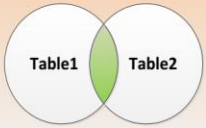
# TSQL JOIN TYPES

Created by Steve Stedman

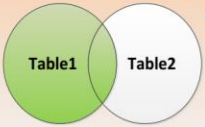

```
SELECT *
  FROM Table1;

SELECT *
  FROM Table2;
```

SELECT from two tables



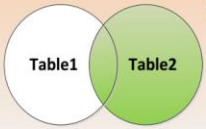

```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id;
```

INNER JOIN



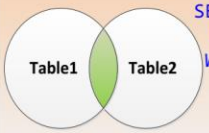

```
SELECT *
  FROM Table1 t1
 LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```

LEFT OUTER JOIN



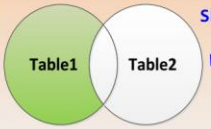

```
SELECT *
  FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```

RIGHT OUTER JOIN



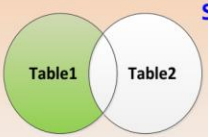

```
SELECT *
  FROM Table1 t1
 WHERE EXISTS (SELECT 1
                 FROM Table2 t2
                WHERE t1.fk = t2.id
               );
```

SEMI JOIN



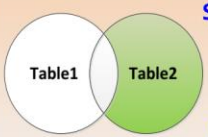

```
SELECT *
  FROM Table1 t1
 WHERE NOT EXISTS (SELECT 1
                     FROM Table2 t2
                    WHERE t1.fk = t2.id
                   );
```

ANTI SEMI JOIN



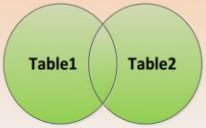

```
SELECT *
  FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t2.id IS NULL;
```

LEFT OUTER JOIN with exclusion
– replacement for a NOT IN



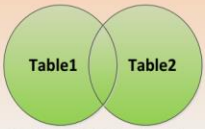

```
SELECT *
  FROM Table1 t1
  RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t1.fk IS NULL;
```

RIGHT OUTER JOIN with exclusion
– replacement for a NOT IN



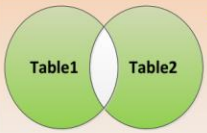

```
SELECT *
  FROM Table1 t1
 FULL OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```

FULL OUTER JOIN



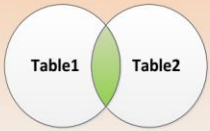

```
SELECT *
  FROM Table1 t1
 CROSS JOIN Table2 t2;
```

CROSS JOIN, the Cartesian product




```
SELECT *
  FROM Table1 t1
  FULL OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t1.fk IS NULL
    OR t2.id IS NULL;
```

FULL OUTER JOIN with exclusion

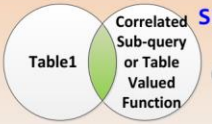



```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk >= t2.id;
```

NON-EQUI INNER JOIN

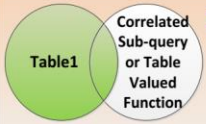
# TSQL JOIN TYPES

Created by Steve Stedman



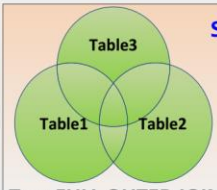

```
SELECT *
  FROM Table1 t1
CROSS APPLY
      [dbo].[someTVF](t1.fk)
      AS t;
```

CROSS APPLY



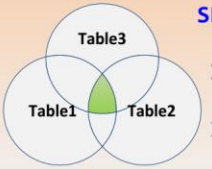

```
SELECT *
  FROM Table1 t1
OUTER APPLY
      [dbo].[someTVF](t1.fk)
       AS t;
```

OUTER APPLY



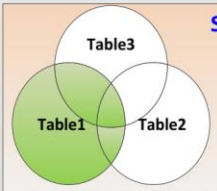

```
SELECT *
  FROM Table1 t1
  FULL OUTER JOIN Table2 t2
    ON t1.fk = t2.id
  FULL OUTER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

Two FULL OUTER JOINS



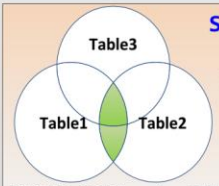

```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id
 INNER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

Two INNER JOINs



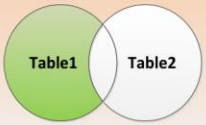

```
SELECT *
  FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
  LEFT OUTER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

Two LEFT OUTER JOINS



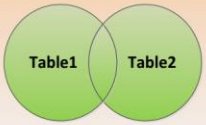

```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id
  LEFT OUTER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

INNER JOIN and a LEFT OUTER JOIN



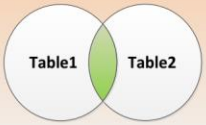

```
SELECT fk as id
  FROM Table1
EXCEPT
SELECT ID
  FROM Table2;
```

EXCEPT




```
SELECT fk as id
  FROM Table1
UNION
SELECT ID
  FROM Table2;
```

UNION




```
SELECT fk as id
  FROM Table1
INTERSECT
SELECT ID
  FROM Table2;
```

INTERSECT

## Sample Schema

**Table 1 (People)**

| | id | Name | fk | fk_table3 |
|---|---|---|---|---|
| 1 | 1 | Steve | 1 | NULL |
| 2 | 2 | Aaron | 3 | NULL |
| 3 | 3 | Mary | 2 | NULL |
| 4 | 4 | Fred | 1 | NULL |
| 5 | 5 | Anne | 5 | NULL |
| 6 | 6 | Beth | 8 | 1 |
| 7 | 7 | Johnny | NULL | 1 |
| 8 | 8 | Karen | NULL | 2 |

**Table 2 (Favorite Colors)**

| | id | FavoriteColor |
|---|---|---|
| 1 | 1 | red |
| 2 | 2 | green |
| 3 | 3 | blue |
| 4 | 4 | pink |
| 5 | 5 | purple |
| 6 | 6 | mauve |
| 7 | 7 | orange |
| 8 | 8 | yellow |
| 9 | 1 | indigo |

**Table 3 (Favorite Foods)**

| | id | dataValue |
|---|---|---|
| 1 | 1 | Pizza |
| 2 | 2 | Burger |
| 3 | 3 | Sushi |

**Note: Column names are very generic to simplify the sample queries.**
**Foreign keys are**
**Table1.fk -> Table2.id**
**Table2.fk_table3 -> Table3.id**

# MySQL JOIN Types

Created by Steve Stedman


SELECT from two tables

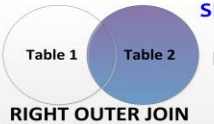
```
SELECT *
  FROM Table1;

SELECT *
  FROM Table2;
```


INNER JOIN

```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id;
```


LEFT OUTER JOIN

```
SELECT *
  FROM Table1 t1
 LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```




RIGHT OUTER JOIN

```
SELECT *
  FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```


SEMI JOIN – Similar to INNER JOIN, with less duplication.

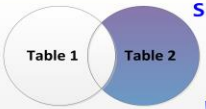
```
SELECT *
  FROM Table1 t1
 WHERE EXISTS (SELECT 1
                 FROM Table2 t2
                WHERE t1.fk = t2.id
               );
```


ANTI SEMI JOIN

```
SELECT *
  FROM Table1 t1
 WHERE NOT EXISTS (SELECT 1
                     FROM Table2 t2
                    WHERE t1.fk = t2.id
                   );
```


LEFT OUTER JOIN with exclusion

```
SELECT *
  FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t2.id is null;
```

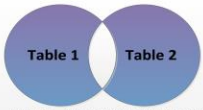



RIGHT OUTER JOIN with exclusion

```
SELECT *
  FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t1.fk is null;
```

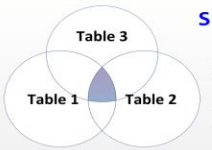

FULL OUTER JOIN

```
SELECT * FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
UNION
SELECT * FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```



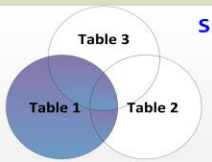

FULL OUTER JOIN with exclusion

```
SELECT * FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t2.id IS NOT NULL
UNION
SELECT * FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t1.ID IS NOT NULL;
```



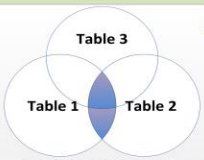

Two INNER JOINs

```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id
 INNER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```




Two LEFT OUTER JOINS

```
SELECT *
  FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
  LEFT OUTER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```




INNER JOIN and a LEFT OUTER JOIN

```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id
  LEFT OUTER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

# Introduction to SQL

Follow me on LinkedIn for more

Steve Nouri

https://www.linkedin.com/in/stevenouri/

# Introduction to SQL

- SQL functions fit into two broad categories:
  - Data definition language
    - SQL includes commands to:
      - Create database objects, such as tables, indexes, and views
      - Define access rights to those database objects
  - Data manipulation language
    - Includes commands to insert, update, delete, and retrieve data within database tables
- SQL is relatively easy to learn
- Basic command set has vocabulary of less than 100 words
- Nonprocedural language
- American National Standards Institute (ANSI) prescribes a standard SQL
- Several SQL dialects exist

# Introduction to SQL (continued)

**TABLE 7.1** SQL Data Definition Commands

| COMMAND OR OPTION | DESCRIPTION |
|---|---|
| CREATE SCHEMA AUTHORIZATION | Creates a database schema |
| CREATE TABLE | Creates a new table in the user's database schema |
| NOT NULL | Ensures that a column will not have null values |
| UNIQUE | Ensures that a column will not have duplicate values |
| PRIMARY KEY | Defines a primary key for a table |
| FOREIGN KEY | Defines a foreign key for a table |
| DEFAULT | Defines a default value for a column (when no value is given) |
| CHECK | Constraint used to validate data in an attribute |
| CREATE INDEX | Creates an index for a table |
| CREATE VIEW | Creates a dynamic subset of rows/columns from one or more tables |
| ALTER TABLE | Modifies a table's definition (adds, modifies, or deletes attributes or constraints) |
| CREATE TABLE AS | Creates a new table based on a query in the user's database schema |
| DROP TABLE | Permanently deletes a table (and thus its data) |
| DROP INDEX | Permanently deletes an index |
| DROP VIEW | Permanently deletes a view |

**TABLE 7.2** SQL Data Manipulation Commands

| COMMAND OR OPTION | DESCRIPTION |
|---|---|
| INSERT | Inserts row(s) into a table |
| SELECT | Selects attributes from rows in one or more tables or views |
| WHERE | Restricts the selection of rows based on a conditional expression |
| GROUP BY | Groups the selected rows based on one or more attributes |
| HAVING | Restricts the selection of grouped rows based on a condition |
| ORDER BY | Orders the selected rows based on one or more attributes |
| UPDATE | Modifies an attribute's values in one or more table's rows |
| DELETE | Deletes one or more rows from a table |
| COMMIT | Permanently saves data changes |
| ROLLBACK | Restores data to their original values |

# Introduction to SQL (continued)

**TABLE 7.2** SQL Data Manipulation Commands (continued)

| COMMAND OR OPTION | DESCRIPTION |
|---|---|
| COMPARISON OPERATORS | |
| =, <, >, <=, >=, <> | Used in conditional expressions |
| LOGICAL OPERATORS | |
| AND/OR/NOT | Used in conditional expressions |
| SPECIAL OPERATORS | Used in conditional expressions |
| BETWEEN | Checks whether an attribute value is within a range |
| IS NULL | Checks whether an attribute value is null |
| LIKE | Checks whether an attribute value matches a given string pattern |
| IN | Checks whether an attribute value matches any value within a value list |
| EXISTS | Checks whether a subquery returns any rows |
| DISTINCT | Limits values to unique values |
| AGGREGATE FUNCTIONS | Used with SELECT to return mathematical summaries on columns |
| COUNT | Returns the number of rows with non-null values for a given column |
| MIN | Returns the minimum attribute value found in a given column |
| MAX | Returns the maximum attribute value found in a given column |
| SUM | Returns the sum of all values for a given column |
| AVG | Returns the average of all values for a given column |

# Data Definition Commands

- Examine simple database model and database tables that will form basis for many SQL examples
- Understand data environment

# Creating the Database

- Following two tasks must be completed:
  - Create database structure
  - Create tables that will hold end-user data
- First task:
  - RDBMS creates physical files that will hold database
  - Tends to differ substantially from one RDBMS to another

# The Database Schema

- Authentication
  - Process through which DBMS verifies that only registered users are able to access database
  - Log on to RDBMS using user ID and password created by database administrator
- Schema
  - Group of database objects—such as tables and indexes—that are related to each other

# Data Types

- Data type selection is usually dictated by nature of data and by intended use
- Pay close attention to expected use of attributes for sorting and data retrieval purposes

**TABLE 7.4 Some Common SQL Data Types**

| DATA TYPE | FORMAT | COMMENTS |
|---|---|---|
| **Numeric** | NUMBER(L,D) | The declaration NUMBER(7,2) indicates numbers that will be stored with two decimal places and may be up to six digits long, including the sign and the decimal place. Examples: 12.32, -134.99. |
| | INTEGER | May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places. |
| | SMALLINT | Like INTEGER, but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT. |
| | DECIMAL(L,D) | Like the NUMBER specification, but the storage length is a *minimum* specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable. |
| **Character** | CHAR(L) | Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as "Smith" and "Katzenjammer" are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes. |
| | VARCHAR(L) or VARCHAR2(L) | Variable-length character data. The designation VARCHAR2(25) will let you store characters up to 25 characters long. However, VARCHAR will not leave unused spaces. Oracle users may use VARCHAR2 as well as VARCHAR. |
| **Date** | DATE | Stores dates in the Julian date format. |

# Creating Table Structures

- Use one line per column (attribute) definition
- Use spaces to line up attribute characteristics and constraints
- Table and attribute names are capitalized
- NOT NULL specification
- UNIQUE specification

- Primary key attributes contain both a NOT NULL and a UNIQUE specification
- RDBMS will automatically enforce referential integrity for foreign keys
- Command sequence ends with semicolon

# SQL Constraints

- NOT NULL constraint
  - Ensures that column does not accept nulls
- UNIQUE constraint
  - Ensures that all values in column are unique
- DEFAULT constraint
  - Assigns value to attribute when a new row is added to table
- CHECK constraint
  - Validates data when attribute value is entered

# SQL Indexes

- When primary key is declared, DBMS automatically creates unique index
- Often need additional indexes
- Using CREATE INDEX command, SQL indexes can be created on basis of any selected attribute
- Composite index
  - Index based on two or more attributes
  - Often used to prevent data duplication

# Data Manipulation Commands

- Adding table rows
- Saving table changes
- Listing table rows
- Updating table rows
- Restoring table contents
- Deleting table rows
- Inserting table rows with a select subquery

# Adding Table Rows

- INSERT
  - Used to enter data into table
  - Syntax:
    - INSERT INTO *columnname*
      VALUES (*value1, value2, … , valuen*);

# Adding Table Rows (continued)

- When entering values, notice that:
  - Row contents are entered between parentheses
  - Character and date values are entered between apostrophes
  - Numerical entries are not enclosed in apostrophes
  - Attribute entries are separated by commas
  - A value is required for each column
- Use NULL for unknown values

# Saving Table Changes

- Changes made to table contents are not physically saved on disk until, one of the following occurs:
  - Database is closed
  - Program is closed
  - COMMIT command is used
- Syntax:
  - COMMIT [WORK];
- Will permanently save any changes made to any table in the database

# Listing Table Rows

- SELECT
  - Used to list contents of table
  - Syntax:
    - SELECT *columnlist*
      FROM *tablename*;
- *Columnlist* represents one or more attributes, separated by commas
- Asterisk can be used as wildcard character to list all attributes

# Updating Table Rows

- UPDATE
  - Modify data in a table
  - Syntax:
    - UPDATE *tablename*
      SET *columnname* = *expression* [, *columnname* = *expression*]
      [WHERE *conditionlist*];
- If more than one attribute is to be updated in row, separate corrections with commas

# Restoring Table Contents

- ROLLBACK
  - Used to restore database to its previous condition
  - Only applicable if COMMIT command has not been used to permanently store changes in database
- Syntax:
  - ROLLBACK;
- COMMIT and ROLLBACK only work with data manipulation commands that are used to add, modify, or delete table rows

# Deleting Table Rows

- DELETE
  - Deletes a table row
  - Syntax:
    - DELETE FROM *tablename*
      [WHERE *conditionlist* ];
- WHERE condition is optional
- If WHERE condition is not specified, all rows from specified table will be deleted

# Inserting Table Rows with a Select Subquery

- INSERT
  - Inserts multiple rows from another table (source)
  - Uses SELECT subquery
    - Query that is embedded (or nested) inside another query
    - Executed first
  - Syntax:
    - INSERT INTO *tablename* SELECT *columnlist* FROM *tablename*;

# Selecting Rows with Conditional Restrictions

- Select partial table contents by placing restrictions on rows to be included in output
  - Add conditional restrictions to SELECT statement, using WHERE clause
- Syntax:
  - SELECT *columnlist*
    FROM *tablelist*
    [ WHERE *conditionlist* ] ;

# Selecting Rows with Conditional Restrictions (continued)

**TABLE 7.6 Comparison Operators**

| SYMBOL | MEANING |
|---|---|
| = | Equal to |
| < | Less than |
| <= | Less than or equal to |
| > | Greater than |
| >= | Greater than or equal to |
| <> or != | Not equal to |

# Arithmetic Operators: The Rule of Precedence

- Perform operations within parentheses
- Perform power operations
- Perform multiplications and divisions
- Perform additions and subtractions

# Arithmetic Operators: The Rule of Precedence (continued)

TABLE 7.7 The Arithmetic Operators

| ARITHMETIC OPERATOR | DESCRIPTION |
|---|---|
| + | Add |
| - | Subtract |
| * | Multiply |
| / | Divide |
| ^ | Raise to the power of (Some applications use ** instead of ^.) |

# Special Operators

- BETWEEN
  - Used to check whether attribute value is within a range
- IS NULL
  - Used to check whether attribute value is null
- LIKE
  - Used to check whether attribute value matches given string pattern
- IN
  - Used to check whether attribute value matches any value within a value list
- EXISTS
  - Used to check if subquery returns any rows

# Advanced Data Definition Commands

- All changes in table structure are made by using ALTER command
  - Followed by keyword that produces specific change
  - Following three options are available:
    - ADD
    - MODIFY
    - DROP

# Changing a Column's Data Type

- ALTER can be used to change data type
- Some RDBMSs (such as Oracle) do not permit changes to data types unless column to be changed is empty

# Changing a Column's Data Characteristics

- Use ALTER to change data characteristics
- If column to be changed already contains data, changes in column's characteristics are permitted if those changes do not alter the data type

# Adding a Column

- Use ALTER to add column
  - Do not include the NOT NULL clause for new column

# Dropping a Column

- Use ALTER to drop column
  - Some RDBMSs impose restrictions on the deletion of an attribute

# Copying Parts of Tables

- SQL permits copying contents of selected table columns so that the data need not be reentered manually into newly created table(s)
- First create the PART table structure
- Next add rows to new PART table using PRODUCT table rows

# Adding Primary and Foreign Key Designations

- When table is copied, integrity rules do not copy, so primary and foreign keys need to be manually defined on new table
- User ALTER TABLE command
  - Syntax:
    - ALTER TABLE *tablename* ADD PRIMARY KEY(*fieldname*);
    - For foreign key, use FOREIGN KEY in place of PRIMARY KEY

# Deleting a Table from the Database

- DROP
  - Deletes table from database
  - Syntax:
    - DROP TABLE *tablename*;

# Advanced Select Queries

- SQL provides useful functions that can:
  - Count
  - Find minimum and maximum values
  - Calculate averages
- SQL allows user to limit queries to only those entries having no duplicates or entries whose duplicates may be grouped

# Aggregate Functions

**TABLE 7.8 Some Basic SQL Aggregate Functions**

| FUNCTION | OUTPUT |
|---|---|
| COUNT | The number of rows containing non-null values |
| MIN | The minimum attribute value encountered in a given column |
| MAX | The maximum attribute value encountered in a given column |
| SUM | The sum of all values for a given column |
| AVG | The arithmetic mean (average) for a specified column |

# Virtual Tables: Creating a View

- View is virtual table based on SELECT query
  - Can contain columns, computed columns, aliases, and aggregate functions from one or more tables
- Base tables are tables on which view is based
- Create view by using CREATE VIEW command

# Joining Database Tables

- Ability to combine (join) tables on common attributes is most important distinction between relational database and other databases
- Join is performed when data are retrieved from more than one table at a time
- Join is generally composed of an equality comparison between foreign key and primary key of related tables

# Joining Tables with an Alias

- Alias can be used to identify source table
- Any legal table name can be used as alias
- Add alias after table name in FROM clause
  - FROM *tablename alias*

# Summary

- SQL commands can be divided into two overall categories:
  - Data definition language commands
  - Data manipulation language commands
- The ANSI standard data types are supported by all RDBMS vendors in different ways
- Basic data definition commands allow you to create tables, indexes, and views
- DML commands allow you to add, modify, and delete rows from tables
- The basic DML commands are SELECT, INSERT, UPDATE, DELETE, COMMIT, and ROLLBACK
- INSERT command is used to add new rows to tables
- SELECT statement is main data retrieval command in SQL

# Summary (continued)

- Many SQL constraints can be used with columns
- The column list represents one or more column names separated by commas
- WHERE clause can be used with SELECT, UPDATE, and DELETE statements to restrict rows affected by the DDL command
- Aggregate functions
  - Special functions that perform arithmetic computations over a set of rows
- ORDER BY clause
  - Used to sort output of SELECT statement
  - Can sort by one or more columns and use either an ascending or descending order
- Join output of multiple tables with SELECT statement
- Natural join uses join condition to match only rows with equal values in specified columns
- Right outer join and left outer join used to select rows that have no matching values in other related table