•Functions always have the same basic syntax:
  <function name>(function arguments)
•There are hundreds of built-in functions.
•We will be concerned with two broad types of functions:
•Scalar functions: Scalar functions operate on a single row at a time.
•Aggregate functions: Aggregate functions operate on multiple rows at a time.

Using Distinct in Aggregate Functions
•You can use the DISTINCT keyword with aggregate functions.
•Doing so means the function will ignore duplicate values in its calculation.

```
SELECT COUNT(DISTINCT StudentKey) AS [Unduplicated]
FROM Session
```

Group By Example
•When a SELECT clause includes an aggregate function and columns that are not a part of that function, you must use the GROUP BY keywords to group by each of the non-included columns.
•This is necessary because you are mixing functions that operate on multiple rows with columns that refer to values in individual rows only.

```
SELECT Key, COUNT(SessionTimeKey) AS [Total Sessions]
FROM Session
GROUP BY Key
```

Having Example
•The HAVING keyword is used when there is an aggregate function in the criteria of a query.

```
SELECT Key, COUNT(SessionTimeKey) AS [Total Sessions]
FROM Session
GROUP BY Key
HAVING COUNT(SessionTimeKey)<4
```

EXISTS and NOT EXISTS
•EXISTS and NOT EXISTS are for use only with subqueries.
•Produce a simple true/false result.
•True if and only if there exists at least one row in result table returned by subquery.
•False if subquery returns an empty result table.
•NOT EXISTS is the opposite of EXISTS.
•As (NOT) EXISTS check only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns.
•Common for subqueries following (NOT) EXISTS to be of form:
  (SELECT * ...)

Query using EXISTS
Find all staff who work in a London branch.

```
SELECT staffNo, fName, lName, position
FROM Staff s
WHERE EXISTS
(SELECT *
 FROM Branch b
 WHERE s.branchNo = b.branchNo AND
       city = 'London');
```

•Note, search condition s.branchNo = b.branchNo is necessary to consider correct branch record for each member of staff.
•If omitted, would get all staff records listed out because subquery:
SELECT * FROM Branch WHERE city='London'
•would always be true and query would be:
SELECT staffNo, fName, lName, position FROM Staff
WHERE true;
•Could also write this query using join construct:
SELECT staffNo, fName, lName, position
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND
city = 'London';

Joins
•In database design and normalization, the data are broken into several discrete tables.
•Joins are the mechanism for recombining the data into one result set.
•We will look at three kinds of joins:
•Inner joins
•Equi joins
•Outer joins

Multi-Table Queries
•Can use subqueries provided result columns come from same table.
•If result columns come from more than one table must use a join.
•To perform join, include more than one table in FROM clause.
•Use comma as separator and typically include WHERE clause to specify join column(s).
•Also possible to use an alias for a table named in FROM clause.
•Alias is separated from table name with a space.
•Alias can be used to qualify column names when there is ambiguity.

Simple Join
List names of all clients who have viewed a property along with any comment supplied.
```
SELECT c.clientNo, fName, lName,
        propertyNo, comment
FROM Client c, Viewing v
WHERE c.clientNo = v.clientNo;
```
•Only those rows from both tables that have identical values in the clientNo columns (c.clientNo = v.clientNo) are included in result.
•Equivalent to equi-join in relational algebra.

Basic INNER JOIN Syntax
```
SELECT <column1, column2>
FROM <table1>
INNER JOIN <table2>
ON <table1>.<column>=<table2>.<column>
```
•Inner joins return related records from each of the tables joined.
```
SELECT LastName,
FirstName,
SessionDateKey,
SessionTimeKey,
StudentKey
SessionStatus
```

```
FROM
INNER JOIN Session
ON .Key = Session.Key
```

Alternative JOIN Constructs

•SQL provides alternative ways to specify joins:

```
FROM Client c JOIN Viewing v ON c.clientNo = v.clientNo
FROM Client JOIN Viewing USING clientNo
FROM Client NATURAL JOIN Viewing
```

•In each case, FROM replaces original FROM and WHERE. However, first produces table with two identical clientNo columns.

Equi Joins

•Equi joins present an alternative way to perform inner joins. Some older RDMSs only support this alternative form. The example below also uses an alias for the table name.

```
SELECT t.Key,
LastName,
FirstName,
SessionDateKey,
SessionTimeKey,
StudentKey
FROM  t,
Session s
WHERE t.Key = s.Key
AND LastName = 'Brown'
```

Three Table Join

For each branch, list staff who manage properties, including city in which branch is located and properties they manage.

```
SELECT b.branchNo, b.city, s.staffNo, fName, lName,
            propertyNo
FROM Branch b, Staff s, PropertyForRent p
WHERE b.branchNo = s.branchNo AND
            s.staffNo = p.staffNo
ORDER BY b.branchNo, s.staffNo, propertyNo;
```

Sorting a join

For each branch, list numbers and names of staff who manage properties, and properties they manage.

```
SELECT s.branchNo, s.staffNo, fName, lName,
          propertyNo
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
ORDER BY s.branchNo, s.staffNo, propertyNo;
```

Multiple Grouping Columns

Find number of properties handled by each staff member.

```
SELECT s.branchNo, s.staffNo, COUNT(*) AS myCount
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo
ORDER BY s.branchNo, s.staffNo;
```

Computing a Join

Procedure for generating results of a join are:

1. Form Cartesian product of the tables named in FROM clause.
2. If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition.
3. For each remaining row, determine value of each item in SELECT list to produce a single row in result table.
4. If DISTINCT has been specified, eliminate any duplicate rows from the result table.
5. If there is an ORDER BY clause, sort result table as required.

•SQL provides special format of SELECT for Cartesian product:

```
SELECT [DISTINCT | ALL] {* | columnList}
FROM Table1 CROSS JOIN Table2
```

Outer Joins

•If one row of a joined table is unmatched, row is omitted from result table.
•Outer join operations retain rows that do not satisfy the join condition.
•Consider following tables:
•The (inner) join of these two tables:

```
SELECT b.*, p.*
FROM Branch1 b, PropertyForRent1 p
WHERE b.bCity = p.pCity;
```

•Result table has two rows where cities are same.
•There are no rows corresponding to branches in Bristol and Aberdeen.
•To include unmatched rows in result table, use an Outer join.

Left Outer Join

List branches and properties that are in same city along with any unmatched properties.

```
SELECT b.*, p.*
FROM Branch1 b LEFT JOIN
PropertyForRent1 p ON b.bCity = p.pCity;
```

•Includes those rows of first (left) table unmatched with rows from second (right) table.
•Columns from second table are filled with NULLs.

Right Outer Join

List branches and properties in same city and any unmatched properties.

```
SELECT b.*, p.*
FROM Branch1 b RIGHT JOIN
PropertyForRent1 p ON b.bCity = p.pCity;
```

•Right Outer join includes those rows of second (right) table that are unmatched with rows from first (left) table.
•Columns from first table are filled with NULLs.

Full Outer Join

List branches and properties in same city and any unmatched branches or properties. (on both sides)

```
SELECT b.*, p.*
FROM Branch1 b FULL JOIN
PropertyForRent1 p ON b.bCity = p.pCity;
```

•Includes rows that are unmatched in both tables.
•Unmatched columns are filled with NULLs.

Outer joins return records that are not matched. The following query returns s that have no sessions scheduled.

```
SELECT <column1>, <column2>
FROM <table1>
LEFT OUTER JOIN <table2>
ON <table1>.<column>=<table2>.<column>
```

```
SELECT t.Key,
LastName,
SessionDateKey
FROM  t
LEFT OUTER JOIN Session s
ON t.Key = s.Key
WHERE SessionDateKey IS Null
```