INFOMDW: Data Wrangling

Data Extraction (SQL)

Hakim Qahtan

Department of Information and Computing Sciences

Utrecht University

Utrecht University

Outlines

- So Far
  - Data types
    - Structured, semi-structured and unstructured
  - Data Models
    - Relational model and entity-relationship model
    - Graphs and trees
    - Objects
  - Data model components
    - Data (structure), semantics/constraints, and operations
  - Databases vs file systems
  - DDL and DML
  - Creating and modifying tables (relations) in SQL

Outlines

- Today
  - Reminder
    - Office hours only by appointment (send email)

  - Operations on databases
    - Relational algebra
    - Data Extraction with SQL

Reading Material for Today

Database System Concepts (7th Edition)

CH 2.6, 3.3 – 3.8.

Relational Algebra

Relational Algebra Operators

- Union $\cup$, intersection $\cap$, difference $-$
- Selection σ
- Projection π
- Join $\bowtie$
- Rename ρ

RA

- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ

Extended RA

Remember: a relation is a **set** of records

Union

$r \cup s$

r

A	B
α	3
β	2

s

A	B
α	1
β	2
γ	3

$r \cup s$

A	B
α	1
β	2
γ	3
α	3

Intersection

$$r \cap s$$

r

A	B
α	3
β	2

s

A	B
α	1
β	2
γ	3

$r \cap s$

A	B
β	2

Difference

$$r - s$$

r

A	B
α	3
β	2

s

A	B
α	1
β	2
γ	3

$r - s$

A	B
α	3

Can you find an expression that is equivalent to $r - s$ using the operators $\cap$ and $\sim$?

Selection

r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{(A=B)\ \wedge(D>5)}\ (r)$

A	B	C	D
α	α	1	7
β	β	23	10

Projection

r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\pi_{A,C}(r)$

A	C
α	1
α	5
β	12
β	23

Join

Cartesian Product

r

A	B
α	3
β	2

s

C	D
α	1
β	2
γ	3

$r \times s$

A	B	C	D
α	3	α	1
α	3	β	2
α	3	γ	3
β	2	α	1
β	2	β	2
β	2	γ	3

Join

Cartesian Product (attributes having the same name)

r

A	B
α	3
β	2

s

A	C
α	1
β	2
γ	3

$r \times s$

r.A	B	s.A	C
α	3	α	1
α	3	β	2
α	3	γ	3
β	2	α	1
β	2	β	2
β	2	γ	3

Rename

Rename operation $\rho_x(E)$ returns the output of the expression E under the name x

r

A	B
α	3
β	2

$r \times \rho_s(r)$

r.A	r.B	s.A	s.B
α	3	α	3
α	3	β	2
β	2	α	3
β	2	β	2

Natural Join

- Let r and s be relations on schemas R and S, respectively.
- Natural join of relations R and S is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from r and t_s from s.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t to the result, where
    - t has the same values as t_r on r
    - t has the same values as t_s on s

Natural Join – Example

r

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

s

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ϵ

$r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

$$r \bowtie s \equiv \pi_{A,r.B,C,r.D,E}(\sigma_{r.B=s.B \land r.D=s.D}(r \times s))$$

Utrecht University

Composition of Operations

r

A	B
α	3
β	2

s

A	B
α	1
β	2
γ	3

$r \times s$

r.A	r.B	s.A	s.B
α	3	α	1
α	3	β	2
α	3	γ	3
β	2	α	1
β	2	β	2
β	2	γ	3

$\sigma_{r.A=s.A}\,(r \times s)$

r.A	r.B	s.A	s.B
α	3	α	1
β	2	β	2

Is composition of operations possible and why?

Summary of Relational Algebra Operators

Symbol (name)	Description
σ (Selection)	Return rows of the input relation that satisfy a predicate.
π (Projection)	Return specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
$\times$ (Cartesian Product)	Return pairs of rows from the *two* input relations.
$\cup$ (Union)	Return the union of tuples from the *two* input relations.
$-$ (Difference)	Return the set of records that exist in the relation before the operator(-) but not in the relation after the operator.
$\cap$ (Intersection)	Return the common tuples in both input relations.
$\bowtie$ (Natural Join)	Return pairs of rows from the *two* input relations that have the same values on all attributes that have the same name.
ρ (Rename)	Returns the outcome of an expression under the specified name

Remarks on RA

- Each Query input is a table (or set of tables)
- Each query output is a table.
- All data in the output table appears in one of the input tables
- Computations with RA:
  - SUM $\gamma_{SUM(tot\_credit)}(student)$
  - AVG $\gamma_{AVG(salary)}(instructor)$
  - MAX $\gamma_{MAX(buget)\rightarrow \mathrm{max}\_budget}(department)$
  - MIN $\gamma_{MIN(budget)\rightarrow \mathrm{min}\_budget}(department)$
  - COUNT $\gamma_{COUNT(tot\_cred\geq 12)\rightarrow high\_cred}(student)$
  - GROUP BY $_{dept\_name}\gamma_{AVG(salary)\rightarrow dept\_avg\_salary}(instructor)$

Utrecht University

Data Extraction Using Structured Query Language (SQL)

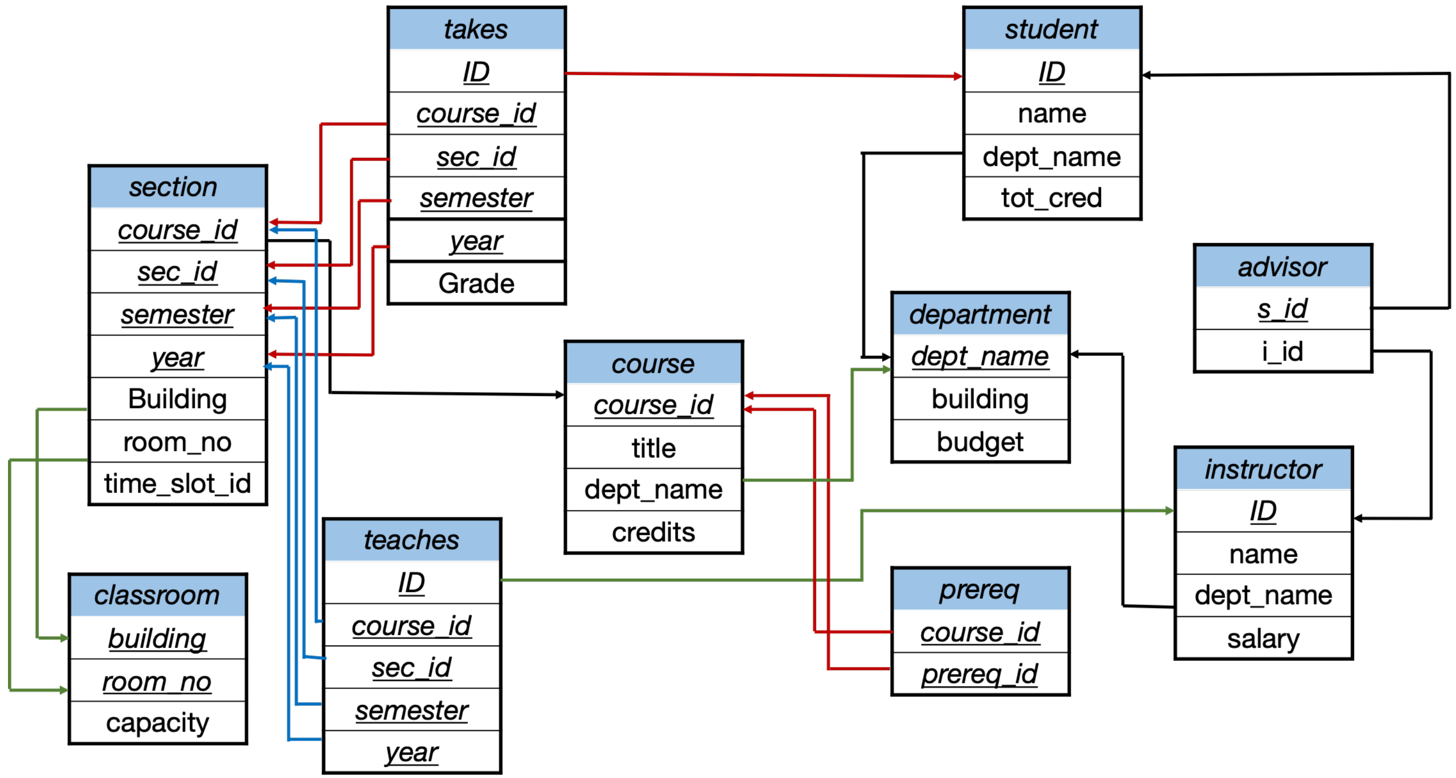
SELECT-FROM-WHERE Statements

SELECT desired attributes

FROM one or more tables

WHERE condition about tuples of the tables

Remember Our Main Database?



Example: The SELECT Clause

- Get the IDs, names and total credits of students who completed at least 24 credits

```
SELECT ID, name, tot_cred
FROM student
WHERE tot_cred >= 24;
```

- The answer is a relation with three attributes (ID, name, tot_cred)
- What if we need only the names of the students?
- Note that: the SQL operator names and keywords are case insensitive
  - e.g. SELECT ≡ Select ≡ select

The SELECT Clause

- Lists the desired attributes in the result of the query
  - Corresponds to the projection operator of the relational algebra
- SQL allows duplicates in relations as well as in query results
- To force the elimination of duplicates, use the keyword **DISTINCT** after select
- Example: find the department names of all instructors whose salary is strictly greater than 60000 without showing the department name more than once

```
SELECT DISTINCT dept_name
FROM instructor
WHERE salary > 60000
```

The SELECT Clause (Cont.)

- The keyword **ALL** specifies that duplicates should not be removed

```
SELECT ALL dept_name
FROM instructor
WHERE salary > 60000
```

- An asterisk in the **SELECT** clause denotes “all attributes”

```
SELECT *  FROM instructor
```

will return all the records from table “instructor”

The SELECT Clause (Cont.)

- An attribute could be a literal with no **FROM** clause

  SELECT ’542’

  Results in a relation with one column and one row with value “’542’

  We can also give the column a name using

  SELECT ’542’ AS V1

- An attribute could be a literal with **FROM** clause

  SELECT ‘A’ FROM instructor

  will return a relation with one column and N rows (the number of tuples in the ‘instructor ’ relation) where each row will contain the value “A”

The SELECT Clause (Cont.)

- **SELECT** clause can contain arithmetic expressions involving the operators *, +, -, and /.

  ```
  SELECT ID, name, salary/12.0
  FROM instructor
  ```

  This query would return a relation with the same number of records as the 'instructor' relation and the attributes (ID, name, salary/12.0), where the yearly salary is replaced by the monthly salary

  We can rename the "salary/12.0" using the **AS** clause

  ```
  SELECT ID, name, salary/12.0 as monthly_salary
  FROM instructor
  ```

The WHERE Clause

- Specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra
- To find all students enrolled in the 'Math.' department

```
SELECT ID, name  FROM student
 WHERE dept_name = 'Math.'
```

- Conditions can be also combined using logical operators (AND, OR, NOT)
  - Find all students in the 'Math.' department who completed a minimum of 24 credits

```
SELECT ID, name  FROM student
WHERE dept_name = 'Math.' AND tot_cred >= 24
```

- Comparisons =, <>, <, >, <=, >=

The FROM Clause

- Lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra
- Find the Cartesian product *’ instructor’ X ’ teaches’*

SELECT * FROM instructor, teaches

  - Generates every possible instructor–teaches pairs, with all attributes from both relations.
  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation’s name (e.g., *instructor.ID*)
    - Similar to the cartesian product in RA
- Cartesian product not very useful directly, but useful when combined with where-clause condition.

SELECT-FROM-WHERE Examples

SELECT name, course_id

FROM instructor, teaches

WHERE instructor.ID = teaches.ID

Returns the names of all instructors who have taught any courses and the course_id

SELECT name, course_id

FROM instructor, teaches

WHERE instructor.ID = teaches.ID AND instructor.dept_name = ‘Art’

Returns the names of all instructors in the Art department who have taught any courses and the course_id

The Rename Operation

- SQL allows renaming relations and attributes using the **AS** clause:

  old-name **AS** *new-name*

- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.

  SELECT DISTINCT *T.name, S.name*
  FROM *instructor* AS *T, instructor* AS *S*
  WHERE *T.salary >= 75000* AND *S.dept_name = 'Comp. Sci.'*

  Returns the names of instructors whose salary is greater than or equal 75000 joined with those in the 'Comp. Sci.' department

- Keyword **AS** is optional and may be omitted

  instructor **AS** *T ≡ instructor T*

Renaming Example: Self-Join

- Sometimes, a query needs to use two copies of the same relation.
- Distinguish copies by renaming the relations.
- Example self-Join

```
SELECT T.name AS N1, S.name AS N2
FROM instructor T, instructor S
WHERE T.salary = S.salary AND T.name < S.name
```

  - Returns the names of instructors who have the same salary
  - Do not produce pairs like (Miller, Miller)
  - Produces pairs in alphabetic order, e.g. (Addison, Miller), not (Miller, Addison)
- Note that we omit AS when renaming the relations

Join Operation

- **JOIN** operations take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the **FROM** clause

Join Operation (Cont.)

- We will consider the following relations in the few coming slides
- Relation *course*

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-490	Game Design	Comp. Sci.	4
CS-315	Boolean Algebra	Comp. Sci.	3

- Relation *prereq*

course_id	prereq_id
BIO-301	BIO-101
CS-490	CS-101
CS-347	CS-201

- Note that:
  - *prereq* information is missing for course CS-315
  - *course information is missing for course CS-347*

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values.

Left Outer Join

```
SELECT *
FROM course
LEFT OUTER JOIN prereq
ON course.course_id = prereq.course_id
```

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-490	Game Design	Comp. Sci.	4	CS-101
CS-315	Boolean Algebra	Comp. Sci.	3	*null*

Right Outer Join

SELECT *
FROM *course*

RIGHT OUTER JOIN prereq

ON course.course_id = prereq.course_id

course_id	prereq_id	title	dept_name	credits
BIO-301	BIO-101	Genetics	Biology	4
CS-490	CS-101	Game Design	Comp. Sci.	4
CS-347	CS-201	*null*	*null*	*null*

- Remember:
  - The order of the attributes in a relation has no meaning

Full Outer Join

```
SELECT *
FROM course
FULL OUTER JOIN prereq
ON course.course_id = prereq.course_id
```

course_id	prereq_id	title	dept_name	credits
BIO-301	BIO-101	Genetics	Biology	4
CS-490	CS-101	Game Design	Comp. Sci.	4
CS-347	CS-201	*null*	*null*	*null*
CS-315	*null*	Boolean Algebra	Comp. Sci.	3

Inner Join

```
SELECT *
FROM course
INNER JOIN prereq
ON course.course_id = prereq.course_id
```

course_id	prereq_id	title	dept_name	credits
BIO-301	BIO-101	Genetics	Biology	4
CS-490	CS-101	Game Design	Comp. Sci.	4

- An SQL INNER JOIN is same as JOIN clause
- Question:
  - What is the difference between the above JOIN and the right/left outer join

String Operations

- SQL includes a string-matching operator for comparisons on character strings.
- The operator **LIKE** uses patterns that are described using two special characters
  - Percent (%). The % character matches any substring.
  - Underscore (_). The _ character matches any character.
- Example: find the names of all instructors whose name includes the substring “Van der”

  SELECT DISTINCT *name*
  FROM *instructor* WHERE *name* LIKE ‘%Van der%’

String Operations (Cont.)

- Match the string “100%”

  LIKE ‘100 \%’ **ESCAPE** '\'

  in that above we use backslash (\) as the escape character
- Patterns are case sensitive.
- Pattern matching examples:
  - ‘Intro%’ matches any string beginning with “Intro”.
  - ‘%Comp%’ matches any string containing “Comp” as a substring.
  - ‘_ _ _’ matches any string of exactly three characters.
  - ‘_ _ _ %’ matches any string of at least three characters.

Range Queries

- SQL includes a **BETWEEN** comparison operator
- Example: Find the names of all instructors with salary between \$90,000 and \$100,000 (that is, $\geq$ \$90,000 and $\leq$ \$100,000)

```
SELECT name
FROM instructor
WHERE salary BETWEEN 90000 AND 100000
```

Tuple Comparison

```
SELECT name, course_id
FROM instructor, teaches
WHERE (instructor.ID, dept_name) = (teaches.ID, ‘Biology’)
```

Set Operations

- Find courses that ran in Fall 2009 or in Spring 2010

  SELECT *course_id* FROM *section* WHERE *sem = 'Fall'* AND *year = 2009*
  UNION
  SELECT *course_id* FROM *section* WHERE *sem = 'Spring'* AND *year = 2010*
- Find courses that ran in Fall 2009 and in Spring 2010

  SELECT *course_id* FROM *section* WHERE *sem = 'Fall'* AND *year = 2009*
  INTERSECT
  SELECT *course_id* FROM *section* WHERE *sem = 'Spring'* AND *year = 2010*
- Find courses that ran in Fall 2009 but not in Spring 2010

  SELECT *course_id* FROM *section* WHERE *sem = 'Fall'* AND *year = 2009*
  EXCEPT
  SELECT *course_id* FROM *section* WHERE *sem = 'Spring'* AND *year = 2010*

Null Values

- It is possible for tuples to have a null value, denoted by *NULL*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *NULL* is *NULL*
  - Example: 5 + *NULL* returns NULL
- The predicate **IS NULL** can be used to check for null values.
  - Example: Find all instructors whose salary is null.

```
SELECT name
FROM  instructor
WHERE salary IS NULL
```

Null Values and Three Valued Logic

- Three values – *true*, *false*, *unknown*
- Any comparison with *null* returns *unknown*
  - Example*: 5 < null   or   null <> null   or   null = null*
- Three-valued logic using the value *unknown*:
  - OR: (*unknown* **OR** *true*)   = *true*,
    (*unknown* **OR** *false*) = *unknown*
    (*unknown* **OR** *unknown*) = *unknown*
  - AND: *(true* **AND** *unknown) = unknown,*
    (false **AND** *unknown) = false,*
    (unknown **AND** *unknown) = unknown*
  - NOT*: (***NOT** *unknown) = unknown*
  - "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of **WHERE** clause predicate is treated as *false* if it evaluates to *unknown*

∧	***T***	***F***	***U***
T	T	F	U
F	F	F	F
U	U	F	U

∨	***T***	***F***	***U***
T	T	T	T
F	T	F	U
U	T	U	U

The IN Operator

- <v> **IN** <S> evaluates to true if the value v matches one of the values in S.
- It can be used to replace a sequence of conditions connected by **OR**
- **Example:**

  SELECT name
  FROM *instructor*
  WHERE *dept_name* IN ('Comp. Sci.', 'Math.', 'Chem.');

This Query is equivalent to:

  SELECT name
  FROM *instructor*
  WHERE *dept_name* = 'Comp. Sci.' OR *dept_name* = 'Math.' OR *dept_name* = 'Chem.'

Wrap-Up

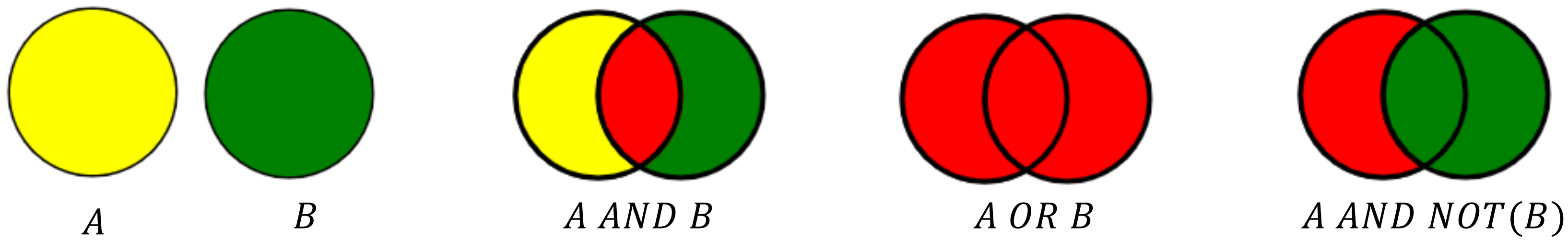
- Summarize what you learned today in 2-minutes

The information in this presentation has been compiled with the utmost care,
but no rights can be derived from its contents.

Boolean Operators – Revision

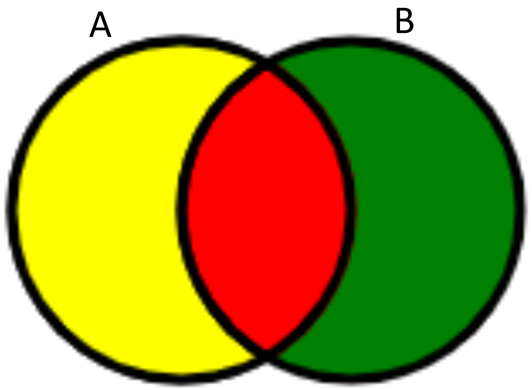
Boolean Operators

- Searching through a database or search engine can often be frustrating
- Boolean Operators create relationships between concepts and terms for better search results
- Most popular Boolean operators are **AND**, **OR** and **NOT**
  - The red areas represent the results of the operators



AND (∧)

A	B	$A \wedge B$
1	1	1
1	0	0
0	1	0
0	0	0



The red area – $(A \wedge B)$

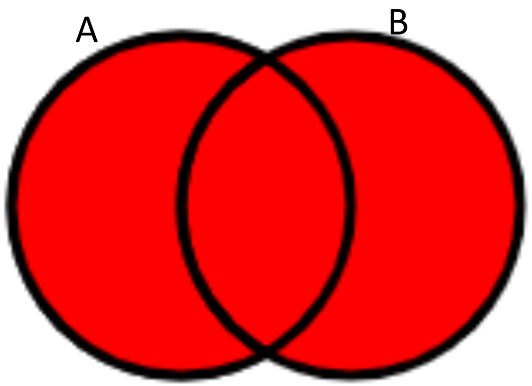
- Retrieves only records that satisfy both conditions
- Example:

  name = “Taylor” AND dept_name = “Chem.”

  Returns all instructors in the Chem. department whose name is Taylor

OR (∨)

A	B	$A \vee B$
1	1	1
1	0	1
0	1	1
0	0	0



The red area – $(A \vee B)$

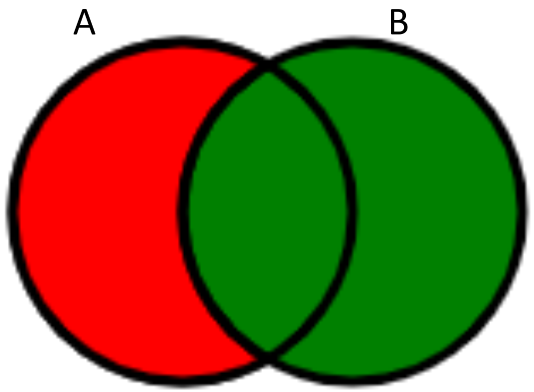
- Retrieves records that satisfy one of the conditions
- Example:

  name = “Taylor” OR dept_name = “Chem.”

  Returns all instructors with the name Taylor and all instructors of the Chem. department

NOT (~)

A	$\sim A$
1	0
0	1



The red area – $A\ AND\ NOT(B)$

- Retrieves only records that satisfy the first condition and doesn't satisfy the second
- Example:

  name = "Taylor" AND

  NOT dept_name = "Chem."

  Returns all instructors with the name Taylor who do not work in the Chem. department

Boolean Equivalence

- Equivalence of two Boolean operations can be easily proven using truth tables
- Equivalence of Boolean operations is useful for optimizing the Boolean queries
- Examples:
  - $\sim(\boldsymbol{A} \wedge \boldsymbol{B}) \equiv \sim \boldsymbol{A} \vee \sim \boldsymbol{B}$

$\boldsymbol{A}$	$\boldsymbol{B}$	$(\boldsymbol{A} \wedge \boldsymbol{B})$	$\sim(\boldsymbol{A} \wedge \boldsymbol{B})$	$\sim \boldsymbol{A} \vee \sim \boldsymbol{B}$
1	1	1	0	0
1	0	0	1	1
0	1	0	1	1
0	0	0	1	1

$\equiv$

  - $(\boldsymbol{A} \wedge \boldsymbol{B}) \wedge \sim \boldsymbol{B} \equiv \boldsymbol{A} \wedge (\boldsymbol{B} \wedge \sim \boldsymbol{B}) \equiv \boldsymbol{false}$
  - $(\boldsymbol{A} \wedge \boldsymbol{B}) \vee (\sim \boldsymbol{B} \wedge \boldsymbol{C}) \vee (\boldsymbol{A} \wedge \boldsymbol{C}) \vee \sim (\boldsymbol{A} \wedge \boldsymbol{C}) \equiv (\boldsymbol{A} \wedge \boldsymbol{C}) \vee \sim (\boldsymbol{A} \wedge \boldsymbol{C}) \equiv \boldsymbol{true}$

Proving Boolean Equivalence

- Truth Table: Helpful when the number of statements in the Boolean expression is small
- Proof by contradiction: assume the expression is false/true and proof that it leads to contradiction
- Using the Boolean axioms.

Boolean Axioms

- Let $\boldsymbol{T}$ and $\boldsymbol{F}$ represent the cases of always True and always False
  - $(\boldsymbol{T} \wedge \boldsymbol{T}) = \boldsymbol{T}$
  - $(\boldsymbol{F} \wedge \boldsymbol{F}) = \boldsymbol{F}$
  - $(\boldsymbol{T} \vee \boldsymbol{T}) = \boldsymbol{T}$
  - $(\boldsymbol{F} \vee \boldsymbol{F}) = \boldsymbol{F}$
  - $(\boldsymbol{T} \wedge \boldsymbol{F}) = (\boldsymbol{F} \wedge \boldsymbol{T}) = \boldsymbol{F}$
  - $(\boldsymbol{T} \vee \boldsymbol{F}) = (\boldsymbol{F} \vee \boldsymbol{T}) = \boldsymbol{T}$
  - $\overline{\boldsymbol{T}} = \boldsymbol{F}$
  - $\overline{\boldsymbol{F}} = \boldsymbol{T}$

Boolean Axioms

- Commutativity
  - $(\boldsymbol{A} \wedge \boldsymbol{B}) = (\boldsymbol{B} \wedge \boldsymbol{A})$
  - $(\boldsymbol{A} \vee \boldsymbol{B}) = (\boldsymbol{B} \vee \boldsymbol{A})$
- Identity
  - $(\boldsymbol{A} \wedge \boldsymbol{T}) = \boldsymbol{A}$
  - $(\boldsymbol{A} \vee \boldsymbol{T}) = \boldsymbol{T}$
  - $(\boldsymbol{A} \wedge \boldsymbol{F}) = \boldsymbol{F}$
  - $(\boldsymbol{A} \vee \boldsymbol{F}) = \boldsymbol{A}$

Boolean Axioms

- Idempotent
  - $(\boldsymbol{A} \wedge \boldsymbol{A}) = \boldsymbol{A}$
  - $(\boldsymbol{A} \vee \boldsymbol{A}) = \boldsymbol{A}$
- Involution
  - $\overline{\overline{\boldsymbol{A}}} = \boldsymbol{A}$
- Complement
  - $(\boldsymbol{A} \wedge \overline{\boldsymbol{A}}) = \boldsymbol{F}$
  - $(\boldsymbol{A} \vee \overline{\boldsymbol{A}}) = \boldsymbol{T}$

Boolean Axioms

- Associativity
  - $(\boldsymbol{A} \wedge \boldsymbol{B}) \wedge \boldsymbol{C} = \boldsymbol{A} \wedge (\boldsymbol{B} \wedge \boldsymbol{C})$
  - $(\boldsymbol{A} \vee \boldsymbol{B}) \vee \boldsymbol{C} = \boldsymbol{A} \vee (\boldsymbol{B} \vee \boldsymbol{C})$
- Distributivity
  - $\boldsymbol{A} \wedge (\boldsymbol{B} \vee \boldsymbol{C}) = (\boldsymbol{A} \wedge \boldsymbol{B}) \vee (\boldsymbol{A} \wedge \boldsymbol{C})$
  - $\boldsymbol{A} \vee (\boldsymbol{B} \wedge \boldsymbol{C}) = (\boldsymbol{A} \vee \boldsymbol{B}) \wedge (\boldsymbol{A} \vee \boldsymbol{C})$

Proving Boolean Equivalence

- Using the Boolean axioms (Example):
  - prove that: $(A \wedge B) \vee (\sim A \wedge C) \vee (B \wedge C) \; = (A \wedge B) \vee (\sim A \wedge C)$

  - LHS = $(A \wedge B) \vee (\sim A \wedge C) \vee (B \wedge C) = (A \wedge B) \vee (\sim A \wedge C) \vee [(B \wedge C) \wedge \boldsymbol{T}]$

$$= (A \wedge B) \vee (\sim A \wedge C) \vee [(B \wedge C) \wedge (A \vee {\sim}A)]$$
$$= (A \wedge B) \vee (\sim A \wedge C) \vee [(B \wedge C \wedge A) \vee (B \wedge C \wedge {\sim}A)]$$
$$= [(A \wedge B) \vee (B \wedge C \wedge A)] \vee [(\sim A \wedge C) \vee (B \wedge C \wedge {\sim}A)]$$
$$= [(A \wedge B) \wedge (\boldsymbol{T} \vee C)] \vee [(\sim A \wedge C) \wedge (\boldsymbol{T} \vee B)]$$
$$= [(A \wedge B)] \vee [(\sim A \wedge C)] \text{ = RHS}$$

  - $\boldsymbol{T}$ means always TRUE.

Utrecht University

Read more FROM

John Kelly, The Essence of Logic (Prentice Hall, 1997)

Chapter 1 is good enough