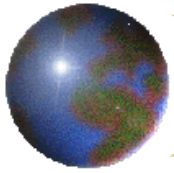


Chapter 6

Relational Algebra



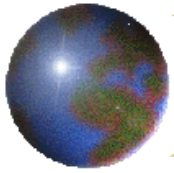
Chapter Outline

- ✚ Unary Relational Operations
- ✚ Relational Algebra Operations from Set Theory
- ✚ Binary relational Operations
- ✚ Additional Relational Operations



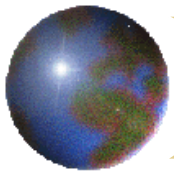
Unary Relational Operations

- ✚ The basic set of operations for relational data model is the **relational algebra**.
- ✚ A sequence of relational algebra operations forms a relational algebra expression.
- ✚ Unary operations take one relation, and return one relation:
 - ✚ SELECT operation
 - ✚ PROJECT operation
 - ✚ Sequences of unary operations
 - ✚ RENAME operation



The SELECT Operation

- ✚ It selects a subset of tuples from a relation that satisfy a SELECT condition.
- ✚ The SELECT operation is denoted by σ (sigma):
$$\sigma_{\langle \text{Selection condition} \rangle} (R)$$
- ✚ The selection condition is a Boolean expression specified on the attributes of relation R



The SELECT Operation

- ✱ The **degree** of the relation resulting from a SELECT operation is the same as that of R

- ✱ For any **selection-condition c**, we have

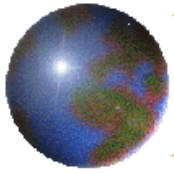
$$| \sigma_{<c>}(R) | \leq | R |$$

- ✱ The SELECT operation is **commutative**, that is,

$$\sigma_{<c1>}(\sigma_{<c2>}(R)) = \sigma_{<c2>}(\sigma_{<c1>}(R))$$

- ✱ We can always **combine cascade** of SELECT operations **into a single SELECT operation** with AND condition.

$$\begin{aligned} & \sigma_{<c1>}(\sigma_{<c2>}(\dots(\sigma_{<cn>}R))) \\ &= \sigma_{<c1> \text{ AND } <c2> \text{ AND } \dots \text{ AND } <cn>}(R) \end{aligned}$$



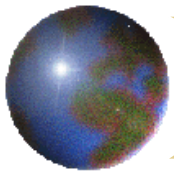
The PROJECT Operation

- The **PROJECT operation**, selects **certain columns from a relation** and **discards the columns** that are not in the PROJECT list

- The PROJECT operation is **denoted by π (pi):**

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

Where **attribute-list** is a list of attributes from the relation R



The PROJECT Operation

- ✚ The **degree** of the relation resulting from a PROJECT operation is equal to the number of attributes in **attribute-list**.
- ✚ If only non-key attributes appear in an attribute-list, duplicate tuples are likely to occur. The PROJECT operation, however, removes any duplicate tuples –this is called **duplicate elimination**
- ✚ The PROJECT operation is **not commutative**

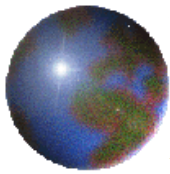


FIGURE 6.1

Results of **SELECT** and **PROJECT** operations.

(a) $\sigma_{(DNO=4 \text{ AND } SALARY>25000) \text{ OR } (DNO=5 \text{ AND } SLARY>30000)}(\text{EMPLOYEE}).$

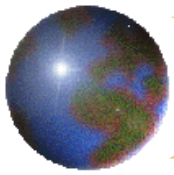
(b) $\pi_{LNAME, FNAME, SALARY}(\text{EMPLOYEE})$

(c) $\pi_{SEX, SALARY}(\text{EMPLOYEE}).$

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Franklin	T	Wong	333445555	1955-12-08	638 Voss,Houston,TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 FireOak,Humble,TX	M	38000	333445555	5

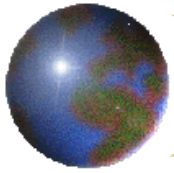
LNAME	FNAME	SALARY
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

SEX	SALARY
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000



Relational Algebra Operations

- ❖ Two relations R_1 and R_2 are said to be **union compatible** if they have the same degree and all their attributes (correspondingly) have the same domain.
- ❖ The **UNION, INTERSECTION, and SET DIFFERENCE** operations are applicable on union compatible relations
- ❖ The resulting relation has the same attribute names as the first relation



The UNION operation

- ✚ The result of UNION operation on two relations, R_1 and R_2 , is a relation, R_3 , that includes all tuples that are either in R_1 , or in R_2 , or in both R_1 and R_2 .

- ✚ The UNION operation is denoted by:

$$R_3 = R_1 \cup R_2$$

- ✚ The UNION operation eliminates duplicate tuples

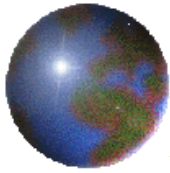


FIGURE 6.3

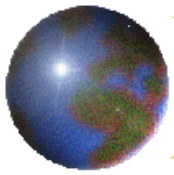
Results of the UNION operation

RESULT = RESULT1 \cup RESULT2.

RESULT1	SSN
	123456789
	333445555
	666884444
	453453453

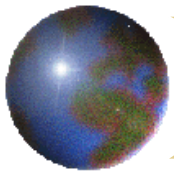
RESULT2	SSN
	333445555
	888665555

RESULT	SSN
	123456789
	333445555
	666884444
	453453453
	888665555



The INTERSECTION operation

- ✚ The result of INTERSECTION operation on two relations, R_1 and R_2 , is a relation, R_3 , that includes all tuples that are in both R_1 and R_2 .
- ✚ The INTERSECTION operation is denoted by:
$$R_3 = R_1 \cap R_2$$
- ✚ The both UNION and INTERSECTION operations are **commutative** and **associative** operations



The SET DIFFERENCE Operation

- ✚ The result of SET DIFFERENCE operation on two relations, R_1 and R_2 , is a relation, R_3 , that includes all tuples that are in R_1 but not in R_2 .
- ✚ The SET DIFFERENCE operation is denoted by:
$$R_3 = R_1 - R_2$$
- ✚ The SET DIFFERENCE (or MINUS) operation **is not commutative**

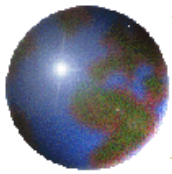


FIGURE 6.4

- (a) Two union-compatible relations. (b) $\text{STUDENT} \cup \text{INSTRUCTOR}$.
(c) $\text{STUDENT} \cap \text{INSTRUCTOR}$. (d) $\text{STUDENT} - \text{INSTRUCTOR}$.
(e) $\text{INSTRUCTOR} - \text{STUDENT}$

(a)

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(b)

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

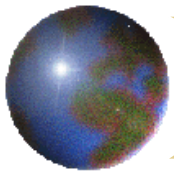
FN	LN
Susan	Yao
Ramesh	Shah

(d)

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson



The CARTESIAN PRODUCT Operation

- ✚ This operation (also known as **CROSS PRODUCT** or **CROSS JOIN**) denoted by:

$$R_3 = R_1 \times R_2$$

- ✚ The resulting relation, R_3 , includes all combined tuples from two relations R_1 and R_2

- ✚ **Degree (R_3) = Degree (R_1) + Degree (R_2)**

- ✚ **$|R_3| = |R_1| \times |R_2|$**

The CARTESIAN PRODUCT (CROSS PRODUCT)

- Building expressions using multiple operations
- Example: $(R1 \times R2)$

$$\sigma_{A=C}(R1 \times R2)$$

A	B
---	---

α	1
β	2

R1

C	D	E
---	---	---

α	10	a
β	10	a
β	20	b
γ	10	b

R2

A	B	C	D	E
---	---	---	---	---

α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

A	B	C	D	E
---	---	---	---	---

α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

R3

Banking Example

- branch (branch_name, branch_city, assets)
- customer (customer_name, customer_street, customer_city)
- account (account_number, branch_name, balance)
- loan (loan_number, branch_name, amount)
- depositor (customer_name, account_number)
- borrower (customer_name, loan_number)

Example Queries

- Find all loans of over \$1200

$$\sigma_{amount > 1200} (loan)$$

- Find the loan number for each loan of an amount greater than \$1200

$$\Pi_{loan_number} (\sigma_{amount > 1200} (loan))$$

loan (loan_number, branch_name, amount)

Example Queries

- Find the names of all customers who have a loan, an account, or both, from the bank

$$\Pi_{customer_name}(borrower) \cup \Pi_{customer_name}(depositor)$$

- Find the names of all customers who have a loan and an account at the bank

$$\Pi_{customer_name}(borrower) \cap \Pi_{customer_name}(depositor)$$

depositor (customer_name, account_number)

borrower (customer_name, loan_number)

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch

$\Pi_{customer_name}(\sigma_{branch_name="Perryridge"}(\sigma_{borrower.loan_number=loan.loan_number}(borrower \times loan)))$

- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank

$\Pi_{customer_name}(\sigma_{branch_name="Perryridge"}(\sigma_{borrower.loan_number=loan.loan_number}(borrower \times loan)))$ —
 $\Pi_{customer_name}(depositor)$

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch

– Answer 1

$$\Pi_{\text{customer_name}}(\sigma_{\text{branch_name} = \text{"Perryridge"}} (\sigma_{\text{borrower.loan_number} = \text{loan.loan_number}} (\text{borrower} \times \text{loan})))$$

– Answer 2

$$\Pi_{\text{customer_name}}(\sigma_{\text{loan.loan_number} = \text{borrower.loan_number}} ((\sigma_{\text{branch_name} = \text{"Perryridge"}} (\text{loan})) \times \text{borrower}))$$

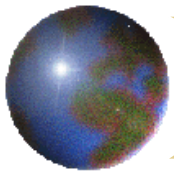
Example Queries

- Find the largest account balance
 - Aggregate max is not directly supported in relational algebra
 - Find those balances that are not the largest
 - Rename account relation as d so that we can compare each account balance with all the others
 - Use set difference to find the max balance accounts

$\Pi_{balance}(account) - \Pi_{account.balance}$

$(\sigma_{account.balance < d.balance} (account \times \rho_d(account)))$

account (account_number, branch_name, balance)



Binary Relational Operations

✚ JOIN operation

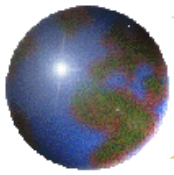
An important operation for any relational database is the **JOIN operation**, because it enables us to **combine related tuples** from two relations into single tuple

✚ The JOIN operation is denoted by:

$$R_3 = R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$$


✚ The degree of resulting relation is

$$\text{degree}(R_1) + \text{degree}(R_2)$$



The JOIN Operation

- ✚ The difference between CARTESIAN PRODUCT and JOIN is that the resulting relation from JOIN consists only those tuples that satisfy the **join condition**
- ✚ The JOIN operation is equivalent to CARTESIAN PRODUCT and then SELECT operation on the result of CARTESIAN PRODUCT operation, if the **select-condition** is the same as the **join condition**


$$R \bowtie_c S = \sigma_c (R \times S)$$

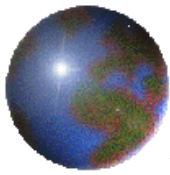
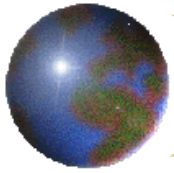


FIGURE 6.6

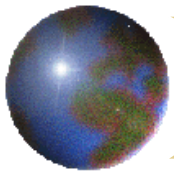
Result of the JOIN operation **DEPT_MGR** ←
DEPARTMENT JOIN _{MGRSSN=SSN} **EMPLOYEE** .

DEPT_MGR	DNAME	DNUMBER	MGRSSN	• • •	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	• • •



The EQUIJOIN Operation

- ✚ If the JOIN operation has equality comparison only (that is, = operation), then it is called an EQUIJOIN operation
- ✚ In the resulting relation on an EQUIJOIN operation, we always have one or more pairs of attributes that have **identical values** in every tuples



The NATURAL JOIN Operation

- ✚ In EQUIJOIN operation, if the two attributes in the join condition have the same name, then in the resulting relation we will have two identical columns. In order to avoid this problem, we define the NATURAL JOIN operation

- ✚ The NATURAL JOIN operation is denoted by:

$$R_3 = R_1 *_{\langle \text{attribute list} \rangle} R_2$$

- ✚ In R_3 only one of the duplicate attributes from the list are kept

Example

- $R = (A, B, C, D)$
- $S = (E, B, D)$
- Result schema = (A, B, C, D, E)
- $r \bowtie s$ is defined as

$$\Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$$

Natural Join Operation – Example

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

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ϵ

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

$r \bowtie s$

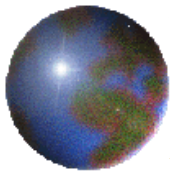


FIGURE 6.7

(a) $\text{PROJ_DEPT} \leftarrow \text{PROJECT} * \text{DEPT.}$

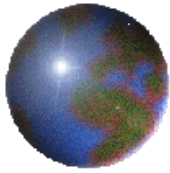
(b) $\text{DEPT_LOCS} \leftarrow \text{DEPARTMENT} * \text{DEPT_LOCATIONS.}$

(a)

PROJ_DEPT	PNAME	<u>PNUMBER</u>	PLOCATION	DNUM	DNAME	MGRSSN	MGRSTARTDATE
	ProductX	1	Bellaire	5	Research	333445555	1988-05-22
	ProductY	2	Sugarland	5	Research	333445555	1988-05-22
	ProductZ	3	Houston	5	Research	333445555	1988-05-22
	Computerization	10	Stafford	4	Administration	987654321	1995-01-01
	Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
	Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

(b)

DEPT_LOCS	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE	LOCATION
	Headquarters	1	888665555	1981-06-19	Houston
	Administration	4	987654321	1995-01-01	Stafford
	Research	5	333445555	1988-05-22	Bellaire
	Research	5	333445555	1988-05-22	Sugarland
	Research	5	333445555	1988-05-22	Houston



A Complete Set of Relational Algebra

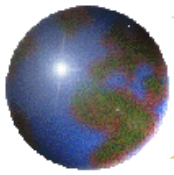
- ✦ The set of relational algebra operations

$$\{\sigma, \pi, \cup, -, \times\}$$

Is a **complete set**. That is, any of the other relational algebra operations can be expressed as a sequence of operations from this set.

For example:

$$R \cap S = (R \cup S) - ((R - S) \cup (S - R))$$



The DIVISION Operation

- ✚ The DIVISION operation is useful for some queries. For example, "Retrieve the name of employees who work on **all** the projects that 'John Smith' works on."

- ✚ The Division operation is denoted by:

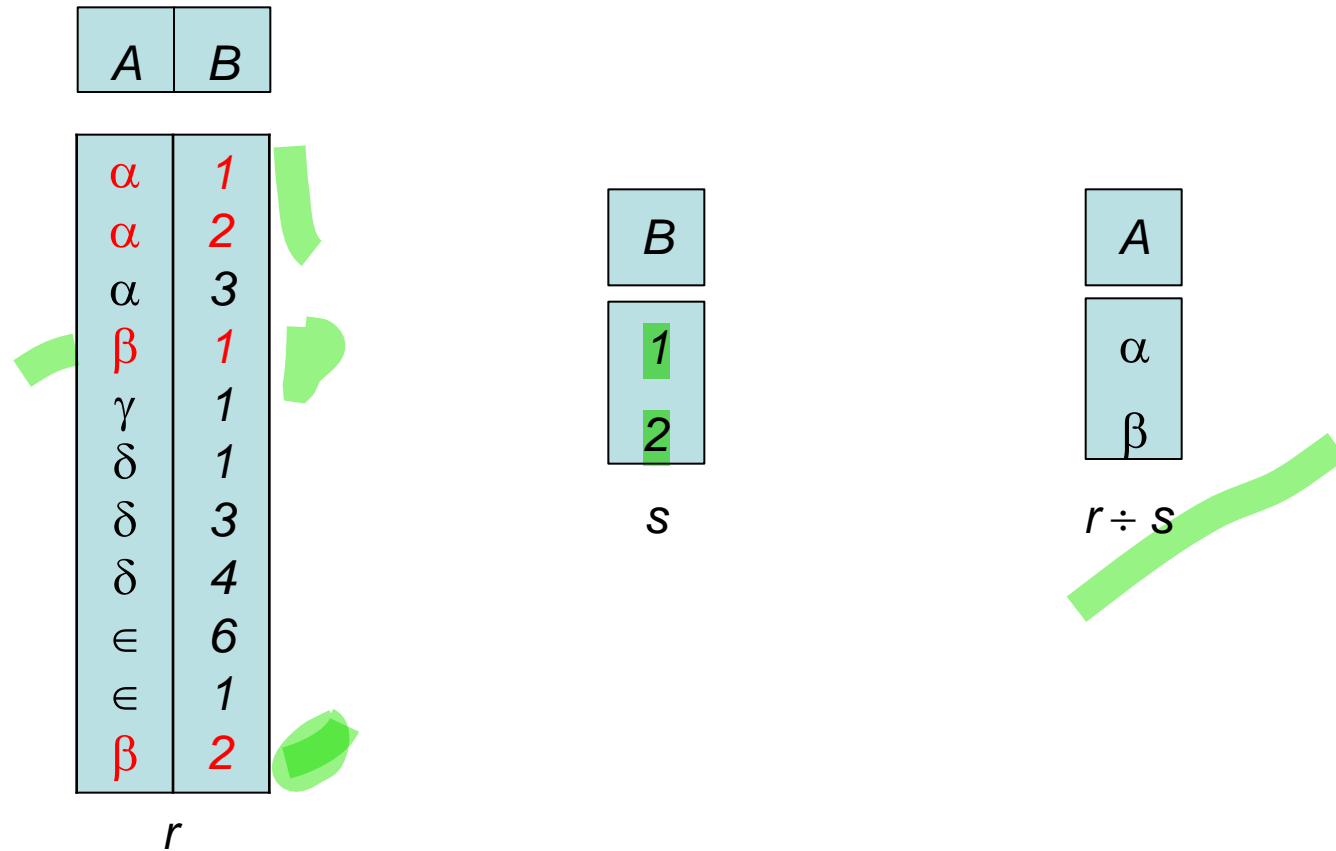

$$R_3 = R_1 \div R_2$$

- ✚ In general, attributes in R_2 are a subset of attributes in R_1

Division Operation

- Binary , Derived Operator.
- Division operator $A \div B$ can be applied if and only if:
- Attributes of B is proper subset of Attributes of A.
- The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
- The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.
- Suited to queries that include the phrase “for all”, “all”, “every”

Division Operation – Example



Another Division Example

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

r

D	E
---	---

a	1
b	1

s

A	B	C
---	---	---

α	a	γ
γ	a	γ

$r \div s$

Example Queries

- Find all customers who have an account at all branches located in Brooklyn city

$$\Pi_{customer_name, branch_name} (depositor \bowtie account) \\ \div \Pi_{branch_name} (\sigma_{branch_city = \text{"Brooklyn"}} (branch))$$

Assignment Operation

- The assignment operation (\leftarrow) provides a convenient way to express complex queries
 - Write query as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as a result of the query
 - Assignment must always be made to a temporary relation variable
- The result to the right of the \leftarrow is assigned to the relation variable on the left of the \leftarrow
 - May use variable in subsequent expressions
 - $\text{PROJ_DEPT} \leftarrow \text{PROJECT} * \text{DEPT}$

Bank Example Queries

- Find the names of all customers who have a loan and an account at bank

$$\Pi_{customer_name}(borrower) \cap \Pi_{customer_name}(depositor)$$

- Find the name of all customers who have a loan at the bank and the loan amount

$$\Pi_{customer_name, loan_number, amount}(borrower \bowtie loan)$$

Bank Example Queries

- Find all customers who have an account from at least the “Downtown” and the “Uptown” branches

– Answer 1

$$\Pi_{customer_name} (\sigma_{branch_name = \text{“Downtown”}} (depositor \bowtie account)) \cap \Pi_{customer_name} (\sigma_{branch_name = \text{“Uptown”}} (depositor \bowtie account))$$

– Answer 2: using a constant relation

$$\Pi_{customer_name, branch_name} (depositor \bowtie account) \div \rho_{temp(branch_name)} (\{(\text{“Downtown”}), (\text{“Uptown”})\})$$

Specifying Joined Tables in the FROM Clause of SQL

- **Joined table**

- Permits users to specify a table resulting from a join operation in the FROM clause of a query

- **The FROM clause in Q1A**

- Contains a single joined table. JOIN may also be called INNER JOIN

```
Q1A:  SELECT  Fname, Lname, Address
      FROM    (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
      WHERE   Dname='Research';
```


Different Types of JOINed Tables in SQL

- Specify different types of join
 - NATURAL JOIN
 - Various types of OUTER JOIN (LEFT, RIGHT, FULL)
- NATURAL JOIN on two relations R and S
 - No join condition specified
 - Is equivalent to an implicit EQUIJOIN condition for each pair of attributes with same name from R and S

NATURAL JOIN

- Rename attributes of one relation so it can be joined with another using NATURAL JOIN:

```
Q1B:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE NATURAL JOIN
                   (DEPARTMENT AS DEPT (Dname, Dno, Mssn,
                                         Msdate)))
        WHERE     Dname='Research';
```

The above works with **EMPLOYEE.Dno = DEPT.Dno** as an implicit join condition

INNER and OUTER Joins

- **INNER JOIN (versus OUTER JOIN)**
 - Default type of join in a joined table
 - Tuple is included in the result only if a matching tuple exists in the other relation
- **LEFT OUTER JOIN**
 - Every tuple in left table must appear in result
 - If no matching tuple
 - Padded with NULL values for attributes of right table

INNER and OUTER Joins

- RIGHT OUTER JOIN

- Every tuple in right table must appear in result
- If no matching tuple
 - Padded with NULL values for attributes of left table

- Full OUTER JOIN

- Every tuple in right and left tables must appear in the result
- If no matching tuple
 - Padded with NULL values for attributes of left and right tables

INNER Join

- INNER JOIN (versus OUTER JOIN)

- Default type of join in a joined table
- Tuple is included in the result only if a matching tuple exists in the other relation

Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

HOD

CID	Name
100	Rohan
102	Sara
104	Jiya

Courses ⋈ HOD

CID	Course	Name
100	Database	Rohan
102	Electronics	Sara

Left OUTER Join

- In **Left outer join**, *all the tuples from the Left relation R1 are included in the resulting relation.* The tuples of R1 which do not satisfy join condition will have values as **NULL** for attributes of R2.
- In short:
 - ▣ All record from **left** table
 - ▣ Only matching records from right table
- **Symbol:** \bowtie
- **Notation:** **R1** \bowtie **R2**

Left OUTER Join

- LEFT OUTER JOIN

- Every tuple in left table must appear in result
- If no matching tuple
 - Padded with NULL values for attributes of right table

Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

HOD

CID	Name
100	Rohan
102	Sara
104	Jiya

Courses ⋈ HOD

CID	Course	Name
100	Database	Rohan
101	Mechanics	NULL
102	Electronics	Sara

Right OUTER Join

- In **Right outer join**, *all the tuples from the right relation R2 are included in the resulting relation*. The tuples of R2 which do not satisfy join condition will have values as **NULL** for attributes of R1.
- In short:
 - **All** record from **right** table
 - Only matching records from left table
- **Symbol:** \bowtie
- **Notation:** **R1** \bowtie **R2**

Right OUTER Join

■ RIGHT OUTER JOIN

- Every tuple in right table must appear in result
- If no matching tuple
 - Padded with NULL values for attributes of left table

Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

HOD

CID	Name
100	Rohan
102	Sara
104	Jiya

Courses ⋈ HOD

CID	Course	Name
100	Database	Rohan
102	Electronics	Sara
104	NULL	Jiya

Full OUTER Join

□ In **Full outer join**, all the tuples from both Left relation $R1$ and right relation $R2$ are included in the resulting relation. The tuples of both relations $R1$ and $R2$ which do not satisfy join condition, their respective unmatched attributes are made **NULL**.

□ In short:

□ All record from **all** table

□ **Symbol:** \bowtie

□ **Notation:** $R1 \bowtie R2$

Full OUTER Join

- Full OUTER JOIN

- Every tuple in right and left tables must appear in the result
- If no matching tuple
 - Padded with NULL values for attributes of left and right tables

Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

HOD

CID	Name
100	Rohan
102	Sara
104	Jiya

Courses ⋈ HOD

CID	Course	Name
100	Database	Rohan
101	Mechanics	NULL
102	Electronics	Sara
104	NULL	Jiya

Assignment: Perform inner and outer joins(left, right and full outer joins)

loan

<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

<i>customer-name</i>	<i>loan-number</i>
Jones	L-170
Smith	L-230
Hayes	L-155

Example: LEFT OUTER JOIN

```
SELECT E.Lname AS Employee_Name  
       S.Lname AS Supervisor_Name  
  
FROM Employee AS E LEFT OUTER JOIN EMPLOYEE AS S  
       ON E.Super_ssn = S.Ssn)
```

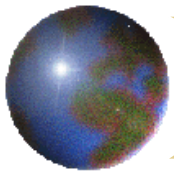
ALTERNATE SYNTAX:

```
SELECT E.Lname , S.Lname  
FROM EMPLOYEE E, EMPLOYEE S  
WHERE E.Super_ssn + = S.Ssn
```

Multiway JOIN in the FROM clause

- FULL OUTER JOIN – combines result if LEFT and RIGHT OUTER JOIN
- Can nest JOIN specifications for a multiway join:

Q2A: **SELECT** Pnumber, Dnum, Lname, Address, Bdate
 FROM ((PROJECT **JOIN** DEPARTMENT **ON**
 Dnum=Dnumber) **JOIN** EMPLOYEE **ON**
 Mgr_ssn=Ssn)
 WHERE Plocation='Stafford';



Additional Relational Operations

- Statistical queries cannot be specified in the basic relational algebra operation
- We define **aggregate functions** that can be applied over numeric values such as, SUM, AVERAGE, MAXIMUM, MINIMUM, and COUNT
- The aggregate function is denoted by

$\sigma_{\langle \text{grouping attributes} \rangle} \delta_{\langle \text{function list} \rangle} (R)$

where aggregation (grouping) is based on the **$\langle \text{attributes-list} \rangle$** . If not present, just 1 group

Aggregate Functions in SQL

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

FILTER ON GROUP BY

```
SELECT Salesperson, SUM(Amount)
AS TotalSales
FROM Sales
GROUP BY Salesperson
HAVING SUM(Amount) > 500;
```


Renaming Results of Aggregation

- Following query returns a single row of computed values from EMPLOYEE table:

Q19:

```
SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG  
       (Salary)  
FROM EMPLOYEE;
```

- The result can be presented with new names:

```
Q19A:      SELECT      SUM (Salary) AS Total_Sal, MAX (Salary) AS
Highest_Sal, MIN (Salary) AS Lowest_Sal, AVG
(Salary) AS Average_Sal
FROM      EMPLOYEE;
```

Aggregate Functions in SQL (cont'd.)

- NULL values are discarded when aggregate functions are applied to a particular column

Query 20. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
Q20:  SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
      FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
      WHERE     Dname='Research';
```

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:  SELECT    COUNT (*)
      FROM      EMPLOYEE;
```

```
Q22:  SELECT    COUNT (*)
      FROM      EMPLOYEE, DEPARTMENT
      WHERE     DNO=DNUMBER AND DNAME='Research';
```

Aggregate Functions on Booleans

- SOME and ALL may be applied as functions on Boolean Values.
- SOME returns true if at least one element in the collection is TRUE (similar to OR)
- ALL returns true if all of the elements in the collection are TRUE (similar to AND)

```
SELECT EmployeeID,  
       CASE  
         WHEN SOME(Completed) THEN 'At least one task completed'  
         ELSE 'No tasks completed'  
       END AS TaskStatus  
FROM Tasks  
GROUP BY EmployeeID;
```

Grouping: The GROUP BY Clause

- **Partition** relation into subsets of tuples
 - Based on **grouping attribute(s)**
 - Apply function to each such group independently
- **GROUP BY** clause
 - Specifies grouping attributes
- **COUNT (*)** counts the number of rows in the group

Examples of GROUP BY

- The grouping attribute must appear in the SELECT clause:

```
Q24:      SELECT      Dno, COUNT (*), AVG (Salary)
           FROM        EMPLOYEE
           GROUP BY    Dno;
```

- If the grouping attribute has NULL as a possible value, then a separate group is created for the null value (e.g., null Dno in the above query)
- GROUP BY may be applied to the result of a JOIN:

```
Q25:      SELECT      Pnumber, Pname, COUNT (*)
           FROM        PROJECT, WORKS_ON
           WHERE        Pnumber=Pno
           GROUP BY    Pnumber, Pname;
```

Grouping: The GROUP BY and HAVING Clauses (cont'd.)

- **HAVING** clause

- Provides a condition to select or reject an entire group:

- **Query 26.** For each project *on which more than two employees work*, retrieve the project number, the project name, and the number of employees who work on the project.

```
Q26:      SELECT      Pnumber, Pname, COUNT (*)
          FROM        PROJECT, WORKS_ON
          WHERE       Pnumber=Pno
          GROUP BY    Pnumber, Pname
          HAVING      COUNT (*) > 2;
```