AY2023/24 Trimester 3

Dr. ZHANG Wei

INF2003: Database Systems

Communication with DBMS
SQL Basics

1

This lecture is about some basics of SQL.

For those who are familiar with SQL, you can skip (part of) this lecture and allocate your time to some more meaningful learning activities.

Of course, you are welcome to join us if you would like to refresh your SQL knowledge.

https://miro.medium.com/max/4230/1*mo7B0h9PHYGmwaJ1ouUl8g.png

3

Communication with DBMS

- What to communicate?
  - **Structure**: create a database, delete a table, etc.
  - **Data**: Get some data. Delete some data. Update some values. Insert some data.
    - Also called data manipulation language (DML).
    - Our focus in this lecture.
- How can we talk to a DBMS?
  - Relational algebra. Mathematical-driven.
  - Relational calculus.
  - **SQL**. Programming-driven.
- Relational algebra: Math is not intuitive, but why people like it?
  - Clean. Precise. Possibly also effective.
  - May be not human-friendly, but computer friendly.
  - Key DBMS competitiveness is not about programming, but math optimization.

https://miro.medium.com/max/4230/1*mo7B0h9PHYGmwaJ1ouUl8g.png

4

SQL Overview

- SQL: Structured Query Language.
  - IBM's legacy. Was called Structured English Query Language.
    - Same motivation as Python. Let us code like speaking English.
  - Rapid **growth**. Now it is a standardized (even ISO) way for database queries.
  - Advantages: **simple** and **declarative**
    - Users do not need to think much, and **software** can do the operation optimization.
- Main types:
  - DDL: data definition language.
    - More for the relational models, i.e., schema, ICs and index.
    - Col-wise.
  - DML: data manipulation language.
    - CRUD: create, read, update, and delete.
    - Can be very complex among different tables.
    - Row-wise.
  - The others, like views and transactions.

https://www.dsdinc.com/wp-content/uploads/2020/02/SQL_mirror.png

5

SQL Basics

- Case sensitive (we can configure).
  - However, for good practice, **do not** use the same word for different things.
- Remember to add a ; after the SQL statement.
- Allows **comments**, which can be either single line or a block.
  - -- This is a comment for SQL2008.
  - /* line 1 line 2 … line n */
- One application often works with **one database**.
  - Many app – one DB more common than one app – many DBs.
- One database often has **many tables**.
  - Relational data model prefers 'relation'.
  - SQL prefers 'table'.
- Sample DDL: **CREATE DATABASE** dbName;
  - Create a new database called dbName.
  - Good practice: check if it exists first.
  - What if it exists?

https://xsolutions.com/wp-content/uploads/2019/06/imgo-3.jpg

6

CREATE TABLE

- Create a new table in a database.
  - Table name: table_name.
  - A list of attributes, or cols, with date types.
  - Can specify DB name, i.e., USE db1 or db1.tbName.
- **Name** your databases and tables **properly**.
  - Analyze the **application** requirements, e.g., StudentICT and Student2019.
  - No right or wrong technically, but quite important non-technically.
  - **First** character shall be a **letter**.
  - **Uppercase** and **lowercase** for different letters in one name.
  - Use **underscore** to connect different words.
  - Follow the **same naming schemes** for a whole project.
- **Domain** or data types:
  - CHAR(n), a string with fixed length n; VARCHAR(n), max-length n.
  - Use objects BLOB, CLOB, TEXT if it is very large, like a full article.
  - INT, an integer of 4B; variants TINYINT (1B), SMALLINT(2B), and BIGINT(8B).
  - Use unsigned value like in C? Not popular (or even available) in DBMS.

```
CREATE TABLE table_name (
       column1 datatype,
       column2 datatype,
       ....
);
```

https://tfwiki.net/mediawiki/images2/thumb/6/68/TLK_Quintessa_face.jpg/400px-TLK_Quintessa_face.jpg

7

More Data Types

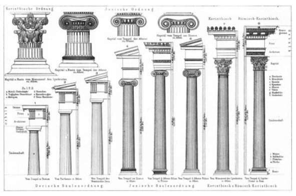
```
CREATE TABLE table_name (
        column1 datatype,
        column2 datatype,
        ....
);
```

- **NUMERIC**(P, D) with:
  - P: precision or the total digits. Determines bytes.
  - D: scale or the digits after the decimal place.
- **FLOAT**(n) with n-bit precision (n in [1, 53]).
- **DOUBLE PRECISION**: (2 floats)
  - Or REAL for some implementations.
  - Approximate number. Do not use for fully precise attributes.
- **DATE**, TIME, TIMESTAMP.
  - Like for sensor data and bank transactions.
  - Can manipulate this in Pandas with datetime data type.
- Can user-define as well, like a **class** in C++.
- Overall, **plan well** for data types, e.g., to save storage.
  - Student ID: shall this be integer? Char? How long? Same for all programs?
  - Database is a technology but shall plan **according to the real applications**.

https://cdn.onlinewebfonts.com/svg/img_517066.png

8

More DDLs



- **CREATE DATABASE** db_name;
- **DROP DATABASE** db_name;
- **DROP TABLE** table_name;
- **ALTER TABLE** table_name **ADD** column_name datatype;
- **ALTER TABLE** table_name **DROP COLUMN** column_name;
- **ALTER TABLE** table_name **MODIFY COLUMN** column_name datatype;
- **GRANT**, **REVOKE**, etc.
- The above DDLs, of course, we can do.
  - BUT think carefully **if we should** before we take action.
- Also be aware that the **grouping** of keywords is **not unique**.
  - Some people also use DCL (data control language for GRANT and REVOKE).
  - Some people use DQL (data query language for SELECT).
  - We follow the simple and popular grouping in this module, only DDL and DML.

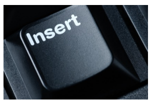
https://en.wikipedia.org/wiki/Column#/media/File:Schema_Saeulenordnungen.jpg break

9

INSERT

- Table after created is still **empty**.
- **Insert** for adding new rows.
  - Can be all the cols, or a **subset**.
    - e.g., the cols with default values.
  - **Order sensitive**.
    - May be different from the relation schema.
    - But specify the col order and let the values follow the **same order**.
- Specify ICs (integrity constraints) when creating the tables.
  - **Primary key** IC: after all cols, or after the key col.
    - For the former we can make >1 cols together as key.
    - **Only 1** primary key.
  - Once specified, no trouble for duplicated rows.
  - Database system will check for us every time.

```
INSERT INTO table_name (col3, col5, ...)
VALUES (val3, val5, ...);
          OR
INSERT INTO table_name
VALUES (val1, val2, ...);
```



```
CREATE TABLE table_name (
    col1 datatype,
    col2 datatype,
    ....,
    PRIMARY KEY (colx)
);
```

10

SELECT

- The **#1 popular** SQL keyword.
  - Indicate the **cols** to return, not necessarily all.
    - If all, just use **SELECT ***.
  - Indicate the **table** with the cols specified.
    - Can be multiple tables, e.g., join them.
  - Indicate the selection **condition**.
    - Refer to the selection operator for relational algebra.
- To **remove duplications**, SELECT **DISTINCT** A1, A2, …
  - Time consuming.

```
SELECT col1, col2, ...
FROM table_name
WHERE condition
```

```
SELECT A1, A2, ...
FROM r1, r2, …
WHERE P
```

1025

11

SELECT Examples

- Q1: Find the name and age of all sailors.
  - If we want all rows and cols, easy, SELECT *.
- Q2: Find **DISTINCT** name and age of all sailors.
- Q3: Find all sailors with rating > 7.
- Can do **generalized projection** with calculations?
  - SELECT (rating/2), age FROM ….
  - Col name depends.
    - Some DBMS just say ‘rating/2’ or variants.
    - We can use keyword **as** to name.

sid	name	rating	age
22	Dustin	7.5	45
31	Lubber	8.0	55
58	Rusty	9.0	35
74	Rusty	7.5	35

Sailors

name	age
Dustin	45.0
Lubber	55.5
Rusty	35.0
Rusty	35.0

name	age
Dustin	45.0
Lubber	55.5
Rusty	35.0

name	age
Lubber	55.5
Rusty	35.0

Ans 1:
```
SELECT name, age
FROM Sailors;
```
Ans 2:
```
SELECT DISTINCT name, age
FROM Sailors;
```
Ans 3:
```
SELECT name, age
FROM Sailors
WHERE rating > 7;
```

12

WHERE with Multiple Tables

- Can be more complex, e.g., mul-tables.
- Many **operators** supported.
  - **Comparison**: >, <, =, <> or != (not equal), etc.
  - **Arithmetic**: +, -, x, /, etc.
  - Refer to any attributes from the input relations.
- Q1: Find the name of the sailors who have reserved boat number 103.

```
SELECT Sailors.name
FROM Sailors, Reserves
WHERE Sailors.sid=Reserves.sid
    AND Reserves.bid=103
```

- Range variables, needed if a relation appears multiple times.

```
SELECT S.name
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=103
```

Sailors

sid	name	rating	age
22	Dustin	7.5	45
31	Lubber	8.0	55
58	Rusty	9.0	35
74	Rusty	7.5	35

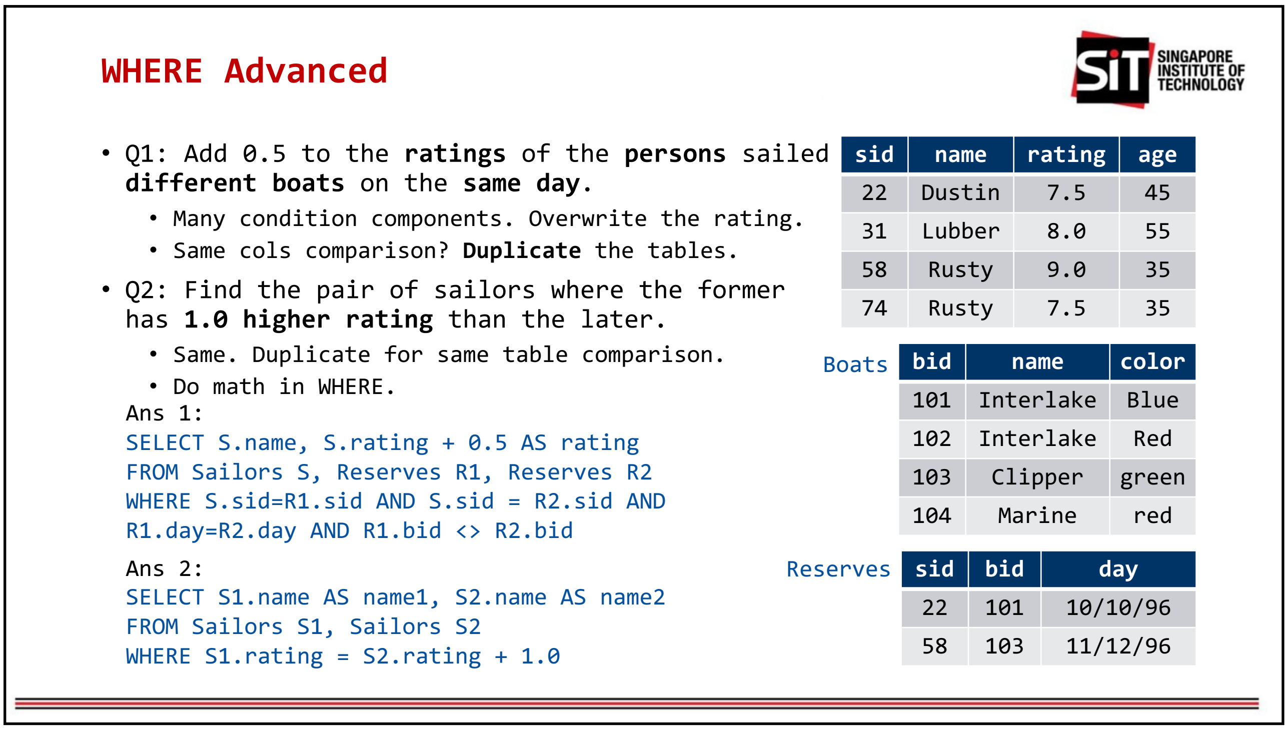
Boats

bid	name	color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	green
104	Marine	red

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

13



WHERE Advanced

- Q1: Add 0.5 to the **ratings** of the **persons** sailed **different boats** on the **same day**.
  - Many condition components. Overwrite the rating.
  - Same cols comparison? **Duplicate** the tables.
- Q2: Find the pair of sailors where the former has **1.0 higher rating** than the later.
  - Same. Duplicate for same table comparison.
  - Do math in WHERE.

Ans 1:

```
SELECT S.name, S.rating + 0.5 AS rating
FROM Sailors S, Reserves R1, Reserves R2
WHERE S.sid=R1.sid AND S.sid = R2.sid AND
R1.day=R2.day AND R1.bid <> R2.bid
```

Ans 2:

```
SELECT S1.name AS name1, S2.name AS name2
FROM Sailors S1, Sailors S2
WHERE S1.rating = S2.rating + 1.0
```

sid	name	rating	age
22	Dustin	7.5	45
31	Lubber	8.0	55
58	Rusty	9.0	35
74	Rusty	7.5	35

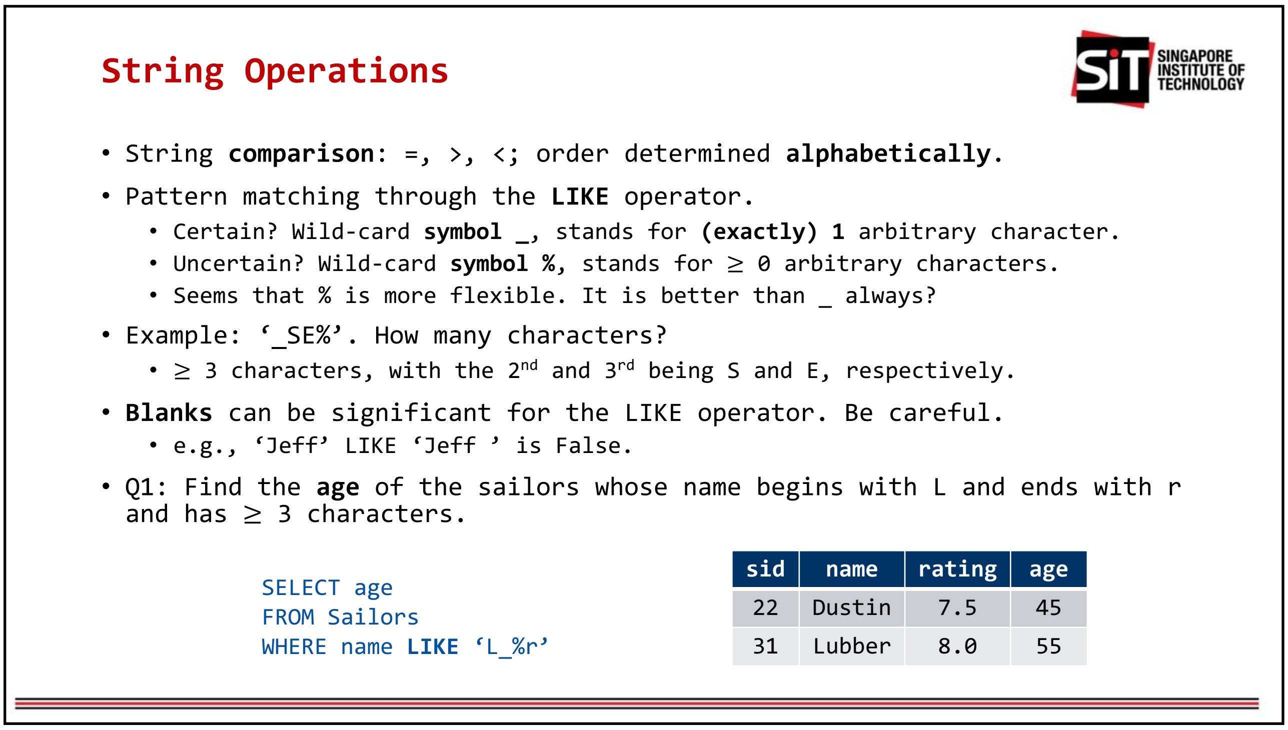
Boats

bid	name	color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	green
104	Marine	red

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

14



String Operations

- String **comparison**: =, >, <; order determined **alphabetically**.
- Pattern matching through the **LIKE** operator.
  - Certain? Wild-card **symbol** _, stands for **(exactly) 1** arbitrary character.
  - Uncertain? Wild-card **symbol** %, stands for ≥ 0 arbitrary characters.
  - Seems that % is more flexible. It is better than _ always?
- Example: '_SE%'. How many characters?
  - ≥ 3 characters, with the 2nd and 3rd being S and E, respectively.
- **Blanks** can be significant for the LIKE operator. Be careful.
  - e.g., 'Jeff' LIKE 'Jeff ' is False.
- Q1: Find the **age** of the sailors whose name begins with L and ends with r and has ≥ 3 characters.

```
SELECT age
FROM Sailors
WHERE name LIKE 'L_%r'
```

sid	name	rating	age
22	Dustin	7.5	45
31	Lubber	8.0	55

15