Structured Query Language (SQL)
DML

SWEN 304

Trimester 2, 2017

Lecturer: Dr Hui Ma

Engineering and Computer Science

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Outline

- Modifying databases: `INSERT`, `DELETE`, and `UPDATE`
- SQL as Language
- Single table queries
- Multiple table queries
- Nested queries
- Aggregate functions
- Reading:
- Chapter 8 of the textbook
- PostgreSQL Manual

UNIVERSITY Database

UNIVERSITY ={STUDENT(StudId, Lname, Fname, Major),
COURSE(CourId, Cname, Points, Dept),
GRADES(StudId, CourId, Grade)}

IC = {GRADES[Id] $\subseteq$ STUDENT[Id],
GRADES[Course_id] $\subseteq$ COURSE[Course_id]}

STUDENT			
StudId	**Lname**	**Fname**	**Major**
300111	Smith	Susan	COMP
300121	Bond	James	MATH
300143	Bond	Jenny	MATH
300132	Smith	Susan	COMP

COURSE			
CourId	**Cname**	**Points**	**Dept**
COMP302	DB sys	15	Engineering
COMP301	softEng	20	Engineering
COMP201	Pr & Sys	22	Engineering
MATH214	DisMat	15	Mathematics

GRADES		
StudId	**CourId**	**Grade**
300111	COMP302	A+
300111	COMP301	A
300111	MATH214	A
300121	COMP301	B
300132	COMP301	C
300121	COMP302	B+
300143	COMP201	ω
300132	COMP201	ω
300132	COMP302	C+

University Database Schema: STUDENT

STUDENT(StudId, Lname, Fname, Major),

```
CREATE TABLE STUDENT (
      StudId   INT
                  NOT NULL
                  DEFAULT 30000
      CONSTRAINT stpk PRIMARY KEY
                  CONSTRAINT StIdRange CHECK
                              (StudId  BETWEEN 300000 AND
399999),
      LName   CHAR(15) NOT NULL,
      FName   CHAR(15) NOT NULL,
      Major   CHAR(25) DEFAULT 'Comp'
      );
```

University Database Schema: COURSE

COURSE(CourId, Cname, Points, Dept),

```
CREATE TABLE COURSE (
      CourId  CHAR(7)  CONSTRAINT cspk  PRIMARY KEY,
      CName  CHAR(15) NOT NULL,
      Points  INT NOT NULL CONSTRAINT pointschk
        CHECK (Points >= 0 AND Points <= 50),
      Dept  CHAR(25)
   );
```

University Database Schema: GRADES

GRADES(StudId, CourId, Grade)

GRADES[Id] $\subseteq$ STUDENT[Id], GRADES[Course_id] $\subseteq$ COURSE[Course_id]

```
CREATE TABLE GRADES (
    StudId  INT  NOT NULL
        CONSTRAINT Gstidrange  CHECK
                (StudId BETWEEN 300000 and 399999),
        CONSTRAINT gsri REFERENCES STUDENT
                ON DELETE CASCADE,
    CourId  CHAR(8) NOT NULL
        CONSTRAINT gpri  REFERENCES COURSE
                ON DELETE NO ACTION,
    Grade  CHAR(2)
        CONSTRAINT grd  CHECK
                (Grade IN ('A+', 'A', 'A-', 'B+', 'B', 'B-', 'C+', 'C', NULL)),
    CONSTRAINT gpk  PRIMARY KEY (StudId, CourId )
    );
```

Modify databases

- Three commands used to modify the database:
  - `INSERT`, `DELETE`, and `UPDATE`

INSERT Command

- Once a database schema has been created we can populate the database by using the INSERT command

- Specify the relation name and a list of values for the tuple

```
INSERT INTO ⟨table_name⟩ [ ⟨attribute_list⟩ ]
(VALUES ( ⟨value_list⟩ ) | SELECT ...)
```

INSERT Command

- Example:

```
INSERT INTO STUDENT
VALUES (111111, 'Bole', 'Ann', Math);
```

- *Note:* The values in VALUES have to appear in the same order as the attributes in the corresponding `CREATE TABLE` command,

```
INSERT INTO STUDENT (FName, LName, StudId )
VALUES('Ann', 'Bole', 111111);
```

- *Note:*
  - Useful when values of attributes declared as `NULL`, or having `DEFAULT` value are missing
  - Not allowed if the missing attribute is declared `NOT NULL`

INSERT Command (continued)

- A form of `INSERT` command that is suitable for creation of temporary tables

```
CREATE TABLE StudentInfo (
    StudId INT PRIMARY KEY,
    LName CHAR(15) NOT NULL,
    NoOfCourses INT);
```

```
INSERT INTO StudentInfo
    SELECT s.StudId, LName, COUNT(*) AS NoOfCourses
          FROM Student s,  Grades g
          WHERE s.StudId = g.StudId
          GROUP BY StudId, LName ;
```

UPDATE Command

- Modify attribute values of one or more selected tuples

```
UPDATE ⟨table_name⟩
SET ⟨attribute_name⟩ = ⟨value_expression⟩
{, ⟨attribute_name⟩ = ⟨value_expression⟩ }
[WHERE ⟨condition⟩ ]
```

- Example:

```
UPDATE GRADES
SET Grade = 'A+'
WHERE CourId = 'C302';
```

DELETE Command

- Removes tuples from a relation
  - Includes a WHERE clause to select the tuples to be deleted

```
DELETE FROM ⟨table_name⟩ [WHERE ⟨condition⟩ ]
```

- Examples:

```
DELETE FROM STUDENT
WHERE FName = 'Susan';
```

```
DELETE FROM STUDENT
WHERE StudId IN
   (SELECT s.StudId
     FROM STUDENT s, GRADES g
     WHERE s.StudId = g.StudId AND CourId = 'C302');
```

```
DELETE FROM STUDENT ;
```

DROP vs DELETE

- DELETE statement performs conditional based deletion, whereas DROP command deletes entire records in the table
- DELETE statement removes only the rows in the table and it preserves the table structure as same whereas DROP command removes all the data in the table and the table structure
- DELETE operation can be rolled back and it is not auto committed, while DROP operation cannot be rolled back in any way as it is an auto committed statement
- DROP is a DDL statement while DELETE is a DML statement

Queries in SQL

- SELECT is the basic SQL statement for retrieving data from a database

```
SELECT  [ DISTINCT ]  ⟨attribute_list⟩
FROM  ⟨table_list⟩
[ WHERE  ⟨condition⟩  ]
```

- ⟨attribute_list⟩ = attributes whose values will be retrieved by query
  - e.g. FName, LName, CName, Grade
  - use "*" to denote all table attributes

Queries in SQL

- ⟨table_list⟩ : refer to relations needed to process the query
- tables containing attributes from ⟨attribute_list⟩ must be included
    - e.g., STUDENT, CORUSE, GRADES

- ⟨condition⟩ is a Boolean expression defining
- the properties of the tuples to be retrieved,
    - e.g., StudId = 007007
- join conditions (optional clause)
    - e.g., STUDENT.StudId = GRADES.StudId

Conditional Expression (Reference)

- Conditional expression of the WHERE clause can be any plausible combination of the following:

```
[(A  θ a]      [A  θ  B ]
[A  IS [ NOT ]  NULL]
[A  [ NOT ]  BETWEEN a1  AND a2 ]
[A  [ NOT ]  LIKE  ⟨pattern⟩  ]
   (string matching )
[A  [ NOT ] SIMILAR TO  ⟨regular expression⟩  ]
[A  [ NOT ]  IN  ⟨value_list⟩ ]
[A  θ ANY   ⟨value_list⟩ ] [A θ SOME  ⟨value_list⟩ ]
[A  θ ALL   ⟨value_list⟩ ]
[(EXIST | NOT EXIST)  ⟨sub query⟩ ]
```

where $\theta \in \{ =, <, <=, >, >=, <> \}$, A and B attributes or function of attributes, $a_i \in dom(A), i = 1,...$

Queries in SQL

- SQL considers a relation/table to be a multiset (or bag) of tuples, not a set ⇒ allows duplicates!
- SQL relations can be constrained to be sets by specifying PRIMARY KEY or UNIQUE attributes

- 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

The University Database

- Suppose for each student only pass grades are recorded in the database

STUDENT

LName	FName	StudId	Major
Smith	Susan	131313	Comp
Bond	James	007007	Math
Smith	Susan	555555	Comp
Cecil	John	010101	Math

COURSE

PName	CourId	Points	Dept
DB Sys	C302	15	Comp
SofEng	C301	15	Comp
DisMat	M214	22	Math
Pr&Sys	C201	22	Comp

GRADES

StudId	CourId	Grade
007007	C302	A+
555555	C302	D
007007	C301	A
007007	M214	A+
131313	C201	B-
555555	C201	C
131313	C302	D
007007	C201	A
010101	C201	D

Single Table Queries

- Retrieve the first and last names of Comp students

```
SELECT FName, LName
FROM STUDENT
WHERE Major = 'Comp';
```

FName	LNam e
Susan	Smith
Susan	Smith

- Find all different grades

```
SELECT DISTINCT Grade
FROM GRADES ;
```

Grade
A+
A
B-
C

Substring Comparisons

- Can extract part of a string with the function

```
substring( ⟨string⟩ , ⟨start pos⟩ , ⟨length⟩ )
```

- Can match to SQL patterns:
  - ⟨string⟩ LIKE ⟨pattern⟩
  - '%' to replace an arbitrary number of characters, and
  - '_' to replace exactly one character
- e.g., Retrieve course names of all 300 level courses
  - have '3' as the second character in CourId

```
SELECT CName FROM COURSE
WHERE CourId LIKE '_3%';
```

or

```
SELECT CName FROM COURSE
WHERE substring(CourId, 2,1) = '3';
```

PName
DB Sys
SofEng

Arithmetic Operations, Sorting

- SQL provides capability to perform four basic arithmetic operations (+, -, *, /) that can be applied to numeric attributes and constants only

```
SELECT 2 + 2;
```

- Sorting of the query result tuples is done using

```
ORDER BY { ⟨attribute_name⟩ [(ASC|DESC)],...}
```

clause after the WHERE clause (ASC is default)

```
SELECT *
FROM GRADES
ORDER BY StudId ASC, CourId DESC;
```

Qualification and Aliasing

- Attributes in different relation schemas can have the same names. How do we prevent ambiguity?
- In the `SELECT` clause, we prefix attributes by table name: `SELECT STUDENT.StudId ...`
- To change name of an attribute in the result, alias the attribute name using `AS` :

```
SELECT CourId AS CourseId
```

- In the `FROM` clause, specify a tuple variable from the table: ...`FROM` COURSE c, GRADES g, STUDENT s
- In the `WHERE` clause, prefix an attribute by the tuple variable: `WHERE c.CourId = g.CourId`

Multiple Table Queries and Joins

- To retrieve data from more than one table, we need a new operation: `JOIN`
- There are different joins:
  - `INNER` (theta join, equi-join, natural join)
  - `OUTER` (left, right, full)
  - Most often, we use the equi-join
- Each join operation is based on concatenating those tuples from two relations, which have such join attribute values, which satisfy the join condition
  - An equi-join concatenates tuples with equal join attribute values
  - An equi-join is most frequently based on a (FK, PK) pair

A JOIN Example

```
SELECT * FROM r1, r2 WHERE r1.B = r2.B ;
```

Join Condition

r1

A	B
1	b1
2	b1
3	b2
4	b3
5	ω

r2

B	C
b1	c1
b2	c1
b3	c1
b4	c2

r1 EQUI-JOIN r2

A	B	B	C
1	b1	b1	c1
2	b1	b1	c1
3	b2	b2	c1
4	b3	b3	c1

- If there is no join condition what is the result?

Multiple Table Queries

- *Retrieve course names with grades, and the surname for student James*

```
SELECT c.CName, Grade, LName AS Surname,
FROM STUDENT s, GRADES g, COURSE c
WHERE FName = 'James' AND s.StudId = g.StudId
AND c.CourId = g.CourId ;
```

  - Conditional expression is blue,
  - Join condition is red

CName	Grade	Surname
DB Sys	A+	Bond
SofEng	A	Bond
DisMat	A+	Bond
Pr&Sys	A	Bond

Nested Queries

- Some queries require comparing a tuple to a collection of tuples (e.g., *students doing courses that have more than 100 students*)
- This task can be accomplished by embedding a SQL query into `WHERE` clause of another query
  - The embedded query is called nested query,
  - The query containing the nested query is called outer query
- The comparison is made by using `IN`, θ `ANY`, θ `SOME`, and θ `ALL` operators, where θ∈{ =, <, <=, >=, >, < > }
- Note: `IN` ⇔ `=ANY` and `IN` ⇔ `=SOME`

Example Nested Query

- *Retrieve first names of students that passed M214*

```
SELECT FName
FROM STUDENT s
WHERE s.StudId  IN
    (SELECT StudId  FROM GRADES
     WHERE CourId = 'M214' AND Grade IS NOT
     NULL);
```

FName
James

- A nested query defined by using IN (or =ANY) operator can be expressed as a single block query

```
SELECT FName
FROM STUDENT s, GRADES g
WHERE s.StudId = g.StudId  AND g.CourId = 'M214'
AND g.Grade IS NOT NULL;
```

Correlated Nested Queries

- Let the variable *s* contain the current tuple of the outer query
- If the nested query doesn't refer to *s* :
  - The nested query computes the same result for each tuple in *s*
  - The outer query and the nested query are said to be uncorrelated
- If a condition in the `WHERE` clause of the nested query refers to some attributes of a relation declared in the outer query, the two queries are said to be correlated
  - Have to compute the inner query for each tuple considered by the outer query
  - Correlated nested queries consume more computer time than uncorrelated ones

Correlated Nested Query

- *Retrieve id's and surnames of those students that passed at least one course*

```
SELECT s.StudId, FName
FROM Student s
WHERE s.StudId  IN
    (SELECT StudId  FROM GRADES
     WHERE s.StudId = StudId AND
                Grade IS NOT NULL);
```

- Evaluation of the query:
  - when s.Stud Id = 131313,
    ⇒ result of the nested query is StudId = {131313},
    ⇒ (131313, Susan) is in the final result
  - When s.Stud Id = 010101,
    ⇒ result of the nested query is StudId = { },
    ⇒ (010101, John) is NOT in the final result

Correlated Nested Query

- Again, the nested query can be expressed as a single block query:

```
SELECT DISTINCT s.StudId, s.LName
FROM STUDENT s, Grades g
WHERE s.StudId = g.StudId AND Grade IS NOT
NULL;
```

- Have to be careful of duplicates!
- This computes an Equi-Join of the relations

EXISTS and NOT EXISTS

- *Retrieve Id's and surnames of students who passed at least one course:*

```
SELECT s.StudId, s.LName FROM STUDENT s
WHERE EXISTS
     (SELECT *  FROM GRADES
      WHERE s.StudId = StudId AND Grade IS NOT NULL);
```

- *Retrieve Id's and surnames of students who didn't pass any course:*

```
SELECT s.StudId, s.LName FROM STUDENT s
WHERE NOT EXISTS
     (SELECT *  FROM GRADES
      WHERE s.StudId = StudId AND Grade IS NOT NULL );
```

Summary

- SQL as DML: INSERT, UPDATE and DELETE
- SQL as a query language
  - Basic Query structure
    - Queries against a single table
    - Queries against multiple tables
    - Substring comparisons
    - Arithmetic operations
    - Sorting
  - Nested queries (outer and inner-nested queries)
    - Correlated nested queries

Next lecture

- SQL advanced options:
  - Joined tables,
  - Aggregate functions
    - Grouping
    - Having
- SQL set operations