Lesson 6

Introduction to SQL
and
SQL Server
Management Studio

SQL Language

- Non-procedural – specify *what* information required rather than *how* to get it
- Free-format – statements can be typed at any locations on the screen
- Used for data definition (this lesson), data manipulation (Lesson 7-11), and data administration (not covered in this course).

SQL Language

- All information in the database is represented as tables
- Each table consists of a set of rows and columns
- Two types of tables:
  - User tables – contain information that is in the database
  - System tables – contains the database description

Objective of SQL

- Allow users to:
  - Create the database and relation structures
  - Perform data manipulation tasks
    - Inserting
    - Modifying
    - Deleting
  - Perform simple and complex queries
- Language should be portable (80-90%) between database engines.

Objective of SQL

- Designed to use tables (relations) to transform inputs into required outputs
- Two major components:
  - Data Definition Language (DDL)
    - for defining database structure and controlling access to the data
    - Comp 1630 will use SQL Server Management Studio
  - Data Manipulation Language (DML)
    - for retrieving and updating data

Writing Commands

- Reserved words – spelled exactly as required
- User-defined words – defined by user
- Statements – build according to a set of syntax rules
- Not case sensitive

Naming Entities and Attributes

- First character must be one of the following:
  - Letter a-z and A-Z
  - Underscore (_), at sign (@) or number sign (#) symbols
- Subsequent characters can be:
  - Letter a-z and A-Z
  - Decimal numbers
  - Underscore (_), at sign (@) or number sign (#) symbols
- Must not be a SQL reserved word

Naming Entities and Attributes

- Certain symbols at the beginning of an identifier have special meaning
  - Local variable or parameter
    - identifier begins with @
  - Temporary table or procedure
    - identifier begins with #
  - Global temporary object
    - identifier begins with ##
  - Do Not begin identifiers with @@ because used in SQL function names

Databases

- Contain the objects used to represent, manage, and access data
- Collection of tables with data, and other objects, such as views, indexes, stored procedures, and triggers
- Support activities performed with the data
- Data usually related to a particular subject or process

Creating the Database

- Creates physical files that will hold database
- Do not create any user objects in the master database because it contains the system tables
- Authentication
  - Process through which DBMS verifies that only registered users are able to access database

Create Database Example

- Using SQL Server Management Studio user interface:
  - Click on Database
  - Right click New Database
  - On General page, enter Database name
  - Enter Path for the location of the data files and the log files

Data Types

- Describes the type of data the column is allowed to hold
- System supplied

Integers

- Positive or negative whole numbers
  - bigint
    from -2^63 (-9,223,372,036,854,775,808)
    through 2^63-1 (9,223,372,036,854,775,807)
  - int
    from -2^31 (-2,147,483,648)
    through 2^31-1 (2,147,483,647)
  - smallint
    from -2^15 (-32,768) through 2^15-1 (32,767)
  - tinyint
    from 0 through 255
  - bit
    either a 1 or 0 value

Decimal and Numeric

- Positive or negative number with fractional parts
  - decimal

    from -10^38+1 through 10^38-1
  - numeric

    functionally equivalent to decimal

Money and Smallmoney

- Monetary or currency data values
  - money

    from -2^63 (-922,337,203,685,477.5808)
    through 2^63-1 (922,337,203,685,477.5807)

    with accuracy to a 10,000th of a monetary unit
  - smallmoney

    from -214,748.3648 through 214,748.3647

    with accuracy to a 10,000th of a monetary unit

Approximate Numeric

- Floating precision number data
  - float

    from -1.79E+308 through 1.79E+308
  - real

    from -3.40E+38 through 3.40E+38

Datetime and Smalldatetime

- Date and time data
  - datetime

    with accuracy of 3/100ths of a second
  - smalldatetime

    with accuracy of one minute

Character Strings

- Non-Unicode character data
  - char

    fixed-length with a maximum length of 8,000 characters
  - varchar

    variable-length with a maximum length of 2^31 characters
  - text

    variable-length with a maximum length of 2^31-1 (2,147,483,647) characters

Binary Strings

- Binary data stores strings of bits
  - Binary

    fixed-length

    with a maximum length of 8,000 bytes
  - Varbinary

    variable-length

    with a maximum length of 2^31 bytes
  - Image

    Variable-length

    with a maximum length of 2^31-1 bytes

Unique Number and Identifier

- timestamp

  database wide (global) automatically generated unique binary number, with precision down to the nanosecond

- uniqueidentifier

  globally unique identifier (GUID)

User-Defined Data Types

- Based on system supplied data types
- Created by the user for custom data storage
- Can enforce data integrity
- Used when several tables must store the same type of data in a column and you must ensure that these columns have exactly the same data type, length, and null ability

User-Defined Data Types

- Can apply a name to a data type that is more descriptive of the types of values to be held in the object
- Must supply:
  - Name
  - System data type upon which new data type is based
  - Nullability

    whether data type allows null values

User-defined Data Types Example

- Using SQL Server Management Studio:
  - Select the correct database
  - Select Programmability
  - Select Types
  - Click on User-defined Data Types
  - Right click New User Defined Data Type
  - In dialog box, type:
    - Name
    - System supplied Data Type
    - Appropriate settings

Tables

- Used for storage and manipulation of data
- Contains rows and columns
- Each row represents a unique record, and each column represents a field within the record
- Table name must follow the rules for naming identifiers

Tables cont'd

- Must be unique in the database
- Column definitions appear in a comma-separated list enclosed in parentheses
- Order of the column definitions determines the left-to-right order of the columns in the table

Tables cont'd

- Column name
  - unique within the table
- Data type
  - identifies the kind of data the column will store
- Required data
  - specifies if the column requires data (allow NULL values)
- Default value
  - optional value when inserting a row into the table if no value is specified for the column

Create Tables Example

- Using SQL Server Management Studio:
  - Click on correct database
  - Choose Tables
  - Right click New Tables
  - Enter:
    - Column Name
    - Data Type
    - Length of data type if character
    - Required data (Allow Nulls)
    - Default Values (Optional)
    - Appropriate settings

Identity Property

- Can provide a unique, incremental value for a column when a new row is added to the table
- Often used with the PRIMARY KEY constraints to serve as the unique row identifier for the table
- Generates sequential numbers
- Only one column in the table can have the IDENTITY property

Identity Property (cont'd)

- Must be defined as an integer data type
- Cannot update a column with IDENTITY property
- Cannot contain NULL values
- Cannot bind defaults and default constraints to the column

Example

- Using SQL Server Management Studio:
  - Setting when creating table
  - Enter:

    Data Type - integer or decimal

    Required data

    Identity Specification
    - (Is Identity) – Yes
    - Identity Increment – 1
    - Identity Seed – 1000

Constraints (cont'd)

- NOT NULL
  - Specify the column that does not accept NULL values
- CHECK
  - Specifies what data can be entered in a particular column
  - Specifies a Boolean search condition that is applied to all value entered for the column
  - Can specify multiple CHECK constraints for each column

Constraints (cont'd)

- UNIQUE
  - Enforces uniqueness of the values in a set of columns
  - Prevents duplicate values from being entered into a column
- PRIMARY KEY
  - Identify the column or set of columns whose values uniquely identify a row in a table
  - Column cannot be NULL

Constraints (cont'd)

- FOREIGN KEY
  - Identify the relationships between tables
  - Defines a column (or combination of columns) in one table whose values match those of the primary key in some other table
- DEFAULT
  - Defines a value that the system will automatically insert into a column when a value is not given

To Do

- Submit Exercise 6 – Management Studio
- Submit Lesson 6 Review Questions
- Post to Lesson 6 Discussion Topic
- Read Chapter 7
- Study Term Project Document

Lesson 7

SQL - Part 1

- **Select … From … Where**
- **Order By …**
- **Operators**
- **Functions**

Writing Queries

- Each clause in a statement should begin on a new line
- Beginning of each clause should line up with the beginning of other clauses
- If a clause has several parts, they should appear on a separate line and be indented using the start of the clause to show the relationship
- Upper case letters are used to represent RESERVED WORDS
- Lower case letters are used to represent user-defined words

Literals

- Literals are constants used in SQL statements
- Non-numeric data values are enclosed in single quotes e.g. 'Comp1630'
- All numeric data values are not enclosed in quotes

Data Manipulation

- **SELECT**
  - Retrieve and display data from one or more database tables
- **INSERT**
  - Adds new rows of data into a table
- **UPDATE**
  - Modifies existing data in a table
- **DELETE**
  - Removes rows of data from a table

SELECT

- Order of the clauses cannot be changed
- There are two mandatory clauses:

  SELECT and **FROM**
- Output from a query is called the result set

SELECT

SELECT [DISTINCT] {*|*column Expression*
[AS *new name*]}
FROM *table name* [*alias*]
[WHERE *condition*]
[GROUP BY *column list*]
[HAVING *condition*]
[ORDER BY *column List*] [DESC]

Arithmetic Operators

+	Add
-	Subtract
*	Multiply
/	Divide

Comparison Operators

=		Equal to
<>	or !=	Not equal to
<		Less than
>		Greater than
<=		Less than or equal to
>=		Greater than or equal to

Logical Operators

AND

- Combine two search conditions where both must be true

OR

- Combine two search conditions when one or the other must be true

NOT

- Select rows where a search condition is false

Precedence not specified

```
SELECT    *
FROM      employee
WHERE     pub_id = '1389'  OR  pub_id = '9999'
   AND    job_id = '11'
```

Incorrect results

emp_id	fname	minit	lname	job_id	job_lvl	pub_id	hire_date
PSA89086M	Pedro	S	Afonso	14	89	1389	1990-12-24
A-C71970F	Aria		Cruz	10	87	1389	1991-10-26
…							
CGS88322F	Carine	G	Schmitt	13	64	1389	1992-07-07
MAS70474F	Margaret	A	Smith	9	78	1389	1988-09-29

(11 row(s) affected)

Precedence specified

```
SELECT     *
FROM       employee
WHERE      (pub_id = '1389'  OR  pub_id = '9999')
   AND     job_id = '11'
```

Correct results

emp_id	fname	minit	lname	job_id	job_lvl	pub_id	hire_date
PCM98509F	Patricia	C	McKenna	11	150	9999	1989-08-01
MAP77183M	Miguel	A	Paolino	11	112	1389	1992-12-07

(2 row(s) affected)

Aggregate Operations

- **COUNT**
  - Returns the number of rows containing non-null values
- **SUM**
  - Returns the sum of the values in a specified column
- **AVG**
  - Returns the average of the values in a specified column
- **MIN**
  - Returns the smallest value in a specified column
- **MAX**
  - Returns the largest value in a specified column

Aggregate Operations

- Operate on a single column of a table and return a single value
- COUNT, MIN and MAX may be used on both numeric and non-numeric columns
- SUM and AVG may be used on numeric columns only
- Aggregate functions operate on non-null values only, except when COUNT (*) is used to count all rows of a table

Retrieve All Columns

- List all the employees

```
SELECT    *
FROM      employee
```

Results

emp_id	fname	minit	lname	job_id	job_lvl	pub_id	hire_date
PMA42628M	Paolo	M	Accorti	13	35	0877	1992-08-27
PSA89086M	Pedro	S	Afonso	14	89	1389	1990-12-24
VPA30890F	Victoria	P	Ashworth	6	140	0877	1990-09-13
…							

Retrieve Specific Columns

- List all the authors displaying their last name, first name and phone number.

```
SELECT   au_lname, au_fname, phone
FROM     authors
```

Results

au_lname	au_fname	phone
White	Johnson	408 496-7223
Green	Marjorie	415 986-7020
Carson	Cheryl	415 548-7723
…		

Use of DISTINCT

- List the unique store numbers found in the sales table.

```
SELECT   DISTINCT    stor_id
FROM      sales
```

Results

```
stor_id
--------
6380
7066
7067
7131
7896
8042
```

WHERE Clause Types

1. Comparison
   - compare value of one expression to that of another expression
2. Range
   - test if value of an expression falls within a specified range
3. Set membership
   - test if value of an expression equals one in a set of values
4. Pattern matching
   - test whether string matches a specified pattern
5. Null
   - test whether a column has a NULL value

Comparison

- List all the titles with a type of 'business'.

```
SELECT    *
FROM      titles
WHERE     type = 'business'
```

Compound Comparison

- List all the titles which are 'business' types or have a publisher id of '1389'.

```
SELECT    *
FROM      titles
WHERE     type = 'business'
   OR     pub_id = '1389'
```

Special Operators

BETWEEN

- Used to check whether attribute value is within a range

IN

- Used to check whether attribute value matches any value within a value list

LIKE

- Used to check whether attribute value matches given string pattern

IS NULL

- Used to check whether attribute value is null

BETWEEN

- List all the sales with an order date between September 1, 1994 and September 30,1994.

```
SELECT *
FROM    sales
WHERE  ord_date  BETWEEN  'Sep 1 1994' AND 'Sep 30 1994'
```

is the same as

```
SELECT *
FROM    sales
WHERE  ord_date >= 'Sep 1 1994' AND  ord_date <= 'Sep 30 1994'
```

IN

- List the title id, title, and type from the titles table where the type is mod_cook or trad_cook.

```
SELECT  title_id, title, type
FROM    titles
WHERE   type  IN  ( 'mod_cook', 'trad_cook' )
```

is the same as

```
SELECT  title_id, title, type
FROM    titles
WHERE   ( type = 'mod_cook'  OR  type = 'trad_cook' )
```

LIKE

- Selects rows containing fields that match specified portions of character strings
- Used with char, varchar, text, datetime and smalldatetime data

% Any string of zero or more characters

_ Any single character

[ ] Any single character within the specified range

Example: **[A-Z]**

[^] Any single character not within the specified range using the caret symbol

Example: **[^A-Z]**

LIKE Example

- List all the books that begin with 'Computer' in their title.

```
SELECT    title_id, title
FROM      titles
WHERE     title  LIKE  'Computer%'
```

Results

title_id	title
PS1392	Computer Phobic …

IS NULL

- List all the books where a price has not been supplied displaying the title id and title.

```
SELECT    title_id, title
FROM      titles
WHERE     price  IS NULL
```

Results

title_id	title
MC3026	The Psychology of Computer Cooking
PC9999	Net Etiquette

ORDER BY

- Rows of an SQL query result table are not arranged in any particular order
- Use the ORDER BY clause to ensure the results of a query are sorted
- Column identifier may be either by:
  - Column name (recommended)
  - Column number

    Identifies an element in the SELECT list by its position within the list with 1 as the first (left-most) element in the list, 2 as the second element in the list, etc.

Example

- List author info ordering the result set by last name.

```
SELECT      au_id, au_lname, au_fname
FROM        authors
ORDER BY    au_lname
```

is the same as

```
SELECT      au_id, au_lname, au_fname
FROM        authors
ORDER BY    2
```

Column Headings

- Column headings in the result set can be changed from the default attribute name.
- If a 'space' is required use quotes.

```
SELECT  fname       AS  FirstName,
        lname       AS  LastName,
        hire_date   AS  'Hire Date'
FROM    employee
```

Results

FirstName	LastName	Hire Date
Paolo	Accorti	1992-08-27 00:00:00.000
Pedro	Afonso	1990-12-24 00:00:00.000
Victoria	Ashworth	1990-09-13 00:00:00.000
…		

Calculated Columns

- List the monthly sales for all the books displaying the title id, title, and monthly sales calculation. Note the missing column heading.

```
SELECT    title_id,
          title,
          ( ytd_sales / 12 )
FROM      titles
```

Results

title_id	title	
BU1032	The Busy Executive's Database Guide	341
BU1111	Cooking with Computers: Surreptitious Balance Sheets	323
BU2075	You Can Combat Computer Stress!	1560

…

Example

- To add a column name for a calculated field use 'AS'

```
SELECT      title_id,
            title,
          ( ytd_sales / 12 )   AS   monthly
FROM        titles
```

Results

title_id	title	monthly
BU1032	The Busy Executive's Database Guide	341
BU1111	Cooking with Computers: Surreptitious Balance Sheets	323
BU2075	You Can Combat Computer Stress!	1560

…

Data Conversion

- Converting an expression of one data type to another.

```
SELECT title_id,
        CONVERT ( CHAR(12), pubdate, 106 )   AS PubDate1,
        FORMAT   ( pubdate, 'dd MMM yyyy' )  AS PubDate2,
        CONVERT ( INT, (ytd_sales / price) )  AS Copies
FROM    titles
WHERE  title_id = 'BU1032'
```

Results

Title_id	PubDate1	PubDate2	Copies
BU1032	12 Jun 1991	12 Jun 1991	205

Date Formats

YY	YYYY	Standard	Format
0	100	Default	mon dd yyyy hh:mi (AM/PM)
1	101	USA	mm/dd/yyyy
2	102	ANSI	yyyy.mm.dd
3	103	British/French	dd/mm/yyyy
4	104	German	dd.mm.yyyy
5	105	Italian	dd-mm-yyyy
6	106	-	dd MMM yyyy (least ambiguous)
7	107	-	Mon dd, yyyy
8	108	-	hh:mm:ss
9	109	-	mon dd yyyy hh:mi:ss:mmm (AM/PM)
10	110	USA	mm-dd-yyyy
11	111	JAPAN	yyyy/mm/dd
12	112	ISO	yyyymmdd
14	114	-	hh:mi:ss:mmm(24h)

Functions

- Date
- Mathematical
- String
- System

Date Functions

- Performs an operation on a date and time input value and return a string, numeric, or date and time value
- Deterministic functions always return the same result any time they are called with a specific set of input values.
- Nondeterministic functions may return different results each time they are called with a specific set of input values.

Function	Determinism
DATEADD	Deterministic
DATEDIFF	Deterministic
DATENAME	Nondeterministic
DATEPART	Deterministic except when used as DATEPART (dw, date). dw, the weekday datepart, depends on the value set by SET DATEFIRST, which sets the first day of the week.
DAY	Deterministic
GETDATE	Nondeterministic
GETUTCDATE	Nondeterministic
MONTH	Deterministic
YEAR	Deterministic

DATEPART

Datepart	Abbreviations
year	yy, yyyy
month	mm, m
day	dd, d
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
dayofyear	dy, y
week	wk, ww
quarter	qq, q

DATEADD

- Returns a new datetime value based on adding an interval to the specified date
- Result is a datetime value equal to the date plus the number of date parts
- If the date parameter is a smalldatetime value, the result is also a smalldatetime

Example

```
SELECT    title_id,
          pubdate,
          DATEADD (DAY, 30, pubdate)  AS  new_date
FROM      titles
WHERE     pubdate >= 'MAY 1 2000'
```

Results

title_id	pubdate	new_date
MC3026	2000-08-06	2000-09-05
PC9999	2000-08-06	2000-09-05

(2 row(s) affected)

DATEDIFF

- Returns the number of date and time boundaries crossed between two specified dates
- Method of counting crossed boundaries makes the result given by DATEDIFF consistent across all data types such as minutes, seconds and milliseconds

Example

```
SELECT      title_id,
            pubdate,
            DATEDIFF ( MONTH, pubdate,GETDATE())
                  AS months
FROM        titles
WHERE       pubdate >= 'MAY 1 2000'
```

Results

title_id	pubdate	months
MC3026	2000-08-06	98
PC9999	2000-08-06	98

(2 row(s) affected)

Mathematical Functions

- Performs a calculation based on input values provided as parameters to the function, and returns a numeric value

ABS	DEGREES	RAND
ACOS	EXP	ROUND
ASIN	FLOOR	SIGN
ATAN	LOG	SIN
ATN2	LOG10	SQUARE
CEILING	PI	SQRT
COS	POWER	TAN
COT	RADIANS	

Example

- ROUND - Returns a numeric expression, rounded to the specified length or precision

```
SELECT     ROUND ( 5678.156, 2 )
```

Results

```
--------
5678.16
```

Example

- SQRT - Returns the square root of a given expression

SELECT SQRT (81)

Results

9.0

String Functions

- Performs an operation on a string (CHAR or VARCHAR) input value and returns a string or numeric value.

ASCII	NCHAR	SOUNDEX
CHAR	PATINDEX	SPACE
CHARINDEX	REPLACE	STR
DIFFERENCE	QUOTENAME	STUFF
LEFT	REPLICATE	SUBSTRING
LEN	REVERSE	UNICODE
LOWER	RIGHT	UPPER
LTRIM	RTRIM	

Example

- CONCATENATION - Use '+' to create one result from multiple character columns

```
SELECT    (au_lname + ', ' + au_fname) AS Name
FROM      authors
```

Results

```
Name
-----------------------------------
Bennet, Abraham
Blotchet-Halls, Reginald
Carson, Cheryl
…..
```

Example

- SUBSTRING - Returns part of a character, binary text, or image expression. Specifies start position and length.

```
SELECT   au_lname,
         SUBSTRING ( au_fname, 1, 1 )  AS initial
FROM     authors
```

Results

au_lname	initial
Bennet	A
Blotchet-Halls	R
Carson	C
…..	

Example

- CHARINDEX - Returns the starting position of the specified expression in the string

e.g. the expression 'wonderful' in the attribute 'notes'

```
SELECT  title_id, notes,
        CHARINDEX ( 'wonderful', notes )  AS pos
FROM    titles
WHERE   title_id = 'TC3218'
```

Results

title_id	notes	pos
TC3218	Profusely illustrated in color, this makes a wonderful …	46

Example

- UPPER - Returns a character expression with lowercase character data converted to uppercase.

```
SELECT    UPPER (RTRIM (au_lname)) + ', ' + au_fname
FROM      authors
```

Results

```
----------------------------------------
BENNET, Abraham
BLOTCHET-HALLS, Reginald
CARSON, Cheryl
……
```

System Functions

- Performs operations and returns information about values, objects, and settings in SQL

Function	Determinism
APP_NAME	Nondeterministic
CASE expression	Deterministic
CAST and CONVERT	Deterministic unless used with **datetime**, **smalldatetime**, or **sql_variant**.
COALESCE	Deterministic
COLLATIONPROPERTY	Nondeterministic
CURRENT_TIMESTAMP	Nondeterministic
CURRENT_USER	Nondeterministic
DATALENGTH	Deterministic
@@ERROR	Nondeterministic
fn_helpcollations	Deterministic
fn_servershareddrives	Nondeterministic
fn_virtualfilestats	Nondeterministic
FORMATMESSAGE	Nondeterministic
GETANSINULL	Nondeterministic
HOST_ID	Nondeterministic
HOST_NAME	Nondeterministic

System Functions

IDENT_CURRENT	Nondeterministic
IDENT_INCR	Nondeterministic
IDENT_SEED	Nondeterministic
@@IDENTITY	Nondeterministic
IDENTITY (Function)	Nondeterministic
ISDATE	Deterministic only if used with the CONVERT function, the CONVERT style parameter is specified and the style parameter is not equal to 0, 100, 9, or 109. Styles 0 and 100 use the default format mon dd yyyy hh:miAM (or PM). Styles 9 and 109 use the default format plus milliseconds mon dd yyyy hh:mi:ss:mmmAM (or PM).
ISNULL	Deterministic
ISNUMERIC	Deterministic
NEWID	Nondeterministic
NULLIF	Deterministic
PARSENAME	Deterministic
PERMISSIONS	Nondeterministic
@@ROWCOUNT	Nondeterministic
ROWCOUNT_BIG	Nondeterministic
SCOPE_IDENTITY	Nondeterministic
SERVERPROPERTY	Nondeterministic
SESSIONPROPERTY	Nondeterministic
SESSION_USER	Nondeterministic
STATS_DATE	Nondeterministic
SYSTEM_USER	Nondeterministic
@@TRANCOUNT	Nondeterministic
USER_NAME	Nondeterministic

Example

- DATALENGTH - Returns the number of bytes used to represent any expression

```
SELECT  DATALENGTH ( pub_name ) AS length,
        pub_name
FROM    publishers
```

Results

length	pub_name
14	New Moon Books
16	Binnet & Hardley
20	Algodata Infosystems
…	

Example

@@ROWCOUNT - Returns the number of rows affected by the last statement

```
SELECT      title_id, title
FROM        titles
WHERE       title_id = 'MC2222‘
IF @@ROWCOUNT  >  0   PRINT   'Row found'
```

Results

```
title_id             title
-----------          ----------------------------------------
MC2222               Silicon Valley Gastronomic Treats

(1 row(s) affected)
Row found
```

Example

- ISNULL - Replaces NULL with the specified replacement value (e.g. 10.00)

```
SELECT    AVG ( ISNULL (price, 10.00) )
FROM      titles
```

Results

```
------------
14.2366
```

To Do

- Submit Exercise 7 – SQL
- Submit Lesson 7 Review Questions
- Post to Lesson 7 Discussion Topic
- Review Chapter 7
- Start Term Project – Part B

Lesson 8

SQL – Part 2

- **Group by … Having**
- **Joins**
- **Unions**
- **Insert/Update/Delete**

GROUP BY

- Used to create one output row for each group
- Usually used with aggregates
- Produces summary values for selected columns

Example

- Find the average price
  for each book type in the titles table

```
SELECT    type,
          AVG (price) AS average
FROM      titles
GROUP BY  type
```

Results

type	average
business	13.7300
mod_cook	11.4900
popular_comp	21.4750
psychology	13.5040
trad_cook	15.9633
UNDECIDED	NULL

GROUP BY

- Every item in the GROUP BY list MUST appear in the SELECT list
- Can form groups within groups
- Separate grouping elements with commas
- If the grouping contains more than one null value, all of the null values are put into a single group

Example

- List the number of types for each pub id

```
SELECT      pub_id,
            COUNT( type ) AS total
FROM        titles
GROUP BY    pub_id
```

Results

pub_id	total
0736	5
0877	7
1389	6

Example

- List the number of books belonging to each type within each publisher.

```
SELECT    pub_id,
          type,
          COUNT (type) AS total
FROM      titles
GROUP BY  pub_id,
          type
```

Results

pub_id	type	total
0736	business	1
1389	business	3
0877	mod_cook	2
...		

Example

- List everything in the discounts table.

```
SELECT    *
FROM      discounts
```

Results

discounttype	stor_id	lowqty	highqty	discount
Initial Customer	NULL	NULL	NULL	10.50
Volume Discount	NULL	100	1000	6.70
Customer Discount	8042	NULL	NULL	5.00

(3 row(s) affected)

Example

- Counts only non nulls for the column

```
SELECT      stor_id,
            COUNT (stor_id) AS total
FROM        discounts
GROUP BY    stor_id
```

Results

stor_id	total
NULL	0
8042	1

GROUP BY

- Similar to DISTINCT if used without aggregates
- Divides a table into groups and returns one row for each group
- Each item in the select list produces a single value per set

Example

- List the unique pub ids in the titles table.

```
SELECT      pub_id
FROM        titles
GROUP BY    pub_id
```

```
SELECT  DISTINCT  pub_id
FROM      titles
```

both obtain the same results

Results

```
pub_id
-------
0736
0877
1389
```

GROUP BY

- Use WHERE clause to specify which rows participate in the aggregate calculations

- Columns in the WHERE clause DO NOT have to be in the SELECT or GROUP BY list

Example

- Returns the rows with an advance greater than 5000

```
SELECT      type,
            AVG (price) AS average
FROM        titles
WHERE       advance > 5000
GROUP BY    type
```

Results

Type	average
business	2.9900
mod_cook	2.9900
popular_comp	21.4750
psychology	14.2950
trad_cook	17.9700

HAVING

- Is like a WHERE clause for groups
- Follows the GROUP BY clause
- Can include aggregates
- Each element in the HAVING clause MUST appear in the SELECT list

Example

- Count the types and eliminate those that include only one book.

```
SELECT      type,
            COUNT (title) AS total
FROM        titles
GROUP BY    type
HAVING      COUNT (title) > 1
```

Results

type	total
business	4
mod_cook	2
popular_comp	3
psychology	5
trad_cook	3

HAVING

- Can have more than one condition included in the HAVING clause
- Combine conditions with logical operators AND, OR, and NOT

Example

- Grouping by publisher, select titles with a price of more than $5 and
- publishers with identification numbers greater than 0800
- paid more than $15000 in advances and
- whose books average less than $20 in price. Order the result set by pub id.

```
SELECT    pub_id,
          SUM (advance)   AS total,
          AVG (price)     AS average
FROM      titles
WHERE     price    > 5.00  AND
          pub_id > '0800'
GROUP BY  pub_id
HAVING    SUM (advance) > 15000  AND
          AVG (price) < 20.00
ORDER BY  pub_id
```

Results

pub_id	total	average
0877	26000	17.8940
1389	30000	18.9760

WITH ROLLUP

- Used in SELECT statements with aggregate functions to generate summary values
- Part of a GROUP BY clause
- Summary values appear as new rows in the result set

Example

List the type, advance, and advance sum for the types 'business' and 'mod_cook'. Also the subtotals by book type and the final total of the advance sum.

```
SELECT   type, advance,
         SUM(advance) AS total
FROM     titles
WHERE    type IN ( 'business', 'mod_cook' )
GROUP  BY type, advance WITH ROLLUP
```

Results

Results

type	advance	total	
business	5000.00	15000.00	
business	10125.00	10125.00	
business	NULL	25125.00	(Subtotal)
mod_cook	0.00	0.00	
mod_cook	15000.00	15000.00	
mod_cook	NULL	15000.00	(Subtotal)
NULL	NULL	40125.00	(Final Total)

Joining Tables

- Join operation lets you retrieve and manipulate data from more than one table in a single SELECT statement
- Two types of joins:
  - INNER join
  - OUTER joins
- Specify joins following the FROM clause
- Identical column names must be qualified with their table names in the SELECT list and WHERE clause

Joining Tables

- Can join more than two tables but you must join tables two at a time.

  e.g. 3 tables require 2 joins

  4 tables require 3 joins

- A join statement:
  - Specifies a column from each table
  - Compares the values in those columns row by row
  - Combines rows with qualifying values into new rows

INNER JOIN

- Displays only the rows that have a match in both joined tables (the table name alias is shown in green).

Example: (ANSI standard)

```
SELECT      a.au_fname, a.au_lname, t.royalty, t.title
FROM        authors  a
INNER JOIN  titleauthor  ta  ON a.au_id     = ta.au_id
INNER JOIN  titles t            ON ta.title_id = t.title_id
WHERE       a.au_lname  = 'Ringer'
    AND     a.au_fname LIKE 'A%‘
ORDER BY    a.au_fname
```

INNER JOIN

- Example: (traditional)

```
SELECT  a.au_fname, a.au_lname, t.royalty, t.title
FROM    authors a,
        titleauthor ta,
        titles t
WHERE   a.au_id = ta.au_id
    AND ta.title_id = t.title_id
    AND a.au_lname = 'Ringer'
    AND a.au_fname LIKE 'A%‘
ORDER BY a.au_fname
```

Results

au_fname	au_lname	royalty	title
Albert	Ringer	12	Is Anger the Enemy?
Albert	Ringer	10	Life Without Fear
Anne	Ringer	24	The Gourmet Microwave
Anne	Ringer	12	Is Anger the Enemy?

OUTER JOIN

- Returns not only matching rows, but also rows with unmatched attribute values for one, or the other table, or both.
- Three types:
  - LEFT outer join
  - RIGHT outer join
  - FULL outer join
- Use only for special purpose since they are not as efficient as inner joins

LEFT OUTER JOIN

- All rows from the left (or 1st mentioned) table are included and the output column from the other table are set to NULL

```
SELECT  p.pub_name, t.title
FROM    publishers p
LEFT OUTER JOIN   titles t  ON p.pub_id = t.pub_id
```

Results

pub_name	title
New Moon Books	You Can Combat Computer Stress!
New Moon Books	Is Anger the Enemy?
New Moon Books	Life Without Fear
New Moon Books	Prolonged Data Deprivation: Four Case Studies
New Moon Books	Emotional Security: A New Algorithm
Binnet & Hardley	Silicon Valley Gastronomic Treats
Binnet & Hardley	The Gourmet Microwave
Binnet & Hardley	The Psychology of Computer Cooking
Binnet & Hardley	Computer Phobic AND Non-Phobic Individuals: Behavior
Binnet & Hardley	Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean
Binnet & Hardley	Fifty Years in Buckingham Palace Kitchens
Binnet & Hardley	Sushi, Anyone?
Algodata Infosystems	The Busy Executive's Database Guide
Algodata Infosystems	Cooking with Computers: Surreptitious Balance Sheets
Algodata Infosystems	Straight Talk About Computers
Algodata Infosystems	But Is It User Friendly?
Algodata Infosystems	Secrets of Silicon Valley
Algodata Infosystems	Net Etiquette
Five Lakes Publishing	NULL
Ramona Publishers	NULL
GGG&G	NULL
Scootney Books	NULL
Lucerne Publishing	NULL

RIGHT OUTER JOIN

- All rows from the right table are included and output column from the other table are set to NULL

```
SELECT      t.title_id, a.au_lname, a.au_fname
FROM        titleauthor t
RIGHT OUTER JOIN  authors a ON  t.au_id = a.au_id
ORDER BY t.title_id
```

Results

title_id	au_lname	au_fname
NULL	Greene	Morningstar
NULL	McBadden	Heather
NULL	Smith	Meander
NULL	Stringer	Dirk
BU1032	Bennet	Abraham
BU1032	Green	Marjorie
BU1111	MacFeather	Stearns
BU1111	O'Leary	Michael
BU2075	Green	Marjorie
BU7832	Straight	Dean
MC2222	del Castillo	Innes
MC3021	DeFrance	Michel
MC3021	Ringer	Anne
PC1035	Carson	Cheryl
PC8888	Dull	Ann
PC8888	Hunter	Sheryl
PC9999	Locksley	Charlene
PS1372	MacFeather	Stearns
PS1372	Karsen	Livia
PS2091	Ringer	Anne
PS2091	Ringer	Albert
PS2106	Ringer	Albert
PS3333	White	Johnson
PS7777	Locksley	Charlene
TC3218	Panteley	Sylvia
TC4203	Blotchet-Halls	Reginald
TC7777	Gringlesby	Burt
TC7777	Yokomoto	Akiko
TC7777	O'Leary	Michael

FULL OUTER JOIN

- If a row from either table does not match the selection criteria, specifies the rows be included in the results set and its output columns that correspond to the other table be set to NULL

```
SELECT a.au_fname, a.au_lname, p.pub_name
FROM    authors a
FULL OUTER JOIN publishers p ON a.city = p.city
ORDER BY p.pub_name, a.au_lname, a.au_fname
```

Results

au_fname	au_lname	pub_name
Reginald	Blotchet-Halls	NULL
Michel	DeFrance	NULL
Innes	del Castillo	NULL
Ann	Dull	NULL
Marjorie	Green	NULL
Morningstar	Greene	NULL
Burt	Gringlesby	NULL
Sheryl	Hunter	NULL
Livia	Karsen	NULL
Charlene	Locksley	NULL
Stearns	MacFeather	NULL
Heather	McBadden	NULL
Michael	O'Leary	NULL
Sylvia	Panteley	NULL
Albert	Ringer	NULL
Anne	Ringer	NULL
Meander	Smith	NULL
Dean	Straight	NULL
Dirk	Stringer	NULL
Johnson	White	NULL
Akiko	Yokomoto	NULL
Abraham	Bennet	Algodata Infosystems
Cheryl	Carson	Algodata Infosystems
NULL	NULL	Binnet & Hardley
NULL	NULL	Five Lakes Publishing
NULL	NULL	GGG&G
NULL	NULL	Lucerne Publishing
NULL	NULL	New Moon Books
NULL	NULL	Ramona Publishers
NULL	NULL	Scootney Books

Recursion

- Recursion involves joining a table to itself
- This requires two different aliases in the FROM and JOIN clauses for the same table

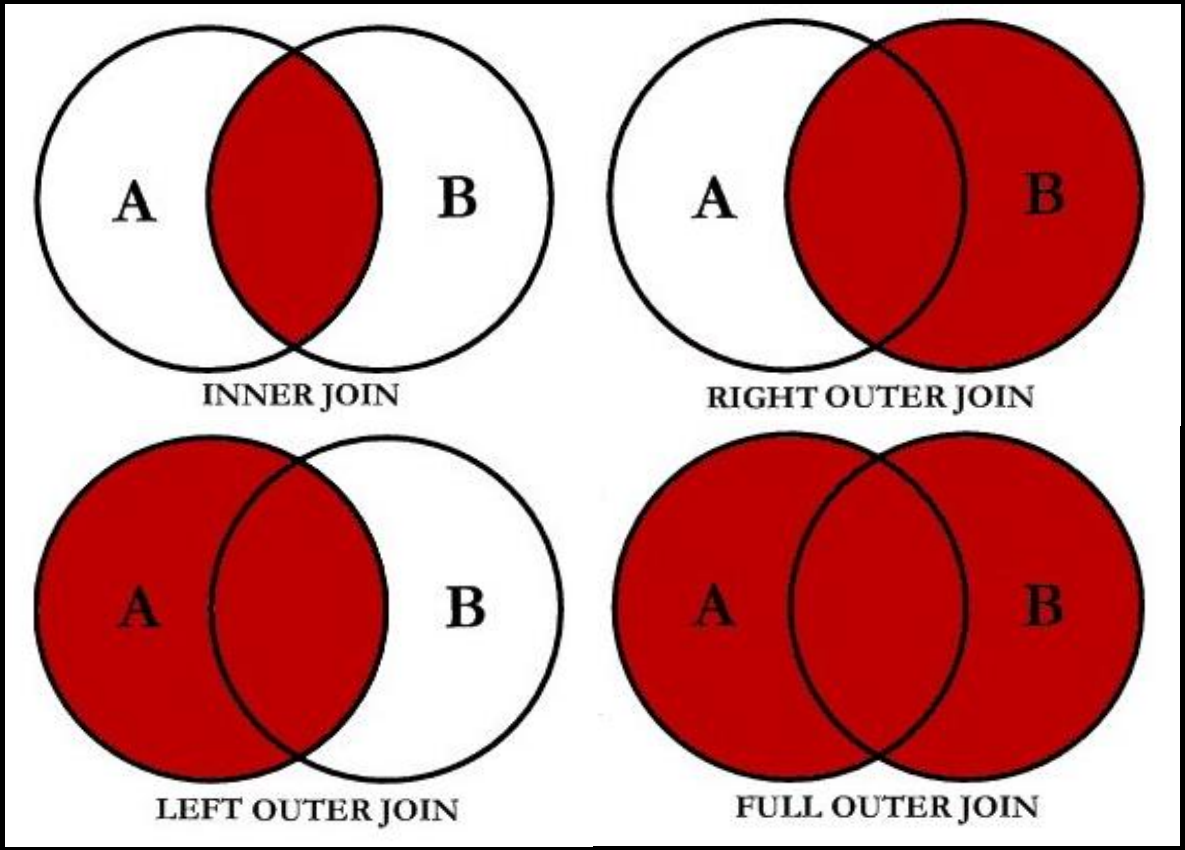
Example

```
SELECT      a.au_id         AS AuthorID,
            a.au_lname      AS LastName,
            a.au_fname      AS FirstName,
            a.state         AS State,
            a.zip           AS Zip
FROM        authors a
JOIN        authors a2      ON a.au_id = a2.au_id
WHERE       a.state   = 'UT'
AND         a.zip     = a2.zip
ORDER BY    a.au_lname, a.au_fname
```

Results

AuthorID	LastName	FirstName	State	Zip
998-72-3567	Ringer	Albert	UT	84152
899-46-2035	Ringer	Anne	UT	84152

Efficiency of Joins



UNIONS

- Combine similar data from multiple tables into one result set
- Select lists MUST contain the same number of columns
- Data types of corresponding columns should be compatible
- Result set uses the column names from the first select list

UNIONS

- Each SELECT statement can have its own WHERE clause
- Can only have one ORDER BY clause, and it must be in the last SELECT statement
- Removes duplicate rows from the result set
- UNION ALL retains duplicates

Example

- List all the authors and publishers who live in the city of Berkeley.

```
SELECT    (au_fname + ' ' + au_lname)   AS Name,
          city                          AS City
FROM      authors
WHERE     city = 'Berkeley'
UNION
SELECT    pub_name,
          city
FROM      publishers
WHERE     city = 'Berkeley‘
ORDER BY Name
```

Results

Name	City
Abraham Bennet	Berkeley
Algodata Infosystems	Berkeley
Cheryl Carson	Berkeley

- Use UNION to display different values for one field depending on what is in another

Example

```
SELECT      '20% off ‘      AS Discount,
            title           AS Title,
            price           AS Price,
            (price * .80)   AS NewPrice
FROM        titles
WHERE       price < 7.00
UNION
SELECT      '10% off ',
            title,
            price,
            (price * .90)
FROM        titles
WHERE       price BETWEEN 7.00 AND 15.00
ORDER BY    title
```

Results

Discount	Title	Price	NewPrice
10% off	Cooking with Computers: Surreptitious Balance Sheets	11.9500	10.755000
10% off	Emotional Security: A New Algorithm	7.9900	7.191000
10% off	Fifty Years in Buckingham Palace Kitchens	11.9500	10.755000
10% off	Is Anger the Enemy?	10.9500	9.855000
10% off	Life Without Fear	7.0000	6.300000
10% off	Sushi, Anyone?	14.9900	13.491000
20% off	The Gourmet Microwave	2.9900	2.392000
20% off	You Can Combat Computer Stress!	2.9900	2.392000

INSERT

- Enter data into a specified table
- If the column names are not used, must type the data values in the same order as the column names in the CREATE TABLE statement
- If specifying the column names, can list the column names in any order as long as the data values and the column names match
- If column allows null values, system will enter null values in the columns where no data value is specified

INSERT

- Attribute entries are separated by commas
- Columns with the IDENTITY (auto-number) property must not be explicitly listed in the column list or value clause
- If an INSERT statement violates a constraint or rule, or if it is the wrong data type, the statement fails and SQL Server displays an error message

Examples

- Add a new employee (not recommended)

```
INSERT INTO  employee
VALUES ( 123, 'Mary', 'S', 'Smith', 3, 37, '0877', 'Sep 25 2005' )
```

- Add a new employee, who has not been assigned a job, job level or publisher id

```
INSERT INTO employee ( emp_id, fname, minit, lname, hire_date )
VALUES ( 127, 'John', 'T', 'Jones', 'Jan 2 2006' )
```

- Copy all rows in employee table to a new table

```
INSERT INTO  new_employee
SELECT       *
FROM         employee
```

UPDATE

- Modify data in a table
- Change values in a single row, group of rows or all the rows in a table
- Specify the rows you want to change and the new value
- SET clause specifies the column and the value

UPDATE

- WHERE clause determines which rows will be updated
- If a search condition is not specified, all the rows in the table in the specified columns will be updated with the values in the SET clause
- Can use computed column values

Example

- Update employee 'H-B39728F' with job id of 13 and job level of 35

```
UPDATE employee
SET        job_id = 13,
           job_lvl = 35
WHERE   emp_id = 'H-B39728F'
```

- Increase all year to date sales by 10% for business book types

```
UPDATE titles
SET        ytd_sales = ROUND ((ytd_sales * 1.10), 0)
WHERE   type = 'business'
```

Example

- Update the year-to-date sales in the titles table by adding the quantity from the sales table to it.

```
UPDATE      titles
SET         t.ytd_sales = (t.ytd_sales + s.qty)
FROM        titles t
INNER JOIN  sales s  ON  t.title_id = s.title_id
```

DELETE

- Delete a table row
- Can delete rows based on data in other tables
- WHERE clause specifies which rows to remove
- Warning - If no search condition is used, ALL rows from the table will be deleted. The table itself is not removed but it is now empty

Example

- Delete employee 'H-B39728F'

```
DELETE FROM     employee
WHERE           emp_id = 'H-B39728F'
```

- Delete ALL employees rows from the employee table

```
DELETE FROM     employee
```

Example

- Delete all the rows from the titleauthor table where the titles contain the word 'computers'.

```
DELETE  titleauthor
FROM    titleauthor ta
INNER JOIN titles t  ON  ta.title_id = t.title_id
WHERE   t.title  LIKE  '%computers%'
```

Batches

- A group of one or more SQL statements sent at one time from an application to SQL Server for execution
- SQL Server compiles the statements of the batch into a single executable unit called an execution plan
- Statements in the execution plan are executed one at a time
- Can be more efficient then submitting statements separately because network traffic is often reduced
- A table cannot be altered and then the new columns referenced in the same batch

Example (DDL)

```
CREATE TYPE key_id FROM int NOT NULL
GO          -- Signals the end of the batch

CREATE TABLE person
(   person_id       key_id,
    last_name       varchar(30) NOT NULL,
    first_name      varchar(15) NOT NULL,
    birth_date      datetime NOT NULL  )
GO          -- Signals the end of the batch
```

SELECT INTO

- Creates a new table and inserts the result set from the query into the new table
- Example:

```
SELECT    *
INTO      titles_new
FROM      titles
WHERE     pub_id = '1389'
```

SELECT INTO

- Creates an identical table definition, with a different table name, with no data by having a FALSE condition in the WHERE clause

```
SELECT    *
INTO      authors_new
FROM      authors
WHERE     1 = 2
```

SELECT INTO

- Creates a new temporary table, and inserts the result set from the query into the new table.

```
SELECT    *
INTO      #temp_titles
FROM      titles
WHERE     pub_id = '1389'
```

To Do

- Submit Exercise 8 – SQL
- Submit Lesson 8 Review Questions
- Post to Lesson 8 Discussion Topic
- Read Chapter 8
- Complete Term Project – Part B

Lesson 9

SQL – Part 3

- **CASE**
- **Views**
- **Subqueries**
- **Indexes**

CASE Statement

- Similar to the IF-THEN-ELSE conditional construct
- Evaluates a list of conditions and returns one of multiple possible results
- If evaluated condition is TRUE, the value of the CASE expression is the result expression
- Only one value will be the result expression

CASE Statements

Two formats:

1. Simple

   compares an expression to a set of simple expressions to determine the result

2. Searched

   valuates a set of Boolean expressions to determine the result

ELSE argument is optional in both cases

Syntax - Simple

```
CASE       input_expression
   WHEN    when_expression   THEN  result_expression
    [ ...n ]
    [
    ELSE     else_result_expression
    ]
END
```

Simple CASE Example

```
SELECT          title_id                AS      TitleID,
    CASE        type
          WHEN  'popular_comp'  THEN    'Popular Computing'
          WHEN  'mod_cook'      THEN    'Modern Cooking'
          WHEN  'business'      THEN    'Business'
          WHEN  'psychology'    THEN    'Psychology'
          WHEN  'trad_cook'     THEN    'Traditional Cooking'
          ELSE   'Not yet categorized'
    END                         AS      Type,
                price           AS      Price
FROM            titles
ORDER BY        price
```

Results

TitleID	Type	Price
MC3026	Not yet categorized	NULL
PC9999	Popular Computing	NULL
MC3021	Modern Cooking	2.99
BU2075	Business	2.99
PS2106	Psychology	7.00
PS7777	Psychology	7.99
PS2091	Psychology	10.95
TC4203	Traditional Cooking	11.95
BU1111	Business	11.95
TC7777	Traditional Cooking	14.99
PS3333	Psychology	19.99
BU1032	Business	19.99
BU7832	Business	19.99
MC2222	Modern Cooking	19.99
PC8888	Popular Computing	20.00
TC3218	Traditional Cooking	20.95
PS1372	Psychology	21.59
PC1035	Popular Computing	22.95

Syntax - Searched

```
CASE
    WHEN       Boolean_expression    THEN result_expression
    [ ...n ]
    [
    ELSE       else_result_expression
    ]
END
```

Searched CASE Example

```
SELECT          title_id                                AS  TitleID,
    CASE
        WHEN    price  IS NULL                          THEN   'Not yet priced'
        WHEN    price < 10                              THEN   'Low Priced Title'
        WHEN    price >= 10  AND  price < 20            THEN   'Medium Priced Title'
        ELSE    'High Priced Title'
     END                                                AS  PriceCategory,
                price                                   AS  Price
FROM            titles
ORDER BY        price
```

Results

TitleID	PriceCategory	Price
MC3026	Not yet priced	NULL
PC9999	Not yet priced	NULL
MC3021	Low Priced Title	2.99
BU2075	Low Priced Title	2.99
PS2106	Low Priced Title	7.00
PS7777	Low Priced Title	7.99
PS2091	Medium Priced Title	10.95
TC4203	Medium Priced Title	11.95
BU1111	Medium Priced Title	11.95
TC7777	Medium Priced Title	14.99
PS3333	Medium Priced Title	19.99
BU1032	Medium Priced Title	19.99
BU7832	Medium Priced Title	19.99
MC2222	Medium Priced Title	19.99
PC8888	High Priced Title	20.00
TC3218	High Priced Title	20.95
PS1372	High Priced Title	21.59
PC1035	High Priced Title	22.95

VIEWS

- Virtual table based on a SELECT query
- Does not exist as a table (virtual) and does not generate a copy of the data
- Produced when the view runs
- Easy way to examine and handle just the data needed
- When you query a view, it looks exactly like any other database table
- Can derive a view from another view

Example

- *Create a view that displays the names of authors, who live in Oakland, and their books*

```
CREATE VIEW oaklanders
AS
SELECT      a.au_fname, a.au_lname, a.city, t.title
FROM        authors a
INNER JOIN  titleauthor ta  ON a.au_id     = ta.au_id
INNER JOIN  titles t        ON ta.title_id = t.title_id
WHERE       a.city = 'Oakland'
```

Example

- To run view

```
SELECT      *
FROM        oaklanders
ORDER BY    au_lname
```

Results

au_fname	au_lname	city	title
Marjorie	Green	Oakland	The Busy Executive's Database Guide
Marjorie	Green	Oakland	You Can Combat Computer Stress!
Livia	Karsen	Oakland	Computer Phobic AND Non-Phobic Individuals
Stearns	MacFeather	Oakland	Cooking with Computers
Stearns	MacFeather	Oakland	Computer Phobic AND Non-Phobic Individuals
Dean	Straight	Oakland	Straight Talk About Computers

VIEWS

- To remove views

DROP VIEW oaklanders

- If any of the tables, views or columns that underlie a view have been dropped or renamed, you cannot run this view
- If columns are added to the underlying tables of the view, the new columns will not appear in the view if defined with the SELECT * clause

VIEWS

- If view columns are not assigned names in the CREATE VIEW clause, the names are inherited from the columns of the underlying tables
- To choose new names: 1) put them inside parentheses following the view name, separated by commas, or 2) assign the new names in the SELECT statement

VIEWS

- New view column names are required when:
  - One or more of the columns in the view are derived from an arithmetic expression, a build-in function, or a constant
  - The view would have more than one column of the same name

Example 1

```
CREATE VIEW   currentinfo
     (PubId, Type, Income, AvgPrice, AvgSales)
AS
SELECT   pub_id,
              type,
              SUM (price * ytd_sales),
              AVG (price),
              AVG (ytd_sales)
FROM      titles
GROUP BY  pub_id,
                 type
```

Example 2

```
CREATE VIEW   currentinfo
AS
SELECT   pub_id                          AS PubId,
         type                            AS Type,
         SUM (price * ytd_sales)         AS Income,
         AVG (price)                     AS AvgPrice,
         AVG (ytd_sales)                 AS AvgSales
FROM     titles
GROUP BY  pub_id,
          type
```

Example

- To run view

```
SELECT      PubId, AvgSales
FROM        currentinfo
ORDER BY    PubId
```

Results

PubId	AvgSales
0736	18722
0736	2391
0877	12139
0877	375
0877	6522
0877	NULL
1389	4022
1389	6437

VIEWS

- Views can be used to provide a level of security
- Table *columns* may be hidden by not including them in the view query and *rows* may be hidden by including a WHERE clause
- To modify the definition of a view use ALTER or DROP the view and CREATE it again with the new definition
- The ALTER command keeps the permissions

Sub-queries

- Return results from an inner query to an outer clause
- Many statements that include sub-queries can also be formulated as joins
- Sub-queries are either:
  - Non-correlated
  - Correlated

Non-correlated

- Evaluated from the inside out
- Example:

```
SELECT    pub_name
FROM      publishers
WHERE     pub_id      IN
        ( SELECT    pub_id
          FROM      titles
          WHERE     type = 'business' )
```

Non-correlated

Inner query is independent and gets evaluated first.

1. Inner query returns id numbers of those publishers that have published business books

2. These values are substituted into the outer query which finds names that go with the identification number in the publishers table

Correlated

- Outer statements provide values for the inner sub-query to use in its evaluation and then the sub-query results are passed back to the outer query

- Example:

```
SELECT    p.pub_name
FROM      publishers p
WHERE     'business'    IN
        ( SELECT   t.type
          FROM     titles t
          WHERE    t.pub_id = p.pub_id )
```

Correlated

- Inner query needs values from the outer query and passes the results to the outer query
- Inner query cannot be evaluated independently
- References the outer query and is executed once for each row in the outer query

Sub-queries

- Example written without sub-queries using join

```
SELECT DISTINCT   p.pub_name
FROM              publishers p
INNER JOIN        titles t   ON  p.pub_id = t.pub_id
WHERE             t.type = 'business'
```

Advantage of Sub-queries

- Calculate an aggregate value adhoc and feed it back to the outer query for comparison

Example

List all the books with prices equal to the minimum book price

Without using sub-query

```
SELECT MIN (price)
FROM   titles
```

Results

```
---------
2.99
```

```
SELECT  title
FROM    titles
WHERE   price = 2.99
```

With using sub-query

```
SELECT   title, price
FROM     titles
WHERE    price  IN
   ( SELECT MIN (price)
     FROM   titles )
```

Results

title	price
You Can Combat Computer Stress!	2.99
The Gourmet Microwave	2.99

Example

- *Find the names of authors who have participated in writing at least one popular computing book*

```
SELECT      a.au_lname, a.au_fname
FROM        authors a
WHERE       a.au_id   IN
          ( SELECT    ta.au_id
            FROM      titleauthor ta
            WHERE     ta.title_id   IN
                    ( SELECT    t.title_id
                      FROM      titles t
                      WHERE     t.type = 'popular_comp' ) )
```

Advantage of Joins

- Can display information from both tables
- With a sub-query, can only display information from the outer join
- JOINs remove the join conditions from the WHERE clause for clarity

Example Using No Subqueries

```
SELECT      a.au_lname, a.au_fname
FROM        authors a
   INNER JOIN   titleauthor ta   ON a.au_id    = ta.au_id
   INNER JOIN   titles t         ON ta.title_id = t.title_id
WHERE       t.type = 'popular_comp'
```

Results

au_lname	au_fname
Carson	Cheryl
Dull	Ann
Locksley	Charlene
Hunter	Sheryl

Subqueries that are an Existence Test

- EXISTS keyword in a WHERE clause tests for the existence of rows
- Keyword EXISTS is not preceded by a column name, constant or other expression
- SELECT list of a subquery introduced by EXISTS almost always consist of an asterisk because you are only testing for the existence or nonexistence of any rows that meet the criteria

Example

- *Find the names of publishers who publish business books*

```
SELECT DISTINCT  p.pub_name
FROM    publishers p
WHERE EXISTS
    ( SELECT   *
      FROM      titles t
      WHERE    t.pub_id = p.pub_id
          AND    t.type = 'business' )
```

Results

```
pub_name
------------------------------
Algodata Infosystems
New Moon Books
```

Example

- Find the names of publishers who DO NOT publish business books

```
SELECT DISTINCT p.pub_name
FROM    publishers p
WHERE NOT EXISTS
   ( SELECT     *
     FROM       titles t
     WHERE      t.pub_id = p.pub_id
          AND   t.type = 'business' )
```

Results

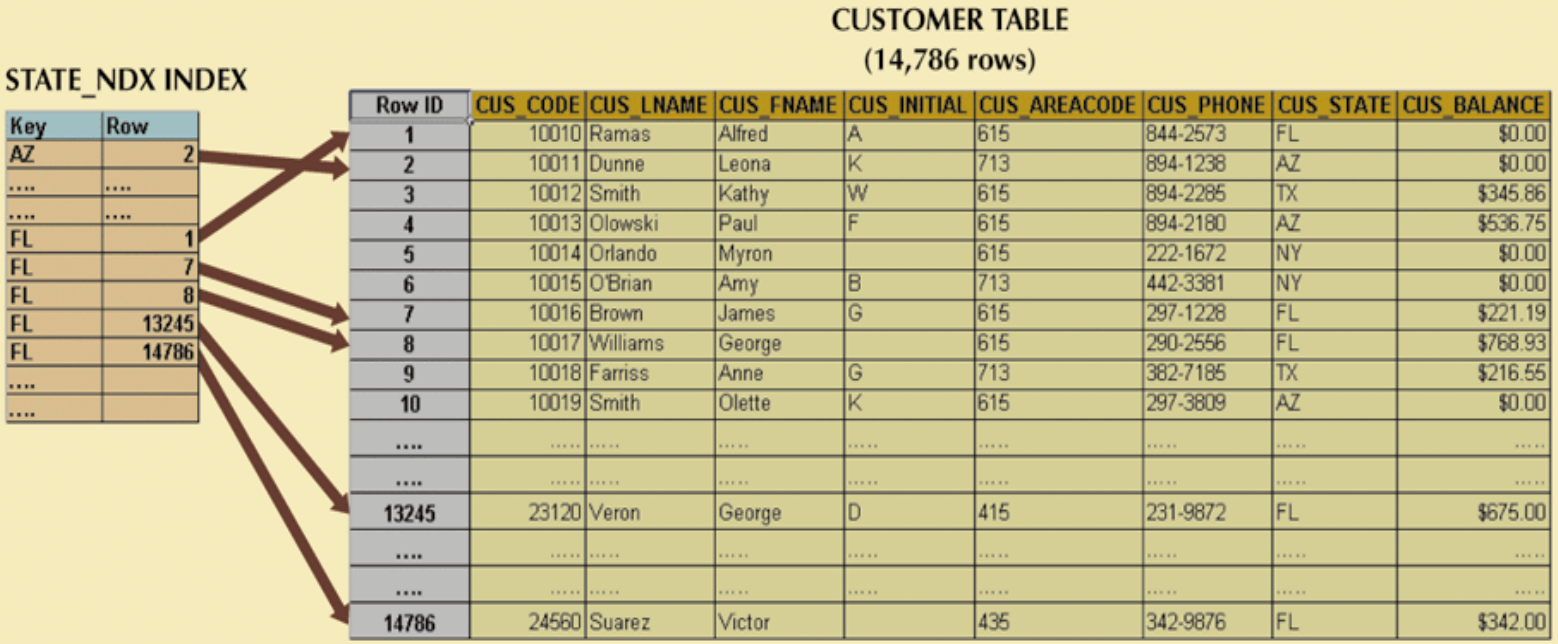
```
pub_name
------------------------------
Binnet & Hardley
Five Lakes Publishing
GGG&G
Lucerne Publishing
Ramona Publishers
Scootney Books
```

Indexes

- A table structure that SQL Server uses to provide fast access to rows of a table based on the values of one or more columns
- Contains data values and pointers to the rows where those values occur
- When no indexes are available, SQL Server must perform a sequential table scan, reading every data page in the table
- More efficient to use indexes to access table than to scan all rows in table sequentially

Indexes

FIGURE 11.3 Index representation for the CUSTOMER table



STATE_NDX INDEX

Key	Row
AZ	2
....	....
....	....
FL	1
FL	7
FL	8
FL	13245
FL	14786
....	
....	

CUSTOMER TABLE
(14,786 rows)

Row ID	CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_STATE	CUS_BALANCE
1	10010	Ramas	Alfred	A	615	844-2573	FL	$0.00
2	10011	Dunne	Leona	K	713	894-1238	AZ	$0.00
3	10012	Smith	Kathy	W	615	894-2285	TX	$345.86
4	10013	Olowski	Paul	F	615	894-2180	AZ	$536.75
5	10014	Orlando	Myron		615	222-1672	NY	$0.00
6	10015	O'Brian	Amy	B	713	442-3381	NY	$0.00
7	10016	Brown	James	G	615	297-1228	FL	$221.19
8	10017	Williams	George		615	290-2556	FL	$768.93
9	10018	Farriss	Anne	G	713	382-7185	TX	$216.55
10	10019	Smith	Olette	K	615	297-3809	AZ	$0.00
....	.....	.....	.....	.....	.....	.....	.....	.....
....	.....	.....	.....	.....	.....	.....	.....	.....
13245	23120	Veron	George	D	415	231-9872	FL	$675.00
....	.....	.....	.....	.....	.....	.....	.....	.....
....	.....	.....	.....	.....	.....	.....	.....	.....
14786	24560	Suarez	Victor		435	342-9876	FL	$342.00

Indexes

- Transparent to users
- Can be added, changed or dropped without affecting the database schema
- Tables can have no index or more than one index
- Can create indexes after there is data in a table

Indexes

- When a primary key is declared, DBMS automatically creates a clustered unique index
- Appropriate to put indexes on columns frequently used in retrievals
- Usually index primary key columns and columns used in joins and sorts
- System determines if and how the index is used for each query

Indexes

CREATE [UNIQUE]
[CLUSTERED | NONCLUSTERED]
INDEX *index name*
ON *table name* (*column name*)

Examples

```
CREATE    INDEX       aulnameind
          ON          authors (au_lname)

CREATE    UNIQUE
          CLUSTERED
          INDEX       auidind
          ON          authors (au_id)
```

Indexes

- Use the sp_helpindex system stored procedure to find information about the indexes on a table

- Example:

  Find the indexes on a sales table.

```
sp_helpindex      sales
```

Types of Indexes

Clustered Index

- Sorted not only logically but physically

Non-clustered Index

- Rows in index are stored in the order of the index key values but data rows are not guaranteed to be in any particular order

Composite Index

- Involves more than one column

Unique Index

- Prevents duplicates rows of data

Clustered Index

- Physical order of table rows is the same as that of the index
- DBMS will re-sort the table rows on an ongoing basis to maintain them in the same order as the index
- Can only have one clustered index per table

Clustered Index

- Usually placed on the column that is:
  - most often retrieved on
  - not be frequently updated
  - accessed in sequence by a range of values
  - used with the GROUP BY or ORDER BY
  - used in joins, such as foreign key columns

```
CREATE CLUSTERED INDEX  titleidind
ON     titles (title_id)
```

Non-clustered Index

- Order of the index does not match the physical order of the rows on disk
- Slower than clustered indexes

```
CREATE INDEX  aulname
ON     authors (au_lname)
```

Composite Index

- Used when two or more columns are best searched as a unit because of their logical relationship
- Must specify all the columns
- Columns do not have to be in the same order as the columns in the CREATE TABLE statement
- Start with the name of the column you use most often in the search

```
CREATE INDEX  aunameind
ON     authors (au_lname, au_fname)
```

Indexes

- Overhead involved in maintenance and use of indexes needs to be balanced against performance improvement gained when retrieving data
- Do not index every column because:
  - Building and maintaining an index takes time and storage space on the database device
  - Inserting, deleting or updating data in indexed columns takes longer than in non indexed columns
- Indexes are not useful when:
  - Columns are rarely referenced in queries
  - Columns only have 2 different values
  - Table is small and contains few rows

Drop Indexes

- Removes one or more indexes from the current database
- If an index has a PRIMARY KEY or a UNIQUE constraint, they must be removed prior to dropping the index

- Example:

```
DROP INDEX      authors aulnameind
```

To Do

- Submit Exercise 9 – SQL
- Submit Lesson 9 Review Questions
- Post to Lesson 9 Discussion Topic
- Review Chapter 8
- Complete Project – Part C

Lesson 10

SQL - Part 4

- **Variables**
- **Sequence Control**
- **Stored Procedures**
- **Triggers**

Variables

- Variables can be either:

  LOCAL

  defined with the DECLARE statement

  GLOBAL

  defined and maintained by SQL Server

Local Variables

- Names are preceded by the @ sign
- Cannot be defined as text or image
- Values are assigned with the SET statement or the SELECT statement
- Often used in a batch or stored procedure

Example

- Create a variable, place a string value into the variable, and display it.

```
DECLARE  @testvar      char(20)

SET      @testvar = 'Comp 1630'

SELECT   @testvar
```

Results

```
----------
Comp 1630
```

Example

- Create a local variable and use it in a SELECT statement to find the authors who live in the state equal to the local variable

```
DECLARE  @state  char(2)
SET      @state = 'CA'
SELECT   au_fname + ' ' + au_lname AS name
FROM     authors
WHERE    state = @state
```

Example

- Use a query to assign a value to a variable

```
DECLARE     @rows   int
SET         @rows =
            ( SELECT COUNT ( au_id )
              FROM authors )
SELECT      @rows
```

Results

```
-----------
23
```

Example

- Use a SELECT statement to assign values to one or more variables
- A variable is assigned the value 'Comp 1630'. The query is run against the publishers table with no rows returned because 1877 does not exist. Note that the variable retains the original value.

```
DECLARE    @variable    char(20)
SET        @variable = 'Comp 1630'
SELECT     @variable = pub_name
FROM       publishers
WHERE      pub_id = '1877'
SELECT     @variable
```

Results

```
pub_name
--------------------
Comp 1630
```

Example

- A variable is assigned the value 'Comp 1630'.
- The value 1877 requested for pub_id does not exists in the publishers table. The sub-query returns no value, therefore, the variable is set to NULL.

```
DECLARE    @variable   char(20)
SET        @variable = 'Comp 1630'
SELECT     @variable =
           ( SELECT     pub_name
             FROM       publishers
            WHERE       pub_id = '1877' )
SELECT     @variable
```

Results

```
 pub_name
--------------------
NULL
```

Global Variables

- Predefined global variables can be used without being declared
- Two @@ signs precede their names
- Many global variables report on the system activity since the last time SQL Server was started, or report information about a connection

Example

@@ ROWCOUNT returns the number of rows the last SQL statement returned

```
SELECT      pub_name
FROM        publishers
WHERE       pub_id = '1877'

IF   @@ROWCOUNT > 0
     PRINT    'FOUND RECORDS'
ELSE
     PRINT    'FOUND NO RECORDS '
```

Results

```
pub_name
------------------------------------
FOUND NO RECORDS
```

Control-of-Flow

- Control-of-flow language controls the flow of execution of SQL statements
- Keywords can be used in ad hoc SQL statements, in batches and in stored procedures

Control-of-Flow

Keyword	Description
BEGIN... END	Defines a statement block
CASE expression	Allows an expression to have conditional return values
DECLARE Statement	Declares local variables
IF... ELSE	Conditional, and optionally, alternate execution when FALSE
PRINT Statement	Displays a user-defined message on the screen
Comments	-- for line comment /* together with */ for comment block
GOTO label	Continues processing at the statement following the label
RAISERROR Statement	Returns a system message entry or a dynamically built message
RETURN	Exits unconditionally
WAITFOR	Sets a delay for statement execution
WHILE	Repeats statements while a specific condition is TRUE
... BREAK	Exits the innermost WHILE loop
... CONTINUE	Restarts a WHILE loop

Example

```
IF ( SELECT AVG (price) FROM titles ) < 15
   BEGIN
        UPDATE    titles
        SET       price = price * 2
   END
ELSE
   BEGIN
        PRINT     'ERROR'
        PRINT     ‘No update performed'
   END
```

Stored Procedures

- Saved collection of SQL statements that can take and return user-supplied parameters
- Procedures can be created for permanent or temporary use within a session
- Stored procedures can be scheduled to run at a specified day and time

Stored Procedures

- Can be:
  - SIMPLE

    One SELECT statement

  - COMPLEX

    Multiple SELECT statements
    using control-of-flow statements

Stored Procedures

- Benefits:
  - Speed

    Reduces network traffic and increases performance
  - Share application logic

    Stored procedures can be incorporated with other stored procedures
  - Security

    Give EXECUTE rights only with no table access

Stored Procedures

- Create using the CREATE PROCEDURE statement
- CREATE PROCEDURE statements cannot be combined with other SQL statements in a single batch
- Objects referenced must exist when executing the procedure

Stored Procedures

- Specify any input and output parameters to the calling procedure before the AS phrase
- Specify the programming statements that perform operations in the database, including calling other stored procedures
- When executed for the first time, the procedure is compiled to determine an optimal access plan to retrieve the data

Stored Procedures

- Must define:
  - Name of the stored procedure
  - Names and data types of its parameters
  - Names and data types of any local variables used by the procedure
  - Sequence of statements executed when the procedure is called

Stored Procedures

- To call a stored procedure:

  EXECUTE *procedure_name* [(*parameters*)]

- To drop a procedure when no longer needed:

  DROP PROCEDURE *procedure_name*

Example

```
CREATE PROCEDURE   sp_DisplayName
(   @parameter       varchar(30)  )
AS
        SELECT     @parameter
GO
```

To run stored procedure:

```
EXEC    sp_DisplayName   'Comp 1630'
```

Example

```
CREATE PROCEDURE   sp_Emp_Info
(       @parameter1         varchar(30),
        @parameter2         varchar(20)          )
AS
   SELECT  emp_id, fname, lname, job_id, hire_date
   FROM     employee
   WHERE  lname  = @parameter1
        AND  fname  = @parameter2
GO
```

To run stored procedure:

```
EXEC   sp_Emp_Info   'Devon', 'Ann'
EXEC   sp_Emp_Info   @parameter2 = 'Ann', @parameter1 = 'Devon'
```

Example

```
CREATE PROCEDURE      sp_USA_Publishers
AS
DECLARE          @count          int
DECLARE          @message        varchar(50)

SELECT           @count = COUNT( pub_id )
FROM             publishers
WHERE            country = 'USA'

IF   @count > 0
     SET   @ message = CONVERT ( CHAR(2), @count ) + ' publishers in the USA'
ELSE
     SET   @ message = 'No publishers in the USA'
SELECT @ message
GO
```

To run stored procedure:

```
EXEC     sp_USA_Publishers
```

Results

6 publishers in the USA

Example

```
CREATE PROCEDURE    sp_Average
(           @parameter1     smallint,
            @parameter2     smallint,
            @parameter3     smallint,
            @parameter4     smallint    OUTPUT      )
AS
SELECT @parameter4 = ((@parameter1 + @parameter2 + @parameter3) / 3)
GO
```

To run stored procedure:

```
DECLARE         @Average  smallint          -- to contain result
EXECUTE         sp_average    5, 3, 7,
                              @Average   OUTPUT
SELECT          'The average is ', @Average
```

Results

The average is	5

Example

```
CREATE PROCEDURE   sp_Author_Info
(    @parameter1    varchar (30) = NULL,
     @parameter2    varchar (20) = NULL     )
AS
BEGIN
     IF      @parameter1  IS NULL  AND  @parameter2  IS NULL
             PRINT  'Please provide name.'
     ELSE
             SELECT         a.au_lname, a.au_fname, t.title, p.pub_name
             FROM           authors a
             INNER JOIN  titleauthor  ta   ON  a.au_id = ta.au_id
             INNER JOIN  titles t          ON  t.title_id = ta.title_id
             INNER JOIN  publishers p      ON  t.pub_id = p.pub_id
             WHERE          a.au_lname     LIKE @parameter1
                AND         a.au_fname     LIKE @parameter2
END
GO
```

Example

```
EXEC  sp_Author_Info  Hunter, Sheryl

au_lname             au_fname    title                          pub_name
-------------------- ----------- ------------------------------ --------------
Hunter               Sheryl      Secrets of Silicon Valley      Algodata Infosystems

(1 row(s) affected)
```

Triggers

- Special type of stored procedure bound to a specific table
- fires when a DML operation is attempted (such as an INSERT, UPDATE, or DELETE)
- Often created to enforce referential integrity or consistency among logically related data in different tables
- Any errors in the transaction or in the trigger will roll back the entire transaction
- Can check data before and/or after a transaction has been made
- Fires once per statement; not once per row

Trigger Types

AFTER Trigger

- Fires after a the DML operation is carried out
- Most common type of trigger
- Used with INSERT, DELETE or UPDATE statements
- Requires ROLLBACK when error occurs

INSTEAD OF Trigger

- Fires instead of a DML operation
- Executes something other than the triggering action
- Used with INSERT, DELETE or UPDATE statements
- Does not require ROLLBACK when an error occurs but must execute DML operation if no error is detected

Order of execution

1. INSTEAD OF trigger
2. Regular constraints
3. AFTER trigger

Trigger Benefits

- Capable of enforcing complex restrictions
- Used to track or log changes to a table
- More than one AFTER trigger can be created on a table
- Only one INSTEAD OF trigger of each type can be created on a table
- Can execute stored procedures

Triggers

- Have access to two special virtual tables called inserted and deleted which are pre-defined for each table
- Structure of both inserted and deleted virtual tables is identical to target table
- Virtual tables are based on the contents of the transaction log and reference the values affected by the INSERT, UPDATE or DELETE statements performed against the target table to which the trigger is attached

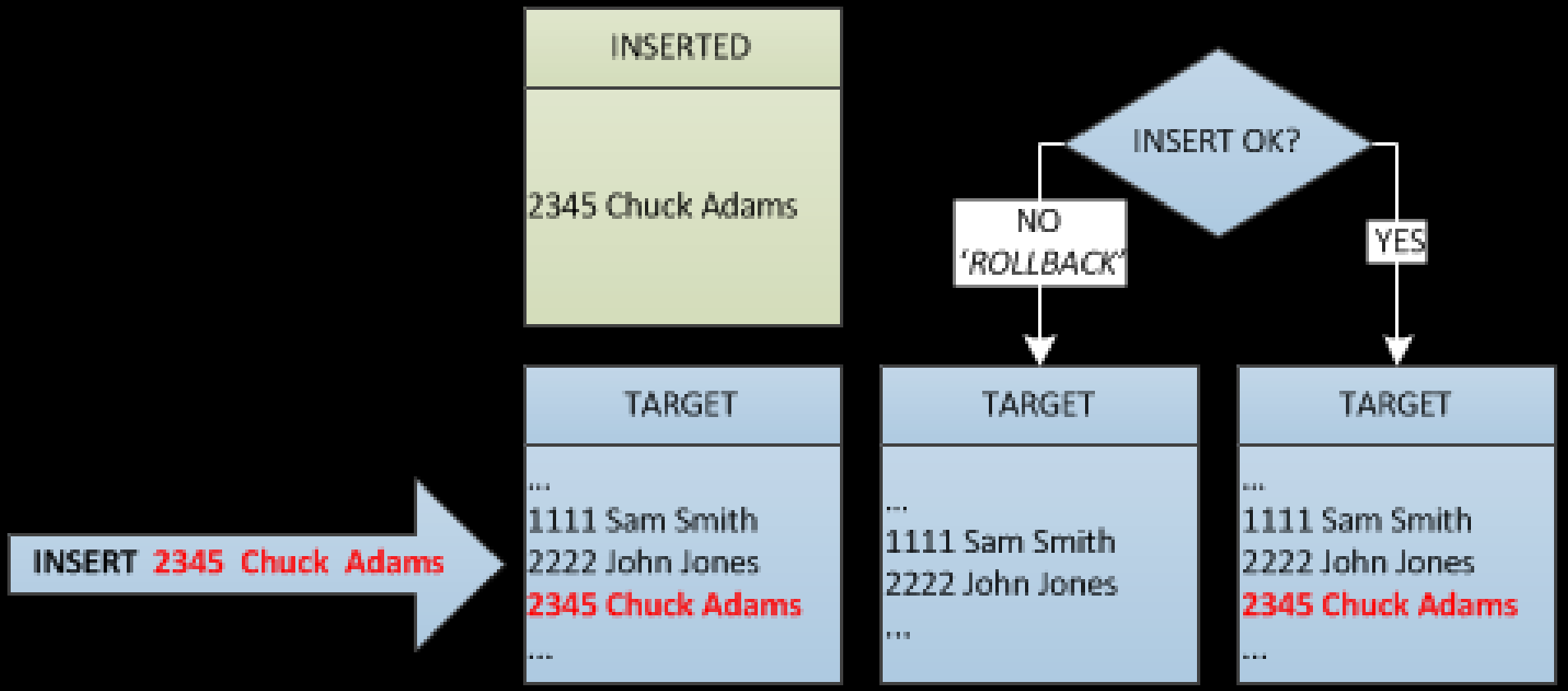
Triggers

- Inserted Virtual Table
  - Contains copies of the new row when an INSERT statement is executed or the updated row when an UPDATE statement is executed
- Deleted Virtual Table
  - Contains the deleted row when a DELETE statement is executed, or the old values when an UPDATE statement is executed

INSERT Triggers

- Used to ensure that the data being inserted into a target table is valid
- When an INSERT transaction is detected by the database, the insert trigger is fired
- When the trigger runs, the inserted data is inserted into both the target table and the inserted virtual table
- Copy of the new row stays in the inserted virtual table until the trigger decides how to implement the data insert

INSERT Trigger Schematic



INSERT Example

```
CREATE TRIGGER   tr_SalesCheck
ON     sales
FOR INSERT
AS
IF ( SELECT qty FROM inserted ) > 50
   BEGIN
        PRINT        'Quantity cannot exceed 50'
        ROLLBACK  TRANSACTION
   END
```

INSERT Example

```
INSERT INTO  sales
VALUES
   (     '6380',
         '6873',
         'Sep 15 1994',
         60,
         'Net 60',
         'BU1032'      )
```

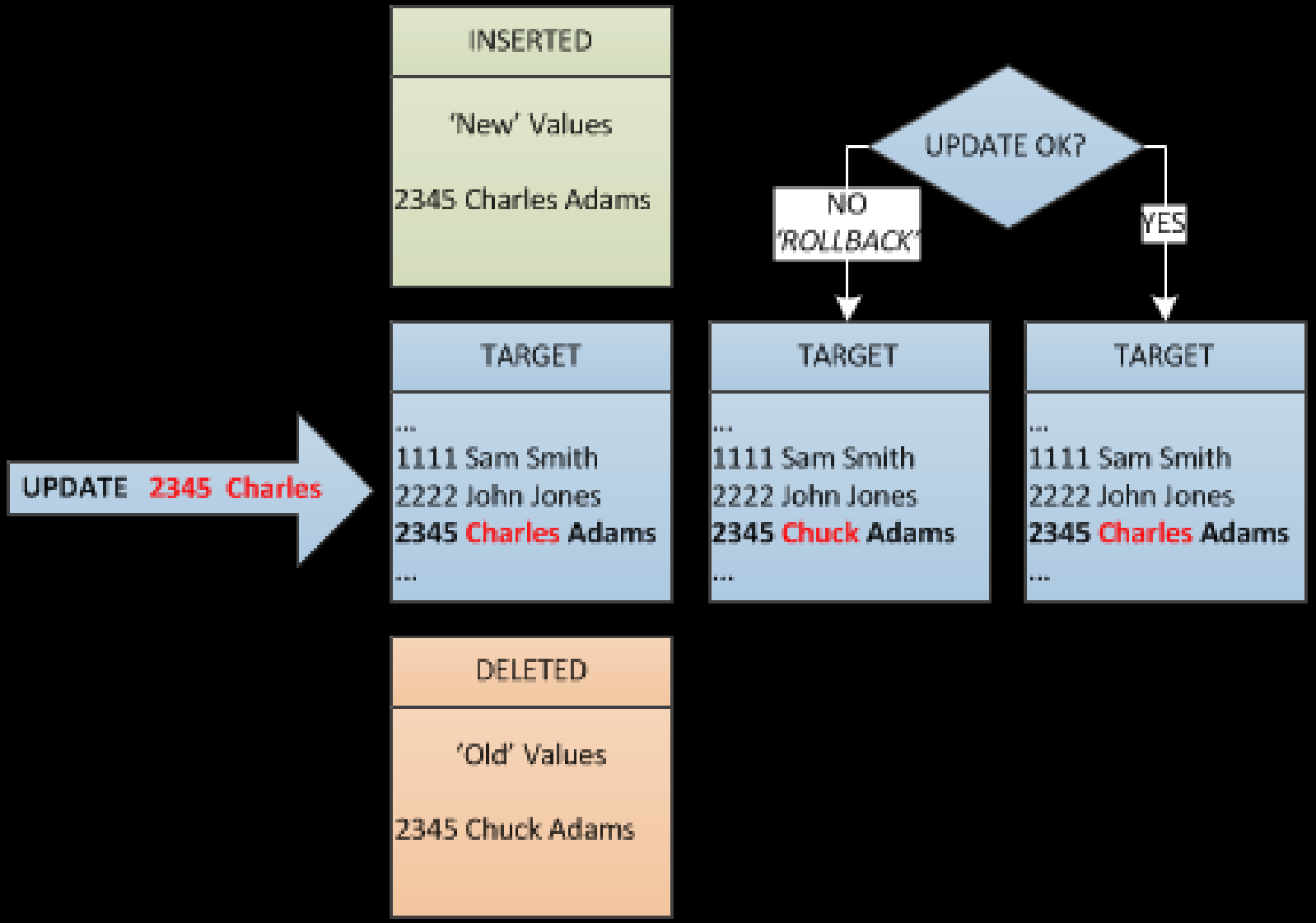
Results

Quantity cannot exceed 50

UPDATE Triggers

- Can occur at both the table level and the column level
- Original data is moved to the deleted virtual table
- New updated rows are then moved to both the target table and the inserted virtual table
- Once the data has been successfully moved, the trigger will check to see if the data can be verified

UPDATE Trigger Schematic



UPDATE Example – Table Level

- Fires when any column in the row is updated

```
CREATE TRIGGER      tr_NoHighQty
ON    sales
FOR UPDATE
AS
IF ( SELECT qty FROM inserted ) > 50
      BEGIN
         PRINT 'Quantity cannot exceed 50'
         ROLLBACK TRANSACTION
      END
```

UPDATE Example

```
UPDATE    sales
SET       qty = 60
WHERE     stor_id = '6380'
   AND    ord_num = '6871'
```

Results

Quantity cannot exceed 50

UPDATE Example – Column Level

- Uses the IF UPDATE (*column_name*) clause which binds the update trigger to a column
- Executes when data in a particular column is altered

```
CREATE TRIGGER   tr_NoHireDate
ON   employee
FOR UPDATE
AS
IF UPDATE (hire_date)
      BEGIN
         PRINT 'The hire date cannot be changed'
         ROLLBACK TRANSACTION
      END
```

UPDATE Example

```
UPDATE    employee
SET       hire_date = 'Jan 10 1990'
WHERE     emp_id = 'PMA42628M'
```

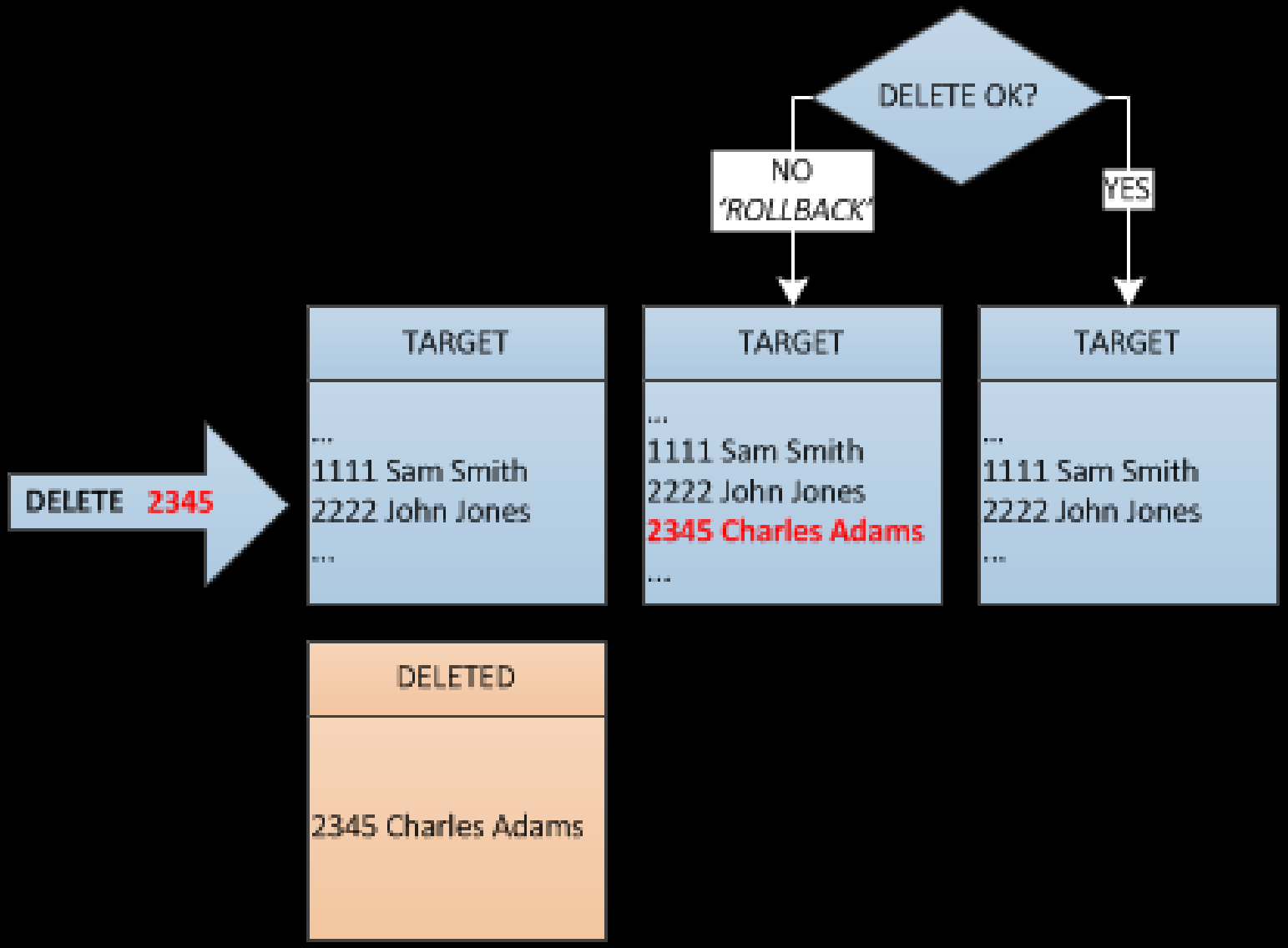
Results

The hire date cannot be changed

DELETE Triggers

- Executed when a DELETE statement is issued against rows in a table
- Can prevent deletion of crucial data
- Deleted rows are moved from the target table to the deleted virtual table

DELETE Trigger Schematic



DELETE Example

```
CREATE TRIGGER      tr_NoDeleteQty
ON    sales
FOR DELETE
AS
IF ( SELECT qty FROM deleted ) > 0
   BEGIN
        PRINT  'Cannot delete non-zero item'
        ROLLBACK TRANSACTION
   END
```

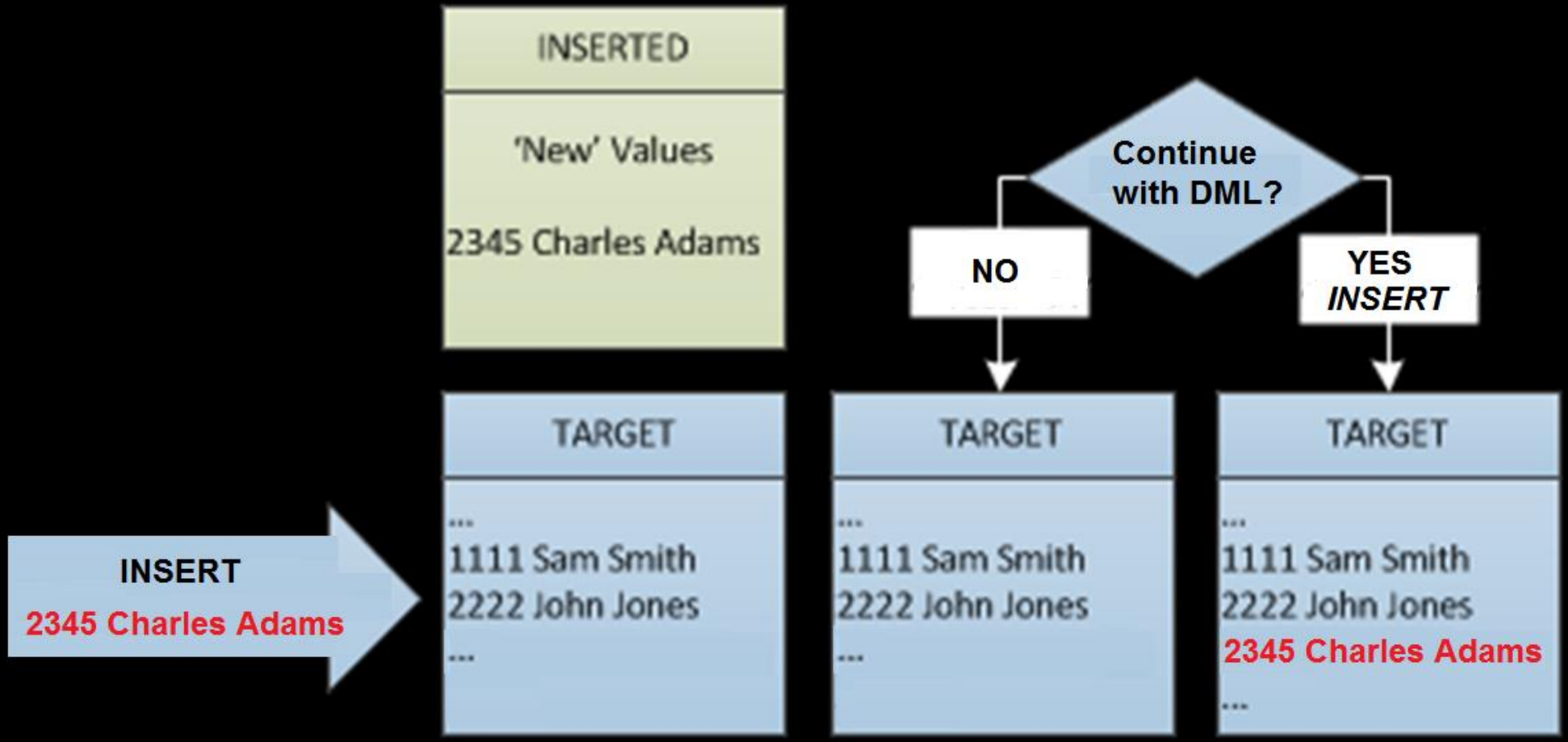
DELETE Example

```
DELETE FROM    sales
WHERE      stor_id = '6380'
    AND    ord_num = '6871'
```

Results

Cannot delete non-zero item

INSTEAD OF Trigger Schematic



INSTEAD OF Example

```
CREATE TRIGGER  tr_SalesCheck
ON      sales
INSTEAD  OF  INSERT, UPDATE
AS
IF ( SELECT qty  FROM inserted ) > 50
    BEGIN
          PRINT   'Quantity cannot exceed 50'
    END
ELSE
    BEGIN
       INSERT   INTO  sales
       SELECT   *
       FROM     inserted
    END
```

INSTEAD OF Example

```
INSERT  INTO  sales
VALUES
   (     '6380',
         '6873',
         'Sep 15 1994',
         60,
         'Net 60',
         'BU1032'      )
```

Results

Quantity cannot exceed 50

To Do

- Submit Exercise 10 – SQL
- Submit Lesson 10 Review Questions
- Post to Lesson 10 Discussion Topic
- Submit Term Project – Part B
- Complete Term Project – Part C and D
- Read Chapters 11, 13, and 15

Lesson 11

1. **Performance Tuning**
2. **Data Warehousing**
3. **Database Security**

1. Database Performance Tuning

- Goal is to execute queries as fast as possible
- Ensure sufficient resources to minimize occurrence of bottlenecks
- Good database performance starts with good database design

Performance Tuning

Client Side

- Objective is to generate SQL query that returns correct answer in least amount of time, using minimum resources at server end
- SQL performance tuning

Server Side

- DBMS environment must be properly configured to respond to clients' requests in fastest way possible, while making optimum use of existing resources
- DBMS performance tuning

SQL Performance Tuning

Evaluated from client perspective

- Most current-generation relational DBMS perform automatic query optimization at the server end
- Most SQL performance optimization techniques are DBMS-specific and are rarely portable

Query Formulation

- Identify what columns and computations are required
- Identify source tables
- Determine how to join tables
- Determine what selection criteria is needed
- Determine in what order to display output

DBMS Performance Tuning

- Includes global tasks such as managing DBMS processes in primary memory and structures in physical storage
- Focuses on setting parameters used for:
  - Data cache
  - SQL cache
  - Sort cache
  - Optimizer mode

DBMS Performance Tuning

- Some general recommendations for creation of databases:
  - Use RAID to provide balance between performance and fault tolerance
  - Minimize disk contention

2. Data Warehouse

A subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision making process

Subject-oriented

- A warehouse is organized around major subjects (such as customers and product sales) rather than major application areas (such as invoicing or inventory stock control).

Integrated

- Since data is coming from several sources, it must be made consistent and integrated to present a unified view of the data to the users.

Time-variant

- Data in the warehouse is only accurate and valid at some point in time or over some time period. It is a snapshot of the operational data that becomes outdated over time.

Non-volatile

- The data is not updated in real time but is refreshed from operational data on a regular basis. It is added as a supplement to the warehouse rather than as a replacement.

Benefits

- Potential high returns on investment
- Competitive advantage
- Increased productivity of corporate decision-makers

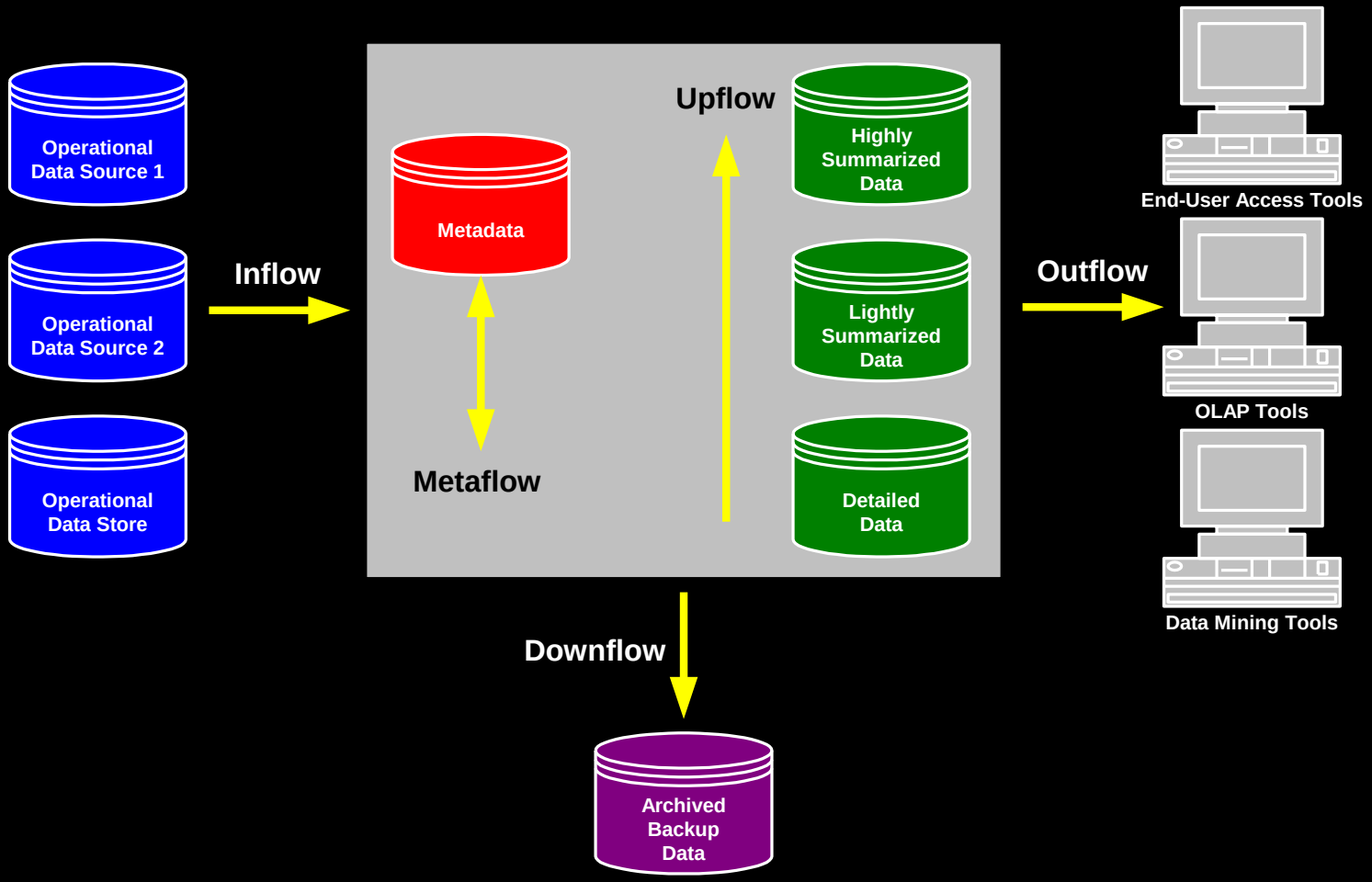
Problems

- Underestimating of resources for data loading
- Hidden problems with source systems
- Required data not captured
- Data homogenization
- Data ownership

Architecture

1. Operational Data
2. Operational Datastore
3. Load Manager
4. Warehouse Manager
5. Query Manager
6. Detailed Data
7. Summarized Data
8. Archive/Backup Data
9. Metadata
10. Access Tools

Data Flows


Operational Data Source 1
Operational Data Source 2
Operational Data Store
Inflow
Metadata
Metaflow
Upflow
Highly Summarized Data
Lightly Summarized Data
Detailed Data
Outflow
End-User Access Tools
OLAP Tools
Data Mining Tools
Downflow
Archived Backup Data

Data Flows

Inflow

- Extraction, cleansing, and loading of the source data

Upflow

- Adding value to the data in the warehouse through summarizing, packaging, and distribution of the data

Downflow

- Archiving and backing-up the data in the warehouse

Outflow

- Making the data available to the end-users

Metaflow

- Managing the metadata

3. Database Security

Database Security

- The mechanisms that protect the database against intentional or accidental threats

Security Considerations

- Theft and fraud
- Loss of confidentiality (secrecy)
- Loss of privacy
- Loss of integrity
- Loss of availability

Threats

Any situation or event, whether intentional or accidental, that may adversely affect a system and consequently the organization.

1. Hardware
2. Software
3. Database
4. Communication Networks
5. Staff

1. Hardware

- Fire/flood/bombs
- Data corruption due to power loss/surge
- Failure of security mechanisms
- Theft of equipment
- Physical damage to equipment
- Electronic interference and radiation

2. Software

- Failure of security mechanisms
- Program alteration
- Theft of programs

3. Database

- Unauthorized changes
- Theft of data
- Data corruption due to power loss/surge

4. Communication Networks

- Wire tapping
- Breaking or disconnection of cables
- EMI (Electromagnetic interference) and radiation

5. Staff

- Using another person's means of access
- Viewing and disclosing unauthorized data
- Inadequate staff training
- Illegal entry by hacker
- Blackmail
- Introduction of viruses

- Creating backdoors
- Program alteration
- Inadequate training
- Inadequate security policies and procedures
- Staff shortage or strikes

Countermeasures

1. Authorization
2. Access controls
3. Views
4. Backup and recovery
5. Integrity
6. Encryption
7. RAID technology

1. Authorization

The granting of a right or privilege that enables a subject to have legitimate access to a system is called authorization

The mechanism that determines whether a user is who he or she claims to be is known as authentication

2. Access Control

Privileges

- read / write / modify

Discretionary Access Control

- SQL defined by owner

Mandatory Access Control

- system defined policies
- security class / clearance / rules

3. Views

- A view is the dynamic result of one or more relational operations on one or more tables to produce another table.
- A view is a virtual table that does not actually exist in the database, but is produced upon request by a particular user, at the time of request.

4. Backup & Recovery

Backup

- The process of periodically taking a copy of the database and journal onto offline storage media.

Journaling

- The process of keeping and maintaining a journal (or log file) of all changes made to the database to enable recovery in the event of failure.

5. Integrity

- Integrity constraints contribute to maintaining a secure database by preventing invalid data from being created, which would compromise database integrity

6. Encryption

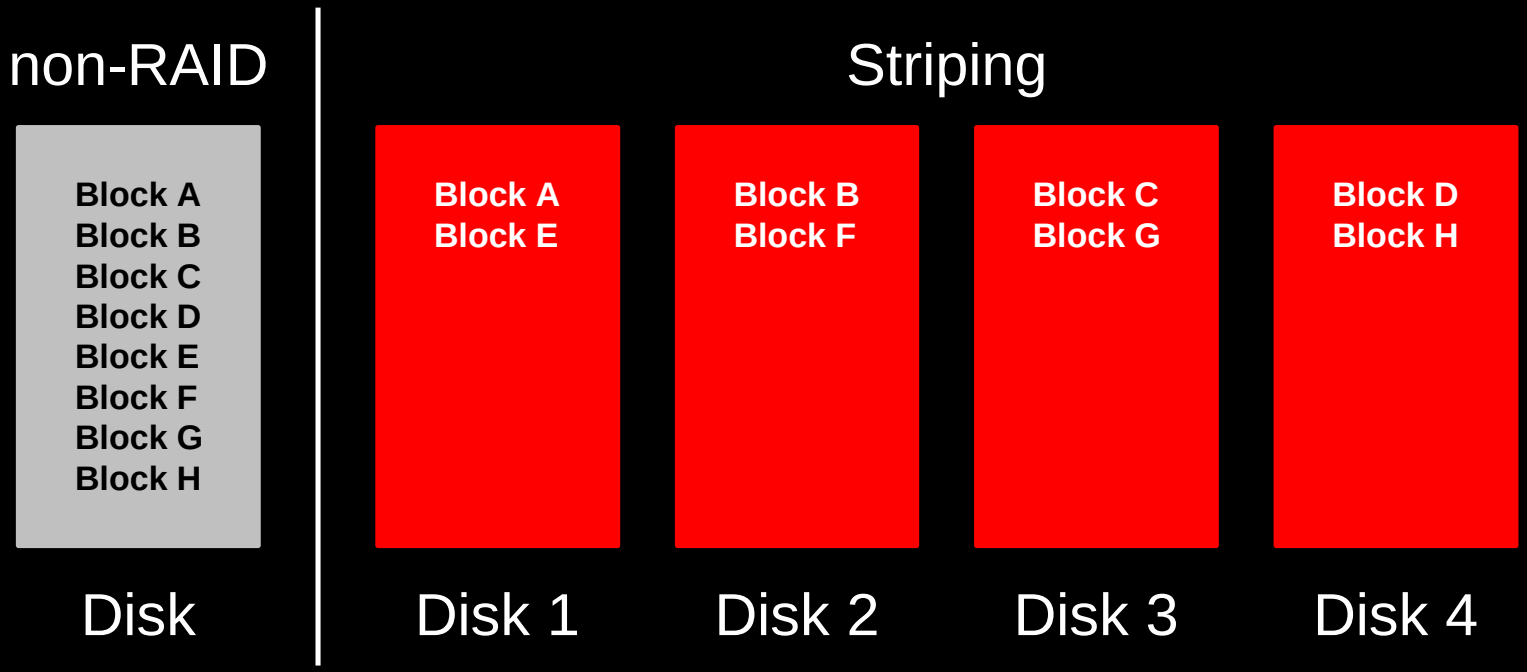
- The encoding of the data by a special algorithm that renders the data unreadable by any program without the decryption key.
- A cryptosystem includes:
  - encryption key
  - encryption algorithm to generate ciphertext
  - decryption key
  - decryption algorithm to generate plaintext

7. RAID

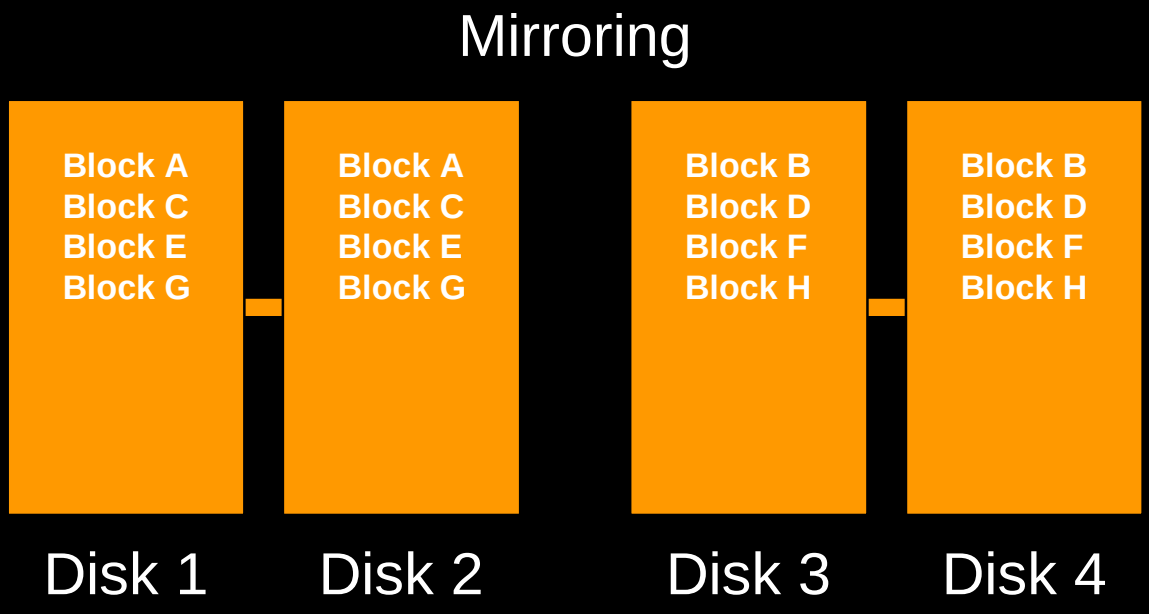
A Redundant Array of Independent Disks provides:

- higher performance via data striping, the simultaneous reading and writing on multiple disks
- enhanced reliability via mirroring and error detection/correction with parity

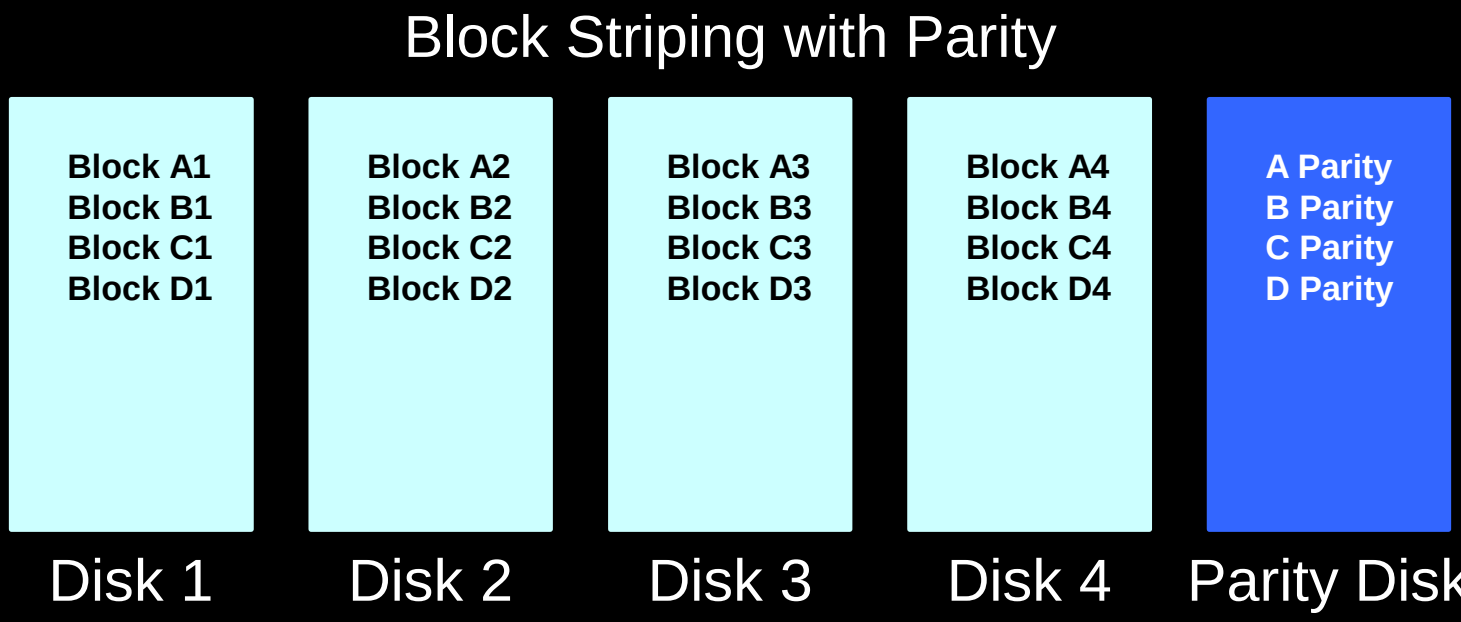
RAID - Level 0


non-RAID
Striping
Block A
Block B
Block C
Block D
Block E
Block F
Block G
Block H
Block A
Block E
Block B
Block F
Block C
Block G
Block D
Block H
Disk
Disk 1
Disk 2
Disk 3
Disk 4

RAID - Level 1

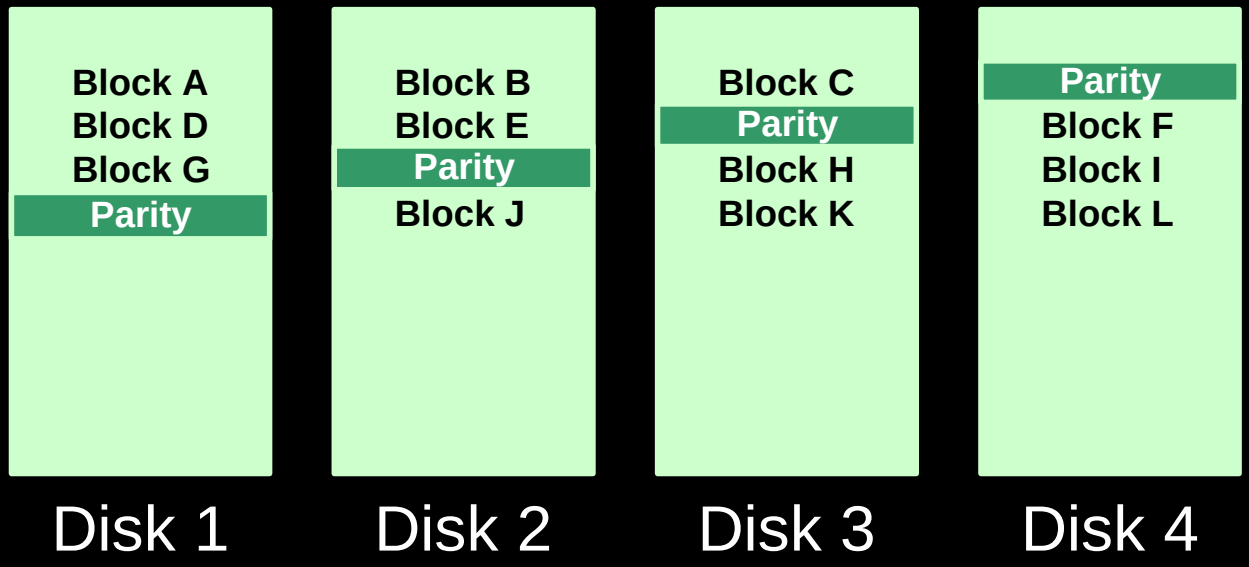

Mirroring
Block A
Block C
Block E
Block G
Block A
Block C
Block E
Block G
Block B
Block D
Block F
Block H
Block B
Block D
Block F
Block H
Disk 1
Disk 2
Disk 3
Disk 4

RAID - Level 3


Block Striping with Parity
Block A1
Block B1
Block C1
Block D1
Block A2
Block B2
Block C2
Block D2
Block A3
Block B3
Block C3
Block D3
Block A4
Block B4
Block C4
Block D4
A Parity
B Parity
C Parity
D Parity
Disk 1
Disk 2
Disk 3
Disk 4
Parity Disk

RAID - Level 5

Striping with Distributed Parity



To Do

- Submit Lesson 11 Review Questions
- Post to Lesson 11 Discussion Topic
- Submit Course Evaluation
- Complete Term Project – Part D
- Prepare for Final Exam