Relational Query Languages

Languages of DBMS
- Data Definition Language DDL
  - define the schema and storage stored in a Data Dictionary
- Data Manipulation Language DML
  - Manipulative populate schema, update database
  - Retrieval querying content of a database
- Data Control Language DCL
  - permissions, access control etc...

Data Manipulation Language
- Theory behind operations is formally defined and equivalent to a first-order logic (FOL)
- Relational Calculus (\(\forall, \exists\)) = Relational Algebra
- Relational algebra is a retrieval query language based on set operators and relational operators

Operations in the Relational Model
- Theory behind operations is formally defined and equivalent to a first-order logic (FOL)
- Relational operators transform either a simple relation or a pair of relations into a result that is a relation
- The result can be used as an operand on later activities
- For every operand and result, relations are free of duplicates
- Operators are tuple oriented or set oriented

Query Operators
- **Relational Algebra**
  - tuple (unary) Selection, Projection
  - set (binary) Union, Intersection, Difference
- tuple (binary) Join, Division
- **Additional Operators**
  - Outer Join, Outer Union
A Retrieval DML Must Express

- Attributes required in a result
- Target list
- Criteria for selecting tuples for that result
- Qualifier
- The relations that take part in the query
  - Set generators
- Independent of the instances in the database
- Expressions are in terms of the database schema

Relational Algebra

SQL Retrieval Statement

SELECT [all|distinct]
(*|{table.*|expr[alias]|view.*})
[, (table.*|expr[alias])...]
FROM table {table[alias]|,table[alias]}...
WHERE condition
[CONNECT BY condition]
[START WITH condition]
[GROUP BY expr [,expr] ...]
[HAVING condition]
[UNION|UNION ALL|INTERSECT|MINUS]
SELECT ...
[ORDER BY (expr|position)
[ASC|DESC],[expr|position][ASC|DESC].
[FOR UPDATE OF column [,column]... [NOWAIT]]

π Project Operator

\[ \pi \text{attribute list}(\text{relation name}) \]

selects a subset of the attributes of a relation

Result = \( \pi \text{attribute list}(\text{relation name}) \)

- attribute list are drawn from the specified relation;
- if the key attribute is in the list then \( \text{card(result)} = \text{card(relation)} \)

π Project Operator

\( \pi \text{tutor}(\text{STUDENT}) \)

\begin{array}{|c|c|c|c|c|}
\hline
\text{STUDENT} & \text{studno} & \text{name} & \text{hons} & \text{tutor} & \text{year} \\
\hline
s1 & jones & ca & bush & 2 \\
\hline
s2 & brown & cis & kahn & 2 \\
\hline
s3 & smith & cs & goble & 2 \\
\hline
s4 & bloggs & ca & goble & 1 \\
\hline
s5 & jones & cs & goble & 1 \\
\hline
s6 & peters & ca & kahn & 3 \\
\hline
\end{array}

π Project Operator SELECT

\begin{array}{|c|c|c|c|c|}
\hline
\text{STUDENT} & \text{name} & \text{hons} & \text{tutor} & \text{year} \\
\hline
s1 & jones & ca & bush & 2 \\
\hline
s2 & brown & cis & kahn & 2 \\
\hline
s3 & smith & cs & goble & 2 \\
\hline
s4 & bloggs & ca & goble & 1 \\
\hline
s5 & jones & cs & goble & 1 \\
\hline
s6 & peters & ca & kahn & 3 \\
\hline
\end{array}

\begin{array}{|c|c|c|c|}
\hline
\text{tutor} & \text{bush} & \text{kahn} & \text{goble} & \text{zobel} \\
\hline
\end{array}

\begin{array}{|c|c|c|c|}
\hline
\text{tutor} & \text{bush} & \text{kahn} & \text{goble} & \text{zobel} \\
\hline
\end{array}

\begin{array}{|c|c|c|c|}
\hline
\text{tutor} & \text{bush} & \text{kahn} & \text{goble} & \text{zobel} \\
\hline
\end{array}

\begin{array}{|c|c|c|c|}
\hline
\text{tutor} & \text{bush} & \text{kahn} & \text{goble} & \text{zobel} \\
\hline
\end{array}

π Project Operator SELECT

select * from student;

select tutor from student;
**σ Select Operator**

selects a subset of the tuples in a relation that satisfy a selection condition

\[ \text{Result} = \sigma_{(\text{selection condition})} (\text{relation name}) \]

- a boolean expression specified on the attributes of a specified relation
- a relation that has the same attributes as the source relation
- stands for the usual comparison operators 'V', '<', '<=', '>', '>=', etc
- clauses can be arbitrarily connected with boolean operators AND, NOT, OR

\[ \text{degree(result)} = \text{degree(relation)}; \]
\[ \text{card(result)} \leq \text{card(relation)} \]

---

**SQL retrieval expressions**

- select studentno, name from student
  where hons != 'ca' and
  (tutor = 'goble' or tutor = 'kahn');
- select * from enrol
  where labmark > 50;
- select * from enrol
  where labmark between 30 and 50;
- select * from enrol
  where labmark in (0, 100);
- select * from enrol
  where labmark is null;
- select * from student
  where name is like 'b%';
- select studno, courseno, exammark + labmark total from enrol
  where labmark is not NULL;

---

**Cartesian Product Operator**

**Definition:**

- The cartesian product of two relations \( R_1(A_1, A_2, \ldots, A_n) \) with cardinality \( i \) and \( R_2(B_1, B_2, \ldots, B_m) \) with cardinality \( j \) is a relation \( R_3 \) with degree \( k=n+m \), cardinality \( i'j \) and attributes \( (A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m) \)
- The result, denoted by \( R_1 \times R_2 \), is a relation that includes all the possible combinations of tuples from \( R_1 \) and \( R_2 \)
- Used in conjunction with other operations

---

**Retrieve tutor who tutors Bloggs**

\[ \pi_{tutor}(\sigma_{\text{name}='bloggs'}(\text{STUDENT})) \]

\[ \text{select tutor from student} \]
\[ \text{where name = 'bloggs';} \]
Cartesian Product Example

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>COURSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>studno</td>
<td>name</td>
</tr>
<tr>
<td>s1</td>
<td>jones</td>
</tr>
<tr>
<td>s2</td>
<td>brown</td>
</tr>
<tr>
<td>s3</td>
<td>smith</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STUDENT X COURSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>studno</td>
</tr>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s2</td>
</tr>
<tr>
<td>s3</td>
</tr>
</tbody>
</table>

X Cartesian Product

<table>
<thead>
<tr>
<th>STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>studno</td>
</tr>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s2</td>
</tr>
<tr>
<td>s3</td>
</tr>
<tr>
<td>s4</td>
</tr>
<tr>
<td>s5</td>
</tr>
<tr>
<td>s6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STAFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>lecturer</td>
</tr>
<tr>
<td>kahn</td>
</tr>
<tr>
<td>bush</td>
</tr>
<tr>
<td>goble</td>
</tr>
<tr>
<td>zobel</td>
</tr>
<tr>
<td>watson</td>
</tr>
<tr>
<td>woods</td>
</tr>
<tr>
<td>capon</td>
</tr>
<tr>
<td>lindsey</td>
</tr>
<tr>
<td>barringer</td>
</tr>
</tbody>
</table>

θ Join Operator

Definition: The join of two relations R1(A1, A2, ..., An) and R2(B1, B2, ..., Bm) is a relation R3 with degree k = n + m and attributes (A1, A2, ..., An, B1, B2, ..., Bm) that satisfy the join condition:

```
Result = R1 \times (\theta \text{ join condition}) R2
```

The result is a concatenated set but only for those tuples where the condition is true. It does not require union compatibility of R1 and R2.

More joins

<table>
<thead>
<tr>
<th>STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>studno</td>
</tr>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s2</td>
</tr>
<tr>
<td>s3</td>
</tr>
<tr>
<td>s4</td>
</tr>
<tr>
<td>s5</td>
</tr>
<tr>
<td>s6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ENROL</th>
</tr>
</thead>
<tbody>
<tr>
<td>studno</td>
</tr>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s2</td>
</tr>
<tr>
<td>s3</td>
</tr>
<tr>
<td>s4</td>
</tr>
<tr>
<td>s5</td>
</tr>
</tbody>
</table>

Natural Join Operator

- Of all the types of θ-join, the equi-join is the only one that yields a result in which the compared columns are redundant to each other—possibly different names but same values.
- The natural join is an equi-join but one of the redundant columns (simple or composite) is omitted from the result.
- Relational join is the principle algebraic counterpart of queries that involve the existential quantifier ∃.
Self Join: Joins on the same relation

\[ \pi_{(\text{lecturer}, \text{roomno}, \text{appraiser}, \text{apprroom})}(\text{staff}) \times \pi_{(\text{staff})}(\text{staff}) \]

Exercise

Get student's name, all their courses, subject of course, labmark for course, lecturer of course and lecturer's roomno for 'ca' students

University Schema

- STUDENT(studno, name, hons, tutor, year)
- ENROL(studno, courseno, labmark, exammark)
- COURSE(courseno, subject, equip)
- STAFF(lecturer, roomno, appraiser)
- TEACH(courseno, lecturer)
- YEAR(yearno, yeartutor)

Set Theoretic Operators

- **Union**, Intersection and Difference
- operands need to be union compatible for the result to be a valid relation

Definition:
Two relations \( R_1(A_1, A_2, ..., A_n) \) and \( R_2(B_1, B_2, ..., B_m) \) are union compatible iff:
- \( n = m \)
- \( \text{dom}(A_i) = \text{dom}(B_i) \) for \( 1 \leq i \leq n \)

**Union Operator**

Definition:
The union of two relations \( R_1(A_1, A_2, ..., A_n) \) and \( R_2(B_1, B_2, ..., B_m) \) is a relation \( R_3(C_1, C_2, ..., C_n) \) such that
- \( \text{dom}(C_i) = \text{dom}(A_i) = \text{dom}(B_i) \) for \( 1 \leq i \leq n \)
- The result \( R_1 \cup R_2 \) is a relation that includes all tuples that are either in \( R_1 \) or \( R_2 \) or in both without duplicate tuples
- The resulting relation might have the same attribute names as the first or the second relation

**Intersection Operator**

Definition:
The intersection of two relations \( R_1(A_1, A_2, ..., A_n) \) and \( R_2(B_1, B_2, ..., B_m) \) is a relation \( R_3(C_1, C_2, ..., C_n) \) such that
- \( \text{dom}(C_i) = \text{dom}(A_i) = \text{dom}(B_i) \) for \( 1 \leq i \leq n \)
- The result \( R_1 \cap R_2 \) is a relation that includes only those tuples in \( R_1 \) that also appear in \( R_2 \)
- The resulting relation might have the same attribute names as the first or the second relation

Retrieve all staff that lecture or tutor

Lecturers \( \pi_{\text{lecturer}} \text{TEACH} \)
Tutors \( \pi_{\text{tutor}} \text{STUDENT} \)
Lecturers \( \cup \) Tutors
Retrieve all staff that lecture and tutor

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>TEACH</th>
</tr>
</thead>
<tbody>
<tr>
<td>studno</td>
<td>name</td>
</tr>
<tr>
<td>s1</td>
<td>jones</td>
</tr>
<tr>
<td>s2</td>
<td>brown</td>
</tr>
<tr>
<td>s3</td>
<td>smith</td>
</tr>
<tr>
<td>s4</td>
<td>bloggs</td>
</tr>
<tr>
<td>s5</td>
<td>jones</td>
</tr>
<tr>
<td>s6</td>
<td>peters</td>
</tr>
</tbody>
</table>

Lecturers \( \pi_{\text{lecturer}} \) \text{TEACH} \\
Tutors \( \pi_{\text{tutor}} \) \text{STUDENT} \\
Lecturers \cap Tutors

− Difference Operator

Definition:
The difference of two relations \( R_1(A_1, A_2, \ldots, A_n) \) and \( R_2(B_1, B_2, \ldots, B_m) \) is a relation \( R_3(C_1, C_2, \ldots, C_p) \) such that

\[
\text{dom}(C_i) = \text{dom}(A_i) - \text{dom}(B_i) \quad \text{for} \quad 1 \leq i \leq n
\]

- The result \( R_1 - R_2 \) is a relation that includes all tuples that are in \( R_1 \) and not in \( R_2 \)
- The resulting relation might have the same attribute names as the first or the second relation

Retrieve all staff that lecture but don’t tutor

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>TEACH</th>
</tr>
</thead>
<tbody>
<tr>
<td>studno</td>
<td>name</td>
</tr>
<tr>
<td>s1</td>
<td>jones</td>
</tr>
<tr>
<td>s2</td>
<td>brown</td>
</tr>
<tr>
<td>s3</td>
<td>smith</td>
</tr>
<tr>
<td>s4</td>
<td>bloggs</td>
</tr>
<tr>
<td>s5</td>
<td>jones</td>
</tr>
<tr>
<td>s6</td>
<td>peters</td>
</tr>
</tbody>
</table>

Lecturers \( \pi_{\text{lecturer}} \) \text{TEACH} \\
Tutors \( \pi_{\text{tutor}} \) \text{STUDENT} \\
Lecturers - Tutors

Outer Join Operation

- In an equi-join, tuples without a ‘match’ are eliminated
- Outer join keeps all tuples in \( R_1 \) or \( R_2 \) or both in the result, padding with nulls
  - Left outer join \( R_1 \) \( R_2 \) 
    - keeps every tuple in \( R_1 \)
  - Right outer join \( R_1 \) \( R_2 \) 
    - keeps every tuple in \( R_2 \)
  - Select * from \( R_1 \), \( R_2 \) where \( R_1.a = R_2.a (+) \)
- Double outer join \( R_1 \) \( R_2 \) 
  - keeps every tuple in \( R_1 \) and \( R_2 \)
  - Select * from \( R_1 \), \( R_2 \) where \( R_1.a (+) = R_2.a (+) \)

Outer Join Operator

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>STAFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>studno</td>
<td>name</td>
</tr>
<tr>
<td>s1</td>
<td>jones</td>
</tr>
<tr>
<td>s2</td>
<td>brown</td>
</tr>
<tr>
<td>s3</td>
<td>smith</td>
</tr>
<tr>
<td>s4</td>
<td>bloggs</td>
</tr>
<tr>
<td>s5</td>
<td>jones</td>
</tr>
<tr>
<td>s6</td>
<td>peters</td>
</tr>
</tbody>
</table>

select * from student, staff 
where tutor = lecturer

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>STAFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>studno</td>
<td>name</td>
</tr>
<tr>
<td>s1</td>
<td>jones</td>
</tr>
<tr>
<td>s2</td>
<td>brown</td>
</tr>
<tr>
<td>s3</td>
<td>smith</td>
</tr>
<tr>
<td>s4</td>
<td>bloggs</td>
</tr>
<tr>
<td>s5</td>
<td>jones</td>
</tr>
<tr>
<td>s6</td>
<td>peters</td>
</tr>
</tbody>
</table>

select * from student, staff 
where tutor = lecturer (+)
Outer Self Join

\[ \pi \left( \text{lecturer}, (\text{staff} \quad \text{appraiser = lecturer}) \quad \text{staff} \right) \]

\[
\begin{array}{llll}
\text{STAFF} & \text{roomno} & \text{appraiser} & \text{approom} \\
\hline
\text{kahn} & \text{IT206} & \text{watson} & \text{IT212} \\
\text{bush} & \text{2.26} & \text{capon} & \text{null} \\
\text{goble} & \text{2.82} & \text{capon} & \text{null} \\
\text{zobel} & \text{3.34} & \text{watson} & \text{IT212} \\
\text{watson} & \text{IT212} & \text{barringer} & \text{2.125} \\
\text{woods} & \text{IT204} & \text{barringer} & \text{2.125} \\
\text{capon} & \text{A14} & \text{watson} & \text{IT212} \\
\text{lindsey} & \text{2.10} & \text{woods} & \text{IT204} \\
\text{barringer} & \text{2.125} & \text{null} & \text{null} \\
\end{array}
\]

- Takes the union of tuples from two relations that are not union compatible
- The two relations, R1 and R2, are partially compatible—only some of their attributes are union compatible
- The attributes that are not union compatible from either relation are kept in the result and tuples without values for these attributes are padded with nulls

Ordering results

```
select * from enrol, student
where labmark is not null and
student.studno = enrol.studno
order by
    hons, courseno, name
default is ascending
```

Completeness of Relational Algebra

- Five fundamental operations
- \( \sigma \quad \pi \quad \cup \quad - \)
- Additional operators are defined as combination of two or more of the basic operations.
- e.g. \( R_1 \cap R_2 = R_1 \cup (R_1 - R_2) \)

\( R_1 \cup_{\text{condition}} R_1 = \sigma_{\text{condition}}(R_1 \times R_2) \)

\( \div \) Division Operation

Definition:
The division of two relations \( R_1(A_1, A_2, ..., A_n) \) with cardinality \( i \) and \( R_2(B_1, B_2, ..., B_m) \) with cardinality \( j \) is a relation \( R_3 \) with degree \( k = n-m \), cardinality \( ij \) and attributes \( (A_1, A_2, ..., A_i, B_1, B_2, ..., B_m) \) that satisfy the division condition
- The principle algebraic counterpart of queries that involve the universal quantifier \( \forall \)
- Relational languages do not express relational division

\( \div \) Division Operation Example

Retrieve the studnos of students who are enrolled on all the courses that Capon lectures on
- Small_ENROL \( \div \) Capon_TEACH

```
\begin{tabular}{|c|c|}
\hline
\text{Small_ENROL} & \text{Capon_TEACH} \\
\hline
\text{studno} & \text{mark} \\
\text{s1} & \text{cs250} \\
\text{s1} & \text{cs260} \\
\text{s2} & \text{cs250} \\
\text{s2} & \text{cs270} \\
\text{s3} & \text{cs270} \\
\text{s4} & \text{cs280} \\
\text{s4} & \text{cs250} \\
\text{s6} & \text{cs250} \\
\hline
\end{tabular}
```
Aggregation Functions

- Aggregation functions on collections of data values: average, minimum, maximum, sum, count
- Group tuples by value of an attribute and apply aggregate function independently to each group of tuples

Example:

<table>
<thead>
<tr>
<th>studno</th>
<th>course</th>
<th>labmark</th>
<th>exammark</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>cs250</td>
<td>65</td>
<td>52</td>
</tr>
<tr>
<td>s1</td>
<td>cs260</td>
<td>80</td>
<td>75</td>
</tr>
<tr>
<td>s1</td>
<td>cs270</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td>s2</td>
<td>cs250</td>
<td>87</td>
<td>55</td>
</tr>
<tr>
<td>s3</td>
<td>cs270</td>
<td>85</td>
<td>71</td>
</tr>
<tr>
<td>s4</td>
<td>cs270</td>
<td>90</td>
<td>80</td>
</tr>
<tr>
<td>s5</td>
<td>cs280</td>
<td>50</td>
<td>51</td>
</tr>
<tr>
<td>s6</td>
<td>cs250</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

 Aggregation Functions in SQL

```sql
select studno, count(*),
    avg(labmark)
from enrol
group by studno
having count(*) >= 2
```

Nested Subqueries

- Complete select queries within a WHERE clause of another outer query
- Creates an intermediate result
- No limit to the number of levels of nesting

List all students with the same tutor as bloggs

```sql
select studno, name, tutor
from student
where tutor = (select tutor
               from student
               where name = 'bloggs')
```

Union compatibility in nested subqueries

```sql
select distinct studno
from enrol
where (courseno, exammark) in
    (select courseno, exammark
     from student s, enrol e
     where s.name = 'bloggs' and e.studno = s.studentno)
```
Nested subqueries set comparison operators
- Outer query qualifier includes a value $v$ compared with a bag of values $V$ generated from a subquery.
- Comparison $v$ with $V$ evaluates TRUE:
  - $v \in V$ if $v$ is one of the elements of $V$.
  - $v = \text{any} \ V$ if $v$ is equal to some value in $V$.
  - $v > \text{any} \ V$ if $v >$ some value in $V$ (same for $<$).
  - $v > \text{all} \ V$ if $v >$ greater than all the values in $V$ (same for $<$).

Subqueries
- May be used in these situations:
  - to define the set of rows to be inserted in the target table of an insert, create table or copy command.
  - to define one or more values to be assigned to existing rows in an update statement.
  - to provide values for comparison in where, having and start with clauses in select, update and delete commands.

Correlated subqueries
- A condition in the where clause of a nested query references some attribute of a relation declared in the outer (parent) query.
- The nested query is evaluated once for each tuple in the outer (parent) query.
  ```sql
  select name
  from student
  where $3 > (select count(*)
  from enrol
  where student.studno = enrol.studno)
  ```

Exists and correlated subqueries
- Exists is usually used to check whether the result of a correlated nested query is empty.
- Exists ($Q$) returns TRUE if there is at least one tuple in the results query $Q$ and FALSE otherwise.
  ```sql
  select name
  from student
  where not exists (select *
  from enrol,
  teach
  where student.studno = enrol.studno
  and enrol.courseno = teach.courseno
  and teach.lecturer = 'Capon')
  ```

Conclusions
- The only logical structure is that of a relation.
- Constraints are formally defined on the concept of domain and key.
- Operations deal with entire relations rather than single record at a time.
- Operations are formally defined and can be combined in a declarative language to implement user queries on the database.
Conclusions on SQL

- Retrieval: many ways to achieve the same result—though different performance costs
- Comprehensive and powerful facilities
- Non-procedural
- Can't have recursive queries
- Limitations are overcome by use of a high-level procedural language that permits embedded SQL statements