UNIT-3

- Formal Relational Query Languages
- Relational Database Design

Formal Relational Query Languages

- Relational Algebra
- Relational Calculus
  - Domain Relational Calculus
  - Tuple Relational Calculus

Relational Query Languages

- A major strength of the relational model: supports simple, powerful *querying* of data.
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - The key: precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.
- Query languages can be categorized into 2 types
  - 1.Procedural Language
  - 2.Non-procedural Language

Relational Query Languages(contd.)

- **Procedural Language:** user instructs the system to perform a sequence of operations on the database to compute the desired result.
  - Relational Algebra is one of the Procedural Language
- **Non Procedural Language:** User describes the desired information without giving a specific procedure for obtaining that information
  - Relational calculus is one of the Non Procedural Language
  - RC again categorized into TRC,DRC

Relational Algebra

- Relational Algebra is formal description of how relational database operates.
- It is a procedural query language, i.e. user must define both "how" and "what" to retrieve.
- It consists of a set of operators that consume either one or two relations as input.
  - An operator produces one relation as its output.

Relational Algebra

- The fundamental operations are :
  - Select
  - Project
  - Union
  - Set difference
  - Cartesian Product
  - Rename
  - Set intersection
  - Division
  - Assignment
  - Natural join

Algebra Operations

- Unary Operations - operate on one relation.
  - select, project and rename operators.
- Binary Operations - operate on pairs of relations.
  - union, set difference, division, Cartesian product, equality join, natural join, join and semi-join operators.

Select Operator

- The **Select** operator selects tuples that satisfies a predicate; e.g. retrieve the employees whose salary is 30,000
  - $\sigma_{Salary\ =\ 30,000}(Employee)$
- <u>Conditions in Selection:</u>
  - Simple Condition: *(attribute)(comparison)(attribute)*
    - *(attribute)(comparison)(constant)*
  - Comparison: $=, \neq, \leq, \geq, <, >$
  - Condition: *combination of simple conditions with AND, OR, NOT*
-

Select Operator Example

Person

Name	Age	Weight
Harry	34	80
Sally	28	64
George	29	70
Helena	54	54
Peter	34	80

$\sigma_{Age \geq 34}(\text{Person})$

Name	Age	Weight
Harry	34	80
Helena	54	54
Peter	34	80

$\sigma_{Age=Weight}(\text{Person})$

Name	Age	Weight
Helena	54	54

Project Operator

- **Project** ($\prod$) retrieves a column. Duplication is not permitted.
- e.g., name of employees:
  - $\prod_{name}(\text{Employee})$

Employee

Name	Age	Salary
Harry	34	80,000
Sally	28	90,000
George	29	70,000
Helena	54	54,280
Peter	34	40,000

$\prod_{name}(\text{Employee})$

Name
Harry
Sally
George
Helena
Peter

Composite Example

Eg: Name of employees earning more than 80,000:

$\prod_{name}(\sigma_{Salary>80,000}(\text{Employee}))$

Employee

Name	Age	Salary
Harry	34	80,000
Sally	28	90,000
George	29	70,000
Helena	54	54,280
Peter	34	40,000

$\sigma_{Salary>80,000}(\text{Employee})$

Name	Age	Salary
Sally	28	90,000

$\prod_{name}(\sigma_{Salary>80,000}(\text{Employee}))$

Name
Sally

Cartesian Product

- In mathematics, it is a set of all pairs of elements (x, y) that can be constructed from given sets, X and Y, such that x belongs to X and y to Y.
- It defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.

Cartesian Product Example

Person

Name	Age	Weight
Harry	34	80
Sally	28	64
George	29	70

City

City
San Jose
Austin

Person X City

Name	Age	Weight	City
Harry	34	80	San Jose
Sally	28	64	San Jose
George	29	70	San Jose
Harry	34	80	Austin
Sally	28	64	Austin
George	29	70	Austin

Example

Cartesian Product (×)

- Allows us to combine information from any two relations. Cartesian product of relations r_1 and r_2 are represented as $r_1 \times r_2$.

r_1

A	B
a_1	b_1
a_2	b_2

×

r_2

B	C	D
b_1	c_1	d_1
b_2	c_2	d_2
b_3	c_3	d_3

=

r

A	r_1.B	r_2.B	C	D
a_1	b_1	b_1	c_1	d_1
a_1	b_1	b_2	c_2	d_2
a_1	b_1	b_3	c_3	d_3
a_2	b_2	b_1	c_1	d_1
a_2	b_2	b_2	c_2	d_2
a_2	b_2	b_3	c_3	d_3

Rename Operator

- In relational algebra, a **rename** is a unary operation written as $\rho_{a/b}(R)$
  - where *a* and *b* are attribute names and *R* is a relation
- The result is identical to *R* except that the *b* field in all tuples is renamed to an *a* field.
- Example
  - $P_{\textbf{employeename/Name}}$(Emp)
  - Changes the name of column 'Name' to 'EmployeeName' in Emp table

Rename Operator Example

Employee

Name	Salary
Harry	80,000
Sally	90,000
George	70,000
Helena	54,280
Peter	40,000

$\rho_{EmployeeName\,/\,Name}(Employee)$

EmployeeName	Salary
Harry	80,000
Sally	90,000
George	70,000
Helena	54,280
Peter	40,000

Union Operator

- The **union** operation is denoted U as in set theory.
- It returns the union (set union) of two compatible relations.
- For a union operation r U s to be legal, we require that,
  - r and s must have the same number of attributes.
  - The domains of the corresponding attributes must be the same.
- As in all set operations, duplicates are eliminated.

Union Operator Example

Student

FN	LN
Susan	Yao
Ramesh	Shah
Barbara	Jones
Amy	Ford
Jimmy	Wang

Professor

FN	LN
John	Smith
Ricardo	Brown
Susan	Yao
Francis	Johnson
Ramesh	Shah

Student U Professor

FN	LN
Susan	Yao
Ramesh	Shah
Barbara	Jones
Amy	Ford
Jimmy	Wang
John	Smith
Ricardo	Brown
Francis	Johnson

Intersection Operator

- Denoted as ∩ .
- For relations R and S, intersection is R ∩ S.
  - Defines a relation consisting of the set of all tuples that are in both R and S.
  - R and S must be union-compatible.
- Expressed using basic operations:
- $R \cap S = R - (R - S)$

Intersection Operator Example

Student

FN	LN
Susan	Yao
Ramesh	Shah
Barbara	Jones
Amy	Ford
Jimmy	Wang

Professor

FN	LN
John	Smith
Ricardo	Brown
Susan	Yao
Francis	Johnson
Ramesh	Shah

Student ∩ Professor

FN	LN
Susan	Yao
Ramesh	Shah

Set Difference Operator

- For relations R and S,
  - Set difference R - S, defines a relation consisting of the tuples that are in relation R, but not in S.
  - Set difference S – R, defines a relation consisting of the tuples that are in relation S, but not in R.

Set Difference Operator Example

Student

FN	LN
Susan	Yao
Ramesh	Shah
Barbara	Jones
Amy	Ford
Jimmy	Wang

Professor

FN	LN
John	Smith
Ricardo	Brown
Susan	Yao
Francis	Johnson
Ramesh	Shah

Professor - Student

FN	LN
John	Smith
Ricardo	Brown
Francis	Johnson

Student - Professor

FN	LN
Barbara	Jones
Amy	Ford
Jimmy	Wang

Division Operator

- The division operator takes as input two relations, called the dividend relation (*r* on schema *R*) and the divisor relation (*s* on schema *S)* such that all the attributes in *S* also appear in *R* and *S* is not empty.
- The output of the division operation is a relation on schema *R* with all the attributes common with *S*.

Division Operator Example

Completed

Student	Task
Fred	Database1
Fred	Database2
Fred	Compiler1
Eugene	Database1
Sara	Database1
Sara	Database2
Eugene	Compiler1

DBProject

Task
Database1
Database2

Completed / DBProject

Student
Fred
Sara

DIVISION

MAIN

A	B
A1	B1
A3	B2
A2	B3
A2	B1
A1	B3
A3	B3
A3	B1

SUB1

B
B1
B3

SUB2

B
B2

SUB3

B
B2
B3

ASSIGNMENT

R1←MAIN ÷ SUB1

R2←MAIN÷SUB2

R3←MAIN÷SUB3

RESULTING TABLES

R1

A
A1
A2
A3

R2

A
A3

R3

A
A3

12

Joins

- It is used to combine information from two or more relations
- Ii is defined as a cross product followed by selections and projections
- Joins are of two types
  - Inner Joins
  - Outer Joins
- **Inner Joins**: An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation.
  - Condition Join, Equijoin, and Natural Join are called inner joins.

Joins(contd..)

- **Outer Joins**: To include all the tuples from the participating relations in the resulting relation. (it allows NULL values)

- There are three kinds of outer joins –
  - left outer join,
  - right outer join, and
  - full outer join.

Notation

Operation	My HTML	Symbol
Projection	PROJECT	π
Selection	SELECT	σ
Renaming	RENAME	ρ
Union	UNION	$\cup$
Intersection	INTERSECTION	$\cap$
Assignment	<-	$\leftarrow$

Operation	My HTML	Symbol
Cartesian product	X	$\times$
Join	JOIN	$\bowtie$
Left outer join	LEFT OUTER JOIN	⟕
Right outer join	RIGHT OUTER JOIN	⟖
Full outer join	FULL OUTER JOIN	⟗
Semijoin	SEMIJOIN	$\ltimes$

Condition Join Operator

- The join operation accepts a join condition c and a pair of relation instances as arguments and returns a relation instance
- $R \bowtie_c S = \sigma_c(R \times S)$

Example: Bag Theta-Join

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R JOIN $_{R.B<S.B}$ S =

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	7	8
1	2	3	4
1	2	7	8

Natural Join Operator

- Natural join is a dyadic operator that is written as *R* lXl *S* where *R* and *S* are relations.
- The result of the natural join is the set of all combinations of tuples in *R* and *S* that are equal on their common attribute names.

Natural Join Example

For an example, consider the tables *Employee* and *Dept* and their natural join:

Employee

Name	EmpID	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Mgr
Finance	George
Sales	Harriet
Production	Charles

Employee lXl Dept

Name	EmpID	DeptName	Mgr
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

Semijoin Operator

- The semijoin is joining similar to the natural join and written as $R \ltimes S$ where R and S are relations.
- The result of the semijoin is only the set of all tuples in R for which there is a tuple in S that is equal on their common attribute names.

Semijoin Example

For an example consider the tables *Employee* and *Dept* and their semi join:

Employee

Name	EmpID	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Mgr
Sales	Harriet
Production	Charles

Employee $\ltimes$ Dept

Name	EmpID	DeptName
Sally	2241	Sales
Harriet	2202	Sales

Outer joins

- **Left outer join**
- The left outer join is written as R =X S where R and S are relations.
- The result of the left outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuples in R that have no matching tuples in S.

Left Outerjoin Example

For an example consider the tables *Employee* and *Dept* and their left outer join:

Employee

Name	EmpID	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Mgr
Sales	Harriet

Employee =X Dept

Name	EmpID	DeptName	Mgr
Harry	3415	Finance	NULL
Sally	2241	Sales	Harriet
George	3401	Finance	NULL
Harriet	2202	Sales	Harriet

Outer joins(contd..)

- **Right outer join**
- The right outer join is written as *R* X= *S* where *R* and *S* are relations.
- The result of the right outer join is the set of all combinations of tuples in *R* and *S* that are equal on their common attribute names, in addition to tuples in *S* that have no matching tuples in *R*.

Right Outerjoin Example

For an example consider the tables *Employee* and *Dept* and their right outer join:

Employee

Name	EmpID	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Mgr
Sales	Harriet
Production	Charles

Employee X= Dept

Name	EmpID	DeptName	Mgr
Sally	2241	Sales	Harriet
Harriet	2202	Sales	Harriet
NULL	NULL	Production	Charles

Full Outer join Example

The **outer join** or **full outer join** in effect combines the results of the left and right outer joins.

Employee

Name	EmpID	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Mgr
Sales	Harriet
Production	Charles

Employee =X= Dept

Name	EmpID	DeptName	Mgr
Harry	3415	Finance	NULL
Sally	2241	Sales	Harriet
George	3401	Finance	NULL
Harriet	2202	Sales	Harriet
NULL	NULL	Production	Charles

Relational algebra Notation

Projection	PROJECT	π
Selection	SELECT	σ
Renaming	RENAME	ρ
Union	UNION	$\cup$
Intersection	INTERSECTION	$\cap$
Assignment	<-	$\leftarrow$

Cartesian product	X	$\times$
Join	JOIN	$\bowtie$
Left outer join	LEFT OUTER JOIN	⟕
Right outer join	RIGHT OUTER JOIN	⟖
Full outer join	FULL OUTER JOIN	⟗
Semijoin	SEMIJOIN	$\ltimes$

SHIRAJ MOHAMED M | MIS

2

Relational Calculus

- Comes in two flavors: *Tuple relational calculus* (TRC) and *Domain relational calculus* (DRC).
- Calculus has *variables, constants, comparison ops, logical connectives* and *quantifiers*.
  - *TRC*: Variables range over (i.e., get bound to) *tuples*.
  - *DRC*: Variables range over *domain elements* (= field values).
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called *formulas*. An answer tuple is essentially an

DRC Formulas

- *Atomic formula:*
  - $\langle x1,x2,...,xn \rangle \in Rname$, or X *op* Y, or X *op* constant
  - *op* is one of $<,>,=,\leq,\geq,\neq$
- *Formula:*
  - an atomic formula, or
  - $\neg p, p \wedge q, p \vee q$, where p and q are formulas, or
  - $\exists X(p(X))$, where X is *a domain variable* or
  - $\forall X(p(X))$, where X is *a domain variable.*
- The use of quantifiers $\exists X$ and $\forall X$ is said to *bind* X.

Free and Bound Variables

- The use of quantifiers $\exists X$ and $\forall X$ in a formula is said to *bind* X.
  - A variable that is not bound is free.
- Let us revisit the definition of a query:

$$\left\{ \langle x1,x2,...,xn \rangle \mid p\left(\langle x1,x2,...,xn \rangle \right) \right\}$$

❖ There is an important restriction: the variables x1, ..., xn that appear to the left of `|' must be the *only* free variables in the formula p(...).

Find sailors rated > 7 who've reserved a red boat

$$\left\{ \langle I,N,T,A \rangle \mid \langle I,N,T,A \rangle \in Sailors \wedge T > 7 \wedge \right.$$
$$\exists\ Ir,Br,D \left(\langle Ir,Br,D \rangle \in \text{Reserves} \wedge Ir = I \wedge \right.$$
$$\left. \left. \exists\ B,BN,C \left(\langle B,BN,C \rangle \in Boats \wedge B = Br \wedge C = 'red' \right) \right) \right\}$$

- Observe how the parentheses control the scope of each quantifier's binding.
- This may look cumbersome, but with a good user interface, it could be intuitive. (MS Access, QBE)

Find sailors who've reserved all boats

$$\{\langle I,N,T,A\rangle \mid \langle I,N,T,A\rangle \in Sailors \wedge$$
$$\forall\ B,BN,C\ (\neg(\langle B,BN,C\rangle \in Boats)\ \vee$$
$$(\exists\ Ir,Br,D\ (\langle Ir,Br,D\rangle \in \text{Reserves} \wedge I=Ir \wedge Br=B)))\}$$

- Find all sailors *I* such that for each 3-tuple $\langle B,BN,C\rangle$ either it is not a tuple in Boats or there is a tuple in Reserves showing that sailor *I* has reserved it.

Find sailors who've reserved all boats (again!)

$$\{\langle I,N,T,A\rangle \mid \langle I,N,T,A\rangle \in Sailors \wedge$$
$$\forall\ \langle B,BN,C\rangle \in Boats$$
$$(\exists \langle Ir,Br,D\rangle \in \text{Reserves}(I=Ir \wedge Br=B))\}$$

- Simpler notation, same query. (Much clearer!)
- To find sailors who've reserved all red boats:

$$\ldots\ (C \neq 'red' \vee \exists \langle Ir,Br,D\rangle \in \text{Reserves}(I=Ir \wedge Br=B))\}$$

Any other way to specify it? Equivalence in logic

Unsafe Queries, Expressive Power

- It is possible to write syntactically correct calculus queries that have an infinite number of answers! Such queries are called *unsafe*.
  - e.g., $\{S \mid \neg(S \in Sailors)\}$
- It is known that every query that can be expressed in relational algebra can be expressed as a safe query in DRC / TRC; the converse is also true.
- *Relational Completeness*: Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus.

Tuple Relational Calculus

- Interested in finding tuples for which a predicate is true. Based on use of <u>tuple variables</u>.
- Tuple variable is a variable that 'ranges over' a named relation: i.e., variable whose only permitted values are tuples of the relation.
- Specify range of a tuple variable *S* as the Staff relation as:
  Staff(S)
- To find set of all tuples S such that P(S) is true:
  {S | P(S)}

Tuple relational calculus

- Similar to DRC except that variables range over **tuples** rather than field values
- For example, the query "Find all sailors with rating above 7" is represented in TRC as follows:

  $\{S \mid S \in Sailors \land S.rating > 7\}$

Example

Find names and ages of sailors with a rating above 7

$$\{P \mid \exists S \in Sailors.\ S.rating > 7 \land P.sname = S.sname \land P.age = S.age\}$$

Recall P ranges over tuple values

Example

Find the names of sailors who have reserved at least two boats

$$\{ P \mid \exists S \in Sailors.\ \exists R_1 \in Reserves.\ \exists R_2 \in Reserves.\ S.sid = R_1.sid \land R_1.sid = R_2.sid \land R_1.bid \neq R_2.bid \land P.sname = S.sname\}$$

Encoding relational calculus

- Can we code up the relational calculus in the relational algebra?
- At the moment, NO!
- Given our syntax we can define 'problematic' queries such as

  $\{S \mid \neg (S \in Sailors)\}$
- This (presumably) means the set of all tuples that are not sailors, which is an infinite set... ☹

Summary

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version.