dots, R_n\}$ be the set of relations referenced in input query Q

1. $S \leftarrow \emptyset$
2. if a pair of relations R_i and R_j in R can be found such that they have a pd on some attribute A , and $R_i \bowtie_C R_j$ is in Q , $A \subseteq C$, place R_i and R_j in S
3. else return FALSE
4. while there is a $R_k \in R - S$ such that $\varphi(R_k)$
 - 4.1 place R_k in S
5. if $S = R$ return TRUE, else return FALSE

where $\varphi(R_k)$ holds if there is a $R_j \in S$ that has a pd with R_k on some attribute B such that $R_j \bowtie_B R_k$ is in Q (or follows from Q)

Distributed DBMSs: homogeneous case

example. Let $Q = R_1 \bowtie_A R_2 \bowtie_B R_3 \bowtie_C R_4$.

Suppose there is a pd between R_1 and R_2 on A ,
between R_2 and R_3 on B , and between R_3 and R_4
on C .

Distributed DBMSs: homogeneous case

example. Let $Q = R_1 \bowtie_A R_2 \bowtie_B R_3 \bowtie_C R_4$.

Suppose there is a pd between R_1 and R_2 on A , between R_2 and R_3 on B , and between R_3 and R_4 on C .

Initially, S is empty. In the first “if” (line 2), R_1 and R_2 can be placed in S .

Distributed DBMSs: homogeneous case

example. Let $Q = R_1 \bowtie_A R_2 \bowtie_B R_3 \bowtie_C R_4$.

Suppose there is a pd between R_1 and R_2 on A , between R_2 and R_3 on B , and between R_3 and R_4 on C .

Initially, S is empty. In the first “if” (line 2), R_1 and R_2 can be placed in S .

Following this, the while-loop (line 4) will first place R_3 in S , and then R_4 in S .

Since the loop terminates with $S = R$, we know that Q can be processed without data transfer (line 5).

Distributed DBMSs: homogeneous case

Now, of course this algorithm won't always be successful. When data transfer is necessary, the challenge becomes that of minimizing communication cost.

Distributed DBMSs: homogeneous case

Now, of course this algorithm won't always be successful. When data transfer is necessary, the challenge becomes that of minimizing communication cost.

suppose for R_1 and R_2 that

	s_1	s_2
R_1	F_{11}	
R_2		F_{22}

and we need to compute $R_1 \bowtie_A R_2$ at s_2

Distributed DBMSs: homogeneous case

solution a. ship F_{11} to s_2 and compute the join.

Distributed DBMSs: homogeneous case

solution a. ship F_{11} to s_2 and compute the join.

Then

$$cost_a = c_0 + c_1|R_1|.$$

Suppose R_1 has n tuples of two attributes, each of some fixed size (say, one byte).

Then

$$cost_a = c_0 + 2nc_1.$$

Distributed DBMSs: homogeneous case

solution b.

1. at s_2 compute $\pi_A(F_{22})$ and ship it to s_1
2. at s_1 compute $R_1 \ltimes R_2 = F_{11} \bowtie \pi_A(F_{22})$ and ship it to s_2
3. compute the final result, using the fact that $R_1 \bowtie R_2 = (R_1 \ltimes R_2) \bowtie R_2$

Distributed DBMSs: homogeneous case

solution b.

1. at s_2 compute $\pi_A(F_{22})$ and ship it to s_1
2. at s_1 compute $R_1 \ltimes R_2 = F_{11} \bowtie \pi_A(F_{22})$ and ship it to s_2
3. compute the final result, using the fact that $R_1 \bowtie R_2 = (R_1 \ltimes R_2) \bowtie R_2$

This requires two joins and two data transfers!

Distributed DBMSs: homogeneous case

Suppose $|\pi_A(R_1)| = n$ and $|\pi_A(R_2)| = 2$. Then

$$cost_{s_2 \rightarrow s_1} = c_0 + 2c_1$$

$$cost_{s_1 \rightarrow s_2} = c_0 + 4c_1$$

$$cost_b = 2c_0 + 6c_1$$

Distributed DBMSs: homogeneous case

Suppose $|\pi_A(R_1)| = n$ and $|\pi_A(R_2)| = 2$. Then

$$cost_{s_2 \rightarrow s_1} = c_0 + 2c_1$$

$$cost_{s_1 \rightarrow s_2} = c_0 + 4c_1$$

$$cost_b = 2c_0 + 6c_1$$

so solution (b) is cheaper than solution (a) when
 $c_0 + 6c_1 < 2nc_1$

- ▶ i.e., when $\pi_{R_1 \cap R_2}(R_2)$ or $R_1 \times R_2$ is small

Distributed DBMSs: homogeneous case

efficient implementation of this semi-join strategy
for processing $R_1 \bowtie R_2$ using Bloom filters

1. initialize a bit vector to all 0's
2. each value of $\pi_{R_1 \cap R_2}(R_2)$ is hashed into the bit vector (i.e., bit is set to 1)
3. bit vector is shipped to s_1

Distributed DBMSs: homogeneous case

efficient implementation of this semi-join strategy
for processing $R_1 \bowtie R_2$ using Bloom filters

1. initialize a bit vector to all 0's
2. each value of $\pi_{R_1 \cap R_2}(R_2)$ is hashed into the bit vector (i.e., bit is set to 1)
3. bit vector is shipped to s_1
4. for each tuple $t \in R_1$, hash $\pi_{R_1 \cap R_2}(t)$ into the filter
5. if t hashes to 1, then it is shipped to s_2 , and otherwise not (note that this is just a scan of R_1 , instead of a full (semi)-join)
6. at s_2 , compute $R_1 \bowtie R_2$

Distributed DBMSs: homogeneous case

efficient implementation of this semi-join strategy
for processing $R_1 \bowtie R_2$ using Bloom filters

1. initialize a bit vector to all 0's
 2. each value of $\pi_{R_1 \cap R_2}(R_2)$ is hashed into the bit vector (i.e., bit is set to 1)
 3. bit vector is shipped to s_1
 4. for each tuple $t \in R_1$, hash $\pi_{R_1 \cap R_2}(t)$ into the filter
 5. if t hashes to 1, then it is shipped to s_2 , and otherwise not (note that this is just a scan of R_1 , instead of a full (semi)-join)
 6. at s_2 , compute $R_1 \bowtie R_2$
- only false positives in step (5)

Distributed DBMSs: homogeneous case

In our example above, the semi-join eliminated all unnecessary tuples from R_1 .

This **reduction** is the advantage of the *semi-join* strategy. Unfortunately, a full reduction isn't always possible.

Distributed DBMSs: homogeneous case

In our example above, the semi-join eliminated all unnecessary tuples from R_1 .

This reduction is the advantage of the *semi-join* strategy. Unfortunately, a full reduction isn't always possible.

Example.

R		S		T	
A	B	B	C	C	A
1	1	1	1	1	2
3	4	3	3	3	3
5	5	5	6	5	5

Note that $R \bowtie S \bowtie T = \emptyset$.

Distributed DBMSs: homogeneous case

In our example above, the semi-join eliminated all unnecessary tuples from R_1 .

This **reduction** is the advantage of the *semi-join* strategy. Unfortunately, a full reduction isn't always possible.

Example.

R		S		T	
A	B	B	C	C	A
1	1	1	1	1	2
3	4	3	3	3	3
5	5	5	6	5	5

Note that $R \bowtie S \bowtie T = \emptyset$.

When is a full reduction possible?

Distributed DBMSs: homogeneous case

definition. A [hypergraph](#) \mathcal{H} is a finite set U and a set E of hyper-edges (i.e., subsets of U).

Distributed DBMSs: homogeneous case

definition. A [hypergraph](#) \mathcal{H} is a finite set U and a set E of hyper-edges (i.e., subsets of U).

A [tree decomposition](#) of \mathcal{H} is a tree T together with a set $B_t \subseteq U$, for each node t of T , such that the following two conditions hold:

1. for every $a \in U$, the set $\{t \mid a \in B_t\}$ is a subtree of T ; and,
2. every hyper-edge of \mathcal{H} is contained in one of the B_t 's.

Distributed DBMSs: homogeneous case

definition. A [hypergraph](#) \mathcal{H} is a finite set U and a set E of hyper-edges (i.e., subsets of U).

A [tree decomposition](#) of \mathcal{H} is a tree T together with a set $B_t \subseteq U$, for each node t of T , such that the following two conditions hold:

1. for every $a \in U$, the set $\{t \mid a \in B_t\}$ is a subtree of T ; and,
2. every hyper-edge of \mathcal{H} is contained in one of the B_t 's.

\mathcal{H} is [acyclic](#) if there exists a tree decomposition T of \mathcal{H} such that $\forall t \in T$, B_t is a hyper-edge of \mathcal{H} .

Distributed DBMSs: homogeneous case

definition. Given a conjunctive query

$$Q(\overline{A}) \leftarrow \alpha_1(\overline{A_1}), \dots, \alpha_n(\overline{A_n})$$

its hypergraph \mathcal{H}_Q is defined as follows. Its node set U is the set of all variables in Q and its hyperedges E are precisely $\overline{A_1}, \dots, \overline{A_n}$.

Q is **acyclic** if \mathcal{H}_Q is acyclic.

Distributed DBMSs: homogeneous case

example.

$$Q(X) \leftarrow R(X, Y), R(Y, Z)$$

Distributed DBMSs: homogeneous case

example.

$$Q(X) \leftarrow R(X, Y), R(Y, Z)$$

is acyclic.

example.

$$Q(X) \leftarrow R(X, Y), R(Y, Z), R(Z, X)$$

Distributed DBMSs: homogeneous case

example.

$$Q(X) \leftarrow R(X, Y), R(Y, Z)$$

is acyclic.

example.

$$Q(X) \leftarrow R(X, Y), R(Y, Z), R(Z, X)$$

is not acyclic (i.e., is cyclic).

Distributed DBMSs: homogeneous case

exercise.

$$Q_1(X) \leftarrow R(X, Y, Z), R(Z, U, V), S(U, Z), \\ S(X, Y), S(V, W)$$

Distributed DBMSs: homogeneous case

exercise.

$$Q_1(X) \leftarrow R(X, Y, Z), R(Z, U, V), S(U, Z), \\ S(X, Y), S(V, W)$$

is acyclic.

exercise.

$$Q_2(X) \leftarrow R(X, Y, Z), R(Z, U, V), R(X, V, W)$$

Distributed DBMSs: homogeneous case

exercise.

$$Q_1(X) \leftarrow R(X, Y, Z), R(Z, U, V), S(U, Z), \\ S(X, Y), S(V, W)$$

is acyclic.

exercise.

$$Q_2(X) \leftarrow R(X, Y, Z), R(Z, U, V), R(X, V, W)$$

is cyclic.

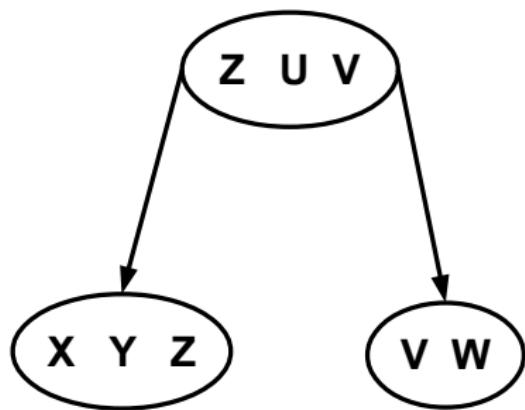
Distributed DBMSs: homogeneous case



The hypergraph \mathcal{H}_{Q_1} of

$$\begin{aligned} Q_1(X) \leftarrow & R(X, Y, Z), R(Z, U, V), S(U, Z), \\ & S(X, Y), S(V, W) \end{aligned}$$

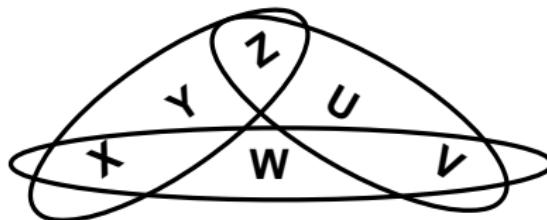
Distributed DBMSs: homogeneous case



The tree decomposition (i.e., join tree) of the hypergraph \mathcal{H}_{Q_1} of

$$\begin{aligned} Q_1(X) \leftarrow & R(X, Y, Z), R(Z, U, V), S(U, Z), \\ & S(X, Y), S(V, W) \end{aligned}$$

Distributed DBMSs: homogeneous case



The hypergraph \mathcal{H}_{Q_2} of

$$Q_2(X) \leftarrow R(X, Y, Z), R(Z, U, V), R(X, V, W)$$

Distributed DBMSs: homogeneous case

Recall from Lecture 2: Let

- ▶ FO be the full TRC
 - ▶ i.e., any of: the RA, SQL (w/o aggregation), non-recursive safe Datalog, TRC
- ▶ $Conj$ be the TRC using only \exists and \wedge
 - ▶ corresponds to: the $\{\sigma, \pi, \times\}$ fragment of RA
 - ▶ corresponds to: single SELECT-FROM-WHERE blocks
 - ▶ corresponds to: single positive safe Datalog rules
 - ▶ aka: conjunctive or SPJ queries
- ▶ $AConj$ be the “acyclic” $Conj$ queries
 - ▶ queries with join trees
 - ▶ aka: acyclic conjunctive queries

Distributed DBMSs: homogeneous case

Recall from Lecture 2: Let

- ▶ FO be the full TRC
 - ▶ i.e., any of: the RA, SQL (w/o aggregation), non-recursive safe Datalog, TRC
- ▶ $Conj$ be the TRC using only \exists and \wedge
 - ▶ corresponds to: the $\{\sigma, \pi, \times\}$ fragment of RA
 - ▶ corresponds to: single SELECT-FROM-WHERE blocks
 - ▶ corresponds to: single positive safe Datalog rules
 - ▶ aka: conjunctive or SPJ queries
- ▶ $AConj$ be the “acyclic” $Conj$ queries
 - ▶ queries with join trees
 - ▶ aka: acyclic conjunctive queries

full reduction is only possible for $AConj$ queries!

Complexity of query evaluation

Then

	<i>FO</i>	<i>Conj</i>	<i>AConj</i>
<i>combined data</i>	PSPACE-complete Logspace	NP-complete Logspace	LOGCFL-complete Linear time

where *combined* means in the size of the query and the database; and *data* means in the size of the database (i.e., for some fixed query)

Complexity of query evaluation

Then

	<i>FO</i>	<i>Conj</i>	<i>AConj</i>
<i>combined data</i>	PSPACE-complete Logspace	NP-complete Logspace	LOGCFL-complete Linear time

where *combined* means in the size of the query and the database; and *data* means in the size of the database (i.e., for some fixed query)

furthermore, determining acyclicity and then, if so, computing a “plan” for full reduction (i.e., a join tree), are both computable in polynomial time

Query containment

Checking query containment for conjunctive queries is NP-complete, as we established in Lecture 6. For FO queries, checking containment is undecidable!

However, checking query containment for acyclic conjunctive queries is tractable (i.e., computable in polynomial time)

Query containment

Checking query containment for conjunctive queries is NP-complete, as we established in Lecture 6. For FO queries, checking containment is undecidable!

However, checking query containment for acyclic conjunctive queries is tractable (i.e., computable in polynomial time)

So, an optimization strategy which can be used alongside the semi-join strategy is to first *minimize* the original query

- ▶ until not possible: remove a clause from the query, checking if the reduced query is equivalent to the original query. If so, eliminate it from the query.

Query containment

example. minimize

$$\begin{aligned} result(T) \leftarrow sales(P_1, S_1, C), cust(C, A_1), part(P_1, T), \\ sales(P_2, S_2, C), cust(C, A_2) \end{aligned}$$

Query containment

example. minimize

$$\begin{aligned} \text{result}(T) \leftarrow & \text{ sales}(P_1, S_1, C), \text{cust}(C, A_1), \text{part}(P_1, T), \\ & \text{sales}(P_2, S_2, C), \text{cust}(C, A_2) \end{aligned}$$

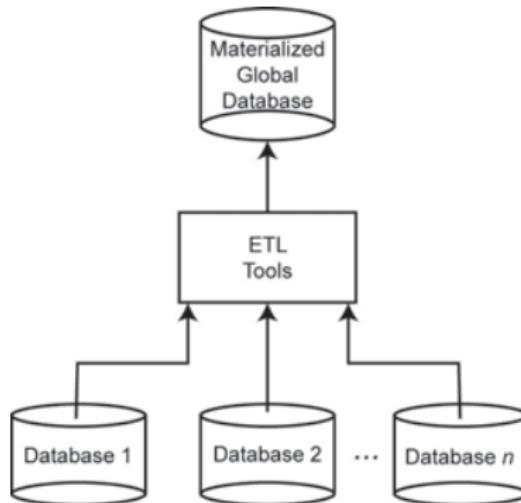
solution.

$$\text{result}(T) \leftarrow \text{sales}(P, S, C), \text{cust}(C, A), \text{part}(P, T)$$

Distributed DBMSs: heterogeneous case

Also known as **federated** or **multi-** databases, e.g.,

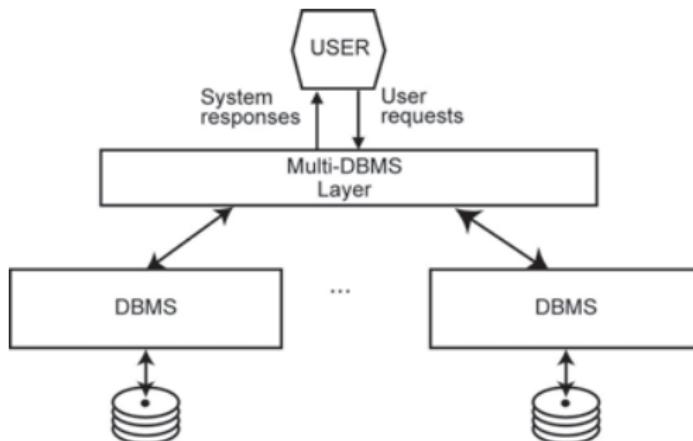
- ▶ data warehouses (materialized federation)
- ▶ mediator systems (logical federation)



Distributed DBMSs: heterogeneous case

Also known as **federated** or **multi-** databases, e.g.,

- ▶ data warehouses (materialized federation)
- ▶ mediator systems (logical federation)



Distributed DBMSs: heterogeneous case

data mappings are the basic “glue” for building heterogeneous distributed DBMSs

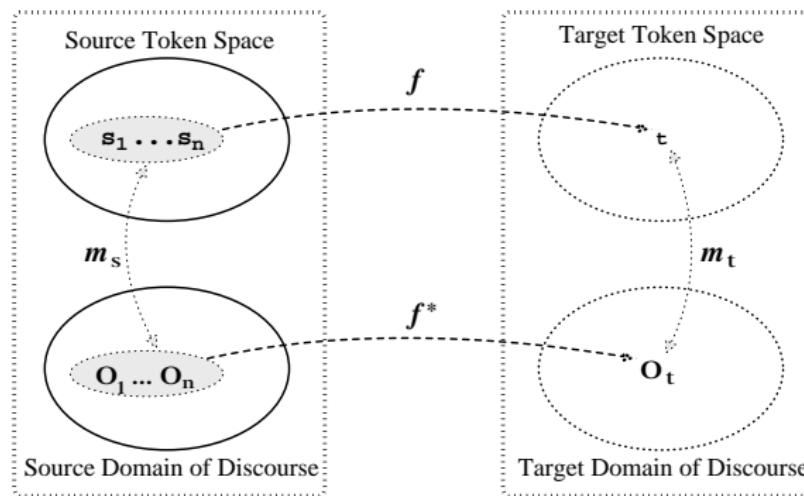
Distributed DBMSs: heterogeneous case

data mappings are the basic “glue” for building heterogeneous distributed DBMSs

two fundamental issues in data mapping

- ▶ dealing with semantic heterogeneity
 - ▶ schema matching
 - ▶ semantic mapping
- ▶ dealing with structural heterogeneity
 - ▶ query discovery

Distributed DBMSs: heterogeneous case



semantic mappings

Distributed DBMSs: heterogeneous case

FlightsA

Flights:				
Carrier	Fee	ATL29	ORD17	
AirEast	15	100	110	
JetWest	16	200	220	

FlightsB

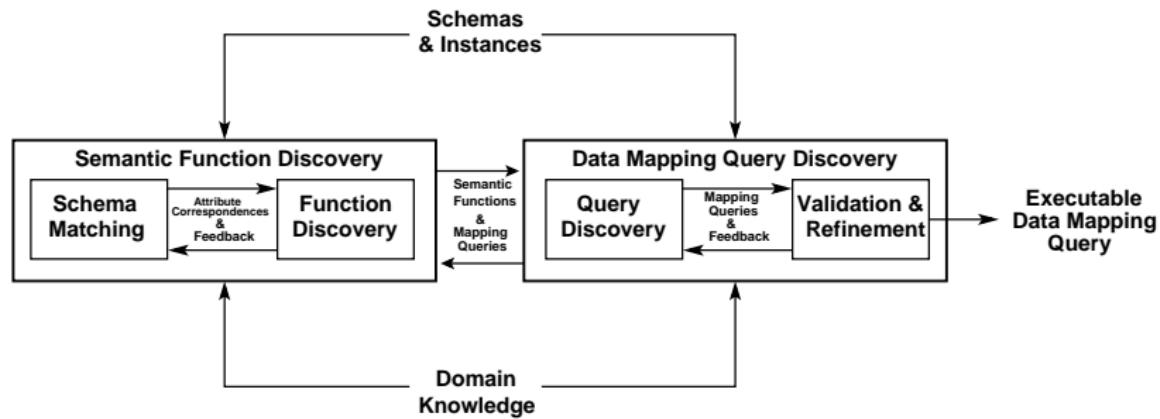
Prices:				
Carrier	Route	Cost	AgentFee	
AirEast	ATL29	100	15	
JetWest	ATL29	200	16	
AirEast	ORD17	110	15	
JetWest	ORD17	220	16	

FlightsC

AirEast:		
Route	BaseCost	TotalCost
ATL29	100	115
ORD17	110	125

JetWest:		
Route	BaseCost	TotalCost
ATL29	200	216
ORD17	220	236

Distributed DBMSs: heterogeneous case



data mapping discovery (i.e., *data integration*) is the process of overcoming structural and semantic heterogeneity

Distributed DBMSs: heterogeneous case

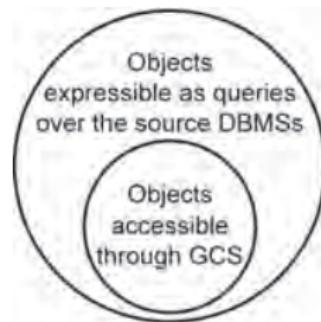
In the wrapper/mediator scenario, there are two fundamental ways of reasoning about the global coordinated schema (GCS), namely

- ▶ **global-as-view** (GAV), where global schema is a view over the local sources; and,
- ▶ **local-as-view** (LAV), where local schemas are views over the global schema.

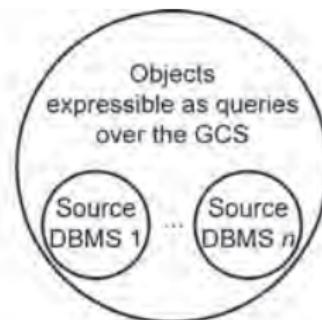
Distributed DBMSs: heterogeneous case

global-as-view (GAV), where global schema is a view over the local sources

- ▶ query processing is just view unfolding, as before
- ▶ difficult to extend/update the mediated schema



(a) GAV

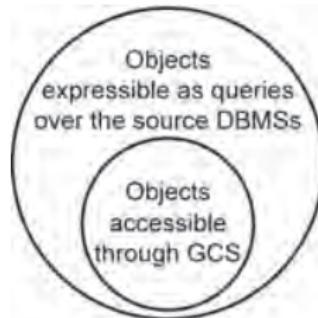


(b) LAV

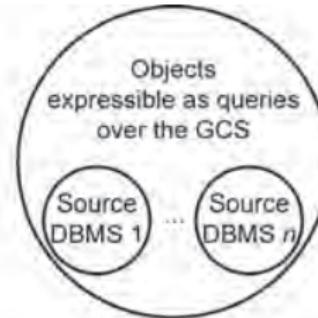
Distributed DBMSs: heterogeneous case

[local-as-view](#) (LAV), where local schemas are views over the global schema

- ▶ easy to extend/update the global schema
- ▶ how to process queries on the global schema???



(a) GAV



(b) LAV

Distributed DBMSs: heterogeneous case

LAV query processing is difficult

- ▶ finding correspondences between attributes of the global schema and local schemas requires comparison with each view
- ▶ this may be very costly, especially if there are many local sources (i.e., local views) – actually, this is NP-complete
- ▶ may not be possible to find an equivalent rewriting of the original query over the local views

Distributed DBMSs: heterogeneous case

Bucket algorithm for LAV query rewriting of query

$$Q(\overline{A}) \leftarrow \alpha_1(\overline{A_1}), \dots, \alpha_n(\overline{A_n})$$

over the global schema

1. for each α_i build a bucket b_i , and insert the head of each local view v into b_i if α_i unifies with some subgoal of v
2. for each view V of the cartesian product of all non-empty buckets, check if $V \subseteq Q$. Such a query is one way of contributing to Q .
3. return the union of all such views

Distributed DBMSs: heterogeneous case

consider the query

$$Q(B) \leftarrow accounts(A, B, R)$$

where *accounts* is now in a mediated schema, over the *accounts* table distributed locally by branch

$$Daccounts(AID, BAL) \leftarrow accounts(AID, BAL, R), \\ R = \text{downtown}$$

$$Uaccounts(AID, BAL) \leftarrow accounts(AID, BAL, R), \\ R = \text{uptown}$$

Distributed DBMSs: heterogeneous case

then Q is rewritten by the bucket algorithm as

$$\begin{aligned} Q(BAL) &\leftarrow Daccounts(AID, BAL) \\ Q(BAL) &\leftarrow Uaccounts(AID, BAL) \end{aligned}$$

Distributed DBMSs

other major issues we haven't touched on

- ▶ query optimization
- ▶ transaction management
- ▶ load balancing
- ▶ data integration solutions
- ▶ ...

Distributed DBMSs

other major issues we haven't touched on

- ▶ query optimization
- ▶ transaction management
- ▶ load balancing
- ▶ data integration solutions
- ▶ ...

to wrap up

- ▶ semi-join strategy for join processing
- ▶ data mapping problem
- ▶ query processing over mediated schemas

Recap

- ▶ Overview of distributed DBMSs
- ▶ Distributed query processing
 - ▶ parallel DBMSs
 - ▶ homogeneous distributed DBMSs (e.g., P2P)
 - ▶ heterogeneous distributed DBMSs (e.g., mediator systems)

Recap

- ▶ Overview of distributed DBMSs
- ▶ Distributed query processing
 - ▶ parallel DBMSs
 - ▶ homogeneous distributed DBMSs (e.g., P2P)
 - ▶ heterogeneous distributed DBMSs (e.g., mediator systems)

Looking ahead

- ▶ Next week: project meetings (sign up!)
- ▶ Following lecture (3 June): transaction management (data consistency, recovering from failures)

Credits

- ▶ our textbook
- ▶ Özsü & Valduriez, 2011
- ▶ Viglas, 2010
- ▶ Libkin, 2004

Transaction management

Lecture 9
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

3 June 2015

Agenda

Today's outline:

- ▶ overview of transactions
- ▶ overview of recovery management
- ▶ overview of concurrency control

The story so far ...

Last lecture, we relaxed distribution of data and query processing...

However, we are still (implicitly) positioning a DBMS as:

1. guardian of a precious commodity
2. a read-only store
3. servicing only one client

The story so far ...

Last lecture, we relaxed distribution of data and query processing...

However, we are still (implicitly) positioning a DBMS as:

1. guardian of a precious commodity
2. a read-only store
3. servicing only one client

of course, 2 and 3 are gross simplifications

how do we relax 2 and 3, while still fulfilling the obligations of 1?

Transactions

this motivates the study of “transaction” management, i.e., the management of database interactions

- ▶ which have side effects (i.e., updates), and
- ▶ which are executing concurrently (for increased throughput, and decreased response times)

Transactions

this motivates the study of “transaction” management, i.e., the management of database interactions

- ▶ which have side effects (i.e., updates), and
- ▶ which are executing concurrently (for increased throughput, and decreased response times)

a key concept in OLTP (Online Transaction Processing) systems.

Transactions

Application	Example Transaction
<i>banking</i>	withdraw money from an account
<i>securities trading</i>	purchase 100 shares of a stock
<i>insurance</i>	pay a premium
<i>inventory control</i>	record fulfillment of an order
<i>manufacturing</i>	log a step of an assembly process
<i>retail</i>	record a sale
<i>government</i>	register an automobile
<i>online shopping</i>	place an order
<i>transportation</i>	track and log a shipment
<i>social</i>	follow a friend; like a post
<i>telecom</i>	connect a phone call

Transactions

A **transaction** is a logical unit of work

- ▶ a finite list of *reads* and *writes* on a fixed collection of independent data objects
- ▶ terminated by an *abort* or *commit*

Transactions

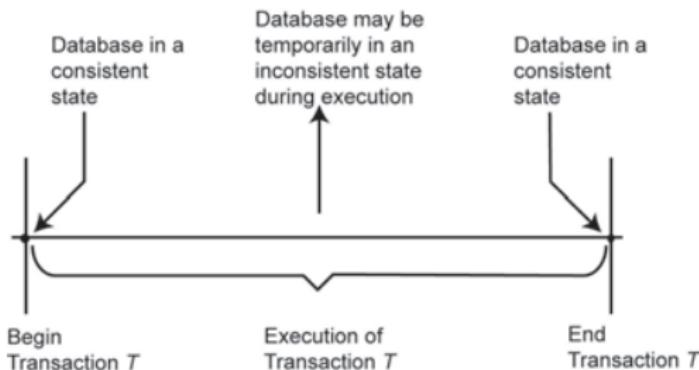
A **transaction** is a logical unit of work

- ▶ a finite list of *reads* and *writes* on a fixed collection of independent data objects
- ▶ terminated by an *abort* or *commit*
- ▶ only interacts with other transactions via read and write operations on DB objects

Transactions

A **transaction** is a logical unit of work

- ▶ a finite list of *reads* and *writes* on a fixed collection of independent data objects
- ▶ terminated by an *abort* or *commit*
- ▶ only interacts with other transactions via read and write operations on DB objects
- ▶ assumed to be consistent
 - ▶ i.e., leaves a consistent DB in a consistent state



Transactions

The SQL standard specifies that a transaction implicitly begins when an SQL statement is executed

- ▶ must be eventually followed by a COMMIT or ROLLBACK statement
- ▶ note, however, that most DBMSs auto-commit by default after each statement
- ▶ in SQL:1999, there is a BEGIN ATOMIC ... END construct for longer transactions, but this syntax has still not been widely adopted

Transactions

For our purposes, we will reason about a transaction as a finite list of reads and writes

Transactions

For our purposes, we will reason about a transaction as a finite list of reads and writes

Example of two different transactions:

- ▶ T_1 : $R_1(A), W_1(A), R_1(B), W_1(B), C_1$
- ▶ T_2 : $R_2(B), R_2(A), W_2(A), W_2(B), C_2$

Transactions

A **schedule** (also called a *history*) for a set of transactions T_1, \dots, T_n is a list of the (read, write, commit, abort) actions of the transactions which respects the order of actions of each T_i

Transactions

A **schedule** (also called a *history*) for a set of transactions T_1, \dots, T_n is a list of the (read, write, commit, abort) actions of the transactions which respects the order of actions of each T_i

A schedule is *complete* if it contains either an abort or a commit for each T_i

Transactions

Transaction example:

- ▶ T_1 : $R_1(A), W_1(A), R_1(B), W_1(B), C_1$
- ▶ T_2 : $R_2(B), R_2(A), W_2(A), W_2(B), C_2$

Transactions

Transaction example:

- ▶ T_1 : $R_1(A), W_1(A), R_1(B), W_1(B), C_1$
- ▶ T_2 : $R_2(B), R_2(A), W_2(A), W_2(B), C_2$

Non-schedule:

$R_1(A), W_1(A), R_1(B), W_2(A), R_2(B), W_1(B), R_2(A), W_2(B)$

Transactions

Transaction example:

- ▶ T_1 : $R_1(A), W_1(A), R_1(B), W_1(B), C_1$
- ▶ T_2 : $R_2(B), R_2(A), W_2(A), W_2(B), C_2$

Non-schedule:

$R_1(A), W_1(A), R_1(B), \textcolor{red}{W}_2(\textcolor{red}{A}), R_2(B), W_1(B), R_2(A), W_2(B)$

Incomplete schedule:

$R_2(B), R_1(A), W_1(A), R_2(A), R_1(B), W_2(A), W_1(B), C_1$

Transactions

Transaction example:

- ▶ T_1 : $R_1(A), W_1(A), R_1(B), W_1(B), C_1$
- ▶ T_2 : $R_2(B), R_2(A), W_2(A), W_2(B), C_2$

Non-schedule:

$R_1(A), W_1(A), R_1(B), \textcolor{red}{W}_2(A), R_2(B), W_1(B), R_2(A), W_2(B)$

Incomplete schedule:

$R_2(B), R_1(A), W_1(A), R_2(A), R_1(B), W_2(A), W_1(B), C_1$

Complete schedule:

$R_1(A), W_1(A), R_2(B), R_1(B), R_2(A), W_1(B), W_2(A), W_2(B), C_1, C_2$

ACID properties

Four desirable properties for the DBMS to enforce

ACID properties

Four desirable properties for the DBMS to enforce

- ▶ atomicity: all or nothing
 - ▶ i.e., if a transaction commits, then all of its effects are made permanent; else, it has no effect at all

ACID properties

Four desirable properties for the DBMS to enforce

- ▶ atomicity: all or nothing
 - ▶ i.e., if a transaction commits, then all of its effects are made permanent; else, it has no effect at all
- ▶ consistency: transactions map between consistent DB states
 - ▶ responsibility of programmer/client, and integrity enforcement mechanisms of the DBMS
 - ▶ transaction manager assumes all transactions are consistent

ACID properties

Four desirable properties for the DBMS to enforce

- ▶ **isolation**: each transaction is independent of all other (concurrent) transactions
 - ▶ i.e., each transaction sees a consistent database at all times
 - ▶ i.e., an executing transaction cannot reveal its results to other concurrent transactions before its commitment

ACID properties

Four desirable properties for the DBMS to enforce

- ▶ isolation: each transaction is independent of all other (concurrent) transactions
 - ▶ i.e., each transaction sees a consistent database at all times
 - ▶ i.e., an executing transaction cannot reveal its results to other concurrent transactions before its commitment
- ▶ durability: committed transactions should persist on stable storage, even if system fails

Architecture

In stable storage (i.e., on disk):

- ▶ stable database
- ▶ stable log, of before/after images for each update to the database

Architecture

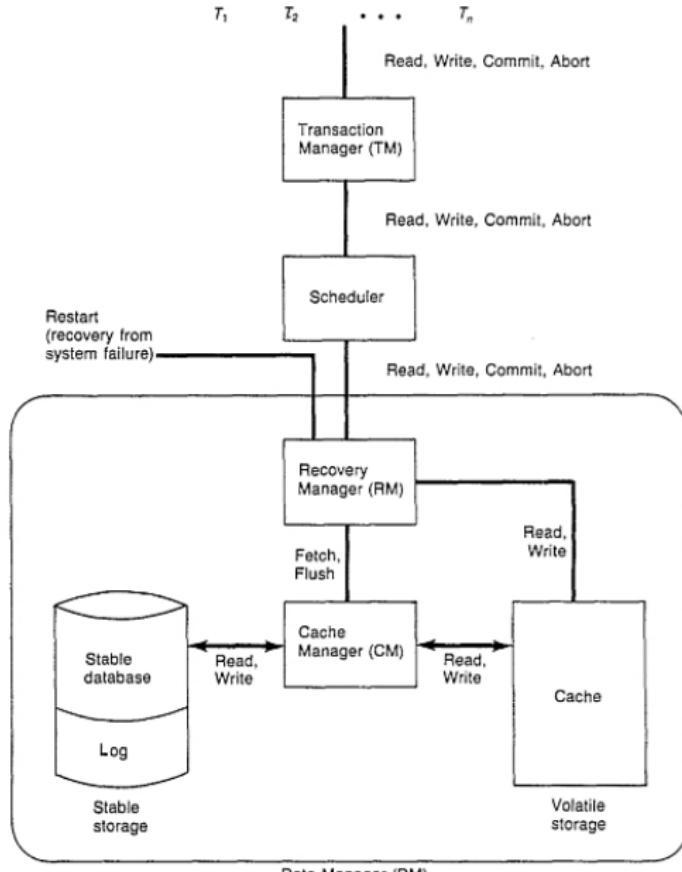
In stable storage (i.e., on disk):

- ▶ stable database
- ▶ stable log, of before/after images for each update to the database

In volatile storage (i.e., in main memory):

- ▶ a working cache (i.e., buffer) of some of the DB pages

Architecture



Architecture

Write-ahead log rule

- ▶ “a committed transaction is a completely logged transaction”
- ▶ i.e., each update must be logged in the stable log before the change itself is recorded in the stable database

Recovery management

Let's first consider how to ensure *atomicity* and *durability*, which is the responsibility of the [recovery manager](#)

Recovery management

Let's first consider how to ensure *atomicity* and *durability*, which is the responsibility of the **recovery manager**

Three types of failure

- ▶ *transaction*: (self) abort
 - ▶ responsibility of the transaction manager

Recovery management

Let's first consider how to ensure *atomicity* and *durability*, which is the responsibility of the **recovery manager**

Three types of failure

- ▶ *transaction*: (self) abort
 - ▶ responsibility of the transaction manager
- ▶ *media*: loss or corruption of stable storage
 - ▶ responsibility of the sys admin

Recovery management

Let's first consider how to ensure *atomicity* and *durability*, which is the responsibility of the **recovery manager**

Three types of failure

- ▶ *transaction*: (self) abort
 - ▶ responsibility of the transaction manager
- ▶ *media*: loss or corruption of stable storage
 - ▶ responsibility of the sys admin
- ▶ *system*: loss or corruption of volatile storage
 - ▶ responsibility of the recovery manager

Recovery management

Upon restart from a system failure, the recovery manager is responsible for

- ▶ **undo**-ing those transactions which were incomplete at the time of failure (atomicity)
- ▶ **redo**-ing committed transactions that didn't make it to stable storage (durability)

i.e., returning the stable database to a consistent state, reflecting all committed transactions at time of failure

Recovery management

Buffer management policy recap

- ▶ A **steal** policy: buffer manager can write buffer to disk before a transaction commits
- ▶ Alternative policy: **no-steal**

Recovery management

Buffer management policy recap

- ▶ A **steal** policy: buffer manager can write buffer to disk before a transaction commits
- ▶ Alternative policy: **no-steal**
- ▶ A **force** policy: all pages updated by a transaction are immediately written to disk when the transaction commits
- ▶ Alternative policy: **no-force**

Recovery management

interaction with cache management

- ▶ a recovery manager **requires undo** if “steals” in cache/buffer are possible
 - ▶ i.e., buffer pages can be forced to the stable database before a commit

Recovery management

interaction with cache management

- ▶ a recovery manager **requires undo** if “steals” in cache/buffer are possible
 - ▶ i.e., buffer pages can be forced to the stable database before a commit
- ▶ a recovery manager **requires redo** if “no-force” of cache/buffer is required upon commits
 - ▶ i.e., buffer pages aren’t necessarily flushed to the stable database upon commits

Recovery management

interaction with cache management

- ▶ a recovery manager **requires undo** if “steals” in cache/buffer are possible
 - ▶ i.e., buffer pages can be forced to the stable database before a commit
- ▶ a recovery manager **requires redo** if “no-force” of cache/buffer is required upon commits
 - ▶ i.e., buffer pages aren’t necessarily flushed to the stable database upon commits

no-steal/force is ideal combination, but impractical

steal/no-force is realistic, so let’s focus on this situation

Recovery management

So, the system fails (e.g., due to a power outage). How does RM determine which transactions to undo and which to redo (without replaying the whole log)?

Recovery management

So, the system fails (e.g., due to a power outage). How does RM determine which transactions to undo and which to redo (without replaying the whole log)?

Via periodically performing a **checkpoint**, when

1. the buffer is flushed to stable storage, and
2. a checkpoint record is written to the stable log, indicating transactions in progress.

Recovery management

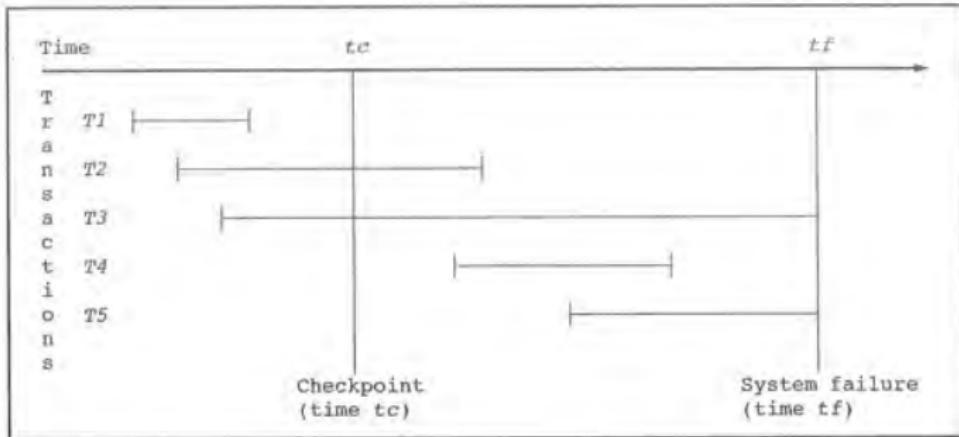
So, the system fails (e.g., due to a power outage). How does RM determine which transactions to undo and which to redo (without replaying the whole log)?

Via periodically performing a **checkpoint**, when

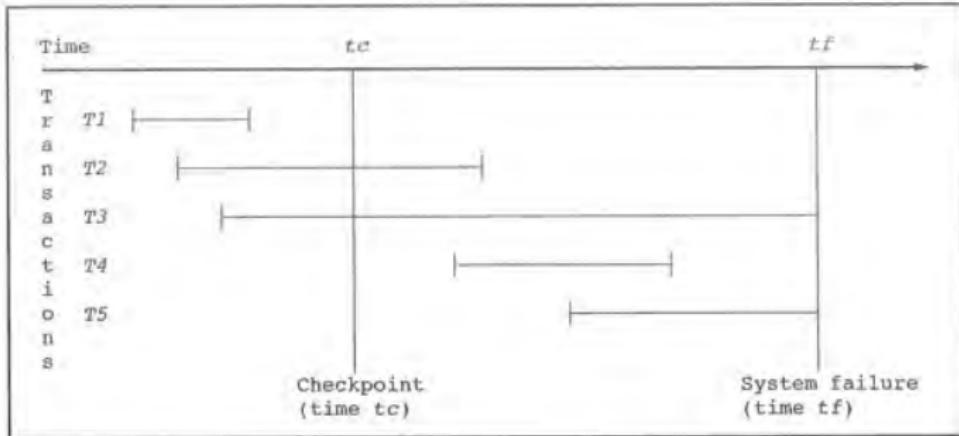
1. the buffer is flushed to stable storage, and
2. a checkpoint record is written to the stable log, indicating transactions in progress.

Tuning the frequency of checkpoints is a critical issue in practice

Recovery management

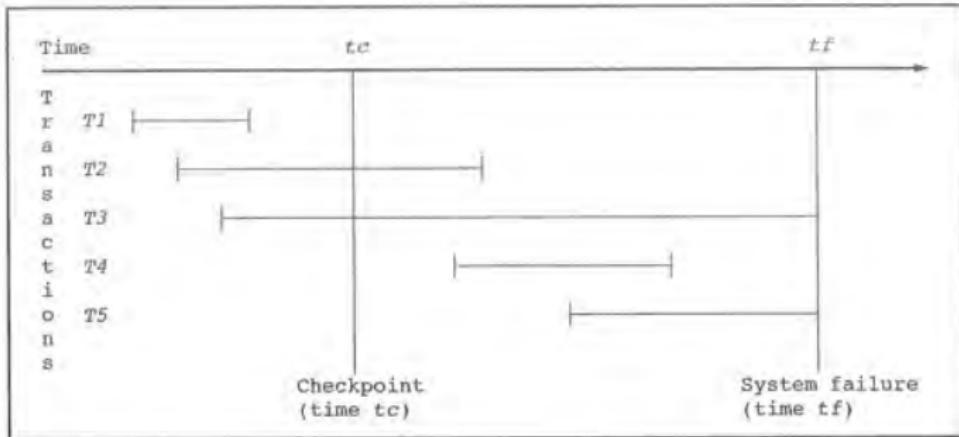


Recovery management



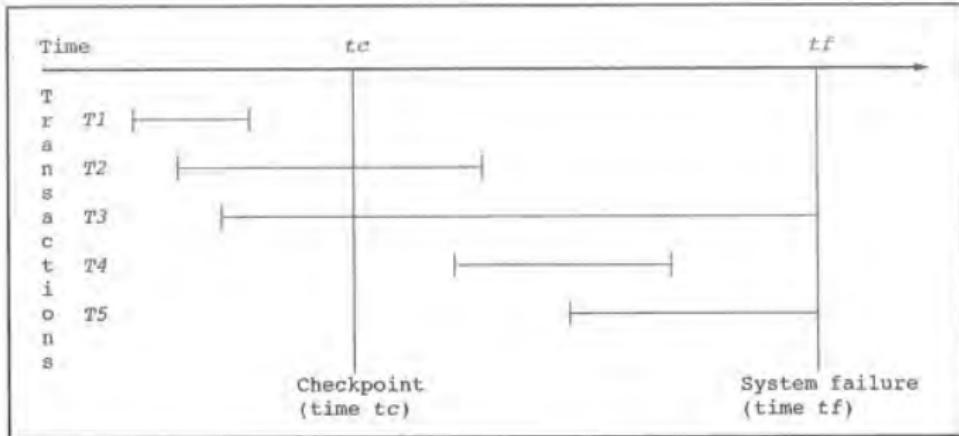
- ▶ T_1 :

Recovery management



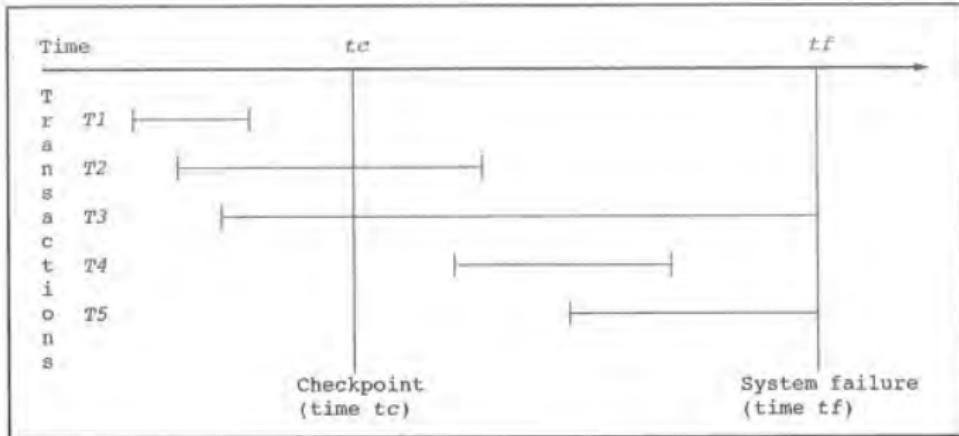
- ▶ T_1 : ignore
- ▶ T_2 :

Recovery management



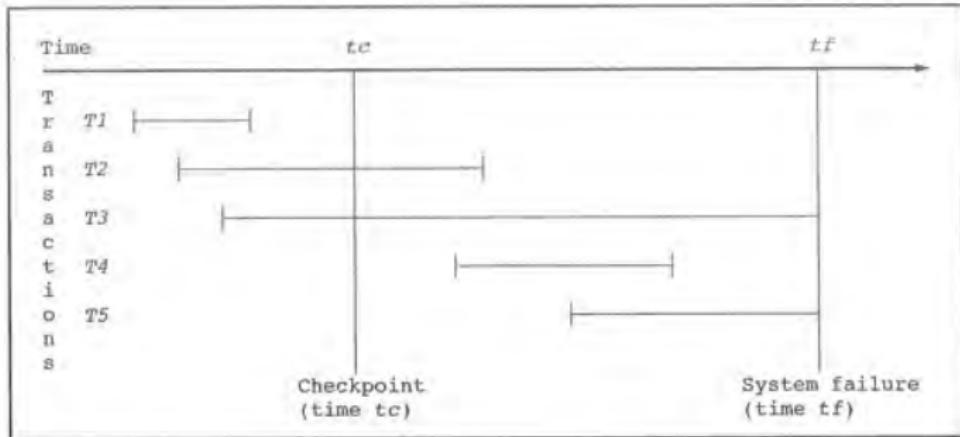
- ▶ T_1 : ignore
- ▶ T_2 : redo
- ▶ T_3 :

Recovery management



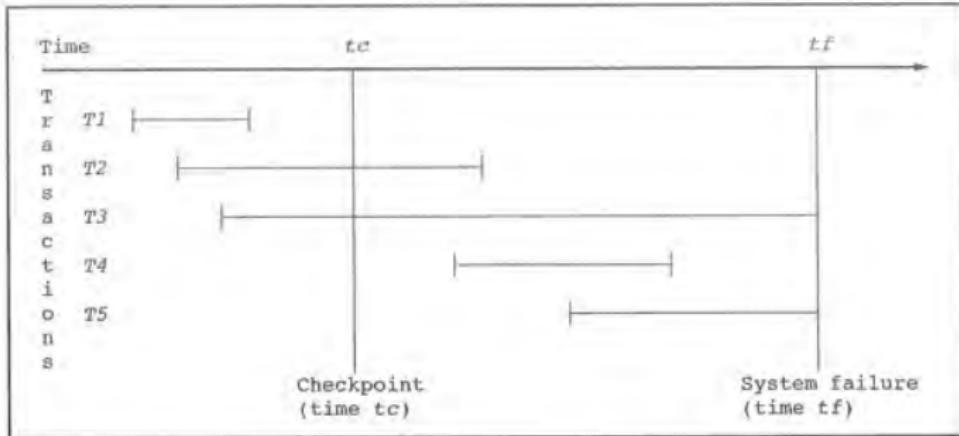
- ▶ T_1 : ignore
- ▶ T_2 : redo
- ▶ T_3 : undo
- ▶ T_4 :

Recovery management



- ▶ T_1 : ignore
- ▶ T_2 : redo
- ▶ T_3 : undo
- ▶ T_4 : redo
- ▶ T_5 :

Recovery management



- ▶ T_1 : ignore
- ▶ T_2 : redo
- ▶ T_3 : undo
- ▶ T_4 : redo
- ▶ T_5 : undo

Recovery management: restart procedure

1. $REDO \leftarrow \emptyset$
2. locate most recent checkpoint C in log
3. $UNDO \leftarrow \{T \mid T \text{ in progress at } C\}$

Recovery management: restart procedure

1. $REDO \leftarrow \emptyset$
2. locate most recent checkpoint C in log
3. $UNDO \leftarrow \{T \mid T \text{ in progress at } C\}$
4. while scanning forward from C to end of log
 - 4.1 if BEGIN TRANSACTION found for transaction T
 - ▶ $UNDO \leftarrow UNDO \cup \{T\}$
 - 4.2 if COMMIT found for transaction T ,
 - ▶ $UNDO \leftarrow UNDO - \{T\}$
 - ▶ $REDO \leftarrow REDO \cup \{T\}$

Recovery management: restart procedure

1. $REDO \leftarrow \emptyset$
2. locate most recent checkpoint C in log
3. $UNDO \leftarrow \{T \mid T \text{ in progress at } C\}$
4. while scanning forward from C to end of log
 - 4.1 if BEGIN TRANSACTION found for transaction T
 - ▶ $UNDO \leftarrow UNDO \cup \{T\}$
 - 4.2 if COMMIT found for transaction T ,
 - ▶ $UNDO \leftarrow UNDO - \{T\}$
 - ▶ $REDO \leftarrow REDO \cup \{T\}$
5. working backwards in the log, undo all actions of each $T \in UNDO$

Recovery management: restart procedure

1. $REDO \leftarrow \emptyset$
2. locate most recent checkpoint C in log
3. $UNDO \leftarrow \{T \mid T \text{ in progress at } C\}$
4. while scanning forward from C to end of log
 - 4.1 if BEGIN TRANSACTION found for transaction T
 - ▶ $UNDO \leftarrow UNDO \cup \{T\}$
 - 4.2 if COMMIT found for transaction T ,
 - ▶ $UNDO \leftarrow UNDO - \{T\}$
 - ▶ $REDO \leftarrow REDO \cup \{T\}$
5. working backwards in the log, undo all actions of each $T \in UNDO$
6. working forwards in the log, redo all actions of each $T \in REDO$

Recovery management: restart procedure

1. $REDO \leftarrow \emptyset$
2. locate most recent checkpoint C in log
3. $UNDO \leftarrow \{T \mid T \text{ in progress at } C\}$
4. while scanning forward from C to end of log
 - 4.1 if BEGIN TRANSACTION found for transaction T
 - ▶ $UNDO \leftarrow UNDO \cup \{T\}$
 - 4.2 if COMMIT found for transaction T ,
 - ▶ $UNDO \leftarrow UNDO - \{T\}$
 - ▶ $REDO \leftarrow REDO \cup \{T\}$
5. working backwards in the log, undo all actions of each $T \in UNDO$
6. working forwards in the log, redo all actions of each $T \in REDO$

note that this procedure itself is a transaction ...

Concurrency control

Let's now look at ensuring *isolation* and *consistency* of transactions, the responsibility of the [transaction manager/scheduler](#)

Concurrency control

Let's now look at ensuring *isolation* and *consistency* of transactions, the responsibility of the [transaction manager/scheduler](#)

Consider the ways in which transactions can interfere with each other, via shared data objects

Obviously, two transactions only reading the same object can't interfere with each other.

Concurrency control

Let's now look at ensuring *isolation* and *consistency* of transactions, the responsibility of the [transaction manager/scheduler](#)

Consider the ways in which transactions can interfere with each other, via shared data objects

Obviously, two transactions only reading the same object can't interfere with each other.

Two actions are said to [conflict](#) on the same data object if at least one of them is a write: write-read (WR), read-write (RW), write-write (WW)

Dirty reads (WR)

Consider

- ▶ A and B are both initially 200 €
- ▶ T_1 which transfers 100 € from A to B
- ▶ T_2 which increases both A and B by 6%

Dirty reads (WR)

Consider

- ▶ A and B are both initially 200 €
- ▶ T_1 which transfers 100 € from A to B
- ▶ T_2 which increases both A and B by 6%

with the following execution history

T_1	T_2
$R_1(A)$	
$W_1(A)$	
	$R_2(A)$
	$W_2(A)$
	$R_2(B)$
	$W_2(B)$
	commit ₂
$R_1(B)$	
$W_1(B)$	
commit ₁	

Dirty reads (WR)

Consider

- ▶ A and B are both initially 200 €
- ▶ T_1 which transfers 100 € from A to B
- ▶ T_2 which increases both A and B by 6%

with the following execution history

T_1	T_2
$R_1(A)$	
$W_1(A)$	
	$R_2(A)$
	$W_2(A)$
	$R_2(B)$
	$W_2(B)$
	commit ₂
$R_1(B)$	
$W_1(B)$	
commit ₁	

what are the final values of A and B ?

Dirty reads (WR)

Final values: $A = 106$ and $B = 312$

does this correspond to the isolated execution of T_1 and T_2 ?

- ▶ if T_1 executed in isolation, followed by T_2 :
 - ▶ final values would be $A = 106$ and $B = 318$
- ▶ if T_2 executed in isolation, followed by T_1 :
 - ▶ final values would be $A = 112$ and $B = 312$

Dirty reads (WR)

Final values: $A = 106$ and $B = 312$

does this correspond to the isolated execution of T_1 and T_2 ?

- ▶ if T_1 executed in isolation, followed by T_2 :
 - ▶ final values would be $A = 106$ and $B = 318$
- ▶ if T_2 executed in isolation, followed by T_1 :
 - ▶ final values would be $A = 112$ and $B = 312$

No! This interference was caused by T_2 reading uncommitted data, also known as a dirty read

Unrepeatable reads (RW)

Consider

- ▶ A is initially 5
- ▶ T_1 increments A by 1
- ▶ T_2 decrements A by 1

Unrepeatable reads (RW)

Consider

- ▶ A is initially 5
- ▶ T_1 increments A by 1
- ▶ T_2 decrements A by 1

with the following execution history

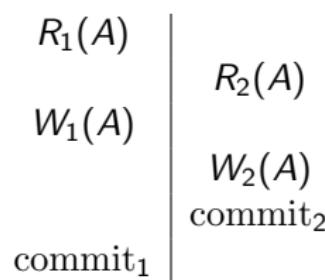
$R_1(A)$	$R_2(A)$
$W_1(A)$	
commit ₁	$W_2(A)$
	commit ₂

Unrepeatable reads (RW)

Consider

- ▶ A is initially 5
- ▶ T_1 increments A by 1
- ▶ T_2 decrements A by 1

with the following execution history



what is the final value of A ?

Unrepeatable reads (RW)

Consider

- ▶ A is initially 5
- ▶ T_1 increments A by 1
- ▶ T_2 decrements A by 1

with the following execution history

$R_1(A)$		$R_2(A)$
$W_1(A)$		$W_2(A)$
		commit ₂
commit ₁		

what is the final value of A ?

how does this compare with isolated execution?

Dirty writes (WW)

Harry and Larry must always have equal salaries.

Consider

- ▶ T_1 sets H and L's salaries to 1000 €
- ▶ T_2 sets H and L's salaries to 2000 €

Dirty writes (WW)

Harry and Larry must always have equal salaries.

Consider

- ▶ T_1 sets H and L's salaries to 1000 €
- ▶ T_2 sets H and L's salaries to 2000 €

with the following execution history

$W_1(H)$	$W_2(L)$
$W_1(L)$	
commit ₁	$W_2(H)$
	commit ₂

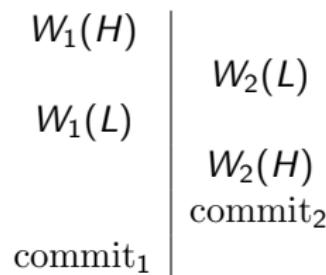
Dirty writes (WW)

Harry and Larry must always have equal salaries.

Consider

- ▶ T_1 sets H and L's salaries to 1000 €
- ▶ T_2 sets H and L's salaries to 2000 €

with the following execution history



what are the final salaries?

Dirty writes (WW)

Harry and Larry must always have equal salaries.

Consider

- ▶ T_1 sets H and L's salaries to 1000 €
- ▶ T_2 sets H and L's salaries to 2000 €

with the following execution history

$W_1(H)$	$W_2(L)$
$W_1(L)$	
commit ₁	$W_2(H)$
	commit ₂

what are the final salaries?

how does this compare with isolated execution?

Concurrency control

How do we avoid these conflicts?

- ▶ we could only allow **serial schedules**,
 - ▶ i.e., for every pair of transactions, all of the actions of one transaction execute before any of the actions of the other
 - ▶ i.e., no two transactions are interleaved

but this is too restrictive.

Concurrency control

How do we avoid these conflicts?

- ▶ we could only allow **serial** schedules,
 - ▶ i.e., for every pair of transactions, all of the actions of one transaction execute before any of the actions of the other
 - ▶ i.e., no two transactions are interleaved
- but this is too restrictive.
- ▶ what we'd like instead is to just enforce *serial behavior/outcomes* for schedules
 - ▶ i.e., those which produce the same output and have the same effect on the stable DB as some complete serial schedule of the same transactions
- also known as **Serializable** schedules

Concurrency control

We will say two schedules are **conflict equivalent** if

1. they involve the same set of actions of the same set of transactions, and
2. they order every pair of conflicting actions of two committed transactions in the same way.

Concurrency control

We will say two schedules are **conflict equivalent** if

1. they involve the same set of actions of the same set of transactions, and
2. they order every pair of conflicting actions of two committed transactions in the same way.

example. consider $T_1 = \langle R(A), R(B), W(B), C \rangle$ and $T_2 = \langle W(A), C \rangle$. Then the following schedules are conflict equivalent.

$$\begin{aligned}S_1 &= \langle T_1 : R(A), T_2 : W(A), T_2 : C, T_1 : R(B), T_1 : W(B), T_1 : C \rangle \\S_2 &= \langle T_1 : R(A), T_1 : R(B), T_2 : W(A), T_2 : C, T_1 : W(B), T_1 : C \rangle\end{aligned}$$

Concurrency control

We will say a schedule S is conflict serializable if it is conflict equivalent to some serial schedule S' .

Concurrency control

We will say a schedule S is **conflict serializable** if it is conflict equivalent to some serial schedule S' . Note: every conflict serializable schedule is serializable.

Concurrency control

We will say a schedule S is **conflict serializable** if it is conflict equivalent to some serial schedule S' . Note: every conflict serializable schedule is serializable.

- ▶ seems impractical to check S , as there are $\mathcal{O}(n!)$ serial schedules over n transactions ...

Concurrency control

We will say a schedule S is **conflict serializable** if it is conflict equivalent to some serial schedule S' . Note: every conflict serializable schedule is serializable.

- ▶ seems impractical to check S , as there are $\mathcal{O}(n!)$ serial schedules over n transactions ...

A nice way to capture potential conflicts between transactions in a schedule S is the **precedence graph** $PG(S)$ for S , which has

- ▶ one node for each committed transaction of S , and
- ▶ an edge from node i to node j iff an action of transaction i precedes and conflicts with one of the actions of transaction j .

Concurrency control

example. recall

$$S_1 = \langle T_1 : R(A), T_2 : W(A), T_2 : C, T_1 : R(B), T_1 : W(B), T_1 : C \rangle$$

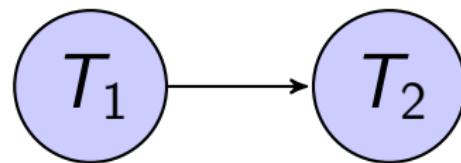
which has $PG(S_1)$ as

Concurrency control

example. recall

$$S_1 = \langle T_1 : R(A), T_2 : W(A), T_2 : C, T_1 : R(B), T_1 : W(B), T_1 : C \rangle$$

which has $PG(S_1)$ as



Concurrency control

example. recall Harry and Larry's salary updates

$$S_{HL} = \langle T_1 : W(H), T_2 : W(L), T_1 : W(L), T_2 : W(H), T_2 : C, T_1 : C \rangle$$

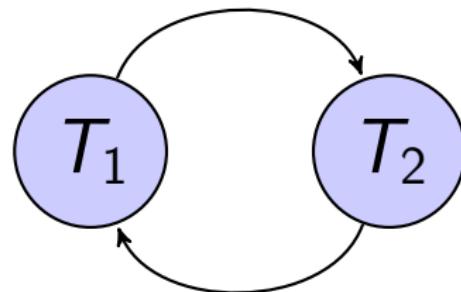
which has $PG(S_{HL})$ as

Concurrency control

example. recall Harry and Larry's salary updates

$$S_{HL} = \langle T_1 : W(H), T_2 : W(L), T_1 : W(L), T_2 : W(H), T_2 : C, T_1 : C \rangle$$

which has $PG(S_{HL})$ as



Concurrency control

This generalizes nicely as follows.

Serializability Theorem. A schedule S is conflict serializable if and only if $PG(S)$ is acyclic.

Concurrency control

exercise. Is the following schedule, over three transactions, conflict serializable?

$$\begin{aligned} T_1 : & R(A), T_1 : W(B), T_2 : R(B), T_2 : R(C), T_3 : R(A), T_3 : W(C), \\ & T_3 : W(E), T_1 : R(E), T_2 : W(D), T_3 : W(F), \\ & \quad T_2 : C, T_1 : C, T_3 : C \end{aligned}$$

Concurrency control

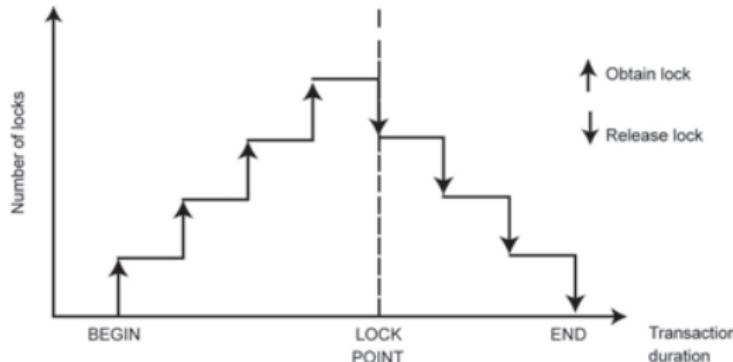
How can the transaction manager implement the Serializability Theorem, since it is still too time consuming to (statically) check and enforce such schedules?

Concurrency control: 2PL

... with two phase locking protocol, which allows concurrency while enforcing serial behavior/effect

A schedule satisfies the 2PL protocol if

1. each transaction first requests and waits for a shared (resp., exclusive) lock if it wants to read (resp., write) a DB object, and
2. a transaction cannot request additional locks once it releases any lock.



Concurrency control: 2PL

fact. If schedule S satisfies 2PL, then S is conflict serializable.

Concurrency control: 2PL

fact. If schedule S satisfies 2PL, then S is conflict serializable.

intuitively, an equivalent serial order of transactions is given by the order in which they enter their “shrinking” phases

- ▶ e.g., if $T_1 \rightarrow T_2$, then T_1 must release its lock first.

Concurrency control: 2PL - lock mgmt

We of course will need a **lock manager** to queue and grant lock requests according to the following **compatibility matrix**

Concurrency control: 2PL - lock mgmt

We of course will need a **lock manager** to queue and grant lock requests according to the following **compatibility matrix**

	read request	write request
read request	compatible	not compatible
write request	not compatible	not compatible

A **read request** is a request for a shared read-only lock

A **write request** is a request for an exclusive read/write lock

Concurrency control: 2PL - lock mgmt

We of course will need a **lock manager** to queue and grant lock requests according to the following **compatibility matrix**

	read request	write request
read request	compatible	not compatible
write request	not compatible	not compatible

A **read request** is a request for a shared read-only lock

A **write request** is a request for an exclusive read/write lock

Compatible means two transactions that request these locks to access the same data item can obtain these locks on that data item at the same time

Concurrency control: 2PL - lock mgmt

The lock manager must ensure that no transactions are **starved**, i.e., made to queue indefinitely due to transaction load handling

Concurrency control: 2PL - lock mgmt

The lock manager must ensure that no transactions are **starved**, i.e., made to queue indefinitely due to transaction load handling

- ▶ this can be addressed by ensuring that earlier requests take precedence over later requests

Concurrency control: 2PL - lock mgmt

The lock manager must ensure that no transactions are **starved**, i.e., made to queue indefinitely due to transaction load handling

- ▶ this can be addressed by ensuring that earlier requests take precedence over later requests

and that **deadlocks** are resolved

- ▶ for example, if transaction T_1 holds an exclusive lock on A and then requests a shared lock on B , while transaction T_2 holds an exclusive lock on B and then requests a shared lock on A

Concurrency control: 2PL - lock mgmt

The lock manager must ensure that no transactions are **starved**, i.e., made to queue indefinitely due to transaction load handling

- ▶ this can be addressed by ensuring that earlier requests take precedence over later requests

and that **deadlocks** are resolved

- ▶ for example, if transaction T_1 holds an exclusive lock on A and then requests a shared lock on B , while transaction T_2 holds an exclusive lock on B and then requests a shared lock on A

Once present, deadlocks are permanent and must be addressed by outside intervention

Concurrency control: 2PL - lock mgmt

Two basic approaches for handling deadlocks

- ▶ prevention
- ▶ detection and recovery

Concurrency control: 2PL - lock mgmt

Deadlock prevention. Allows transactions to be forcefully aborted (i.e., rolled back). Suppose T_i requests a lock held by T_j .

- ▶ Under a **wait-die scheme** if T_i is older than T_j , then T_i waits. Otherwise T_i is aborted and restarted with the same timestamp.
- ▶ Under a **wound-wait scheme** if T_i is older than T_j , then T_j is aborted and restarted with the same timestamp. Otherwise T_i waits.

Concurrency control: 2PL - lock mgmt

Deadlock prevention. Allows transactions to be forcefully aborted (i.e., rolled back). Suppose T_i requests a lock held by T_j .

- ▶ Under a **wait-die scheme** if T_i is older than T_j , then T_i waits. Otherwise T_i is aborted and restarted with the same timestamp.
- ▶ Under a **wound-wait scheme** if T_i is older than T_j , then T_j is aborted and restarted with the same timestamp. Otherwise T_i waits.

In other words, granted locks can be **preempted**, unlike under wait-die.

Concurrency control: 2PL - lock mgmt

Deadlock prevention. Allows transactions to be forcefully aborted (i.e., rolled back). Suppose T_i requests a lock held by T_j .

- ▶ Under a **wait-die scheme** if T_i is older than T_j , then T_i waits. Otherwise T_i is aborted and restarted with the same timestamp.
- ▶ Under a **wound-wait scheme** if T_i is older than T_j , then T_j is aborted and restarted with the same timestamp. Otherwise T_i waits.

In other words, granted locks can be **preempted**, unlike under wait-die.

Deadlocks can also be handled by **timeouts on lock requests**, but it is difficult in practice to tune timeout length

Concurrency control: 2PL - lock mgmt

Deadlock detection and recovery. A deadlock can be detected by a cycle in the [wait-for graph](#) for the transactions, having a node for each transaction and an edge from T_i to T_j iff T_i is waiting for T_j to release a lock on some object

Concurrency control: 2PL - lock mgmt

Deadlock detection and recovery. A deadlock can be detected by a cycle in the [wait-for graph](#) for the transactions, having a node for each transaction and an edge from T_i to T_j iff T_i is waiting for T_j to release a lock on some object

- ▶ in our example above, there is an edge from T_1 to T_2 and an edge from T_2 to T_1

Concurrency control: 2PL - lock mgmt

Deadlock detection and recovery. A deadlock can be detected by a cycle in the [wait-for graph](#) for the transactions, having a node for each transaction and an edge from T_i to T_j iff T_i is waiting for T_j to release a lock on some object

- ▶ in our example above, there is an edge from T_1 to T_2 and an edge from T_2 to T_1

So, with some tuned frequency, we perform [cycle detection](#) in the WFG, and then [break cycles](#) by selecting a victim transaction(s) to abort

- ▶ based on some cost function of transaction duration, transaction size and expected remaining duration, number of times already aborted, cycle size, ...

Concurrency control: recoverability

the RM deals with *system* failures. We still need to deal with *transaction* failures (i.e., aborts)

Concurrency control: recoverability

the RM deals with *system* failures. We still need to deal with *transaction* failures (i.e., aborts)

to recover, the TM must undo all of an aborted transaction writes, replacing updated values with before-images from the log.

Concurrency control: recoverability

the RM deals with *system* failures. We still need to deal with *transaction* failures (i.e., aborts)

to recover, the TM must undo all of an aborted transactions writes, replacing updated values with before-images from the log.

unfortunately, this isn't always possible ...

Concurrency control: recoverability

Consider

- ▶ A and B are both initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets B to $A + B$

Concurrency control: recoverability

Consider

- ▶ A and B are both initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets B to $A + B$

with the following execution history

$W_1(A)$	
	$R_2(A)$
	$R_2(B)$
	$W_2(B)$
	commit ₂
abort ₁	

Concurrency control: recoverability

Consider

- ▶ A and B are both initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets B to $A + B$

with the following execution history

$W_1(A)$	
	$R_2(A)$
	$R_2(B)$
	$W_2(B)$
	commit ₂
abort ₁	

it is impossible to recover from this history!
 T_2 shouldn't commit before T_1 ...

Concurrency control: recoverability

A schedule is **recoverable** if, for every transaction T that commits, T 's commit follows the commit of every transaction whose changes T read.

Concurrency control: recoverability

A schedule is **recoverable** if, for every transaction T that commits, T 's commit follows the commit of every transaction whose changes T read.

this is the bare minimum we require for isolation and consistency, to deal with aborts.

Concurrency control: recoverability

A schedule is **recoverable** if, for every transaction T that commits, T 's commit follows the commit of every transaction whose changes T read.

this is the bare minimum we require for isolation and consistency, to deal with aborts.

however, in practice this is still not enough

Concurrency control: recoverability

Consider

- ▶ A and B are both initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets B to $A + B$

Concurrency control: recoverability

Consider

- ▶ A and B are both initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets B to $A + B$

with the following execution history

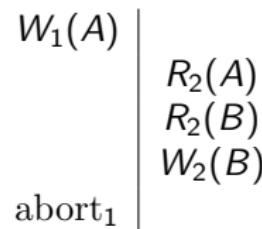
$W_1(A)$	
$R_2(A)$	
$R_2(B)$	
$W_2(B)$	
abort ₁	

Concurrency control: recoverability

Consider

- ▶ A and B are both initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets B to $A + B$

with the following execution history



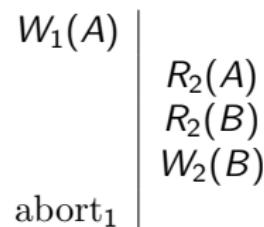
to recover, kill T_1 , and restore $A = 1 \dots$

Concurrency control: recoverability

Consider

- ▶ A and B are both initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets B to $A + B$

with the following execution history



to recover, kill T_1 , and restore $A = 1 \dots$
and, kill T_2 and restore $B = 1$

Concurrency control: recoverability

Consider

- ▶ A and B are both initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets B to $A + B$

with the following execution history

$W_1(A)$	
$R_2(A)$	
$R_2(B)$	
$W_2(B)$	
abort ₁	

to recover, kill T_1 , and restore $A = 1 \dots$
and, kill T_2 and restore $B = 1$
i.e., a cascading abort

Concurrency control: recoverability

in general, uncontrollably many aborts are possible,
which is unacceptable in practice

- ▶ hence, TM should disallow cascading aborts

Concurrency control: recoverability

in general, uncontrollably many aborts are possible, which is unacceptable in practice

- ▶ hence, TM should disallow cascading aborts

A schedule **avoids cascading aborts** (ACA) if it ensures that every transaction reads only those values that were written by committed transactions.

- ▶ also ensures recoverability

Concurrency control: recoverability

in general, uncontrollably many aborts are possible, which is unacceptable in practice

- ▶ hence, TM should disallow cascading aborts

A schedule **avoids cascading aborts** (ACA) if it ensures that every transaction reads only those values that were written by committed transactions.

- ▶ also ensures recoverability

Note that “recoverable” is a semantic notion, whereas ACA is a practical notion

Concurrency control: recoverability

Finally, consider

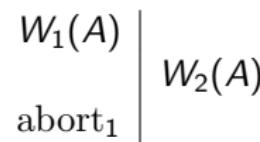
- ▶ A is initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets A to 3

Concurrency control: recoverability

Finally, consider

- ▶ A is initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets A to 3

with the following execution history

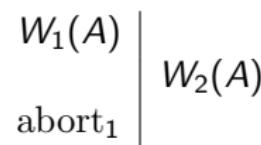


Concurrency control: recoverability

Finally, consider

- ▶ A is initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets A to 3

with the following execution history



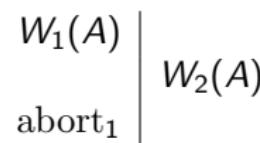
what is the value of A after undoing T_1 ?

Concurrency control: recoverability

Finally, consider

- ▶ A is initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets A to 3

with the following execution history



what is the value of A after undoing T_1 ?

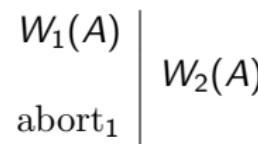
$A = 1$, whereas it should be 3!

Concurrency control: recoverability

Finally, consider

- ▶ A is initially 1
- ▶ T_1 sets A to 2
- ▶ T_2 sets A to 3

with the following execution history



what is the value of A after undoing T_1 ?

$A = 1$, whereas it should be 3!

now suppose T_2 aborts.

then $A = 2$ (using pre-image), but it should be 1!

Concurrency control: recoverability

So, in order to handle aborts just using pre/post-images from the stable log, the TM should delay writes on an object until after all transactions previously issuing writes on that object have aborted/committed.

Concurrency control: recoverability

So, in order to handle aborts just using pre/post-images from the stable log, the TM should delay writes on an object until after all transactions previously issuing writes on that object have aborted/committed.

A schedule is **strict** if it ensures that every transaction reads and writes only those data objects that were written to by committed transactions.

- ▶ also ensures ACA, and hence recoverability

Concurrency control: S2PL

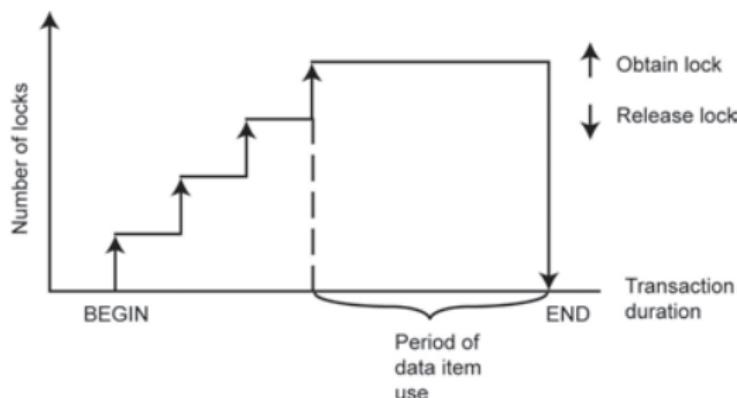
In practice, the **strict two phase locking** protocol is commonly followed to permit concurrency while enforcing serial and practically recoverable behavior/effect

- ▶ i.e., ensures strict and conflict serializable schedules

Concurrency control: S2PL

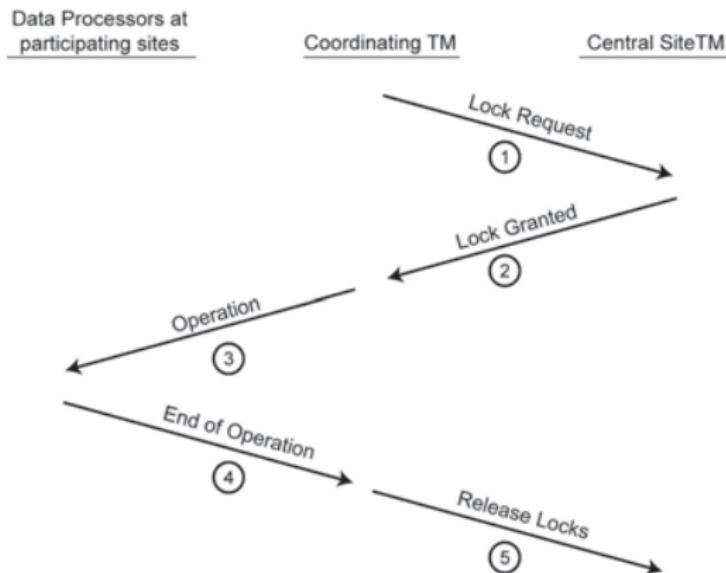
A schedule satisfies the S2PL protocol if

1. each transaction first requests and waits for a shared (resp., exclusive) lock if it wants to read (resp., write) a DB object, and
2. all locks held by a transaction are released when the transaction is complete (i.e., after commit/abort is acknowledged)



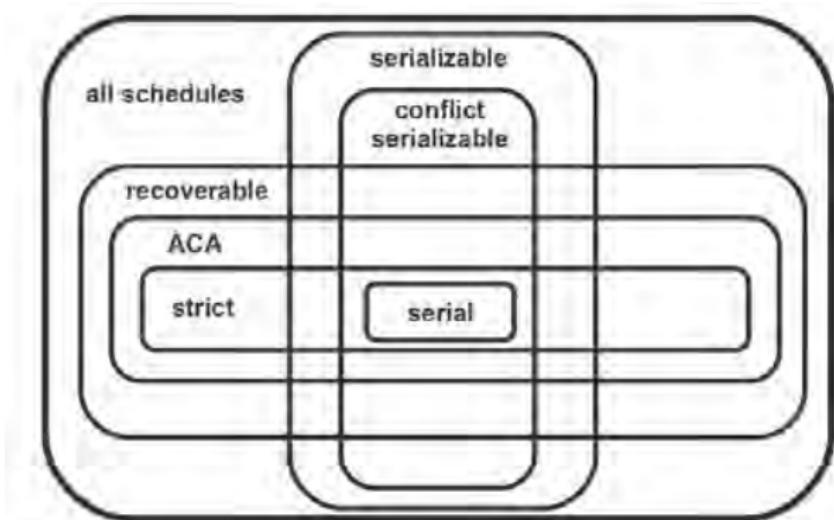
Concurrency control

Note that (S)2PL can be easily extended to distributed DBMSs

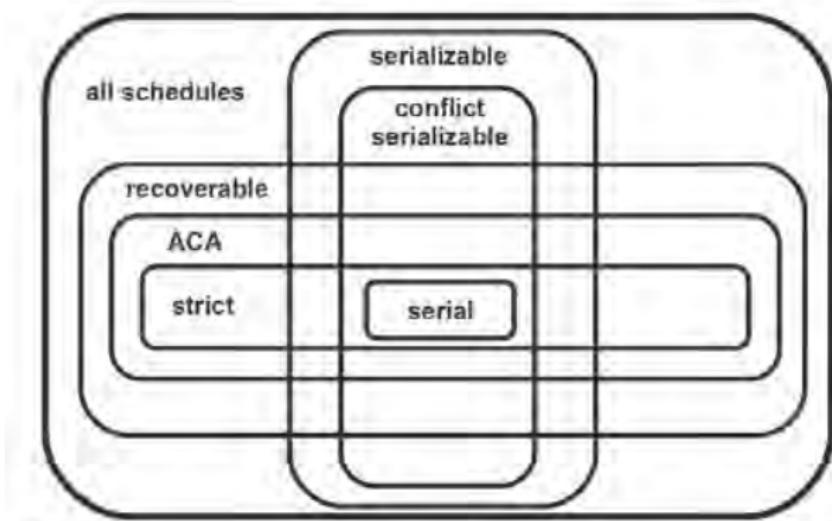


... and many more variants have been developed

Concurrency control

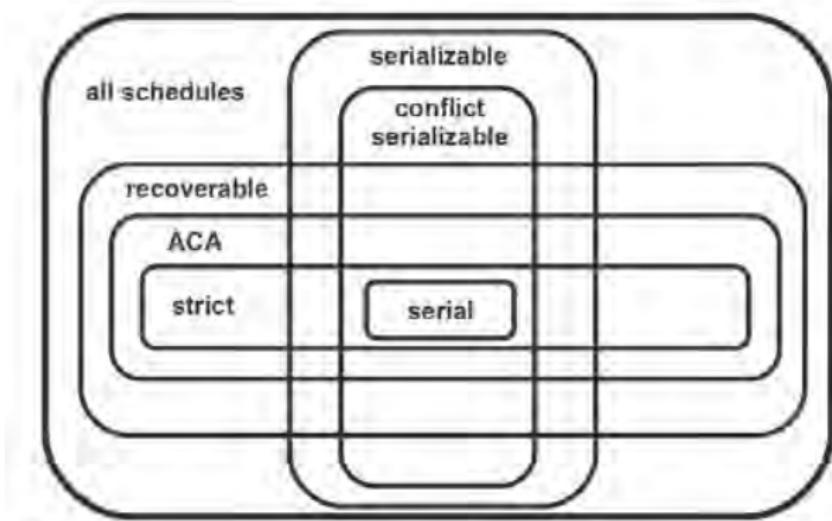


Concurrency control



Note that all classes above are distinct.

Concurrency control



Note that all classes above are distinct.

Exercise. demonstrate separation of the classes of serializable and conflict serializable transactions

Recap

overview of transactions

- ▶ concurrent units of consistent work
- ▶ ACID properties

overview of recovery management

- ▶ UNDO/REDO

overview of concurrency control

- ▶ serializable schedules, 2PL
- ▶ recoverable schedules, S2PL

Looking ahead ...

This Friday

- ▶ NoSQL and Graph databases

Next week

- ▶ Performance tuning, course summary, and exam review (Wednesday)
- ▶ First batch of project presentations: teams 1-8 (Friday)
- ▶ Physical hand-in of written assignment (Friday)

Credits

- ▶ our textbook
- ▶ Özsü & Valduriez, 2011
- ▶ Bernstein et al, 1987
- ▶ Bernstein et al, 2009
- ▶ Date, 2004

What's the big deal about big data? *NoSQL, Graphs, and Linked Data*

Lecture 10
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

5 June 2015

Agenda

1. Introduction and overview of linked and graph data, in the context of related research in database technology, and in general, data science
2. Models for graph and linked data
 - ▶ RDF and RDFS data model
 - ▶ graph database models
3. Introduction to querying over graph and linked data.
 - ▶ query languages for graphs
 - ▶ SPARQL 1.1
 - ▶ models for linked data
 - ▶ querying linked data

What's the big deal about big data?

Big Data & Data Science

Our ability to generate and capture data only continues to explode

Indeed, we live in a world that is

- ▶ increasingly driven by this generation and consumption of massive ever-growing data sets,
- ▶ which in turn is driven by both technology and a rapidly increasing “datafication” of all aspects of our public and private lives, and well, of most everything.

Managing this **Big Data** is now central to commerce, entertainment, government, education, science, ...

... and, is a core research issue in the emerging discipline of **Data Science**.

Big Data & Data Science

Due to datafication and technology trends, applications now must face data scale, heterogeneity, and distribution as **the norm**, rather than as exceptions, as in the past

- ▶ often captured as the four V's of “volume”, “variety”, “veracity”, and “velocity”

Big Data & Data Science

Due to datafication and technology trends, applications now must face data scale, heterogeneity, and distribution as **the norm**, rather than as exceptions, as in the past

- ▶ often captured as the four V's of “volume”, “variety”, “veracity”, and “velocity”

Furthermore, there is increased potential for extracting significant untapped **value from data**

- ▶ e.g., the successes of Google and Facebook are essentially built on extracting value from data

Big Data: NoSQL systems

However, there is increased difficulty in extracting “value” (a fifth V), due to the other four V's

In data management, embracing and confronting these challenges has led to an explosion of **NoSQL** systems, as alternatives to the dominant paradigm of structured data (i.e., “SQL” stores)

Big Data: NoSQL systems

Serious relaxations in “traditional” data management assumptions include

- ▶ uncertain data
- ▶ structural heterogeneity
 - ▶ semi-structured, denormalized data
 - ▶ query paradigms for such loosely structured data
- ▶ looser data consistency
- ▶ semantic heterogeneity

NoSQL systems typically embrace one or more of these

Big Data: NoSQL systems

Serious relaxations in “traditional” data management assumptions include

- ▶ uncertain data
- ▶ structural heterogeneity
 - ▶ semi-structured, denormalized data
 - ▶ query paradigms for such loosely structured data
- ▶ looser data consistency
- ▶ semantic heterogeneity

NoSQL systems typically embrace one or more of these

Note, though, that many of these relaxations predate NoSQL and even “SQL” systems (e.g., CODASYL and the network data model) ...

Big Data: NoSQL systems

Let's briefly look at relaxing **consistency** in distributed data management

ACID: the gold-standard for consistent data management

- ▶ updates to data should be atomic (A)
- ▶ transactions (i.e., blocks of read/write work) preserve all data consistency (C) rules, including having a single up-to-date copy of the data
- ▶ transactions should be performed in isolation (I) of one another
- ▶ updates should be durable (D), i.e., hard state, i.e., persistent

Big Data: NoSQL systems

Let's briefly look at relaxing **consistency** in distributed data management

ACID: the gold-standard for consistent data management

- ▶ updates to data should be atomic (A)
- ▶ transactions (i.e., blocks of read/write work) preserve all data consistency (C) rules, including having a single up-to-date copy of the data
- ▶ transactions should be performed in isolation (I) of one another
- ▶ updates should be durable (D), i.e., hard state, i.e., persistent

can be costly to enforce in a distributed context (i.e., using standard techniques, can lead to high latency)

Relaxed consistency

In the context of Big Data, data is often distributed and replicated, and the focus is on the *availability* of the data (i.e., low latency)

Hence, NoSQL systems often give no consistency guarantees, or only the guarantee of “eventual” consistency when transactions span nodes

- ▶ also known as BASE: Basically Available Soft-state services with Eventual-consistency

Justification for relaxed consistency

CAP Theorem. any networked shared-data system can have at most two of three desirable properties

- ▶ consistency (C) equivalent to having a single up-to-date copy of the data;
- ▶ high availability (A) of that data; and,
- ▶ tolerance to network partitions (P).

(Brewer, PODC 2000; Brewer, Computer, Feb 2012; Gilbert and Lynch, SIGACT News 2002)

Justification for relaxed consistency

CAP Theorem. any networked shared-data system can have at most two of three desirable properties

- ▶ **consistency** (C) equivalent to having a single up-to-date copy of the data;
- ▶ high **availability** (A) of that data; and,
- ▶ tolerance to network **partitions** (P).

(Brewer, PODC 2000; Brewer, Computer, Feb 2012; Gilbert and Lynch, SIGACT News 2002)

note that CAP only talks about partitioned systems, and not normal operation!

By explicitly handling partitions, designers can optimize consistency and availability, thereby achieving some tradeoff of all three.

Shades of relaxed consistency

PACELC. In case of partitions (P) does the system choose availability (A) or consistency (C)?; else (E), does it choose lower latency (L) or consistency (C)?

Shades of relaxed consistency

PACELC. In case of partitions (P) does the system choose availability (A) or consistency (C)?; else (E), does it choose lower latency (L) or consistency (C)?

- ▶ PA/EL: Dynamo (Amazon's key-value store), Cassandra (Apache's key-value store), Riak (key-value store)
- ▶ PA/EC: MongoDB (key-value/JSON store)
- ▶ PC/EL: PNUTS (Yahoo's key-value store)
- ▶ PC/EC (i.e., full ACID): VoltDB/H-Store (shared-nothing relational store)

(Abadi, Computer, Feb 2012)

Remarks

Consistency is not a binary choice, but rather lies on a continuum, based on context.

In practice, completely eliminating ACID consistency during normal operation is almost never justified by the CAP theorem.

Remarks

Consistency is not a binary choice, but rather lies on a continuum, based on context.

In practice, completely eliminating ACID consistency during normal operation is almost never justified by the CAP theorem.

For further discussion, see

- ▶ Lloyd et al. SOSP 2011
- ▶ Thomson et al. SIGMOD 2012
- ▶ Birman et al. Computer, Feb 2012

Big Data: NoSQL systems

Let's now consider data **structural heterogeneity**, driven by the need for so-called “polyglot persistence”

Much modern data doesn't cleanly map to a tight relational structure

- ▶ missing or multi-valued attributes
 - ▶ e.g., not all site visitors have exactly one phone number
- ▶ nested/hierarchical structure
 - ▶ ontologies, folksonomies
 - ▶ JSON, XML
- ▶ loose structure
 - ▶ social networks
 - ▶ chem-/bio- networks

Big Data: NoSQL systems

Four broad classes of approaches taken by “NoSQL” systems:

1. **key-value databases** (essentially scalable hash tables)
 - ▶ example systems: BerkelyDB, Tokyo Cabinet
2. **document databases** (generalization of key-value model to include nested structure, e.g., JSON and XML)
 - ▶ example systems: MongoDB, CouchDB, Couchbase
3. **column-family stores** (hybrid of key-value and relational model, where rows can have different schemas)
 - ▶ example systems: Cassandra, Amazon SimpleDB
4. **graph databases**
 - ▶ example systems: Neo4j, IBM DB2 RDF GraphStore

Big Data: NoSQL systems

Four broad classes of approaches taken by “NoSQL” systems:

1. **key-value databases** (essentially scalable hash tables)
 - ▶ example systems: BerkelyDB, Tokyo Cabinet
2. **document databases** (generalization of key-value model to include nested structure, e.g., JSON and XML)
 - ▶ example systems: MongoDB, CouchDB, Couchbase
3. **column-family stores** (hybrid of key-value and relational model, where rows can have different schemas)
 - ▶ example systems: Cassandra, Amazon SimpleDB
4. **graph databases**
 - ▶ example systems: Neo4j, IBM DB2 RDF GraphStore

All of these are firmly rooted in applications arising in web data management ...

Web Data

The explosion of the Web was both a precursor to and accelerant for the rise of Big Data.

Historically, web data has been modeled as

- ▶ hypertext and SGML
- ▶ text and HTML
- ▶ XML and JSON
- ▶ ...

Primarily focusing on the metaphor of the “document” ...

Web Data

The explosion of the Web was both a precursor to and accelerant for the rise of Big Data.

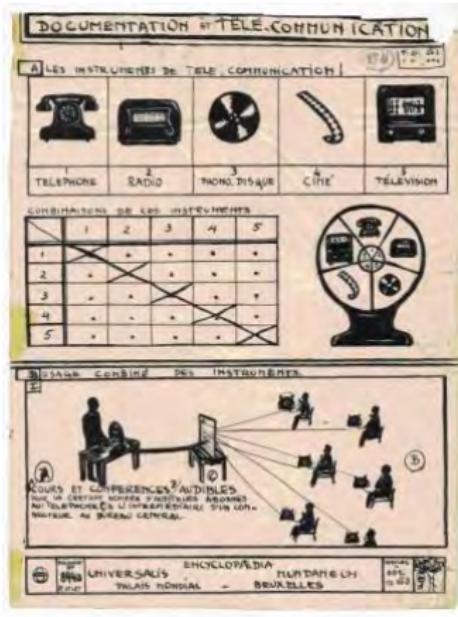
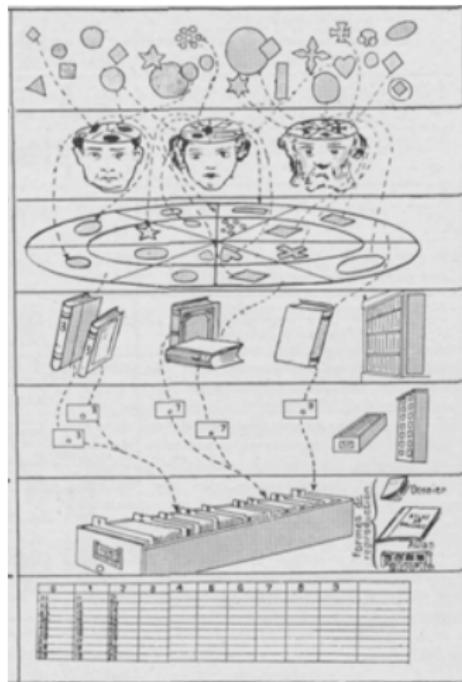
Historically, web data has been modeled as

- ▶ hypertext and SGML
- ▶ text and HTML
- ▶ XML and JSON
- ▶ ...

Primarily focusing on the metaphor of the “document” ...

But early “web” scientists had even broader visions for human knowledge ...

Web Data



The Mundaneum

as conceived by the Belgian author and peace activist [Paul Otlet](#) (1868-1944)

Web Data: linked data

This vision has resurfaced recently in the [Linked Data](#) initiative, which is essentially a vision for modeling and sharing graph data on the web, using web standards

Web Data: linked data

This vision has resurfaced recently in the [Linked Data](#) initiative, which is essentially a vision for modeling and sharing graph data on the web, using web standards

Linked data principles:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs so that they can discover more things

e.g., BBC, NYTimes, Wikipedia, Uniprot, PubMed, DBLP, ...

Web Data: linked data

This vision has resurfaced recently in the [Linked Data](#) initiative, which is essentially a vision for modeling and sharing graph data on the web, using web standards

Linked data principles:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs so that they can discover more things

e.g., BBC, NYTimes, Wikipedia, Uniprot, PubMed, DBLP, ...

Linked *open* data: public sector data as linked data

- ▶ e.g, dati.gov.it, data.gov.uk, data.gouv.fr, data.overheid.nl, data.gov, data.eu, datameti.go.jp, ...

Our focus today

We study the intersection of big data and web data, namely [linked data and graph data](#)

Our focus will be on linked and graph [data management](#) (and not information retrieval, although the distinction is increasingly fuzzy)

- ▶ i.e., scaleably implementing variations of first order logic and their application

Specifically, we will consider the challenges of [modeling and querying](#) linked and graph data

References & Credits, 1/2

- ▶ The end of theory: the data deluge makes the scientific method obsolete.
Chris Anderson. *Wired Magazine* 16(7), 2008.
http://www.wired.com/science/discoveries/magazine/16-07/pb_theory
- ▶ The second economy. W. Brian Arthur. *McKinsey Quarterly*, 2011.
http://www.mckinsey.com/insights/strategy/the_second_economy
- ▶ The rise of big data. Kenneth Cukier and Viktor Mayer-Schoenberger.
Foreign Affairs, May/June 2013.
<http://www.foreignaffairs.com/articles/139104/kenneth-neil-cukier-and-viktor-mayer-schoenberger/the-rise-of-big-data>
- ▶ The unreasonable effectiveness of data. Alon Halevy, Peter Norvig, and Fernando Pereira. *IEEE Intelligent Systems*, March/April 2009.
http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//pubs/archive/35179.pdf

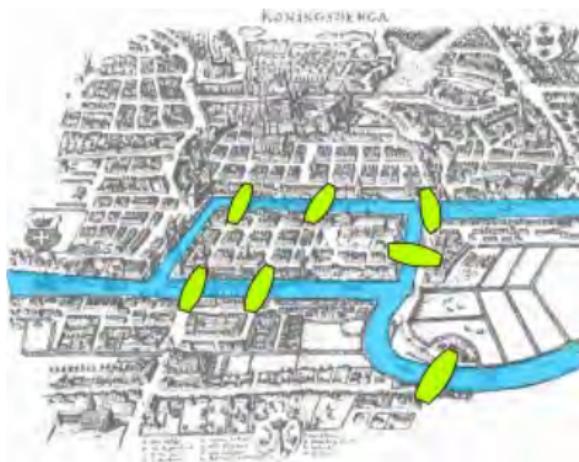
References & Credits, 2/2

- ▶ *Towards a Big Data reference architecture.* Markus Maier. MSc thesis, Eindhoven University of Technology, 2013.
http://www.win.tue.nl/~gfletche/Maier_MSc_thesis.pdf
- ▶ *Frontiers in massive data analysis.* National Research Council of the National Academies. Washington, D.C., 2013.
http://www.nap.edu/catalog.php?record_id=18374 (register for free PDF)
- ▶ *On being a data skeptic.* Cathy O'Neil. O'Reilly Media, 2013.
http://cdn.oreillystatic.com/oreilly/radarreport/0636920032328/On_Being_a_Data_Skeptic.pdf
- ▶ *NoSQL distilled: a brief guide to the emerging world of polyglot persistence.* Pramod J. Sadalage and Martin Fowler. Addison-Wesley, 2013. <http://martinfowler.com/nosql.html>

graph data

Graphs (aka, Networks)

Euler (1735): the seven bridges of Königsberg (Kalinigrad)

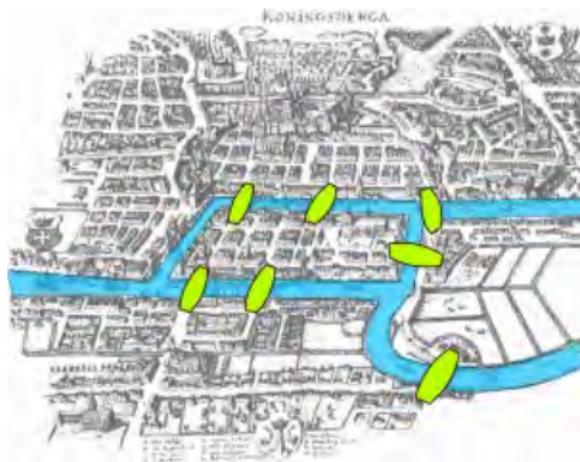


(wikipedia.org)

Can you walk through the city using each bridge exactly once?

Graphs (aka, Networks)

Euler (1735): the seven bridges of Königsberg (Kalinigrad)

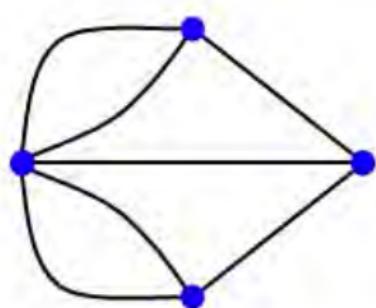


(wikipedia.org)

Can you walk through the city using each bridge exactly once?
Euler found that a solution does not exist. To prove this, he invented **graph theory**.

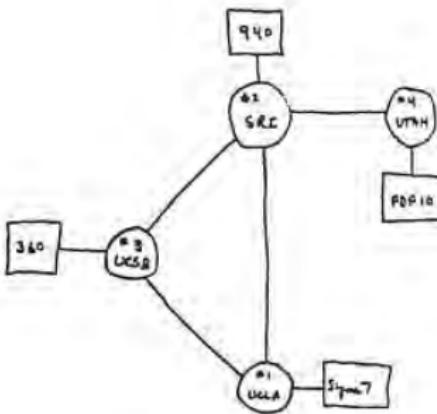
Graphs

Euler (1735): the seven bridges of Königsberg (Kaliningrad)



(wikipedia.org)

Graphs



THE ARPA NETWORK

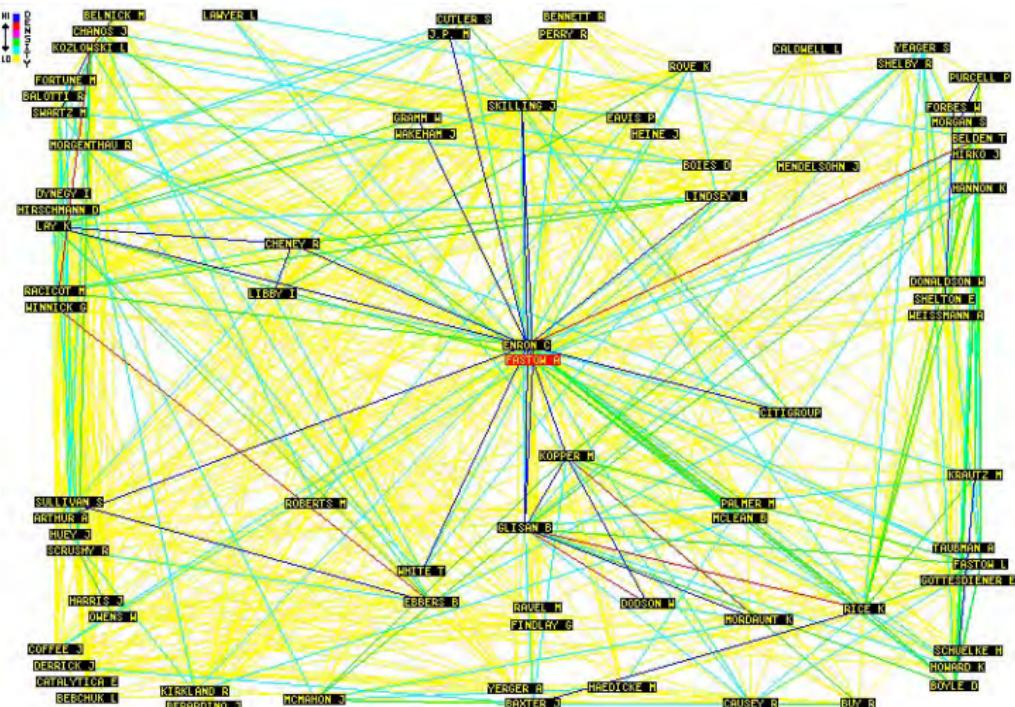
DEC 1969

4 NODES

FIGURE 6.2 Drawing of 4 Node Network
(Courtesy of Alex McKenzie)

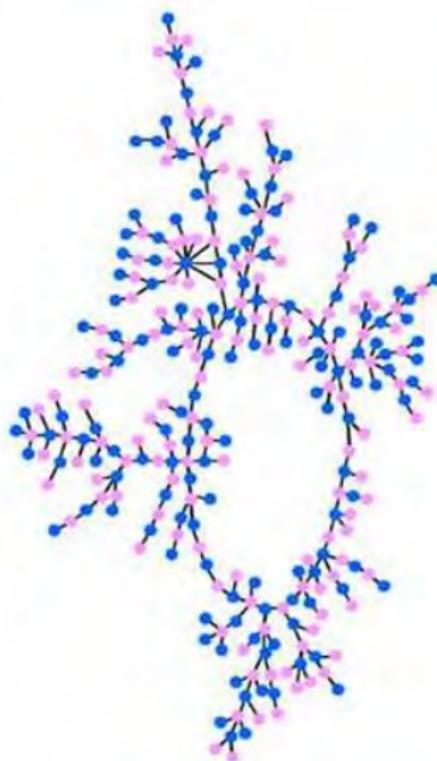
the Internet circa 1969 (Kearns, UPenn)

Graphs



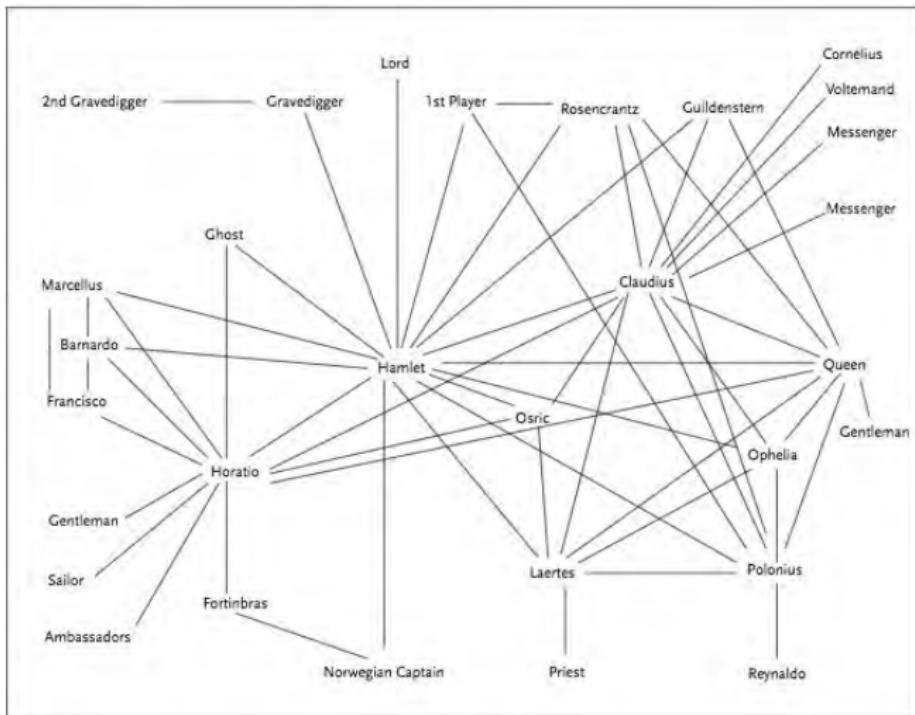
the Enron email network (Kearns, UPenn)

Graphs



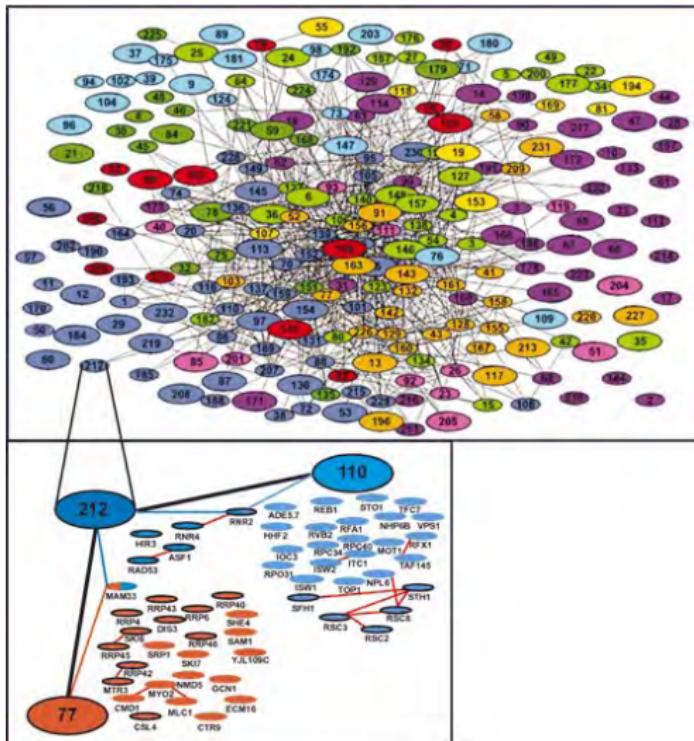
a highschool boyfriend-girlfriend network (Kearns, UPenn)

Graphs



the Hamlet social network (crookedtimber.org)

Graphs



a yeast protein network (genomenewsnetwork.org)

Web as a graph

The web as an object of study: a global macro-structure perspective on the web

- ▶ resources as nodes
- ▶ edges between nodes which link to each other

in contrast to the local perspective of one resource/document



(Ortega and Aguillo 2009)

Web as a graph

The web as an object of study

- ▶ the study of the **structure** of web and online social networks is a major theme of data/web science, e.g.,
 - ▶ connectivity and path length
 - ▶ degree of nodes

Web as a graph

The web as an object of study

- ▶ the study of the **structure** of web and online social networks is a major theme of data/web science, e.g.,
 - ▶ connectivity and path length
 - ▶ degree of nodes
- ▶ also, the study of **processes** on this structure is a major theme, e.g.,
 - ▶ searching
 - ▶ ranking, popularity
 - ▶ information dissemination
 - ▶ protection against destruction, attack
 - ▶ linked data query processing and optimization

Graphs

Many, many applications of graph theory

- ▶ physical, biological, social, chemical
- ▶ communication (e.g., phone)
- ▶ data/relationships (e.g., XML)
- ▶ travel, transportation
- ▶ chip design
- ▶ linguistics
- ▶ mathematics
- ▶ ...

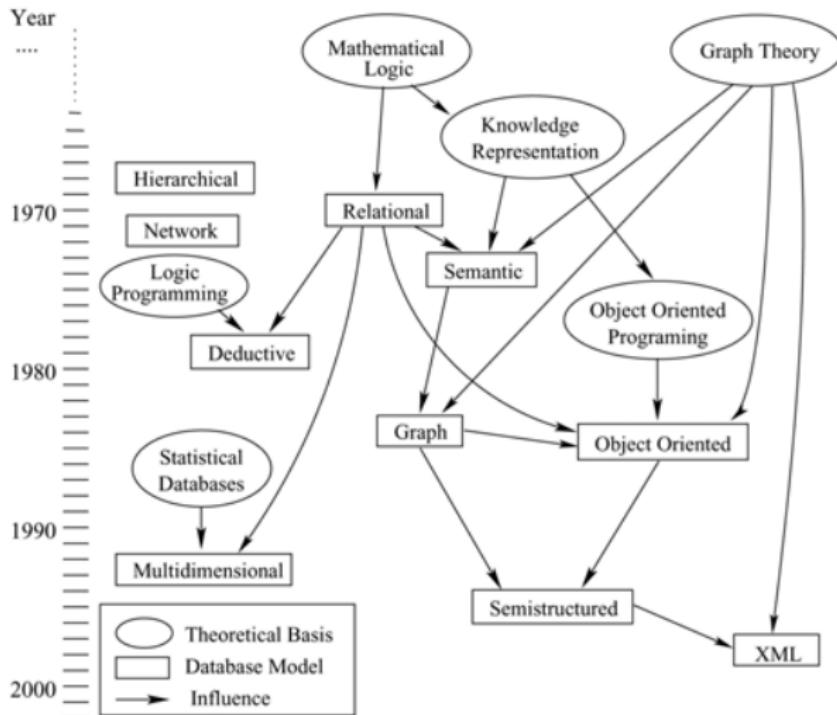


Things are interesting only in so far as they relate themselves to other things; only then can you put two and two together and tell stories about them. Such is science itself and such is all the knowledge that interests mankind.

– D'Arcy Wentworth Thompson
(1860-1948)

graph database models

Historical context: database models



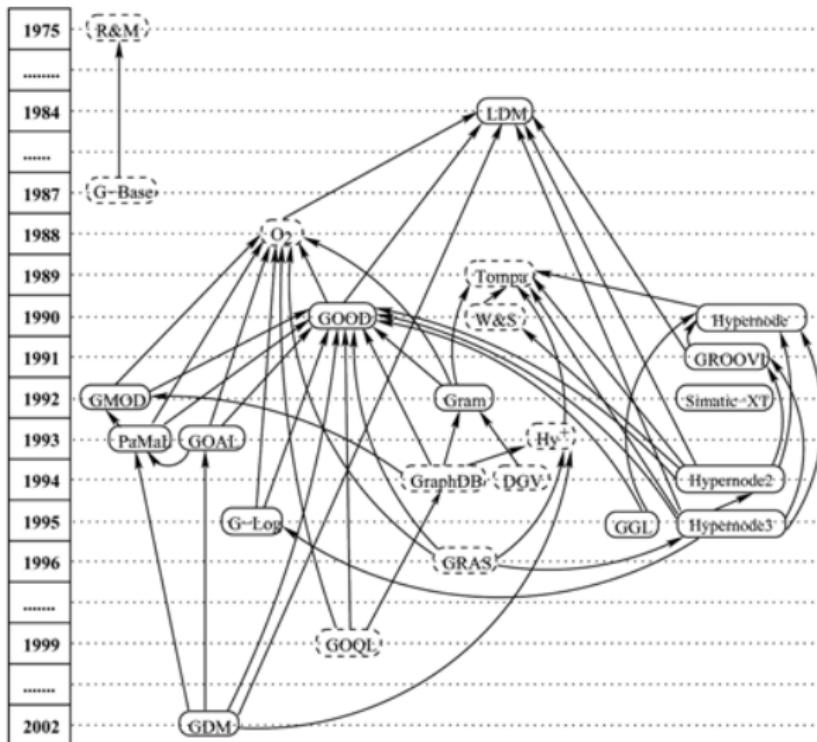
(We follow Angles and Gutierrez 2008 in this section)

What makes a database a graph database?

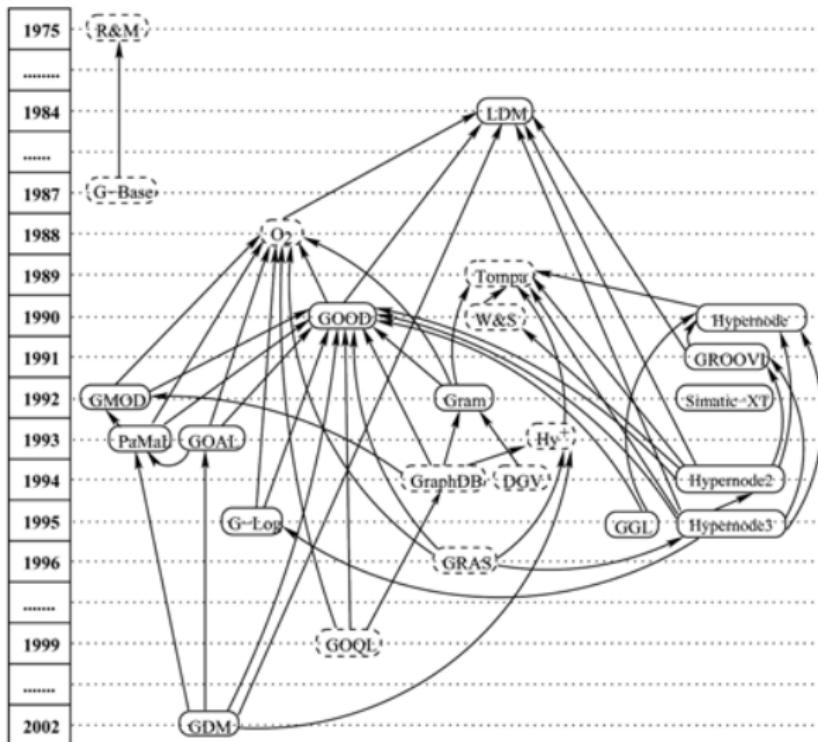
We can loosely identify three characteristics

1. Data, schemas, queries, and/or updates are represented by graphs, or their generalization.
2. Data manipulation is by graph transformations or operations on graph features such as paths, neighborhoods, diameter, etc.
3. There are possibly facilities for expressing appropriate integrity constraints such as label typing and domain/range restrictions.

Historical context: graph database models



Historical context: graph database models



As a representative sample, let's consider the thread $LDM \leftarrow GOOD \leftarrow GDM$

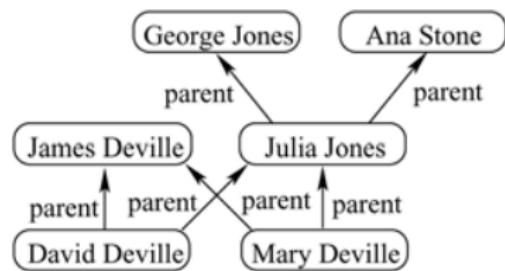
One thread: the Logical Data Model

LDM. (Kuper and Vardi, 1984)

- ▶ unifies relational, hierarchical, and network models
- ▶ provides constructs for physical addressing
- ▶ schemas and instances are node-labeled graphs
- ▶ three node types: basic (terminal nodes with atomic data), composition (tuples built from children), and collection (sets with elements taken from child)
- ▶ algebraic and logical querying

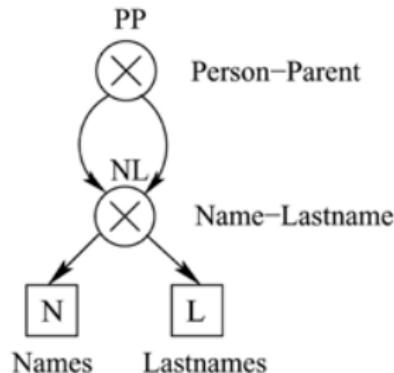
A graph ...

NAME	LASTNAME	PERSON	PARENT
George	Jones	Julia	George
Ana	Stone	Julia	Ana
Julia	Jones	David	James
James	Deville	David	Julia
David	Deville	Mary	James
Mary	Deville	Mary	Julia



One thread: LDM

Schema



Instance

$I(N)$		$I(L)$		$I(NL)$		$I(PP)$	
l	$val(l)$	l	$val(l)$	l	$val(l)$	l	$val(l)$
1	George	7	Jones	10	(1,7)	16	(12,10)
2	Ana	8	Stone	11	(2,8)	17	(12,11)
3	Julia	9	Deville	12	(3,7)	18	(14,13)
4	James			13	(4,9)	19	(14,12)
5	David			14	(5,9)	20	(15,13)
6	Mary			15	(6,9)	21	(15,12)

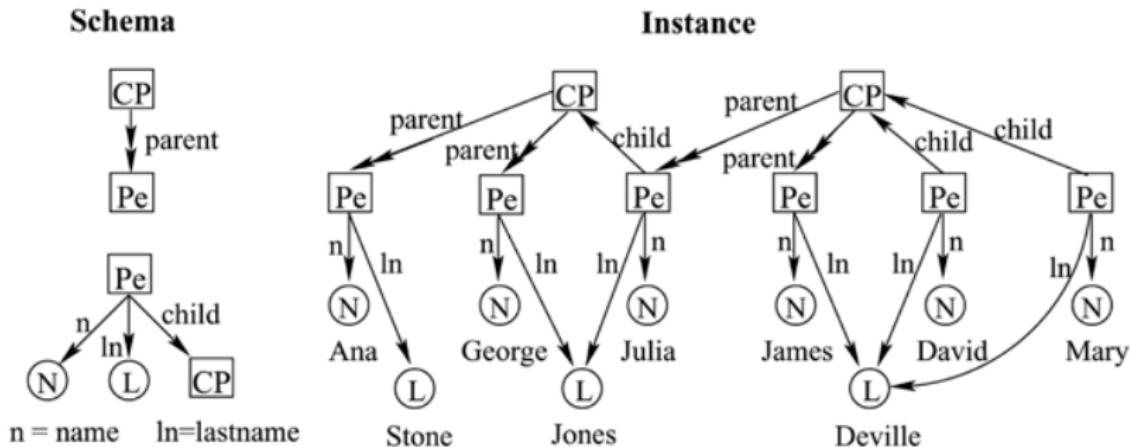
basic nodes (squares, labeled with atomic type), composition (circle with \times), collection (circle with $*$, not shown here)

One thread: the Graph-Oriented Object Database model

GOOD. (Gyssens, Paredaens, and Van Gucht, 1990)

- ▶ purely graph model, inspired and motivated by object databases
- ▶ schemas, instances, and queries are node- and edge-labeled graphs
- ▶ two node types: “printable” and “non-printable”

One thread: GOOD



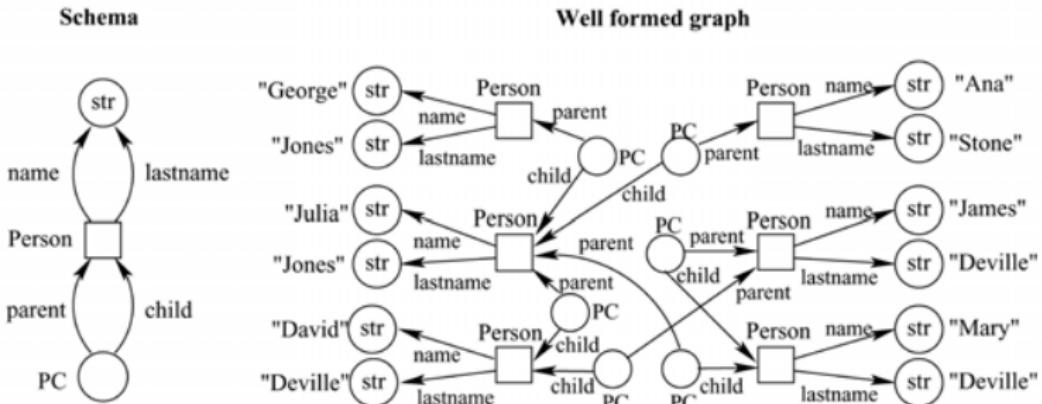
“printable” nodes (circle), “non-printable” nodes (square),
functional (i.e., having a unique value) edges (single arrow),
non-functional edges (double arrow)

One thread: the Graph Data Model

GDM. (Hidders 2002)

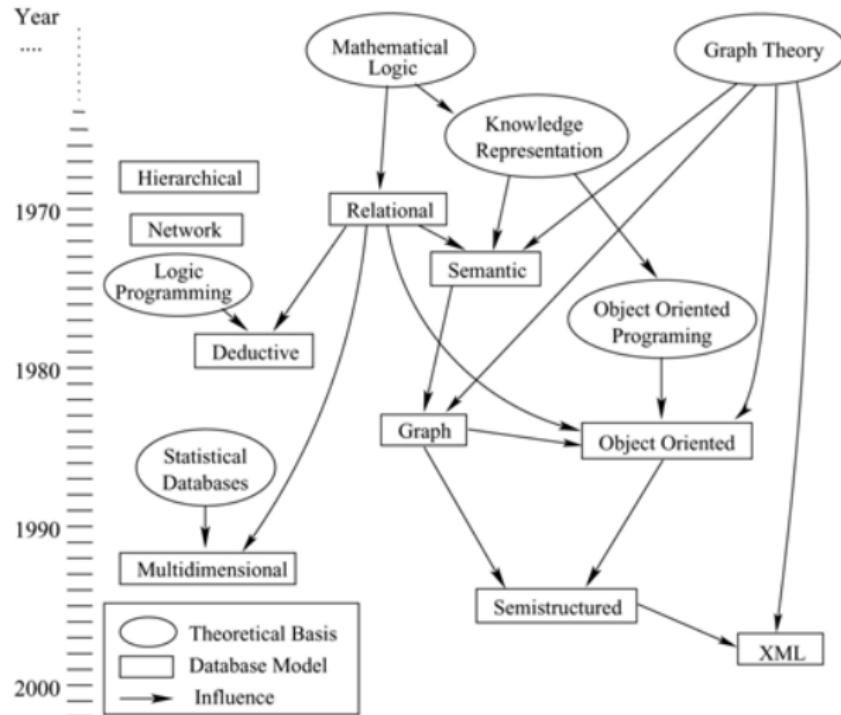
- ▶ builds on GOOD to include: complex objects, n-ary relations, inheritance, and other advanced modeling and querying features
- ▶ also includes a pattern-based update language (GUL)
- ▶ three node types: objects, composite-valued, basic-valued

One thread: GDM



object nodes (square, with class label(s)), composite-value nodes (round empty), basic value nodes (round with type label)

Related models: OO, OEM, and XML data



Related models: OO, OEM, XML, ...

Object exchange model (OEM) and object-oriented (OO) data

- ▶ OO: flurry of research and industry activity in the 80's and 90's, much work wrapped up into SQL standard
- ▶ OEM: self-describing, "semi-structured" model with nesting and node-identity

Related models: OO, OEM, XML, ...

Object exchange model (OEM) and object-oriented (OO) data

- ▶ OO: flurry of research and industry activity in the 80's and 90's, much work wrapped up into SQL standard
- ▶ OEM: self-describing, "semi-structured" model with nesting and node-identity

Extensible Markup Language (XML)

- ▶ ordered tree structured data exchange format, with OEM-like identity and references, also self describing

Related models: OO, OEM, XML, ...

Object exchange model (OEM) and object-oriented (OO) data

- ▶ OO: flurry of research and industry activity in the 80's and 90's, much work wrapped up into SQL standard
- ▶ OEM: self-describing, "semi-structured" model with nesting and node-identity

Extensible Markup Language (XML)

- ▶ ordered tree structured data exchange format, with OEM-like identity and references, also self describing

Resource Description Framework (RDF) ...

RDF

Towards triples

A data model for web data?

Some major problems with classical data models

- ▶ evolving schemas
- ▶ data heterogeneity

Lessons learned in the past decades:

- ▶ metadata is data
- ▶ it's all about relationships

Design for change

Towards triples

Basic requirements:

1. flexibly capture “things” and their relationships
2. don’t force artificial data/metadata distinctions
3. support full spectrum between structured and unstructured data

The RDF data model

Resource description framework (RDF)

- ▶ W3C recommendation data model for (semantic) web data
- ▶ Graph-based data model, embodying lessons learned from previous data models
- ▶ Uses web identifiers (URIs) to identify resources ("things")
- ▶ Uses triples to state relationships between things
- ▶ serialization formats: N-triples, N3, RDF/XML

... an old idea (circa 1870)



Charles S. Peirce
(1839-1914)

DESCRIPTION

OF A

NOTATION FOR THE LOGIC OF RELATIVES,

RESULTING FROM AN AMPLIFICATION OF THE
CONCEPTIONS OF

BOOLE'S CALCULUS OF LOGIC.

Charles S. Peirce.
By C. S. PEIRCE.

EXTRACTED FROM THE MUSEUM OF THE AMERICAN ACADEMY, VOL. IX.

CAMBRIDGE:
WELCH, BIGELOW, AND COMPANY,
PRINTERS TO THE UNIVERSITY.
1870.

The RDF data model: triples

Relationships: Triples

- ▶ Basic data structure
- ▶ has form (subject, predicate, object)
- ▶ “subject *has relationship predicate to object*”

The RDF data model: triples

Relationships: Triples

- ▶ Basic data structure
- ▶ has form (subject, predicate, object)
- ▶ “subject *has relationship predicate to object*”

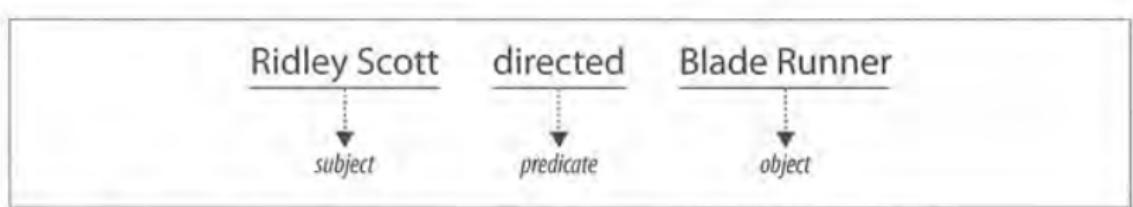


Figure 2-1. Sentence diagram showing a subject-predicate-object relationship

The RDF data model: triples

Relationships: Triples

- ▶ Basic data structure
- ▶ has form (subject, predicate, object)
- ▶ “subject *has relationship predicate to object*”

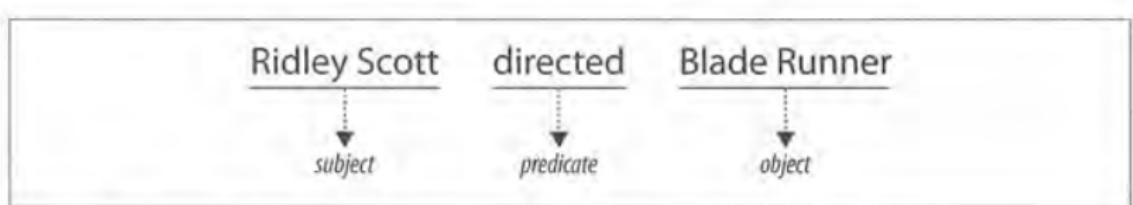


Figure 2-1. Sentence diagram showing a subject-predicate-object relationship

An **RDF graph** is a finite set of triples.

Sources of RDF data

As part of the LOD vision, a wide variety of data becoming available in RDF:

- ▶ Friend of a friend
- ▶ Wikipedia
- ▶ Uniprot gene database
- ▶ Bio2RDF
- ▶ BBC, New York Times
- ▶ open government data (UK, Australia, US, NL, Japan, ...)
- ▶ ...

Richer data modeling

RDF allows us to easily express facts

- ▶ John is the father of Mary
 $\{(john, fatherOf, mary)\}$
- ▶ Geoffrey knows that John is the father of Mary
 $\{(geoff, knows, S), (S, subject, john), (S, predicate, fatherOf), (S, object, mary)\}$

Richer data modeling

RDF allows us to easily express facts

- ▶ John is the father of Mary
 $\{(john, \text{fatherOf}, mary)\}$
- ▶ Geoffrey knows that John is the father of Mary
 $\{(geoff, \text{knows}, S), (S, \text{subject}, john), (S, \text{predicate}, \text{fatherOf}), (S, \text{object}, mary)\}$

But, we'd like to be able to express more generic knowledge

- ▶ All fathers are male
- ▶ If a person has a daughter, then they are a parent

This kind of knowledge is often referred to as *schematic*,
ontological, or *terminological* knowledge

RDF Schema

RDFS

- ▶ part of the W3C recommendation for RDF, for schema knowledge
- ▶ uses a vocabulary, with pre-defined semantics

RDF Schema

RDFS

- ▶ part of the W3C recommendation for RDF, for schema knowledge
- ▶ uses a vocabulary, with pre-defined semantics
- ▶ every RDFS document is an RDF document
 - ▶ **models are data:** data and “meta” data live side by side. RDFS triples are just data with no intrinsic behavior, other than that provided by associated semantics of keywords

RDF Schema

RDFS

- ▶ part of the W3C recommendation for RDF, for schema knowledge
- ▶ uses a vocabulary, with pre-defined semantics
- ▶ every RDFS document is an RDF document
 - ▶ **models are data:** data and “meta” data live side by side.
RDFS triples are just data with no intrinsic behavior, other than that provided by associated semantics of keywords
- ▶ vocabulary is generic, not bound to a specific application area
 - ▶ allows to specify semantics of user-defined vocabularies (its a kind of meta vocabulary)
 - ▶ implementations are responsible for interpreting the vocabulary defined using RDF Schema

RDF Schema: classes

Classes

- ▶ classes stand for **sets of things** (i.e., sets of URIs)

RDF Schema: classes

Classes

- ▶ classes stand for **sets of things** (i.e., sets of URIs)
- ▶ “book:uri is a member of the class ex:textbook”
(book:uri, rdf:type, ex:textbook)

RDF Schema: classes

Classes

- ▶ classes stand for **sets of things** (i.e., sets of URIs)
- ▶ “book:uri is a member of the class ex:textbook”
(book:uri, rdf:type, ex:textbook)
- ▶ a URI can belong to several classes
(book:uri, rdf:type, ex:textbook)
(book:uri, rdf:type, ex:popularbook)

RDF Schema: classes

Classes

- ▶ classes stand for **sets of things** (i.e., sets of URIs)
- ▶ “book:uri is a member of the class ex:textbook”
(book:uri, rdf:type, ex:textbook)
- ▶ a URI can belong to several classes
(book:uri, rdf:type, ex:textbook)
(book:uri, rdf:type, ex:popularbook)
- ▶ classes can be arranged in hierarchies: “each textbook is a book”
(ex:textbook, rdfs:subClassOf, ex:book)

RDF Schema: properties

An RDF property is a relation between subject resources and object resources

(ex:John, ex:isMarriedTo, ex:Sally)
(ex:isMarriedTo, rdf:type, rdf:Property)

RDF Schema: properties

An RDF property is a relation between subject resources and object resources

(ex:John, ex:isMarriedTo, ex:Sally)
(ex:isMarriedTo, rdf:type, rdf:Property)

property restrictions

- ▶ Allows us to state that a certain property can only be between things of a certain rdf:type
 - ▶ E.g. when a is married to b , then both a and b are Persons
- ▶ Expressed by rdfs:domain and rdfs:range, which are instances of rdf:Property

(ex:isMarriedTo, rdfs:domain, ex:Person)
(ex:isMarriedTo, rdfs:range, ex:Person)

References & Credits

- ▶ Towards well-behaved schema evolution. Rada Chirkova and George H. L. Fletcher. *WebDB*, Providence, Rhode Island, 2009.
<http://www.win.tue.nl/~gfletche/papers-final/ChirkovaFletcherWebDB09.pdf>
- ▶ Foundations of semantic web databases. Claudio Gutierrez, Carlos A. Hurtado, Alberto O. Mendelzon, and Jorge Pérez. *J. Comput. Syst. Sci.* 77(3): 520-541, 2011. <http://users.dcc.uchile.cl/~cgutierrez/papers/ghmp10.pdf>
- ▶ *Linked data: evolving the web into a global data space*. Tom Heath and Christian Bizer. Synthesis Lectures on the Semantic Web, Morgan & Claypool, 2011. <http://linkeddatabook.com/editions/1.0/>
- ▶ *Foundations of semantic web technologies*. Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. Chapman & Hall/CRC, 2009.
http://www.semantic-web-book.org/page/Foundations_of_Semantic_Web_Technologies
- ▶ *Programming the semantic web*. Toby Segaran, Colin Evans, and Jamie Taylor. O'Reilly Media, 2009.

graph queries

Graphs

For simplicity, we consider next navigating over directed graphs whose edges are labeled by symbols from a finite, nonempty set of labels Λ .

Formally, then, a **graph** is a relational structure G , consisting of

- ▶ a set of nodes N and,
- ▶ for every $R \in \Lambda$, a relation $G(R) \subseteq N \times N$, the set of edges with label R .

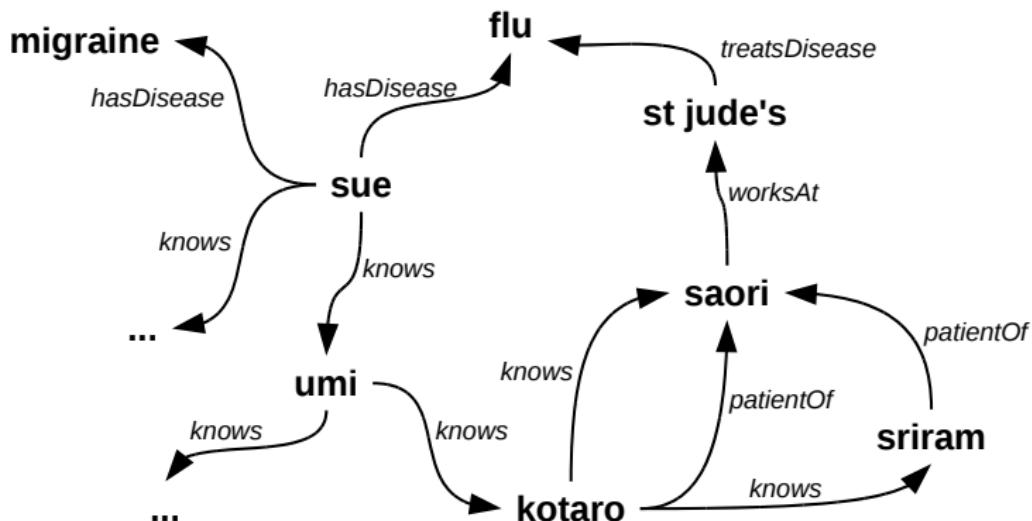
Graphs

For example, suppose our domain is **people**, **hospitals**, and **diseases**, and we have relationships (i.e., edge labels)

$$\Lambda = \{\text{knows}, \text{worksAt}, \text{patientOf}, \text{hasDisease}, \text{treatsDisease}\}.$$

Graphs

A small fragment of such a graph



Query language capabilities

Graph query languages typically feature one or both of the following basic capabilities

- ▶ subgraph matching
- ▶ finding nodes connected by paths

and possibly one or more advanced features such as *approximate matching* and *comparing paths*

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

Essentially, this consists of conjunctive queries on graphs

- ▶ an **edge pattern** is a triple (n_1, ℓ, n_2) where n_1 and n_2 can be either constants $n \in N$ or variables, and $\ell \in \Lambda$

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

Essentially, this consists of conjunctive queries on graphs

- ▶ an **edge pattern** is a triple (n_1, ℓ, n_2) where n_1 and n_2 can be either constants $n \in N$ or variables, and $\ell \in \Lambda$
- ▶ a **query** is then a pattern $head \leftarrow body$ where $head$ and $body$ are sets of edge patterns such that every variable occurring in $head$ occurs in $body$
 - ▶ alternatively, $head$ is a list of zero or more of the variables (possibly with repetition) appearing in $body$

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

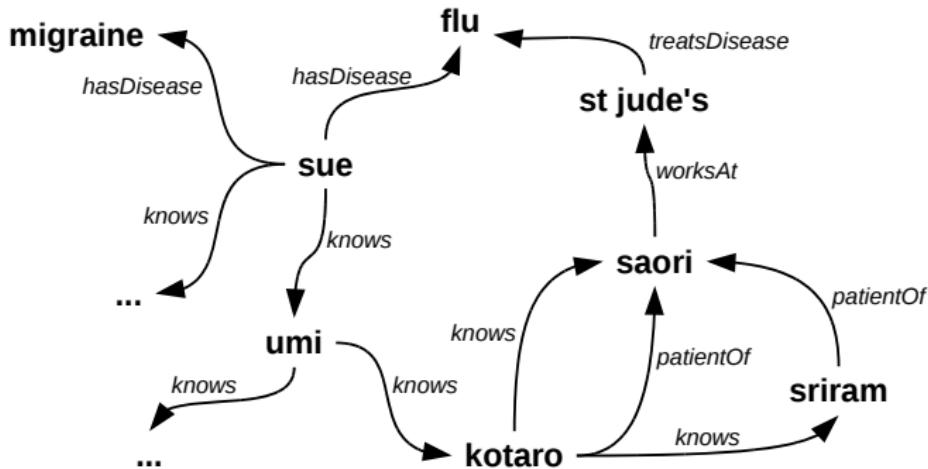
Essentially, this consists of conjunctive queries on graphs

- ▶ an **edge pattern** is a triple (n_1, ℓ, n_2) where n_1 and n_2 can be either constants $n \in N$ or variables, and $\ell \in \Lambda$
- ▶ a **query** is then a pattern $head \leftarrow body$ where $head$ and $body$ are sets of edge patterns such that every variable occurring in $head$ occurs in $body$
 - ▶ alternatively, $head$ is a list of zero or more of the variables (possibly with repetition) appearing in $body$
- ▶ the semantics $Q(G)$ of evaluating query Q on graph G is based on embeddings of $body$ in G

$$Q(G) = \{h(head) \mid h(body) \subseteq G\}$$

where h is a **homomorphism**, i.e., a function with domain $N \cup Variables$ and range N that is the identity on N

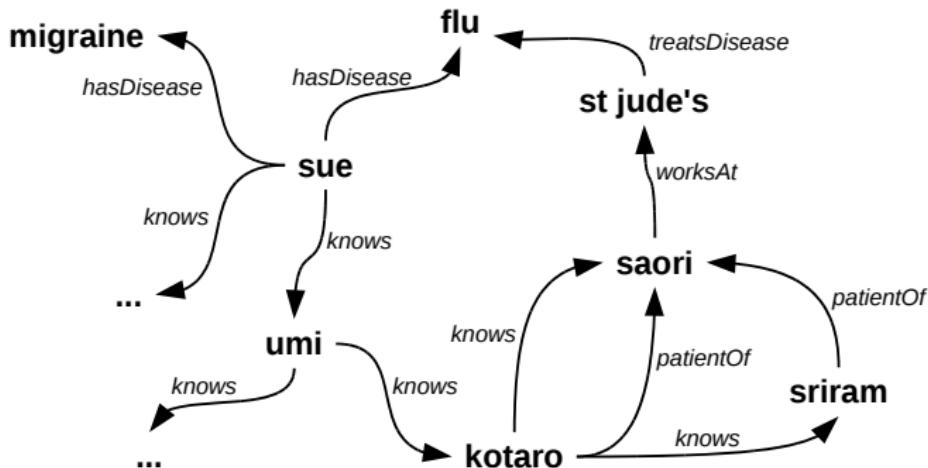
Subgraph matching



Example: People and the doctors of their friends

$$\begin{aligned} Q &= (?p, \text{friendDoctor}, ?d) \leftarrow (?p, \text{knows}, ?f), (?f, \text{patientOf}, ?d) \\ Q(G) &= \{(umi, \text{friendDoctor}, saori), (kotaro, \text{friendDoctor}, saori), \dots\} \end{aligned}$$

Subgraph matching



Example: People who know someone who knows a doctor.

$$Q = \langle ?p \rangle \leftarrow (?p, \text{knows}, ?f), (?f, \text{knows}, ?d), (?po, \text{patientOf}, ?d)$$

$$Q(G) = \{\langle \text{umi} \rangle, \dots\}$$

Subgraph matching

Recall that the combined complexity of conjunctive queries is NP-complete. This is due to the homomorphic semantics of query evaluation.

Hence, relaxations have been proposed recently based on a *simulation-based* semantics for subgraph matching (Fan et al. 2012, 2013)

- ▶ quadratic complexity (instead of intractable)

Path matching

- Regular path queries return all paths (i.e., pairs of nodes) connected by some regular expression over edge labels
- ▶ i.e., queries of the form

$$\langle ?x, ?y \rangle \leftarrow (?x, r, ?y)$$

where r is a regular expression over Λ

Path matching

Regular path queries return all paths (i.e., pairs of nodes) connected by some regular expression over edge labels

- ▶ i.e., queries of the form

$$\langle ?x, ?y \rangle \leftarrow (?x, r, ?y)$$

where r is a regular expression over Λ

- ▶ for example, the “knowing” social network is

$$\langle ?x, ?y \rangle \leftarrow (?x, \text{knows}^+, ?y)$$

and the general social network is

$$\langle ?x, ?y \rangle \leftarrow (?x, (\text{knows} \cup \text{patientOf})^+, ?y)$$

Path matching

Regular path queries return all paths (i.e., pairs of nodes) connected by some regular expression over edge labels

- ▶ i.e., queries of the form

$$\langle ?x, ?y \rangle \leftarrow (?x, r, ?y)$$

where r is a regular expression over Λ

- ▶ for example, the “knowing” social network is

$$\langle ?x, ?y \rangle \leftarrow (?x, \text{knows}^+, ?y)$$

and the general social network is

$$\langle ?x, ?y \rangle \leftarrow (?x, (\text{knows} \cup \text{patientOf})^+, ?y)$$

- ▶ polynomial time complexity
- ▶ many variations, such as Conjunctive RPQs and Extended CRPQs, have been intensively studied

References & Credits

- ▶ Survey of graph database models. Renzo Angles and Claudio Gutiérrez. *ACM Comput. Surv.* 40(1), 2008. <http://users.dcc.uchile.cl/~cgutierrez/papers/surveyGDB.pdf>
- ▶ Querying graph databases. Pablo Barceló Baeza. *PODS 2013*.
<http://users.dcc.uchile.cl/~pbarcelo/pods001i-barcelo.pdf>
- ▶ Graph pattern matching revised for social network analysis. Wenfei Fan. *ICDT 2012*. <http://homepages.inf.ed.ac.uk/wenfei/qsx/reading/icdt12.pdf>
- ▶ Incremental graph pattern matching. Wenfei Fan, Xin Wang, and Yinghui Wu. *ACM Trans. Database Syst.* 38(3):18, 2013.
<http://tods.acm.org/accepted/2013/FanIncremental.pdf>
- ▶ Query languages for graph databases. Peter T. Wood. *SIGMOD Record* 41(1):50-60, 2012.
<http://www.sigmod.org/publications/sigmod-record/1203/pdfs/08.principles.wood.pdf>

SPARQL

SPARQL

- ▶ SPARQL: SPARQL Protocol And RDF Query Language
 - ▶ Query language for data from RDF documents
 - ▶ W3C specification since January 2008
 - ▶ Extremely successful in practice
 - ▶ Key technology in the Linked Data framework

SPARQL

- ▶ SPARQL: SPARQL Protocol And RDF Query Language
 - ▶ Query language for data from RDF documents
 - ▶ W3C specification since January 2008
 - ▶ Extremely successful in practice
 - ▶ Key technology in the Linked Data framework
 - ▶ (Backwards compatible) revision finished in March 2013

SPARQL

- ▶ SPARQL: SPARQL Protocol And RDF Query Language
 - ▶ Query language for data from RDF documents
 - ▶ W3C specification since January 2008
 - ▶ Extremely successful in practice
 - ▶ Key technology in the Linked Data framework
 - ▶ (Backwards compatible) revision finished in March 2013
- ▶ Parts of the SPARQL specification:
 - ▶ Query language: our focus
 - ▶ Result format: encode results in XML
 - ▶ Query protocol: transmitting queries and results

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title      ?title .
  ?book ex:author     ?author . }
```

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title      ?title .
  ?book ex:author     ?author . }
```

- ▶ Main part is a query pattern (WHERE)
 - ▶ Patterns use RDF Turtle syntax
 - ▶ Variables can be used, even in predicate positions (?variable)

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title      ?title .
  ?book ex:author     ?author . }
```

- ▶ Main part is a query pattern (WHERE)
 - ▶ Patterns use RDF Turtle syntax
 - ▶ Variables can be used, even in predicate positions (?variable)
- ▶ Abbreviations for URIs (PREFIX)

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title      ?title .
  ?book ex:author     ?author . }
```

- ▶ Main part is a query pattern (WHERE)
 - ▶ Patterns use RDF Turtle syntax
 - ▶ Variables can be used, even in predicate positions (?variable)
- ▶ Abbreviations for URIs (PREFIX)
- ▶ Graph(s) to be queried (FROM)

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
?book ex:title      ?title .
?book ex:author     ?author . }
```

- ▶ Main part is a query pattern (WHERE)
 - ▶ Patterns use RDF Turtle syntax
 - ▶ Variables can be used, even in predicate positions (?variable)
- ▶ Abbreviations for URIs (PREFIX)
- ▶ Graph(s) to be queried (FROM)
- ▶ Query result based on selected variables (SELECT)
 - ▶ other query forms are CONSTRUCT (output new graph), ASK (boolean), and DESCRIBE (informative description)

SPARQL: Basic graph patterns

```
{<ex:book1 ex:title, "title 1">,
 <ex:book1, ex:publishedBy, http://elsevier.com>,
 <ex:book1, ex:author, johndoe>,
 <ex:book2, ex:publishedBy, http://elsevier.com>,
 <ex:book2, ex:title, "title 1">,
 <ex:book3, ex:publishedBy, http://springer.com>,
 <ex:book3, ex:author, janedoe>,
 <ex:book4, ex:publishedBy, http://elsevier.com>, ... }
```

Basic graph patterns (BGP) form the core of SPARQL (the WHERE clause). In essence, a BGP is a finite set of triples, wherein elements of the triples may be atoms or variables. The semantics of a BGP P with respect to a graph G is the set of all bindings β of the variables in P to atoms such that $\beta(P) \subseteq G$.

SPARQL: Basic graph patterns

```
{ <ex:book1 ex:title, "title 1">,
  <ex:book1, ex:publishedBy, http://elsevier.com>,
  <ex:book1, ex:author, johndoe>,
  <ex:book2, ex:publishedBy, http://elsevier.com>,
  <ex:book2, ex:title, "title 1">,
  <ex:book3, ex:publishedBy, http://springer.com>,
  <ex:book3, ex:author, janedoe>,
  <ex:book4, ex:publishedBy, http://elsevier.com>, ... }
```

For example, in

```
(?book, publishedBy, elsevier), (?book, title, ?title),
                           (?book, author, ?author)
```

variable $?book$ binds to all those Elsevier books with a known title and author. In the database G above, the only valid binding for $?book$ is “`ex:book1`”.

SPARQL: SELECT versus CONSTRUCT

The SELECT query form generates vectors of variable bindings

$$\langle ?title, ?author \rangle \leftarrow (?book, publishedBy, elsevier),
(?book, title, ?title), (?book, author, ?author)$$

SPARQL: SELECT versus CONSTRUCT

The SELECT query form generates vectors of variable bindings

$$\langle ?title, ?author \rangle \leftarrow (?book, publishedBy, elsevier),
(?book, title, ?title), (?book, author, ?author)$$

The CONSTRUCT query form, on the other hand, generates an RDF graph

$$(?book, rdf : type, fullBook) \leftarrow (?book, publishedBy, elsevier),
(?book, title, ?title), (?book, author, ?author)$$

SPARQL: SELECT versus CONSTRUCT

```
PREFIX ex: <http://example.org/>
CONSTRUCT { ?book rdf:type ex:fullBook }
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
?book ex:title ?title .
?book ex:author ?author . }
```

SPARQL

Further features

- ▶ OPTIONAL clauses (introduces null values)
- ▶ FILTER clauses, for specifying filter conditions
- ▶ UNION of BGPs
- ▶ Modifiers: postprocess query result set, e.g.: ORDER BY ?age LIMIT 10 OFFSET 5

SPARQL

Further features

- ▶ OPTIONAL clauses (introduces null values)
- ▶ FILTER clauses, for specifying filter conditions
- ▶ UNION of BGPs
- ▶ Modifiers: postprocess query result set, e.g.: ORDER BY ?age LIMIT 10 OFFSET 5

Further features (SPARQL 1.1)

- ▶ aggregates, negation, subqueries, expressions in SELECT, path recursion, federation
- ▶ we will explore these in the following

SPARQL 1.1

Property paths. Allows us to introduce RPQs into BGPs.

```
PREFIX ex: <http://example.org/>
SELECT ?name
FROM <http://example.org/example>
WHERE
{ ?book    ex:publishedBy <http://elsevier.com> .
  ?book    ex:title        ?title .
  ?book    ex:author       ?author .
  ?author  foaf:knows/foaf:name ?name
}
```

SPARQL 1.1

Property paths. Allows us to introduce RPQs into BGPs.

```
PREFIX ex: <http://example.org/>
SELECT ?name
FROM <http://example.org/example>
WHERE
{ ?book    ex:publishedBy <http://elsevier.com> .
  ?book    ex:title        ?title .
  ?book    ex:author       ?author .
  ?author  foaf:knows/foaf:knows/foaf:name ?name
}
```

SPARQL 1.1

Property paths. Allows us to introduce RPQs into BGPs.

```
PREFIX ex: <http://example.org/>
SELECT ?name
FROM <http://example.org/example>
WHERE
{ ?book    ex:publishedBy <http://elsevier.com> .
  ?book    ex:title        ?title .
  ?book    ex:author       ?author .
  ?author  foaf:knows+/foaf:name ?name
}
```

SPARQL 1.1

Federation. Allows for evaluating BGPs on non-local service points, via the SERVICE keyword.

```
PREFIX ex: <http://example.org/>
SELECT ?title ?phone
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title      ?title .
  ?book ex:author     ?author .
  SERVICE <http://people.example.org/sparql>
    { ?author foaf:phone ?phone . }
}
```

SPARQL 1.1

Federation. Note that SERVICE point can be dynamically retrieved

```
PREFIX ex: <http://example.org/>
SELECT ?title ?phone
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title ?title .
  ?book ex:author ?author .
  ?author ex:authority ?source .
  SERVICE ?source
  { ?author foaf:phone ?phone . }
}
```

SPARQL

SPARQL: Based on matching simple graph patterns

- ▶ simple SQL-like syntax
- ▶ flexible and expressive, for loosely structured RDF data
 - ▶ Grouping, optionals, and alternatives
 - ▶ Filters: extra-logical result restrictions
 - ▶ Variety of query forms
- ▶ key standard in linked data initiative
 - ▶ try out the DBpedia end point <http://live.dbpedia.org/sparql>

Basic graph pattern processing: in brief

$(?person, knows, mary), (?person, livesIn, ?city), (?city, locatedIn, Italy)$

Basic graph pattern processing: in brief

$(?person, knows, mary), (?person, livesIn, ?city), (?city, locatedIn, Italy)$

- ▶ each triple pattern evaluates in a graph G to the set of matching triples in G

$(?person, knows, mary) \Rightarrow_G \{(john, knows, mary), (sue, knows, mary), \dots\}$

Basic graph pattern processing: in brief

$(?person, knows, mary), (?person, livesIn, ?city), (?city, locatedIn, Italy)$

- ▶ each triple pattern evaluates in a graph G to the set of matching triples in G

$(?person, knows, mary) \Rightarrow_G \{(john, knows, mary), (sue, knows, mary), \dots\}$

- ▶ BGP is evaluated by *joining* these sets on common variables

$\{(john, knows, mary), (sue, knows, mary), \dots\}$

\bowtie

$\{(john, livesIn, Eindhoven), (sue, livesIn, Trento), \dots\}$

\bowtie

$\{(Rome, locatedIn, Italy), (Trento, locatedIn, Italy), \dots\}$

Storage in external memory: in brief

Two basic storage alternatives

1. flat file of triples, one per line
 - ▶ ... sorted on some subset of positions (e.g., by subject values)

Storage in external memory: in brief

Two basic storage alternatives

1. flat file of triples, one per line
 - ▶ ... sorted on some subset of positions (e.g., by subject values)
2. index data structure: maps a search key to a set of matching triples
 - ▶ B+tree: logarithmic lookup cost, sorted access
 - ▶ hash table: essentially constant lookup cost, point access

Storage: on the varieties of triple indexes

state of the art in the market: value-based indexing

- ▶ MAP (2007)
- ▶ HexTree (2008)
- ▶ TripleT (2009)

Storage: on the varieties of triple indexes

state of the art in the market: value-based indexing

- ▶ MAP (2007)
- ▶ HexTree (2008)
- ▶ TripleT (2009)

still in the research lab: structure-based indexing

- ▶ group triples both by the values they share and by the *structure* they share in the graph (e.g., similar neighborhood topology)
- ▶ structure-based indexing also successful in XML data management

Storage: on the varieties of (value-based) triple indexes

Let's focus on value-based indexes

For graph G , let

$$\begin{aligned}\mathcal{S}(G) &= \{s \mid (s, p, o) \in G\} \\ \mathcal{P}(G) &= \{p \mid (s, p, o) \in G\} \\ \mathcal{O}(G) &= \{o \mid (s, p, o) \in G\} \\ \mathcal{A}(G) &= \mathcal{S}(G) \cup \mathcal{P}(G) \cup \mathcal{O}(G)\end{aligned}$$

Storage: on the varieties of triple indexes

MAP: index all permutations of S, P, and O

$$SPO = \{s\#p\#o \mid (s, p, o) \in G\}$$

$$SOP = \{s\#o\#p \mid (s, p, o) \in G\}$$

$$PSO = \{p\#s\#o \mid (s, p, o) \in G\}$$

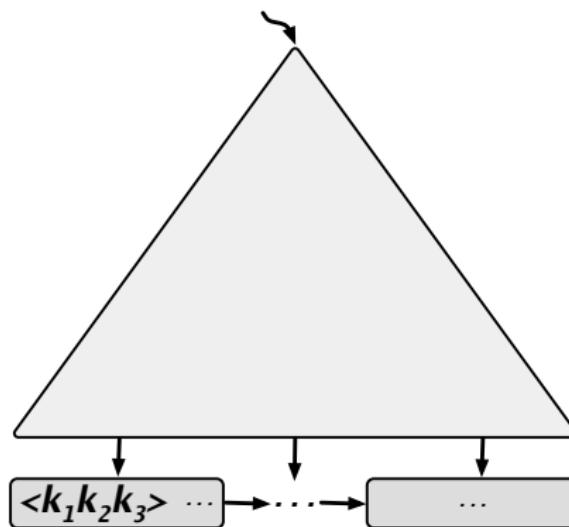
$$POS = \{p\#o\#s \mid (s, p, o) \in G\}$$

$$OSP = \{o\#s\#p \mid (s, p, o) \in G\}$$

$$OPS = \{o\#p\#s \mid (s, p, o) \in G\}$$

Storage: on the varieties of triple indexes

MAP: index all permutations of S, P, and O



... times six

Storage: on the varieties of triple indexes

MAP: index all permutations of S, P, and O

Most popular approach to triple storage in practice

- ▶ RDF-3X: open source triple store
 - ▶ <http://code.google.com/p/rdf3x/>
- ▶ Virtuoso: open source and commercial industrial-strength triple store
 - ▶ <http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/>
- ▶ Sesame/BigData: open source and commercial industrial-strength triple store
 - ▶ <http://www.openrdf.org/>
 - ▶ <http://www.systap.com/bigdata.htm>

Storage: on the varieties of triple indexes

HexTree: index all pairs of S, P, and O

$$SP = \{s\#p \mid (s, p, o) \in G\}$$

$$SO = \{s\#o \mid (s, p, o) \in G\}$$

$$PS = \{p\#s \mid (s, p, o) \in G\}$$

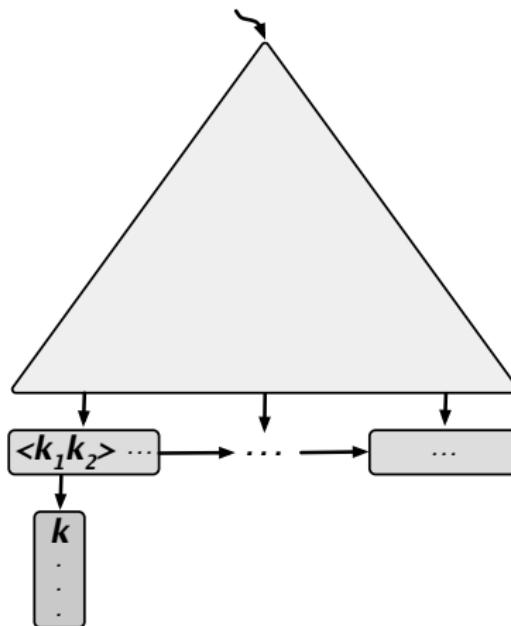
$$PO = \{p\#o \mid (s, p, o) \in G\}$$

$$OS = \{o\#s \mid (s, p, o) \in G\}$$

$$OP = \{o\#p \mid (s, p, o) \in G\}$$

Storage: on the varieties of triple indexes

HexTree: index all pairs of S, P, and O



... times six

Storage: on the varieties of triple indexes

HexTree: partners share payload

$$SP \rightarrow \{o \in \mathcal{O}(G) \mid (s, p, o) \in G\}$$

$$PS \rightarrow \{o \in \mathcal{O}(G) \mid (s, p, o) \in G\}$$

Storage: on the varieties of triple indexes

MAP and HexTree

- ▶ *strengths:* support all native joins expressible in the SPARQL WHERE clause, using fast “merge” joins, with no constraints on “schema”

Storage: on the varieties of triple indexes

MAP and HexTree

- ▶ *strengths*: support all native joins expressible in the SPARQL WHERE clause, using fast “merge” joins, with no constraints on “schema”
- ▶ *weaknesses*
 - ▶ lots of (redundant) storage space
 - ▶ access to multiple data structures to perform joins

Storage: on the varieties of triple indexes

MAP and HexTree

- ▶ *strengths*: support all native joins expressible in the SPARQL WHERE clause, using fast “merge” joins, with no constraints on “schema”
- ▶ *weaknesses*
 - ▶ lots of (redundant) storage space
 - ▶ access to multiple data structures to perform joins
 - ▶ in general, weak data locality

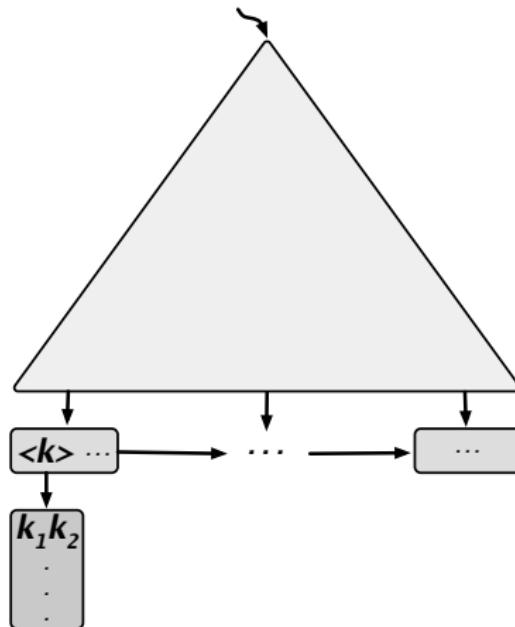
Storage: on the varieties of triple indexes

TripleT

- ▶ index $\mathcal{A}(G)$
- ▶ just one data structure – a three-way triple tree

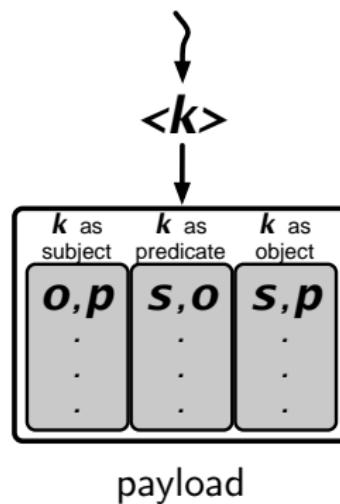
Storage: on the varieties of triple indexes

TripleT



Storage: on the varieties of triple indexes

TripleT



Storage: on the varieties of triple indexes

TripleT advantages

- ▶ retains strengths of MAP and HexTree

Storage: on the varieties of triple indexes

TripleT advantages

- ▶ retains strengths of MAP and HexTree
- ▶ reduced storage space
- ▶ reduced key size (i.e., shallower trees)

Storage: on the varieties of triple indexes

TripleT advantages

- ▶ retains strengths of MAP and HexTree
- ▶ reduced storage space
- ▶ reduced key size (i.e., shallower trees)
- ▶ single data structure to perform joins

Storage: on the varieties of triple indexes

TripleT advantages

- ▶ retains strengths of MAP and HexTree
- ▶ reduced storage space
- ▶ reduced key size (i.e., shallower trees)
- ▶ single data structure to perform joins
- ▶ in general, stronger data locality

Storage: on the varieties of triple indexes

TripleT advantages

- ▶ retains strengths of MAP and HexTree
- ▶ reduced storage space
- ▶ reduced key size (i.e., shallower trees)
- ▶ single data structure to perform joins
- ▶ in general, stronger data locality

Leads to significantly reduced storage and query processing costs

Storage: on the varieties of triple indexes

TripleT advantages

- ▶ retains strengths of MAP and HexTree
- ▶ reduced storage space
- ▶ reduced key size (i.e., shallower trees)
- ▶ single data structure to perform joins
- ▶ in general, stronger data locality

Leads to significantly reduced storage and query processing costs

idea appears in various guises in products

- ▶ open source TU/e implementation <https://github.com/b-w/TripleT>
- ▶ open source 4store engine <http://4store.org>
- ▶ and storage of RDF in so-called column stores
 - ▶ open source MonetDB <http://www.monetdb.org/>
 - ▶ commercial Vertica <http://www.vertica.com/>

Storage: on the varieties of triple indexes

Also, relational DB-based RDF solutions are available in industrial-strength IBM DB2 and Oracle Database 11G products.

See:

- ▶ <http://www.ibm.com/developerworks/data/tutorials/dm-1205rdfdb210/index.html>
- ▶ <http://www.oracle.com/technetwork/database/options/semantic-tech/index.html>

Both use various hybrids of the MAP, HexTree, and TripleT indexes

References & Credits

- ▶ An extensible framework for query optimization on TripleT-based RDF stores. Bart Wolff, George Fletcher, James Lu. LWDM 2015, Brussels.
<http://www.win.tue.nl/~gfletche/papers-final/wolff-camera.pdf>
- ▶ Querying semantic data on the web. Marcelo Arenas, Claudio Gutierrez, Daniel P. Miranker, Jorge Perez, and Juan Sequeda. *SIGMOD Record* 41(4): 6-17, 2012.
<http://www.sigmod.org/publications/sigmod-record/1212/pdfs/03.principles.arenas.pdf>
- ▶ *Linked data: evolving the web into a global data space.* Tom Heath and Christian Bizer. Synthesis Lectures on the Semantic Web, Morgan & Claypool, 2011.
<http://linkeddatabook.com/editions/1.0/>

Models and queries for linked data

Formal models for querying the web of data

What does it even mean to query the web?

Formal models for querying the web of data

What does it even mean to query the web?

There have been (relatively) few investigations which attempt directly address this question

- ▶ Mendelzon and Milo (1998)
- ▶ Abiteboul and Vianu (2000)
- ▶ Bouquet, Ghidini, and Serafini (2010)
- ▶ Hartig and Freytag (2012)

We next take a brief look at each of these major efforts, each of which help us demarcate the borders of practical LOD query processing solutions

(MM) The [web of data](#) is modeled as a *finite* virtual database with schema

$$\begin{aligned} & \textit{Node}(id, \overline{A_N}), \\ & \textit{Link}(source, destination, \overline{A_L}), \\ & R_1(\overline{A_1}), \dots, R_n(\overline{A_n}) \end{aligned}$$

such that (1) *id* is a key for *Node*, (2) the inclusion dependency $\pi_{source}(\textit{Link}) \subseteq \pi_{id}(\textit{Node})$ holds; and, (3) the R_i 's are normal arbitrary relational schemas, which are materialized locally.

The *id* field of *Node* and the *source, destination* fields of *Link* contain URIs.

The other fields contain node content and edge/node labels, etc.

The *Node* and *Link* tables are never fully available to the user.
The only way to access *Node* is by specifying a URI. The only way
to access *Link* is by specifying a source URI.

The *Node* and *Link* tables are never fully available to the user. The only way to access *Node* is by specifying a URI. The only way to access *Link* is by specifying a source URI.

- ▶ some “seed” nodes may be stored in one of the R'_i , e.g., as bookmarks or results from a search engine
- ▶ Since URI's are recursively enumerable, users can also try to “guess” valid Node ID's (but never know when to stop ...)

The *Node* and *Link* tables are never fully available to the user.
The only way to access *Node* is by specifying a URI. The only way to access *Link* is by specifying a source URI.

- ▶ some “seed” nodes may be stored in one of the R'_i , e.g., as bookmarks or results from a search engine
- ▶ Since URI's are recursively enumerable, users can also try to “guess” valid Node ID's (but never know when to stop ...)

Hence, access to the web of data is by **navigation** from known nodes to new nodes, following *Links*

Mendelzon and Milo

The query model is arbitrary computable queries, with a
Node/Link oracle

Mendelzon and Milo

The query model is arbitrary computable queries, with a *Node/Link* oracle

MM establish that the following queries are **computable**:

- a. Find all nodes reachable from the node with URI u
- b. Find all nodes on a cycle of length at most 3 involving node with URI u

while the following are **not computable**:

- c. Find all nodes
- d. Find all nodes with no incoming links
- e. Find all nodes referencing the node with URI u (though **eventually** computable)

Mendelzon and Milo

The query model is arbitrary computable queries, with a *Node/Link* oracle

MM establish that the following queries are **computable**:

- a. Find all nodes reachable from the node with URI u
- b. Find all nodes on a cycle of length at most 3 involving node with URI u

while the following are **not computable**:

- c. Find all nodes
- d. Find all nodes with no incoming links
- e. Find all nodes referencing the node with URI u (though **eventually** computable)

MM also study the subtle impact of updates on computability (e.g., (a.) becomes “eventually” computable, depending on how fast the query engine is ...)

Abiteboul and Vianu

(AV) The [web of data](#) is modeled as Mendelzon and Milo except with the crucial difference that instances are [infinite](#).

We believe this captures the intuition that exhaustive exploration of the Web is – or will soon become – prohibitively expensive ... our model draws a sharp distinction between exhaustive exploration of the Web and more controlled types of computation. (AV)

Abiteboul and Vianu

(AV) The [web of data](#) is modeled as Mendelzon and Milo except with the crucial difference that instances are [infinite](#).

We believe this captures the intuition that exhaustive exploration of the Web is – or will soon become – prohibitively expensive ... our model draws a sharp distinction between exhaustive exploration of the Web and more controlled types of computation. (AV)

Hence, “find all nodes reachable from the node with URI u ” moves from being computable to being [eventually](#) computable ...

Abiteboul and Vianu

(AV) The [web of data](#) is modeled as Mendelzon and Milo except with the crucial difference that instances are [infinite](#).

We believe this captures the intuition that exhaustive exploration of the Web is – or will soon become – prohibitively expensive ... our model draws a sharp distinction between exhaustive exploration of the Web and more controlled types of computation. (AV)

Hence, “find all nodes reachable from the node with URI u ” moves from being computable to being [eventually](#) computable ...

AV study various weaker query languages as well, showing, e.g., that all positive FO queries are eventually computable, even if recursion is introduced.

Abiteboul and Vianu

(AV) The [web of data](#) is modeled as Mendelzon and Milo except with the crucial difference that instances are [infinite](#).

We believe this captures the intuition that exhaustive exploration of the Web is – or will soon become – prohibitively expensive ... our model draws a sharp distinction between exhaustive exploration of the Web and more controlled types of computation. (AV)

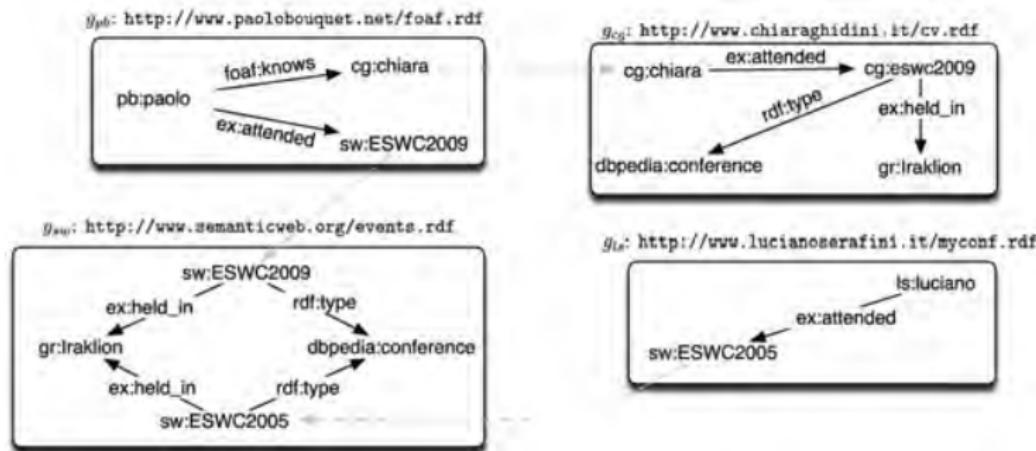
Hence, “find all nodes reachable from the node with URI u ” moves from being computable to being [eventually](#) computable ...

AV study various weaker query languages as well, showing, e.g., that all positive FO queries are eventually computable, even if recursion is introduced.

No study of updates has been made in this model, although it has been extended by Spielmann et al. PODS 2002, to study efficient distributed processing of queries

Bouquet, Ghidini, and Serafini

BGS give formal semantics for two models of query processing on a *finite web of data*, using the RDF standard. BGS do not study the impact of updates.

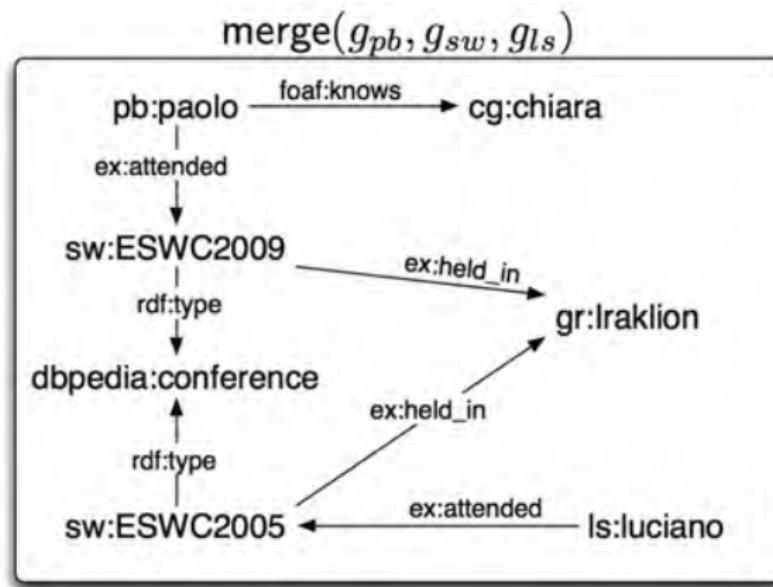


four LOD sources g_{pb} , g_{ls} , g_{cg} , and g_{sw}

In particular, URI's resolve to finite sets of triples. Then, query evaluation is over the “merge” (i.e., union, with BNode renaming) of a set of graphs identified by the model.

Bouquet, Ghidini, and Serafini

In the direct access query model, queries are evaluated over the merge of graphs resolved from some known finite set of URI's.

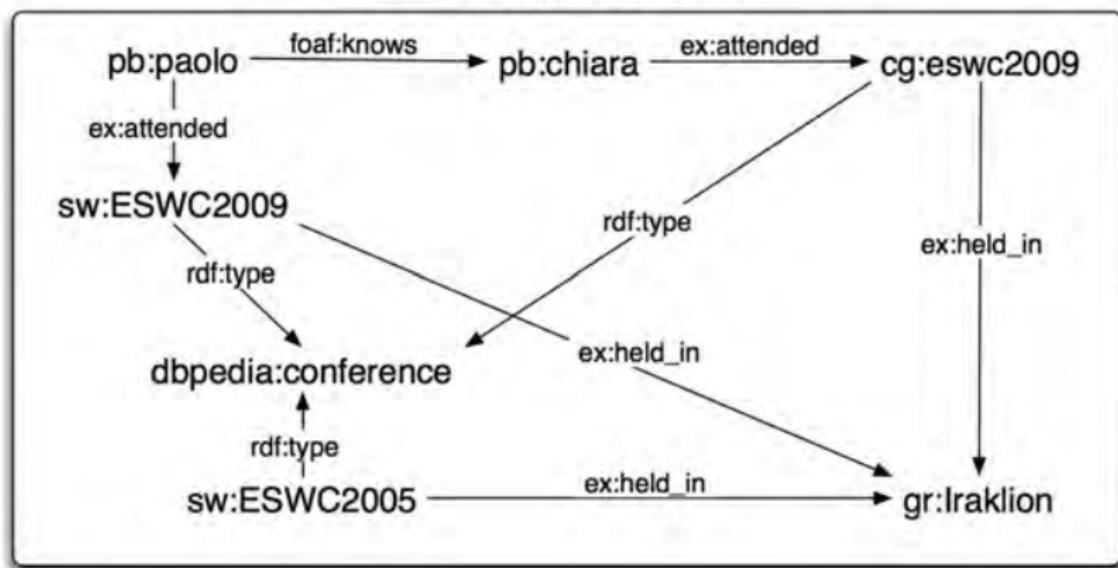


direct access on g_{pb} , g_{ls} , and g_{sw}

Bouquet, Ghidini, and Serafini

In the **navigational** query model, queries are evaluated over the merge of graphs resolved by following all outgoing links, starting from some known finite set of URI's.

$\text{merge}(g_{pb}, g_{cg}, g_{sw})$



navigation from g_{pb}

HF give a formal semantics for a model of query processing on an *infinite web of data*. HF do not study the impact of updates.

In essence, they follow the AV framework, with the navigational query semantics of BGS, abstracting away from the LOD standards, and also concrete models of navigation

- ▶ e.g., they consider, in addition to the BGS model of resolving all outgoing links, resolving only those links in current partial query result bindings

They also adapt the computability notions of AV to this setting.

Formal models for querying the web of data

We see that these general models can be categorized along two axes

- ▶ Whether or not the web of data is **bounded** or **unbounded**
 - ▶ *bounded*: models of Mendelzon and Milo; models of Bouquet et al.
 - ▶ *unbounded*: Abiteboul and Vianu; Hartig and Freytag

Formal models for querying the web of data

We see that these general models can be categorized along two axes

- ▶ Whether or not the web of data is **bounded** or **unbounded**
 - ▶ *bounded*: models of Mendelzon and Milo; models of Bouquet et al.
 - ▶ *unbounded*: Abiteboul and Vianu; Hartig and Freytag
- ▶ Whether or not the web of data is treated as **static** or **dynamic**
 - ▶ *static*: first model of Mendelzon and Milo; models of Bouquet et al.; Abiteboul and Vianu; Hartig and Freytag
 - ▶ *dynamic*: second model of Mendelzon and Milo

Formal models for querying the web of data

We see that these general models can be categorized along two axes

- ▶ Whether or not the web of data is **bounded** or **unbounded**
 - ▶ *bounded*: models of Mendelzon and Milo; models of Bouquet et al.
 - ▶ *unbounded*: Abiteboul and Vianu; Hartig and Freytag
- ▶ Whether or not the web of data is treated as **static** or **dynamic**
 - ▶ *static*: first model of Mendelzon and Milo; models of Bouquet et al.; Abiteboul and Vianu; Hartig and Freytag
 - ▶ *dynamic*: second model of Mendelzon and Milo

Note that there has been no formal study of a **dynamic unbounded** model for the web! Clearly, the real web as we know it satisfies both of these conditions ...

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several **fundamental challenges** have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- ▶ Data sources are not reliable

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several **fundamental challenges** have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- ▶ Data sources are not reliable
- ▶ Consumer behavior cannot be anticipated

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several **fundamental challenges** have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- ▶ Data sources are not reliable
- ▶ Consumer behavior cannot be anticipated
- ▶ Data sources are not always trustworthy

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several **fundamental challenges** have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- ▶ Data sources are not reliable
- ▶ Consumer behavior cannot be anticipated
- ▶ Data sources are not always trustworthy
- ▶ You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several **fundamental challenges** have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- ▶ Data sources are not reliable
- ▶ Consumer behavior cannot be anticipated
- ▶ Data sources are not always trustworthy
- ▶ You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)
- ▶ Universal cost models cannot be maintained

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several **fundamental challenges** have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- ▶ Data sources are not reliable
- ▶ Consumer behavior cannot be anticipated
- ▶ Data sources are not always trustworthy
- ▶ You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)
- ▶ Universal cost models cannot be maintained
- ▶ Query execution is non-deterministic

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several **fundamental challenges** have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- ▶ Data sources are not reliable
- ▶ Consumer behavior cannot be anticipated
- ▶ Data sources are not always trustworthy
- ▶ You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)
- ▶ Universal cost models cannot be maintained
- ▶ Query execution is non-deterministic
- ▶ Standards do not guarantee interoperability (i.e., homogeneity)

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several **fundamental challenges** have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- ▶ Data sources are not reliable
- ▶ Consumer behavior cannot be anticipated
- ▶ Data sources are not always trustworthy
- ▶ You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)
- ▶ Universal cost models cannot be maintained
- ▶ Query execution is non-deterministic
- ▶ Standards do not guarantee interoperability (i.e., homogeneity)
- ▶ Data sources are not coordinated centrally, both in alignment of URI references and in vocabulary

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several **fundamental challenges** have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- ▶ Data sources are not reliable
- ▶ Consumer behavior cannot be anticipated
- ▶ Data sources are not always trustworthy
- ▶ You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)
- ▶ Universal cost models cannot be maintained
- ▶ Query execution is non-deterministic
- ▶ Standards do not guarantee interoperability (i.e., homogeneity)
- ▶ Data sources are not coordinated centrally, both in alignment of URI references and in vocabulary
- ▶ You can guarantee at most two of the following properties at a time: alignment (i.e., currency), coverage, efficiency

Practical models for querying the web of data

Towards overcoming these challenges for practical LOD query processing solutions, **three basic approaches** can be identified (Hartig and Langegger 2010, Rakhmawati et al. 2013)

1. traditional search engines
2. federation of sources
3. discovery-driven query engines

Let's next discuss and highlight the pros/cons of these approaches

Traditional search engines

There are several applications of the search paradigm to discovery of LOD (e.g., Sindice and Falcons). Here, the web of data is crawled and indexed, with various limited query interfaces provided to clients

Traditional search engines

There are several applications of the search paradigm to discovery of LOD (e.g., Sindice and Falcons). Here, the web of data is crawled and indexed, with various limited query interfaces provided to clients

pros.

- ▶ coverage is quite good, potentially including a significant portion of the web of data
- ▶ response times and throughput are quite high
- ▶ data is relatively up to date

cons.

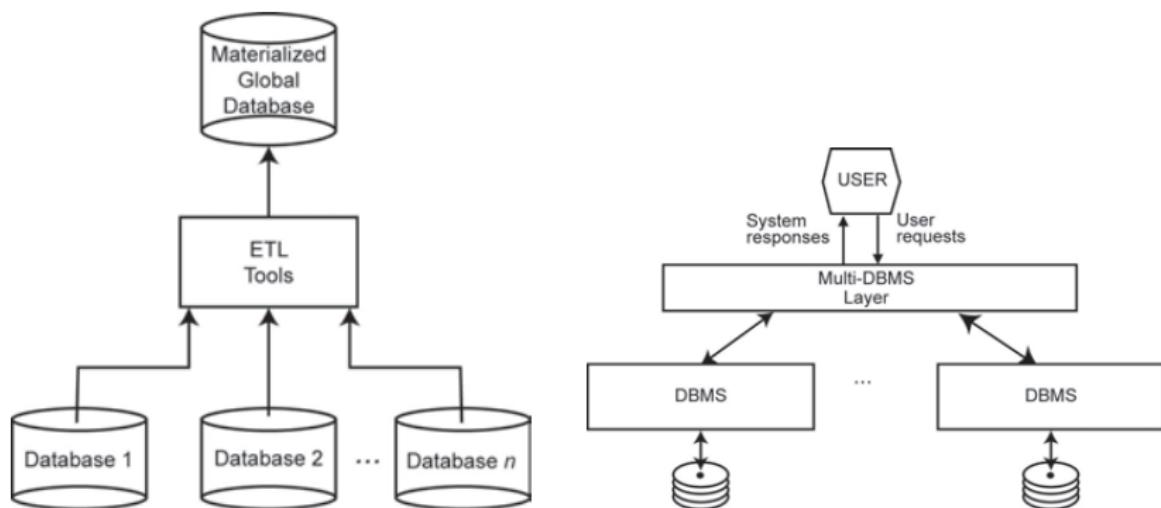
- ▶ precision and recall can both be too low for some applications
- ▶ dynamic data sources can grow stale, depending on frequency of crawls
- ▶ limited query access, client must do query processing

Federation of sources

Also traditionally known as distributed heterogeneous- or multi-databases.

Two basic classes

- ▶ data warehouses (materialized federation)
- ▶ mediator systems (logical federation)



Federation of sources

pros.

- ▶ response times and throughput are quite high (for materialized federations)
- ▶ data is up to date (for mediated federations)
- ▶ precision and recall are both high
- ▶ richer query access languages

cons.

- ▶ response times and throughput can be quite low (for mediated federations)
- ▶ data can be out of date (for materialized federations)
- ▶ coverage is quite poor, limited to known and integrated sources
- ▶ query evaluation costs can grow (for mediated federations)

Link-traversal-based query execution

The LTBQE approach closely follows the online navigational models of HF.

This is a new approach that is still in an early stage of study. The basic idea is to interleave query pattern matching with navigation and retrieval of LOD resources

- ▶ resources might be seeded, or just based on URLs in the query

Link-traversal-based query execution

The LTBQE approach closely follows the online navigational models of HF.

This is a new approach that is still in an early stage of study. The basic idea is to interleave query pattern matching with navigation and retrieval of LOD resources

- ▶ resources might be seeded, or just based on URIs in the query

In general, solution strategies of first research proposals focus engineering effort along three axes (Hartig 2013)

- ▶ how data sources are selected (live/cold-start or index-driven)
- ▶ how data sources are ranked (scoring, and selecting top-k)
- ▶ how data retrieval and result construction are intertwined (separate retrieval/construction, or fully intertwined)

Link-traversal-based query execution

pros.

- ▶ data is up to date
- ▶ coverage can be quite high
- ▶ precision is high
- ▶ richer query access languages

cons.

- ▶ response times and throughput can be quite low
- ▶ query evaluation costs can grow

References & Credits, 1/2

- ▶ Querying semantic data on the web. Marcelo Arenas, Claudio Gutierrez, Daniel P. Miranker, Jorge Perez, and Juan Sequeda. *SIGMOD Record* 41(4): 6-17, 2012.
<http://www.sigmod.org/publications/sigmod-record/1212/pdfs/03.principles.arenas.pdf>
- ▶ Queries and computation on the web. Serge Abiteboul and Victor Vianu. *Theor. Comput. Sci.* 239(2):231-255, 2000. [http://dx.doi.org/10.1016/S0304-3975\(99\)00221-2](http://dx.doi.org/10.1016/S0304-3975(99)00221-2)
- ▶ A formal model of queries on interlinked RDF graphs. Paolo Bouquet, Chiara Ghidini, and Luciano Serafini. *AAAI Spring Symposium 2010*.
<http://www.aaai.org/ocs/index.php/SSS/SSS10/paper/viewFile/1185/1441>
- ▶ An overview on execution strategies for linked data queries. Olaf Hartig. *Datenbank-Spektrum* 13(2):89-99, 2013.
https://cs.uwaterloo.ca/~ohartig/files/Hartig_LDQueryExec_DBSpektrum2013_Preprint.pdf

References & Credits, 2/2

- ▶ A database perspective on consuming linked data on the web. Olaf Hartig and Andreas Langegger. *Datenbank-Spektrum* 10(2):57-66, 2010.
https://cs.uwaterloo.ca/~ohartig/files/Hartig_QueryingLD_DB_Spektrum_Preprint.pdf
- ▶ Foundations of traversal based query execution over linked data. Olaf Hartig and Johann-Christoph Freytag. *HT 2012*. <http://arxiv.org/abs/1108.6328>
- ▶ Formal models of web queries. Alberto O. Mendelzon and Tova Milo. *Inf. Syst.* 23(8):615-637, 1998. <ftp://ftp.cs.toronto.edu/db/papers/infosysMM.ps.gz>
- ▶ Querying over federated SPARQL endpoints - a state of the art survey. Nur Aini Rakhmawati, Jürgen Umbrich, Marcel Karnstedt, Ali Hasnain, and Michael Hausenblas. DERI Technical Report 2013-06-07, 2013.
<http://arxiv.org/abs/1306.1723>
- ▶ Eight fallacies when querying the web of data. Jürgen Umbrich, Claudio Gutierrez, Aidan Hogan, Marcel Karnstedt, and Josiane Xavier Parreira. *DESWEB 2013*. <http://www.deri.ie/sites/default/files/publications/fallacies.pdf>
- ▶ The ACE theorem for querying the web of data. Jürgen Umbrich, Claudio Gutierrez, Aidan Hogan, Marcel Karnstedt, and Josiane Xavier Parreira. *WWW 2013*. <http://www.deri.ie/sites/default/files/publications/www13-poster.pdf>

Recap

1. Introduction and overview of linked and graph data, in the context of related research in data science
2. Models for graph and linked data
 - ▶ RDF and RDFS data model
 - ▶ graph database models
3. Introduction to querying over graph and linked data.
 - ▶ query languages for graphs
 - ▶ SPARQL 1.1
 - ▶ models for linked data
 - ▶ querying linked data

Looking ahead

Next week

- ▶ Performance tuning, course summary, and exam review (Wednesday)
- ▶ First batch of project presentations: teams 1-8 (Friday)
- ▶ Physical hand-in of written assignment (Friday)

Database tuning and Course review

Lecture 11
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

10 June 2015

Agenda

- ▶ Mini-lecture on DB Tuning
- ▶ Review of course and prep for final exam

DB tuning

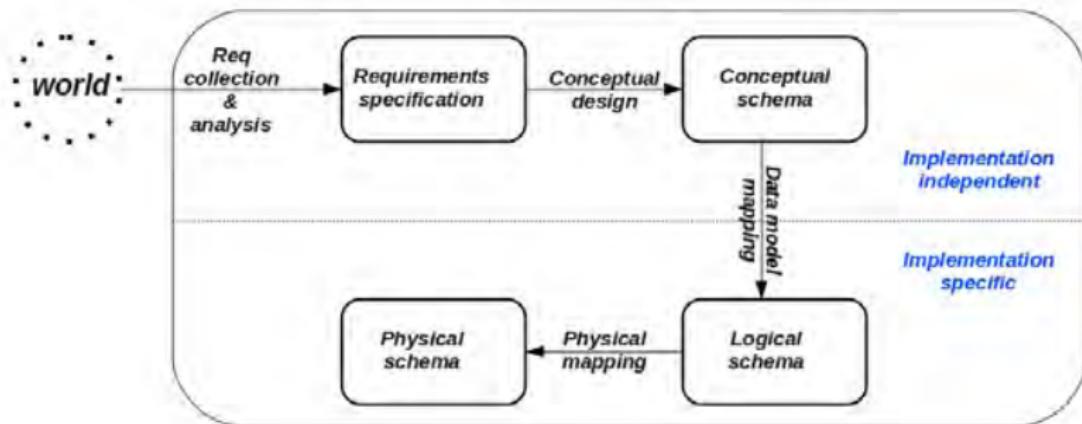
DB tuning: performance improvement, short of more/better hardware, in terms of response time and throughput.

DB tuning

DB tuning: performance improvement, short of more/better hardware, in terms of response time and throughput.

DB tuning is **not** about changing data semantics.

DB design



DB design process

DB design

“Goodness” in DB design

- ▶ **Conceptual.** Accurately reflect the semantics of use in the modeled domain.
- ▶ **Logical.** Accurately reflect conceptual model and disallow redundancies and update anomalies, as best possible.

DB design

“Goodness” in DB design

- ▶ **Conceptual.** Accurately reflect the semantics of use in the modeled domain.
- ▶ **Logical.** Accurately reflect conceptual model and disallow redundancies and update anomalies, as best possible.
- ▶ **Physical.** Accurately reflect logical model and efficiently and reliably support use of data.

DB design

“Goodness” in DB design

- ▶ **Conceptual.** Accurately reflect the semantics of use in the modeled domain.
- ▶ **Logical.** Accurately reflect conceptual model and disallow redundancies and update anomalies, as best possible.
- ▶ **Physical.** Accurately reflect logical model and efficiently and reliably support use of data.

The distinction between design and tuning is somewhat fuzzy

- ▶ design is about *semantics*
- ▶ tuning is about *efficient use*

Job of the DBA: design, tuning, protection, maintenance

DB tuning

Tuning is driven by observed and/or expected **workload**

- ▶ list of queries and updates, and their frequencies
 - ▶ relations involved
 - ▶ attributes involved
 - ▶ selection and join predicates

DB tuning

Tuning is driven by observed and/or expected **workload**

- ▶ list of queries and updates, and their frequencies
 - ▶ relations involved
 - ▶ attributes involved
 - ▶ selection and join predicates

Workload is never truly random, and hence we can leverage and support **regular patterns of use**

DB tuning

Tuning is driven by observed and/or expected **workload**

- ▶ list of queries and updates, and their frequencies
 - ▶ relations involved
 - ▶ attributes involved
 - ▶ selection and join predicates

Workload is never truly random, and hence we can leverage and support **regular patterns of use**

Let's consider tuning the basic components: buffers, indexes, schemas, queries

Tuning buffers and indexes

Six guidelines.

(1) Two types of cache:

- ▶ page cache
- ▶ procedure cache, for recent query plans

DBA can usually tweak cache size and replacement policy

Tuning buffers and indexes

Six guidelines.

(1) Two types of cache:

- ▶ page cache
- ▶ procedure cache, for recent query plans

DBA can usually tweak cache size and replacement policy

(2) To index or not to index: don't create indexes unnecessarily

- ▶ wasted space
- ▶ confuses query optimizer
- ▶ cost of maintenance can outweigh benefits

Tuning buffers and indexes

(3) Choice of search key: based on workload

- ▶ exact match vs. range selection
- ▶ multi-attribute

Tuning buffers and indexes

(3) Choice of search key: based on workload

- ▶ exact match vs. range selection
- ▶ multi-attribute

(4) Hash-based vs. tree-based

- ▶ usually B+tree
- ▶ except when
 - ▶ frequent join – index on join attributes of inner relation
 - ▶ equality selections in query

Tuning buffers and indexes

- (5) Balance the cost of index maintenance: if extremely dynamic, may not be worth it.

Tuning buffers and indexes

- (5) Balance the cost of index maintenance: if extremely dynamic, may not be worth it.
- (6) To cluster or not to cluster: not always necessary on primary key (often the default) as this grouping might not give you much
 - ▶ look at workload

Tuning buffers and indexes

Example.

```
SELECT M.ID, M.DeptID  
FROM Managers M  
WHERE M.Name = "Pointy-Haired Boss"
```

Tuning buffers and indexes

Example.

```
SELECT M.ID, M.DeptID  
FROM Managers M  
WHERE M.Name = "Pointy-Haired Boss"
```

Default is a clustered index on primary key Managers.ID

- ▶ not useful for this query
- ▶ if manager names are fairly unique, build unclustered index on Managers.Name, or, if warranted, build clustered index

Tuning buffers and indexes

Example.

```
SELECT M.ID, M.DeptID  
FROM Managers M  
WHERE M.Name = "Pointy-Haired Boss"
```

Default is a clustered index on primary key Managers.ID

- ▶ not useful for this query
- ▶ if manager names are fairly unique, build unclustered index on Managers.Name, or, if warranted, build clustered index
- ▶ also holds for Managers.Age and Managers.Salary
- ▶ Hash-based or Ordered index?

Tuning buffers and indexes

Example.

ProjectPart(projectID, partID)

PartSupplier(partID, supplierID)

```
SELECT P.projID, S.supplierID  
FROM ProjectPart P, PartSupplier S  
WHERE P.partID = S.partID
```

Tuning buffers and indexes

Example.

ProjectPart(projectID, partID)

PartSupplier(partID, supplierID)

```
SELECT P.projID, S.supplierID  
FROM ProjectPart P, PartSupplier S  
WHERE P.partID = S.partID
```

It seems natural to build an index on `PartSupplier.partID` to help out index nested loops join with `PartSupplier` as inner relation.

Tuning buffers and indexes

Example.

ProjectPart(projectID, partID)

PartSupplier(partID, supplierID)

```
SELECT P.projID, S.supplierID  
FROM ProjectPart P, PartSupplier S  
WHERE P.partID = S.partID
```

It seems natural to build an index on `PartSupplier.partID` to help out index nested loops join with `PartSupplier` as inner relation.

However, potentially many rows of `PartSupplier` will join with each row of `ProjectPart`, leading to large result size!

- ▶ need to use some other join algorithm
- ▶ hence, we should not build this index!

Tuning schemas

Three basics.

- 1: denormalize for read-heavy queries, and pay a higher price for maintaining data consistency

Tuning schemas

Three basics.

1: denormalize for read-heavy queries, and pay a higher price for maintaining data consistency

- ▶ for example, if the join on partID is very common, join ProjectPart and PartSupplier as one relation
ProjectPartSupplier(projectID, partID, supplierID)

Tuning schemas

Three basics.

1: **denormalize** for read-heavy queries, and pay a higher price for maintaining data consistency

- ▶ for example, if the join on partID is very common, join ProjectPart and PartSupplier as one relation
ProjectPartSupplier(projectID, partID, supplierID)

2: **vertical partitioning**. place infrequently used columns in a separate table (smaller tuples = smaller pages = more records per I/O)

Tuning schemas

Three basics.

1: denormalize for read-heavy queries, and pay a higher price for maintaining data consistency

- ▶ for example, if the join on partID is very common, join ProjectPart and PartSupplier as one relation
ProjectPartSupplier(projectID, partID, supplierID)

2: vertical partitioning. place infrequently used columns in a separate table (smaller tuples = smaller pages = more records per I/O)

- ▶ for example, place rarely accessed attributes such as hireDate and hireReferral in separate table from Managers
- ▶ cf. column stores such as MonetDB

Tuning schemas

3: horizontal partitioning. place infrequently used rows in a separate table (fewer tuples = smaller tables = fewer I/O's)

Tuning schemas

3: horizontal partitioning. place infrequently used rows in a separate table (fewer tuples = smaller tables = fewer I/O's)

- ▶ for example, place fired or retired managers in a separate table from Managers
- ▶ cf. google file system; distributed DB design

Tuning queries

Five heuristics.

- (1) avoid sorts: DISTINCT, UNION, INTERSECT, EXCEPT.

Tuning queries

Five heuristics.

- (1) avoid sorts: DISTINCT, UNION, INTERSECT, EXCEPT.
for example, in the query

```
SELECT DISTINCT M.ID, M.DeptID  
FROM Managers M  
WHERE M.Name = "Pointy-Haired Boss"
```

the DISTINCT is unnecessary, as ID is a key for Managers, so
is also a key for any subset of Managers

Tuning queries

(1, cont.) consider next

```
SELECT DISTINCT M.ID, D.Phone  
FROM Managers M, Departments D  
WHERE M.DID = D.DID
```

Tuning queries

(1, cont.) consider next

```
SELECT DISTINCT M.ID, D.Phone  
FROM Managers M, Departments D  
WHERE M.DID = D.DID
```

Is DISTINCT necessary?

Tuning queries

(1, cont.) consider next

```
SELECT DISTINCT M.ID, D.Phone  
FROM Managers M, Departments D  
WHERE M.DID = D.DID
```

Is DISTINCT necessary? No, since DID is a key for
Departments (and ID is a key for Managers)

Tuning queries

(1, cont.) consider next

```
SELECT DISTINCT M.ID, D.Phone  
FROM Managers M, Departments D  
WHERE M.DID = D.DID
```

Is DISTINCT necessary? No, since DID is a key for Departments (and ID is a key for Managers)

The relationship among DISTINCT, keys and joins can be generalized:

- ▶ Call a table T *privileged* if the fields returned by the SELECT contain a key of T .

Tuning queries

(1, cont.) consider next

```
SELECT DISTINCT M.ID, D.Phone  
FROM Managers M, Departments D  
WHERE M.DID = D.DID
```

Is DISTINCT necessary? No, since DID is a key for Departments (and ID is a key for Managers)

The relationship among DISTINCT, keys and joins can be generalized:

- ▶ Call a table T *privileged* if the fields returned by the SELECT contain a key of T .
- ▶ Let R be an unprivileged table. Suppose that R is joined on equality by its key field to some other table S , then we say R reaches S .
- ▶ Now, define reaches to be transitive. So, if R_1 reaches R_2 and R_2 reaches R_3 , then say that R_1 reaches R_3 .

Tuning queries

(1, cont.) Reaches Theorem.

There will be no duplicates among the records returned by a selection, even in the absence of DISTINCT, if one of the two following conditions hold:

- ▶ Every table mentioned in the FROM clause is privileged.
- ▶ Every unprivileged table reaches at least one privileged table.

Tuning queries

(2) avoid unnecessary scans: “ \neq ” in selection condition can often be rewritten

- ▶ for example, credits $\neq 3$ might be rewritten as
credits = 1 OR ... credits = 4

Tuning queries

(2) avoid unnecessary scans: “ \neq ” in selection condition can often be rewritten

- ▶ for example, credits $\neq 3$ might be rewritten as credits = 1 OR ... credits = 4

(3) instead of creating new indexes, check for usage of existing indexes

- ▶ for example, Name = ‘‘Fred’’ OR Salary = 4000
- ▶ if separate indexes exist on Name and Salary, compiler might not catch this
- ▶ instead, rewrite as union of two queries

Tuning queries

- (4) help the optimizer reuse procedure cache. for example,

```
SELECT P.Name  
FROM Professor P  
WHERE P.DeptID = 'Math'
```

and we will also want profs from EE, ME, IE,

Tuning queries

- (4) help the optimizer reuse procedure cache. for example,

```
SELECT P.Name  
FROM Professor P  
WHERE P.DeptID = 'Math'
```

and we will also want profs from EE, ME, IE,

The optimizer most likely won't reuse the plan for the first query in these later queries, even though that plan is cached. Instead, we should use a host variable:

```
...WHERE P.DeptID = :deptID ...
```

Tuning queries

(5) decorrelate and unnest complex queries.

Tuning queries

(5) decorrelate and unnest complex queries. for example,

```
SELECT P.Name  
FROM Professor P  
WHERE EXISTS ( SELECT *  
    FROM department D  
    WHERE D.Name = 'Mathematics'  
        AND  
        P.DeptID = D.ID )
```

Tuning queries

(5) decorrelate and unnest complex queries. for example,

```
SELECT P.Name  
FROM Professor P  
WHERE EXISTS ( SELECT *  
    FROM department D  
    WHERE D.Name = 'Mathematics'  
        AND  
        P.DeptID = D.ID )
```

Typical optimizer will generate a plan which executes the inner correlated query **for each Profs tuple!** We should decorrelate if possible.

Tuning queries

(5) decorrelate and unnest complex queries. for example,

```
SELECT P.Name  
FROM Professor P  
WHERE EXISTS ( SELECT *  
    FROM department D  
    WHERE D.Name = 'Mathematics'  
        AND  
        P.DeptID = D.ID )
```

Typical optimizer will generate a plan which executes the inner correlated query **for each Profs tuple!** We should decorrelate if possible.

```
SELECT P.Name  
FROM Professor P  
WHERE P.DeptID IN ( SELECT D.ID  
    FROM department D  
    WHERE D.Name = 'Mathematics' )
```

Tuning queries

(5, cont.) However, optimizer won't recognize the implicit join here, and won't make use of available indexes. Hence, we should unnest if possible

```
SELECT P.Name  
FROM Professor P, Department D  
WHERE P.DeptID = D.ID AND D.Name = 'Mathematics'
```

Tuning queries

(5, cont.) However, optimizer won't recognize the implicit join here, and won't make use of available indexes. Hence, we should unnest if possible

```
SELECT P.Name  
FROM Professor P, Department D  
WHERE P.DeptID = D.ID AND D.Name = 'Mathematics'
```

In general, optimizers don't recognize equivalence of such plans, and so it is up to the client/DBA to tune.

DB tuning: tools

All major systems provide tools and means for tuning (and, third-party support as well).

DB tuning: tools

All major systems provide tools and means for tuning (and, third-party support as well).

Example. SQLite DB tuning.

- ▶ SQLite analyzer, from SQLite.
- ▶ SQLite Database Browser, open source.
- ▶ SQLite Manager, commercial product.
- ▶ SQLite Expert, commercial product.

DB tuning: tools

All major systems provide tools and means for tuning (and, third-party support as well).

Example. SQLite DB tuning.

- ▶ SQLite analyzer, from SQLite.
- ▶ SQLite Database Browser, open source.
- ▶ SQLite Manager, commercial product.
- ▶ SQLite Expert, commercial product.

Configuration:

- ▶ PRAGMA cacheSize
- ▶ PRAGMA cacheSpill to set steal/no-steal cache policy.
- ▶ PRAGMA walAutocheckpoint, to set checkpoint policy for the write-ahead log.
- ▶ ANALYZE, REINDEX, and VACUUM, to rebuild/clean the DB and stats
- ▶ EXPLAIN, to view query plan in virtual machine code

DB tuning: recap

- ▶ DB tuning is **not** about changing data semantics.
- ▶ Tuning is driven by observed and/or expected **workload** which is never truly random
- ▶ basic tuning components: buffers, indexes, schemas, queries
- ▶ every DB system has many, many tuning knobs, and lots of tool support for tuning

DB tuning: recap

- ▶ DB tuning is **not** about changing data semantics.
- ▶ Tuning is driven by observed and/or expected **workload** which is never truly random
- ▶ basic tuning components: buffers, indexes, schemas, queries
- ▶ every DB system has many, many tuning knobs, and lots of tool support for tuning

Nice book on DB tuning. *Database Tuning: principles, experiments, and troubleshooting techniques*. Shasha and Bonnet, 2002.

Exercise

SalesPerson(personID, regionID, pName, address, age, hireDate,
taxCode, healthCode, parkingSpot, ...)

Region(regionID, rName, area, rCode, history, taxOffice, ...)

Sales(personID, regionID, amount)

Exercise

SalesPerson(personID, regionID, pName, address, age, hireDate,
taxCode, healthCode, parkingSpot, ...)

Region(regionID, rName, area, rCode, history, taxOffice, ...)

Sales(personID, regionID, amount)

Very common query:

```
SELECT DISTINCT P.personID, P.address, P.pName, R.rName  
FROM SalesPerson P, Region R, Sales S  
WHERE P.personID = S.personID AND R.regionID = S.regionID  
      AND P.regionID = R.regionID
```

Can we tune the schema and/or query to improve response time?

Course review

Recap

Major topics

Relational data in external memory (lectures 1-4)

- ▶ storage, sorting, indexing

Recap

Major topics

Relational data in external memory (lectures 1-4)

- ▶ storage, sorting, indexing

Query processing (lecture 5)

- ▶ Implementing σ , π , \bowtie , \cup , $-$

Recap

Major topics

Relational data in external memory (lectures 1-4)

- ▶ storage, sorting, indexing

Query processing (lecture 5)

- ▶ Implementing σ , π , \bowtie , \cup , $-$

Views (lecture 6)

- ▶ Histograms, reasoning about views

Recap

Major topics

Relational data in external memory (lectures 1-4)

- ▶ storage, sorting, indexing

Query processing (lecture 5)

- ▶ Implementing σ , π , \bowtie , \cup , $-$

Views (lecture 6)

- ▶ Histograms, reasoning about views

Query optimization (lecture 7)

- ▶ logical plans, physical plans, dynamic programming

Recap

Major topics, cont.

Distributed query processing (lecture 8)

- ▶ parallel, distributed, and mediator systems; acyclic queries

Recap

Major topics, cont.

Distributed query processing (lecture 8)

- ▶ parallel, distributed, and mediator systems; acyclic queries

Transaction management (lecture 9)

- ▶ ACID, 2PL, S2PL

Recap

Major topics, cont.

Distributed query processing (lecture 8)

- ▶ parallel, distributed, and mediator systems; acyclic queries

Transaction management (lecture 9)

- ▶ ACID, 2PL, S2PL

NoSQL, graphs, and linked data (lecture 10)

Recap

Major topics, cont.

Distributed query processing (lecture 8)

- ▶ parallel, distributed, and mediator systems; acyclic queries

Transaction management (lecture 9)

- ▶ ACID, 2PL, S2PL

NoSQL, graphs, and linked data (lecture 10)

Tuning and recap (lecture 11)

Old final exam

Five questions, three hours. Let's practice the first three or four.

Old final exam

Five questions, three hours. Let's practice the first three or four.

- (1) Consider the following conjunctive query Q :

$$\text{result}(A, B, C, D, E, F) \leftarrow r(A, B, C), s(A, F, E), t(E, D, C), u(A, C, E)$$

1. Give the hypergraph representation of Q .
2. Is Q acyclic? If so, can it be made cyclic by removing one hyperedge? Otherwise, can it be made acyclic by removing one hyperedge?

Old final exam

- (2) Given a schedule S over transactions $\{T_1, \dots, T_n\}$, the “strong graph” associated with S is the directed graph $sg(S)$ having exactly one node for each transaction of S and an edge from T_i to T_j (for $i \neq j$) if and only if in S , for some object X there is an action $\alpha_i(X)$ of T_i on X which appears before an action $\alpha_j(X)$ of T_j on X .

Prove or disprove the following claims.

1. If $sg(S)$ is acyclic, then S is conflict serializable.
2. If S is conflict serializable, then $sg(S)$ is acyclic.

Old final exam

(3) Consider the following conjunctive queries.

$Q_1 : \text{result}(A) \leftarrow r(A, B), r(A, C), s(B, D, E), s(B, F, F)$

$Q_2 : \text{result}(X) \leftarrow r(X, Y), r(X, W), s(Y, W, W), t(X)$

Is it the case that $Q_2 \subseteq Q_1$? Prove your answer.

Old final exam

- (4) Consider the “semi-difference” relational algebra operator, defined as

$$\begin{aligned} R \triangleright S &= \{r \in R \mid \neg \exists s \in S (r \bowtie s \in R \bowtie S)\} \\ &= R - (R \bowtie S). \end{aligned}$$

Formally prove or disprove the following proposals for relational algebra equivalences.

1. $\sigma_\theta(R \triangleright S) = \sigma_\theta(R) \triangleright S$, where θ is a standard single-table selection condition which mentions only attributes in R (i.e., $atts(\theta) \subseteq atts(R) - atts(S)$).
2. $R \bowtie S = R \triangleright (R \triangleright S)$.