Advanced Data Management for Data Analysis

Stefan Manegold

Data Management @ LIACS

Group leader Database Architectures
Centrum Wiskunde & Informatica (CWI)
Amsterdam

s.manegold@liacs.leidenuniv.nl
http://www.cwi.nl/~manegold/

ADM: Agenda

- 07.09.2022: Lecture 1: **Introduction**
- 14.09.2022: Lecture 2: **SQL Recap**

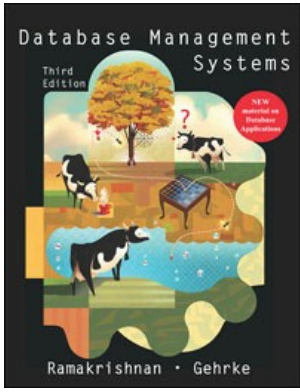
  (plus Assignment 1 [in groups; 3 weeks]: TPC-H benchmark)
- 21.09.2022: Lecture 3: **Column-Oriented Database Systems (1/6) - Motivation & Basic Concepts**
- 28.09.2022: Lecture 4: **Column-Oriented Database Systems (2a/6) - Selected Execution Techniques (1/2)**
- 05.10.2022: Lecture 5: **Column-Oriented Database Systems (2b/6) - Selected Execution Techniques (2/2)**

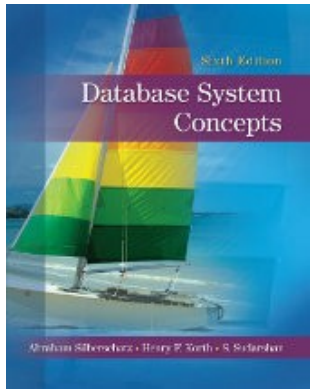
  (plus Assignment 3 [in groups; 3 weeks]: Compression techniques)
- 12.10.2022: Lecture 6: **Column-Oriented Database Systems (3/6) - Cache Conscious Joins**
- 19.10.2022: Lecture 7: **Column-Oriented Database Systems (4/6) - "Vectorized Execution"**
- ~~26.10.2022~~: ***No lecture!***
- 02.11.2022: Lecture 8: **DuckDB: An embedded database for data science (1/2) (guest lecture & *hands-on*)**

  (plus Assignment 2 [individual; 2 weeks]: Analysing NYC Cab dataset with DuckDB)
- 09.11.2022: Lecture 9: **DuckDB: An embedded database for data science (2/2) (guest lecture & *hands-on*)**
- 16.11.2022: Lecture 10: **Branch Misprediction & Predication**

  (plus Assignment 4 [individual; 2 weeks]: Predication)
- 23.11.2022: Lecture 11: **Column-Oriented Database Systems (5/6) - Adaptive Indexing**
- 30.11.2022: Lecture 12: **Column-Oriented Database Systems (6/6) - Progressive Indexing**

ADM: Expected Background





- **Database systems (e.g.):**
  - *Ramakrishnan*, *Gehrke*: **Database Management Systems (3rd International Edition)**, McGraw-Hill, 2003 (ISBN 0-07-246563-8)
  - *A. Silberschatz*, *H. F. Korth*, *S. Sudarshan:* **Database System Concepts (7th Edition),** McGraw-Hill, 2010 (ISBN 0-07-352332-1)

    book: https://www.db-book.com/

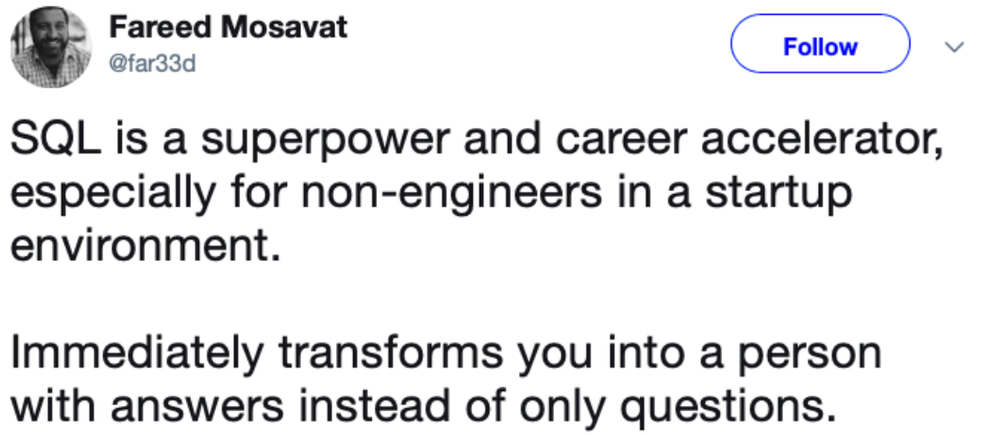
    slides: https://www.db-book.com/slides-dir/index.html
  - *Andy Pavlo:* **Introduction to Database Systems** course @ CMU incl. slides and videos on YouTube: https://15445.courses.cs.cmu.edu/fall2019/

ADM: SQL recap

Focus on
“analytical” SQL,
i.e.,
queries that *inspect and analyze* the data
rather than
transactions that *modify* the data.

Why SQL

- Decoupling of query and execution
- "Declarative"
- "I want these rows, figure out how to get them for me"
- Allows for optimisation
- Allows for changes to physical storage without changing queries
- Huge renaissance of SQL in ~ last 5 years



SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

1974

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

ABSTRACT: In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.

Episode 11: Interview with Don Chamberlin, designer of SQL database language

October 12, 2017 | Pramod Shashidhara

Episode 11: Interview wit...

00:00:00

libsyn

DOWNLOAD

Post Views: 19,465

Don Chamberlin holds a B.S. degree from Harvey Mudd College and a Ph.D. from Stanford University. For many years, he worked at Almaden Research Center, researching database languages and systems. He was a member of the System R research team that developed much of today's relational database technology and, together with Ray Boyce, he designed the original SQL database language.

https://www.mappingthejourney.com/single-post/2017/10/12/episode-11-interview-with-don-chamberlin-designer-of-sql-database-language/

Sample Dataset

Students

SID	FIRST	LAST
101	Ann	Smith
102	Michael	Jones
103	Richard	Turner
104	Maria	Brown

Results

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	9
103	H	1	5
103	M	1	7

- **SID**: Student ID
- **CAT**: Exercise Category:
  - **H**: Homework
  - **M**: Mid-term exam
- **ENO**: Exercise Number

SQL Schema

good!

Column/Attribute Name

```
CREATE TABLE RESULTS (
  SID INTEGER,
  CAT CHAR(1),
  ENO INTEGER,
  POINTS INTEGER)
```

Data Type

Most important types: `INTEGER`, `DECIMAL`, `STRING`

Name	Aliases	Description
bigint	int8	signed eight-byte integer
bigserial	serial8	autoincrementing eight-byte integer
bit [(*n*)]		fixed-length bit string
bit varying [(*n*)]	varbit	variable-length bit string
boolean	bool	logical Boolean (true/false)
box		rectangular box on a plane
bytea		binary data ("byte array")
character [(*n*)]	char [(*n*)]	fixed-length character string
character varying [(*n*)]	varchar [(*n*)]	variable-length character string
cidr		IPv4 or IPv6 network address
circle		circle on a plane
date		calendar date (year, month, day)
double precision	float8	double precision floating-point number (8 bytes)
inet		IPv4 or IPv6 host address
integer	int, int4	signed four-byte integer
interval [*fields*] [(*p*)]		time span
json		textual JSON data
jsonb		binary JSON data, decomposed
line		infinite line on a plane
lseg		line segment on a plane
macaddr		MAC (Media Access Control) address
money		currency amount
numeric [(*p*, *s*)]	decimal [(*p*, *s*)]	exact numeric of selectable precision
path		geometric path on a plane
pg_lsn		PostgreSQL Log Sequence Number
point		geometric point on a plane
polygon		closed geometric path on a plane
real	float4	single precision floating-point number (4 bytes)
smallint	int2	signed two-byte integer
smallserial	serial2	autoincrementing two-byte integer
serial	serial4	autoincrementing four-byte integer
text		variable-length character string
time [(*p*)] [without time zone]		time of day (no time zone)
time [(*p*)] with time zone	timetz	time of day, including time zone
timestamp [(*p*)] [without time zone]		date and time (no time zone)
timestamp [(*p*)] with time zone	timestamptz	date and time, including time zone
tsquery		text search query
tsvector		text search document
txid_snapshot		user-level transaction ID snapshot
uuid		universally unique identifier
xml		XML data

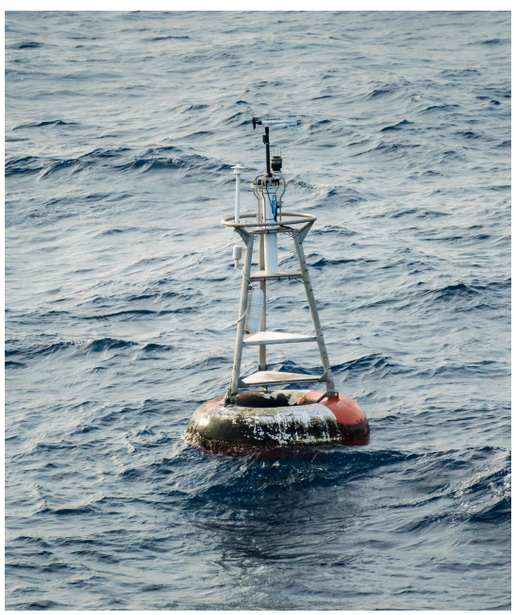
PostgreSQL 9.5
Data Types

Builtin SQL types

CHAR ['(' *length* ')'] | CHARACTER ['(' *length* ')']	UTF-8 character string with length upperbound limit. CHAR or CHARACTER without the "(*length*)" specification are treated as CHAR(1). **Note**: currently no spaces are padded at the end
VARCHAR '(' *length* ')' | CHARACTER VARYING '(' *length* ')'	UTF-8 character string with length upperbound limit
TEXT | CLOB | CHARACTER LARGE OBJECT | STRING	UTF-8 character string with unbounded length
BLOB | BINARY LARGE OBJECT	bytes string with unbounded length
BOOLEAN | BOOL	logic values: true or false
TINYINT	8 bit signed integer between -127 and 127
SMALLINT	16 bit signed integer between -32767 and 32767
INT | INTEGER	32 bit signed integer between -2147483647 and 2147483647
BIGINT	64 bit signed integer between -9223372036854775807 and 9223372036854775807
HUGEINT	128 bit signed integer between $-2^{127}+1$ and $+2^{127}-1$ (±170141183460469231731687303715884105727) **Note**: HUGEINT is only available on platforms with a C-compiler that supports the __int128 or __int128_t data type (e.g., recent gcc, clang, & icc on Linux or MacOS X) and from Jul2015 release onwards
DECIMAL '(' *Prec* ',' *Scale* ')' | DEC '(' *Prec* ',' *Scale* ')' | NUMERIC '(' *Prec* ',' *Scale* ')'	Exact decimal number with precision *Prec* and scale *Scale*. *Prec* must be between 1 and 18 (or 38 when HUGEINT is also supported). *Scale* must be between 0 and *Prec*
DECIMAL '(' *Prec* ')' | DEC '(' *Prec* ')' | NUMERIC '(' *Prec* ')'	Exact decimal number with precision *Prec* and scale 0. *Prec* must be between 1 and 18 (or 38 when HUGEINT is also supported).
DECIMAL | DEC | NUMERIC	Exact decimal number with precision 18 and scale 3.
REAL	32 bit floating point approximate number
FLOAT | DOUBLE | DOUBLE PRECISION	64 bit floating point approximate number
FLOAT '(' *Prec* ')'	floating point approximate number with binary precision *Prec*. *Prec* must be between 1 and 53. FLOAT(24) is same as REAL, FLOAT(53) is same as DOUBLE

NULL

- The relational model allows **missing attribute values**, *i.e.*, table entries may be empty.
- Formally, the set of possible values (the domain) for an attribute is extended by a new special value "null."
- "Null" is *not* the number 0 or the empty string.
  A null value is different from all values of any data type.



NULL

- Null values are used to model a variety of real-world scenarios:[1]
  - ▷ **A value exists (in the real world), but is not known.**

    In table `STUDENTS`, `EMAIL` might be missing for a student.
  - ▷ **No value exists.**

    A student might not have an e-mail address; not everyone has a first and last name.
  - ▷ **The attribute is not applicable for this tuple.**

    Some exercises are for training only: no points will be given.
  - ▷ **Any value will do.**

[1] A committee once found 13 different meaning for null values.

Hannes: NULL = dunno

Basic Query Syntax

Projection

Column Reference with Alias

```
SELECT   SID, S.FIRST AS F
FROM     STUDENTS AS S
WHERE    S.LAST = 'Smith'
```

Table with Alias

Selection

Condition

String Literal

```
SELECT   SID, S.FIRST
FROM     STUDENTS AS S
WHERE    S.LAST = 'Smith'
```

S

SID	FIRST	LAST
101	Ann	Smith
102	Michael	Jones
103	Richard	Turner
104	Maria	Brown

```
WHERE    S.LAST = 'Smith'
```

Row Selection

SID	FIRST	LAST
101	Ann	Smith
102	Michael	Jones
103	Richard	Turner
104	Maria	Brown

```
SELECT   SID, S.FIRST
```

Column Projection

Quoting

```
SELECT   "SID", "S"."FIRST" AS "F"
FROM     "STUDENTS" AS "S"
WHERE    "LAST" = 'Smith'
```

Safe bet!

Expressions

Arithmetic in Projection

String Functions

Integer Literal

```
SELECT   SID + 42, LENGTH(FIRST)
FROM     STUDENTS
WHERE    LAST = 'Smith' AND (SID + 1) > 102
```

Logical Conjunction

Functions? Documentation!

Bells & Whistles

No Duplicates

```
SELECT DISTINCT FIRST
FROM            STUDENTS
WHERE           LAST = 'Smith'
ORDER BY        FIRST DESC
LIMIT           10
```

Result Sorting

Top 10

Aggregates

```
SELECT COUNT(*)
FROM   RESULTS
```

Aggregation Function

COUNT
MIN
MAX
AVG
SUM
STDEV
VAR
...

```
SELECT MIN(POINTS), MAX(POINTS)
FROM   RESULTS
WHERE  CAT = 'H'
```

Changes Result Cardinality!

```
SELECT MIN(POINTS), MAX(POINTS)
FROM   RESULTS
WHERE  CAT = 'H'
```

Results

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	9
103	H	1	5
103	M	1	7

MIN(POINTS)	MAX(POINTS)
5	10

```
WHERE  CAT = 'H'
```

```
SELECT MIN(POINTS), MAX(POINTS)
```

Grouped Aggregates

```
SELECT ENO, AVG(POINTS)
FROM RESULTS
WHERE CAT = 'H'
GROUP BY ENO
```

Grouping Attribute

As many result rows as groups

```
SELECT ENO, AVG(POINTS)
FROM RESULTS
WHERE CAT = 'H'
GROUP BY ENO
```

Results

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	9
103	H	1	5
103	M	1	7

ENO=1

...	ENO	POINTS
	1	10
	1	9
	1	5

ENO=2

...	ENO	POINTS
	2	8
	2	9

ENO	AVG(POINTS)
1	8
2	8.5

WHERE CAT = 'H'

GROUP BY ENO

SELECT ENO,
AVG(POINTS)

Filtering Grouped Aggregates

```
SELECT ENO, AVG(POINTS)
FROM RESULTS
WHERE CAT = 'H'
GROUP BY ENO
HAVING AVG(POINTS) > 8.2
```

Group filter criteria
no column references!

```
SELECT ENO, AVG(POINTS)
FROM RESULTS
WHERE CAT = 'H'
GROUP BY ENO
HAVING AVG(POINTS) > 8.2
```

ENO=1

...	ENO	POINTS
	1	10
	1	9
	1	5

ENO=2

...	ENO	POINTS
	2	8
	2	9

GROUP BY ENO

ENO	AVG(POINTS)
1	8
2	8.5

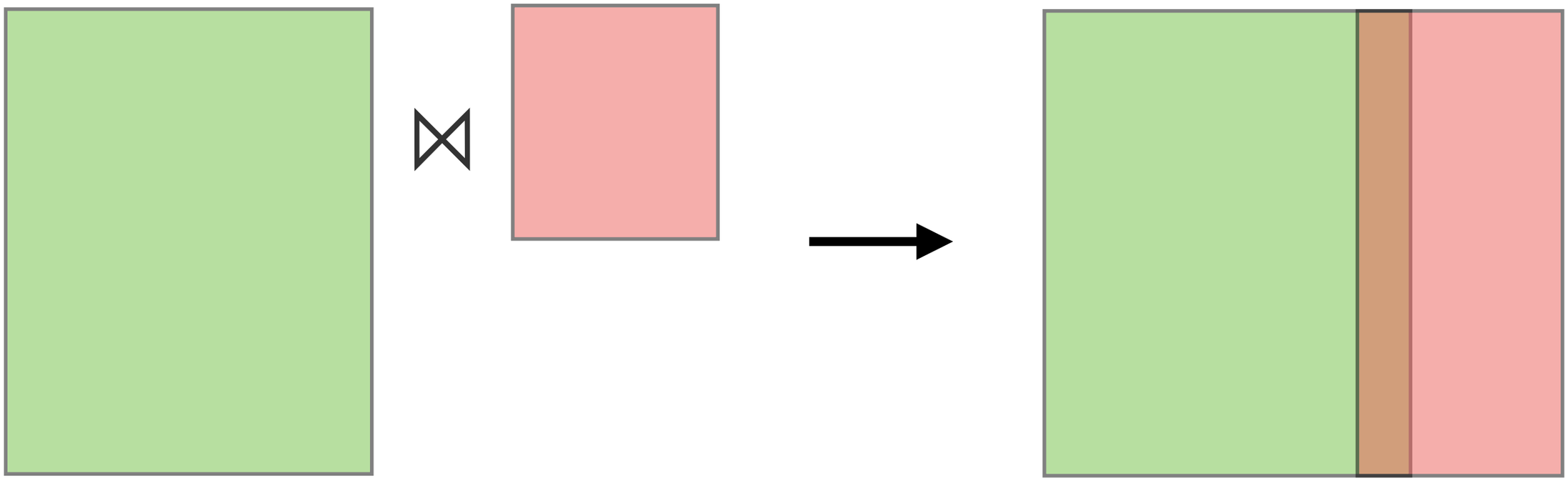
HAVING
AVG(POINTS) > 8.2

ENO	AVG(POINTS)
2	8.5

SELECT ENO,
AVG(POINTS)

Joins

Horizontal Combination of Tables



Rectangular Result

Inner Joins

```
SELECT R.ENO, R.POINTS
FROM   STUDENTS S JOIN RESULTS R
  ON   (S.SID = R.SID)
WHERE  S.LAST = 'Smith'
```

Join Condition

Students

SID	FIRST	LAST
101	Ann	Smith
102	Michael	Jones
103	Richard	Turner
104	Maria	Brown

$\bowtie_{SID}$

Results

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	9
103	H	1	5
103	M	1	7

S.SID	S.FIRST	S.LAST	R.SID	R.CAT	R.ENO	R.POINTS
101	Ann	Smith	101	H	1	10
101	Ann	Smith	101	H	2	8
101	Ann	Smith	101	M	1	12
102	Michael	Jones	102	H	1	9
102	Michael	Jones	102	H	2	9
102	Michael	Jones	102	M	1	9
103	Richard	Turner	103	H	1	5
103	Richard	Turner	103	M	1	7

```
SELECT R.ENO, R.POINTS
FROM   STUDENTS S JOIN RESULTS R
  ON   (S.SID = R.SID)
WHERE  S.LAST = 'Smith'
```

S.SID	S.FIRST	S.LAST	R.SID	R.CAT	R.ENO	R.POINTS
101	Ann	Smith	101	H	1	10
101	Ann	Smith	101	H	2	8
101	Ann	Smith	101	M	1	12
102	Michael	Jones	102	H	1	9
102	Michael	Jones	102	H	2	9
102	Michael	Jones	102	M	1	9
103	Richard	Turner	103	H	1	5
103	Richard	Turner	103	M	1	7

```
SELECT R.ENO, R.POINTS
FROM   STUDENTS S JOIN RESULTS R
  ON   (S.SID = R.SID)
WHERE  S.LAST = 'Smith'
```

Students

SID	FIRST	LAST
101	Ann	Smith
102	Michael	Jones
103	Richard	Turner
104	Maria	Brown

⋈ SID

Results

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	9
103	H	1	5
103	M	1	7

```
WHERE  S.LAST = 'Smith'
```

S.SID	S.FIRST	S.LAST	R.SID	R.CAT	R.ENO	R.POINTS
101	Ann	Smith	101	H	1	10
101	Ann	Smith	101	H	2	8
101	Ann	Smith	101	M	1	12

Outer Joins

```
SELECT S.LAST, R.POINTS
FROM   STUDENTS S LEFT OUTER JOIN RESULTS R
  ON   (S.SID = R.SID)
```

https://blog.jooq.org/2016/07/05/say-no-to-venn-diagrams-when-explaining-joins/

Students

SID	FIRST	LAST
101	Ann	Smith
102	Michael	Jones
103	Richard	Turner
104	Maria	Brown

⋈ SID

Results

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	9
103	H	1	5
103	M	1	7

S.SID	S.FIRST	S.LAST	R.SID	R.CAT	R.ENO	R.POINTS
101	Ann	Smith	101	H	1	10
101	Ann	Smith	101	H	2	8
101	Ann	Smith	101	M	1	12
102	Michael	Jones	102	H	1	9
102	Michael	Jones	102	H	2	9
102	Michael	Jones	102	M	1	9
103	Richard	Turner	103	H	1	5
103	Richard	Turner	103	M	1	7
104	Maria	Brown	**NULL**	**NULL**	**NULL**	**NULL**

```
SELECT S.LAST, R.POINTS
FROM   STUDENTS S LEFT OUTER JOIN RESULTS R
  ON   (S.SID = R.SID)
```

S.SID	S.FIRST	S.LAST	R.SID	R.CAT	R.ENO	R.POINTS
101	Ann	Smith	101	H	1	10
101	Ann	Smith	101	H	2	8
101	Ann	Smith	101	M	1	12
102	Michael	Jones	102	H	1	9
102	Michael	Jones	102	H	2	9
102	Michael	Jones	102	M	1	9
103	Richard	Turner	103	H	1	5
103	Richard	Turner	103	M	1	7
104	Maria	Brown	NULL	NULL	NULL	**NULL**

Eeek!

More Joining

- Natural Join: Automatic condition from matching attributes
- Cross Join: Cartesian product (Big Data)
- Theta Join: More complex condition than `A=B`
- ...

Join Syntax

```
SELECT R.ENO, R.POINTS
FROM   STUDENTS S JOIN RESULTS R
  ON   (S.SID = R.SID)
WHERE  S.LAST = 'Smith'
```

Explicit Join condition in FROM

```
SELECT R.ENO, R.POINTS
FROM   STUDENTS S, RESULTS R
WHERE  S.SID = R.SID
  AND  S.LAST = 'Smith'
```

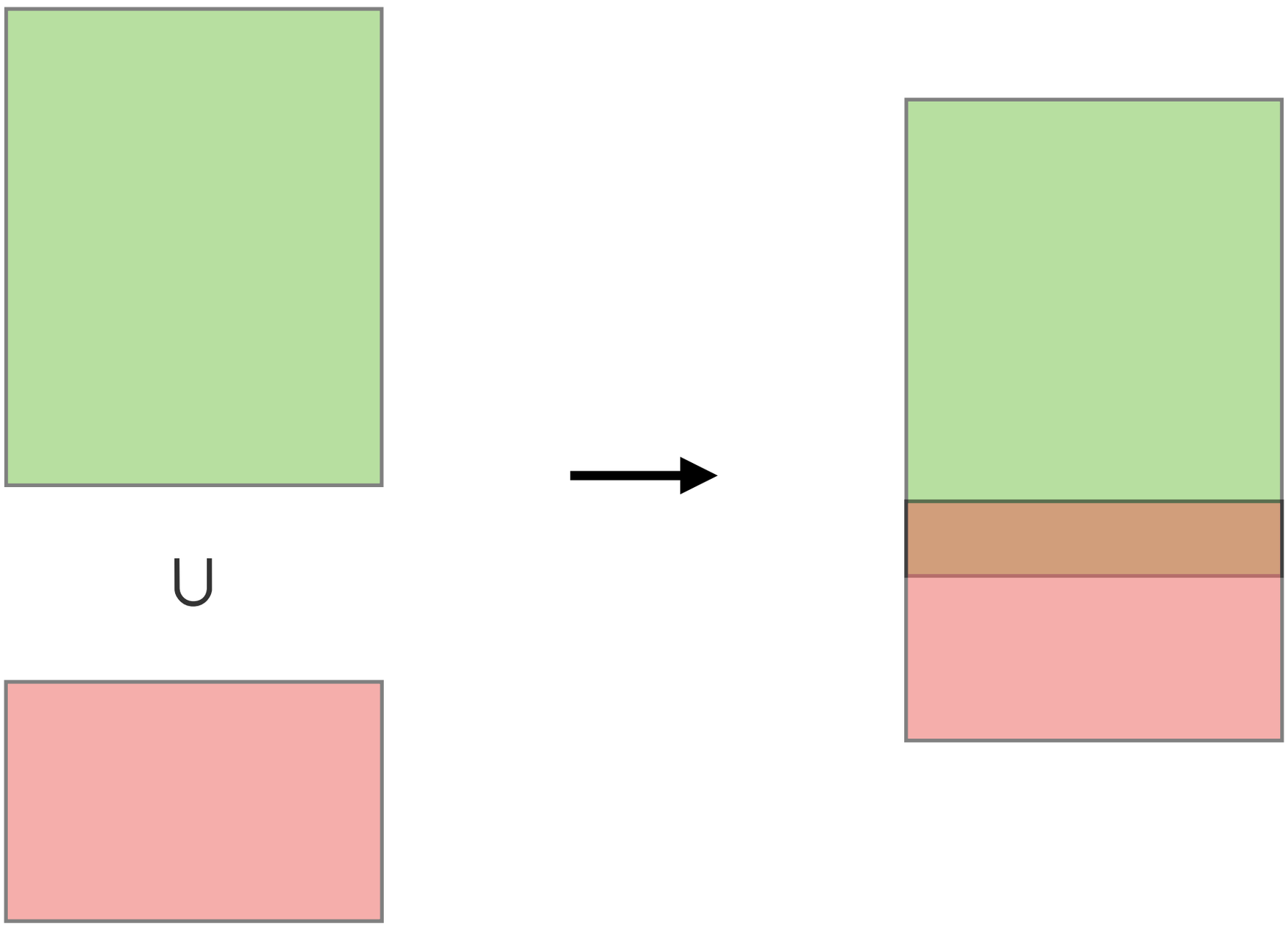
Implicit Join condition in WHERE

```
SELECT R.ENO, R.POINTS
FROM   STUDENTS S, RESULTS R
WHERE  S.LAST = 'Smith'
```

Forgot Join condition what happens?

Set Operations

Vertical Combination of Tables



Set Operations

- Compare content of two tables and produce result by comparing all attributes
  - `UNION` / `UNION ALL`: Combine two independent tables
  - `EXCEPT`: Rows from the first table except rows found is in the second table
  - `INTERSECT`: Only rows appearing in both tables
- Number of columns and types have to match between tables

Combining Results

First Query

```
SELECT SID, CAT
FROM   RESULTS
WHERE  SID = 103
```

Set Operation

```
UNION ALL
```

```
SELECT ENO, POINTS
FROM   RESULTS
WHERE  POINTS >= 10
```

← Second Query

Columns need to match!

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	9
103	H	1	5
103	M	1	7

```
SELECT SID, CAT
FROM   RESULTS
WHERE  SID = 103
```

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	9
103	H	1	5
103	M	1	7

```
SELECT POINTS, CAT
FROM   RESULTS
WHERE  POINTS >= 10
```

?	?
103	H
103	M
10	H
12	M

UNION ALL

Results makes no sense...

Evaluation Order

```
4     SELECT   R.ENO, AVG(POINTS) AS AV
      FROM     STUDENTS S JOIN RESULTS R
1       ON     (S.SID = R.SID)
2     WHERE    S.LAST = 'Smith'
3     GROUP BY R.ENO
      ORDER BY AV
5     LIMIT    10
```

Conceptually, not actually

Subqueries

- Nested queries
  - Filters, Comparisons, `EXISTS` / `x IN`
  - Table-Producing, `FROM (SELECT ... )`
- Somewhat similar to functions in programming
- Can refer to attributes from outer query ("correlated")

Scalar Subqueries

"Get the exercise number for which the highest score was given"

No duplicates

```
SELECT DISTINCT ENO
FROM   RESULTS
WHERE  POINTS = 12
```

Hard-Coded Constant :(

```
SELECT DISTINCT ENO
FROM   RESULTS
WHERE  POINTS = (
   SELECT MAX(POINTS) FROM RESULTS )
```

Scalar Subquery

Correlated Subqueries

"Get the students without results"

```
SELECT FIRST, LAST
FROM STUDENTS
WHERE NOT EXISTS (
   SELECT *
   FROM RESULTS
   WHERE RESULTS.SID = STUDENTS.SID )
```

Set Subquery

Column from outer query!

```sql
SELECT FIRST, LAST
FROM STUDENTS
WHERE NOT EXISTS (
   SELECT *
   FROM RESULTS
   WHERE RESULTS.SID = STUDENTS.SID )
```

Subquery is re-run for every row in STUDENTS*

```
NOT EXISTS (SELECT * FROM RESULTS WHERE RESULTS.SID = 101) → False
NOT EXISTS (SELECT * FROM RESULTS WHERE RESULTS.SID = 102) → False
NOT EXISTS (SELECT * FROM RESULTS WHERE RESULTS.SID = 103) → False
NOT EXISTS (SELECT * FROM RESULTS WHERE RESULTS.SID = 104) → True
```

Students

SID	FIRST	LAST
101	Ann	Smith
102	Michael	Jones
103	Richard	Turner
104	Maria	Brown

Results

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	9
103	H	1	5
103	M	1	7

Table-Producing Subqueries

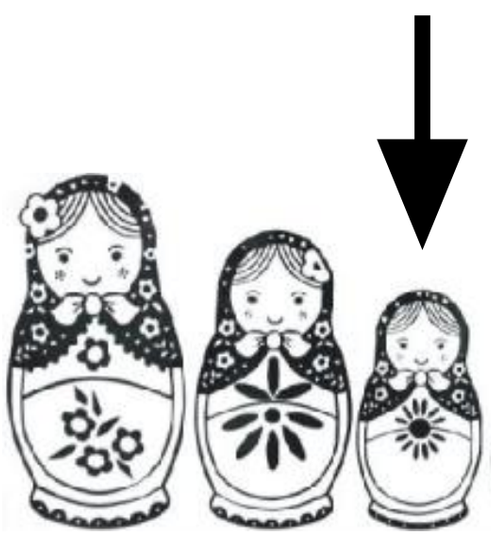
```
SELECT   ENO, AVG(POINTS) AS AV
FROM     STUDENTS JOIN RESULTS USING(SID)
WHERE    LAST = 'Smith'
GROUP BY ENO
ORDER BY AV LIMIT 10
```

```
SELECT ENO, AVG(POINTS) AS AV FROM (
  SELECT ENO, POINTS FROM (
    SELECT LAST, ENO, POINTS
    FROM STUDENTS S JOIN RESULTS R USING (SID))
  WHERE LAST='Smith')
GROUP BY ENO ORDER BY AV LIMIT 10
```

ProTip

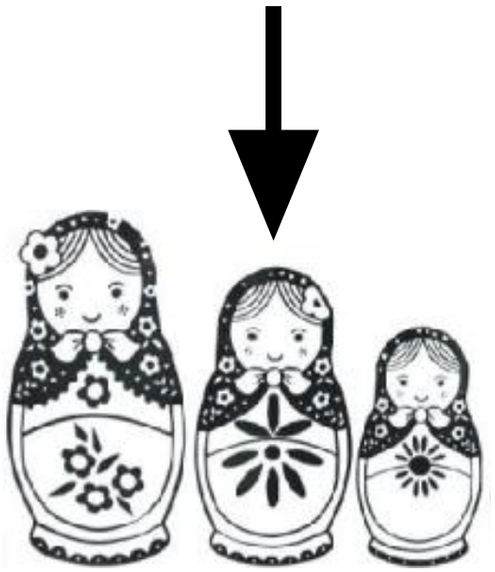
```
SELECT ENO, AVG(POINTS) AS AV FROM (
  SELECT ENO, POINTS FROM (
    SELECT LAST, ENO, POINTS
    FROM STUDENTS S JOIN RESULTS R USING (SID))
  WHERE LAST='Smith')
GROUP BY ENO ORDER BY AV LIMIT 10
```

LAST	ENO	POINTS
Smith	1	10
Smith	2	8
Smith	1	12
Jones	1	9
Jones	2	9
Jones	1	9
Turner	1	5
Turner	1	7



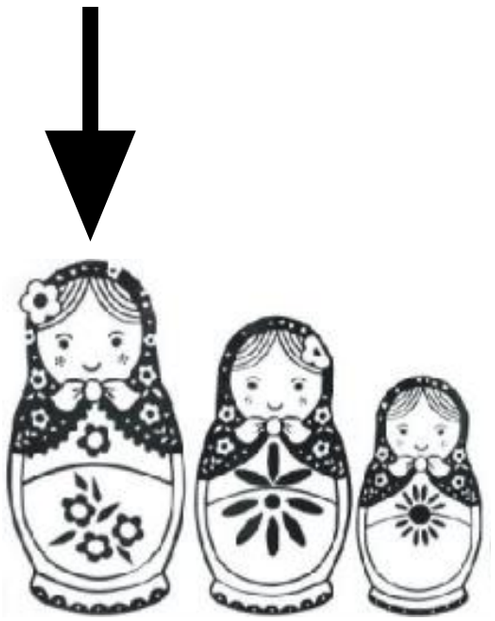
```
SELECT ENO, AVG(POINTS) AS AV FROM (
  SELECT ENO, POINTS FROM (
    SELECT LAST, ENO, POINTS
    FROM STUDENTS S JOIN RESULTS R USING (SID))
  WHERE LAST='Smith')
GROUP BY ENO ORDER BY AV LIMIT 10
```

ENO	POINTS
1	10
2	8
1	12



```
SELECT ENO, AVG(POINTS) AS AV FROM (
  SELECT ENO, POINTS FROM (
    SELECT LAST, ENO, POINTS
    FROM STUDENTS S JOIN RESULTS R USING (SID))
  WHERE LAST='Smith')
GROUP BY ENO ORDER BY AV LIMIT 10
```

ENO	AV
2	8
1	11



There's more

- Window Functions (!)
- Common Table Expressions
- Data cube operators
- User-Defined Functions
- ...
- Random SQL Questions?

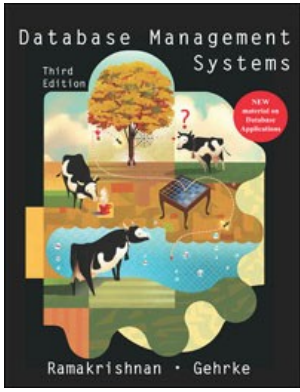
Query Construction

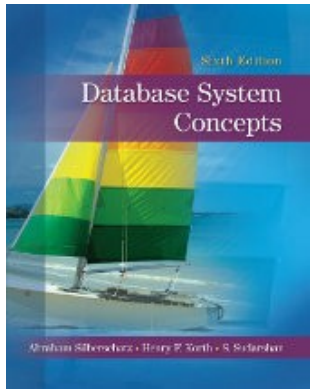
- Given a problem and some tables, construct the SQL query that answers them
  - Sometimes impossible, tables do not contain enough information
- Multiple steps:
  - Collect tables and join if needed
  - Aggregate if needed
  - Project
- Tip: Make virtual intermediates explicit

SQL - Sample Question

- Consider the following schema:
- `SCAN(DATE, HOUR, LICENSE_PLATE)`
- `CARS(LICENSE_PLATE, MODEL, IS_TAXI)`
- Find the most popular car models for taxis
- Count the number of distinct taxi license plates for every hour over all days

ADM: Expected Background





- **Database systems (e.g.):**
  - *Ramakrishnan*, *Gehrke*: **Database Management Systems (3rd International Edition)**, McGraw-Hill, 2003 (ISBN 0-07-246563-8)
  - *A. Silberschatz*, *H. F. Korth*, *S. Sudarshan:* **Database System Concepts (7th Edition),** McGraw-Hill, 2010 (ISBN 0-07-352332-1)

    book: https://www.db-book.com/

    slides: https://www.db-book.com/slides-dir/index.html
  - *Andy Pavlo:* **Introduction to Database Systems** course @ CMU incl. slides and videos on YouTube: https://15445.courses.cs.cmu.edu/fall2019/

ADM: Agenda

- 07.09.2022: Lecture 1: **Introduction**
- 14.09.2022: Lecture 2: **SQL Recap**

  (plus Assignment 1 [in groups; 3 weeks]: TPC-H benchmark)
- 21.09.2022: Lecture 3: **Column-Oriented Database Systems (1/6) - Motivation & Basic Concepts**
- 28.09.2022: Lecture 4: **Column-Oriented Database Systems (2a/6) - Selected Execution Techniques (1/2)**
- 05.10.2022: Lecture 5: **Column-Oriented Database Systems (2b/6) - Selected Execution Techniques (2/2)**

  (plus Assignment 3 [in groups; 3 weeks]: Compression techniques)
- 12.10.2022: Lecture 6: **Column-Oriented Database Systems (3/6) - Cache Conscious Joins**
- 19.10.2022: Lecture 7: **Column-Oriented Database Systems (4/6) - "Vectorized Execution"**
- ~~26.10.2022~~: ***No lecture!***
- 02.11.2022: Lecture 8: **DuckDB: An embedded database for data science (1/2) (guest lecture & *hands-on*)**

  (plus Assignment 2 [individual; 2 weeks]: Analysing NYC Cab dataset with DuckDB)
- 09.11.2022: Lecture 9: **DuckDB: An embedded database for data science (2/2) (guest lecture & *hands-on*)**
- 16.11.2022: Lecture 10: **Branch Misprediction & Predication**

  (plus Assignment 4 [individual; 2 weeks]: Predication)
- 23.11.2022: Lecture 11: **Column-Oriented Database Systems (5/6) - Adaptive Indexing**
- 30.11.2022: Lecture 12: **Column-Oriented Database Systems (6/6) - Progressive Indexing**

ADM: Literature (1/2)

- **Column-Oriented Database Systems (1/6) - Motivation & Basic Concepts**
  - “An overview of cantor: a new system for data analysis“. Ilkka Karasalo, Per Svensson. SSDBM 1983.
  - “A decomposition storage model“. George P. Copeland, Setrag Khoshafian. SIGMOD Conference, 1985.
  - “Cache Conscious Algorithms for Relational Query Processing“. Ambuj Shatdal, Chander Kant, Jeffrey F. Naughton. VLDB 1994.
  - “MIL Primitives for Querying a Fragmented World“. Peter A. Boncz, Martin L. Kersten. VLDB J. 8(2): 101-119, 1999.
  - “Database Architecture Optimized for the New Bottleneck: Memory Access“. Peter A. Boncz, Stefan Manegold, Martin L. Kersten. VLDB 1999.
  - “DBMSs On A Modern Processor: Where Does Time Go?“. Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, David A. Wood. VLDB 1999.
  - “Weaving Relations for Cache Performance ("PAX")“. Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, Marios Skounakis. VLDB 2001.
  - “A Case for Fractured Mirrors“. Ravishankar Ramamurthy, David J. DeWitt, Qi Su. VLDB 2002.
  - “Data Morphing: An Adaptive, Cache-Conscious Storage Technique“. Richard A. Hankins, Jignesh M. Patel. VLDB 2003.
  - “Clotho: Decoupling Memory Page Layout from Storage Organization“. Minglong Shao, Jiri Schindler, Steven W. Schlosser, Anastassia Ailamaki, Gregory R. Ganger. VLDB 2004.
  - “MonetDB-X100 - A DBMS In The CPU Cache“. Marcin Zukowski, Peter A. Boncz, Niels Nes, Sándor Héman. IEEE Data Eng. Bull. 28(2): 17-22, 2005.
  - “"One size fits all": an idea whose time has come and gone“. Michael Stonebraker, Ugur Çetintemel. ICDE 2005.
  - “Performance Tradeoffs in Read-Optimized Databases“. Stavros Harizopoulos, Velen Liang, Daniel J. Abadi, Samuel Madden. VLDB 2006.
  - …

ADM: Literature (2/2)

- **Column-Oriented Database Systems (1/6) - Motivation & Basic Concepts** *(cont.)*
  - …
  - “One Size Fits All? - Part 2: Benchmarking Results“. Michael Stonebraker, Chuck Bear, Ugur Çetintemel, Mitch Cherniack, Tingjian Ge, Nabil Hachem, Stavros Harizopoulos, John Lifter, Jennie Rogers, Stanley B. Zdonik. CIDR 2007.
  - “C-Store: A Column-oriented DBMS“. Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, Stanley B. Zdonik. VLDB 2005.
  - “Breaking the memory wall in MonetDB“. Peter A. Boncz, Martin L. Kersten, Stefan Manegold. Commun. ACM 51(12): 77-85, 2008.
  - “Column-Stores vs Row-Stores: How Different are They Really?“. Daniel J. Abadi, Samuel Madden, Nabil Hachem. SIGMOD Conference 2008.
  - “DSM vs. NSM: CPU performance tradeoffs in block-oriented query processing“. Marcin Zukowski, Niels Nes, Peter A. Boncz. DaMoN 2008.
  - “Fast Scans and Joins Using Flash Drives“. Mehul A. Shah, Stavros Harizopoulos, Janet L. Wiener, Goetz Graefe. DaMoN 2008.
  - “Read-Optimized Databases, In-Depth“. Allison L. Holloway, David J. DeWitt. Proc. VLDB Endow. 1(1): 502-513, 2008.
  - “Teaching an Old Elephant New Tricks“. Nicolas Bruno. CIDR 2009.
  - “Query Processing Techniques for Solid State Drives“. Dimitris Tsirogiannis, Stavros Harizopoulos, Mehul A. Shah, Janet L. Wiener, Goetz Graefe. SIGMOD Conference 2009.
  - “MonetDB: Two Decades of Research in Column-oriented Database Architectures“. Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, K. Sjoerd Mullender, Martin L. Kersten. IEEE Data Eng. Bull. 35(1): 40-45, 2012.
  - “The Vertica Analytic Database: C-Store 7 Years Later“. Andrew Lamb, Matt Fuller, Ramakrishna Varadarajan, Nga Tran, Ben Vandiver, Lyric Doshi, Chuck Bear. Proc. VLDB Endow. 5(12): 1790-1801, 2012.