Lecture 10: SQL Operation, Relationships, and Data Integrity
(Structured Query Language III)

EGCI 321: DATABASE SYSTEM (WEEK 5)

Outline

Section I

- TOP clause
- LIKE Operator
- SQL Wildcards
- IN Operator
- BETWEEN Operator
- Joins
- UNION Operator
- SELECT INTO Statement
- CREATE DATABASE Statement
- CREATE TABLE Statement
- Constraints
- CREATE INDEX Statement
- VIEW Statement
- SQL Server Data Types

Section II

- Relationship and Data Integrity
- DataJoin

TOP Clause

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

SQL Server Syntax

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

Example (MSSQL)

```
SELECT TOP 5  *
FROM employee
```

Example (MYSQL)

```
SELECT *
FROM employee
LIMIT 5
```

LIKE Operator

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

SQL LIKE Syntax

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

Example

```
SELECT * FROM employee
WHERE Firstname LIKE 'e%'
```

SQL Wildcards

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

SQL wildcards must be used with the SQL LIKE operator.

- % : The percent sign represents zero, one, or multiple characters.
- _ : The underscore represents a single character.

IN Operator

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

SQL IN Syntax

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

Example

```
SELECT * FROM employee
WHERE LASTNAME IN ('HENDERSON','KWAN')
```

BETWEEN Operator

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

SQL BETWEEN Syntax

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

Example

```
SELECT *
FROM employee
WHERE LastName
BETWEEN 'Henderson' AND 'Kwan'
```

Joins

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

Different SQL JOINs

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

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

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

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

INNER JOIN

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

SQL INNER JOIN Syntax

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

Example

```
SELECT p.PROJNAME, d.Deptname
FROM project p
INNER JOIN department d
ON p.DEPTNO=d.DEPTNO
```

UNION Operator

The SQL UNION operator combines two or more SELECT statements.

SQL UNION Syntax

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

Example

```
SELECT * FROM department
UNION
SELECT * FROM department2
```

SELECT INTO Statement (MS SQL)

- The SELECT INTO statement selects data from one table and inserts it into a different table.
- The SELECT INTO statement is most often used to *create backup copies of tables*.

SQL SELECT INTO Syntax

```
SELECT column_name(s)
INTO new_table_name [IN externaldatabase]
FROM old_tablename
```

Example (MS SQL)

```
SELECT *
INTO employee_Backup
FROM employee
```

Example (MYSQL)

```
CREATE TABLE newemp (
SELECT * FROM employee
);
```

CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a database.

SQL CREATE DATABASE Syntax

```
CREATE DATABASE database_name
```

Example

```
CREATE DATABASE my_db
```

CREATE TABLE Statement

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

SQL CREATE TABLE Syntax

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

Example

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

Constraints

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

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

List of Constraints

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

NOT NULL Constraint

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

Example

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

UNIQUE Constraint

- The UNIQUE constraint uniquely identifies each record in a database table.
- The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

Example

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

Naming of a UNIQUE constraint

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

PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain unique values.
- A primary key column cannot contain NULL values.

Example

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

Namming of PRIMARY KEY constraint

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

FOREIGN KEY Constraint

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

- The "P_Id" column in the "Persons" table is the **PRIMARY KEY** in the "Persons" table.
- The "P_Id" column in the "Orders" table is a **FOREIGN KEY** in the "Orders" table.

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

FOREIGN KEY Constraint (cont.)

Example (MS SQL)

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

Example (MYSQL)

```
CREATE TABLE orders(
    orderID int not null,
    orderNumber int not null,
    personID int,
    PRIMARY KEY(orderID),
    FOREIGN KEY (PersonID) REFERENCES persons(P_id))
```

Naming of a FOREIGN KEY constraint

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

CHECK Constraint (MSSQL)

The **CHECK** constraint is used to limit the value range that can be placed in a column.

- If you define a **CHECK** constraint on a single column it allows only certain values for this column.
- If you define a **CHECK** constraint on a table it can limit the values in certain columns based on values in other columns in the row.

Example:

```
CREATE TABLE Persons
(
    P_Id          int   NOT NULL CHECK (P_Id>0),
    LastName      varchar(255)   NOT NULL,
    FirstName     varchar(255),
    Address       varchar(255),
    City          varchar(255)
)
```

Naming of a CHECK constraint

```
CREATE TABLE Persons
(
    P_Id          int NOT NULL,
    LastName      varchar(255) NOT NULL,
    FirstName     varchar(255),
    Address       varchar(255),
    City          varchar(255),
    CONSTRAINT chk_Person    CHECK (P_Id>0 AND City='Sandnes')
)
```

CREATE INDEX Statement

The **CREATE INDEX** statement is used to create indexes in tables.

Indexes allow the database application to find data fast; without reading the whole table.

SQL CREATE INDEX Syntax

```
CREATE INDEX index_name
ON table_name (column_name)
```

SQL CREATE UNIQUE INDEX Syntax

```
CREATE UNIQUE INDEX index_name
ON table_name (column_name)
```

Example:

```
CREATE INDEX PIndex
ON Persons (LastName)
```

VIEW Statement

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

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

SQL CREATE VIEW Syntax

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

Example:

```
CREATE VIEW     CurrentProductList AS
SELECT          ProductID,ProductName
FROM            Products
WHERE           Discontinued=No
```

SQL Server Data Types

Character strings:

Data type	Description	Storage
char(n)	Fixed-length character string. Maximum 8,000 characters	n
varchar(n)	Variable-length character string. Maximum 8,000 characters	
varchar(max)	Variable-length character string. Maximum 1,073,741,824 characters	
text	Variable-length character string. Maximum 2GB of text data	

Unicode strings:

Data type	Description	Storage
nchar(n)	Fixed-length Unicode data. Maximum 4,000 characters	
nvarchar(n)	Variable-length Unicode data. Maximum 4,000 characters	
nvarchar(max)	Variable-length Unicode data. Maximum 536,870,912 characters	
ntext	Variable-length Unicode data. Maximum 2GB of text data	

Binary types:

Data type	Description	Storage
bit	Allows 0, 1, or NULL	
binary(n)	Fixed-length binary data. Maximum 8,000 bytes	
varbinary(n)	Variable-length binary data. Maximum 8,000 bytes	
varbinary(max)	Variable-length binary data. Maximum 2GB	
image	Variable-length binary data. Maximum 2GB	

Number types:

Data type	Description	Storage
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

Date types:

Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

Other data types:

Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a cursor used for database operations
table	Stores a result-set for later processing

Section II: SQL Relationship and Data Integrity

Creating a Foreign Key in SQL

SQL Foreign Key Expression

FOREIGN KEY RFEREENCES *ParentTableName* (*ForeignKeyColumn*)

Example

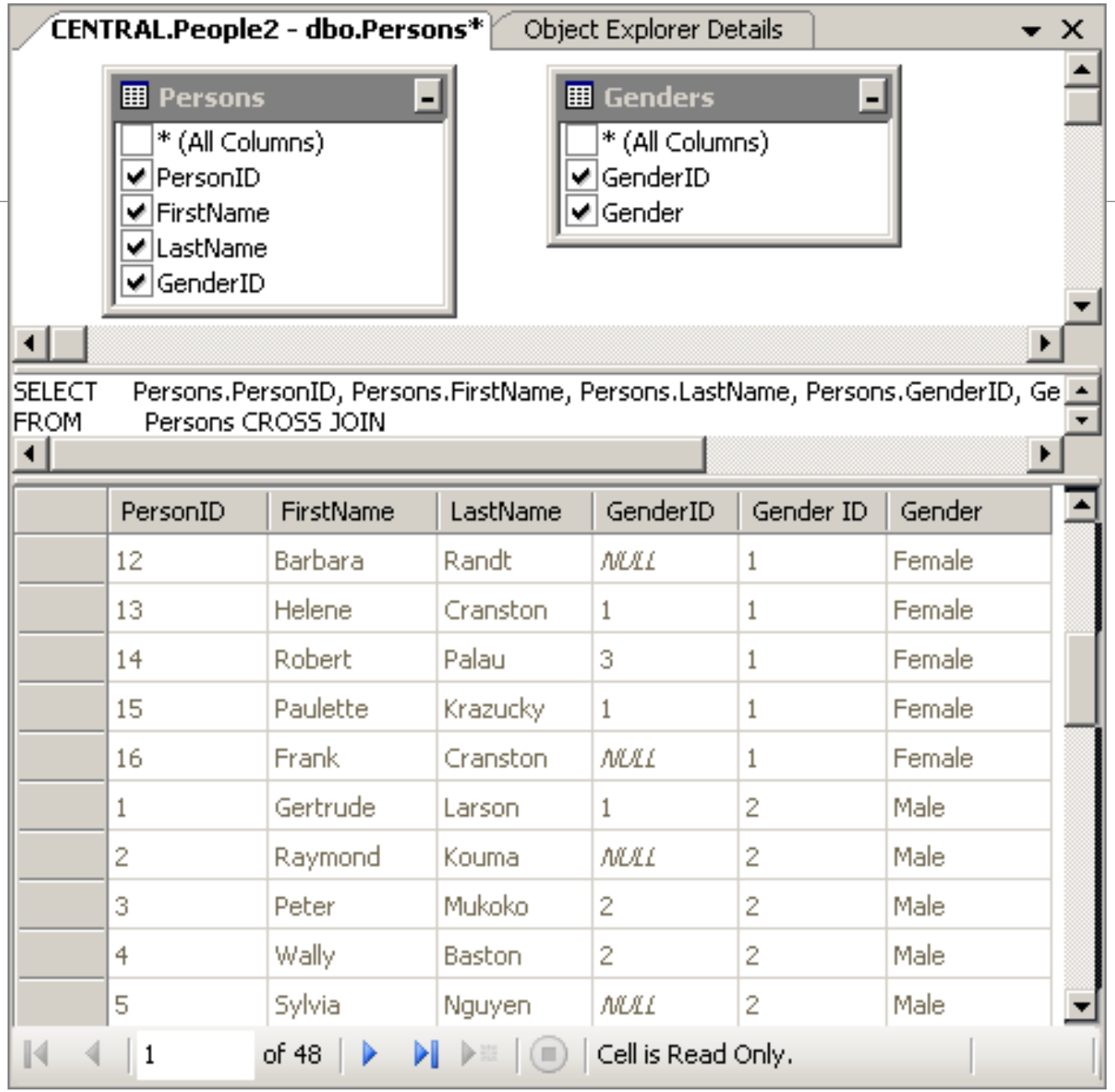
```
//MS SQL
    CREATE TABLE  Persons
    (         PersonID     int identity(1,1) PRIMARY KEY NOT
                           NULL,
              FirstName    varchar(20),
              LastName     varchar(20) NOT NULL,
              GenderID     int NULL FOREIGN KEY REFERENCES
              Genders(GenderID)
    );
```

```
//MYSQL
CREATE TABLE  Persons
(         PersonID     int Auto_Increment PRIMARY KEY NOT NULL,
          FirstName    varchar(20),
          LastName     varchar(20) NOT NULL,
          GenderID     int NULL,
    FOREIGN KEY (GenderID) REFERENCES Genders(GenderID)
);
```

Data Join

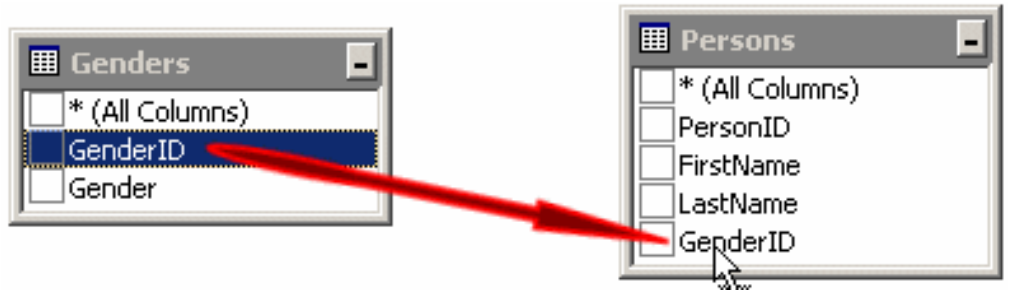
Cross Joins

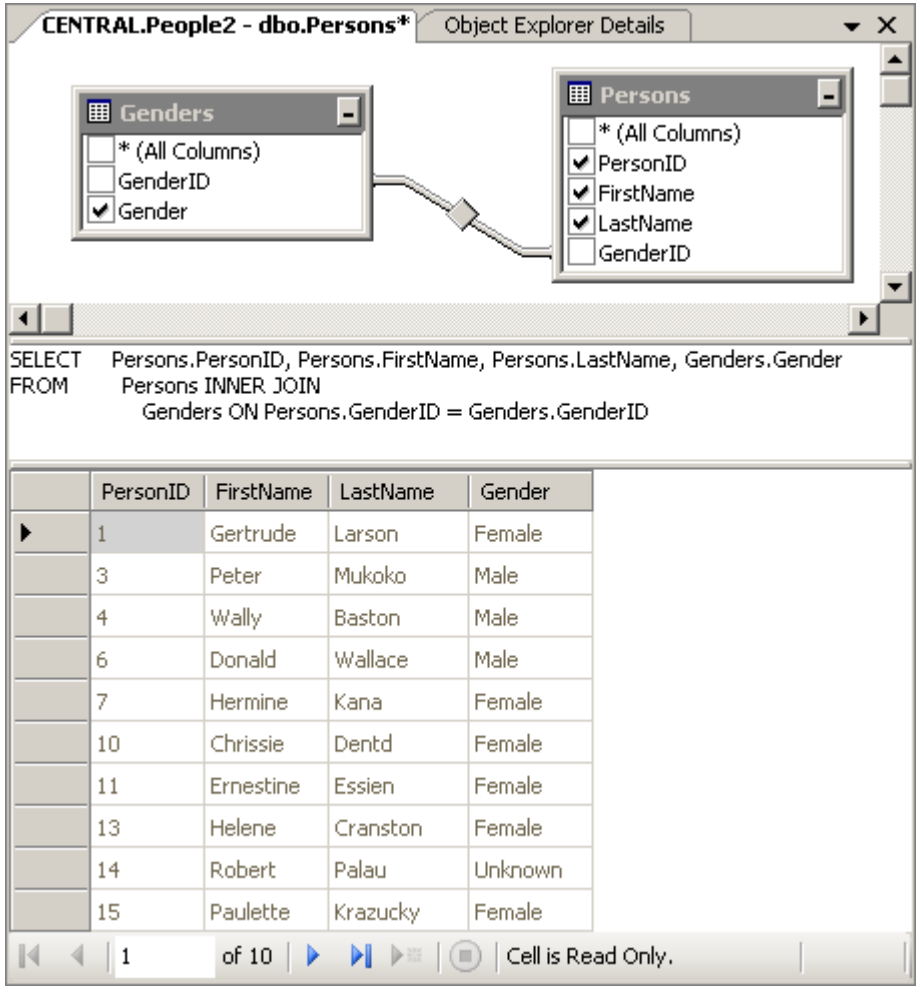
```
SELECT Person.PersonID,
  Person.FirstName,
  Person.LastName,
  Gender.GenderID,
  Gender.Gender
FROM Person CROSS JOIN
  Gender
```



INNER JOIN

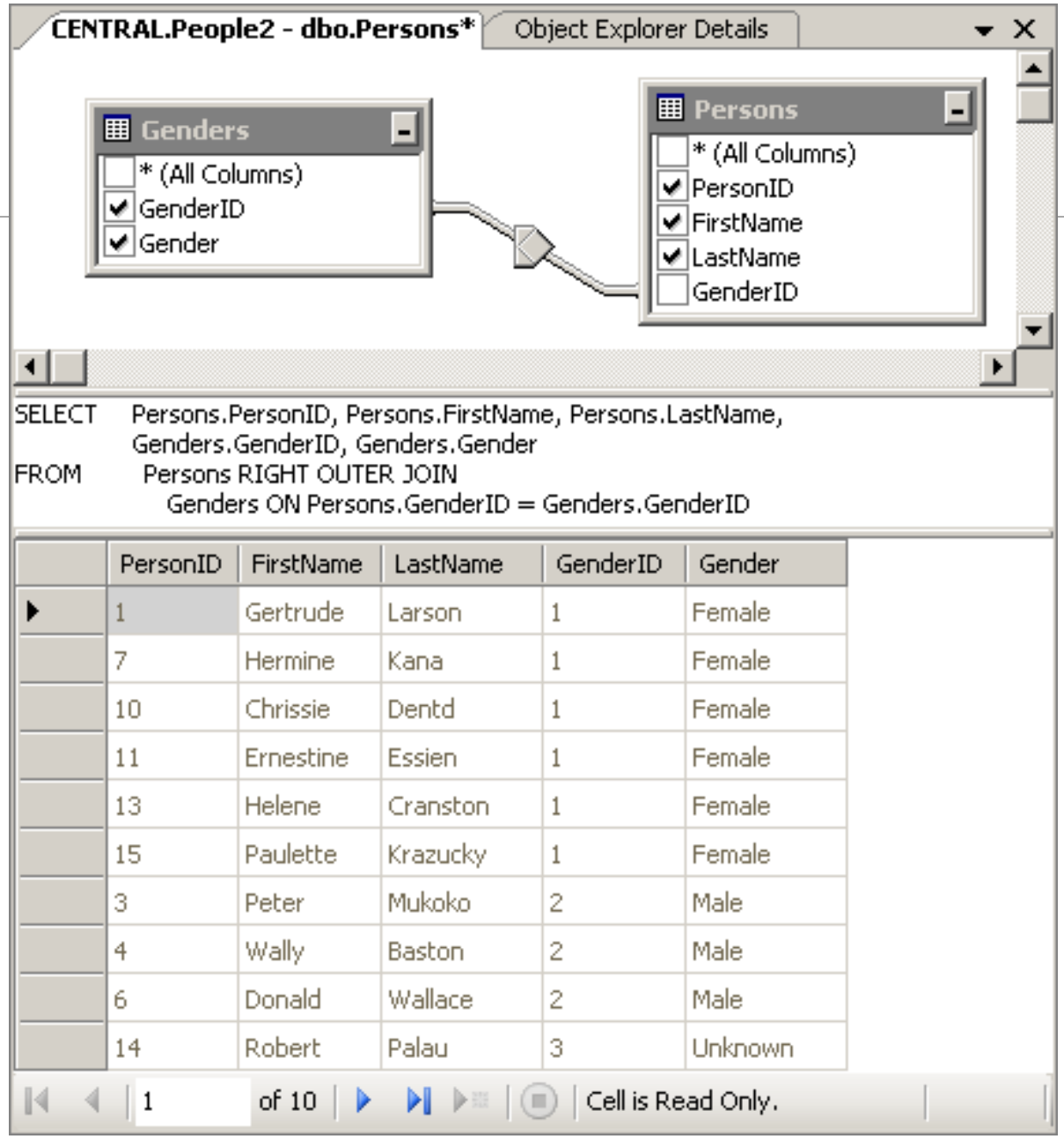
```
SELECT    Person.PersonID, Person.FirstName,
   Person.LastName, Person.GenderID,
   gender.GenderID AS Gender_ID,
   gender.Gender
FROM   Person
 INNER JOIN   gender
ON Person.GenderID = gender.GenderID
```





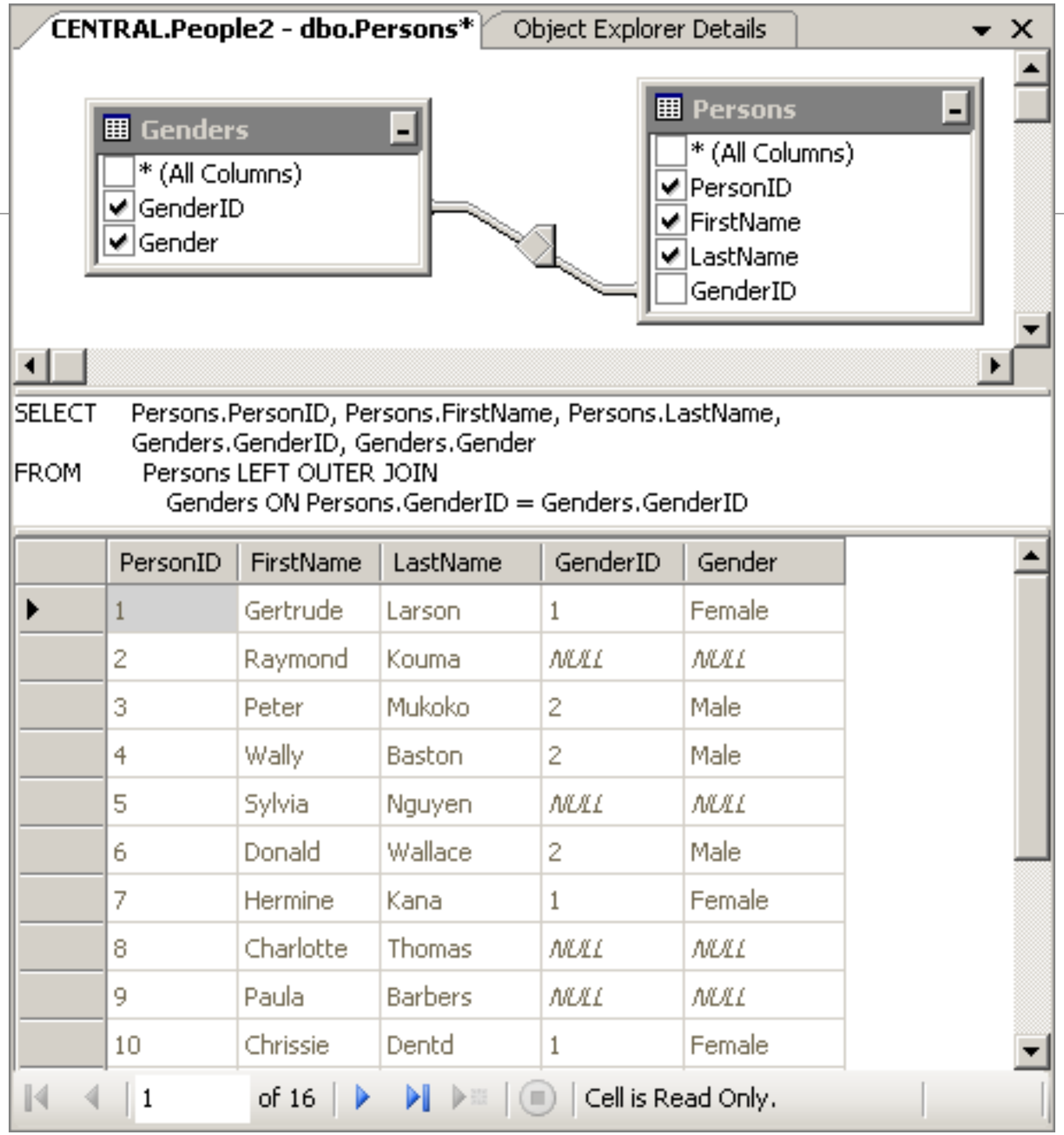
Right Outer Joins

```
SELECT   P.PersonID, P.FirstName,
   P.LastName, G.GenderID, G.Gender
FROM  Person AS P
 RIGHT OUTER JOIN
         gender AS G
ON   P.GenderID = G.GenderID
```



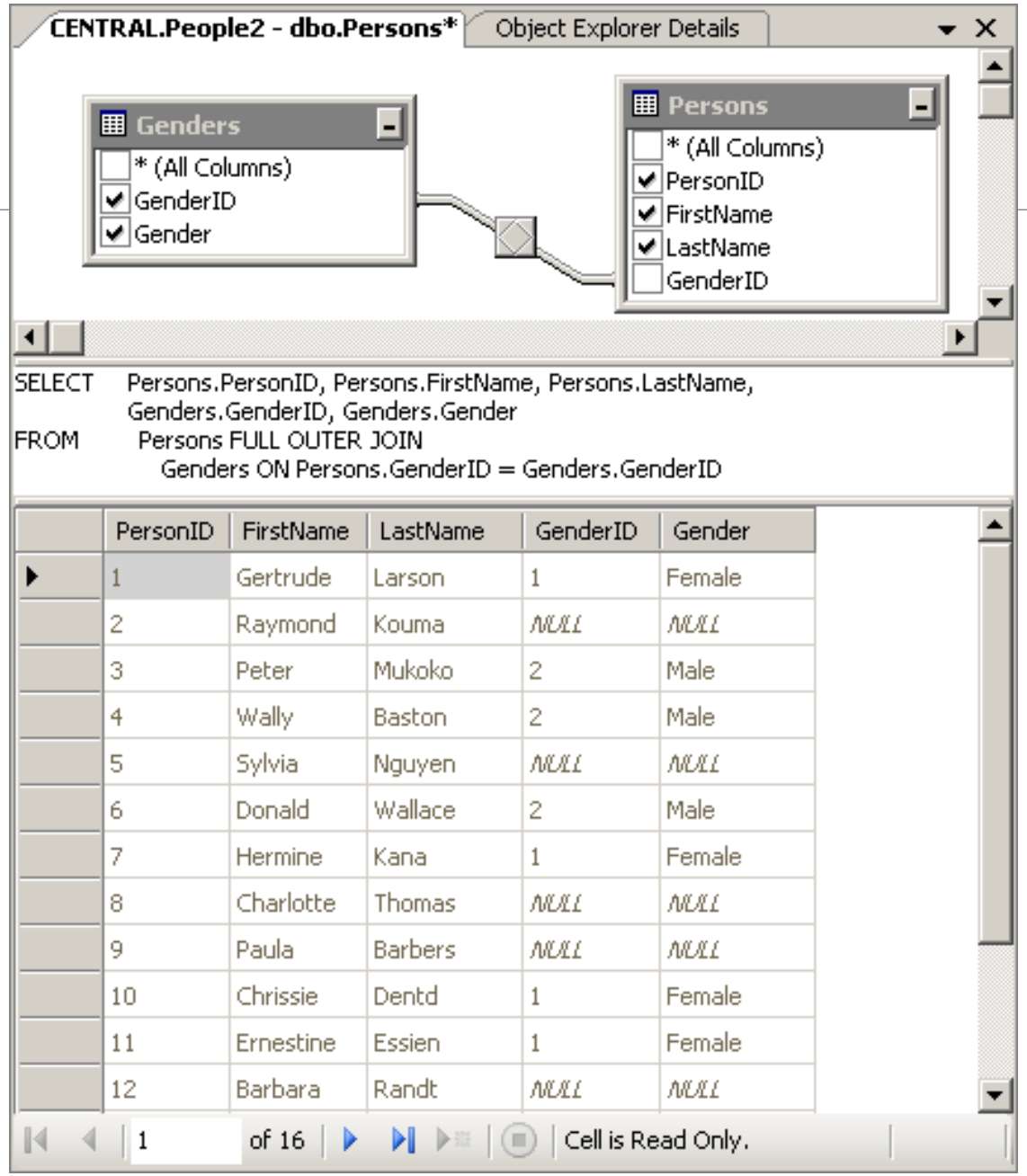
Left Outer Joins

```
SELECT P.PersonID, P.FirstName,
   P.LastName, G.GenderID, G.Gender
FROM Person AS P
LEFT OUTER JOIN Gender AS G
ON P.GenderID = G.GenderID
```



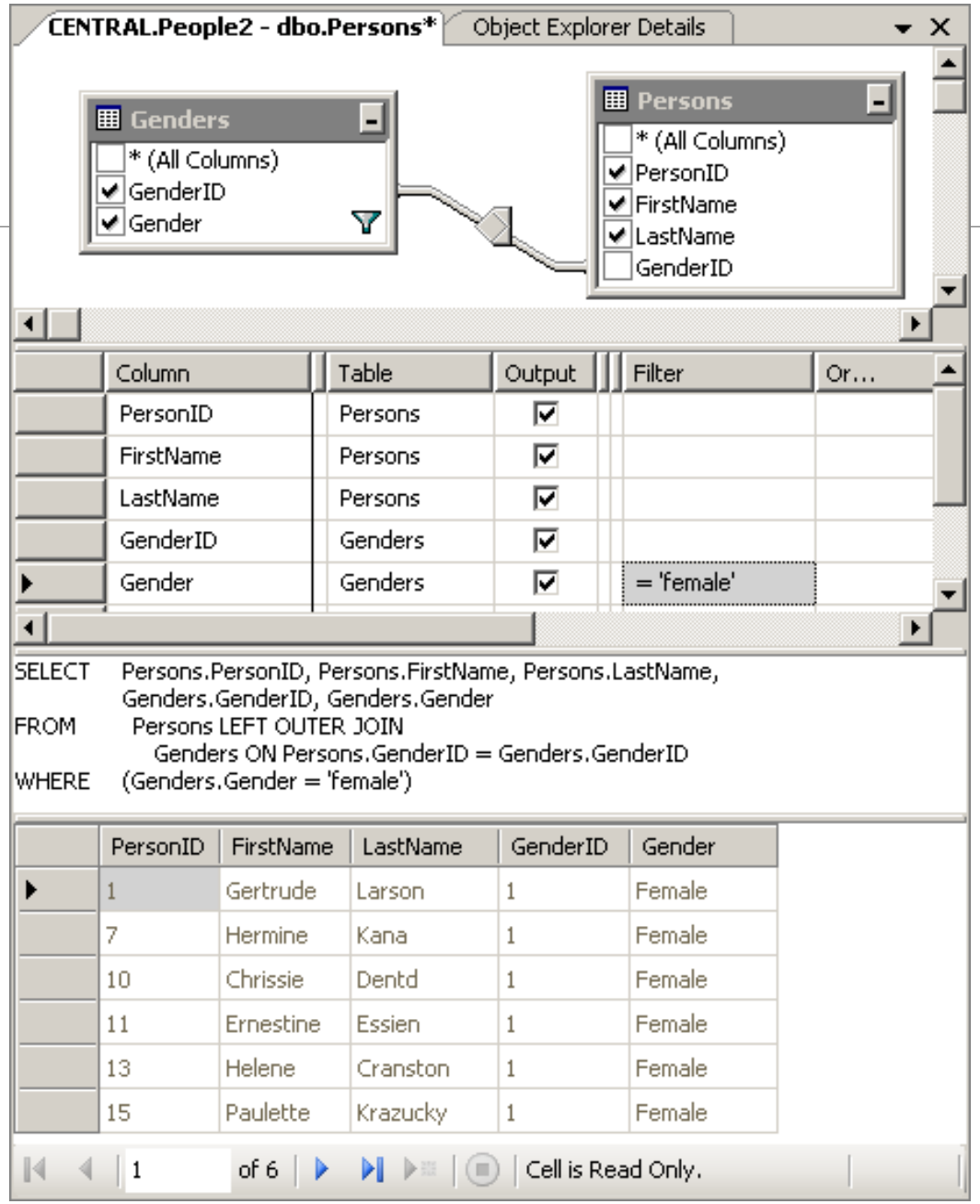
Full Outer Joins (MSSQL)

```
SELECT P.PersonID, P.FirstName,
   P.LastName, G.GenderID,
   G.Gender
FROM Person AS P
FULL OUTER JOIN Gender AS G
ON P.GenderID = G.GenderID
```



Full Outer Joins (MYSQL)

```
SELECT P.PersonID, P.FirstName, P.LastName,
   G.GenderID, G.Gender
FROM Person AS P
LEFT OUTER JOIN Gender AS G
ON P.GenderID = G.GenderID
UNION
SELECT P.PersonID, P.FirstName, P.LastName,
   G.GenderID, G.Gender
FROM Person AS P
RIGHT OUTER JOIN Gender AS G
ON P.GenderID = G.GenderID
```



Reference

1. Ramakrishnan R, Gehrke J., Database management systems, 3rd ed., New York (NY): McGraw-Hill, 2003.
2. SQL Tutorial, http://www.w3schools.com/sql, 2010.
3. Microsoft SQL Server 2005 Lessons, http://www.functionx.com/sqlserver 2005, May 2007.