Structure Query Language

(SQL)

What is SQL? How does it work? How is it being used…?

What you will learn?

- Overview of DBMS/RDMBS/MS SQL 2008
- Database Objects
- Data Manipulation and Data Retrieval
- Grouping and Summarizing Data
- Built-in Functions
- Modifying Table Structures
- Working with Sub-queries
- Joining Multiple Tables

Overview of DBMS/RDMBS/MS SQL 2008

What is a database(DBMS)?

- ✓ DBMS means Data Base Management System
- ✓ Database is a structured collection of data.
- ✓ Database stores records relating to each other in a table
- ✓ Tables are uniquely identified by their names and consists of rows and columns
- ✓ Columns contain the column name, data type and any other attributes for the column
- ✓ Rows contain the records or data for the columns
- ✓ Visualize table as a tabular arrangement of data – vertical columns and horizontal rows
- ✓ Database Applications:
  - ✓ Banking: all transactions
  - ✓ Airlines: reservations, schedules
  - ✓ Universities: registration, grades
  - ✓ Sales: customers, products, purchases
  - ✓ Manufacturing: production, inventory, orders, supply chain
  - ✓ Human resources: employee records, salaries, tax deductions
- ✓ Databases touch all aspects of our lives

What is a relational database?

- ✓ Most common data management scheme
- ✓ Data is organized into two-dimensional tables of rows and columns
  - ✓ Data is decomposed into its simplest form
  - ✓ Normalization reduces data inconsistency
  - ✓ Referential Integrity

Attributes

- Example of tabular data in the relational model

Customer-id	*customer-name*	*customer-street*	*customer-city*	*account-number*
192-83-7465	Johnson	Alma	Palo Alto	A-101
019-28-3746	Smith	North	Rye	A-215
192-83-7465	Johnson	Alma	Palo Alto	A-201
321-12-3123	Jones	Main	Harrison	A-217
019-28-3746	Smith	North	Rye	A-201

Relational database[RDBMS]

Following databases falls under RDMBS

- ✓ IBM DB2
- ✓ Microsoft Access
- ✓ Microsoft SQL Server
- ✓ MySQL
- ✓ NonStop SQL
- ✓ Oracle
- ✓ The SAS system
- ✓ SQLBase
- ✓ SQLite

account-number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

customer-id	*account-number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Microsoft SQL Server 2008

- **Description:**
  - Microsoft SQL Server is a relational database management system developed by Microsoft. As a database, it is a software product whose primary function is to store and retrieve data as requested by other software applications, be it those on the same computer or those running on another computer across a network (including the Internet)

- **Components of SQL Server:**
  - SQL server database engine
  - SSAS- Sql server analysis services
  - SSIS- Sql server integration services
  - SSRS-Sql server reporting services

- **Why we are using this:**
  - This is widely used in Fractal and much demanded by client wherever data is big in volume.

SQL and Capabilities

- ✓ SQL is a language used to retrieve data from databases.
  - ✓ can execute queries against a database
  - ✓ can retrieve data from a database
  - ✓ can insert records in a database
  - ✓ can update records in a database
  - ✓ can delete records from a database
  - ✓ can create new databases
  - ✓ can create new tables in a database
  - ✓ can create stored procedures in a database
  - ✓ can create views in a database
  - ✓ can set permissions on tables, procedures, and views

SQL DDL and DML

- ✓ SQL is more than just a “query” language
- ✓ It has two parts:
  - ✓ SQL Data Definition Language
    - ✓ CREATE TABLE – creates a new database table
    - ✓ ALTER TABLE – alters a database table
    - ✓ DROP TABLE – deletes a database table
  - ✓ SQL Data Manipulation Language
    - ✓ SELECT – queries and fetches data from a database table
    - ✓ UPDATE – modifies one or more rows in a database table
    - ✓ DELETE FROM – deletes one or more rows from a database table
    - ✓ INSERT INTO – inserts new rows in a database table

Domain(Data) Types in SQL

- ✓ Each column in a table has a “data type” or “domain type” associated to it
- ✓ The domain type is specified while creating the table

- ✓ The commonly used domain types are
  - ✓ **char(n)** Fixed length character string with user specified length n
  - ✓ **varchar(n)** Variable length character string with user specified maximum length n
  - ✓ **int(n)** Integer with user specified maximum number of digits
  - ✓ **smallint(n)** Small integer with user specified maximum number of digits
  - ✓ **numeric(p,d)** Fixed point number, with user specified precision of p digits, with n digits to the right of the decimal point
  - ✓ date Dates usually in ‘**yyyy-mm-dd’** format

Primary & Foreign Keys

- ✓ **Primary Keys**:
  - ✓ The Primary key defines the column(s) that uniquely identify every row in the table. Remember to specify all columns within the primary key as NOT NULL
  - ✓ You can have only a single primary key constraint defined for a table

- ✓ **Foreign Keys**:
  - ✓ Foreign keys is used to implement referential integrity between tables within your database.
  - ✓ By creating foreign keys you can ensure that related tables cannot contain invalid rows

Constraints

- ✓ Constraints are basically the limitation that you want to put on your database while inserting new data in the table
- ✓ Default constraints allow you to specify a value that is written to the column if the application does not supply a value
- ✓ Default constraints apply only to new rows added with an INSERT statement
- ✓ You can define default constraints for either NULL or NOT NULL columns
- ✓ The following chart describes the types of restrictions you can place on columns and tables by using database-level constraints and triggers

Constraint	Column	Row	Table	External
Not Null	✓	✗	✗	✗
Check	✓	✓	✗	✓
Unique	✗	✗	✓	✗
Primary Key	✓	✗	✓	✗
Foreign Key	✓	✗	✗	✓
Index	✗	✗	✓	✗
Trigger	✓	✓	✗	✓

Database Objects

Database Objects

- ✓ Handle storage, manipulation, retrieval of data & data integrity . Available database objects are:
  - ✓ Tables - a table is a set of data elements (values) that is organized columns rows.
  - ✓ Views - a view is the result set of a *stored* query.
  - ✓ Temporary Table – It could be very useful to keep temporary data and will be deleted when the current client session terminates.
  - ✓ Rules - rules are the requirements and restrictions for columns.
  - ✓ Constraints - are user-defined structures that let you restrict the behaviors of columns
  - ✓ Stored Procedures - is a subroutine. Complex processing that requires execution of several SQL statements can be moved into stored procedures.
  - ✓ Triggers - is automatically executed in response to certain events on a particular table or view in a database.
  - ✓ Indexes - is a data structure that improves the speed of data retrieval operations on a database table

Rules for Naming Database Objects

- There are a series of standard naming rules for Data base objects
  - **Standard Identifiers**
    - Contain up to 128 characters
    - Include alphabets, symbols & numbers
    - Rules to be followed while using standard identifiers
    - First character should be an alphabet [a – z or A – Z]
    - This can be followed by alphabets, numbers, or any of the special characters @, $, # or
  - **Delimited Identifiers**
    - If an identifier does not comply with the rules for the format identifiers, it must always be delimited
    - Used when names contain embedded spaces, or when reserved words are used

Guidelines for naming database objects

- **Guidelines for naming database objects**
  - Use short names
  - Use meaningful names
  - Follow a clear naming convention
  - Use an identifier that distinguishes the type of object
  - Keep the user name and object name unique

- **Data base objects can be referred in several ways**
  - Fully Qualified Names:
    - Complete name of an SQL Server object includes four identifiers - Server name, Database name, Owner name, Object name
    - An object name that specifies all four parts is known as a fully qualified name
  - Partially Qualified Names:
    - When referencing an object, the user is not required to specify the server, database, and owner always
    - Intermediate identifiers can be omitted as long as their position is indicated by periods

Creating Database

```
CREATE DATABASE database_name
ON
( [ NAME = logical_file_name, ]
FILENAME = 'os_file_name'
[, SIZE = size]
[, MAXSIZE = { max_size | UNLIMITED } ];
)
```

Name of the database to be created

Logical database name (Optional)

Location of the database to be created (Optional)

Example:

```
CREATE DATABASE Test ON
( NAME = Test,
  FILENAME = 'c:\Test.mdf',
  SIZE = 4,
  MAXSIZE = 10,
  FILEGROWTH = 1 );
```

Types of Database File:

- ✓ Primary Data base File (.mdf) /
- ✓ Main database file – Mandatory
- ✓ Secondary Data base File (.ndf)
- ✓ Log Data base File (.ldf)
- ✓ Archive Data base File

Alter Database

- ✓ Altering DB by Adding New file / Removing Existing File

Syntax:

```
ALTER DATABASE database
{ ADD FILE < filespec > [ ,...n ]
| ADD LOG FILE < filespec > [ ,...n ]
| REMOVE FILE logical_file_name
| MODIFY FILE < filespec >
| MODIFY NAME = new_dbname } ;
```

Example:

```
ALTER DATABASE Test
ADD FILE (NAME = addnewfile_data, FILENAME = 'c:\addnewfile.ndf',
SIZE = 5MB, MAXSIZE = 100MB, FILEGROWTH = 5MB) ;
```

- ✓ Removing file from the database
  ALTER DATABASE Test REMOVE FILE Testlog ;

- ✓ Modifying a file
  ALTER DATABASE Test MODIFY FILE (NAME = Testlog, SIZE = 20MB) ;

Create Table

- ✓ The general syntax for a CREATE TABLE command is:

```
CREATE TABLE r (A_1 D_1, A_2 D_2, …, A_n D_n,
        (integrity-constraint_1),
        …
        (integrity-constraint_m)
    )
```

- ✓ Given a row, the values for some columns may be missing, unknown or not applicable for that row
- ✓ Such values are designated with the SQL keyword NULL
- ✓ Using SQL, one can specify an “integrity constraint” on a column ensuring it does not contain null values for any row

- ✓ The primary key is a combination of columns which uniquely identify each row
  - ✓ No two rows can have same values in all the columns of the primary key
  - ✓ A table may or may not have a primary key

Create Table . . .

```
CREATE TABLE MEMBERS (
                MEMBER_ID VARCHAR2(6) NOT NULL,
                MEMBER_NAME VARCHAR2(10),
                MEMBER_AGE NUMBER,
                ADDRESS VARCHAR2(40));
```

Member_id	Member_name	Member_age	Address

Book_id	Book_name	Author_name	Category

```
CREATE TABLE  BOOKS (
                BOOK_ID VARCHAR2(6) NOT
NULL,
                BOOK_NAME VARCHAR2(20),
                AUTHOR_NAME VARCHAR2(15),
                CATEGORY VARCHAR2(15)
                );
```

```
CREATE TABLE TRACKER (
                BOOK_ID VARCHAR2(6) NOT NULL,
                MEMBER_ID VARCHAR2(6),
                ISSUE_DATE DATE,
                ISSUED_BY VARCHAR2(15));
```

Book_id	Member_Id	Issue_date	Issued_By

Drop & Alter Table

- ✓ The ALTER TABLE command is used to add columns to an existing table
- ✓ All rows in the table are assigned NULL as the value for the RETURN_DATE
- ✓ The ALTER TABLE command can also be used to remove existing columns

```
ALTER TABLE TRACKER
        ADD RETURN_DATE DATE;
```

```
DROP TABLE TRACKER;
```

- ✓ The DROP TABLE command deletes the specified table from the database
- ✓ DROP TABLE TRACKER

Data Manipulation and Retrieval

Data Manipulation | Insert Data

✓ Use the INSERT command to enter data into a table.

Syntax :

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

✓ Types of Insert :

Insert the exact values as per table structure (columns).

:

```
INSERT  INTO  SalesStaff1  VALUES (1, 'Stephen', 'Jiang');
```

Insert multiples values at same time i.e. using single query.

:

```
INSERT  INTO  SalesStaff1  VALUES
    (2, 'Michael', 'Blythe'),
    (3, 'Linda', 'Mitchell'),
    (4, 'Jillian', 'Carson');
```

Insert only required values into a table from another table.

:

```
INSERT  INTO  SalesStaff1(StaffID, FirstName)
SELECT StaffID, FirstName FROM SalesStaff2 WHERE  StaffID=8;
```

Data Manipulation | Insert Data

- ✓ Inserting value in a table having IDENTITY column.
- ✓ Consider below Person table

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

- ✓ IDENTITY keyword is used for auto-increment.
- ✓ Default starting value is 1 and increments by 1 for each new record.
- ✓ To specify that the "P_Id" column should start at value 10 and increment by 5, change the identity to IDENTITY(10,5).

```
INSERT INTO Person (FirstName,LastName) VALUES ('Lars','Monsen');
```

- ✓ No Need to provide value for IDENTITY column as it is entered automatically by SQL server.

Data Manipulation | Updating Data

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

Syntax :

```
UPDATE table_name
SET column1=value1,column2=value2,...
WHERE some_column=some_value;
```

```
UPDATE SalesStaff1
SET FirstName='Alfred', DeptNo=10 WHERE = StaffID=3;
```

```
UPDATE table1 A
SET A.Fld1 = ( SELECT B.Fld1 FROM table2 B WHERE A.Fld2 = B.Fld2 );
```

If WHERE clause is omitted in above query then all records will get updated.

Data Manipulation | Copying a Table

- ✓ To create a copy of table we can use SELECT * INTO query .
- ✓ Syntax :

```
SELECT * INTO new_table_name FROM old_table_name;
```

- ✓ The above query will create new table with same table structure as that of old table and all rows will get copied.
- ✓ Ex:

```
SELECT *  INTO Person_cpy FROM Person;
```

The new table must not exist already in the database.

Data Manipulation | Delete & Truncate

DELETE (DML)

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

Syntax :

To delete rows based on condition

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

To delete all rows from table

```
DELETE * FROM table_name;
```

```
DELETE FROM Customers WHERE CustName='Jim Anders' AND ContactName='Maria Anders';
DELETE * FROM Customers;
```

TRUNCATE (DDL)

✓ Records removed by the TRUNCATE TABLE statement cannot be restored.

✓ TRUNCATE TABLE removes all rows from a table, but the table structure and its columns, constraints, indexes remain.

Syntax :

```
TRUNCATE TABLE table_name;
```

```
TRUNCATE TABLE Customers;
```

Data Manipulation | Delete …

✓ Difference Between DELETE and TRUNCATE

DELETE	TRUNCATE
Logs delete operation thus making it slow.	Does not log any activity, therefore it is faster.
Can be rolled back.	Cannot be rolled back.
Can have criteria (WHERE clause) to delete.	There is no criteria, all records are truncated.

Data Retrieval | Select & Where

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

```
SELECT column_name1,column_name2  FROM  table_name;
```

```
SELECT CustomerName FROM Customers WHERE Country='Mexico’;
```

```
SELECT  *  FROM Customers;
```

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

```
SELECT column_name1 FROM table_name WHERE column_name 2 operator value;
```

```
SELECT * FROM Customers WHERE Country='Mexico';
```

Data Retrieval | Operators in WHERE Clause

Operator	Description
=	Equal
<>	Not equal. Note: In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Data Retrieval | Fetching Random Data

- ✓ **Comparison operators: = > < <= >= <> (or !=)**

```
SELECT * FROM Customers WHERE Country <> 'UK';
```

- ✓ **BETWEEN ... AND ...**

```
SELECT * FROM products WHERE Price BETWEEN 10 AND 20;
SELECT * FROM products WHERE CompanyName BETWEEN  A  AND  F ;
```

- ✓ **LIKE (with wildcard: % _)**

```
SELECT * FROM Customers WHERE ContactTitle LIKE 'Sales%';
```

- ✓ **IN (list)**

```
SELECT * FROM Customers WHERE State IN ('OR', 'WA', 'CA') ;
```

- ✓ **AND, NOT, OR with any of the above**

```
SELECT * FROM Customers WHERE Country = 'USA' AND City = 'New York';
```

- ✓ **IS NULL, IS NOT NULL**

```
SELECT * FROM Customers WHERE PostalCode IS NOT NULL;
SELECT * FROM products WHERE Discount IS NULL;
```

Data Retrieval | Fetching Distinct Values

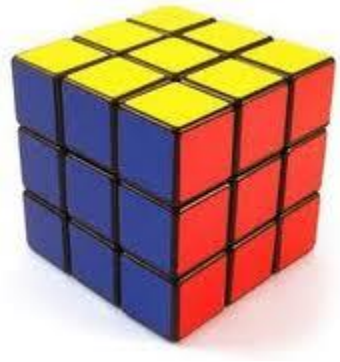
- ✓ In a table, a column may contain many duplicate values and sometimes you only want to list the different (distinct) values.

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

```
SELECT DISTINCT column_name1,column_name 2 FROM table_name;
```

```
SELECT DISTINCT Country FROM Customers;
```

Grouping and Summarizing Data



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.

✓ This is very useful for large tables with thousands of records

SELECT TOP number|percent column_name(s) FROM table_name

SELECT TOP 1 Country_Code, Vendor_Value_MLC, Vendor_Volume_MSU
FROM [AAIJK BABY CARE]

COUNTRY_CODE	DATABASE	VENDOR_VALUE_MLC	VENDOR_VOLUME_MSU
IN	DP	33.32184	0.01546311

SELECT TOP 1* FROM [[AAIJK BABY CARE]

SELECT TOP 60 percent * FROM CUSTOMER

Aggregate Functions

- ✓ With SQL and SQL Server you can use lots of built-in functions or you may create your own functions.
- ✓ SQL has many built-in functions for performing calculations on data.
- ✓ We have 2 categories of functions, Aggregate functions return a single value, calculated from values in a column, while scalar functions return a single value, based on the input value.

- ✓ Aggregate functions - examples:
  - ➢ AVG() - Returns the average value
  - ➢ COUNT() - Returns the number of rows
  - ➢ MAX() - Returns the largest value
  - ➢ MIN() - Returns the smallest value
  - ➢ SUM() - Returns the sum

- ✓ Scalar functions - examples:
  - ➢ UPPER() - Converts a field to upper case
  - ➢ LOWER() - Converts a field to lower case
  - ➢ ROUND() - Rounds a numeric field to the number of decimals specified
  - ➢ GETDATE() - Returns the current system date and time

Aggregate Functions…

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

```
SELECT AVG(column_name)  FROM   table_name

SELECT  Avg(VENDOR_VALUE_MLC)    FROM   [AAIJK BABY CARE]
```

AVG_VENDOR_VALUE_MLC
229483.726994896

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

✓ 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

SELECT Count(Country_Code) AS CNT_VENDOR_VALUE_MLC
FROM  [AAIJK BABY CARE]
```

CNT_VENDOR_VALUE_MLC
1080174

Group By

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

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

Example

```
SELECT COUNTRY_CODE,  Avg(VENDOR_VOLUME_MSU) AS AVG_VENDOR_VOLUME_MSU
FROM [AAIJK BABY CARE]
GROUP BY COUNTRY_CODE
```

COUNTRY_CODE	AVG_VENDOR_VOLUME_MSU
ID	2.65908836939374
IN	1.08605703692316
JP	0.954831321140269
PH	1.33917439045886
VN	2.07322729027707

Having Clause

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

```
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
```

Example

```
SELECT  COUNTRY_CODE, Avg(VENDOR_VOLUME_MSU) AS AVG_VENDOR_VOLUME_MSU
FROM  [AAIJK BABY CARE]
GROUP BY COUNTRY_CODE
HAVING Avg(VENDOR_VOLUME_MSU)>2
```

COUNTRY_CODE	AVG_VENDOR_VOLUME_MSU
ID	2.65908836939374
VN	2.07322729027707

Built-in Functions

Built-in Functions | Numeric Functions

- ✓ The **ABS(m)** function returns the exponential value of the specified float expression.

```
SELECT ABS(m)

SELECT ABS(-52.25)
```

- ✓ The **MOD(m,n)** function returns the remainder of m divided by n

```
SELECT MOD(m, n);

SELECT MOD(96,3);
```

- ✓ The **POW(m,n)** function returns m raised to the nth power

```
SELECT POW(m, n);

SELECT POW(8,3);
```

Built-in Functions | Numeric Functions

- ✓ The **ROUND (m [, n])** function returns m rounded to the nth decimal place

```
SELECT ROUND(m)

SELECT  ROUND(25.65, 2)
```

- ✓ The **TRUNC(m, n)** function returns m truncated to the nth decimal place

```
SELECT TRUNC(m, n) FROM   table_name;

SELECT  TRUNC(25.85, 2)
```

- ✓ The **CEILING(m)** smallest integer greater than or equal to m

```
SELECT CEILING( m )

SELECT  CEILING(52.58)
```

Similarly, there is **FLOOR (m)** function which returns greatest integer smaller than or equal to m

Built-in Functions | String Functions

- ✓ The **CHARINDEX()** function searches an expression for another expression and returns its starting position if found.

```
SELECT CHARINDEX ( expressionToFind ,expressionToSearch [ , start_location ] )

SELECT CHARINDEX( 'e','abcdef', 1)
```

- ✓ The **REVERSE(s)** function Returns the reverse order of a string value

```
SELECT REVERSE ( s )

SELECT REVERSE( ‘ abcd ‘ )
```

- ✓ The **LENGTH(s)** function returns the number of characters in s

```
SELECT LENGTH(column_name);

SELECT LENGTH(‘123456’)
```

Built-in Functions | String Functions

- ✓ The **LTRIM (s)** function returns s with blanks removed up to the first character

```
SELECT LTRIM(s)

SELECT LTRIM('  hello')
```

Similarly, we have RTRIM() which return s with blanks removed after the last character

- ✓ The **LEFT (s, n)** function returns the left part of a character string with the specified number of characters.

```
SELECT LEFT ( character_expression , integer_expression )

SELECT LEFT ( 'abcd' , 2 )
```

Similarly, RIGHT (s, n) function returns the right part of a character string with the specified number of characters.

Built-in Functions | Date Functions

✓ The **ISDATE (d)** function returns 1 or 0 depending on whether the argument is a valid date or not

```
SELECT ISDATE(d)

SELECT ISDATE('NAME')          'returns 0
SELECT ISDATE(GetDate())       'returns 1
```

✓ The **DAY (d)** function returns day number of the month

```
SELECT DAY(d)

SELECT DAY ('01/31/2012')      'returns 31
```

✓ The **MONTH (d)** function returns month number of the year

```
SELECT MONTH (d)

SELECT MONTH ('01/31/2012')    'returns 1
```

Similarly, we have **YEAR(d)** function

Built-in Functions | Date Functions

✓ The **DATEPART()** function is used to return a single part of a date/time, such as year, month, day, hour, minute, etc.

SELECT DATEPART(datepart, date)

SELECT DATEPART(yyyy,'01/31/2012')

Where date is a valid date expression and datepart can be one of the following:

datepart	Abbreviation
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw, w
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns

Modifying Table Structures

Modifying Table Structure

- ✓ After a table has been in use for some time, users often discover that they want to store additional information about the entities represented in the table.

- ✓ It includes
  - ✓ Changing Table Definition
  - ✓ Changing Column Definition
  - ✓ Changing Constraint Definition

- ✓ The ALTER TABLE statement can:
  - ✓ Add a column definition to a table
  - ✓ Drop a column from a table
  - ✓ Change the default value for a column
  - ✓ Add or drop a primary key for a table
  - ✓ Add or drop a new foreign key for a table
  - ✓ Add or drop a uniqueness constraint for a table
  - ✓ Add or drop a check constraint for a table

Changing Table Definition | Add Column

- ✓ The most common use of the ALTER TABLE statement is to add a column to an existing table.
- ✓ The column definition clause in the ALTER TABLE statement is just like the one in the CREATE TABLE statement, and it works the same way.
- ✓ The new column is added to the end of the column definitions for the table, and it appears as the rightmost column in subsequent queries.

ALTER TABLE table_name ADD column_name datatype;

ALTER TABLE CUSTOMERS ADD CONTACT_PHONE CHAR(10);

(*Add a minimum inventory-level column to the PRODUCTS table)*

ALTER TABLE PRODUCTS ADD MIN_QTY INTEGER NOT NULL WITH DEFAULT 0;

OR

ALTER TABLE PRODUCTS ADD COLUMN MIN_QTY INTEGER NOT NULL WITH DEFAULT 0

you cannot simply declare the new column NOT NULL, because the DBMS would assume NULL values for the column in the existing rows, immediately violating the constraint!

Changing Table Definition | Drop Column

- ✓ The ALTER TABLE statement can be used to drop one or more columns from an existing table when they are no longer needed.

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

- ✓ To handled the potential data-integrity problems posed by DELETE and UPDATE statements *y*ou can specify one of two drop rules:

RESTRICT: If any other objects in the database (foreign keys, constraints, and so on) depend on the column to be dropped, the ALTER TABLE statement fails with an error and the column is not dropped.

CASCADE: Any other objects in the database (foreign keys, constraints, and so on) that depend on the column are *also dropped as a cascaded effect of the ALTER* TABLE statement.

Dropping a column can pose the data-integrity issues.

Changing Table Definition | Alter Column

- ✓ The ALTER TABLE statement can be used to alter columns of an existing table when needed.
- ✓ Alter column means changing data type of a column.

  ALTER TABLE table_name
  ALTER COLUMN column_name datatype

<u>Changing Default Value of a Column</u>

- ✓ You can create a DEFAULT definition as part of the table definition when you create a table.
- ✓ If a table already exists, you can add DEFAULT definition to it. Each column in a table can contain one DEFAULT definition.
- ✓ If a DEFAULT definition already exists, you can modify or delete it.

  ALTER TABLE table_name
  ADD CONSTRAINT constraint_name DEFAULT value For column_name

Default value should match with the data type of the column

Add & Drop Primary | Foreign Key

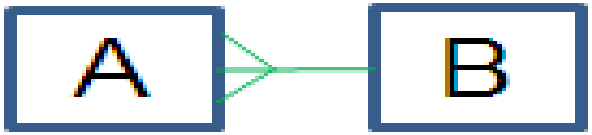
✓ The other common use for the ALTER TABLE statement is to change or add primary key definitions for a table.

```
ALTER TABLE table_name
 ADD CONSTRAINT constraint_name PRIMARY KEY (column1,column2,column_n)

ALTER TABLE table_name
DROP CONSTRAINT constraint_name ;
```

✓ The other common use for the ALTER TABLE statement is to change or add foreign key definitions for a table.

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name  FOREIGN KEY (column_name)
      REFERENCES primary_key_table_name(primary_key_column_name);

ALTER TABLE table_name
DROP CONSTRAINT constraint_name ;
```

Working with Sub-Queries



Sub Queries

- ✓ A Sub-Query is a SQL query nested inside a larger query.
- ✓ A Sub-Query may occur in :
  - ✓ A SELECT clause
  - ✓ A FROM clause
  - ✓ A WHERE clause
- ✓ A Sub-Query can be treated as an inner query, which is a SQL query placed as a part of another query called as outer query.
- ✓ A Sub-Query is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

The inner query executes first before its parent query

Sub Queries - Rules

There are a few rules that Sub-Queries must follow:

- ✓ Sub-Queries must be enclosed within parentheses.
- ✓ A Sub-Query can have only one column in the SELECT clause, unless multiple columns are in the main query for the Sub-Query to compare its selected columns.
- ✓ An ORDER BY cannot be used in a Sub-Query, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a Sub-Query.
- ✓ Sub-Queries that return more than one row can only be used with multiple value operators, such as the IN operator.
- ✓ The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- ✓ A Sub-Query cannot be immediately enclosed in a set function.
- ✓ The BETWEEN operator cannot be used with a Sub-Query; however, the BETWEEN can be used within the Sub-Query.

Sub Queries Example

- ✓ The general syntax for Sub-Query:

```
SELECT column_name [, column_name ]
FROM table1 [, table2 ]
WHERE column_name OPERATOR
                    (SELECT column_name [, column_name ]
                    FROM table1 [, table2 ]
                    [WHERE])
```

Example:

```
SELECT * FROM CUSTOMERS
WHERE ID IN (SELECT ID FROM CUSTOMERS WHERE SALARY > 4500)
```

Sub Queries | IN & Exists Operator

- ✓ Sub-Queries that are introduced with the keyword **IN** take the general form:
  WHERE expression [NOT] IN (Sub-Query)
- ✓ The only difference in the use of the IN operator with Sub-Queries is that the list does not consist of hard-coded values.

Example:

```
SELECT emp_last_name "Last Name“, emp_first_name "First Name"
FROM employee
WHERE emp_ssn  IN
(SELECT dep_emp_ssn      FROM dependent WHERE dep_gender = 'M')
```

- ✓ When a Sub-Query uses the **EXISTS** operator, the Sub-Query functions as an existence test.
- ✓ The WHERE clause of the outer query tests for the existence of rows returned by the inner query.
- ✓ The Sub-Query does not actually produce any data; rather, it returns a value of TRUE or FALSE.

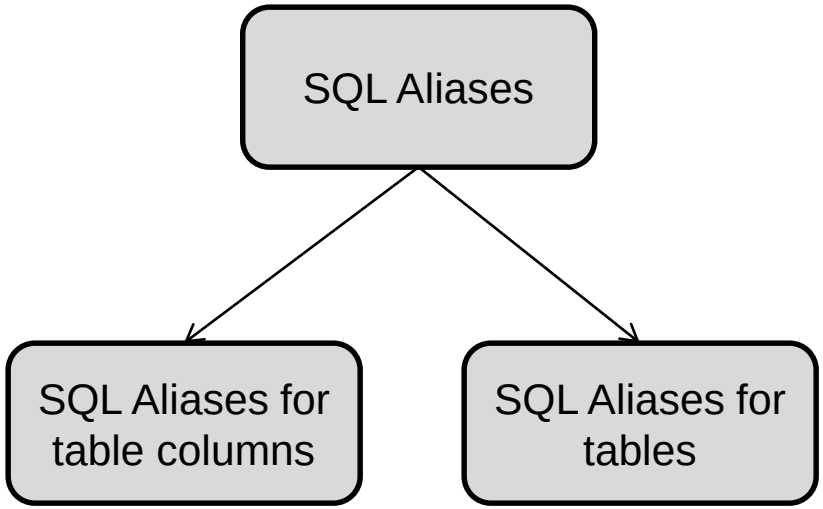
Example:

```
SELECT emp_last_name "Last Name", emp_first_name "First Name”
FROM employee
WHERE EXISTS
(SELECT *     FROM dependent WHERE emp_ssn = dep_emp_ssn)
```

Joining Multiple Tables

SQL Aliases

- ✓ SQL aliases are used to temporarily rename a table or a column heading.
- ✓ Basically aliases are created to make column names more readable.
- ✓ Aliases can be useful when:
  - ✓ There are more than one table involved in a query
  - ✓ Functions are used in the query
  - ✓ Column names are big or not very readable
  - ✓ Two or more columns are combined together



SQL Aliases for table columns

- ✓ **SQL Alias Syntax for Columns**

  SELECT *column_name* AS *alias_name*
  FROM *table_name;*

- ✓ **Example 1.**

  SELECT CustomerName AS Customer, ContactName AS [Contact Person]
  FROM Customers;

  Tip: It require double quotation marks or square brackets if the column name contains spaces:

- ✓ **Example 2.**

  In the following SQL statement we combine four columns (Address, City, PostalCode, and Country) and create an alias named "Address":

  SELECT CustomerName, Address+', '+City+', '+PostalCode+', '+Country AS Address
  FROM Customers;

SQL Aliases for table

- ✓ **SQL Alias Syntax for Columns**

  SELECT *column_name*
  FROM *table_name* AS *alias_name;*

- ✓ **Example**

  SELECT o.OrderID, o.OrderDate, c.CustomerName
  FROM Customers AS c, Orders AS o
  WHERE c.CustomerName='Alfreds Futterkiste';

- ✓ **Same SQL statement without aliases**

  SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
  FROM Customers, Orders
  WHERE Customers.CustomerName='Alfreds Futterkiste';

SQL Joins

- ✓ An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

- ✓ Types the different SQL JOINS are as follows
  - ✓ **INNER JOIN**: Returns all rows when there is at least one match in BOTH tables.
  - ✓ **LEFT JOIN**: Return all rows from the left table, and the matched rows from the right table.
  - ✓ **RIGHT JOIN**: Return all rows from the right table, and the matched rows from the left table.
  - ✓ **FULL JOIN**: Return all rows when there is a match in ONE of the tables.
  - ✓ **SELF JOIN** is used to join a table to itself, as if the table were two tables, temporarily renaming at least one table in the SQL statement.
  - ✓ **CARTESIAN JOIN** or **CROSS JOIN** returns the Cartesian product of the sets of records from the two or more joined tables.

SQL Joins – Inner Join

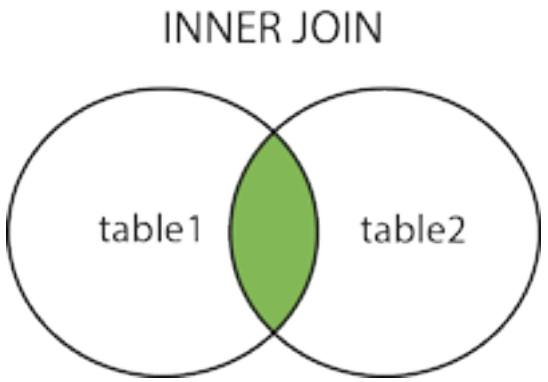
✓ The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.

SQL INNER JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name=table2.column_name*;

OR

SELECT *column_name(s)*
FROM *table1*
JOIN *table2*
ON *table1.column_name=table2.column_name*;



SQL Joins – Inner Join | Example

Employee

ENO	ENAME	DEPT
100	Raji	CSE
200	Eswar	EEE
300	Kailash	ECE
400	Durga	IT

Salary

ENO	DESIG	SALARY
100	LECT	15000
200	Sr. LECT	20000
500	PROF	35000
600	Ass. PROF	30000

```
Select    e.eno, ename, dept, desig, salary
FROM      Employee e Inner Join Salary s
          On e.eno = s.eno;
```

```
Select    e.eno, ename, dept, desig, salary
From      employee e , salary s
Where     e.eno = s.eno;
```

Output

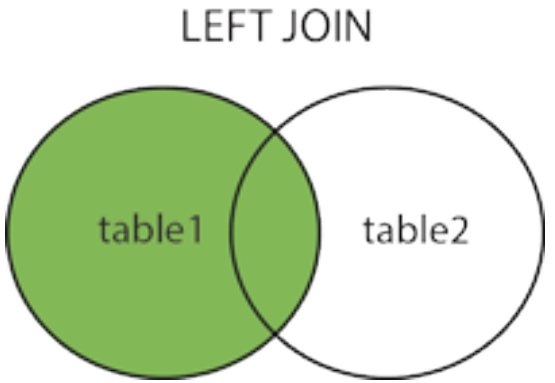
ENO	ENAME	DEPT	DESIG	SALARY
100	Raji	CSE	LECT	15000
200	Eswar	EEE	Sr. LECT	20000

SQL Joins – Left Join

- ✓ The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

SQL LEFT JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
LEFT JOIN *table2*
ON *table1.column_name=table2.column_name*;



SQL Joins – Left Join | Example

Employee

ENO	ENAME	DEPT
100	Raji	CSE
200	Eswar	EEE
300	Kailash	ECE
400	Durga	IT

Salary

ENO	DESIG	SALARY
100	LECT	15000
200	Sr. LECT	20000
500	PROF	35000
600	Ass. PROF	30000

```
Select   e.eno, ename, dept, desig, salary
From     employee e Left Outer Join salary s
         On e.eno = s.eno;
```

```
Select   s.eno, ename, dept, desig, salary
From     employee e Left Outer Join salary s
         On e.eno = s.eno;
```

ENO	ENAME	DEPT	DESIG	SALARY
100	Raji	CSE	LECT	15000
200	Eswar	EEE	Sr. LECT	20000
300	Kailash	ECE	Null	Null
400	Durga	IT	Null	Null

ENO	ENAME	DEPT	DESIG	SALARY
100	Raji	CSE	LECT	15000
200	Eswar	EEE	Sr. LECT	20000
500	Null	Null	PROF	35000
600	Null	Null	Ass. PROF	30000

SQL Joins – Right Join

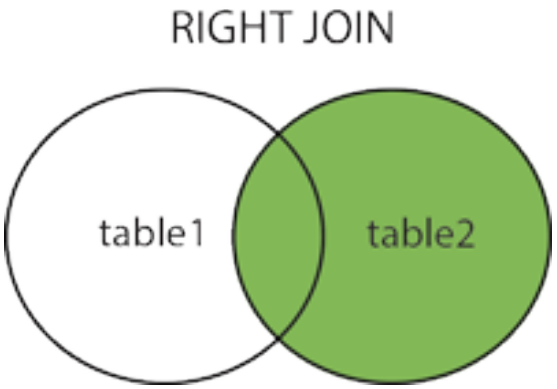
- ✓ The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

SQL Right JOIN Syntax

SELECT *column_name(s)*
FROM *table1* RIGHT JOIN *table2*
ON *table1.column_name=table2.column_name*;

OR

SELECT *column_name(s)*
FROM *table1* RIGHT OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;



SQL Joins – Right Join | Example

Employee

ENO	ENAME	DEPT
100	Raji	CSE
200	Eswar	EEE
300	Kailash	ECE
400	Durga	IT

Salary

ENO	DESIG	SALARY
100	LECT	15000
200	Sr. LECT	20000
500	PROF	35000
600	Ass. PROF	30000

```
Select   s.eno, ename, dept, desig, salary
From     employee e Right Outer Join salary s
         On e.eno = s.eno;
```

ENO	ENAME	DEPT	DESIG	SALARY
100	Raji	CSE	LECT	15000
200	Eswar	EEE	Sr. LECT	20000
500	Null	Null	PROF	35000
600	Null	Null	Ass. PROF	30000

```
Select   e.eno, ename, dept, desig, salary
From     employee e Right Outer Join salary s
         On e.eno = s.eno;
```

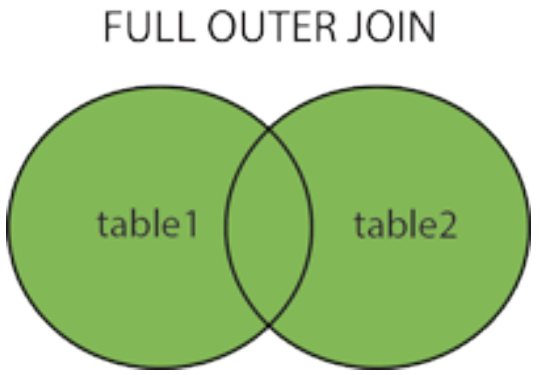
ENO	ENAME	DEPT	DESIG	SALARY
100	Raji	CSE	LECT	15000
200	Eswar	EEE	Sr. LECT	20000
300	Kailash	ECE	Null	Null
400	Durga	IT	Null	Null

SQL Joins – Full Outer Join

- ✓ The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

SQL Full Outer JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```



SQL Joins – Full Outer Join | Example

Employee

ENO	ENAME	DEPT
100	Raji	CSE
200	Eswar	EEE
300	Kailash	ECE
400	Durga	IT

Salary

ENO	DESIG	SALARY
100	LECT	15000
200	Sr. LECT	20000
500	PROF	35000
600	Ass. PROF	30000

```
Select  s.eno, ename, dept, desig, salary
From employee e Full Outer Join salary s On e.eno = s.eno;
```

ENO	ENAME	DEPT	DESIG	SALARY
100	Raji	CSE	LECT	15000
200	Eswar	EEE	Sr. LECT	20000
300	Kailash	ECE	Null	Null
400	Durga	IT	Null	Null
500	Null	Null	PROF	35000
600	Null	Null	Ass. PROF	30000

SQL Joins – Self Join | Example

ENO	ENAME	DEPT	DESIG	MGR NO	SALARY
100	Raji	CDW	CEO	100	150000
200	Eswar	CDW	Sr. Mgr	100	75000
300	Kailash	CDW	Mgr	200	50000
400	Durga	CDW	Sr. B.A.	300	35000
500	Shanmuga	CDW	B.A.	300	20000
600	Ganesh	CDW	B.A.	300	20000

```
Select   e2.ename as 'Employee Name' ,  e2.ename as 'Manager Name'
From     employee  e1, employee e2
Where    e2.mgrno = e1.eno;
```

Employee Name	Manager Name
Raji	Raji
Eswar	Raji
Kailash	Eswar
Durga	Kailash
Shanmuga	Kailash
Ganesh	Kailash

SQL Joins – Cross Join | Example

✓ A cross join is the foundation upon which inner joins are built.
✓ A cross join returns the Cartesian product of the sets of rows from the joined tables

Employee

ENO	ENAME	DEPT
100	Raji	CSE
200	Eswar	EEE
300	Kailash	ECE

Salary

DESIG	SALARY
LECT	15000
Sr. LECT	20000

```
Select   eno, ename, dept, desig, salary
From     employee e , salary s;
```

ENO	ENAME	DEPT	DESIG	SALARY
100	Raji	CSE	LECT	15000
100	Raji	CSE	Sr. LECT	20000
200	Eswar	EEE	LECT	15000
200	Eswar	EEE	Sr. LECT	20000
300	Kailash	ECE	LECT	15000
300	Kailash	ECE	Sr. LECT	20000

SQL Joins – Union

- ✓ The UNION operator is used to combine the result-set of two or more SELECT statements.
- ✓ Each SELECT statement within the UNION must have the same number of columns.
- ✓ The columns must also have similar data types.
- ✓ the columns in each SELECT statement must be in the same order.
- ✓ 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 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.

SQL UNION ALL Syntax

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

SQL Joins – Union | Example

CDW

ENO	ENAME	DEPT
100	Raji	CSE
200	Eswar	EEE
300	Kailash	ECE

ICICI

ENO	ENAME	DEPT
100	Raji	CSE
400	Siva	EEE
500	Durga	ECE

```
Select eno, ename, dept    From CDW
Union
Select eno, ename, dept    From ICICI;
```

```
Select eno, ename, dept    From CDW
Union all
Select eno, ename, dept    From ICICI;
```

ENO	ENAME	DEPT
100	Raji	CSE
200	Eswar	EEE
300	Kailash	ECE
400	Siva	EEE
500	Durga	ECE

ENO	ENAME	DEPT
100	Raji	CSE
200	Eswar	EEE
300	Kailash	ECE
100	Raji	CSE
400	Siva	EEE
500	Durga	ECE

SQL Joins – Minus | Example

- ✓ The MINUS operates on two SQL statements.
- ✓ It takes all the results from the first SQL statement, and then subtract out the ones that are present in the second SQL statement to get the final answer.
- ✓ If the second SQL statement includes results not present in the first SQL statement, such results are ignored.

CDW

ENO	ENAME	DEPT
100	Raji	CSE
200	Eswar	EEE
300	Kailash	ECE

ICICI

ENO	ENAME	DEPT
100	Raji	CSE
400	Siva	EEE
500	Durga	ECE

```
Select eno, ename, dept From CDW
 Minus
Select eno, ename, dept From ICICI;
```

```
Select eno, ename, dept From ICICI
 Minus
Select eno, ename, dept From CDW;
```

ENO	ENAME	DEPT
200	Eswar	EEE
300	Kailash	ECE

ENO	ENAME	DEPT
400	Siva	EEE
500	Durga	ECE

SQL Joins – Intersect | Example

✓ INTERSECT operates on two SQL statements.
✓ INTERSECT command acts as an AND operator (value is selected only if it appears in both statements).

CDW

ENO	ENAME	DEPT
100	Raji	CSE
200	Eswar	EEE
300	Kailash	ECE

ICICI

ENO	ENAME	DEPT
100	Raji	CSE
400	Siva	EEE
500	Durga	ECE

```
Select    eno, ename, dept    From CDW

Intersect

Select    eno, ename, dept    From ICICI
```

ENO	ENAME	DEPT
100	Raji	CSE

Thank you