CS157A: Introduction to Database Management Systems

Chapter 6:

The Database Language SQL- Part I

Suneuy Kim

Introduction to SQL

- SQL stands for "Structured Query Language" and is pronounced "sequel".
- Supported by all major commercial relational databases.
- Standardized
  - ANSI SQL, SQL92 (SQL2), SQL-99 (SQL3), SQL:2003
- Very high-level language
  - Describes "what to do" rather than "how to do it"
  - Role of query optimizer becomes extremely important.
- SQL is case-insensitive

e.g.) Keywords FROM and from are the same.

But, strings 'From' and 'from' are different.

SQL

- Two aspects of SQL: DDL and DML
- Data Definition Language (DDL)
  - Create table, Drop table, Alter table
- Data Manipulation Language (DML)
  - select, insert, delete, update

SQL

- Relations in SQL
  - Tables – stored relations (CREATE TABLE)
  - Views – relations defined by a computation, not stored (CREATE VIEW … AS)
  - Temporary tables – created by the SQL processor, not stored

SQL Primitive Data Types

- CHAR(n): a fixed-length string of up to n characters, pads by trailing blanks
- VARCHAR(n): a string of up to n characters, uses an end marker or string length

- BIT(n): bit strings of length n
- BIT VARYING(n): bit strings of length up to n

- BOOLEAN – TRUE, FALSE and UNKNOWN

- INT(or INTEGER), SHORTINT
- FLOAT(or REAL), DOUBLE PRECISION, DECIMAL(m, d) (e.g., 0123.45 is a value for DECIMAL(6,2), NUMERIC(m,d)

- DATE, TIME, TIMESTAMP

MySQL Example

```
CREATE TABLE TEST
(name CHAR(10),
 bit  BIT(5)
);
INSERT  INTO TEST
VALUES('Smith', B'011');
```


	name	bit
▶	Smith	3

Date and Time

- Date constant: DATE '1948-05-03'
- Time constant
  - TIME '15:00:02.5' two and a half seconds past three o'clock
- Timestamp: TIMESTAMP '1948-05-14 12:00:00' // noon on May 14, 1948
- Compare with < or >

MySQL Example

```
CREATE TABLE TEST1
(day DATE,
 time TIME,
 tstamp TIMESTAMP
);
INSERT  INTO TEST1 VALUES
('2014-01-03',
 '15:00:02.5',
 '2014-01-03 15:10:02.5');
```

	day	time	tstamp
▶	2014-01-03	15:10:02	2014-01-03 15:10:02

Running Example

Book(title, author, # of copies left)

User (uID, uName, age, #of books loaned)

Loan (uID, title, date, overdue)

Book

title	author	#copies

User

uID	uName	age	#books

Loan

uID	title	date	overdue

Table Declaration

```
CREATE TABLE BOOK
(title VARCHAR(50),
 author VARCHAR(30),
 copies INT,
 PRIMARY KEY(title)
);
```

Declaring Keys

- Key: an attribute or a set of attributes that uniquely defines each tuple.
- Use PRIMARY KEY or UNIQUE

Primary key vs. Unique

- To identify a row of data in a table.
- Cannot be null
- There can be only one *primary key* per table

- To ensure that data is not duplicated in two rows in the database.
- One row in the database is allowed to have null for the value of the unique key constraint.
- There can be more than one *unique key* per table

Declaring Keys

```
CREATE TABLE USER
(uID INT PRIMARY KEY,
 uNAME VARCHAR(30),
 age INT,
 loaned INT
);
```

```
CREATE TABLE LOAN
(uID INT,
 title VARCHAR(50),
 loanDate DATE,
 overdue BOOLEAN,
 PRIMARY KEY(uID,
title,loanDate)
);
```

Primary key vs. Unique key

```
CREATE TABLE Student
( sID INT PRIMARY KEY,
  username VARCHAR(10) UNIQUE
  gpa DOUBLE,
  SS# INT UNIQUE
…
);
```


- sID is university registry number which may be auto-generated.
- username is for logging in and is set by the student.

Modifying Relation Schemas

- DROP TABLE R
- ALTER TABLE MovieStar ADD phone CHAR (16);
- ALTER TABLE MovieStar DROP birthday;

Select

select A1, A2, …, An ← result of query (3)

from R1, R2, …, Rm ← target relations (1)

where condition ← filter (2)

Is equivalent to

$\pi_{A1,A2,\ldots An}(\sigma_{condition}(R1 \times R2 \times \ldots \times Rm))$

Select: Example

[Q] The titles of movies made by MGM Studios that either were made after 1970 or were less than 90 minutes long

[SQL]

```
SELECT title
FROM Movies
WHERE (year > 1970 OR length < 90)
AND studioName = 'MGM';
```

String Comparison

- Ignores pads and compares actual strings

e.g.) Comparing CHAR(n) with VARCHAR(n)

- < or <= compares strings in lexicographical order
- Pattern matching using LIKE

LIKE

- s LIKE p or s NOT LIKE p where s is a string and p is a pattern with the optional use of % and _.
  - % can match any sequence of **0 or more** characters in s
  - _ matches any **one** character in s
- Example
  - title like 'Star_ _ _ _'
  - title LIKE '%''s%' ;

    Two consecutive apostrophes in a string represent a single apostrophe and do not end the string.
- s LIKE 'x%%x%' ESCAPE 'x'
  - x is an escape character. x% is a character %.
  - The middle % means any string.

LIKE

```
select  uID, title
from Loan
where title like '%amb%';

select  *
from Loan
where title like '%amb%';
```

NULL Values

- Meaning depends on context: unknown, inapplicable, withheld
- 2 important rules to operate on a NULL value
  - When we operate on a NULL and any value using an arithmetic operator (like + or x), the result is NULL
  - When we compare a NULL with any value using a comparison operator (like = or >), the result is UNKNOWN
- NOTE: NULL is a value that can appear in tuples, but it is not a constant. We cannot use NULL explicitly as an operand.

e.g.) NULL + 3, NULL = 3 (not allowed),

- To ask if x has the value NULL, x IS NULL or x IS NOT NULL.

Comparing NULL's to Values

- The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN.
- Comparing any value (including NULL itself) with NULL yields UNKNOWN.
- A tuple is included in a result iff the condition is TRUE (not FALSE or UNKNOWN).

The Truth Value UNKNOWN

x	y	x AND y	x OR y	NOT x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Pitfalls Regarding Nulls

Intuition can be violated for some cases

- If the domain of attribute x is an integer and its value is currently NULL, x * 0 or x – x returns NULL.
- What do you expect from this query ?

```
select uID, uName, age
from User
where age > 20 or age <= 20;
```

Note: Users with null age will not be included.

- Consider the following query. What is the output ? (Let's assume, null is in age, not in loaned.)


```
select uID, uName, age, loaned
from User
where age > 20 or loaned < 5;
```


- What if you change `or` to `and` ?

Single Relation Query

```
select uID, uName, age, loaned
from User
where age < 18;
```



```
select *
from User
where age < 18;
```

uID	uName	age	#books

t

- Implicit tuple variable, say t, iterates over all tuples.
- If the current tuple t satisfies the condition of the Where clause, include it in the result.

Query involving multiple relations

1. Start with the **product** of all the relations in the FROM clause.
2. Apply the selection condition from the WHERE clause.
3. Project onto the list of attributes and expressions in the SELECT clause.

Query involving two relations

```
select uName,  title
from User, Loan
where User.uID = Loan.uID  and  overdue = true;
```



```
select distinct uName,  title
from User, Loan
where User.uID = Loan.uID  and  overdue = true;
```

Disambiguating Attributes

```
select distinct title, author
from Book, Loan
where Book.title = Loan.title
    and copies < 2 and overdue = true;
```


→What is wrong with this query ?

→How would you fix it ?

Query combining three relations

```
select User.uID, uName, age, Loan.title,  copies
from User, Loan, Book
where  User.uID = Loan.uID
                    and Loan.title = Book.title;
```

order by

```
select User.uID, uName, age, Loan.title,  copies
from Book, User, Loan
where Loan.uID = User.uID and Loan.title = Book.title
order by age desc;
```


NOTE:

- The order by clause follows the where clause and any other clause.
- The ordering is performed after from and where clauses, just before the select clause is applied.

Tuple Variables

- select A1, A2, …, An

  from R1, R2, …, Rm ← implicit tuple variables

  where R1.A1 < R2.A1

- If a query combines relations with different names, `RelationName.attributeName` will disambiguate attributes.

Explicit Tuple-Variables

- Sometimes, a query needs to use two copies of the same relation.
- Use it in the from clause (AS can be omitted.)

  e.g.) from User AS U1, User AS U2

  → from User U1, User U2

- Tuple variables can be used to rename relations, even when not essential. → See the example on the next page.

Explicit Tuple Variables

```
select U.uID, uName, age, L.title, overdue
from User U, Book B, Loan L
where U.uID = L.uID and L.title = B.title;
```

Explicit Tuple Variables

[Q] Pairs of users with the same age

```
select U1.uID, U1.uName, U1.age,
       U2.uID, U2.uName, U2.age
from User U1, User U2
where U1.age = U2.age;
```


→Does it produce what we want ?

→What to add more ? U1.uID <> U2.uID ?

→What about U1.uID < U2.UID ? Why ?

Union

```
select author as name from Book
union
select uName as name from User;

select author as name from Book
union all  ← add duplicate
select uName as name from User;
```

Intersect

```
select uID from Loan where title ='Bambi'
intersect
select uID from Loan where title = 'Lion King';
```



```
select distinct L1.uID
from Loan L1, Loan L2
where L1.uID = L2.uID  and L1.title = 'Bambi' and
L2.title = 'Lion King';
```

Difference

```
select uID from Loan where title ='Bambi'
except
select uID from Loan where title = 'Lion King';
```

Subqueries

- (select-from-where) can be used as a value in select, from and where clause.
- Example
  - In place of a relation in the FROM clause, we can use a subquery and then query its result.
  - Note: if you are using a subquery in the **from** clause, must use a tuple-variable to name tuples of the subquery result.

Subqueries that produce scalar values

```
Movies(title,year,length,genre,studioName,producerC#)
MovieExec(name, address, cert#, netWorth)
```

[Q]To find the producer name of Star Wars


```
SELECT name
FROM MovieExec
WHERE cert# =
  (SELECT producerC#
   FROM Movies
   WHERE title = 'Star Wars');
```

Conditions Involving Relations

- EXIST R true if and only if R is not empty
- NOT EXIST R true if and only if R is empty.
- Assuming R is a unary relation, with a scalar value s
  - s IN R true if and only if s is equal to one of the values in R
  - s > ALL R true if and only s is greater than **every value** of R (s <> ALL R is the same as s NOT IN R.)
  - s > ANY R true if and only if s is greater than **at least one** value in
  - IN, ALL, and ANY can be negated by NOT

    NOT s >= ALL R: s is not the maximum

    NOT s > ANY R: s is the minimum

Conditions Involving Tuples

If a tuple t has the same number of components as a relation R, you may compare t and R using IN, ALL, an ANY.

Example: Conditions involving tuples

```
Movies(title,year,length,genre,studioName,producerC#)
StarIn(movieTitle, movieYear, starName)
MovieExec(name, address, cert#, netWorth)
```

[Q] To find the producer name of Harrison Ford's movies


```
SELECT name
FROM MovieExec
WHERE cert# IN
  (SELECT producerC#
   FROM Movies
   WHERE (title, year) IN
     (SELECT movieTitle, movieYear
      FROM StarsIn
      WHERE starName = 'Harrison Ford'
     )
  );
```

Alternative to Subqueries

[Q] To find ids and ages of users whose loan being overdue


```
select uID,age
from  User
where uID in
(select uID from Loan where overdue = true);

select distinct User.uID, age
from User, Loan
where User.uID = Loan.uID and overdue=true;
```

Duplicates matter ?

[Q] To find ages of users who are being overdue.

```
select age
from User
where uID in
(select  uID from Loan where overdue = true);
```



```
/* The following doesn't work with/without distinct */
select  distinct age
from User, Loan
where (User.uID = Loan.uID and  overdue = true);
```

Correlated Subqueries

- If a subquery uses a tuple variable from outside subquery, it will be evaluated many times once for each value of the tuple variable.
- Example: To find the user names appearing two or more times in User relation


```
SELECT uNAME
FROM USER old
WHERE uID != ANY
(SELECT uID FROM USER WHERE uNAME =old.uNAME);
```

- Alternative query?

Scoping Rules (for attribute names)

- Without a qualifier dot (.)
  - An attribute in a subquery belongs to one of the relations in the FROM clause of the subquery if the relation has the attribute.
  - If not, look at the immediately surrounding subquery, and continue until you find it.
- With a qualifier dot (.)

  `tupleVariable.attributeName` describes which relation the attribute belongs to and can also disambiguate attributes with the same name.

Rewriting difference using subqueries

```
select uID from Loan where title ='Bambi'
except
select uID from Loan where title = 'Lion King';

select uID from User where
  uID in (select uID from Loan where title = 'Bambi' )
  and
  uID not in (select uID from Loan where title = 'Lion King');
```

Example: exists

EXISTS R returns true if and only if R is not empty.

[Q] To find books written by the same author


```
select title, author
from Book B1
where exists
      (select * from Book B2
       where  B1.author = B2.author and
              B1.title <> B2.title);
```

Example: not exist

NOT EXISTS R returns true if R is empty.

[Q] To find the book with the **maximum** number of copies.


```
select  title, copies
from Book B1
where not exists
 (select * from Book B2 where B1.copies < B2.copies);
```



```
select uName, age
from User U1
where not exists
(select * from User U2 where  U1.age < U2.age) ;
```

Does it work ?

To find a maximum age from the user,


```
select distinct U1.uName,  U1.age
from User U1, User U2
where U1.age > U2.age;
```


No. It shows all users except for the youngest.

Example: all

With a scalar value s,

s > ALL R is true if and only if s is greater than **every value** in unary relation R.


```
select  uName, age
from User
where age >= all (select age from User)
```

This works assuming that age is not being null.

To handle null ages

```
select  uName, age
from User U1
where age >=all(select age
                from User u2
                where  U2.age is NOT NULL);
```

Example: any
(SQLight doesn't support any)

s > ANY R is true if and only if s is greater than **at least one** value in unary relation R.

[Q] To find users not borrowing the smallest number of books.


```
select  uID, uName, loaned
from User
where loaned > any (select loaned from User);
```

```
select uID
from User U1
where exists (select loaned
                from User U2
                 where U1.loaned > U2.loaned);
```

Does it work ?

To find users(s) who borrowed 'The Silver Sun' but not 'Lion King'

select uID from User where

uID = any (select uID from Loan where title = 'The Silver Sun') and

uID **<> any** (select uID from Loan where title = 'Lion King');

No.

How to fix it ?

```
select uID from User where
uID = any (select uID from Loan where title =
'The Silver Sun' )  and
uID <>all (select uID from Loan where title =
'Lion King');
```


Note: not uID = any is equivalent to uID <> all

Column alias and Where clause

The standard SQL doesn't allow where clause references a column alias. Think of the order of clause evaluation!

Example: Query with such an error


```
select uID, uName, age, loaned, 10-loaned as quotaLeft
from User
where  quotaLeft < 1;
```

Subquery in From clause

```
select *
from ( select uID, uName, loaned, 10-loaned as quotaLeft
from User) QL
where QL.quotaLeft < 1;
```


Note: you must give the result of subquery a tuple variable alias such as QL.

Example: Subquery in From clause

```
select BambiUser.avgAge - NonBambiUser.avgAge
from

(select avg(age) as avgAge from User where uID in
(select uID from Loan where title = 'Bambi' ) ) as
BambiUser,

(select avg(age) as avgAge from User where uID not
in (select uID from Loan where title = 'Bambi' ) ) as
NonBambiUser;
```

Correlated Subqueries in Where clause

[Q] To pair up each loaned book with the author and the age of the oldest borrower of the book


```
select distinct Book.title, author, age
from User, Book, Loan
where Book.title=Loan.title and User.uID = Loan.uID
        and age >= all   (select age
                          from User, Loan
                          where User.uID = Loan.uID and
                          Book.title =Loan.title and
                          age is not null);
```

title	author	age
Bambi	Felix Salten	23
Database Systems	Jennifer Widom	28
Eye of Sierras	Robin Jones Gunn	10
Lion King	Dan Ferguson	35
The Silver Sun	Nancy Springer	67

Subqueries in Select

```
select title, author,
(select distinct  age
from User, Loan
where Book.title=Loan.title and User.uID = Loan.uID
and age >= all
     (select age
      from User, Loan
      where User.uID = Loan.uID and Book.title =Loan.title and age is not null)
) as age
from  Book;
```


Note:
You can alias the column as you want, not necessarily age.
In MySQL, if you don't alias it, the entire subquery appears as the column name.

title	author	age
Bambi	Felix Salten	23
Database Systems	Jennifer Widom	28
Evening in the Ashes	Dorothy Love	NULL
Every Perfect Gift	Dorothy Love	NULL
Eye of Sierras	Robin Jones Gunn	10
Faraway Child	Amy Maida Wadsworth	NULL
Lion King	Don Ferguson	35
Sisterchicks Say Oho La La !	Robin Jones Gunn	NULL
The Path's Secrets	Sayyid Haydar Amuli	NULL
The Sage and the Lace	James Dove	NULL
The Silver Sun	Nancy Springer	67

Note: Books which no one checked out have null age.

Does it work ?

[Q] To list book and names of users who borrow the book

```
select title, author,
(select distinct  uName
from User, Loan
where User.uID = Loan.uID and Book.title=Loan.title
) as uName
from  Book;
```


No. When a subquery is used in select, it has to return exactly one result.