

Dot Net Team - TDG – Injazat
ahmed.elbaz@injazat.com

SQL PROGRAMMING OVERVIEW




Agenda

- History of SQL
- SQL Fundamentals
- Data Definition
- Data Modifications
- Single Table SELECT Statements
- Joins
- Set Operators
- Aggregate Functions
- Subqueries
- Views
- Stored Procedures & Functions
- Triggers
- The Mechanics of Query Processing




History of SQL

- During the 1970s(IBM), *Structured English Query Language(SEQUEL)*
 - *Select Query Language (SQL)*, which included commands allowing data to be read only for reporting and record look-up.
 - *Structured Query Language*
 - American National Standards Institute and officially called the *ANSI SQL* standard, established in 1986
- 



History of SQL (cont'd)

- In 1987, SQL became an international standard and was registered with the International Organization for Standardization (ISO) using its previously copyrighted title, ANSI SQL
- 

SQL revisions


Year	Name	Comments
1986	SQL-86	First published by ANSI. Ratified by ISO in 1987.
1989	SQL-89	Minor revision, adopted as FIPS 127-1.
1992	SQL-92	Major revision (ISO 9075), Entry Level SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	Added regular expression matching, recursive queries, triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features.
2003	SQL:2003	Introduced XML-related features, window functions, standardized sequences, and columns with auto-generated values (including identity-columns).
2006	SQL:2006	ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form.
2008	SQL:2008	Defines more flexible windowing functions

History of SQL Server

- SQL Server was originally a Sybase product created for IBM's OS/2 platform.
- The SQL Server team eventually rewrote the product from scratch.
- In late 1998 , SQL Server 7.0 was released.
- In 2000, SQL Server 2000 was released with many useful new features.
- SQL Server 2005, The storage and retrieval engine has been completely rewritten, the .NET Framework has been incorporated.



Transact SQL (T-SQL)

- T-SQL is Microsoft's implementation of a SQL.
 - T-SQL is the language used to talk to SQL Server.
 - SQL Server 2000 implements ANSI-92.
 - SQL Server 2005 implements ANSI-99.
 - T-SQL does not fully conform to any of the more recent ANSI versions, but it does implement many of the selected features.
- 




SQL Fundamentals


SQL is the only way to build, manipulate and access a relational database.






What is a Relational Database?

- In simple terms, a relational database is a set of tables
 - a relational database is a set of *relations*
 - Each table keeps information about aspects of one thing, such as a customer, an order, a product, or a team.
 - It is possible to set up constraints on the data in individual tables and also between tables.
 - For example, when designing the database, we might specify that an order entered in the Order table must exist for a customer who exists in the Customer table.
 - A *data model* provides us with information about how the data items in the database are interrelated
- 




The first step
to write good query is
the database design





Table

- Columns and Rows
 - Each row must be able to stand on its own, without a dependency to other rows in the table.
 - The row must represent a single, complete instance of the entity the table was created to represent.
 - Each column in the row contains specific attributes that help define the instance
- 




Relationships

- A relationship is a logical link between two entities
- Relationships can be defined as follows:
 - One-to-zero or many
 - One-to-one or many
 - One-to-exactly-one
 - Many-to-many
- The many-to-many relationship requires three tables because a many-to-many constraint would be unenforceable




Relationships (cont'd)

- Entity Relationship Diagram (ERD). The ERD allows the database designer to conceptualize the database design during planning.
 - Primary keys (unique column or combination of columns)
 - Foreign keys (a column that references primary key of another table)
 - To efficiently manage the data in your table you need to be able to uniquely identify each individual row in the table.
 - The concept of maintaining foreign keys is known as "referential integrity".
- 



Primary key

- A non-data key that has a more efficient or smaller data type associated with it
 - A non-descriptive key doesn't represent anything else with the exception of being a value that uniquely identifies each row or individual instance of the entity in a table.
 - Simplify the joining of tables and provide the basis for a "Relation."
 - Primary keys can never be NULL and they must be unique
 - Primary keys can also be combinations of columns
- 



Normalization

Each table describes one thing only






Normalization

- Normalization is the process of efficiently organizing data in a database.
- There are two goals of the normalization process:
 - eliminating redundant data (for example, storing the same data in more than one table)
 - ensuring data dependencies make sense (only storing related data in a table).
- Both of these are worthy goals as they:
 - reduce the amount of space a database consumes
 - ensure that data is logically stored.




First Normal Form – 1NF

- Eliminate duplicative columns from the same table.
 - Create separate tables for each group of related data and identify each row with a unique column (the primary key).
- 



Second Normal Form – 2NF

- Remove subsets of data that apply to multiple rows of a table and place them in separate tables.
 - Create relationships between these new tables and their predecessors through the use of foreign keys.
 - 2NF attempts to reduce the amount of redundant data in a table by extracting it, placing it in new table(s) and creating relationships between those tables.
- 

Third Normal Form – 3NF

- The uniqueness of a row depends on the key, the whole key, and nothing but the key.
- Already meet the requirements of both 1NF and 2NF
- Remove columns that are not fully dependent upon the primary key.
- Fourth and Fifth Normal Form

Normalization Example

- we will take the classic example of an **Invoice** and level it to the Third Normal Form.
- We will also construct an **Entity Relationship Diagram** (ERD) of the database as we go.
- Just memorize the 3 normal forms them for now:
 - No repeating elements or groups of elements
 - No partial dependencies on a concatenated key
 - No dependencies on non-key attributes

Normalization Example (cont'd)

orders.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Invoice No.	Date	Cust. No.	Cust. Name	Cust. Address	Cust. City	Cust. State	Item ID	Item Description	Item Qty.	Item Price	Item Total	Order Total Price
2	125	9/13/2002	56	Foo, Inc.	23 Main St., Thorpleburg	Thorpleburg	TX	563	56" Blue Fre	4	\$ 3.50	\$ 14.00	\$ 82.00
3								851	Spline End i	32	\$ 0.25	\$ 8.00	\$ 82.00
4								652	3" Red Free	5	\$ 12.00	\$ 60.00	\$ 82.00
5	126	9/14/2002	2	Freens R Us	1600 Pennsylvania Avenue	Washington	DC	563	56" Blue Fre	500	\$ 3.50	\$1,750.00	\$ 10,750.00
6								652	3" Red Free	750	\$ 12.00	\$9,000.00	\$ 10,750.00

orders : Table

order_id	order_date	customer_id	customer_name	customer_addre	customer_city	customer	item_id	item_description	item_qty	item_price	item_total_price	order_total_price
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	563	56" Blue Freen	4	\$3.50	\$14.00	\$82.00
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	851	Soline End (Xtra	32	\$0.25	\$8.00	\$82.00
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	652	3" Red Freen	5	\$12.00	\$60.00	\$82.00
126	9/14/2002	2	Freens R Us	1600 Pennsylv	Washington	DC	563	56" Blue Freen	500	\$3.50	\$1,750.00	\$10,750.00
126	9/14/2002	2	Freens R Us	1600 Pennsylv	Washington	DC	652	3" Red Freen	750	\$12.00	\$9,000.00	\$10,750.00

Record: 6 of 6

1NF: No Repeating Elements or Groups of Elements

- NF₁ addresses two issues:
 - A row of data cannot contain repeating groups of similar data (**atomicity**).
 - Each row of data must have a unique identifier (or **Primary Key**).

orders

order_id (PK)

order_date

customer_id

customer_name

customer_address

customer_city

customer_state

item_id (PK)

item_description

item_qty

item_price

item_total_price

order_total_price

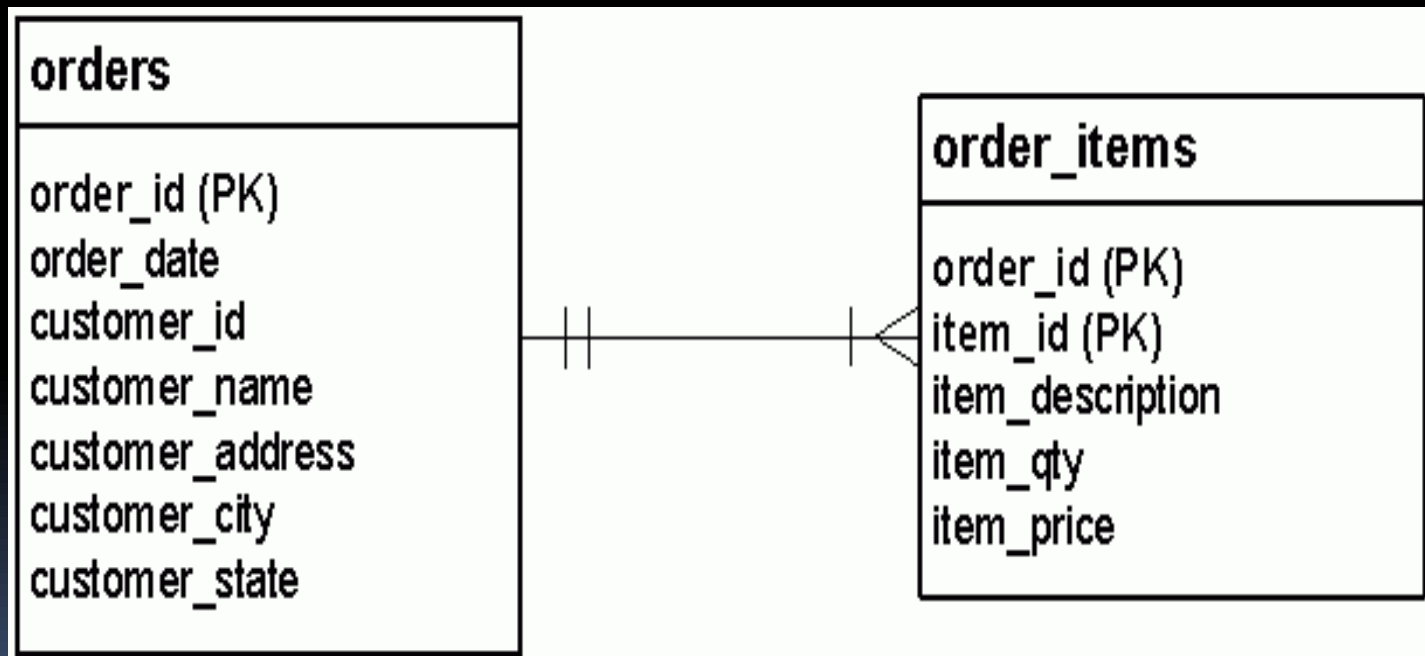
2NF: No Partial Dependencies on a Concatenated Key

- NF2 analysis of the orders table:

orders	
order_id (PK)	
order_date	✗
customer_id	?
customer_name	?
customer_address	?
customer_city	?
customer_state	?
item_id (PK)	
item_description	✗
item_qty	✓
item_price	✗
item_total_price	
order_total_price	

2NF (cont'd)

- each order can be associated with any number of order-items, but **at least one**;
- each order-item is associated with one order, and **only one**.



2NF (cont'd)

- Remember, NF2 *only* applies to tables with a concatenated primary key.
- Now **orders** table has a single-column primary key, it has passed 2NF.
- **order_items** still has a concatenated PK.

order_items

order_id (PK)

item_id (PK)

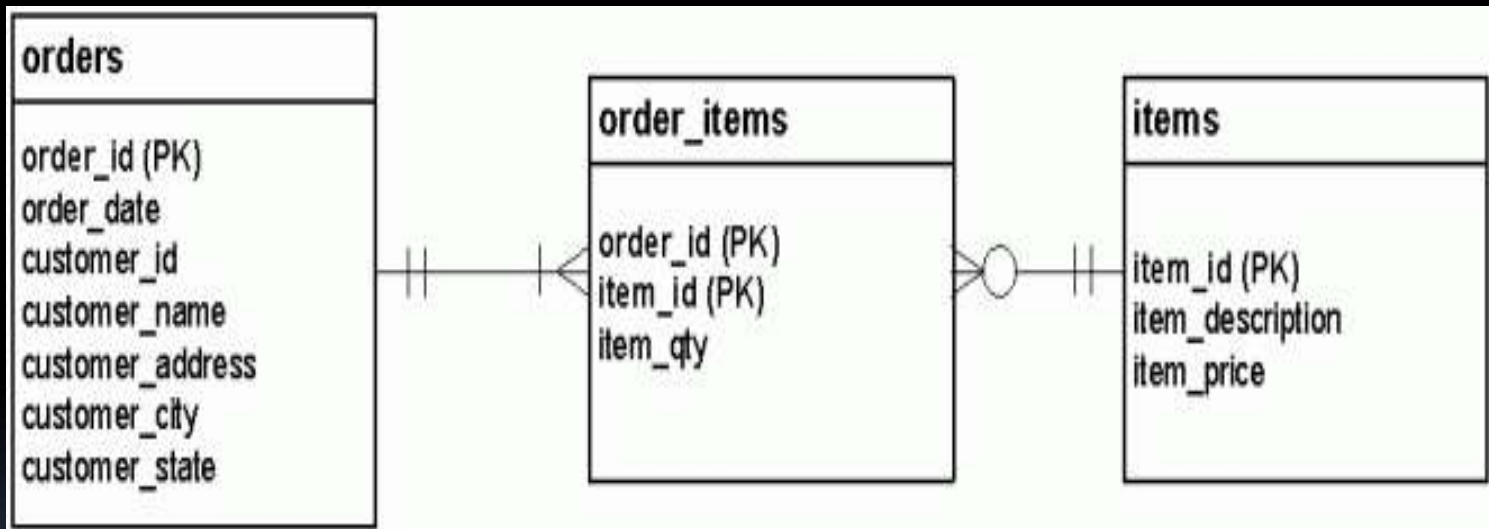
item_description ✗

item_qty ✓

item_price ✗


2NF (cont'd)

- Each order can have many items; each item can belong to many orders.

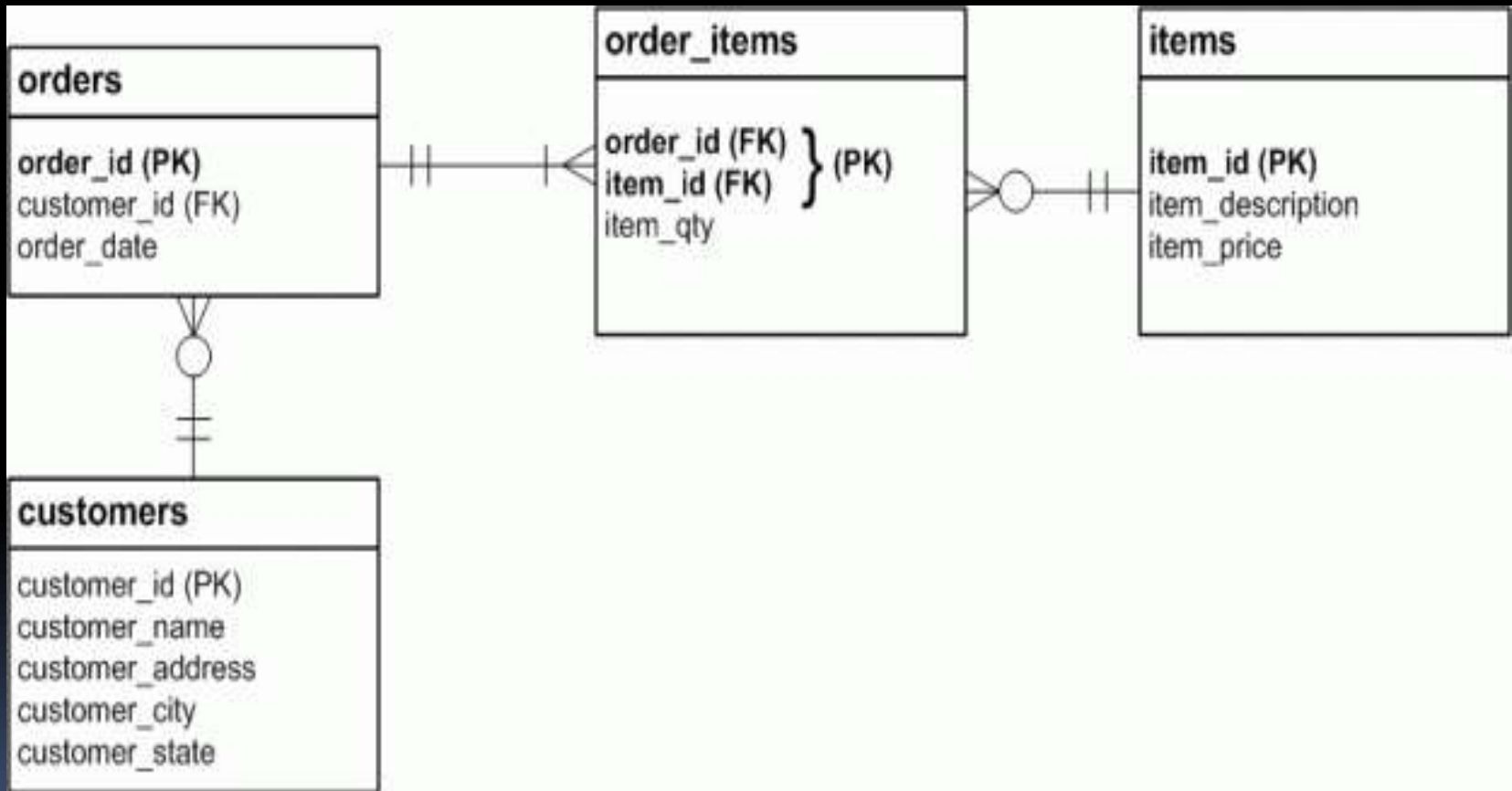




3NF: No Dependencies on Non-Key Attributes


- the repeating Customer information.
 - if a customer places more than one order then we have to input all of that customer's contact information again.
 - each order is made by one, and **only** one customer.
 - each customer can make any number of orders, including zero.
- 

3NF (cont'd)






To Normalize or to De-normalize?

- Depending on how a database is to be used.
 - data input or reporting.
 - fully normalized databases require complex queries to support reporting and business analysis requirements.
 - If you are designing a new database system to support a typical business process, you will usually want to follow these rules completely and normalize all of your data structures.
 - when you have a large volume of data to extract and analyze through reports and business intelligence tools, you will strategically break the rules of normal form, creating redundant values in fewer, larger tables.
 - OLAP vs. OLTP
- 



Connections and Transactions

SQL architecture important pieces
need to be understood






Connections

- A connection is created anytime a process attaches to SQL Server.
- The connection is established with defined security and connection properties.
- For example, a connection can specify which database to connect to on the server and how to manage memory resident objects.
- Connection pooling.



Transactions

- Transaction is a collection of dependent data modifications that is controlled so that it completes entirely or not at all.
 - For example, bank transfer from an account to another account.
 - database objects designed to maintain data integrity in a transactional environment
 - Locks
 - Constraints
 - Keys
 - Indexes
- 

Query operations are divided into three different categories

- **Data Definition Language (DDL)** — DDL statements are used to create and manage the objects in a database. They can be used to create, modify, and drop databases, tables, indexes, views, stored procedures, and other objects.
 - Examples include CREATE, ALTER, and DROP.
- **Data Control Language (DCL)** — DCL statements control the security permissions for users and database objects. Some objects have different permission sets. You can grant or deny these permissions to a specific user or users who belong to a database role or Windows user group.
 - Examples include GRANT, REVOKE, and DENY.
- **Data Manipulation Language (DML)** — DML statements are used to work with data. This includes statements to retrieve data, insert rows into a table, modify values, and delete rows.
 - Examples include SELECT, INSERT, UPDATE, and DELETE.

Data Definition

- Data Types
 - Each database column has to have a valid data type
 - Data types are only partially standardized
 - Numbers:
 - TinyInt, Int, BigInt, Numeric, Decimal, Float, Real, money
 - Characters
 - Char(length), Nchar, Varchar, Nvarchar
 - Character Large Object (CLOB)
 - Varchar(Max), Nvarchar(Max), Text, Ntext
 - Separated data pages
 - Large Object (LOB)
 - Varbinary(max), Varbinary, Binary, Image
 - Stream operations

Data Types (cont'd)

- Large Value Types (LVT)
 - Varchar(max), Nvarchar(max), or Varbinary(max)
 - Used with parameters
- Date and Time
 - Smalldatetime, datetime, date, time, timestamp
- XML Data Type
 - store complete XML documents or well-formed XML fragments
 - Object-Relational Database Management System (ORDBMS)
- Table, uniqueidentifier , cursor, sql_variant Data Types
- User-Defined Type

Create Tables

- Defines the structure of the table

```
CREATE TABLE Divisions  
(DivisionID INT NOT NULL,  
DivisionName Varchar(40) NOT NULL);
```

Alter Tables

- Modifies the structure of the table

ALTER TABLE Divisions

ADD DivisionCity Varchar(40) NOT NULL;

or **ALTER COLUMN** DivisionNameVarchar(80)
NOT NULL;


or **DROP COLUMN** DivisionCityVarchar(40)
NOT NULL;

Data Definition (cont'd)

- DROP Tables
 - **DROP TABLE** Divisions;
- Constraints
 - Used to enforce valid data in columns
 - NOT NULL (the column will not allow null values)
 - CHECK (value is checked against constants or other columns)
 - PRIMARY KEY (enforces uniqueness of primary key)
 - UNIQUE (enforces uniqueness of alternate keys)
 - FOREIGN KEY (specifies a relationship between tables)



Data Definition (cont'd)

- Indexes (like a virtual table with pointers to physical table)
 - In general, rows are unsorted
 - Indexes are sorted on the indexed value
 - Indexes are created on a single column or combination of columns
 - NOTE: Null means unknown or missing value, not blank or zero
- 

Data Modification

- INSERT
 - Adds new rows to a table

```
INSERT INTO Departments  
(DivisionID, DepartmentID, DepartmentName)  
VALUES (1, 100, 'Accounting');
```


Data Modification (cont'd)

- UPDATE
 - Modifies the column values in existing rows

UPDATE Employee

SET CurrentSalary = CurrentSalary + 100

WHERE Employee = 4;


Data Modification (cont'd)

- DELETE
 - Removes rows from a table

```
DELETE FROM Employees  
WHERE Employee = 7;
```



TRANSACTIONS

- A set of INSERT/UPDATE/DELETE operations that belong together as a logical unit of work
 - COMMIT (ends the transaction with success and makes updates permanent)
 - ROLLBACK (ends the transaction with failure and undoes the updates)
- 



TRANSACTIONS (cont'd)

BEGIN TRAN T₁;

UPDATE table1 ...;

UPDATE table2 ...;

INSERT INTO table3 ...;



COMMIT TRAN T₁;



SELECT

- SELECT select_list (describes the columns of the result set.)
- [INTO new_table_name] (specifies that the result set is used to create a new table)
- FROM table_list (Contains a list of the tables from which the result set data is retrieved)
- [WHERE search_conditions] (filter that defines the conditions each row in the source tables must meet)
- [GROUP BY group_by_list] (partitions the result set into groups based on the values)
- [HAVING search_conditions] (an additional filter that is applied to the result set)
- [ORDER BY order_list [ASC | DESC]] (defines the order in which the rows in the result set are sorted)

Single Table SELECT

```
SELECT EmployeeID, FirstName, LastName  
, CurrentSalary*12 AS YearlySalary  
FROM Employees  
WHERE EmployeeID = 3;
```

- DISTINCT can be used to suppress duplicates

WHERE

- The WHERE clause specifies a filter condition
- Conditional Operators
 - = <> > < >= <=
 - IN , NOT IN (test for several values)
 - BETWEEN, NOT BETWEEN (intervals)
 - IS NULL, IS NOT NULL
 - LIKE, NOT LIKE (% or _)
 - EXISTS, NOT EXISTS (sub queries)
- Conditions can be combined with NOT AND OR


ORDER BY

- The ORDER BY statement defines the sort order of the rows

```
SELECT LastName,FirstName,CurrentSalary
FROM Employees
ORDER BY CurrentSalary DESC
        ,LastName ASC
        ,FirstName ASC;
```

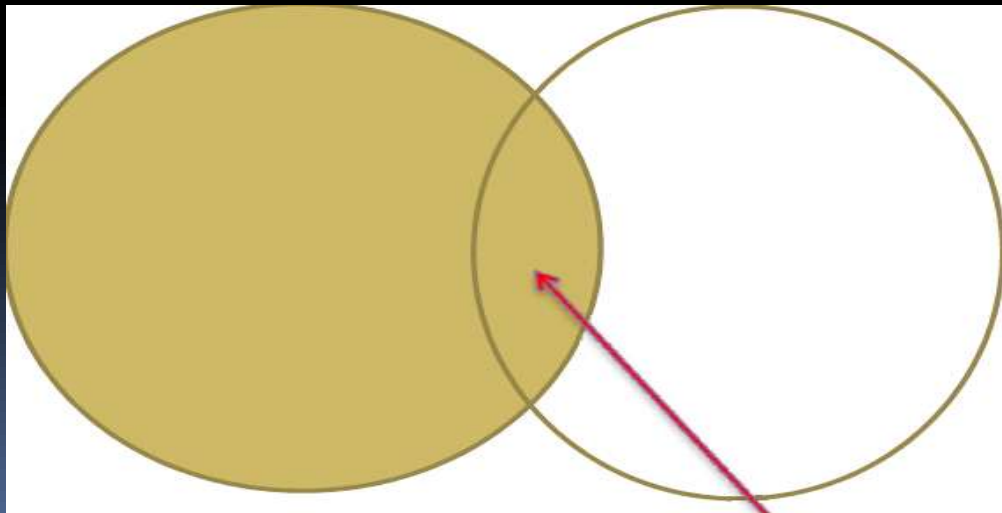



NULL

- NULL is not the same as blank
 - NULL is not the same as zero
 - NULL is not the same as the text 'NULL'
 - NULL is not equal to any value
 - NULL is not different to any value
 - NULL is not equal to NULL
 - In SQL Server, NULL sorts as the lowest value
- 

Joins (INNER)

- Joins are used to combine information from multiple tables
- Basic Syntax of an INNER Join (excludes data that does NOT satisfy the join)



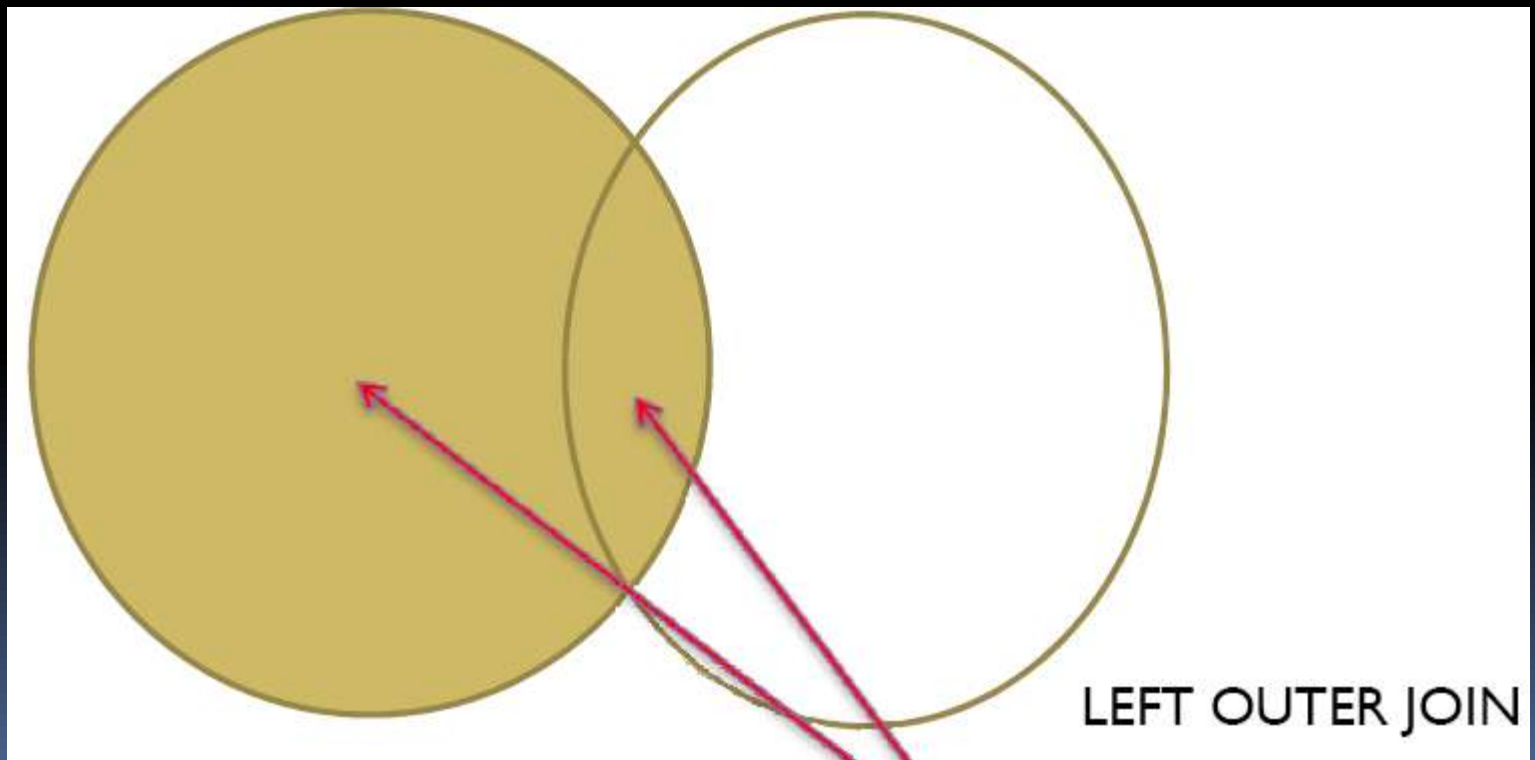
Joins (INNER) (cont'd)

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

- Inner joins are default, so the word Inner can be omitted

Joins (OUTER)

- Basic Syntax of an OUTER Join (include rows from one table that have no matching row)



Joins (OUTER) (cont'd)

- table1 LEFT OUTER JOIN table2
 - all rows from table 1 will be included
- table1 RIGHT OUTER JOIN table2
 - all rows from table 2 will be included
- table1 FULL OUTER JOIN table2
 - all rows from each table will be included

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

Outer Join Example

- **SELECT** region.region_nbr, region.region_name,
branch.branch_nbr, branch.branch_name
FROM region
LEFT OUTER JOIN branch
ON branch.region_nbr = region.region_nbr
ORDER BY region.region_nbr

Table: REGION

region_nbr	region_name
100	East Region
200	Central Region
300	Virtual Region
400	West Region

Table: BRANCH

branch_nbr	branch_name	region_nbr	employee_count
108	New York	100	10
110	Boston	100	6
212	Chicago	200	5
404	San Diego	400	6
415	San Jose	400	3

Outer Join Example (cont'd)

- Here is the result. Note that the "Virtual Region" is included in the results even though it has no rows in the **branch** table. This is the difference between the INNER JOIN and OUTER JOIN.

region_nbr	region_name	branch_nbr	branch_name
100	East Region	108	New York
100	East Region	110	Boston
200	Central Region	212	Chicago
300	Virtual Region	NULL	NULL
400	West Region	404	San Diego
400	West Region	415	San Jose

Joins (CROSS) Cartesian Product

- Each row in one table is paired to every row in the other table

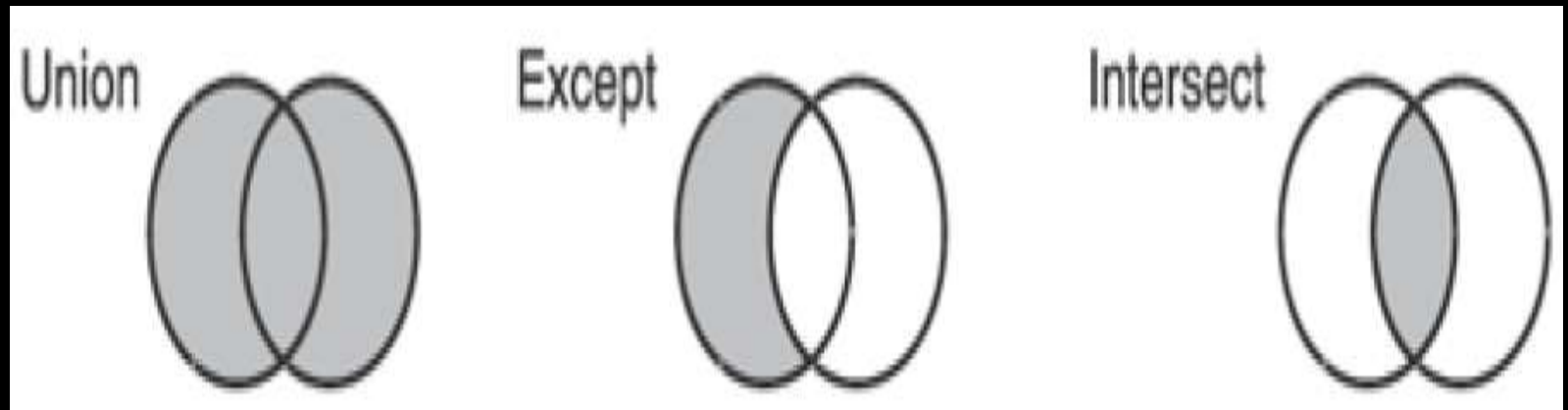
SELECT * FROM employee

CROSS JOIN department

A	I	A	I
B	2	A	2
	3	A	3
		B	1
		B	2
		B	3

Set Operators

- UNION, INTERSECT and EXCEPT



UNION

```
SELECT zip FROM hotel.customer  
WHERE zip > '90000'  
UNION [ALL]  
SELECT zip FROM hotel.hotel  
WHERE zip > '90000'
```

- Creates one table with rows from both SELECTs
- Suppresses duplicate rows



INTERSECT

```
SELECT zip FROM hotel.customer  
WHERE zip < '30000'
```

```
INTERSECT
```

```
SELECT zip FROM hotel.hotel  
WHERE zip < '30000'
```

- 
- Returns the rows that occur in both queries

EXCEPT

```
SELECT zip FROM hotel.hotel  
WHERE zip < '30000'
```


EXCEPT

```
SELECT zip FROM hotel.customer  
WHERE zip < '30000'
```

- Returns the rows from one query except those that occur in the other basically a minus



Row Functions

- Row functions take input parameters and return a value
 - Math Functions
 - String Functions
 - Date and Time Functions
 - Data Type Conversions
 - CASE Statements
 - NULL Functions
- 

Row Functions - Math

- ABS, DEGREES, RAND, ACOS
 - EXP, ROUND, ASIN, LOG, SIN
 - ATN2, LOG10, SQRT, CEILING
 - FLOOR, SIGN, ATAN, PI
 - SQUARE, COS, POWER
 - TAN, COT, RADIANS
-
- `SELECT ROUND(123.9994,3)`
 - Here is the result set. 123.9990

Row Functions - String

- ASCII, NCHAR, SOUNDEX, CHAR, PATINDEX
- SPACE, CHARINDEX, DIFFERENCE, REPLACE
- STUFF, LEFT, REPLICATE, SUBSTRING
- QUOTENAME, STR, LEN, REVERSE
- UNICODE, LOWER, RIGHT
- UPPER, LTRIM, RTRIM
- `SELECT LEFT('abcdefg',2)`
 - Here is the result set. ab

Row Functions – Date & Time

- DATEADD, DATEDIFF
- DATENAME, DATEPART
- DAY, MONTH, YEAR
- GETDATE, GETUTCDATE

```
SELECT GETDATE()
```

- Here is the result set: July 7 2009 4:20 PM

Row Functions – Data Type

```
SELECT CAST( 'abc' AS varchar(5) )
```

```
SELECT
```

```
SUBSTRING(Name, 1, 30) AS ProductName,
```

```
ListPrice
```

```
FROM Production.Product
```

```
WHERE
```

```
CONVERT(int, ListPrice) LIKE '3%';
```

Row Functions - Case

- The CASE expression enables many forms of conditional processing to be placed into a SQL statement.

```
SELECT title, price,  
       Budget = CASE price  
                WHEN price > 20.00 THEN 'Expensive'  
                WHEN price BETWEEN 10.00 AND 19.99 THEN 'Moderate'  
                WHEN price < 10.00 THEN 'Inexpensive'  
                ELSE 'Unknown'  
                END,  
FROM titles
```

Row Functions - Null

- A null value in an expression causes the result of the expression to be null
- Testing for NULL in a WHERE clause
- **ISNULL**(check_expression, replacement_value)
- **NULLIF**(@Value1, @Value2)

```
SELECT Name, Weight  
FROM Production.Product  
WHERE Weight IS NULL;
```

Aggregate Functions

- SUM, AVG, MAX, MIN, COUNT

```
SELECT AVG (UnitPrice * Quantity) As AveragePrice  
FROM WidgetOrders  
WHERE Continent = "North America"
```

```
SELECT COUNT(*) AS 'Number of Large Orders'  
FROM WidgetOrders  
WHERE Quantity > 100
```

```
SELECT MAX(Quantity * UnitPrice) As 'Largest Order'  
FROM WidgetOrders
```

Aggregate Functions – GROUP BY

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

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

Customer	SUM(OrderPrice)
Hansen	2000
Nilsen	1700
Jensen	2000

Aggregate Functions – HAVING

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

SELECT Customer, SUM(OrderPrice)

FROM Orders

GROUP BY Customer

HAVING SUM(OrderPrice)<2000

Customer	SUM(OrderPrice)
Nilsen	1700

Subqueries

- A SELECT statement embedded into another statement is called a subquery

```
SELECT SUM(Sales) FROM Store_Information
WHERE Store_name IN
(SELECT store_name FROM Geography
WHERE region_name = 'West')
```

- IN or NOT IN will handle extra lists returned by the sub query Operators (>, <, =, etc.) can be used when single values are returned

Correlated Subqueries

- If a subquery may reference columns from the main query table, this is called a correlated

```
SELECT DISTINCT c.LastName, c.FirstName  
FROM Person.Contact c
```

```
JOIN HumanResources.Employee e
```

```
ON e.ContactID = c.ContactID
```

```
WHERE 5000.00 IN
```

```
(SELECT Bonus FROM Sales.SalesPerson sp
```

```
WHERE e.EmployeeID = sp.SalesPersonID);
```


Views - Inline

- A view is a virtual table based on the result-set of an SQL statement
- An inline view is a table within the

```
SELECT height
```

```
FROM
```

```
(SELECT height FROM test WHERE id = @id  
ORDER BY id DESC, acc_date DESC, height  
DESC)
```

```
WHERE ROWNUM = 1;
```


Views - Stored

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

- A view does not store data
- If you do insert, update, delete on a view the data values in the underlying tables will be updated
 - This is not possible on all views
 - Computed columns cannot be updated
 - Not permitted on aggregate views or views with set operators
 - Join views may or may not be updateable



Stored Procedures & Functions

- **Precompiled execution.**
 - **Reduced client/server traffic.**
 - **Efficient reuse of code and programming abstraction.**
 - **Centralize maintenance.**
 - **Enhanced security controls.** You can grant users permission to execute a stored procedure independently of underlying table permissions.
- 

Stored Procedures


- `CREATE PROCEDURE` sp_GetInventory
@location varchar(10)
`AS`
`SELECT` Product, Quantity
`FROM` Inventory
`WHERE` Warehouse = @location
- `EXECUTE` sp_GetInventory 'FL'

User Defined Functions

```
CREATE FUNCTION whichContinent
(@Country nvarchar(15))
RETURNS varchar(30)
AS BEGIN
    declare @Return varchar(30)
    select @return = case @Country
    when 'Argentina' then 'South America'
    when 'Belgium' then 'Europe'
    when 'Brazil' then 'South America'
    else 'Unknown'
    End
    return @return
end
```




Stored Procedures VS Functions

- Stored procedures are called independently, using the EXEC command, while functions are called from within another SQL statement.
 - Stored procedure allow you to enhance application security by granting users and applications permission to use stored procedures.
 - Functions must always return a value (either a scalar value or a table). Stored procedures may return a scalar value, a table value or nothing at all.
- 



Triggers

- A special kind of stored procedure that executes automatically when a user attempts the specified data-modification statement on the specified table.
 - Microsoft® SQL Server™ allows the creation of multiple triggers for any given INSERT, UPDATE, or DELETE statement.
- 

Triggers (cont'd)


- **CREATE TRIGGER** *trigger_name*
ON *table*
[WITH ENCRYPTION]
{
 {{ **FOR** | **AFTER** | **INSTEAD OF** }} {{ **[DELETE]** [,] **[INSERT]** [,] **[UPDATE]** }
 [NOT FOR REPLICATION]
 AS
 sql_statement [...n]
 }
 |
 {{ **FOR** | **AFTER** | **INSTEAD OF** }} {{ **[INSERT]** [,] **[UPDATE]** }
 [NOT FOR REPLICATION]
 AS
 { **IF UPDATE** (*column*)
 [{ **AND** | **OR** } **UPDATE** (*column*)]
 [...n]
 | **IF** (**COLUMNS_UPDATED**()) { *bitwise_operator* } *updated_bitmask*
 { *comparison_operator* } *column_bitmask* [...n]
 }
 sql_statement [...n]
 }
}

Triggers (cont'd)

- `CREATE TRIGGER` trigAddStudents
ON Students
FOR INSERT
AS
DECLARE @Newname VARCHAR(100)
SELECT @Newname =(SELECT Name FROM
INSERTED)
PRINT 'THE STUDENT ' + @Newname + ' IS
ADDED.';



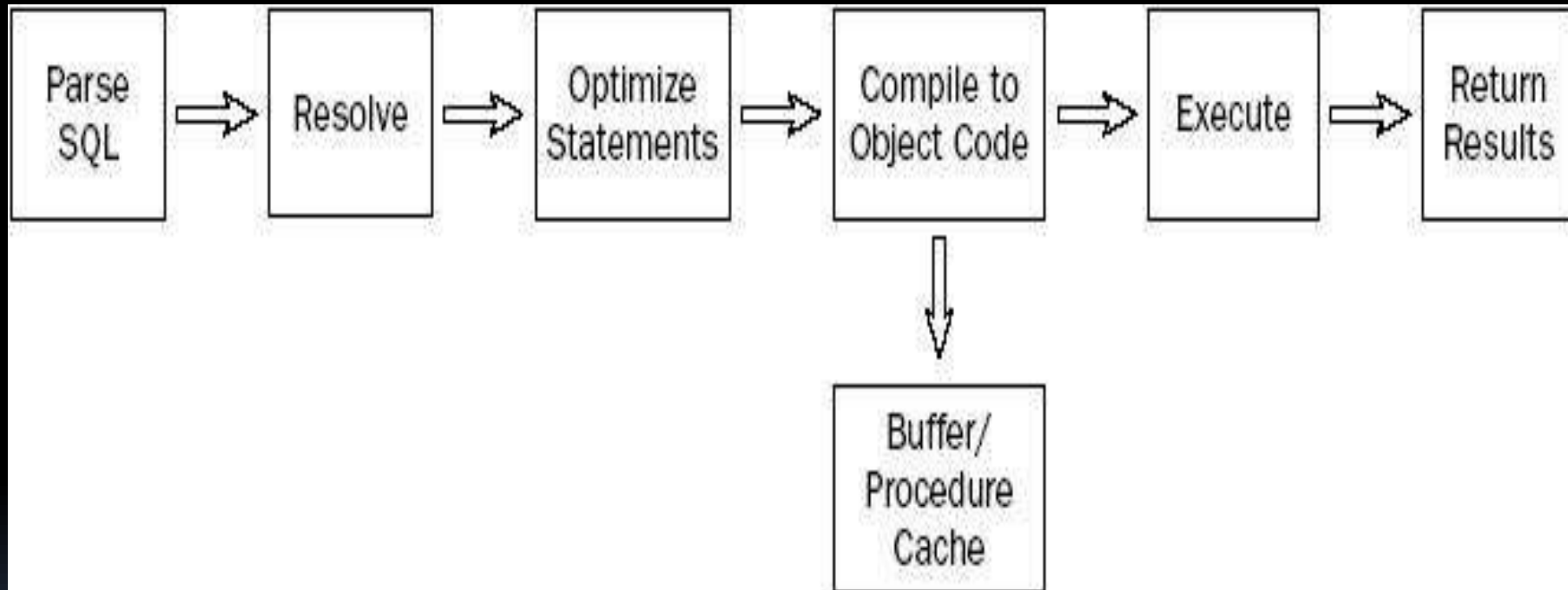
The Mechanics of Query Processing

- Query processor
 - Complex queries are broken down into individual steps / smaller queries
 - This list of steps is known as an *execution plan*.
 - The query's syntax may actually be rewritten by the query optimizer into a standard form of SQL.
 - Before SQL Server can send instructions to the computer's processor, these commands must be compiled into low-level computer instructions, or object code.
- 

Query Processing (cont'd)

- The optimized, compiled query is placed into an in-memory cache. Depending on how the query is created
- View or stored procedure, the execution plan and cache are saved with that object in the database, called *procedure cache*.
- Ad-hoc queries, the cached compiled query and execution plan is held into memory as *buffer cache* and reused until the user and client application's connection is closed.
- If the same query is executed multiple times, it should run faster and more efficiently after the first time

Query Processing (cont'd)





Thanks & Happy Coding