

# Databases

## Lecture 7 - Subqueries and Views

Emily Lucia Antosch

HAW Hamburg

15.04.2025

# Contents

- 1. Introduction ..... 2
- 2. Views ..... 6
- 3. Subqueries & Views ..... 64
- 4. License Notice ..... 107

# 1. Introduction

---

# 1.1 Where are we right now?

- Last time, we looked at the way we can normalize our database to improve our database design.
- Today, we'll be discussing
  - how you can work with subqueries and views,
  - how subqueries and views are powerful tools that you can use to design complex databases and
  - how stacking views can enable complex solutions.

# 1.1 Where are we right now?

1. Introduction
2. Basics
3. SQL
4. Entity-Relationship-Model
5. Relationships
6. Constraints
8. Subqueries & Views
9. Transactions
10. Database Applications
11. Integrity, Trigger & Security

## 1.2 What is the goal of this chapter?

- At the end of this lesson, you should be able to
  - create views in PostgreSQL and use them effectively
  - use subqueries in different points in the development of your database to enable your ideas and
  - know best practices and tricks that allow for complex data structures.

## 2. Views

---

## 2.1 Relational Algebra

### Overview

- While SQL is the what, Relational Algebra is the how!
- In mathematics an algebra is a values range combined with defined operations
- Relational Algebra: The values range is the content of the database; operations are functions to calculate the query results
  - a set of operations for the relational model
- Relational Calculus: Descriptive approach that is based on mathematical logic
  - higher-level declarative language for specifying relational queries, e.g., no order of operations, only what information the result should contain



## 2.1 Relational Algebra

### Overview

- Algebra operations produce new relations
- These can be further manipulated using operations of the same algebra
- Sequence of relational algebra operations: relational algebra expression
- The result of a relational algebra expression is also a relation representing the result of a database query (retrieval request)

## 2.1 Relational Algebra

### Overview

- Algebra operations can be divided into two groups
  - First group consists of operations developed specifically for relational databases
    - i.e., Selection, Projection, and Join
  - Second group includes set operations from mathematical set theory
    - i.e., Union, Intersection, Set Difference, and Cartesian Product

## 2.1 Relational Algebra

### Overview

- Order of explanation
  1. Selection
  2. Projection
  3. Renaming
  4. Union, Intersection, Set Difference
  5. Cartesian Product
  6. Join (Equijoin, Natural Join)

### Queries

SELECT – Basic form

SELECT	<attribute list>	→ Projection
FROM	<table list>	
WHERE	<condition>	→ Selection

- <attribute list> is a list of attribute names (columns) whose values are to be retrieved by the query
- <table list> is a list of the relation names (e.g., tables) required to process the query
- <condition>: optional conditional (Boolean) expression that identifies the tuples to be retrieved by the query

# 2.1 Relational Algebra

## Selection

- Selection( $\sigma$ ): mask out rows
  - Specify, which rows should remain (subset of the tuple)
    - Usage of selection: Specify, which tuples are interesting
    - Selection condition is a Boolean expression (condition)
    - The condition may contain complex expressions (combinations)
  - Specify, which relation is meant
    - Notice that  $R$  is generally a relational algebra expression whose result is a relation, e.g., a relation
  - Syntax:  $\sigma_{\text{selection condition}}(R)$



### Example

$\sigma_{\text{Salary} > 30,000}(\text{Employee}) \cup \sigma_{(\text{DNr}=4 \text{ AND } \text{Salary} > 30,000) \text{ OR } (\text{DNr}=5 \text{ AND } \text{Salary} > 25,000)}(\text{Employee})$

# 2.1 Relational Algebra

## Selection



### Example

### Selection

#### Info

- Selection is unary (apply to a single relation)
- The degree of the relation resulting from a Selection is the same as the degree of  $R$
- The number of tuples in the resulting relation is always less than or equal to the number of tuples in  $R$

## 2.1 Relational Algebra

### Selection

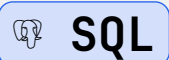
- Selection condition is typically specified in the `WHERE` clause of a SQL query



### Example

$\sigma_{\text{salary} > 30,000}(\text{Employee})$

```
1  SELECT * FROM Employee WHERE Salary > 30,000
```





## 2.1 Relational Algebra

### Projection

- Projection ( $\pi$ ): mask out columns
  - Specify, which columns should remain
  - Specify, which relation is meant
  - Syntax:  $\pi_{\text{attribute list}}(R)$



### Example

$\pi_{\text{SSN,lastName}}(\text{Person})$

### Projection

#### Info

- The degree of the result is equal to the number of attributes in attribute list
- If the attribute list includes only non-key attributes of  $R$ , duplicate tuples are likely to occur
  - The Projection removes any duplicate tuples, so, the result of the Projection is a set of distinct tuples, and hence a valid relation
- The number of tuples in a relation resulting from a Projection is always less than or equal to the number of tuples in  $R$

# 2.1 Relational Algebra

## Projection

The Projection attribute list is specified in the SELECT clause of a SQL query.



### Example

$\pi_{\text{LastName}}(\text{Person})$

1	SELECT
2	LASTNAME
3	FROM
4	PERSON



*Person*

<u>SSN</u>	Last Name	First Name	Mobile	...
123456789	Miller	Jane	0044 7701 123456	...
234567891	Miller	Steven	0044 7701 123457	...
345678912	Smith	Maria		...
...	...	...	...	...



$\sigma_{LastName="Miller"} (Person)$

<u>SSN</u>	Last Name	First Name	Mobile	...
123456789	Miller	Jane	0044 7701 123456	...
234567891	Miller	Steven	0044 7701 123457	...

## 2.1 Relational Algebra

### Renaming

- Renaming ( $\rho$ ): Column gets new name
  - Specify, which column
  - Specify, which new name
  - Specify, which relation
  - Set theory: Union ( $\cup$ ), Intersection ( $\cap$ ) and Set Difference ( $-$ ) are only defined for the same relation schema
    - To achieve similar relation schema use projection and renaming
  - Renaming allows the renaming of attributes and relations

## 2.1 Relational Algebra

### Renaming

- Renaming in SQL is done using the keyword **AS**



### Example

$\rho_{\text{surname}} \leftarrow \text{LastName}(\text{Person})$

```
1  SELECT SSN, LastName AS Surname
2  FROM PERSON;
```



*Person*

<u>SSN</u>	Last Name	First Name	Mobile	...
123456789	Miller	Jane	0044 7701 123456	...
234567891	Miller	Steven	0044 7701 123457	...
345678912	Smith	Maria		...
...	...	...	...	...



$\pi_{SSN, LastName}(Person)$

<u>SSN</u>	Last Name
123456789	Miller
234567891	Miller
345678912	Smith
...	...

### Union

- Union, intersection, and set difference can only be applied on two relations that are union compatible
  - Union compatible means that the two relations have the same number of attributes and
  - each corresponding pair of attributes has the same domain



## 2.2 Union, Intersection and Difference

### Union

- Union  $\cup$ 
  - Example: Retrieve the Social Security numbers of all employees who
    - either work in department 5
    - or directly supervise an employee who works in department 5



### Example

```
DEP5_EMPS  $\leftarrow \sigma_{\text{DNR}=5}(\text{Employees})$  RESULT1  $\leftarrow \pi_{\text{SSN}}(\text{DEP5\_EMPS})$   
RESULT2  $\leftarrow \pi_{\text{Superssn}}(\text{DEP5\_EMPS})$  RESULT  $\leftarrow \text{RESULT1} \cup \text{RESULT2}$ 
```

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

### Union

- Union  $\cup$ 
  - Example: Retrieve the Social Security numbers of all employees who
    - either work in department 5
    - or directly supervise an employee who works in department 5



### Example

```
1  (SELECT ssn FROM EMPLOYEE
2  WHERE DEPARTMENT_NUM = 5)
3  UNION
4  (SELECT Superssn FROM EMPLOYEE
5  WHERE DEPARTMENT_NUM = 5)
```



## 2.2 Union, Intersection and Difference

### Union

**Student**

FirstName	LastName
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wand
Ernest	Gilbert

**Instructor**

FirstName	LastName
John	Smith
Ricardo	Brown
Susan	Yao
Francis	Johnson
Ramesh	Shah

**Student  $\cup$  Instructor**

FirstName	LastName
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wand
Ernest	Gilbert
John	Smith
Ricardo	Brown
Francis	Johnson

## 2.2 Union, Intersection and Difference

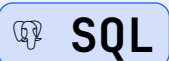
### Intersection

- Intersection  $\cap$ 
  - The result is a relation that includes all tuples in both  $R$  and  $S$
  - Commutative  $R \cap S = S \cap R$
  - Duplicate tuples are eliminated



### Example

```
1 SELECT supplier_id FROM SUPPLIERS
2 INTERSECT
3 SELECT SUPPLIER_ID FROM ORDERS;
```



## 2.2 Union, Intersection and Difference

### Union, Intersection and Difference

## 2.2 Union, Intersection and Difference

### Student

FirstName	LastName
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wand
Ernest	Gilbert

### Instructor

FirstName	LastName
John	Smith
Ricardo	Brown
Susan	Yao
Francis	Johnson
Ramesh	Shah

### Student $\cap$ Instructor

FirstName	LastName
Susan	Yao
Ramesh	Shah



## 2.2 Union, Intersection and Difference

### Union, Intersection and Difference

- Set Difference (-)
  - The result  $(R - S)$  is a relation that includes all tuples that are in  $R$  but not in  $S$
  - Not commutative:  $R - S \neq S - R$
  - Duplicate tuples are eliminated



### Example

```
1  STUDENT except INSTRUCTOR;  
2  STUDENT MINUS INSTRUCTOR;
```



# 2.2 Union, Intersection and Difference

## Union, Intersection and Difference

## 2.2 Union, Intersection and Difference

**Student**

FirstName	LastName
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wand
Ernest	Gilbert

**Instructor**

FirstName	LastName
John	Smith
Ricardo	Brown
Susan	Yao
Francis	Johnson
Ramesh	Shah

**Student - Instructor**

FirstName	LastName
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wand
Ernest	Gilbert

**Instructor - Student**

FirstName	LastName
John	Smith
Ricardo	Brown
Francis	Johnson

# 2.2 Union, Intersection and Difference

## Union, Intersection and Difference

## 2.2 Union, Intersection and Difference

### Student

FirstName	LastName
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wand
Ernest	Gilbert

### Instructor

FirstName	LastName
John	Smith
Ricardo	Brown
Susan	Yao
Francis	Johnson
Ramesh	Shah

### Student $\cup$ Instructor

FirstName	LastName
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wand
Ernest	Gilbert
John	Smith
Ricardo	Brown
Francis	Johnson

### Student - Instructor

FirstName	LastName
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wand
Ernest	Gilbert

### Instructor - Student

FirstName	LastName
John	Smith
Ricardo	Brown
Francis	Johnson

### Student $\cap$ Instructor

FirstName	LastName
Susan	Yao
Ramesh	Shah

## 2.2 Union, Intersection and Difference

### Union, Intersection and Difference

- In SQL, there are three operations — UNION, INTERSECT, and EXCEPT — that correspond to the set operations described here
- In addition, there are multiset operations — UNION ALL, INTERSECT ALL, and EXCEPT ALL — that do not eliminate duplicates

## 2.2 Union, Intersection and Difference

### Cartesian Product

- Cartesian Product
  - This is also a binary set operation
  - Relations do not have to be union compatible
  - The result ( $A \times B$ ) is the combination of each tuple of the first relation  $A$  with each tuple of the second one  $B$
  - In general, the result of  $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$  is a relation  $Q$  with degree  $n + m$  attributes  $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$
  - If  $R$  has  $n_R$ -tuples, and  $S$  has  $n_S$ -tuples, then  $Q$  will have  $n_R \cdot n_S$ -tuples
  - In SQL, the Cartesian Product can be realized by using the `CROSS JOIN` option in joined tables.

## 2.2 Union, Intersection and Difference

### Cartesian Product

Person	SSN	Last Name	First Name	Mobile
	123456789	Miller	Jane	0044 7701 123456
	234567891	Miller	Steven	0044 7701 123457
	...	...	...	...

House	Address	Phone
	221 Baker Street, 1NW London	0044 20 7946 0000
	112 Baker Street, 1NW London	0044 20 7946 1000
	...	...



*Person × House*

SSN	Last Name	First Name	Mobile	Address	Phone
123456789	Miller	Jane	0044 7701 123456	221 Baker Street, 1NW London	0044 20 7946 0000
123456789	Miller	Jane	0044 7701 123456	112 Baker Street, 1NW London	0044 20 7946 1000
234567891	Miller	Steven	0044 7701 123457	221 Baker Street, 1NW London	0044 20 7946 0000
234567891	Miller	Steven	0044 7701 123457	112 Baker Street, 1NW London	0044 20 7946 1000
...	...	...	...	...	...



### Cartesian Product



#### Example

Retrieve a list of names of each female employee and her dependents

```
FEMALE_EMPS  $\leftarrow \sigma_{\text{sex}=\text{F}}(\text{EMPLOYEE})$  EMPNAMES  $\leftarrow$   
 $\pi_{\text{FName, LName, SSN}}(\text{FEMALE_EMPS})$  EMP_DEPENDENTS  $\leftarrow \text{EMPNAMES} \times$   
DEPENDENT ACTUAL_DEPENDENTS  $\leftarrow \sigma_{\text{SSN}=\text{ESSN}}(\text{EMP_DEPENDENTS})$   
RESULT  $\leftarrow \pi_{\text{Fname, Lname, Dependent\_name}}(\text{ACTUAL\_DEPENDENTS})$ 
```

**FEMALE\_EMPS**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

**EMPNAMES**

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

## 2.2 Union, Intersection and Difference

### Cartesian Product

**EMP\_DEPENDENTS**

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

### Cartesian Product



#### Example

```
FEMALE_EMPS  $\leftarrow \sigma_{\text{sex}=\text{F}}(\text{EMPLOYEE})$  EMPNAMES  $\leftarrow$   
 $\pi_{\text{FName, LName, SSN}}(\text{FEMALE\_EMPS})$  EMP_DEPENDENTS  $\leftarrow \text{EMP\_NAMES} \times$   
DEPENDENT ACTUAL_DEPENDENTS  $\leftarrow \sigma_{\text{SSN}=\text{ESSN}}(\text{EMP\_DEPENDENTS})$   
RESULT  $\leftarrow \pi_{\text{Fname, Lname, Dependent\_name}}(\text{ACTUAL\_DEPENDENTS})$ 
```

**ACTUAL\_DEPENDENTS**

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...

**RESULT**

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

⋮  
|

## 2.2 Union, Intersection and Difference

### Join

- Join ( $\bowtie$ ):
  - Combine related tuples from two relations into single “longer” tuples
  - **Very important concept!**
  - Specify, which tables should be combined
  - The same attribute name merges
  - Without same attributes: the join is the cartesian product
  - There are different types of joins, which are presented later in more detail
  - Comparison to Cartesian Product: The result has one tuple for each combination of tuples of the two relations whenever the combination satisfies the join condition

## Join



### Example

Retrieve all attributes of the managers of each department.

$DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{Mgr\_SSN = SSN(EMPLOYEE)}$

DEPT\_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...





### Join



#### Example

With Cartesian Product:  $\text{EMP\_DEPENDENTS} \leftarrow \text{EMP\_NAMES} \times \text{DEPENDENT}$   
 $\text{ACTUAL\_DEPENDENT} \leftarrow \sigma_{\text{SSN}=\text{ESSN}}(\text{EMP\_DEPENDENTS})$

- With Equijoin:

$\text{ACTUAL\_DEPENDENTS} \leftarrow \text{EMP\_NAMES} \bowtie_{\text{SSN}=\text{ESSN}}(\text{DEPENDENT})$

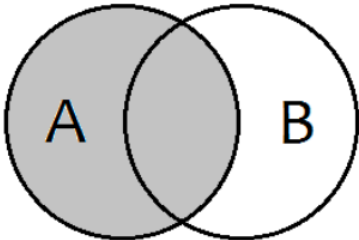
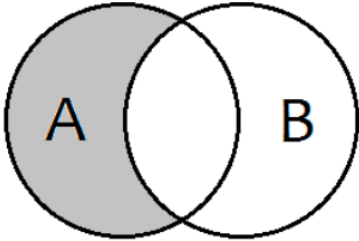
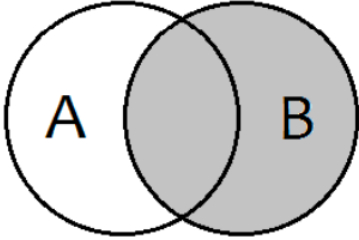
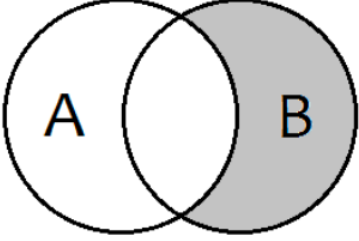
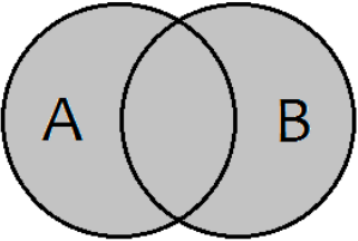
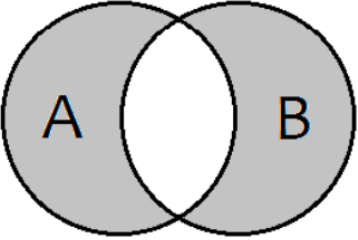
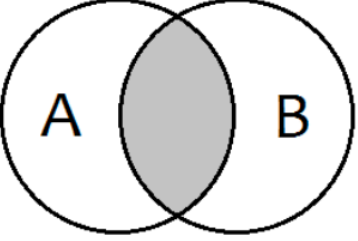
## 2.2 Union, Intersection and Difference

### Join

- Variations of JOIN:
  - Equijoin
    - Used comparison operator is = only
    - Two attributes requires the values to be identical in every tuple in the result
  - Natural Join
    - Two join attributes (or each pair of join attributes) have the same name in both relations.
    - If this is not the case, a renaming operation is applied first

## 2.2 Union, Intersection and Difference

### Join

	<pre>SELECT * FROM TableA a LEFT JOIN TableB b ON a.Key = b.Key</pre>	<h2>SQL JOINS</h2>
	<pre>SELECT * FROM TableA a LEFT JOIN TableB b ON a.Key = b.Key WHERE b.Key IS NULL</pre>	
	<pre>SELECT * FROM TableA a RIGHT JOIN TableB b ON a.Key = b.Key</pre>	
	<pre>SELECT * FROM TableA a RIGHT JOIN TableB b ON a.Key = b.Key WHERE a.Key IS NULL</pre>	
	<pre>SELECT * FROM TableA a FULL OUTER JOIN TableB b ON a.Key = b.Key</pre>	
	<pre>SELECT * FROM TableA a FULL OUTER JOIN TableB b ON a.Key = b.Key WHERE a.Key IS NULL OR b.Key IS NULL</pre>	
	<pre>SELECT * FROM TableA a INNER JOIN TableB b ON a.Key = b.Key</pre>	

Source: <https://huklee.github.io/2017/01/28/021.SQL-all-kinds-of-join-queries/>

## 2.2 Union, Intersection and Difference

### Size of Result

- How many tuples are in my result set (cardinality)?
  - Interesting question for end user ("I'll just print it!")
  - Interesting question for programmer ("Program is running forever?!")
  - Interesting question for DBMS creator ("I'll start with operation 1 and do operation 2 afterwards")
- The answer to this question depends on involved operations...

## 2.2 Union, Intersection and Difference

### Size of Result

- The answer to this question depends on involved operations:
- Projection
  - Upper bounds: number of tuples in the projected relation
  - Lower bounds: 1 (for not empty original relation)
  - Rule: If the projected attribute contains a key candidate, then the cardinality of the result is equal to the amount of tuples
  - This rule also applies if the attributes of the current database state are coincidentally a key candidate

## 2.2 Union, Intersection and Difference

### Size of Result

- The answer to this question depends on involved operations:
- Selection
  - The cardinality of the selection depends on the selection conditions
  - Upper bounds: amount of tuples
  - Lower bounds: 0
  - Selection is used to restrict the number of tuples, thus, the upper bounds is rarely present in practice

## 2.2 Union, Intersection and Difference

### Size of Result

- The answer to this question depends on involved operations:
- Cartesian Product
  - Cardinality is the product of the cardinalities of participating relations
  - Thus, the cartesian product is always an "expensive" operation
- Join
  - Upper bounds: Product of cardinalities of participating relations
  - Lower bounds: 0
  - Thus, the join operation may be an "expensive" operation



## 2.2 Union, Intersection and Difference

### Operations

- Minimal set of operations
  - It's sufficient if a language provides the operations  $\rho, \sigma, \pi, \cup, -, \times$ 
    - The language is then “relational complete”, meaning “everything” is requestable
    - The operations are also independent, therefore none of it are dispensable
  - Other operations are representable by these operations:
    - Example:  $R \cap S \Leftrightarrow (R \cup S) - ((R - S) \cup (S - R))$
  - Important for the implementation of a DBMS and for the optimization of queries

## 2.2 Union, Intersection and Difference

### Operations

- Selection ( $\sigma$  (sigma))
- Projection ( $\pi$  (pi))
- Renaming ( $\rho$  (rho))
- Union ( $\cup$ )
- Set Difference (or Except, Minus,  $-$ )
- Cartesian Product ( $\times$ )
- All other operations can be built from these!

## 2.2 Union, Intersection and Difference Operations

Operation	Purpose	Notation
Selection	Selects all tuples that satisfy the selection condition from a relation $R$	$\sigma_{\langle selection\ condition \rangle}(R)$
Projection	Produces a new relation with only some of the attributes of $R$ , and removes duplicate tuples	$\pi_{\langle attribute\ list \rangle}(R)$
Renaming	Column in the result relation gets new name	$\rho_{new\ name \leftarrow attribute\ name}(R)$
Join	Produces all combinations of tuples from $R_1$ and $R_2$ that satisfy the join condition	$R_1 \bowtie_{\langle join\ condition \rangle} R_2$
Equijoin	Produces all the combinations of tuples from $R_1$ and $R_2$ that satisfy a join condition with only equality comparisons	$R_1 *_{\langle join\ condition \rangle} R_2$

## 2.2 Union, Intersection and Difference Operations

Operation	Purpose	Notation
Union	Produces a relation that includes all the tuples in $R_1$ or $R_2$ or both $R_1$ and $R_2$ ; $R_1$ and $R_2$ must be union compatible	$R_1 \cup R_2$
Intersection	Produces a relation that includes all the tuples in both $R_1$ and $R_2$ ; $R_1$ and $R_2$ must be union compatible	$R_1 \cap R_2$
Set Difference	Produces a relation that includes all the tuples in $R_1$ that are not in $R_2$ ; $R_1$ and $R_2$ must be union compatible	$R_1 - R_2$
Cartesian Product	Produces a relation that has the attributes of $R_1$ and $R_2$ and includes as tuples all possible combinations of tuples from $R_1$ and $R_2$	$R_1 \times R_2$

## 2.2 Union, Intersection and Difference

### Relational Calculus

- The relational algebra constructs the query result by applying operations and an order (project on  $X$ , select the  $Y$  and combine that with  $R_2, \dots$ )
- In contrast, the relational calculus are using a descriptive approach
- Calculus are logic-based approaches like the predicate logic
- Therefore, sets are characterized that correspond with the query result
- Calculus has variables, constants, comparison operations, logical operations and quantifier

## 2.2 Union, Intersection and Difference

### Relational Calculus

- Two types of relational calculus
  - Tuple relational calculus: variables declare tuples (are bounded to them)
  - Domain relational calculus: variables declare domain elements (thus, values range of attributes)
- Expressions in the calculus are called formula
- A result tuple is more or less an assignment of constants to variables, so that the formula is evaluated as TRUE

## 2.2 Union, Intersection and Difference

### Relational Calculus



#### Example

- In mathematics:
  - $\{x^2 \mid x \in N \wedge x^3 < 1000 \wedge x^3 > 0\}$
  - This defines the set of all square numbers that cube number is between 0 and 1000.
- Relational calculus:
  - $A = \{x \mid \text{Person}(x, y) \wedge y = \text{'Jones'}\} = \{2\}$
  - By usage of complex expressions (formula), the calculus has the same expressiveness as the relational algebra

## 2.2 Union, Intersection and Difference

### Relational Calculus

- Query languages for relational database schemas are mathematical substantiated
- The mathematical basis are the relational algebra and the relational calculus
- The relational algebra defines few operations, with that every request is expressible: Projection, Selection, Join, Renaming, Union, Set difference
- The relational calculus characterizes sets, which corresponds with the query result
- The relational calculus is descriptive, because it doesn't have to define an order of operations that construct the result
- Relational algebra and relational calculus have the same expressiveness



### 3. Subqueries & Views

---

## 3.1 Subqueries

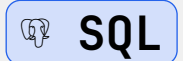
### Nested Queries

- SELECT returns relation: a (multi-)set
- Result of SELECT can be included in query
  - WHERE clause
  - also, for UPDATE, DELETE
  - HAVING clause
  - FROM clause
  - SELECT clause (in column list)
- So, we have two (or more) SELECTs:
  - Outer SELECT
  - Nested (or inner) SELECT: subquery

## 3.1 Subqueries

### Nested Queries

```
1 INSERT INTO UNDERPAID (lname, fname) SELECT lname, fname  
FROM Employee WHERE salary < 1000;
```



- WHERE clause belongs to SELECT

## 3.1 Subqueries

**WHERE x in**

- In general, the nested query will return a table (relation), which is a set or multiset of tuples
- Check if value is a member of set

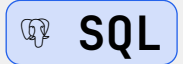
**WHERE a IN (1,4,9)**

- Set can be result of query

**WHERE a IN (SELECT x FROM y)**

WHERE x in

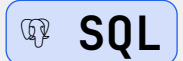
```
1  SELECT DISTINCT Essn FROM WORKS_ON WHERE (Pno) IN (SELECT  
   Pno FROM WORKS_ON WHERE Essn='123456789');
```



## 3.1 Subqueries

WHERE x in

```
1  SELECT title FROM books WHERE isbn IN (SELECT isbn  
    FROM recommended_books)
```

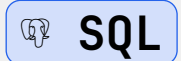


- The subqueries are independent of each other, because they do not access the same tables

## 3.1 Subqueries

WHERE x in

```
1  SELECT lastname FROM person WHERE 1.0 IN (SELECT grade
    FROM exam WHERE person.no = exam.no)
```

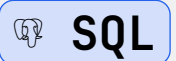


- In this example both queries depend on each other, because the second query references a part of the first relation ("person").

## 3.1 Subqueries

WHERE x in

```
1  SELECT E.Fname, E.Lname FROM EMPLOYEE AS E WHERE E.Ssn IN  
   ( SELECT Essn FROM DEPENDENT AS D WHERE E.Sex = D.Sex );
```





## 3.1 Subqueries

WHERE x in

- The operator IN can also be used for explicit enumerations

```
1  WHERE value IN ( value1 , value2 , value3 , ... );
```



```
2  WHERE colour IN ( 'red' , 'blue' );
```

```
3
```

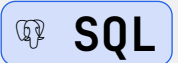
```
4  SELECT DISTINCT Essn FROM WORKS_ON WHERE Pno IN (1, 2, 3);
```

## 3.1 Subqueries

WHERE x in

- Example from last chapter (about set operations):

```
1  (SELECT DISTINCT Pnumber
2  FROM PROJECT, DEPARTMENT, EMPLOYEE
3  WHERE Dnum=Dno AND Mgr_ssn=Ssn AND Lname="Wong")
4  UNION
5  (SELECT DISTINCT Pnumber
6  FROM PROJECT, WORKS_ON, EMPLOYEE
7  WHERE Pnumber= Pno AND Essn= Ssn AND Lname="Wong");
```

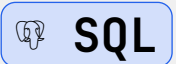


## 3.1 Subqueries

WHERE x in

- Example from last chapter (alternative statement using subqueries):

```
1 SELECT DISTINCT Pnumber
2 FROM PROJECT
3 WHERE Pnumber IN
4   (SELECT Pnumber FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE
5    Dnum=Dnumber AND Mgr_ssn=Ssn AND Lname="Wong")
6 OR Pnumber IN
7   (SELECT Pno FROM WORKS_ON, EMPLOYEE WHERE Essn=Ssn AND
8    Lname="Wong" );
```



## 3.1 Subqueries

WHERE x in

- Special case: Nested query returns only one value
  - In such cases, it is permissible to use = instead of IN for the comparison operator
- Example: `SELECT * FROM y WHERE x = ( SELECT MAX(x) FROM y );`

## 3.1 Subqueries

WHERE x in

- In general, a query written with nested SELECT-FROM-WHERE blocks and using the = or IN comparison operators can always be expressed as a single block query.
- Example from before:

```
1 SELECT E.Fname, E.Lname
2 FROM EMPLOYEE AS E
3 WHERE E.Ssn IN
4 (SELECT Essn FROM DEPENDENT AS D WHERE E.Sex=D.Sex);
```

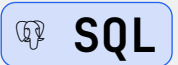


## 3.1 Subqueries

WHERE x in

- Example from last slide (Alternative statement without a subquery)

```
1 SELECT E.Fname, E.Lname FROM EMPLOYEE AS E, DEPENDENT AS D  
   WHERE E.Ssn=D.Essn AND E.Sex=D.Sex;
```



## 3.1 Subqueries

### Other Comparison Operators

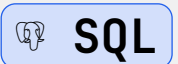
- `=ANY` (`=SOME`)
  - operator returns TRUE if the value  $v$  is equal to some value in the set  $V$
  - is equivalent to IN
  - Other operations can be combined with ANY, e.g., `>`, `>=`, `<`, `<=`, and `<>`
- Example: Persons who have borrowed a book: `SELECT name FROM Person WHERE PNr = ANY (SELECT PNr FROM book );`

## 3.1 Subqueries

### Other Comparison Operators

- ALL
  - Comparison operations can be combined with ALL, e.g., >, >=, <, <=, and <>
  - Example: Employees earning more money as the employees of department 5

```
1 SELECT Lname, Fname FROM EMPLOYEE WHERE Salary > ALL  
   (SELECT Salary FROM EMPLOYEE WHERE Dno=5) ;
```



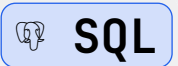


## 3.1 Subqueries

### Other Comparison Operators

- Often several queries give the same result
  - but might have difference in the performance!

1



```
2 SELECT * FROM book WHERE price <= ALL (SELECT price FROM book);
```

```
3 SELECT * FROM book WHERE price = (SELECT MIN(price) FROM book);
```

```
4 SELECT * FROM book WHERE price >= ALL (SELECT price FROM book);
```

```
5 SELECT * FROM book WHERE price = (SELECT MAX(price) FROM book);
```

```
6 SELECT * FROM book WHERE price > ANY (SELECT price FROM book);
```

```
7 SELECT * FROM book WHERE price > (SELECT MIN(price) FROM book);
```

## 3.1 Subqueries

### Other Comparison Operators

- Often several queries give the same result
  - but might have difference in the performance!
  - Strategy depends on DBMS, probably equivalent if no index on price, otherwise, the second version will be (much) faster

## 3.1 Subqueries

### Correlated Nested Queries

- Uncorrelated
  - Outer and nested query are independent
  - Nested query must be computed only once
- Correlated
  - Nested query depends on columns of outer query
  - Result of a correlated nested query is different for each tuple of the relation(s) outer query
  - A nested query is evaluated once for each tuple (or combination of tuples) in the outer query

### ? Question

But what about performance?

## 3.1 Subqueries

### Correlated Nested Queries



#### Example

Who has borrowed books for 9.99\$ ?

```
SELECT name FROM pers WHERE 9.99 IN (SELECT price FROM  
1 book WHERE pers.PNr = book.PNr ) ; SELECT name FROM pers,  
book WHERE pers.PNr = book.PNr AND book.price = 9.99;
```



## 3.1 Subqueries

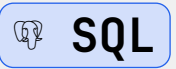
### Correlated Nested Queries



#### Example

From Before: Who is lead by a female superior?

```
1 SELECT E.Fname, E.Lname FROM EMPLOYEE AS E WHERE E.Ssn IN  
   ( SELECT Essn FROM DEPENDENT AS D WHERE D.Sex='F' );
```



## 3.1 Subqueries

### Correlated Nested Queries

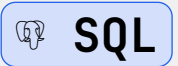
- In general, we can have several levels of nested queries
  - possible ambiguity among attribute names if attributes of the same name exist:
    - one in a relation in the FROM clause of the outer query, and
    - another in a relation in the FROM clause of the nested query
- The rule is that a reference to an unqualified attribute refers to the relation declared in the **innermost nested query**.

## 3.1 Subqueries

### Ambiguity of Attributes

- Example: Retrieve the name of each employee who has a female dependent:

```
1 SELECT E.Fname, E.Lname FROM EMPLOYEE AS E WHERE E.Ssn IN  
   ( SELECT Essn FROM DEPENDENT AS D WHERE D.Sex='F' );
```





## 3.1 Subqueries

### Ambiguity of Attributes

#### ! Memorize

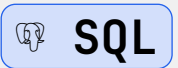
It is generally advisable to create tuple variables (aliases) for all the tables referenced in an SQL query to avoid potential errors and ambiguities!!!

## 3.1 Subqueries

### EXISTS

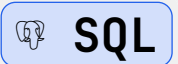
- The operator EXISTS (NOT EXISTS) provides the possibility to check if the result of another query is empty (FALSE) or not (TRUE)

```
1  SELECT isbn FROM book WHERE EXISTS (SELECT * FROM borrowed  
WHERE book.libraryno = borrowed.libraryno)
```



- This example provides as result a set of all borrowed books
- Typically, the usage is so that the DBMS may decide, which column should be examined:

```
1  ... EXISTS (SELECT * ...
```

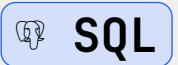


## 3.1 Subqueries

### EXISTS

Alternative SQL-statement from before:

```
SELECT E.Fname, E.Lname FROM EMPLOYEE AS E WHERE EXISTS  
1 (SELECT * FROM DEPENDENT AS D WHERE E.Ssn = D.Essn AND  
   E.Sex = D.Sex);
```



## 3.1 Subqueries

### EXISTS

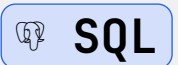
- EXISTS and NOT EXISTS are typically used in conjunction with a correlated nested query



### Example

Retrieve the names of employees who have no dependents

```
1  SELECT Fname, Lname FROM EMPLOYEE WHERE NOT EXISTS  
   ( SELECT * FROM DEPENDENT WHERE Ssn=Essn );
```



# 3.1 Subqueries

## EXISTS



### Example

List the names of managers who have at least one dependent

```
SELECT Fname, Lname FROM EMPLOYEE WHERE EXISTS ( SELECT *  
1 FROM DEPENDENT WHERE Ssn=Essn ) AND EXISTS ( SELECT * FROM  
DEPARTMENT WHERE Ssn=Mgr_ssn );
```



## 3.1 Subqueries

### In FROM

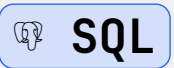
- SELECT returns a new relation so that, we can select values from it
- Necessary: give a name to the relation



### Example

Set an alias for a SELECT statement in the FROM statement

```
1 SELECT tab_a.x , newtab_b.y FROM tab_a , (SELECT v1, v2  
FROM tab_b) AS newtab_b;
```



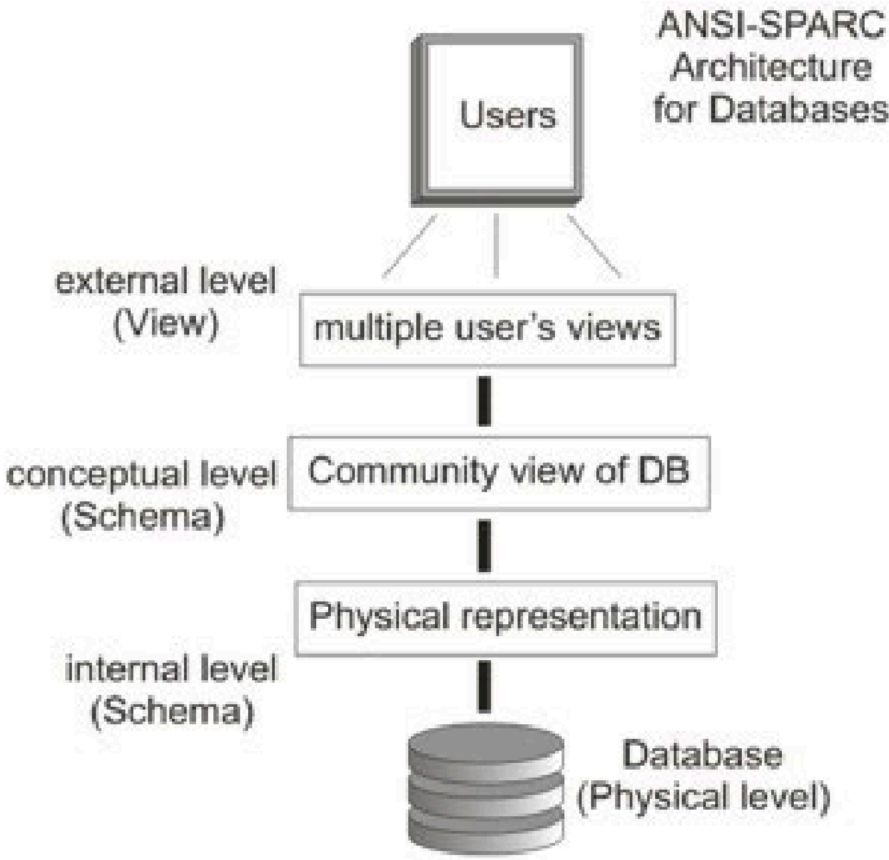
# 3.1 Subqueries

## ANSI-SPARC Model

External Schema

Conceptual Schema

Internal Schema



## 3.2 Views

### Basics

- User or application specific views on data
- Only relevant portions of the data
- A view in SQL terminology is a single table that is derived from other tables
  - Other tables can be base tables or previously defined views
- A view is considered to be a virtual table
  - In contrast to base tables
  - Limits the possible update operations
  - No limitations on querying a view



## 3.2 Views

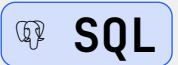
### Use Cases

- Hide some information
  - Example: Salary not viewable for colleagues
  - Can see only employees of same department?
- Convert data for different users
  - Price in \$, EUR, etc.
- Backward Compatibility
  - Example: Add some columns, but old applications do `SELECT *`
- Simplification: Hide away complex queries
  - Example: Data Dictionary Views (all tables)

## 3.2 Views

### Syntax

```
1 CREATE [OR REPLACE] VIEW <vname> AS <query> ;
```



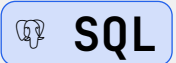
- <query> is an arbitrary SELECT statement

## 3.2 Views

### Syntax

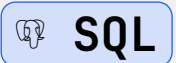
- Example for a sample view:

```
1 CREATE VIEW vPerson AS
2 SELECT Name , Id , BirthDate FROM person ;
```



- Can rename columns in view:

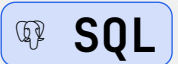
```
1 CREATE VIEW vPerson ( lname , pnr , bd ) AS
2 SELECT Name , Id , BirthDate FROM person
```



## 3.2 Views

### Syntax

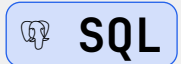
```
1 CREATE VIEW v_WORKS_ON1 AS
2 SELECT Fname, Lname, Pname, Hours
3 FROM EMPLOYEE, PROJECT, WORKS_ON
4 WHERE Ssn=Essn AND Pno=Pnumber;
```



## 3.2 Views

### Syntax

```
1 CREATE VIEW v_DEPT_INFO
2 (Dept_name, No_of_ems, Total_sal) AS
3 SELECT Dname, COUNT(*), SUM(Salary)
4 FROM
5 DEPARTMENT, EMPLOYEE
6 WHERE
7 Dnumber=Dno
8 GROUP BY Dname;
```



## 3.2 Views

### Queries

- A view is supposed to be always up-to-date
  - If we modify the tuples in the base tables on which the view is defined, the view must automatically reflect these changes
  - View realized at the time when we specify a query on the view

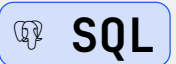
```
1 SELECT Fname, Lname
2 FROM
3 v_WORKS_ON1
4 WHERE
5 Pname="ProductX";
```



### DROP

- Views can be dropped

```
1 DROP VIEW v_WORKS_ON1;
```



## 3.2 Views

### Implementation

- Two strategies:
  1. Query Modification
    - Transforming the view query into a query on the underlying base tables

```
1 SELECT Fname, Lname
2 FROM EMPLOYEE, PROJECT, WORKS_ON
3 WHERE Ssn=Essn
4 AND Pno=Pnumber
5 AND Pname= "ProductX";
```





## 3.2 Views

### Implementation

- Two strategies:

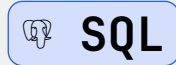
#### 2. View Materialization

- Physically creating a temporary view
- Incremental update of materialized view
- If the view is not queried for a certain period of time, the system may then automatically remove the physical table and recompute it from scratch on new queries

## 3.2 Views

### Materialized Views

```
1 CREATE MATERIALIZED VIEW  
   <name> AS SELECT ...
```



- Traditional views
  - SELECT is performed when needed
  - Performance penalty
- Materialized view
    - Store select statement and selected data
    - Problems
      - Store data twice
      - When to update selected data?
    - Rules for updating: event vs. time triggered
    - Selected data can be updated
      - manually
      - on a regular basis (every night)

- event triggered (update to base table)

## 4. License Notice

---

## 4.1 Attribution

The basis of this course stems from: Professor Dr. Ulrike Herster The following license applies to all unmodified and modified material used throughout this course.

### ! License

The publication or sharing of slides, image and sound recordings of this course are not permitted.

© Professor Dr. Ulrike Herster

This video and the presented script are protected by copyright. The use is only permitted in relation with the course of study. It is not permitted to forward or republish it in other places (especially in the internet).