Topic 3:
Information Retrieval

Querying Relational Database

SQL

- SQL -- (Structured Query Language)
- History
  - Developed by IBM (system R) in the 70's
  - SQL standards
    - SQL-86
    - SQL-89
    - SQL-92
    - SQL-99 (current standard)
Advantages of SQL

- Set-at-a-time
- Declarative language
- Usage
  - stand-alone
  - imbedded
- Similar to nature language

Selection

- Form queries for extracting information out of the database.
- The result of a selection statement is a relation (table).
- It is possible that the result relation is empty.
- The result is a snapshot of the database, at the moment when the selection is executed.
Selection Syntax

Basic components:
- SELECT clause
- FROM clause
- WHERE clause
- ORDER BY clause
- GROUP BY clause

Get what
The data source
The criteria
How the info is ordered
How the info is organized

DB Schema

Student (sid, name, dept, age, GPA)
Course (cid, name, description)
Enrollment (sid, cid, term, midterm, final, grade)
SELECT Clause

- The target of the query.
- Specify what to get from the database.
- Syntax:
  - *
  - Column list
  - Math expression
  - Aggregate functions

FROM clause

- Specify the data source of the query.
- May be one or more relation(s).
- May be intermediate result.
Simple cases

SELECT *
FROM Student

SELECT name, age
FROM Student

SELECT Student.name, Student.age
FROM Student

SELECT S.name, S.age
FROM Student AS S

---

How to write a query?

- Step 1:
  □ Identify which table to query
  □ Put the table name in the FROM clause

- Step 2:
  □ Identify the information to return
  □ List the column names in the SELECT clause.
WHERE clause

- The condition to be satisfied
- Simple condition
  - Comparison operators: >, <, >=, <=, =, <>
  - Logical connectives: AND, OR, NOT

How to write a query?

- Step 1:
  - Identify which table to query
  - Put the table name in the FROM clause
- Step 2:
  - Identify the information to return
  - List the column names in the SELECT clause.
- Step 3:
  - Identify the predicates to be satisfied.
  - Write each predicates
  - Link the predicates using proper logical connectives to form the WHERE clause
Example

- Query: Find the name and GPA of Informatics students.

```
SELECT name, GPA
FROM student
WHERE dept = "Info"
```

Example

- Query: Find the name of the students who are older than 20.

```
SELECT name
FROM student
WHERE age > 20
```
### DISTINCT

```
SELECT DISTINCT col1, col2,...
FROM ......
WHERE ......
```

- Eliminate duplicated rows from the result.

### Exercise

- Query: Find the departments where the younger students (<17 years old) are in.

```
SELECT DISTINCT dept
FROM student
WHERE age < 17
```
ORDER BY Clause

ORDER BY expression [ ASC | DESC ],
expression [ ASC | DESC ],
......

- Specify the order of the query result
- Ordering (ascending/descending) can be specified on each expression.

How to write a query?

- Step 1:
  - Identify which table to query
  - Put the table name in the FROM clause
- Step 2:
  - Identify the information to return
  - List the column names in the SELECT clause.
- Step 3:
  - Identify the predicates to be satisfied.
  - Write each predicates
  - Link the predicates using proper logical connectives to form the WHERE clause
- Step 4
  - Identify the order in which the results are to be returned
  - Write the ORDER BY clause.
Query with Order BY

- Query: find the info of the Informatics students, ordered by their age.

```
SELECT *
FROM Student
WHERE dept = 'Info'
ORDER BY age;
```

FROM clause

- Specify the data source of the query.
- May be one or more relation(s).
- May be intermediate result.
Query in more than one relations

- Find the name of the students who take I308

```
SELECT student.name
FROM student, enrollment
WHERE student.sid = enrollment.sid
AND enrollment.cid = 'I308'
```

<table>
<thead>
<tr>
<th>Sid</th>
<th>Name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>012</td>
<td>Tom</td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>Mary</td>
<td></td>
</tr>
<tr>
<td>022</td>
<td>Jackie</td>
<td></td>
</tr>
<tr>
<td>298</td>
<td>Bob</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sid</th>
<th>Cid</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>012</td>
<td>I308</td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>I201</td>
<td></td>
</tr>
<tr>
<td>012</td>
<td>I450</td>
<td></td>
</tr>
<tr>
<td>012</td>
<td>I101</td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>I301</td>
<td></td>
</tr>
<tr>
<td>022</td>
<td>I308</td>
<td></td>
</tr>
<tr>
<td>022</td>
<td>I501</td>
<td></td>
</tr>
<tr>
<td>298</td>
<td>I308</td>
<td></td>
</tr>
</tbody>
</table>
How to write a query?

- **Step 1:**
  - Identify the tables to query
  - Put the table name in the FROM clause

- **Step 2:**
  - Identify the information to return
  - List the column names in the SELECT clause.

- **Step 3:**
  - Put down the join conditions in the WHERE clause.
  - Identify the predicates to be satisfied.
  - Write each predicate
  - Link the predicates using proper logical connectives to as the rest of the WHERE clause.

- **Step 4**
  - Identify the order in which the results are to be returned
  - Write the ORDER BY clause.

---

**Like Condition**

- A special value comparison operator for string matching
- Find string that matches the given pattern
  - `%` matches zero to more characters
  - `_` matches any single character
  - `\` escape

```
column_name LIKE pattern
```
Query with Like

- Query: find the info of students whose first name is Kevin.

```sql
SELECT *
FROM student
WHERE dept = 'Info'
AND name LIKE 'Kevin%';
```

Between Condition

- A special value comparison operator
- Satisfied when the value of the columns is between the two border values given in the query.

```
column_name BETWEEN val1 AND val2
```
Query with Between

- Query: find the number of students in Info who are between 25 and 30.

```
SELECT count(sid)
FROM student
WHERE dept = 'Info'
AND age BETWEEN 25 AND 30;
```

Aggregate Functions

- Summarize the query results
- Aggregate functions:
  - `count()`
  - `sum()`
  - `max()`
  - `min()`
  - `avg()`
Query with Aggregation Functions

- Query: find the number of Info students.

```sql
SELECT student.sid
FROM student
WHERE student.dept = 'Info'
```

```sql
SELECT count(student.sid)
FROM student
WHERE student.dept = 'Info'
```

How to write a query?

- **Step 1:**
  - Identify the tables to query
  - Put the table name in the FROM clause

- **Step 2:**
  - Identify the information to return
  - List the column names or aggregate functions in the SELECT clause.

- **Step 3:**
  - Put down the join conditions in the WHERE clause.
  - Identify the predicates to be satisfied.
  - Write each predicate
  - Link the predicates using proper logical connectives to as the rest of the WHERE clause

- **Step 4:**
  - Identify the order in which the results are to be returned
  - Write the ORDER BY clause.
**GROUP BY clause**

- The GROUP BY clause can be used in a SELECT statement to collect data across multiple records and group the results by one or more columns.

```
GROUP BY column1, column2, ... column_n;
```

- When using the GROUP BY clause, only the group-by columns and aggregate function can appear in the SELECT clause.

---

**Query with Group BY**

- Query: find the average final grade for each class offered by the university

```
SELECT cid, avg(final) FROM Enrollment GROUP BY cid;
```
How to write a query?

- **Step 1:**
  - Identify the tables to query
  - Put the table name in the FROM clause
- **Step 2:**
  - Identify the information to return
  - List the column names or aggregate functions in the SELECT clause.
  - If the results are to be returned per group, add GROUP BY clause with proper group by columns, add group identifier (columns) in the SELECT clause.
- **Step 3:**
  - Put down the join conditions in the WHERE clause.
  - Identify the predicates to be satisfied.
  - Write each predicates
  - Link the predicates using proper logical connectives to as the rest of the WHERE clause
- **Step 4:**
  - Identify the order in which the results are to be returned
  - Write the ORDER BY clause.

```
SELECT cid, avg(final)
FROM Enrollment
GROUP BY cid;
```
Having Clause

\[ \text{HAVING condition1} \ldots \text{condition}_n; \]

- used in combination with the \texttt{GROUP BY} clause.
- used in a \texttt{SELECT} statement to filter the records that a \texttt{GROUP BY} returns.

Query with Having

- Query: find the courses that have at least 3 students registered

```
SELECT cid
FROM Enrollment
GROUP BY cid
HAVING count(sid) >= 3;
```
**How to write a query?**

- **Step 1:**
  - Identify the tables to query
  - Put the table name in the **FROM** clause

- **Step 2:**
  - Identify the information to return
  - List the column names or aggregate functions in the **SELECT** clause.
  - If the results are to be returned per group, add **GROUP BY** clause with proper group by columns.

- **Step 3:**
  - Put down the join conditions in the **WHERE** clause.
  - Identify the predicates to be satisfied.
  - Write each predicates
  - Link the predicates using proper logical connectives to as the rest of the **WHERE** clause

- **Step 4:**
  - Find out what are the predicates each group has to satisfy
  - Put these predicates in the **HAVING** clause.

- **Step 5:**
  - Identify the order in which the results are to be returned
  - Write the **ORDER BY** clause.
Example

- Query: find the courses that have at least 3 students registered

```sql
SELECT cid
FROM Enrollment
GROUP BY cid
HAVING count(sid) >= 3;
```

Proofread your SQL statement

- Whether the clauses work together
  - If there is a HAVING clause, there must be a GROUP BY clause.
- Have you included all the predicates
  - If there are more than one table in the FROM clause, there must be a join condition to link every pair of tables.
  - Check query requirements.
- Does the query return the required information.
  - DISTINCT
    - Return the group identifier when there is GROUP BY clause.
- Group predicate vs. tuple predicate
  - Should the predicate be in the WHERE clause or HAVING clause.
- Additional checking for HAVING clause.
  - The columns to be returned (in the SELECT clause) are either the group by columns, or aggregate functions.
SQL Queries with Set Operations

- Set operations:
  - UNION
  - INTERSECT
  - EXCEPT
- Must be used between tables of the same schema

Set operation

- UNION
- INTERSECT
- EXCEPT
Example

- Query: Find the name of all students who are in Informatics or Business.

```
SELECT name
FROM student
WHERE dept = 'Info'
UNION
SELECT name
FROM student
WHERE dept = 'Business'
```

Nested Queries

- Nested query: a query that has another query embedded within it.
- The embedded query is called sub-query.
- Sub-query may appear in FROM clause, WHERE clause and HAVING clause.
- Linking main query and sub-queries
  - (NOT) IN sub-query
  - (NOT) EXIST sub-query
  - Comp-op ANY/ALL sub-query
Example

- File the name of the students who have taken I308

```sql
SELECT name
FROM student S
WHERE EXISTS
  (SELECT *
   FROM enrollment E
   WHERE E.sid = S.sid
   AND E.cid = 'I308')
```

Example

- File the name of the students who have taken I308.

```sql
SELECT name
FROM student S
WHERE S.sid IN
  (SELECT E.sid
   FROM enrollment E
   WHERE E.cid = 'I308')
```
Example

- File the name of the students who got the highest score in I308

```sql
SELECT name
FROM student S, enrollment E1
WHERE S.sid = E1.sid
AND E1.cid = 'I308'
AND E1.score >= ANY
    (SELECT E2.score
     FROM enrollment E2
     WHERE E2.cid = 'I308')
```