SQL BASICS with the SmallBankDB  
Stefano Grazioli & Mike Morris

This handout covers all the basic SQL statements. Examples based on the SmallBank database are provided after each statement.

To connect from VS, go to tools > Connect to Database. Enter MS Sql Server and f-sg6m-s4. Test the connection. You should see the SmallbankDB in the Server explorer window.

It might require a VPN if not logged on the McIntire network.

# 1. SINGLE TABLE QUERIES

## SELECTING DATA FROM A TABLE

1. **Choosing all fields (columns)**

SELECT \*

FROM *table\_name;*

SELECT \*

FROM Customer;

1. **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.

1. **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 “Name”

FROM Customer;

1. **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)

1. **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;

1. **Selecting rows: the *where* clause**

SELECT Select\_list

FROM table

WHERE search\_conditions;

Example:

SELECT \*

FROM Customer

WHERE f\_name = ‘Carl’;

* In SQL, string 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;

1. **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;

1. **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 system
* A “join” correspond to the intuitive operation of using the values in one column in one table and matching them with the values of another column in another table.
* Joins implement the relations 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. Their format is *tableName.attributeName*
* 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 540 entries. This is very rarely what you intended. Bottom line: remember to include the **where** clause!
* The**where** clause restricts the entries to those where the join condition is true.
* The column set to be displayed can come from either one of the tables, or from both.

1. **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

1. **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 3 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.