This reading describes alternative ways of expressing joins in SQL. We do not recommend using the techniques described in this reading, but you should familiarize yourself with them so you can read and understand SQL queries that use them.

SQL-92-Style Joins and SQL-89-Style Joins

In the video lectures describing joins in SQL, the following join syntax is used:

```
SELECT ...
  FROM toys JOIN makers
    ON toys.maker_id = makers.id;
```

Notice the **JOIN** keyword between the table names, and the **ON** keyword followed by the join condition. This is called a *SQL-92-style join*, or *explicit join syntax*, and it is usually considered to be the best syntax to use for joins in SQL.

However, many SQL engines also support another join syntax, called the *SQL-89-style join*, or *implicit join syntax*. In this syntax, you use a comma-separated list of table names in the **FROM** clause, and you specify the join condition in the **WHERE** clause:

```
SELECT ...
  FROM toys, makers
    WHERE toys.maker_id = makers.id;
```

With most SQL engines, this join query returns exactly the same result as the previous one.

With both join styles, you can use table aliases (**t** and **m** in this example):

```
SELECT ...
  FROM toys AS t JOIN makers AS m
    ON t.maker_id = m.id;


SELECT ...
  FROM toys AS t, makers AS m
  WHERE t.maker_id = m.id;
```

With both styles, the **AS** keyword before each table alias is optional.

When you use a SQL-89-style join, the SQL engine always performs an *inner* join. With this syntax, there is no way to specify any other type of join. If you want to use one of the other types of joins (left outer, right outer, full outer), then you must use a SQL-92-style join. Because of this limitation, and because the SQL-89-style join syntax makes it harder to understand the intent of the query, we recommend using SQL-92-style joins.

Unqualified Column References in Join Condition

In the join condition that comes after the **ON** keyword in a join query, the references to the corresponding columns are typically *qualified* with table names or table aliases. For example, when joining the toys table (alias **t**) and makers table (alias **m**), the join condition is specified as:

```
ON t.maker_id = m.id
```

However, in the case where a bare column name unambiguously identifies a column, most SQL engines allow you to use a bare column name. For example, since there is no column named **maker_id** in the makers table, the table alias **t** is not required in this join condition. So you could specify the join condition as:

```
ON maker_id = m.id
```

But because there are columns named **id** in both tables, the table alias **m** is required in this join condition. If you omit the table alias **m**, then the SQL engine will throw an error indicating that the column reference **id** is ambiguous.

In join conditions, we recommend *always* qualifying column names with table names or table aliases, whether or not they are strictly required. Doing this makes your queries safer and clearer.

The USING Keyword

In some join queries, the names of the two corresponding columns in the join condition are identical. For example, in this query, the corresponding columns in the employees and offices table are both named **office_id**:

```
SELECT ...
  FROM employees e JOIN offices o
    ON e.office_id = o.office_id;
```

When the corresponding columns in the join condition have identical names, some SQL engines allow you to use a shorthand notation to specify the join condition. Instead of using the **ON** keyword and specifying the condition as an equality expression, you use the **USING** keyword and specify the common join key column name in parentheses after **USING**:

```
SELECT ...
  FROM employees e JOIN offices o
    USING (office_id);
```

Natural Joins

When the corresponding columns in the join condition have identical names, some SQL engines will allow you to omit the join condition, and will automatically join the tables on all the pairs of columns that have identical names in the left and right tables. To make a SQL engine do this, you need to specify the keyword **NATURAL** before the other join keywords. For example:

```
SELECT ...
  FROM employees e NATURAL JOIN offices o;
```

MySQL and PostgreSQL support natural joins, but Hive and Impala do not. In the SQL engines that support it, you can use the keyword **NATURAL** with any type of join; for example: **NATURAL LEFT OUTER JOIN** or **NATURAL INNER JOIN**.

Omitting Join Conditions

What happens if you attempt to perform a join without specifying the join condition, and you do *not* specify **NATURAL** before the join keywords?

For example, you might run a query like this:

```
SELECT *
  FROM toys JOIN makers;
```

Notice that no join condition is specified. With some SQL engines (including PostgreSQL), this throws an error. But with other SQL engines (including Impala, Hive, and MySQL) this performs what's called a *cross join*. In a cross join, the SQL engine iterates through each row in the table on the left side and combines it with every row in the table on the right side. So the result set includes every possible combination of the rows in the left table and the rows in the right table. The number of rows in the result set is the product (multiplication) of the number of rows in the left table and the number of rows in the right table (in this example, 3 x 3 = 9):

id	name	price	maker_id	id	name	city
21	Lite-Brite	14.47	105	105	Hasbro	Pawtucket, RI
21	Lite-Brite	14.47	105	106	Ohio Art Company	Bryan, OH
21	Lite-Brite	14.47	105	107	Mattel	Segundo, CA
22	Mr. Potato Head	11.50	105	105	Hasbro	Pawtucket, RI
22	Mr. Potato Head	11.50	105	106	Ohio Art Company	Bryan, OH
22	Mr. Potato Head	11.50	105	107	Mattel	Segundo, CA
23	Etch A Sketch	29.99	106	105	Hasbro	Pawtucket, RI
23	Etch A Sketch	29.99	106	106	Ohio Art Company	Bryan, OH
23	Etch A Sketch	29.99	106	107	Mattel	Segundo, CA

In most cases, the result of a cross join is meaningless. The rows of the result contain values with no correspondence. If you don't realize that you have performed a cross join, you might be misled by the results. In addition, when performed on large tables, a cross join can return a dangerously large number of rows.

There are some specific cases when cross joins are useful, and in most SQL dialects, you can explicitly specify **CROSS JOIN** in your SQL statement to make it clear that you are performing a cross join. This is discussed in a video in the upcoming honors lesson.

So unless you intend to perform a cross join, and you understand the risks of this and how to interpret the output, we recommend specifying the join condition in every join query.