UNIT-3:
STRUCTURED QUERY LANGUAGE

Data Definition, Constraints, and Schema Changes in SQL2

Structured Query Language (SQL) was designed and implemented at IBM Research. Created in late 70's, under the name of SEQUEL

A standard version of SQL (ANSI 1986), is called SQL86 or SQL1. A revised version of standard SQL, called SQL2 (or SQL92).

SQL are going to be extended with objectoriented and other recent database concepts. Consists of

A Data Definition Language (DDL) for declaring database schemas

Data Manipulation Language (DML) for modifying and querying database instances

In SQL, relation, tuple, and attribute are called *table*, *row*, and *columns* respectively. The SQL commands for data definition are *CREATE*, *ALTER*, and *DROP*.

The *CREATE TABLE* Command is used to specify a new table by giving it a name and specifying its attributes (columns) and constraints.

Data types available for attributes are:

- *Numeric* integer, real (formated, such as *DECIMAL(10,2)*)
- *CharacterString* fixedlength and varyinglength
- *BitString* fixedlength, varyinglength
- *Date* in the form YYYYMMDD
- *Time* in the form HH:MM:SS
- *Timestamp* includes both the *DATE* and *TIME* fields
- *Interval*to increase/decrease the value of date, time, or timestamp

4.2 Basic Queries in SQL

SQL allows a table (relation) to have two or more tuples that are identical in all their attributes values. Hence, an **SQL table** is not a set of tuple, because a set does not allow two identical members; rather it is a multiset of tuples.

A basic query statement in SQL is the *SELECT* statement.

The *SELECT* statement used in SQL has no relationship to the *SELECT* operation of relational algebra.

The SELECT Statement

The syntax of this command is:

SELECT <attribute list>
FROM <table list>
WHERE <Condition>;

Query 0: Retrieve the birthday and address of the employee(s) whose name is _John B. Smith'

Q0: SELECT BDATE, ADDRESS

FROMEMPLOYEE

WHERE FNAME = _John' AND MINIT =_B' ANDLNAME = _SMITH'

Query 1: Retrieve the name and address of all employee who work for the _Research' Dept.

Q1: SELECT FNAME, LNAME, ADDRESS

FROMEMPLOYEE, DEPARTMENT

WHERE DNAME = _Research' AND DNUMBER = DNO

Query2: For every project located in _Stafford', list the project number, the controlling departmentnumber, and the department manager's last name, address, and birthdate.

Q2: SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE

FROMPROJECT, DEPARTMENT,EMPLOYEE

WHERE DNUM=DNUMBER AND MGRSSN=SSN AND PLOCATION =
_Stafford'

Dealing with Ambiguous Attribute Names and Renaming (Aliening)

Ambiguity in the case where attributes are same name need to qualify the attribute using DOT separator

e.g., WHERE DEPARTMENT.DNUMBER=EMPLOYEE.DNUMBER

More Ambiguity in the case of queries that refer to the same relation twice

Query 8: For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor

Q8:

```
SELECT E.FNAME, E.LNAME, S.FNAME,S.LNAME
FROM   EMPLOYEE AS E, EMPLOYEEAS S
          WHERE E.SUPERSSN=S.SSN
```

Unspecified WHEREClause and Use of Asterisk (*)

A missing WHEREclause indicates no conditions, which means all tuples are selected

In case of two or more table, then all possible tuple combinations are selected

Example: Q10: Select all EMPLOYEE SSNs , and all combinations of EMPLOYEE SSN and DEPARTMENT DNAME

```
SELECT   SSN, DNAME

FROM      EMPLOYEE, DEPARTMENT
```

More

To retrieve all the attributes, use * in SELECT clause

Retrieve all employees working for Dept. 5

```
SELECT *

FROM    EMPLOYEE

WHERE DNO=5
```

Substring Comparisons, Arithmetic Operations, and Ordering

like, binary operator for comparing strings
%, wild card for strings
_, wild card for characters
||, concatenate operation for strings

(name like '%a_') is true for all names having _a' as second letter from the end.

Partial strings are specified by using '

```
SELECTFNAME, LNAME
FROMEMPLOYEE
        WHERE   FNAME LIKE '%Mc%';
```

In order to list all employee who were born during 1960s we have the followings:

```
           SELECT   FNAME, LNAME
.       FROM     EMPLOYEE
        WHERE   BDATE LIKE '6_______';
```

SQL also supports addition, subtraction, multiplication and division (denoted by +, , *, and /, respectively) on numeric values or attributes with numeric domains.

Examples: Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

```
SELECT  FNAME, LNAME, 1.1*SALARY
FROM    EMPLOYEE, WORKS_ON, PROJECT
WHERE   SSN=ESSN AND PNO=PNUMBER AND PNAME='ProductX';
```

Retrieve all employees in department number 5 whose salary between $30000 and $40000.

```
SELECT  *
FROM    EMPLOYEE
WHERE   (SALARY BETWEEN 30000 AND 40000) AND DNO=5;
```

It is possible to order the tuples in the result of a query.

```
SELECT  DNAME, LNAME, FNAME, PNAME
FROM    DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
WHERE   DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER
ORDER BY DNAME, LNAME, FNAME;
```

The default order is in ascending order, but user can specify

```
ORDER BY DNAME DESC, LNAME ASC, FNAME, ASC;
```

Tables as Sets in SQL

SQL treats table as a **multiset,** which means duplicate tuples are OK

SQL does not delete duplicate because Duplicate elimination is an expensive operation (sort and delete) user may be interested in the result of a query in case of aggregate function, we do not want to eliminate duplicates

To eliminate duplicate, use DISTINCT
examples

Q11: Retrieve the salary of every employee , and (Q!2) all distinct salary values

```
Q11: SELECT ALL SALARY
     FROM   EMPLOYEE

Q12: SELECT DISTINCT SALARY
     FROM   EMPLOYEE
```

More Complex SQL Queries

Complex SQL queries can be formulated by composing nested SELECT/FROM/WHERE clauses within the WHEREclause of another query

Example: Q4: Make a list of Project numbers for projects that involve an employee whose last name is _Smith', either as a worker or as a manger of the department that controls the project

```
Q4      SELECT DISTINCT PNUMBER
        FROM PROJECT
       WHERE PNUMBER IN (SELECT PNUMBER
                         FROM PROJECT, DEPARTMENT, EMPLOYEE
                         WHERE DNUM=DNUMBER AND MGRSSN=SSN AND
LNAME=_Smith'
 OR  PNUMBER IN (SELECT PNO
                 FROM WORKS_ON, EMPLOYEE
                 WHERE ESSN=SSN AND LNAME=_Smith')
```

IN operator and set of union compatible tuples

Example:

```
SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE (PNO, HOURS) IN (SELECT PNO, HOURS
                       FROM  WORKS_ON
                       WHERE SSN=_123456789'
```

ANY, SOME and >, <=,<>,etc.

The keyword ALL

In addition to the IN operator, a number of other comparison operators can be used to compare a single value v to a set of multiset V.

ALL V returns TRUE if v is greater than all the value in the set

Select the name of employees whose salary is greater than the salary of all the employees in department 5

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SALARY > ALL (SELECT SALARY
```

FROM EMPLOYEE WHERE DNO=5);

Ambiguity in nested query

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN (SELECT ESSN
                FROM DEPENDENT
                 WHERE  ESSN=E.SSN  AND  E.FNAM=DEPENDENT_NAME  AND
SEX=E.SEX
```

Correlated Nested Query

Whenever a condition in the WHEREclause of a nested query references some attributes of a relation declared in the outer query, the two queries are said to be correlated. The result of a correlated nested query is different for each tuple (or combination of tuples) of the relation(s) the outer query.

In general, any nested query involving the = or comparison operator IN can always be rewritten as a single block query

```
SELECT E.FNAME, E.LNAME

FROM EMPLOYEE E, DEPENDENT D

WHERE E.SSN=D.ESSN AND E.SEX=D.SEX AND E.FNAME =D.DEPENDENT=NAME
```

Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12:    SELECTE.FNAME, E.LNAME
        FROM    EMPLOYEE AS E
        WHERE   E.SSN IN (SELECT   ESSN
                          FROM     DEPENDENT
                          WHERE    ESSN=E.SSN AND
                          E.FNAME=DEPENDENT_NAME)
```

In Q12, the nested query has a different result for each tuple in the outer query.

The original SQL as specified for SYSTEM R also had a CONTAINS comparison operator, which is used in conjunction with nested correlated queries This operator was dropped from the language, possibly because of the difficulty in implementing it efficiently Most implementations of SQL do not have this operator The CONTAINS operator compares two sets of values , and returns TRUE if one set contains all values in the other set (reminiscent of the division operation of algebra).

Query 3: Retrieve the name of each employee who works on all the projects controlled by department number 5.

Q3: SELECT FNAME, LNAME

FROM EMPLOYEE WHERE ((SELECT PNO FROM WORKS_ON WHERE SSN=ESSN)

CONTAINS (SELECT PNUMBER FROM PROJECT WHERE DNUM=5))

In Q3, the second nested query, which is not correlated with the outer query, retrieves the project numbers of all projects controlled by department 5.

The first nested query, which is correlated, retrieves the project numbers on which the employee works, which is different for each employee tuple because of the correlation.

THE EXISTS AND UNIQUE FUNCTIONS IN SQL

EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not We can formulate Query 12 in an alternative form that uses EXISTS as Q12B below.

Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE E
WHERE EXISTS (SELECT *
                FROM DEPENDENT
                    WHERE E.SSN=ESSN AND SEX=E.SEX AND
E.FNAME=DEPENDENT_NAME
```

Query 6: Retrieve the names of employees who have no dependents.

```
Q6:     SELECTFNAME, LNAME
        FROM    EMPLOYEE
        WHERE   NOT EXISTS(SELECT       *
                        FROM    DEPENDENT
                        WHERE   SSN=ESSN)
```

In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If none exist , the EMPLOYEE tuple is selected EXISTS is necessary for the expressive power of SQL

EXPLICIT SETS AND NULLS IN SQL

It is also possible to use an explicit (enumerated) set of values in the WHEREclause rather than a nested query Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

Retrieve SSNs of all employees who work on project number 1,2,3

```
SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE PNO IN (1,2,3)
```

Null example

SQL allows queries that check if a value is NULL (missing or undefined or not applicable) SQL uses IS or IS NOT to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .

Retrieve the names of all employees who do not have supervisors

SELECT FNAME, LNAME FROM EMPLOYEE WHERE SUPERSSN IS NULL

Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

Join Revisit

Retrieve the name and address of every employee who works for _Search' department

SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
WHERE DNAME=_Search'

Aggregate Functions

Include COUNT, SUM, MAX, MIN, and AVG

Query 15: Find the sum of the salaries of all employees the _Research' dept, and the max salary, the min salary, and average:

SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY) AVG(SALARY)

FROM EMPLOYEE

WHERE DNO=FNUMBER AND DNAME=_RSEARCH'

Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q16:    SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
        FROM    EMPLOYEE, DEPARTMENT
        WHERE   DNO=DNUMBER AND DNAME='Research'
```

Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

```
Q17:    SELECTCOUNT (*)
        FROM    EMPLOYEE
Q18:    SELECTCOUNT (*)
        FROM    EMPLOYEE, DEPARTMENT
        WHERE   DNO=DNUMBER AND DNAME='Research'
```

Example of grouping

In many cases, we want to apply the aggregate functions to subgroups of tuples in a relation Each subgroup of tuples consists of the set of tuples that have the same value for the grouping attribute(s)

The function is applied to each subgroup independently

SQL has a GROUP BYclause for specifying the grouping attributes, which must also appear in the SELECTclause

For each project, select the project number, the project name, and the number of employees who work on that projet

```
SELECT PNUMBER, PNAME, COUNT(*)

FROM PROJECT, WORKS_ON

WHERE PNUMBER=PNO

GROUP BY PNUMBER, PNAME
```

In Q20, the EMPLOYEE tuples are divided into groupseach group having the same value for the grouping attribute DNO The COUNT and AVG functions are applied to each such group of tuples separately.The SELECTclause includes only the grouping attribute and the functions to be applied on each group of tuples. A join condition can be used in conjunction with grouping

Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q21:    SELECT   PNUMBER, PNAME, COUNT (*)
        FROM     PROJECT, WORKS_ON
        WHERE    PNUMBER=PNO
        GROUP BY          PNUMBER, PNAME
```

In this case, the grouping and functions are applied after the joining of the two relations

THE HAVINGCLAUSE:

Sometimes we want to retrieve the values of these functions for only those groups that satisfy certain conditions. The HAVINGclause is used for specifying a selection condition on groups (rather than on individual tuples)

Query 22: For each project on which more than two employees work , retrieve the project number, project name, and the number of employees who work on that project.

```
Q22:    SELECT   PNUMBER, PNAME, COUNT (*)
        FROM     PROJECT, WORKS_ON
        WHERE    PNUMBER=PNO
        GROUP BY          PNUMBER, PNAME
        HAVING   COUNT (*) > 2
```

Update Statements in SQL

The Insert Command

INSERT INTO EMPLOYEE

VALUES (_Richard‘,‘K‘,‘Marini‘,653298653‘,‘30dec52‘,98 Oak Forest, Katy, TX‘,‘M‘,37000,‘987654321‘,4)

More on Insert

Use explicit attribute names:

INSERT INTO EMPLOYEE (FNAME, LNAME,SSN)

VALUES (_Richard‘,‘Marini‘, _653298653‘

The DELECT Command

DELETE FROM EMPLOYEE

WHERE LNAME=_Brown‘

The UPDATE Command

Used to modify values of one or more selected tuples

Change the location and controlling department number of project number 10 to _Bellaire‘ and 5 respectively

UPDATE PROJECT

SET PLOCATION = _Bellaire‘, DNUM=5

Where PNUMBER=10;

Views in SQL

A view refers to a single table that is derived from other tables

CREATE VIEW WORKS_ON1
AS SELECT FNAME, LNAME, PNAME, HOURS
FROM EMPLOYEE, PROJECT, WORKS_ON WHERE SSN=ESSN AND PNO=PNUMBER

More on View

CREATE VIEW DEPT_INFO(DEPT_NAME, NO_OF_EMPLS, TOTAL_SAL)

AS SELECT DNAME, COUNT(*), SUM(SALARY)

FROM DEPARTMENT, EMPLOYEE

WHERE DNUMBER=DNO

GROUP BY DNAME

More on view

Treat WORKS_ON1 like a base table as follows

SELECT FNAME, LNAME

FROM WORKS_ON1

WHERE PNMAE=_PROJECTX'

Main advantage of view:

Simplify the specification of commonly used queries

More on View

A View is always up to date;

A view is realized at the time we specify(or execute) a query on the view

DROP VIEW WORKS_ON1

Updating of Views

Updating the views can be complicated and ambiguous

In general, an update on a view on defined on a single table w/o any aggregate functions can be mapped to an update on the base table
More on Views

We can make the following observations:

A view with a single defining table is updatable if we view contain PK or CK of the base

table View on multiple tables using joins are not updatable

View defined using grouping/aggregate are not updatable

Specifying General Constraints

Users can specify certain constraints such as semantics constraints

CREATE ASSERTION SALARY_CONSTRAINT

CHECK (NOT EXISTS (SELECT * FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D

WHERE E.SALARY > M. SALARY **AND** E.DNO=D.NUMBER **AND** D.MGRSSN=M.SSN))

5.3 Additional features

Granting and revoking privileges

Embedding SQL statements in a general purpose languages (C, C++, COBOL, PASCAL)

SQL can also be used in conjunction with a general purpose programming language, such as PASCAL, COBOL, or PL/I. The programming language is called the host language. The embedded SQL statement is distinguished from programming language statements by prefixing it with a special character or command so that a preprocessor can extract the SQL statements. In PL/I the keywords EXEC SQL precede any SQL statement. In some implementations, SQL statements are passed as parameters in procedure calls. We will use PASCAL as the host programming language, and a "$" sign to identify SQL statements in the program. Within an embedded SQL command, we may refer to program variables, which are prefixed by a "%" sign. The programmer should declare program variables to match the data types of the database attributes that the program will process.These program variables may or may not have names that are identical to their corresponding attributes.

Example: Write a program segment (loop) that reads a social security number and prints out some information from the corresponding EMPLOYEE tuple

```
E1:     LOOP:= 'Y';
        while LOOP = 'Y' do
                begin
                writeln('input social security number:');

                readln(SOC_SEC_NUM);
                $SELECT FNAME, MINIT, LNAME, SSN, BDATE,
        ADDRESS, SALARY
                  INTO %E.FNAME, %E.MINIT, %E.LNAME, %E.SSN,
                          %E.BDATE, %E.ADDRESS, %E.SALARY
                  FROM EMPLOYEE
                  WHERE SSN=%SOC_SEC_NUM ;
                writeln( E.FNAME, E.MINIT, E.LNAME,
        E.SSN, E.BDATE, E.ADDRESS, E.SALARY);
                writeln('more social security numbers (Y or N)?
                '); readln(LOOP)
                end;
```

In E1, a single tuple is selected by the embedded SQL query; that is why we are able to assign its attribute values directly to program variables. In general, an SQL query can retrieve many tuples. The concept of a cursor is used to allow tupleatatime processing by the PASCAL programCURSORS: We can think of a cursor as a pointer that points to a single tuple (row) from the result of a query.The cursor is declared when the SQL query command is specified. A subsequent OPEN cursor command fetches the query result and sets the cursor to a position before the first row in the result of the query; this becomes the current row for the cursor. Subsequent FETCH commands in the program advance the cursor to the next row and copy its attribute values into PASCAL program variables specified in the FETCH command. An implicit variable SQLCODE communicates to the program the status of SQL embedded commands. An SQLCODE of 0 (zero) indicates successful execution. Different codes are returned to indicate exceptions and errors. A special END_OF_CURSOR code is used to terminate a loop over the tuples in a query result. A CLOSE cursor command is issued to indicate that we are done with

the result of the query When a cursor is defined for rows that are to be updated the clause FOR UPDATE OF must be in the cursor declaration, and a list of the names of any attributes that will be updated follows.The condition WHERE CURRENT OF cursor specifies that the current tuple is the one to be updated (or deleted)

Example: Write a program segment that reads (inputs) a department name, then lists the names of employees who work in that department, one at a time. The program reads a raise amount for each employee and updates the employee's salary by that amount.

```
E2:     writeln('enter the department name:'); readln(DNAME);
        $SELECT DNUMBER INTO %DNUMBER
          FROM DEPARTMENT
          WHERE DNAME=%DNAME;
        $DECLARE EMP CURSOR FOR
                                SELECT SSN, FNAME, MINIT, LNAME, SALARY
                                FROM EMPLOYEE
                                WHERE DNO=%DNUMBER
          FOR UPDATE OF SALARY;
        $OPEN EMP;
        $FETCH EMP INTO %E.SSN, %E.FNAME, %E.MINIT,
                                                %E.LNAME, %E.SAL;

        while SQLCODE = 0
                do begin
writeln('employee name: ', E.FNAME, E.MINIT, E.LNAME);
writeln('enter raise amount: '); readln(RAISE);
$UPDATE EMPLOYEE SET SALARY = SALARY + %RAISE
        WHERE CURRENT OF EMP;
$FETCH EMP INTO %E.SSN, %E.FNAME, %E.MINIT,
                                                %E.LNAME, %E.SAL;
                end;
        $CLOSE CURSOR EMP;
```

SQL is a standard language for accessing and manipulating databases.

What is SQL?

1. SQL stands for Structured Query Language
2. SQL lets you access and manipulate databases
3. SQL is an ANSI (American National Standards Institute) standard

What Can SQL do?

1. SQL can execute queries against a database
2. SQL can retrieve data from a database
3. SQL can insert records in a database
4. SQL can update records in a database
5. SQL can delete records from a database 2i 39 h 0
6. SQL can create new databases
7. SQL can create new tables in a database
8. SQL can create stored procedures in a database
9. SQL can create views in a database
10. SQL can set permissions on tables, procedures, and views

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Below is an example of a table called "Persons":

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

The table above contains three records (one for each person) and five columns (P_Id, LastName, FirstName, Address, and City).

SQL Statements

Most of the actions you need to perform on a database are done with SQL statements.

The following SQL statement will select all the records in the "Persons" table:

```
SELECT * FROM Persons
```

In this tutorial we will teach you all about the different SQL statements.

Keep in Mind That...

2.2 SQL is not case sensitive

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

We are using MS Access and SQL Server 2000 and we do not have to put a semicolon after each SQL statement, but some database programs force you to use it.

SQL DML and DDL

SQL can be divided into two parts: The Data Manipulation Language (DML) and the Data Definition Language (DDL).

The query and update commands form the DML part of SQL:

2.8 **SELECT** - extracts data from a database
2.9 **UPDATE** - updates data in a database
2.10 **DELETE** - deletes data from a database
2.11 **INSERT INTO** - inserts new data into a database

The DDL part of SQL permits database tables to be created or deleted. It also define indexes (keys), specify links between tables, and impose constraints between tables. The most important DDL statements in SQL are:

3.1 **CREATE DATABASE** - creates a new database
3.2 **ALTER DATABASE** - modifies a database
3.3 **CREATE TABLE** - creates a new table
3.4 **ALTER TABLE** - modifies a table
3.5 **DROP TABLE** - deletes a table

The SQL SELECT Statement

The SELECT statement is used to select data from a database.

The result is stored in a result table, called the result-set.

SQL SELECT Syntax

```
SELECT column_name(s)
FROM table_name
```

and

```
SELECT * FROM table_name
```

Note: SQL is not case sensitive. SELECT is the same as select.

An SQL SELECT Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to select the content of the columns named "LastName" and "FirstName" from the table above.

We use the following SELECT statement:

```
SELECT LastName,FirstName FROM Persons
```

The result-set will look like this:

LastName	FirstName
Kumari	Mounitha
Kumar	Pranav
Gubbi	Sharan

SELECT * Example

Now we want to select all the columns from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
```

Tip: The asterisk (*) is a quick way of selecting all columns!

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

SQL SELECT DISTINCT Statement

This chapter will explain the SELECT DISTINCT statement.

The SQL SELECT DISTINCT Statement

In a table, some of the columns may contain duplicate values. This is not a problem, however, sometimes you will want to list only the different (distinct) values in a table.

The DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax

```
SELECT DISTINCT column_name(s)
FROM table_name
```

SELECT DISTINCT Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to select only the distinct values from the column named "City" from the table above.

We use the following SELECT statement:

```
SELECT DISTINCT City FROM Persons
```

The result-set will look like this:

City
Bangalore
Tumkur

The WHERE clause is used to filter records.

The WHERE Clause

The WHERE clause is used to extract only those records that fulfill a specified criterion.

SQL WHERE Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
```

WHERE Clause Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to select only the persons living in the city "Bangalore" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City='Bangalore'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore

Quotes Around Text Fields

SQL uses single quotes around text values (most database systems will also accept double quotes).

Although, numeric values should not be enclosed in quotes.
For text values:

```
This is correct:
SELECT * FROM Persons WHERE FirstName='Pranav'
This is wrong:
SELECT * FROM Persons WHERE FirstName=Pranav
```

For numeric values:

```
This is correct:
SELECT * FROM Persons WHERE Year=1965
This is wrong:
SELECT * FROM Persons WHERE Year='1965'
```

Operators Allowed in the WHERE Clause

With the WHERE clause, the following operators can be used:

Operator	Description
<>	Not equal
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

Note: In some versions of SQL the <> operator may be written as !

SQL AND & OR Operators

The AND & OR operators are used to filter records based on more than one condition.

The AND & OR Operators

The AND operator displays a record if both the first condition and the second condition is true.

The OR operator displays a record if either the first condition or the second condition is true.

AND Operator Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to select only the persons with the first name equal to "Pranav" AND the last name equal to "Kumar":

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName='Pranav'
AND LastName='Kumar'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Kumar	Pranav	Yelhanka	Bangalore

OR Operator Example

Now we want to select only the persons with the first name equal to "Pranav" OR the first name equal to "Mounitha":

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName='Pranav'
OR FirstName='Mounitha'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore

Combining AND & OR

You can also combine AND and OR (use parenthesis to form complex expressions).

Now we want to select only the persons with the last name equal to "Kumar" AND the first name equal to "Pranav" OR to "Mounitha":

We use the following SELECT statement:

```
SELECT * FROM Persons WHERE
LastName='Kumar'
AND (FirstName='Pranav' OR FirstName='Mounitha')
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Kumar	Pranav	Yelhanka	Bangalore

SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set.

The ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by a specified column.

The ORDER BY keyword sort the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax

```
SELECT column_name(s)
```

```
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

ORDER BY Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur
4	Nilsen	Tom	Vingvn 23	Tumkur

Now we want to select all the persons from the table above, however, we want to sort the persons by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
3	Gubbi	Sharan	Hebbal	Tumkur
2	Kumar	Pranav	Yelhanka	Bangalore
1	Kumari	Mounitha	VPura	Bangalore
4	Nilsen	Tom	Vingvn 23	Tumkur

ORDER BY DESC Example

Now we want to select all the persons from the table above, however, we want to sort the persons descending by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName DESC
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
4	Nilsen	Tom	Vingvn 23	Tumkur
3	Gubbi	Sharan	Hebbal	Tumkur
2	Kumar	Pranav	Yelhanka	Bangalore
1	Kumari	Mounitha	VPura	Bangalore

SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

The INSERT INTO Statement

The INSERT INTO statement is used to insert a new row in a table.

SQL INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

```
INSERT INTO table_name
VALUES (value1, value2, value3,...)
```

The second form specifies both the column names and the values to be inserted:

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

SQL INSERT INTO Example

We have the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to insert a new row in the "Persons" table.

We use the following SQL statement:

```
INSERT INTO Persons
VALUES (4,'Nilsen', 'Johan', 'Bakken 2', 'Tumkur')
```

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur
4	Nilsen	Johan	Bakken 2	Tumkur

Insert Data Only in Specified Columns

It is also possible to only add data in specific columns.

The following SQL statement will add a new row, but only add data in the "P_Id", "LastName" and the "FirstName" columns:

```
INSERT INTO Persons (P_Id, LastName, FirstName)
VALUES (5, 'Tjessem', 'Jakob')
```

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur
4	Nilsen	Johan	Bakken 2	Tumkur
5	Tjessem	Jakob		

SQL UPDATE Statement

The UPDATE statement is used to update records in a table.

The UPDATE Statement

The UPDATE statement is used to update existing records in a table.

SQL UPDATE Syntax

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

SQL UPDATE Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur
4	Nilsen	Johan	Bakken 2	Tumkur
5	Tjessem	Jakob		

Now we want to update the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

```
UPDATE Persons
SET Address='Nissestien 67', City='Bangalore'
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur
4	Nilsen	Johan	Bakken 2	Tumkur
5	Tjessem	Jakob	Nissestien 67	Bangalore

SQL UPDATE Warning

Be careful when updating records. If we had omitted the WHERE clause in the example above, like this:

```
UPDATE Persons
SET Address='Nissestien 67', City='Bangalore'
```

The "Persons" table would have looked like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	Nissestien 67	Bangalore
2	Kumar	Pranav	Nissestien 67	Bangalore
3	Gubbi	Sharan	Nissestien 67	Bangalore
4	Nilsen	Johan	Nissestien 67	Bangalore
5	Tjessem	Jakob	Nissestien 67	Bangalore

SQL DELETE Statement

The DELETE statement is used to delete records in a table.

The DELETE Statement

The DELETE statement is used to delete rows in a table.

SQL DELETE Syntax

```
DELETE FROM table_name
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

SQL DELETE Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur
4	Nilsen	Johan	Bakken 2	Tumkur
5	Tjessem	Jakob	Nissestien 67	Bangalore

Now we want to delete the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

```
DELETE FROM Persons
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur
4	Nilsen	Johan	Bakken 2	Tumkur

Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM
table_name or
DELETE * FROM table_name
```

Note: Be very careful when deleting records. You cannot undo this statement!

SQL TOP Clause

The TOP Clause

The TOP clause is used to specify the number of records to return.

The TOP clause can be very useful on large tables with thousands of records. Returning a large number of records can impact on performance.

Note: Not all database systems support the TOP clause.

SQL Server Syntax

```
SELECT TOP number|percent column_name(s)
FROM table_name
```

SQL SELECT TOP Equivalent in MySQL and Oracle

MySQL Syntax

```
SELECT column_name(s)
FROM table_name
LIMIT number
```

Oracle Syntax

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number
```

SQL TOP Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur
4	Nilsen	Tom	Vingvn 23	Tumkur

Now we want to select only the two first records in the table above.

We use the following SELECT statement:

```
SELECT TOP 2 * FROM Persons
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore

SQL TOP PERCENT Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur
4	Nilsen	Tom	Vingvn 23	Tumkur

Now we want to select only 50% of the records in the table above.

We use the following SELECT statement:

```
SELECT TOP 50 PERCENT * FROM Persons
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore

SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

The LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.

SQL LIKE Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

LIKE Operator Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to select the persons living in a city that starts with "B" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE 'B%'
```

The "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore

Next, we want to select the persons living in a city that ends with an "r" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE '%r'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
3	Gubbi	Sharan	Hebbal	Tumkur

Next, we want to select the persons living in a city that contains the pattern "mk" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE '%mk%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
3	Gubbi	Sharan	Hebbal	Tumkur

It is also possible to select the persons living in a city that NOT contains the pattern "mk" from the "Persons" table, by using the NOT keyword.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City NOT LIKE '%mk%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore

SQL Wildcards

SQL wildcards can be used when searching for data in a database.

SQL Wildcards

SQL wildcards can substitute for one or more characters when searching for data in a database.

SQL wildcards must be used with the SQL LIKE operator.

With SQL, the following wildcards can be used:

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for exactly one character
[charlist]	Any single character in charlist
[^charlist] or [!charlist]	Any single character not in charlist

SQL Wildcard Examples

We have the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Using the % Wildcard

Now we want to select the persons living in a city that starts with "sa" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE 'Ba%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore

Using the _ Wildcard

Now we want to select the persons with a first name that starts with any character, followed by "ri" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
```

```
WHERE FirstName LIKE '_ri'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore

Next, we want to select the persons with a last name that starts with "P", followed by any character, followed by "an", followed by any character, followed by "v" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName LIKE 'P_an_v'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Kumar	Pranav	Yelhanka	Bangalore

Using the [charlist] Wildcard

Now we want to select the persons with a first name that starts with "b" or "s" or "p" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName LIKE '[bsp]%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Next, we want to select the persons with a last name that do not start with "b" or "s" or "p" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName LIKE '[!bsp]%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore

SQL IN Operator

The IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

SQL IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

IN Operator Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to select the persons with a last name equal to "Kumari" or "Gubbi" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName IN ('Kumari','Gubbi')
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

SQL BETWEEN Operator

The BETWEEN operator is used in a WHERE clause to select a range of data between two values.

The BETWEEN Operator

The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates.

SQL BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

BETWEEN Operator Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to select the persons with a last name alphabetically between "Kumari" and "Gubbi" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName
BETWEEN 'Kumari' AND 'Gubbi'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Note: The BETWEEN operator is treated differently in different databases.

In some databases a person with the LastName of "Kumari" or "Gubbi" will not be listed (BETWEEN only selects fields that are between and excluding the test values).

In other databases a person with the last name of "Kumari" or "Gubbi" will be listed (BETWEEN selects fields that are between and including the test values).

And in other databases a person with the last name of "Kumari" will be listed, but "Gubbi" will not be listed (BETWEEN selects fields between the test values, including the first test value and excluding the last test value).

Therefore: Check how your database treats the BETWEEN operator.

Example 2

To display the persons outside the range in the previous example, use NOT BETWEEN:

```
SELECT * FROM Persons
WHERE LastName
```

```
NOT BETWEEN 'Kumari' AND 'Gubbi'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Kumar	Pranav	Yelhanka	Bangalore

SQL Alias

With SQL, an alias name can be given to a table or to a column.

SQL Alias

You can give a table or a column another name by using an alias. This can be a good thing to do if you have very long or complex table names or column names.

An alias name could be anything, but usually it is short.

SQL Alias Syntax for Tables

```
SELECT column_name(s)
FROM table_name
AS alias_name
```

SQL Alias Syntax for Columns

```
SELECT column_name AS alias_name
FROM table_name
```

Alias Example

Assume we have a table called "Persons" and another table called "Product_Orders". We will give the table aliases of "p" an "po" respectively.

Now we want to list all the orders that "Mounitha Kumari" is responsible for.

We use the following SELECT statement:

```
SELECT po.OrderID, p.LastName, p.FirstName
FROM Persons AS p,
Product_Orders AS po
WHERE p.LastName='Kumari'
WHERE p.FirstName='Mounitha'
```

The same SELECT statement without aliases:

```
SELECT Product_Orders.OrderID, Persons.LastName, Persons.FirstName
FROM Persons,
Product_Orders
WHERE Persons.LastName='Kumari'
```

```
WHERE Persons.FirstName='Mounitha'
```

As you'll see from the two SELECT statements above; aliases can make queries easier to both write and to read.

SQL Joins

SQL joins are used to query data from two or more tables, based on a relationship between certain columns in these tables.

SQL JOIN

The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables.

Tables in a database are often related to each other with keys.

A primary key is a column (or a combination of columns) with a unique value for each row. Each primary key value must be unique within the table. The purpose is to bind data together, across tables, without repeating all of the data in every table.

Look at the "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Note that the "P_Id" column is the primary key in the "Persons" table. This means that **no** two rows can have the same P_Id. The P_Id distinguishes two persons even if they have the same name.

Next, we have the "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Note that the "O_Id" column is the primary key in the "Orders" table and that the "P_Id" column refers to the persons in the "Persons" table without using their names.

Notice that the relationship between the two tables above is the "P_Id" column.

Different SQL JOINs

Before we continue with examples, we will list the types of JOIN you can use, and the differences between them.

3.6 **JOIN**: Return rows when there is at least one match in both tables

3.7 **LEFT JOIN**: Return all rows from the left table, even if there are no matches in the right table

3.8 **RIGHT JOIN**: Return all rows from the right table, even if there are no matches in the left table

3.9 **FULL JOIN**: Return rows when there is a match in one of the tables

SQL INNER JOIN Keyword

SQL INNER JOIN Keyword

The INNER JOIN keyword return rows when there is at least one match in both tables.

SQL INNER JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
INNER JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

PS: INNER JOIN is the same as JOIN.

SQL INNER JOIN Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the persons with any orders.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
INNER JOIN Orders
```

```
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	FirstName	OrderNo
Kumari	Mounitha	22456
Kumari	Mounitha	24562
Gubbi	Sharan	77895
Gubbi	Sharan	44678

The INNER JOIN keyword return rows when there is at least one match in both tables. If there are rows in "Persons" that do not have matches in "Orders", those rows will NOT be listed.

SQL LEFT JOIN Keyword

SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).

SQL LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
LEFT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

PS: In some databases LEFT JOIN is called LEFT OUTER JOIN.

SQL LEFT JOIN Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the persons and their orders - if any, from the tables above.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
LEFT JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	FirstName	OrderNo
Kumari	Mounitha	22456
Kumari	Mounitha	24562
Gubbi	Sharan	77895
Gubbi	Sharan	44678
Kumar	Pranav	

The LEFT JOIN keyword returns all the rows from the left table (Persons), even if there are no matches in the right table (Orders).

SQL RIGHT JOIN Keyword

SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword Return all rows from the right table (table_name2), even if there are no matches in the left table (table_name1).

SQL RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
RIGHT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

PS: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

SQL RIGHT JOIN Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3

2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the orders with containing persons - if any, from the tables above.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
RIGHT JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	FirstName	OrderNo
Kumari	Mounitha	22456
Kumari	Mounitha	24562
Gubbi	Sharan	77895
Gubbi	Sharan	44678
		34764

The RIGHT JOIN keyword returns all the rows from the right table (Orders), even if there are no matches in the left table (Persons

SQL FULL JOIN Keyword

SQL FULL JOIN Keyword

The FULL JOIN keyword return rows when there is a match in one of the tables.

SQL FULL JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
FULL JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

SQL FULL JOIN Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the persons and their orders, and all the orders with their persons.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
FULL JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	FirstName	OrderNo
Kumari	Mounitha	22456
Kumari	Mounitha	24562
Gubbi	Sharan	77895
Gubbi	Sharan	44678
Kumar	Pranav	
		34764

The FULL JOIN keyword returns all the rows from the left table (Persons), and all the rows from the right table (Orders). If there are rows in "Persons" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Persons", those rows will be listed as well.

SQL UNION Operator

The SQL UNION operator combines two or more SELECT statements.

The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

SQL UNION Syntax

```
SELECT column_name(s) FROM table_name1
UNION
SELECT column_name(s) FROM table_name2
```

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL.

SQL UNION ALL Syntax

```
SELECT column_name(s) FROM table_name1
UNION ALL
SELECT column_name(s) FROM table_name2
```

PS: The column names in the result-set of a UNION are always equal to the column names in the first SELECT statement in the UNION.

SQL UNION Example

Look at the following tables:

"Employees_India":

E_ID	E_Name
01	Kumari, Mounitha
02	Kumar, Pranav
03	Kumar, Stephen
04	Gubbi, Sharan

"Employees_USA":

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	Kumar, Stephen
04	Scott, Stephen

Now we want to list **all the different** employees in Norway and USA.

We use the following SELECT statement:

```
SELECT E_Name FROM Employees_India
UNION
SELECT E_Name FROM Employees_USA
```

The result-set will look like this:

E_Name
Kumari, Mounitha
Kumar, Pranav
Kumar, Stephen
Gubbi, Sharan
Turner, Sally
Kent, Clark
Scott, Stephen

Note: This command cannot be used to list all employees in India and USA. In the example above we have two employees with equal names, and only one of them will be listed. The UNION command selects only distinct values.

SQL UNION ALL Example

Now we want to list **all** employees in India and USA:

```
SELECT E_Name FROM Employees_India
UNION ALL
SELECT E_Name FROM Employees_USA
```

Result

E_Name
Kumari, Mounitha
Kumar, Pranav
Kumar, Stephen
Gubbi, Sharan
Turner, Sally
Kent, Clark
Kumar, Stephen
Scott, Stephen

SQL SELECT INTO Statement

The SQL SELECT INTO statement can be used to create backup copies of tables.

The SQL SELECT INTO Statement

The SELECT INTO statement selects data from one table and inserts it into a different

table. The SELECT INTO statement is most often used to create backup copies of tables.

SQL SELECT INTO Syntax

We can select all columns into the new table:

```
SELECT *
INTO new_table_name [IN externaldatabase]
FROM old_tablename
```

Or we can select only the columns we want into the new table:

```
SELECT column_name(s)
```

```
INTO new_table_name [IN externaldatabase]
FROM old_tablename
```

SQL SELECT INTO Example

Make a Backup Copy - Now we want to make an exact copy of the data in our "Persons"

table. We use the following SQL statement:

```
SELECT *
INTO Persons_Backup
FROM Persons
```

We can also use the IN clause to copy the table into another database:

```
SELECT *
INTO Persons_Backup IN 'Backup.mdb'
FROM Persons
```

We can also copy only a few fields into the new table:

```
SELECT LastName,FirstName
INTO Persons_Backup
FROM Persons
```

SQL SELECT INTO - With a WHERE Clause

We can also add a WHERE clause.

The following SQL statement creates a "Persons_Backup" table with only the persons who lives in the city "Bangalore":

```
SELECT LastName,Firstname
INTO Persons_Backup
FROM Persons
WHERE City='Bangalore'
```

SQL SELECT INTO - Joined Tables

Selecting data from more than one table is also possible.

The following example creates a "Persons_Order_Backup" table contains data from the two tables "Persons" and "Orders":

```
SELECT Persons.LastName,Orders.OrderNo
INTO Persons_Order_Backup
FROM Persons
INNER JOIN Orders
ON Persons.P_Id=Orders.P_Id
```

SQL CREATE DATABASE Statement

The CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a database.

SQL CREATE DATABASE Syntax

```
CREATE DATABASE database_name
```

CREATE DATABASE Example

Now we want to create a database called "my_db".

We use the following CREATE DATABASE statement:

```
CREATE DATABASE my_db
```

Database tables can be added with the CREATE TABLE statement.

SQL CREATE TABLE Statement

The CREATE TABLE Statement

The CREATE TABLE statement is used to create a table in a database.

SQL CREATE TABLE Syntax

```
CREATE TABLE table_name
(
column_name1   data_type,
column_name2   data_type,
column_name3 data_type,
....
)
```

The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server, go to our complete Data Types reference.

CREATE TABLE Example

Now we want to create a table called "Persons" that contains five columns: P_Id, LastName, FirstName, Address, and City.

We use the following CREATE TABLE statement:

```
CREATE TABLE Persons
```

```
(
P_Id int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

The P_Id column is of type int and will hold a number. The LastName, FirstName, Address, and City columns are of type varchar with a maximum length of 255 characters.

The empty "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City

The empty table can be filled with data with the INSERT INTO statement

SQL Constraints

SQL Constraints

Constraints are used to limit the type of data that can go into a table.

Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

We will focus on the following constraints:

4.1 NOT NULL
4.2 UNIQUE
4.3 PRIMARY KEY
4.4 FOREIGN KEY
4.5 CHECK
4.6 DEFAULT

The next chapters will describe each constraint in details.

SQL NOT NULL Constraint

By default, a table column can hold NULL values.

SQL NOT NULL Constraint

The NOT NULL constraint enforces a column to NOT accept NULL values.

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL enforces the "P_Id" column and the "LastName" column to not accept NULL values:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

SQL UNIQUE Constraint

SQL UNIQUE Constraint

The UNIQUE constraint uniquely identifies each record in a database table.

The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

Note that you can have have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "P_Id" column when the "Persons" table is created:

SQL PRIMARY KEY Constraint

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database

table. Primary keys must contain unique values.

A primary key column cannot contain NULL values.

Each table should have a primary key, and each table can have only one primary key.

SQL PRIMARY KEY Constraint on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "P_Id" column when the "Persons" table is created:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
PRIMARY KEY (P_Id)
)
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

To DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT pk_PersonID
```

SQL FOREIGN KEY Constraint

SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Let's illustrate the foreign key with an example. Look at the following two tables:

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3

3	22456	2
4	24562	1

Note that the "P_Id" column in the "Orders" table points to the "P_Id" column in the "Persons" table.

The "P_Id" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "P_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy link between tables.

The FOREIGN KEY constraint also prevents that invalid data is inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

SQL FOREIGN KEY Constraint on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "P_Id" column when the "Orders" table is created:

MySQL:

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)
)
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on

To DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY fk_PerOrders
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
DROP CONSTRAINT fk_PerOrders
```

The TRUNCATE TABLE Statement

What if we only want to delete the data inside the table, and not the table itself?

Then, use the TRUNCATE TABLE statement:

```
TRUNCATE TABLE table_name
```

SQL ALTER TABLE Statement

The ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

SQL ALTER TABLE Syntax

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype
```

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name
DROP COLUMN column_name
```

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype
```

SQL ALTER TABLE Example

Look at the "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to add a column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons
ADD DateOfBirth date
```

Notice that the new column, "DateOfBirth", is of type date and is going to hold a date. The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server, go to our complete Data Types reference.

The "Persons" table will now like this:

P_Id	LastName	FirstName	Address	City	DateOfBirth
1	Kumari	Mounitha	VPura	Bangalore	
2	Kumar	Pranav	Yelhanka	Bangalore	
3	Gubbi	Sharan	Hebbal	Tumkur	

Change Data Type Example

Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons
ALTER COLUMN DateOfBirth year
```

Notice that the "DateOfBirth" column is now of type year and is going to hold a year in a two-digit or four-digit format.

DROP COLUMN Example

Next, we want to delete the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons
DROP COLUMN DateOfBirth
```

The "Persons" table will now like this:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

SQL Views

A view is a virtual table.

This chapter shows how to create, update, and delete a view.

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

SQL CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

SQL CREATE VIEW Examples

If you have the Northwind database you can see that it has several views installed by default.

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table. The view is created with the following SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName
FROM Products
WHERE Discontinued=No
```

We can query the view above as follows:

```
SELECT * FROM [Current Product List]
```

Another view in the Northwind sample database selects every product in the "Products" table with a unit price higher than the average unit price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName,UnitPrice
FROM Products
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)
```

We can query the view above as follows:

```
SELECT * FROM [Products Above Average Price]
```

Another view in the Northwind database calculates the total sale for each category in 1997. Note that this view selects its data from another view called "Product Sales for 1997":

```
CREATE VIEW [Category Sales For 1997] AS
SELECT DISTINCT CategoryName,Sum(ProductSales) AS
CategorySales FROM [Product Sales for 1997]
GROUP BY CategoryName
```

We can query the view above as follows:

```
SELECT * FROM [Category Sales For 1997]
```

We can also add a condition to the query. Now we want to see the total sale only for the category "Beverages":

```
SELECT * FROM [Category Sales For 1997]
WHERE CategoryName='Beverages'
```

SQL Updating a View

You can update a view by using the following syntax:

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Now we want to add the "Category" column to the "Current Product List" view. We will update the view with the following SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName,Category
FROM Products
WHERE Discontinued=No
```

SQL Dropping a View

You can delete a view with the DROP VIEW command.

SQL DROP VIEW Syntax

```
DROP VIEW view_name
```

SQL Date Functions

SQL Dates

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets complicated.

Before talking about the complications of querying for dates, we will look at the most important built-in functions for working with dates.

SQL has many built-in functions for performing calculations on data.

SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

5.1 AVG() - Returns the average value
5.2 COUNT() - Returns the number of rows
5.3 FIRST() - Returns the first value
5.4 LAST() - Returns the last value
5.5 MAX() - Returns the largest value
5.6 MIN() - Returns the smallest value
5.7 SUM() - Returns the sum

SQL Scalar functions

SQL scalar functions return a single value, based on the input value.

Useful scalar functions:

6.1 UCASE() - Converts a field to upper case
6.2 LCASE() - Converts a field to lower case
6.3 MID() - Extract characters from a text field
6.4 LEN() - Returns the length of a text field
6.5 ROUND() - Rounds a numeric field to the number of decimals specified
6.6 NOW() - Returns the current system date and time
6.7 FORMAT() - Formats how a field is to be displayed

Tip: The aggregate functions and the scalar functions will be explained in details in the next chapters.

SQL AVG() Function

The AVG() Function

The AVG() function returns the average value of a numeric column.

SQL AVG() Syntax

```
SELECT AVG(column_name) FROM table_name
```

SQL AVG() Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Kumari

2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Kumari
4	2008/09/03	300	Kumari
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find the average value of the "OrderPrice" fields.

We use the following SQL statement:

```
SELECT AVG(OrderPrice) AS OrderAverage FROM Orders
```

The result-set will look like this:

OrderAverage
950

Now we want to find the customers that have an OrderPrice value higher than the average OrderPrice value.

We use the following SQL statement:

```
SELECT Customer FROM Orders
WHERE OrderPrice>(SELECT AVG(OrderPrice) FROM Orders)
```

The result-set will look like this:

Customer
Kumari
Nilsen
Jensen

SQL COUNT() Function

The COUNT() function returns the number of rows that matches a specified criteria.

SQL COUNT(column_name) Syntax

The COUNT(column_name) function returns the number of values (NULL values will not be counted) of the specified column:

```
SELECT COUNT(column_name) FROM table_name
```

SQL COUNT(*) Syntax

The COUNT(*) function returns the number of records in a table:

```
SELECT COUNT(*) FROM table_name
```

SQL COUNT(DISTINCT column_name) Syntax

The COUNT(DISTINCT column_name) function returns the number of distinct values of the specified column:

```
SELECT COUNT(DISTINCT column_name) FROM table_name
```

Note: COUNT(DISTINCT) works with ORACLE and Microsoft SQL Server, but not with Microsoft Access.

SQL COUNT(column_name) Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Kumari
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Kumari
4	2008/09/03	300	Kumari
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to count the number of orders from "Customer Nilsen".

We use the following SQL statement:

```
SELECT COUNT(Customer) AS CustomerNilsen FROM Orders
WHERE Customer='Nilsen'
```

The result of the SQL statement above will be 2, because the customer Nilsen has made 2 orders in total:

CustomerNilsen
2

SQL COUNT(*) Example

If we omit the WHERE clause, like this:

```
SELECT COUNT(*) AS NumberOfOrders FROM Orders
```

The result-set will look like this:

NumberOfOrders
6

which is the total number of rows in the table.

SQL COUNT(DISTINCT column_name) Example

Now we want to count the number of unique customers in the "Orders" table.

We use the following SQL statement:

```
SELECT COUNT(DISTINCT Customer) AS NumberOfCustomers FROM Orders
```

The result-set will look like this:

NumberOfCustomers
3

which is the number of unique customers (Kumari, Nilsen, and Jensen) in the "Orders" table.

SQL MAX() Function

The MAX() Function

The MAX() function returns the largest value of the selected column.

SQL MAX() Syntax

```
SELECT MAX(column_name) FROM table_name
```

SQL MAX() Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Kumari
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Kumari
4	2008/09/03	300	Kumari
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find the largest value of the "OrderPrice" column.

We use the following SQL statement:

```
SELECT MAX(OrderPrice) AS LargestOrderPrice FROM Orders
```

The result-set will look like this:

LargestOrderPrice
2000

SQL MIN() Function

The MIN() Function

The MIN() function returns the smallest value of the selected column.

SQL MIN() Syntax

```
SELECT MIN(column_name) FROM table_name
```

SQL MIN() Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Kumari
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Kumari
4	2008/09/03	300	Kumari
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find the smallest value of the "OrderPrice" column.

We use the following SQL statement:

```
SELECT MIN(OrderPrice) AS SmallestOrderPrice FROM Orders
```

The result-set will look like this:

SmallestOrderPrice
100

SQL SUM() Function

The SUM() Function

The SUM() function returns the total sum of a numeric column.

SQL SUM() Syntax

```
SELECT SUM(column_name) FROM table_name
```

SQL SUM() Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Kumari
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Kumari
4	2008/09/03	300	Kumari
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find the sum of all "OrderPrice" fields".

We use the following SQL statement:

```
SELECT SUM(OrderPrice) AS OrderTotal FROM Orders
```

The result-set will look like this:

OrderTotal
5700

SQL GROUP BY Statement

Aggregate functions often need an added GROUP BY statement.

The GROUP BY Statement

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

SQL GROUP BY Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

SQL GROUP BY Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Kumari
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Kumari

4	2008/09/03	300	Kumari
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find the total sum (total order) of each customer.

We will have to use the GROUP BY statement to group the customers.

We use the following SQL statement:

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
```

The result-set will look like this:

Customer	SUM(OrderPrice)
Kumari	2000
Nilsen	1700
Jensen	2000

Let's see what happens if we omit the GROUP BY statement:

```
SELECT Customer,SUM(OrderPrice) FROM Orders
```

The result-set will look like this:

Customer	SUM(OrderPrice)
Kumari	5700
Nilsen	5700
Kumari	5700
Kumari	5700
Jensen	5700
Nilsen	5700

The result-set above is not what we wanted.

Explanation of why the above SELECT statement cannot be used: The SELECT statement above has two columns specified (Customer and SUM(OrderPrice). The "SUM(OrderPrice)" returns a single value (that is the total sum of the "OrderPrice" column), while "Customer" returns 6 values (one value for each row in the "Orders" table). This will therefore not give us the correct result. However, you have seen that the GROUP BY statement solves this problem.

GROUP BY More Than One Column

We can also use the GROUP BY statement on more than one column, like this:

```
SELECT Customer,OrderDate,SUM(OrderPrice) FROM Orders
GROUP BY Customer,OrderDate
```

SQL HAVING Clause

The HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

SQL HAVING Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value
```

SQL HAVING Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Kumari
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Kumari
4	2008/09/03	300	Kumari
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find if any of the customers have a total order of less than 2000.

We use the following SQL statement:

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice)<2000
```

The result-set will look like this:

Customer	SUM(OrderPrice)
Nilsen	1700

Now we want to find if the customers "Kumari" or "Jensen" have a total order of more than 1500.

We add an ordinary WHERE clause to the SQL statement:

```
SELECT Customer,SUM(OrderPrice) FROM Orders
WHERE Customer='Kumari' OR Customer='Jensen'
GROUP BY Customer
HAVING SUM(OrderPrice)>1500
```

The result-set will look like this:

Customer	SUM(OrderPrice)
Kumari	2000
Jensen	2000

SQL UCASE() Function

The UCASE() Function

The UCASE() function converts the value of a field to uppercase.

SQL UCASE() Syntax

```
SELECT UCASE(column_name) FROM table_name
```

SQL UCASE() Example

We have the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to select the content of the "LastName" and "FirstName" columns above, and convert the "LastName" column to uppercase.

We use the following SELECT statement:

```
SELECT UCASE(LastName) as LastName,FirstName FROM Persons
```

The result-set will look like this:

LastName	FirstName
KUMARI	Mounitha
KUMAR	Pranav
GUBBI	Sharan

SQL LCASE() Function

The LCASE() Function

The LCASE() function converts the value of a field to lowercase.

SQL LCASE() Syntax

```
SELECT LCASE(column_name) FROM table_name
```

SQL LCASE() Example

We have the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to select the content of the "LastName" and "FirstName" columns above, and convert the "LastName" column to lowercase.

We use the following SELECT statement:

```
SELECT LCASE(LastName) as LastName,FirstName FROM Persons
```

The result-set will look like this:

LastName	FirstName
Kumari	Mounitha
Kumar	Pranav
Gubbi	Sharan

SQL MID() Function

The MID() Function

The MID() function is used to extract characters from a text field.

SQL MID() Syntax

```
SELECT MID(column_name,start[,length]) FROM table_name
```

Parameter	Description
column_name	Required. The field to extract characters from.
start	Required. Specifies the starting position (starts at 1).
length	Optional. The number of characters to return. If omitted, the MID() function returns the rest of the text.

SQL MID() Example

We have the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

We use the following SELECT statement:

```
SELECT MID(City,1,4) as SmallCity FROM Persons
```

The result-set will look like this:

SmallCity
Bang
Bang
Tumk

SQL LEN() Function

The LEN() Function

The LEN() function returns the length of the value in a text field.

SQL LEN() Syntax

```
SELECT LEN(column_name) FROM table_name
```

SQL LEN() Example

We have the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Kumari	Mounitha	VPura	Bangalore
2	Kumar	Pranav	Yelhanka	Bangalore
3	Gubbi	Sharan	Hebbal	Tumkur

Now we want to select the length of the values in the "Address" column above.

We use the following SELECT statement:

```
SELECT LEN(Address) as LengthOfAddress FROM Persons
```

The result-set will look like this:

LengthOfAddress
5
8
6

SQL ROUND() Function

The ROUND() Function

The ROUND() function is used to round a numeric field to the number of decimals specified.

SQL ROUND() Syntax

```
SELECT ROUND(column_name,decimals) FROM table_name
```

Parameter	Description
column_name	Required. The field to round.
decimals	Required. Specifies the number of decimals to be returned.

SQL ROUND() Example

We have the following "Products" table:

Prod_Id	ProductName	Unit	UnitPrice
1	Jarlsberg	1000 g	10.45
2	Mascarpone	1000 g	32.56
3	GorgonzMounitha	1000 g	15.67

Now we want to display the product name and the price rounded to the nearest integer.

We use the following SELECT statement:

```
SELECT ProductName, ROUND(UnitPrice,0) as UnitPrice FROM Persons
```

The result-set will look like this:

ProductName	UnitPrice
Jarlsberg	10
Mascarpone	33
GorgonzMounitha	16

SQL NOW() Function

The NOW() Function

The NOW() function returns the current system date and time.

SQL NOW() Syntax

```
SELECT NOW() FROM table_name
```

SQL NOW() Example

We have the following "Products" table:

Prod_Id	ProductName	Unit	UnitPrice
1	Jarlsberg	1000 g	10.45
2	Mascarpone	1000 g	32.56
3	GorgonzMounitha	1000 g	15.67

Now we want to display the products and prices per today's date.
We use the following SELECT statement:

```
SELECT ProductName, UnitPrice, Now() as PerDate FROM Persons
```

The result-set will look like this:

ProductName	UnitPrice	PerDate
Jarlsberg	10.45	30/09/2012
Mascarpone	32.56	30/09/2012
GorgonzMounitha	15.67	30/09/2012

SQL FORMAT() Function

The FORMAT() Function

The FORMAT() function is used to format how a field is to be displayed.

SQL FORMAT() Syntax

```
SELECT FORMAT(column_name,format) FROM table_name
```

Parameter	Description
column_name	Required. The field to be formatted.
format	Required. Specifies the format.

SQL FORMAT() Example

We have the following "Products" table:

Prod_Id	ProductName	Unit	UnitPrice
1	Jarlsberg	1000 g	10.45
2	Mascarpone	1000 g	32.56
3	GorgonzMounitha	1000 g	15.67

Now we want to display the products and prices per today's date (with today's date displayed in the following format "YYYY-MM-DD").

We use the following SELECT statement:

```
SELECT ProductName, UnitPrice, FORMAT(Now(),'YYYY-MM-DD') as PerDate
FROM
```