Basic SQL

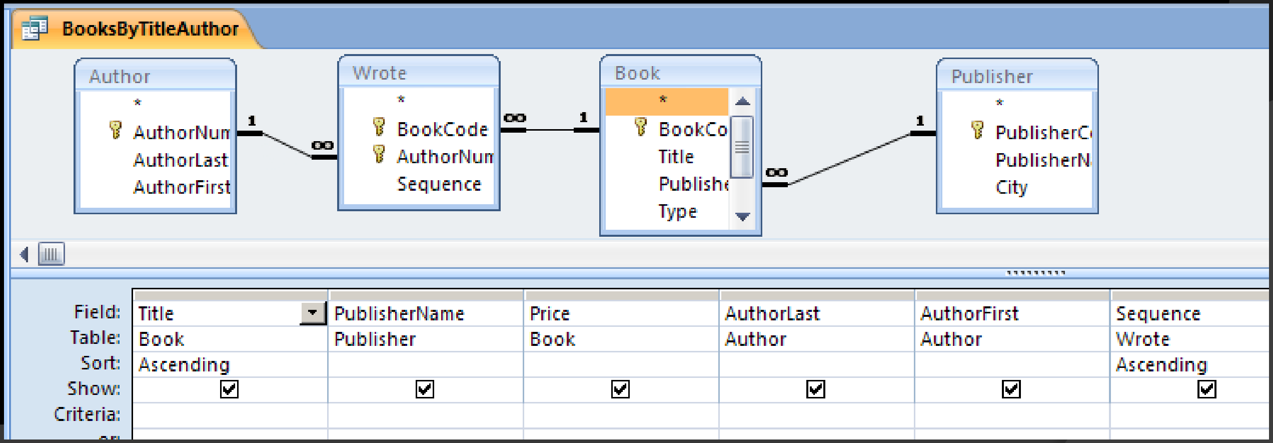
Relational Database Languages

Tuple relational calculus

- ALPHA (Codd, 1970s)
- QUEL (based on ALPHA)
- Datalog (rule-based, like PROLOG)

Domain relational calculus

- QBE (used in Access)



Field:	Title	PublisherName	Price	AuthorLast	AuthorFirst	Sequence
Table:	Book	Publisher	Book	Author	Author	Wrote
Sort:	Ascending					Ascending
Show:	✓	✓	✓	✓	✓	✓
Criteria:						

History of SQL

Standards:
- SQL-86 (ANSI, ISO)
- SQL-89
- SQL-92 (SQL2)
- SQL:1999 (SQL3)
- SQL:2003
- SQL:2006
- SQL:2008
- SQL:2011

IBM developed SEQUEL in early 70s
(Structured English Query Language)
Renamed SQL
(Structured Query Language)

Many different flavors of SQL:
sqlplus, SQLServer, MySQL, etc.

Here: Oracle SQL

Parts of SQL

DDL

Creating (CREATE), Modifying (ALTER), and Removing (DROP)

Catalogs
Schemas
Relations (Tables)
Constraints
Domains
Triggers

DML

Retrieving (SELECT), Inserting (INSERT), Modifying (UPDATE), and Removing (DELETE).

DCL

Grant privileges (GRANT)
Revoke privileges (REVOKE)

SELECT

Combination of select and project operations.

Basic Syntax:

SELECT *attribute_list*
FROM *table_list*
WHERE *condition*
GROUP BY *attribute_list*
HAVING *condition*
ORDER BY *attribute_list*;

Example:

```
SELECT      LastName, SID
FROM        student
WHERE       career = 'UGRD' AND
            started < 2008;
```

SELECT * and empty WHERE

```
SELECT *
FROM student;
```

```
SELECT *
FROM student, studentgroup;
```

```
SELECT *
FROM student, studentgroup
WHERE presidentID = SID
```

SELECT Examples

University

- List the SSNs of all students
- List all names of all student groups
- List the names of all programs
- List the name of all students in the IT department

Duplicates in SQL

Duplicates can occur if no key attribute is selected

SQL keeps duplicates, for several reasons:

- Cheaper to implement
  (duplicate elimination is expensive)
- Duplicates might be required
  (e.g., aggregate functions)

Removal of duplicates can be forced using DISTINCT

SELECT DISTINCT SELECT ALL

Example:

List cities in which students live (university).

Excursion:
How to write SQL Queries

How to Write Simple SQL Query

3 Stages

Before you write the SQL

Writing the SQL

After Writing the SQL

Before you write the SQL

Make sure you **understand** the problem.

    Clarify if necessary

Do it by **hand**.

    Reflect on what you did.

Writing the SQL

Start with **FROM**

which tables are involved, how often?

Then do **WHERE**

first join all tables (n tables need n-1 equals) - test

then add particular conditions

Finally, do **SELECT**

What info do you need to display

After Writing the SQL

- Test
  - Run the query
  - Compare output to what you expected
    - sanity check: does it make sense?
  - In case of problems: read query

End of Excursion

Renaming (Aliases)

If necessary we can rename attributes:

```
SELECT LastName || ', ' || FirstName AS Name
FROM student;
```

and tables (drop AS in Oracle for table renaming)

```
SELECT S.Lastname, SG.Name
FROM student AS S, studentgroup AS SG
WHERE S.SID = SG.PresidentID;
```

Renaming Examples

University

- List the names of all students and expected graduation year
- List the names of student groups and the names of their presidents

Conditions

=	Equality
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
<> or !=	not equal (depends on system)
LIKE	Allows Wildcards (Oracle)
	% (any number of characters)
	_ (single character)

Also (Oracle): REGEXP_LIKE (text, pattern)

Operators

+	Addition (works for dates in Oracle/Access)
-	Subtraction (works for dates in Oracle/Access)
*	Multiplication
/	Division
||	Concatenation (for strings)
and, or, not	Boolean operations

Functions (Oracle) Strings and Numbers

String type:

Length(s)	Length of string s
Rtrim(s), Ltrim(s)	Delete trailing (leading) spaces

Numeric type:

Round(x)	Round x to integer
Round(x,k)	Round x to k positions
Abs(x)	Absolute value of x
Exp(x)	e^x

See http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/functions.htm#i1482196

Functions (Oracle) Date/Timestamp

extract	date to integers
last_day	last day of month
months_between	months between two dates
to_char(date, fmt)	format date (e.g. MM, MON, MONTH)

```
SELECT extract (year from current_date)
FROM dual;
```

Between and Ordering

For number types, ranges can be defined using BETWEEN.

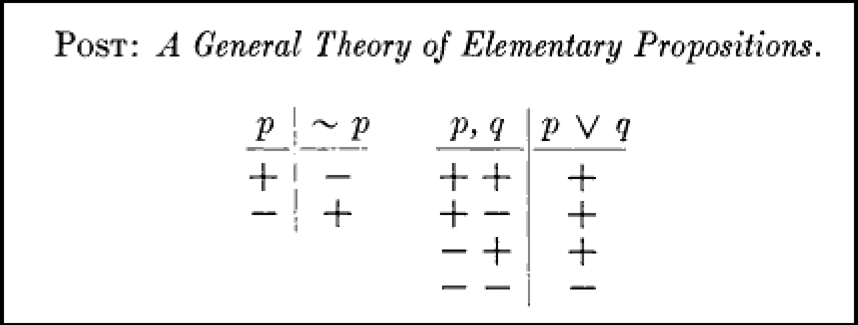
```
SELECT LastName, SID
FROM student
WHERE started between 2001 and 2003;
```

The ORDER BY clauses allows ordered output
- multiple attributes, expressions allowed
- ASC (default) and DESC to specify order

```
SELECT LastName, SID
FROM student
ORDER BY started ;
```

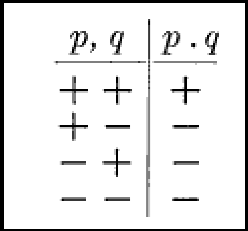
```
SELECT LastName, SID
FROM student
ORDER BY started  DESC, LastName;
```

Truth and Tables


Post: *A General Theory of Elementary Propositions.*

p	$\sim p$
+	−
−	+

p, q	$p \vee q$
+ +	+
+ −	+
− +	+
− −	−



p, q	$p \cdot q$
+ +	+
+ −	−
− +	−
− −	−

Definition: A function is a *truth-function* iff its truth only depends on the truth of its arguments.

Theorem (Post): Every truth-function can be written using only negation and disjunction. (Or negation and conjunction.)

Logic examples I

• Students in computer science and information systems
• Graduate students unless they are PhDs.
• List all graduate students in computer science, computer gaming and information systems.
• List graduate and undergraduate students not from Chicago that started before 2010 and after 2012.
• List courses that violate the rule: All gaming courses must be 300 or above, all computer science courses must be 400 or above and IT courses must be at most 200 level.
• Check whether there are students that violate the following rules: Computer gaming students must be undergraduates, information system students must be graduates.
• List students that violate the rule: A PhD student cannot be an undergraduate.

Logic examples II

- List students that have both a graduate and an undergraduate record (go by name).
- List courses that have both grad and undergrad versions (go by title).
- List students that don’t have a SSN listed.
- List years in which no undergraduate computer science student started.

Logic examples II

- List students that have both a graduate and an undergraduate record (go by name).
- List courses that have both grad and undergrad versions (go by title).
- List students that don’t have a SSN listed.
- List years in which no undergraduate computer science student started.

Some of these we can’t do yet (4), for some we need additional knowledge (3), and some we can do (1,2) but there must be better solutions.

Nulls

Reasons:

- we don’t know value
- value isn’t applicable
- we don’t know whether value is applicable

Null is not a value (well), to test for it use

is null

is not null

Examples:

- List students that don’t have a SSN listed.
- List studentgroups that don’t currently have a president.
- List students that do have a SSN listed.
- We require that students that don’t have a last name do enter a first name. List students that violate this requirement.

Nulls in Conditions

Example

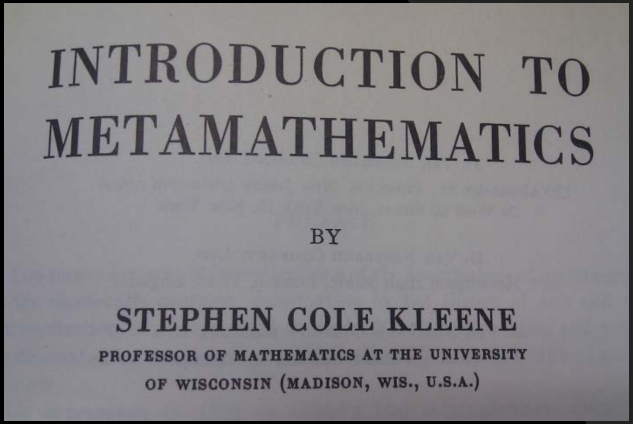
```
SELECT * FROM student WHERE SSN = 123123123;
SELECT * FROM student WHERE NOT(SSN = 123123123);
```

?

Three-valued Logic

p	*q*	*p* OR *q*	*p* AND *q*
True	True	True	True
True	Unknown	True	Unknown
True	False	True	False
Unknown	True	True	Unknown
Unknown	Unknown	Unknown	Unknown
Unknown	False	Unknown	False
False	True	True	False
False	Unknown	Unknown	False
False	False	False	False

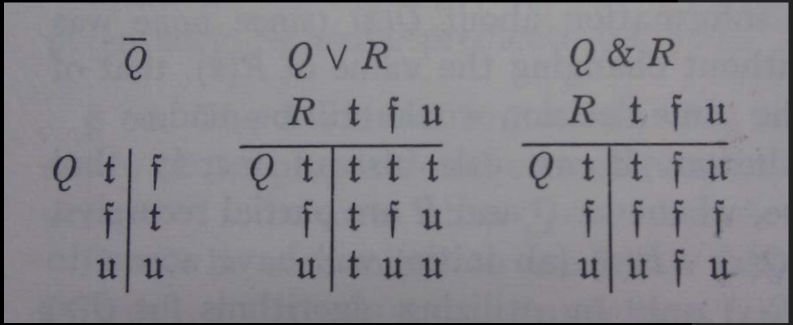
p	NOT p
True	False
Unknown	Unknown
False	True


INTRODUCTION TO METAMATHEMATICS

BY

STEPHEN COLE KLEENE

PROFESSOR OF MATHEMATICS AT THE UNIVERSITY OF WISCONSIN (MADISON, WIS., U.S.A.)


Q	$\overline{Q}$
t	f
f	t
u	u

$Q \vee R$	R: t	f	u
Q: t	t	t	t
f	t	f	u
u	t	u	u

$Q \& R$	R: t	f	u
Q: t	t	f	u
f	f	f	f
u	u	f	u

WHERE conditions: have to be true
CHECK constraints: can't be false

Nulls in Expressions

Null in Operations

null ○ x = null   (○ operation like +, -, *, /, etc.)

```
SELECT EmpID, salary * (1 + level * .05)
FROM employee;
```

Null in Functions

f(…, null, …) = null (for most functions f, not all, e.g. ||)

```
SELECT least(price_new, price_used)
FROM bookwprices;
```

Solution: coalesce(x, y, z, …) evaluates to first non-null value

Objections to Nulls

• we don't know what the Null value represents

Codd suggested two values:

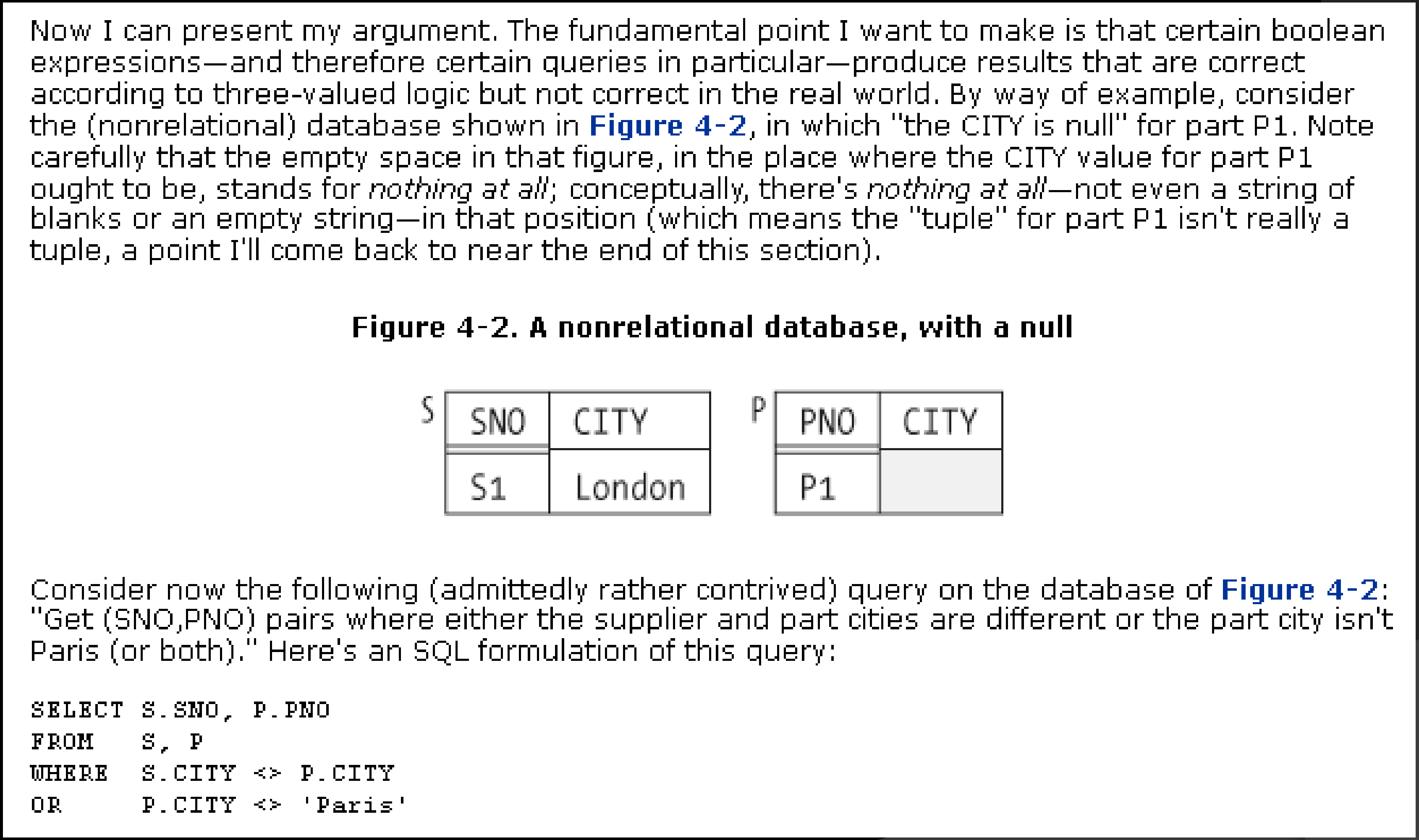
Missing Applicable/Missing Inapplicable

• 3-valued behavior is counterintuitive and leads to SQL programs behaving incorrectly in presence of Nulls

• wastes space (needs extra indicator bit)

E.g., see the Third Manifesto: http://en.wikipedia.org/wiki/The_Third_Manifesto

Date’s Example



Now I can present my argument. The fundamental point I want to make is that certain boolean expressions—and therefore certain queries in particular—produce results that are correct according to three-valued logic but not correct in the real world. By way of example, consider the (nonrelational) database shown in **Figure 4-2**, in which "the CITY is null" for part P1. Note carefully that the empty space in that figure, in the place where the CITY value for part P1 ought to be, stands for *nothing at all*; conceptually, there's *nothing at all*—not even a string of blanks or an empty string—in that position (which means the "tuple" for part P1 isn't really a tuple, a point I'll come back to near the end of this section).

Figure 4-2. A nonrelational database, with a null

S

SNO	CITY
S1	London

P

PNO	CITY
P1	

Consider now the following (admittedly rather contrived) query on the database of **Figure 4-2**: "Get (SNO,PNO) pairs where either the supplier and part cities are different or the part city isn't Paris (or both)." Here's an SQL formulation of this query:

```
SELECT S.SNO, P.PNO
FROM   S, P
WHERE  S.CITY <> P.CITY
OR     P.CITY <> 'Paris'
```

From Chris Date: SQL and Relational Theory, O’Reilly, 2009.