Projeto e Administração de Bancos de Dados

Prof. Daniel S. Kaster

Departamento de Computação
Universidade Estadual de Londrina
dskaster@uel.br

Relational query languages

- Relational algebra
  - Procedural language
  - Indicates what to retrieve and the sequence of operators
- Relational calculus
  - Declarative language (non-procedural)
  - Only indicates what to retrieve
  - Relational calculus has two flavors, based on the type of the variables employed
    - Tuple relational calculus and domain relational calculus
- SQL
  - The ”de facto” standard in DBMS
  - SQL foundation heavily relies on relational algebra and relational calculus; however, also includes a lot of non-relational stuff

Relational algebra

Relational Algebra Overview

- Relational algebra is the basic set of operations for the relational model
- These operations enable a user to specify **basic retrieval requests** (or **queries**)
- The result of an operation is a *new relation*, which may have been formed from one or more *input* relations
  - This property makes the algebra "closed" (all objects in relational algebra are relations)

Relational Algebra Overview (2)

- The **algebra operations** thus produce new relations
  - These can be further manipulated using operations of the same algebra
- A sequence of relational algebra operations forms a **relational algebra expression**
  - The result of a relational algebra expression is also a relation that represents the result of a database query (or retrieval request)

Relational Algebra Operations

- Relational Algebra consists of several groups of operations
  - Unary Relational Operations
    - SELECT (symbol: σ (sigma))
    - PROJECT (symbol: π (pi))
    - RENAME (symbol: ρ (rho))
  - Relational Algebra Operations From Set Theory
    - UNION ($\cup$), INTERSECTION ($\cap$), DIFFERENCE or MINUS (–)
    - CARTESIAN PRODUCT ($\times$)
  - Binary Relational Operations
    - JOIN ($\bowtie$ – several variations of JOIN exist)
    - DIVISION ($\div$)
  - Additional Relational Operations
    - OUTER JOINS (⟗)
    - AGGREGATE FUNCTIONS (these compute summary of information; for example: SUM, COUNT, AVG, MIN, MAX)

Unary Relational Operations: SELECT

- The SELECT operation (denoted by σ (sigma)) is used to select a *subset* of the tuples from a relation based on a **selection condition**
  - The selection condition acts as a **filter**
  - Keeps only those tuples that satisfy the qualifying condition
  - Tuples satisfying the condition are *selected* whereas the other tuples are discarded (*filtered out*)
- Examples:
  - Select the EMPLOYEE tuples whose department number is 4:

$$\sigma_{\text{DNO} = 4}\ (\text{EMPLOYEE})$$

  - Select the employee tuples whose salary is greater than $30,000:

$$\sigma_{\text{SALARY} > 30{,}000}\ (\text{EMPLOYEE})$$

Unary Relational Operations: SELECT (2)

- SELECT Operation Properties
  - The SELECT operation $\sigma_{\text{<selection condition>}}(R)$ produces a relation S that has the same schema (same attributes) as R
  - SELECT σ is commutative:
    - $\sigma_{\text{<condition1>}}(\sigma_{\text{< condition2>}}(R)) = \sigma_{\text{<condition2>}}(\sigma_{\text{< condition1>}}(R))$
  - Because of commutativity property, a cascade (sequence) of SELECT operations may be applied in any order:
    - $\sigma_{\text{<cond1>}}(\sigma_{\text{<cond2>}}(\sigma_{\text{<cond3>}}(R)) = \sigma_{\text{<cond2>}}(\sigma_{\text{<cond3>}}(\sigma_{\text{<cond1>}}(R)))$
  - A cascade of SELECT operations may be replaced by a single selection with a conjunction of all the conditions:
    - $\sigma_{\text{<cond1>}}(\sigma_{\text{< cond2>}}(\sigma_{\text{<cond3>}}(R)) = \sigma_{\text{<cond1> AND < cond2> AND < cond3>}}(R)))$
  - The number of tuples in the result of a SELECT is less than (or equal to) the number of tuples in the input relation R

Unary Relational Operations: PROJECT

- PROJECT Operation is denoted by π (pi)
- This operation keeps certain *columns* (attributes) from a relation and discards the other columns.
  - PROJECT creates a vertical partitioning
    - The list of specified columns (attributes) is kept in each tuple
    - The other attributes in each tuple are discarded
- Example: To list each employee's first and last name and salary, the following is used:

$$\pi_{\text{LNAME, FNAME,SALARY}}(\text{EMPLOYEE})$$

Unary Relational Operations: PROJECT (2)

- The general form of the *project* operation is:

$$\pi_{\text{<attribute list>}}(R)$$

  - π (pi) is the symbol used to represent the *project* operation
  - <attribute list> is the desired list of attributes from relation R
- The project operation *removes any duplicate tuples*
  - This is because the result of the *project* operation must be a *set of tuples* and mathematical sets *do not allow* duplicate elements
  - Notice: SQL employs bags!

Binary Relational Operations from Set Theory: UNION, INTERSECTION and SET DIFFERENCE

- UNION is a binary operation, denoted by ∪
  - The result of R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R and S
  - Duplicate tuples are eliminated
- INTERSECTION is denoted by ∩
  - The result of the operation R ∩ S, is a relation that includes all tuples that are in both R and S
- SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
  - The result of R – S, is a relation that includes all tuples that are in R but not in S
    - The two operand relations R and S must be "type compatible" (or UNION compatible)

Binary Relational Operations from Set Theory: Type Compatibility

- Type Compatibility of operands is required for the binary set operation UNION ∪, INTERSECTION ∩, and SET DIFFERENCE –
- R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) are type compatible if:
  - they have the same number of attributes, and
  - the domains of corresponding attributes are type compatible (i.e., dom(Ai)=dom(Bi) for i=1, 2, ..., n)
- The resulting relation for R1∪R2, R1∩R2, or R1–R2 has the same attribute names as the *first* operand relation R1 (by convention)

Binary Relational Operations from Set Theory: CARTESIAN PRODUCT

- CARTESIAN (or CROSS) PRODUCT (×)
  - This operation is used to combine tuples from two relations in a combinatorial fashion
  - Denoted by R(A1, A2, ..., An) × S(B1, B2, ..., Bm)
  - Result is a relation Q with degree n + m attributes:
    - Q(A1, A2, ..., An, B1, B2, ..., Bm), in that order
  - The resulting relation state has one tuple for each combination of tuples, one from R and one from S
  - Hence, if R has n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then R×S will have $n_R * n_S$ tuples
  - The two operands do NOT have to be "type compatible"

Binary Relational Operations from Set Theory: CARTESIAN PRODUCT (2)

- Generally, CROSS PRODUCT is not a meaningful operation
  - However, the sequence of CARTESIAN PRODUCT followed by SELECT is used quite commonly to identify and select related tuples from two relations
  - This sequence of operations is very important for any relational database with more than a single relation, because it allows us combine related tuples from various relations
  - This sequence is so useful that it was embedded into a single operation: JOIN
    - There are many variations of the JOIN operation

Other Binary Relational Operations: JOIN

- The JOIN Operation (denoted by $\bowtie$)
  - The general form of a join operation on two relations R(A1, A2, ..., An) and S(B1, B2, ..., Bm) is:

    $$R \bowtie_{\text{<join condition>}} S$$
  - R and S are relations (that might result from general relational algebra expressions)
  - The join condition is a Boolean (conditional) expression specified on the attributes in the cartesian product between relations R and S
    - The join condition has the same form of a select condition (i.e., it may include conjunctions and/or disjunctions of terms, etc.), where the terms usually have the form $A_i\ \theta\ B_i$
    - If θ is always =, the operation is called an EQUIJOIN
      - Otherwise, it is called as THETA JOIN (or θ-JOIN)

Other Binary Relational Operations: JOIN (2)

- Given a JOIN R(A1, A2, ..., An) $\bowtie_{R.Ai=S.Bj}$ S(B1, B2, ..., Bm)
  - Result is a relation Q with degree n + m attributes
    - Q(A1, A2, ..., An, B1, B2, ..., Bm), in that order
  - The resulting relation state has one tuple for each combination of tuples (r from R and s from S), but only if they satisfy the join condition r[Ai]=s[Bj]
  - Hence, if R has n_R tuples, and S has n_S tuples, then the join result will generally have *less than* $n_R * n_S$ tuples.
  - Only related tuples (based on the join condition) will appear in the result

Additional Relational Operations: OUTER JOIN

- In joins, tuples without a *matching* (or *related*) tuple are eliminated from the join result
  - Tuples with null in the join attributes are also eliminated
- A set of operations, called OUTER joins, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the join, regardless of whether or not they have matching tuples in the other relation
- There are three variations:
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN

Additional Relational Operations: OUTER JOIN (2)

- The LEFT OUTER JOIN operation (⟕)
  - Keeps every tuple in the first or left relation R in $R ⟕_{<\text{join condition}>} S$
  - If no matching tuple is found in S, then the attributes of S in the join result are filled or “padded” with null values
- The RIGHT OUTER JOIN (⟖)
  - Keeps every tuple in the second or right relation S in the result of $R ⟖_{<\text{join condition}>} S$, padding with null values as needed
- The FULL OUTER JOIN (⟗)
  - $R ⟗_{<\text{join condition}>} S$ keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed

Additional Relational Operations: Aggregate Functions and Grouping

- A type of request that cannot be expressed in the basic relational algebra is to specify mathematical **aggregate functions** on collections of values from the database
- Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples
  - These functions are used in simple statistical queries that summarize information from the database tuples
  - Common functions applied to collections of numeric values include
    - COUNT, SUM, AVERAGE, MAXIMUM, and MINIMUM

Additional Relational Operations: Aggregate Functions and Grouping

- Aggregate functions using the operator ℑ (*script F*) as follows:

  $_{\text{<grouping attributes>}}\,\Im\,_{\text{<function list>}}\,(R)$

  - The grouping attributes are attributes from relation R
  - The function list includes aggregation functions involving attributes from R
    - Examples: AVG(weight), COUNT(*), SUM(balance+overdraft_limit), COUNT(DISTINCT(location))
  - The resulting relation has as **attributes** the grouping attributes (if any) plus one attribute for storing the result of each aggregate function and as **tuples** one tuple for each distinct group with the corresponding aggregates
    - Important: Nulls are skipped in the computation of aggregation functions

Other Binary Relational Operations: DIVISION

- The DIVISION operation is applied to two relations
  - $R(Z) \div S(X)$, where X is a subset of Z
  - It is usually employed to represent queries that require the notion of ***for all***
- Let $Y = Z - X$ (and hence $Z = X \cup Y$); that is, let Y be the set of attributes of R that are not attributes of S:
  - The result of DIVISION is a relation T(Y) that includes a tuple t if tuples t_R appear in R with $t_R[Y] = t$, and with
    - $t_R[X] = t_s$ *for every tuple* t_s in S
  - For a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with *every* tuple in S

Examples of Queries

1) Retrieve the name and address of all employees who work for the ‘Research’ department
2) Retrieve the names of employees who work on a project controlled by department 5
3) Retrieve the names of employees who work on all the projects controlled by department 5
4) Retrieve the names of all projects which do not have women working on them
5) Retrieve the names of the employees whose salary is the greatest in the department they work, per sex

A Database State for Company

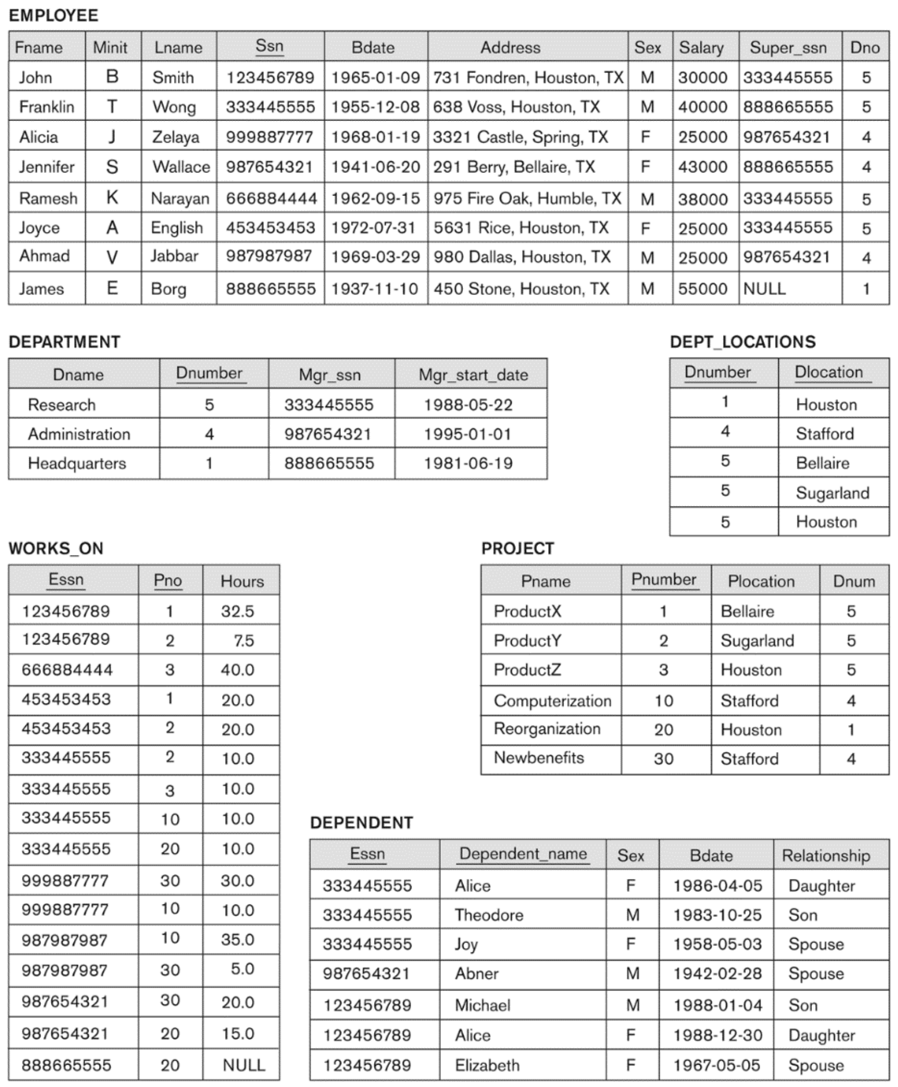


Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	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

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Relational calculus

Relational Calculus

- A **relational calculus** expression creates a new relation, which is specified in terms of variables that range over rows of the stored database relations (in **tuple calculus**) or over columns of the stored relations (in **domain calculus**)
- In a calculus expression, there is *no order of operations* to specify how to retrieve the query result – a calculus expression specifies only what information the result should contain
  - This is the main distinguishing feature between relational algebra and relational calculus
  - Relational calculus is considered to be a **nonprocedural** or **declarative** language while relational algebra can be considered as a **procedural** way of stating a query

Tuple Relational Calculus

- The tuple relational calculus is based on specifying a number of tuple variables (conversely, the domain relational calculus is based on domain variables)
- Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as its value any individual tuple from that relation
- A simple tuple relational calculus query is of the form

  {t | COND(t)}

  - where t is a tuple variable and COND(t) is a conditional expression involving t
  - The result of such a query is the set of all tuples t that satisfy COND(t)

Tuple Relational Calculus (2)

- Two special symbols called quantifiers can appear in formulas; these are the universal quantifier (∀) and the existential quantifier (∃)
- Informally, a tuple variable t is **bound** if it is quantified, meaning that it appears in an (∀t) or (∃t) clause; otherwise, it is **free**
- If F is a formula, then so are (∃t)(F) and (∀t)(F), where t is a tuple variable
  - The formula (∃t)(F) is true if the formula F evaluates to true for some (at least one) tuple assigned to free occurrences of t in F; otherwise (∃t)(F) is false
  - The formula (∀t)(F) is true if the formula F evaluates to true for every tuple (**in the universe**) assigned to free occurrences of t in F; otherwise (∀t)(F) is false

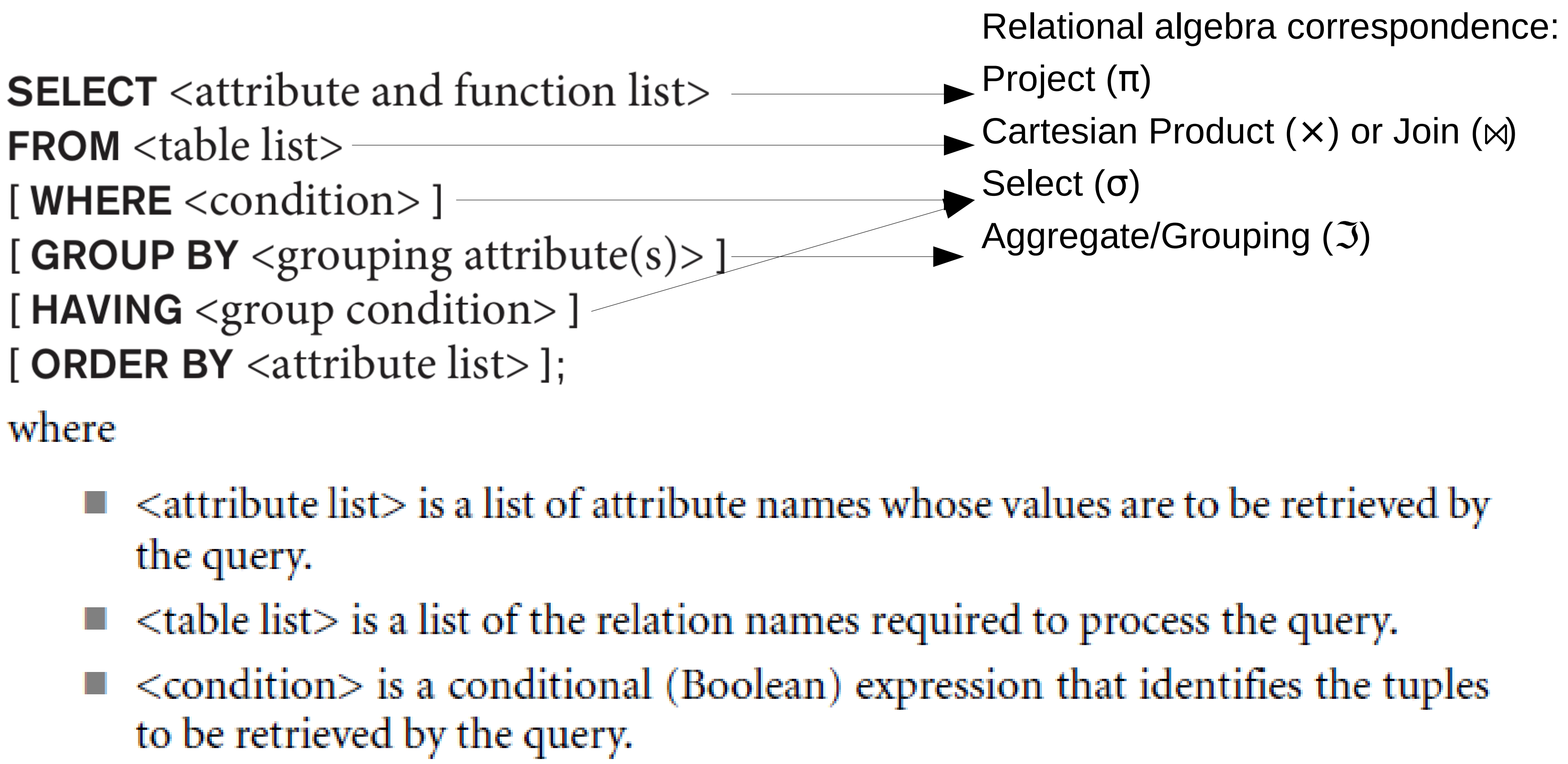
Retrieval queries in SQL

Retrieval Queries in SQL

- SELECT statement
  - One basic statement for retrieving information from a database
- SQL allows a table to have two or more tuples that are identical in all their attribute values
  - Unlike relational model (relational model is strictly set-theory based)
  - Multiset or bag behavior
  - Tuple-id may be used as a key

The SELECT Structure of SQL Queries

- Basic form of the SELECT statement:



Relational algebra correspondence:

Project (π)

Cartesian Product (×) or Join (⋈)

Select (σ)

Aggregate/Grouping (ℑ)

SELECT <attribute and function list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>];

where

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

Aliasing, and Renaming

- Aliases or tuple variables
  - Are needed to refer to the same table more than once in a query
  - Can also be used just to shorten the query
- Renaming of attributes
  - Are needed to distinguish attributes that are homonyms
  - Can also be used for convenience

Example: For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor

```
SELECT  E.Fname, E.Lname,
        S.Fname AS Sup_Fname, S.Sname AS Sup_Sname
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.Super_ssn=S.Ssn;
```

Joined Tables

- The join operation can be represented by:
  - A cartesian product followed by a selection, or
  - A joined table
    - Specify different types of join
      - [INNER] JOIN, NATURAL JOIN, LEFT|RIGHT|FULL [OUTER] JOIN

    - Example: Retrieve all attributes of the manager of the "Research" department

```
SELECT e.*
FROM employee AS e JOIN department AS d
  ON e.dno=d.dnumber
WHERE d.name='Research';
```

Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary
- Built-in aggregate functions
  - COUNT, SUM, MAX, MIN, AVG, STDDEV_POP, ...
  - NULLs are discarded
- Grouping
  - Create subgroups of tuples before summarizing
- To select entire groups, HAVING clause is used
  - HAVING corresponds to a posterior select operation
- Aggregate functions can be used in the SELECT clause or in a HAVING clause

Aggregate Functions in SQL (2)

- Examples
  - Retrieve the number of employees, the number of supervisees and the number supervisors

```
SELECT COUNT(*) AS num_employees,
  COUNT(super_ssn) AS num_supervisees,
  COUNT(DISTINCT(super_ssn)) AS num_supervisors
FROM employee;
```

  - Retrieve the average salary for employees per sex

```
SELECT sex, AVG(salary) AS avg_sal
FROM employee
GROUP BY sex;
```

  - Retrieve the names of the departments with at least 3 employees

```
SELECT d.dname
FROM department d JOIN employee e ON d.dnumber=e.dno
GROUP BY d.dname
HAVING COUNT(*) >= 3;
```

Aggregate Functions in SQL (3)

- Pitfalls when combining the WHERE and the HAVING Clause
  - Example: Retrieve the total number of employees whose salaries exceed $30,000 in each department, but only for departments where at least 3 employees work
  - **Incorrect** query:

```
SELECT dno, COUNT(*)
FROM employee
WHERE salary > 30000
GROUP BY dno
HAVING COUNT(*) >= 3;
```

  - **Correct** query:

```
SELECT dno, COUNT(*)
FROM employee
WHERE salary > 30000
  AND dno IN (SELECT dno FROM employee
                      GROUP BY dno HAVING HAVING COUNT(*) >= 3)
GROUP BY dno;
```

Tables as Sets in SQL

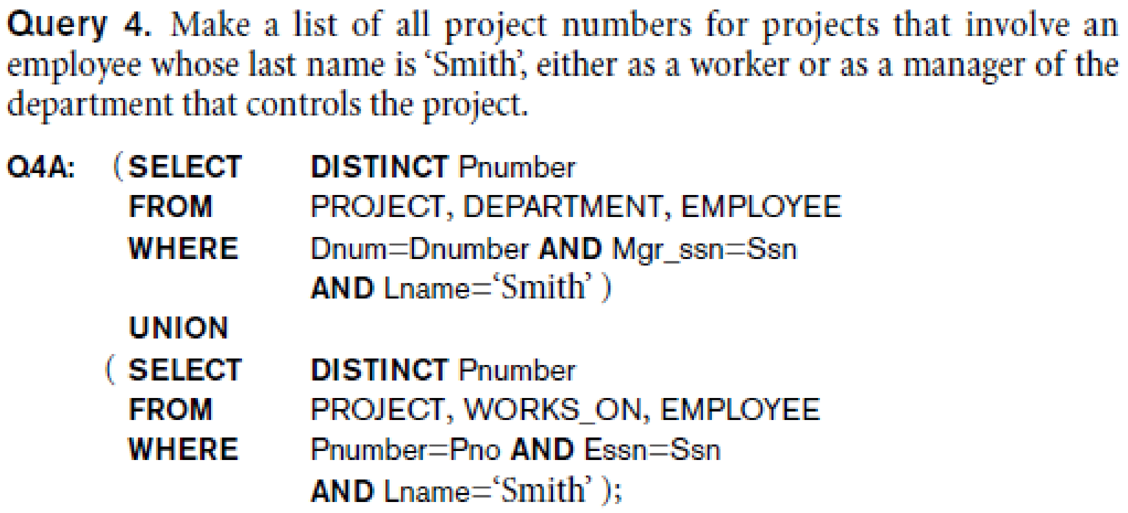
- SQL does not automatically eliminate duplicate tuples in query results
- Use the keyword **DISTINCT** in the SELECT clause
  - Only distinct tuples should remain in the result

Example:

```
SELECT DISTINCT E.SSN
FROM EMPLOYEE E, DEPENDENT D
WHERE E.SSN=D.ESSN;
```

Tables as Sets in SQL (2)

- Set operations
  - UNION, EXCEPT (difference), INTERSECT
    - Corresponding multiset operations: UNION ALL, EXCEPT ALL, INTERSECT ALL
  - Type compatibility is needed for these operations to be valid



Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A:  (SELECT    DISTINCT Pnumber
        FROM     PROJECT, DEPARTMENT, EMPLOYEE
        WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn
                 AND Lname='Smith' )
        UNION
      ( SELECT   DISTINCT Pnumber
        FROM     PROJECT, WORKS_ON, EMPLOYEE
        WHERE    Pnumber=Pno AND Essn=Ssn
                 AND Lname='Smith' );
```

Nested Queries

- Nested queries are complete "select-from-where-group by-having-order by" blocks within WHERE clause of another query (called the outer query)
- Comparison operators
  - IN (v IN S): evaluates to TRUE if v is one of the elements in the set S
    - S is usually the result of a nested query
  - ANY or SOME (v θ ANY S (or SOME)): returns TRUE if the value v is θ to some value in the set S
    - θ can be any of >, >=, <, <=, and <>
    - When θ is =, ANY/SOME behaves like the IN operator
  - ALL (v θ ALL S): returns TRUE if the value is θ of all values from S
  - EXISTS: evaluates to true if the result of the nested query is not empty
    - "For all" queries must use NOT EXISTS

Uncorrelated Nested Queries

- The nested query is evaluated only once
- Examples:
  - Retrieve the employees whose salary is greater than every salary from an employee who works for department 5

```
SELECT * FROM employee
WHERE salary > ALL (
  SELECT salary FROM employee
  WHERE dno=5);
```

  - Retrieve all projects that do not have any woman working on them

```
SELECT * FROM project
WHERE pnumber NOT IN (
  SELECT w.pno
  FROM employee e JOIN works_on w ON e.ssn=w.essn
  WHERE e.sex=’F’);
```

Correlated Nested Queries

- The nested query is evaluated once for each tuple in the outer query
- Examples:
  - Retrieve the employees who have no female dependent

```
SELECT * FROM employee e
WHERE NOT EXISTS (
  SELECT * FROM dependent d
  WHERE e.ssn=d.essn AND d.sex=’F’);
```

Representing the Relational Division using NOT EXISTS

- Example: Retrieve the names of employees who work on all the projects controlled by department 5

```
SELECT e.fname, e.minit, e.lname FROM employee e
WHERE NOT EXISTS (
  (SELECT p.pnumber FROM project p WHERE p.dnum=5)
  EXCEPT
  (SELECT w.pno FROM works_on w WHERE e.ssn= w.essn)
);

SELECT e.fname, e.minit, e.lname FROM employee e
WHERE NOT EXISTS (
  SELECT * FROM project p
  WHERE p.dnum=5 AND NOT EXISTS
        (SELECT * FROM works_on w WHERE w.essn=e.ssn AND w.pno=p.pnumber)
);
```

Comparisons Involving NULL and Three-Valued Logic

- Meanings of NULL
  - Unknown value
  - Unavailable or withheld value
  - Not applicable attribute
- Each individual NULL value considered to be different from every other NULL value
- SQL uses a three-valued logic:
  - TRUE, FALSE, and UNKNOWN (like Maybe)

Comparisons Involving NULL and Three-Valued Logic (2)

- Operations involving NULL return NULL
  - Examples:
    - NULL + 1 results in NULL
    - (balance + overdraft_limit) can be NULL
- Comparisons with NULL return UNKNOWN
  - Examples:
    - SELECT * FROM employee WHERE super_ssn=NULL;
      - Returns an empty result
    - SELECT * FROM employee WHERE sex<>‘M’;
      - Does not include in the result employees whose sex is either ‘M’ or NULL
- Check whether an attribute value is NULL (IS NULL or IS NOT NULL)
    - SELECT * FROM employee WHERE super_ssn IS NULL;
      - Returns the employees who do not have a direct supervisor

Comparisons Involving NULL and Three-Valued Logic (3)

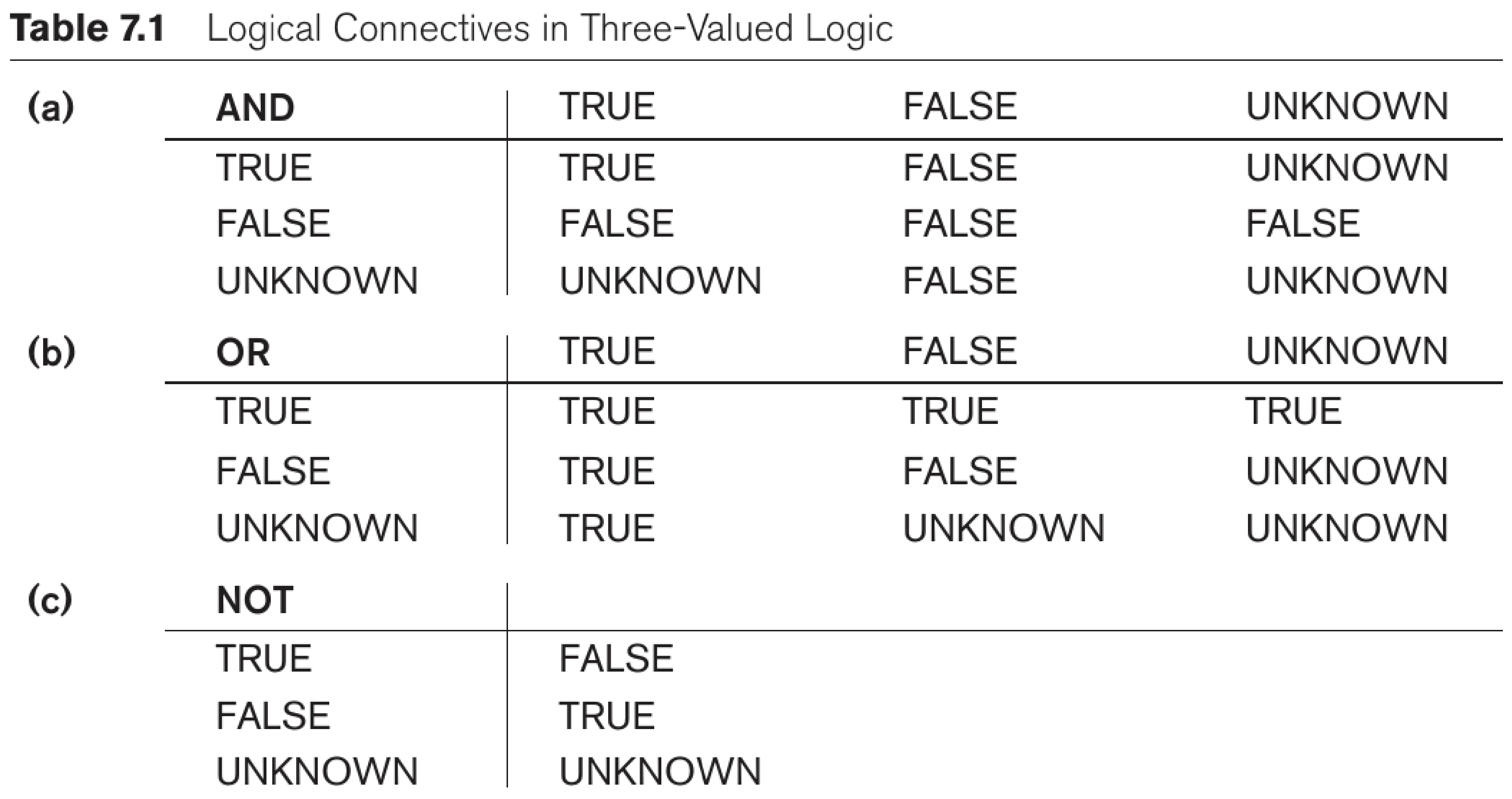


Table 7.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Useful Clauses involving NULLs

- NULLIF
  - Returns NULL if the expressions are equal
  - Example:

```
SELECT AVG(NULLIF(salary, 0.00)) AS avg_sal
FROM employee;
```

- COALESCE
  - Returns the first non-NULL value in a list
  - Example:

```
SELECT COALESCE(update_time, create_time,
‘Unknown’) AS last_updated
FROM file_table;
```

The CASE Statement

- The CASE statement allows to define conditional instructions in SQL queries
- Syntax:

```
-- simple CASE
CASE ("column_name")
  WHEN "value1" THEN "result1"
  WHEN "value2" THEN "result2"
  ...
  [ELSE "resultN"]
END
```

```
-- searched CASE
SELECT CASE
  WHEN "condition1" THEN "result1"
  WHEN "condition2" THEN "result2"
  ...
  [ELSE "resultN"]
END
```

The CASE Statement (2)

- Example of CASE in queries:

```
SELECT ssn, fname, lname,
  CASE (sex)
    WHEN 'M' THEN 'Male'
    WHEN 'F' THEN 'Female'
    ELSE 'Other'
  END,
  CASE
    WHEN age(dt_nasc) >= 65 THEN 'Elder'
    WHEN age(dt_nasc) < 18 THEN 'Youth'
    ELSE 'Adult'
  END AS age_class
FROM employee;
```

The CASE Statement (3)

- Example of CASE in updates:

```
UPDATE employee
SET salary = (
  CASE
    WHEN dno=5 THEN salary*1.15
    WHEN dno=4 THEN salary*1.1
    ELSE salary*1.05
  END);
```

The CASE Statement (4)

- Example of CASE in aggregates:

```
SELECT dno,
  COUNT(
    CASE WHEN salary>30000 THEN 1
      ELSE NULL
    END) AS num_greater_30000,
  COUNT(
    CASE WHEN salary<=30000 THEN 1
      ELSE NULL
    END) AS num_upto_30000
FROM employee
GROUP BY dno;
```

The WITH Clause

- SQL:1999 added the with clause to define “statement scoped views”
- They are not stored in the database schema: instead, they are only valid in the query they belong to
- The with clause is also known as common table expression (CTE)
  - Syntax

```
WITH expression_name1 [(column_list)] AS (SELECT...)
            [expression_name2 [(column_list)] AS
(SELECT...)]
        ...
SELECT… FROM… query_table1...
```

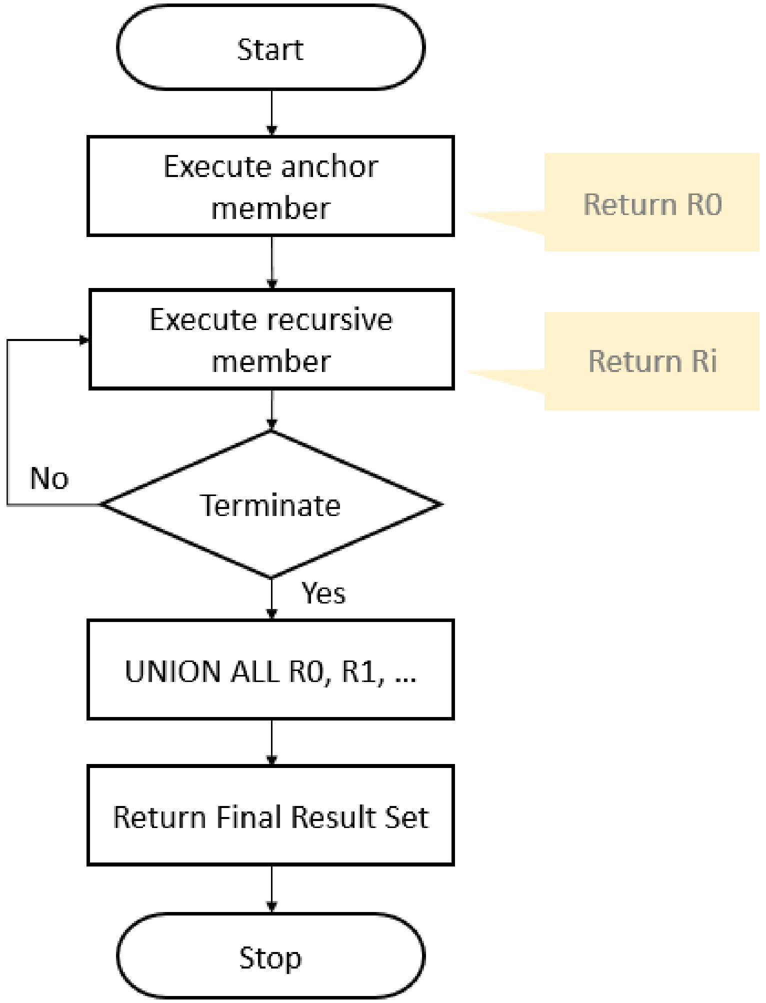
WITH RECURSIVE

- A recursive CTE is a CTE that references itself
- It is useful in querying hierarchical or self-referenced data
  - Examples: multi-level references
- Structure:

```
WITH RECURSIVE expression_name [(column_list)] AS
(
   --a) Anchor member
   initial_query
   UNION ALL
   --b) Recursive member that references expression_name with a termination
condition
   recursive_query
)
--c) references expression name
SELECT *
FROM expression_name
```

WITH RECURSIVE (2)

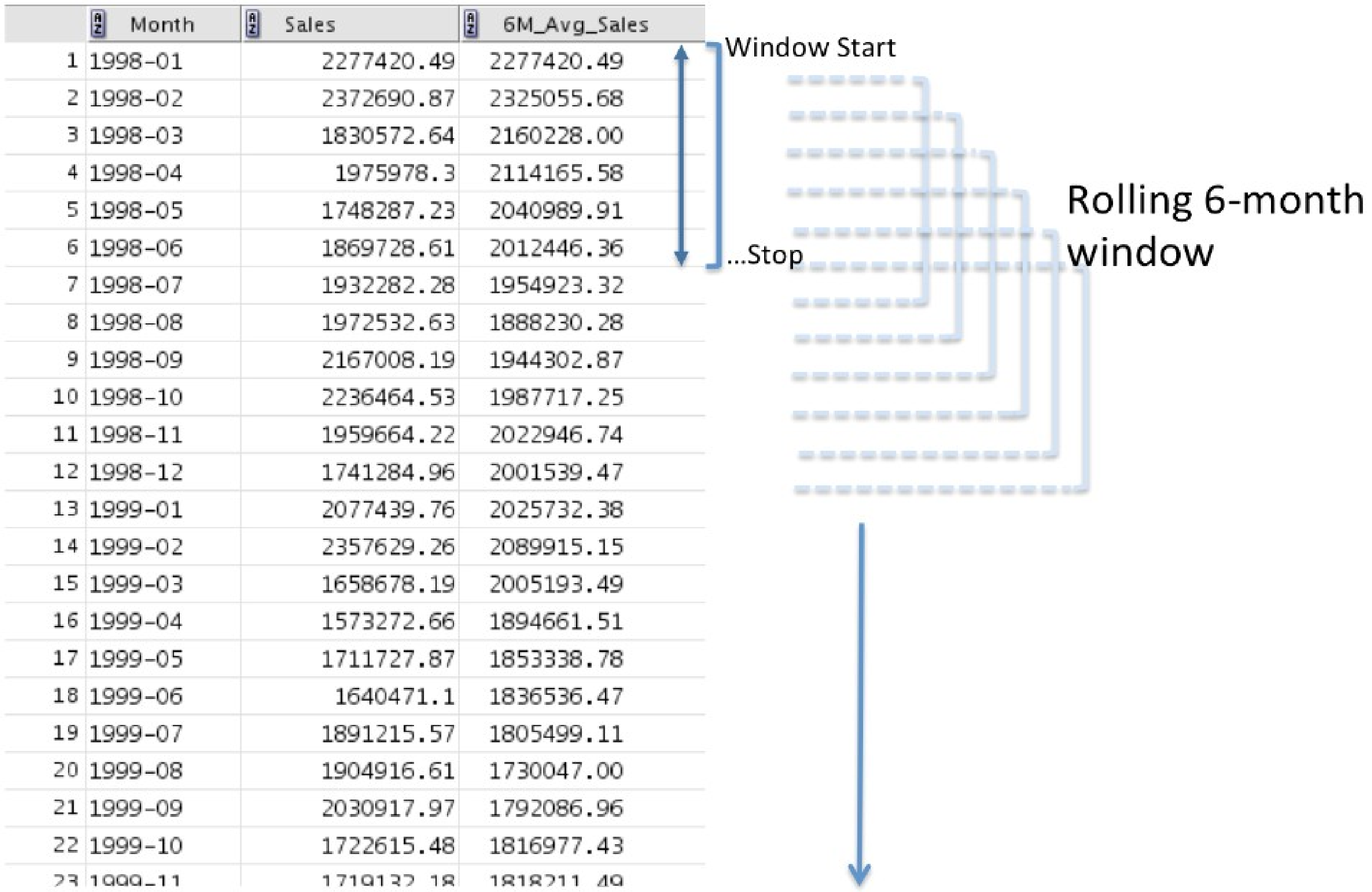
- Query execution



Analytical Functions

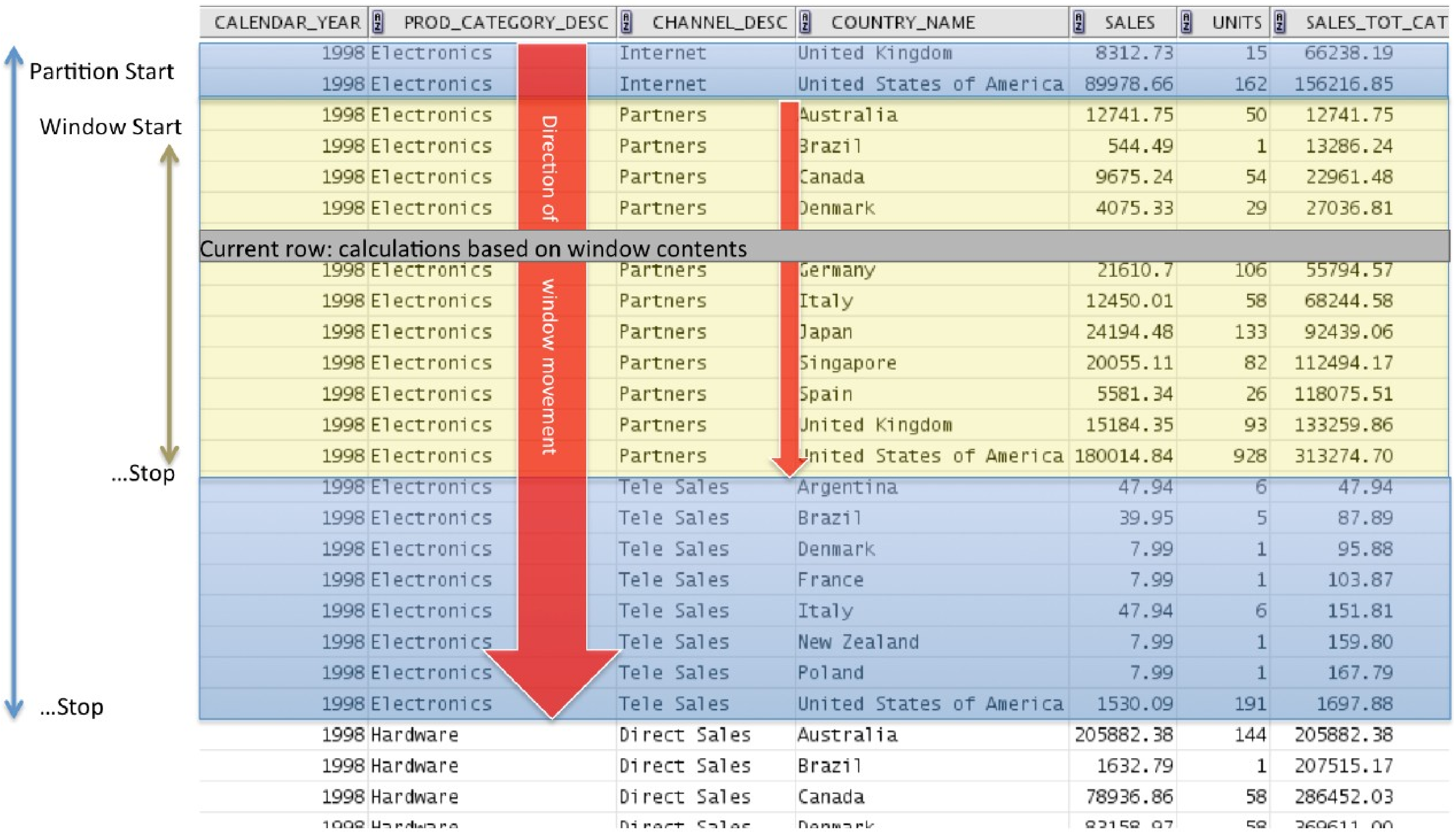
- Analytical functions allow computing several summaries of data
  - Statistics in general, including summations, averages, percentiles, etc.
  - Positioning analyses: first/last, predecessor/sucessor, etc.
- They rely on computations over sliding windows

Computation over sliding windows



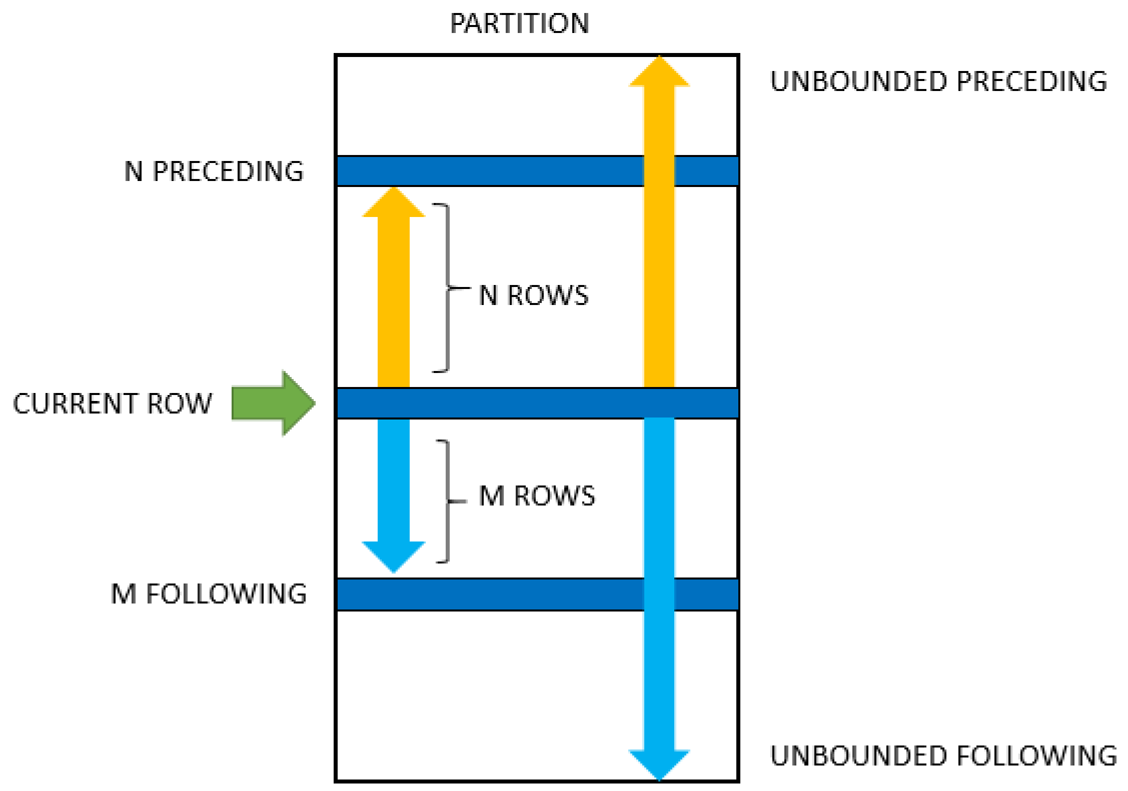
	Month	Sales	6M_Avg_Sales
1	1998-01	2277420.49	2277420.49
2	1998-02	2372690.87	2325055.68
3	1998-03	1830572.64	2160228.00
4	1998-04	1975978.3	2114165.58
5	1998-05	1748287.23	2040989.91
6	1998-06	1869728.61	2012446.36
7	1998-07	1932282.28	1954923.32
8	1998-08	1972532.63	1888230.28
9	1998-09	2167008.19	1944302.87
10	1998-10	2236464.53	1987717.25
11	1998-11	1959664.22	2022946.74
12	1998-12	1741284.96	2001539.47
13	1999-01	2077439.76	2025732.38
14	1999-02	2357629.26	2089915.15
15	1999-03	1658678.19	2005193.49
16	1999-04	1573272.66	1894661.51
17	1999-05	1711727.87	1853338.78
18	1999-06	1640471.1	1836536.47
19	1999-07	1891215.57	1805499.11
20	1999-08	1904916.61	1730047.00
21	1999-09	2030917.97	1792086.96
22	1999-10	1722615.48	1816977.43
23	1999-11	1719132.18	1818211.49

Using Partitioning


Partition Start
Window Start
...Stop
...Stop
Direction of window movement
Current row: calculations based on window contents

CALENDAR_YEAR	PROD_CATEGORY_DESC	CHANNEL_DESC	COUNTRY_NAME	SALES	UNITS	SALES_TOT_CAT
1998	Electronics	Internet	United Kingdom	8312.73	15	66238.19
1998	Electronics	Internet	United States of America	89978.66	162	156216.85
1998	Electronics	Partners	Australia	12741.75	50	12741.75
1998	Electronics	Partners	Brazil	544.49	1	13286.24
1998	Electronics	Partners	Canada	9675.24	54	22961.48
1998	Electronics	Partners	Denmark	4075.33	29	27036.81
1998	Electronics	Partners	Germany	21610.7	106	55794.57
1998	Electronics	Partners	Italy	12450.01	58	68244.58
1998	Electronics	Partners	Japan	24194.48	133	92439.06
1998	Electronics	Partners	Singapore	20055.11	82	112494.17
1998	Electronics	Partners	Spain	5581.34	26	118075.51
1998	Electronics	Partners	United Kingdom	15184.35	93	133259.86
1998	Electronics	Partners	United States of America	180014.84	928	313274.70
1998	Electronics	Tele Sales	Argentina	47.94	6	47.94
1998	Electronics	Tele Sales	Brazil	39.95	5	87.89
1998	Electronics	Tele Sales	Denmark	7.99	1	95.88
1998	Electronics	Tele Sales	France	7.99	1	103.87
1998	Electronics	Tele Sales	Italy	47.94	6	151.81
1998	Electronics	Tele Sales	New Zealand	7.99	1	159.80
1998	Electronics	Tele Sales	Poland	7.99	1	167.79
1998	Electronics	Tele Sales	United States of America	1530.09	191	1697.88
1998	Hardware	Direct Sales	Australia	205882.38	144	205882.38
1998	Hardware	Direct Sales	Brazil	1632.79	1	207515.17
1998	Hardware	Direct Sales	Canada	78936.86	58	286452.03
1998	Hardware	Direct Sales	Denmark	83158.97	58	369611.00

Window Definition



Syntax

```
analytic_function([ arguments ]) OVER
(analytic_clause)

analytic_clause
[ PARTITION BY { expr[, expr ]... | ( expr[,
expr ]... ) } ]
[ order_by_clause [ windowing_clause ] ]

order_by_clause
ORDER [ SIBLINGS ] BY
{ expr | position | c_alias }
[ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
  [, { expr | position | c_alias }
     [ ASC | DESC ]
     [ NULLS FIRST | NULLS LAST ]
  ]...
```

```
windowing_clause
{ ROWS | RANGE }
{ BETWEEN
  { UNBOUNDED PRECEDING
  | CURRENT ROW
  | value_expr { PRECEDING |
                 FOLLOWING }
  } AND
  { UNBOUNDED FOLLOWING
  | CURRENT ROW
  | value_expr { PRECEDING |
                 FOLLOWING }
  }
| { UNBOUNDED PRECEDING
  | CURRENT ROW
  | value_expr PRECEDING
  }
}
```