SQL BASICS WITH THE SMALLBANKDB

STEFANO GRAZIOLI & MIKE MORRIS

This handout covers the most important SQL statements. The examples provided throughout are based on the SmallBank database discussed in class.

To connect with Visual Studio from the lab, go to tools > Connect to Database. Enter MS Sql Server and f-sg6m-s4.comm.virginia.edu. Test the connection. You should see the SmallbankDB in the Server explorer window.

To connect with Toad from the lab, go to connect > MS Sql Server and f-sg6m-s4.comm.virginia.edu. You should see the SmallbankDB in the Server explorer window.

Connecting from home is a little more convoluted. Ask the instructor.

1. SINGLE TABLE QUERIES

SELECTING DATA FROM A TABLE

1) Choosing all fields (columns)

```
SELECT *
FROM table_name;

SELECT *
FROM Customer;
```

2) Choosing a selected list of fields (columns)

```
SELECT column_name [, column_name, …]
FROM table_name;

SELECT f_name, l_name, date_of_birth
FROM Customer;
```

- The order in which you list the columns affects the way in which they are presented in the resulting output.
- Items within [ ] are optional.

3) Temporarily renaming columns in query results

```
SELECT column_name AS "column heading" [, column_name AS
"column_heading"]
FROM table_name;
```

Example:

```
SELECT f_name as "Customer Name"
FROM Customer;
```

4) Including calculated columns in the results

```
SELECT date_due, rate, principal, rate * principal
FROM loan;
```

- If necessary, use parentheses to clarify order of precedence in a computation, as in a * (b+c)

5) Eliminating duplicate query results with *distinct*

If you use the keyword *distinct* after the keyword SELECT, you will only get unique rows. Example:

```
SELECT rate
FROM Loan;
```

VS.

```
SELECT distinct rate
FROM Loan;
```

6) Selecting rows: the *where* clause

```
SELECT Select_list
FROM table
WHERE search_conditions;
```

Example:

```
SELECT *
FROM Customer
WHERE f_name = 'Carl';
```

- In SQL, strings are delimited by single quotes, as in 'Carl'

AVAILABLE SEARCH CONDITIONS OPERATORS

Comparison operators (=, <, >, !=. <>, <= ,>=)

```
SELECT * FROM loan
WHERE principal > 100000000;
```

- Ranges (**between** and **not between**; inclusive of the end values)

```
SELECT * FROM loan
WHERE rate BETWEEN 7.5 AND 8.5;
```

- Lists (**in** and **not in**)

```
SELECT *
FROM Customer
WHERE city IN ('Cville', 'Roanoke', 'Lexington');
```

- Character matches (**like** and **not like**)

```
SELECT f_name, l_name
FROM Customer
WHERE l_name LIKE 'Fos%';

SELECT f_name, l_name
FROM Customer
WHERE l_name LIKE '_oster';
```

- "%" (matches any string of zero or more characters) and "_" (matches any one character). In addition to those, brackets can be used to include either ranges or sets of characters.
- Combinations of previous options using logical operators **and**, **or**, and **not**

```
SELECT f_name, l_name
FROM Customer
WHERE l_name LIKE 'Fos%' AND City NOT IN ('Austin', 'Dallas');
```

SUMMARIZING, GROUPING, AND SORTING QUERY RESULTS

1) Aggregate functions

- Types of aggregate functions: sum, avg, count, count(*), max, min

```
SELECT SUM (principal) FROM loan;
```

```
SELECT AVG (rate) FROM loan;

SELECT MIN(rate), MAX(rate), COUNT(rate)
FROM loan;
```

- The **where** clause can be used to define the set of rows to which the aggregate functions apply

```
SELECT AVG (principal)
FROM loan
WHERE rate > 8.5;
```

- Difference between **count** and **count(*)**: **count** returns the number of non-null values in a specific column, whereas **count(*)** returns the number of rows.

```
SELECT COUNT(*) FROM customers;

SELECT COUNT(city) FROM customers;
```

- The keyword **distinct** can be used with sum, avg, and count to eliminate duplicate values before the calculations are made. Distinct appears inside the parenthesis and before the column name.

```
SELECT COUNT(DISTINCT city) FROM customers;
```

2) Using aggregate functions with groupings

- The **group by** clause can be used in select statements to divide a table into groups and get results (normally aggregates) separately for each group.

```
SELECT rate, AVG(principal)
FROM loan
GROUP BY rate;
```

- The **where** clause can be used in a statement with **group by**. Only those rows that satisfy the condition will be included in the grouping.

```
SELECT rate, AVG(principal)
FROM loan
WHERE principal > 50000000
GROUP BY rate;
```

- The types of groups that will be included in the answer set can be limited with the having keyword. **Having** sets conditions for groups in the same way **where** sets conditions for individual rows. Aggregate functions can be used in a **having** clause.

```
SELECT rate, AVG(principal)
FROM loan
```

```
GROUP BY rate
HAVING AVG(principal) > 50000000;
```

3) Sorting query results with the order by clause

- An **order by** clause is used to request the results of data retrieval in either ascending (**ASC**, which is the default) or descending (**DESC**) order by one or several (max 16) columns

```
SELECT *
FROM loan
ORDER BY rate;
```

- Multiple sorts are possible

```
SELECT *
FROM customer
ORDER BY l_name, f_name;
```

MULTIPLE TABLE QUERIES

SELECTING DATA FROM MULTIPLE TABLES: RELATIONAL JOINS

- Relational joins are a tool for combining data from multiple tables
- They are the characteristic feature of the relational database management systems
- A "join" corresponds to the intuitive operation of combining the data in two tables by using the values in one column in the first table and matching them with the values of another column in the second table.
- Joins exploit the relationships between tables. In the most common case, a join matches a foreign key in one table and the primary key in the other.
- Queries that include multiple joins are possible. These queries "hop" from one table to the next, to the next, to the next.

1) Joining tables using a foreign key/primary key combination

```
SELECT l_id, principal, date_due, loan_officer.lo_id, l_name
FROM loan, loan_officer
WHERE loan.lo_id = loan_officer.lo_id;
```

- Table name qualifiers (customer and product in the example above) are used when a column name is not unique and we have to clarify to which column we are referring. The format is *tableName.attributeName*
- The where clause restricts the entries to those where the join condition is true.
- If the **where** clause is (accidentally) omitted, SQL returns a result that contains the "Cartesian product" of the tables, i.e., all possible combinations of all the rows from all the tables. Thus, if the customer table contained 30 entries and the product table contained 18 entries, the Cartesian product consists of (30x18=) 540 entries. This is very rarely what you intended. Bottom line: remember to include the **where** clause!
- The column set to be displayed can come from either one of the tables, or from both.
- There are several styles to write joins. You might be familiar with a different one. That is ok. They are all equivalent when used correctly. Use the style that you find easier.

2) Adding elements to the *where* clause

```
SELECT l_id, principal, date_due, loan_officer.lo_id, l_name
FROM loan, loan_officer
WHERE loan.lo_id = loan_officer.lo_id
AND principal > 10000000;
```

- Any combination of logical operators can be used to combine conditions in the **where** clause

3) Joining three or more tables

- Joins are not limited to two tables; however, you will seldom see queries with more than 6 or 7 tables joined together. "Normal" is 2-4 tables. Here is an example with 4 tables.

```
SELECT customer.f_name, customer.l_name
FROM loan_officer, loan, customer_in_loan, customer
WHERE loan_officer.l_name = 'Romani'
AND loan_officer.lo_id = loan.lo_id
AND loan.l_id = customer_in_loan.l_id
AND customer_in_loan.c_ssn = customer.c_ssn;
```

- The columns used to join the tables (order number and product number above) may be included in the ***select*** statement but do not have to be.
- What does this query compute? Make sure that you understand.

INSERTING, UPDATING and DELETING rows

INSERTING A NEW ROW INTO A TABLE

```
INSERT INTO table_name (column1, column2, column3…)
VALUES (value1, value2, value3, …)
```

- If the order of the values is the same as the order of the columns in the table, the column specification can be omitted (see example below)
- Strings are delimited by single quotes
- You need permission from the sys admin to insert rows in a table.

```
INSERT INTO Customer
VALUES (2323, 'John', 'Smith', 'Cville', 'VA')
```

Or

```
INSERT INTO Customer (f_name, l_name, c_id, state, city)
VALUES ('John', 'Smith', 2323, 'VA', 'Cville')
```

- Inserting a duplicated primary key will give you an error.

UPDATING ONE OR MORE ROW IN A TABLE

```
UPDATE table_name
SET column = value
WHERE condition
```

- Depending on the condition, one or more (or none) rows will be changed
- Careful with UPDATE! There is no 'undo'

```
UPDATE Customer
SET f_name = 'Jane'
WHERE c_id = 2323

UPDATE Customer
SET city = 'Charlottesville'
WHERE city = 'Cville'
```

DELETING ONE OR MORE ROW IN A TABLE

```
DELETE FROM table_name
WHERE condition
```

- Depending on the condition, one or more (or none) rows will be changed
- **Careful with DELETE!** There is no 'undo' and if you forget to specify the condition, the whole table will be cleared.

```
DELETE FROM Customer
WHERE c_id = 2323
```

- The next example deletes multiple rows. If you forget to specify the condition, the whole table will be cleared. Again, there is no 'undo'.

```
DELETE FROM Customer
WHERE city = 'Cville'
```