CS 143 Discussion Session

Week 2

TA and reader

- TA Mingda: for the TA work in the first half: discussion, Piazza.
- TA Jin: for the TA work in the second half: discussion, Piazza.
- Reader: Qiqi for grading and regrading work of Projects and questions about (re)grading for Proj on Piazza.
- For the contents about Project 1: Mingda. For the regrading about Project 1: Qiqi.
- For the contents about Project 2: Jin.

  For the regrading about Project 2: Qiqi.
- Homework 1-3: Mingda. Homework 4-6: Jin. (contents or grading).

Reminder

- Project 1 has been released.
- The penalty for late submission:
- Each day extra 5% off. Until Friday (4 days).
- Then, we will not accept your submission.

SQL

- Structured Query Language
  - Designed for RDBMS (Relational Database Management System)
  - Declarative Language
- Relational Algebra
  - SQL's mathematical interpretation

SQL

- Queries can be categorized to:

- Data Definition Language (DDL) statements are used to define the database structure or schema. Some examples:
  - CREATE - to create objects in the database
  - ALTER - alters the structure of the database
  - DROP - delete objects from the database
- Data Manipulation Language (DML) statements are used for managing data within objects. Some examples:
  - SELECT - retrieve data from the database
  - INSERT - insert data into a table
  - UPDATE - updates existing data within a table
  - DELETE - deletes records from a table

DDL

CREATE

CREATE TABLE table_name {

attribute1 data_type,

attribute2 data_type,

…

PRIMARY KEY (attribute *n*),

FOREIGN KEY (attribute *m*) REFERENCES Table2(attribute *x*),

CHECK (constraint)

}

Primary Key

- The PRIMARY KEY constraint uniquely identifies each record in a table.

- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key. And the primary key can consist of single or multiple columns (fields).

Example

- CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  PRIMARY KEY (ID)
)

Foreign Key

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Example

"Persons" table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.
The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

Foreign key

- CREATE TABLE Orders (
  OrderID int NOT NULL,
  OrderNumber int NOT NULL,
  PersonID int,
  PRIMARY KEY (OrderID),
  FOREIGN KEY (PersonID) REFERENC
  ES Persons(PersonID)
  );

Other constraints

- NOT NULL:
  - The NOT NULL constraint enforces a column to NOT accept NULL values.
- E.g. CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255) NOT NULL,
  Age int
  );

Other constraints

- UNIQUE:
- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

E.g.

```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

Other constraints

- Check:
- The CHECK constraint is used to limit the value range that can be placed in a column.
- CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  CHECK (Age>=18)
  );

DML

SELECT attributes, aggregates

FROM relations(tables)

WHERE conditions

GROUP BY attributes

HAVING conditions on aggregates

ORDER BY attributes, aggregates

- Evaluation order:
  - FROM → WHERE → GROUP BY → HAVING → ORDER BY →SELECT

- Relational Algebra Counterparts
  - FROM $R_1, ..., R_m$: $R_1 \times \cdots \times R_m$
  - WHERE C: σ_C
  - SELECT $A_1, ..., A_n$: $\pi_{A_1, \ldots, A_n}$

Comparison Operators

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=. But we use <> for this course.

Example for SQL:

- Given a table of students:
- (ID, age, Country, City, Name)
- 1. Select:
- *SELECT Country FROM students;*
- -- selects ALL (including the duplicates) values from the "Country" column
- 2. Distinct:
- *SELECT DISTINCT Country FROM students;*

Example for SQL:

- 3. Where:
- SELECT * FROM Customers
  WHERE Country='China';
- 4. 'Where' with 'and' 'or' 'not'
- SELECT *column1*, *column2*, ...
  FROM *table_name*
  WHERE *condition1* AND *condition2* AND *c*
  ondition3 ...;

Example for SQL:

- 5. Not syntax
  - SELECT *column1*, *column2*, ...
    FROM *table_name*
    WHERE NOT *condition*;

SELECT * FROM Customers
WHERE NOT Country='Germany';

-- Will select all fields from "Customers" where country is NOT "Germany".

Aggregates

- The MIN() function returns the smallest value of the selected column.
- The MAX() function returns the largest value of the selected column.
- The COUNT() function returns the number of rows that matches a specified criteria.
- The AVG() function returns the average value of a numeric column.
- The SUM() function returns the total sum of a numeric column.

E.g.

- SELECT COUNT(ProductID)
  FROM Products;
- SELECT AVG(Price)
  FROM Products;
- SELECT MAX(Price) AS LargestPrice
  FROM Products;
- -- AS is alias.

Group by and Having

- When aggregate operators are not be applied to all (qualifying) tuples.
  - Sometimes, we want to apply them to each of several *groups* of tuples.
  - E.g.
  - SELECT COUNT(CustomerID), Country
    FROM Customers
    GROUP BY Country;
  - SELECT COUNT(CustomerID), Country
    FROM Customers
    GROUP BY Country
    ORDER BY COUNT(CustomerID) DESC;
  - -- DESC: High -> Low

- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.
  - E.g. SELECT COUNT(CustomerID), Country
    FROM Customers
    GROUP BY Country
    HAVING COUNT(CustomerID) > 5;
- -- The SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

Having

- Condition tests on the groups
  - Executed after group by
- Conditions on aggregates should appear in the HAVING clause
- We can rewrite a query not to have a HAVING clause – Can be complicated!

Important points

- Tables in FROM clause
  - cross product, not natural join
- SELECT clause
  - Projection, not selection in RA
- SQL uses bag semantics, duplicates not removed
  - SELECT distinct … to remove duplicates
- Tables/Attributes can also be renamed
  - GPA (AS) grade

Set Operators

- ∩: INTERSECT, ∪: UNION, −: EXCEPT
- Set operators should have the same schema for operands
  - In practice, it is okay to have just compatible types
- Set operators follow set semantics and remove duplicates
  - keep duplicates, use UNION ALL, INSERSECT ALL, EXCEPT ALL (bag semantics)

E.g.

- SELECT City FROM Customers
  UNION
  SELECT City FROM Suppliers
  ORDER BY City;

Other set operators

- Set membership
  - IN, NOT IN
- Set comparison operator
  - > ALL, < SOME, = SOME, ..., etc.

E.g.

- SELECT * FROM Customers
  WHERE Country IN (SELECT Country F
  ROM Suppliers);
- SELECT * FROM Customers
  WHERE Country IN ('Germany', 'France',
  'UK');

NULL

- A field with a NULL value is a field with no value.
- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- We will have to use the IS NULL and IS NOT NULL operators instead.
-

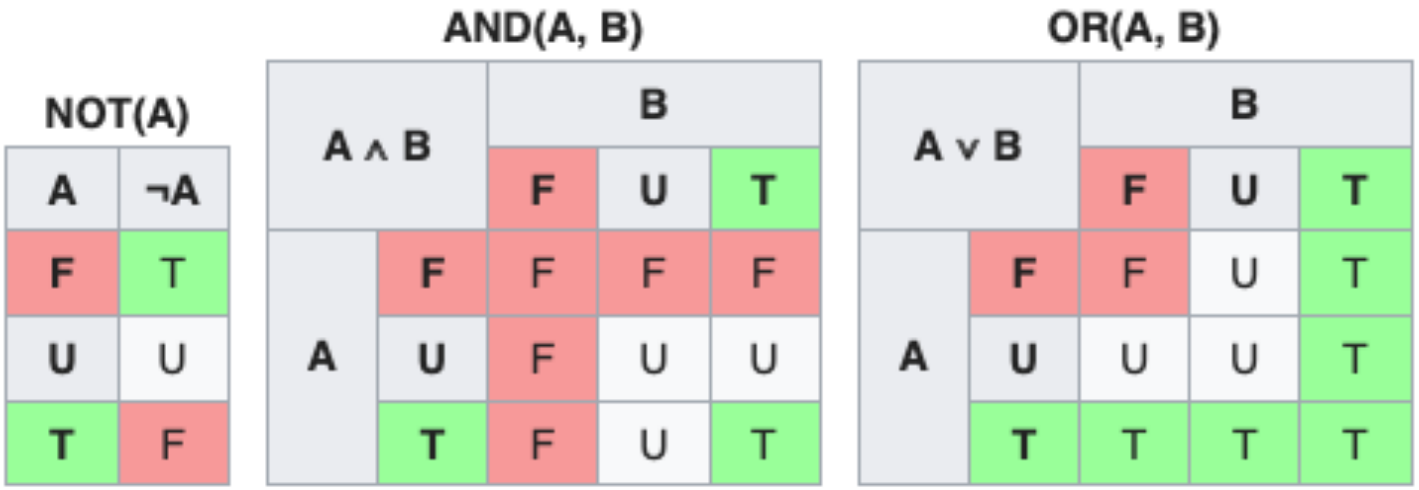
Count()

- COUNT(*) returns the number of items in a group. This includes NULL values and duplicates.
- COUNT(ALL *expression*) evaluates *expression* for each row in a group, and returns the number of nonnull values. ALL expression = expression. So COUNT(expression) for this line can work.
- COUNT(DISTINCT *expression*) evaluates *expression* for each row in a group, and returns the number of unique, nonnull values.

NULL value

- NULL:
- Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned) or *inapplicable* (e.g., no spouse's name).
  - SQL provides a special value *null* for such situations.
- The presence of *null* complicates many issues. E.g.:
  - Special operators needed to check if value is/is not *null*.
  - What about AND, OR and NOT connectives?
  - We need a 3-valued logic (true, false and *unknown*).

(F, false; U, unknown; T, true)



NOT(A)

A	¬A
F	T
U	U
T	F

AND(A, B)

A ∧ B		B		
		F	U	T
A	F	F	F	F
	U	F	U	U
	T	F	U	T

OR(A, B)

A ∨ B		B		
		F	U	T
A	F	F	U	T
	U	U	U	T
	T	T	T	T

(–1, false; 0, unknown; +1, true)

- Enjoy your long weekend!