Handout - Advanced T-SQL

1.SUMMARIZING, GROUPING, AND SORTING QUERY RESULTS

1.1 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;

1.2 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;
```

1.3 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;
```

2. 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.

2.1 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 the rows from each of the tables. Thus, if the customer table contained 3 entries and the product table contained 18 entries, the Cartesian product consists of 54 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.

2.2 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

2.3 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 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 c