MySQL for Developers

SQL-4501 Release 2.2

D61830GC10
Edition 1.0

ORACLE

Day 2

- DRL
    - Joins
    - Subquery
- Views
- Indexes
- Meta Data
- DCL

Joins

Joins

- What is a join operation?
  - A join is an operation upon two tables
  - Creates new rows by combining (joining) rows from two tables
  - Combined rows form a new table

Order of tables is not of real importance

- When creating a Cartesian product, table processing order influences the order of columns and rows
- This is not that important though:
  - The row order is not of importance from a relational point of view
  - The column order is not that important as long as each column can still be identified
- Changing the order in which tables are processed does not change the information content of the product table

Joins and Foreign Keys

- In many cases, rows are joined according to a foreign key
  - In the example, rows were retained in case the `CountryCode` column in the `SimpleCity` table matched the `Code` column in the `SimpleCountry` table
- Joining based on a foreign key is a very common pattern
  - For each row in the referencing table, the join operation 'looks up' data in the referenced table

Joining in SQL using a Cartesian product

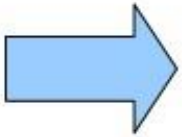
- Cartesian product using the 'comma join'
- Separate multiple table names with a comma (“,“)


```
SELECT *
FROM   SimpleCity, SimpleCountry;
```

```
+--------+----------+-------------+------+----------------+---------+
| CityID | CityName | CountryCode | Code | CountryName    | Capital |
+--------+----------+-------------+------+----------------+---------+
|    456 | London   | GBR         | CAN  | Canada         |    1822 |
|   1820 | London   | CAN         | CAN  | Canada         |    1822 |
|    456 | London   | GBR         | GBR  | United Kingdom |     456 |
|   1820 | London   | CAN         | GBR  | United Kingdom |     456 |
+--------+----------+-------------+------+----------------+---------+
```

Using WHERE to retain matching rows

- The WHERE clause can be used to retain only those rows that satisfy a condition
  - We can write a condition to require matching SimpleCity and SimpleCountry rows

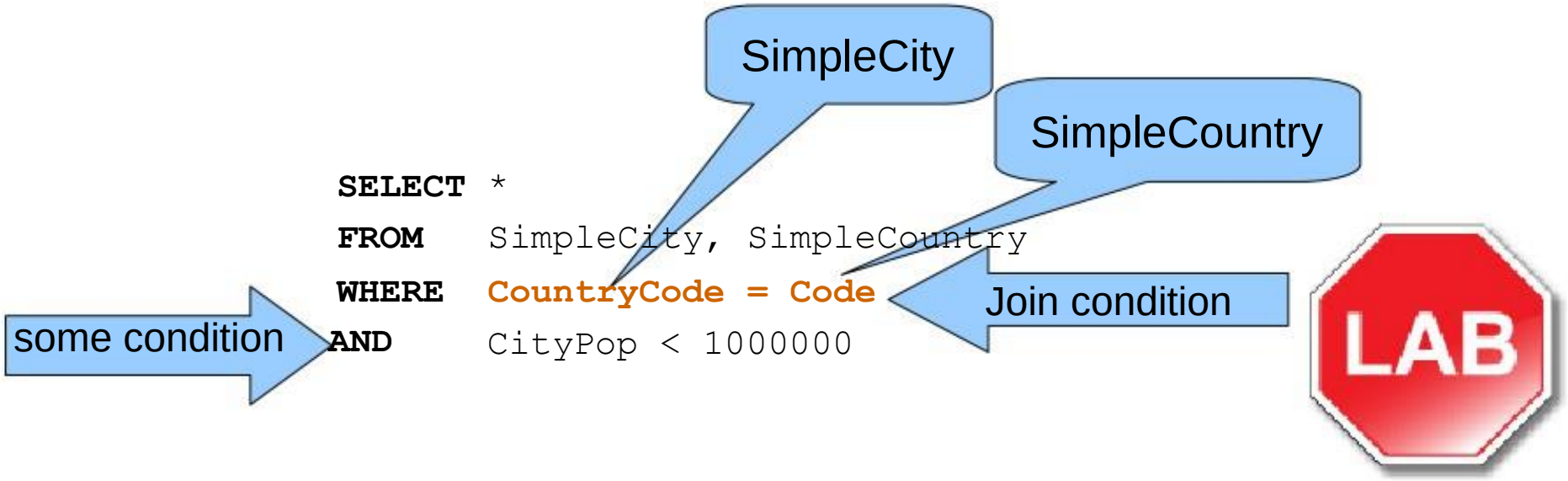


```
SELECT *
FROM   SimpleCity, SimpleCountry
WHERE  CountryCode = Code
```

```
+--------+----------+-------------+------+----------------+---------+
| CityID | CityName | CountryCode | Code | CountryName    | Capital |
+--------+----------+-------------+------+----------------+---------+
|   1820 | London   | CAN         | CAN  | Canada         |    1822 |
|    456 | London   | GBR         | GBR  | United Kingdom |     456 |
+--------+----------+-------------+------+----------------+---------+
```

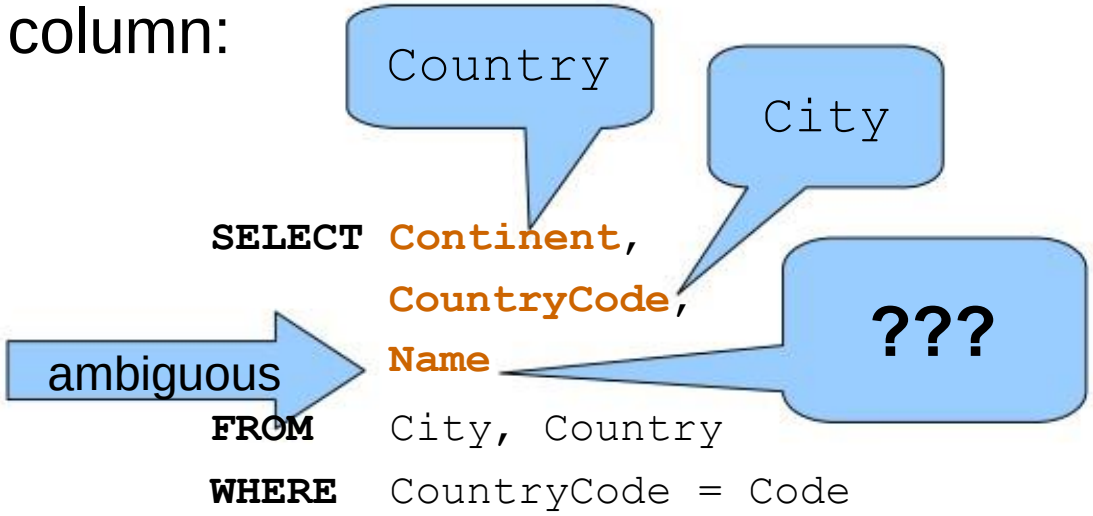
The Join Condition

- The WHERE is 'just' an ordinary WHERE clause
  - The WHERE clause can contain any condition
  - requiring matching rows is 'just' a condition
  - Still, we like to consider the condition special
- A *join condition* is the condition that compares the columns of two joined tables



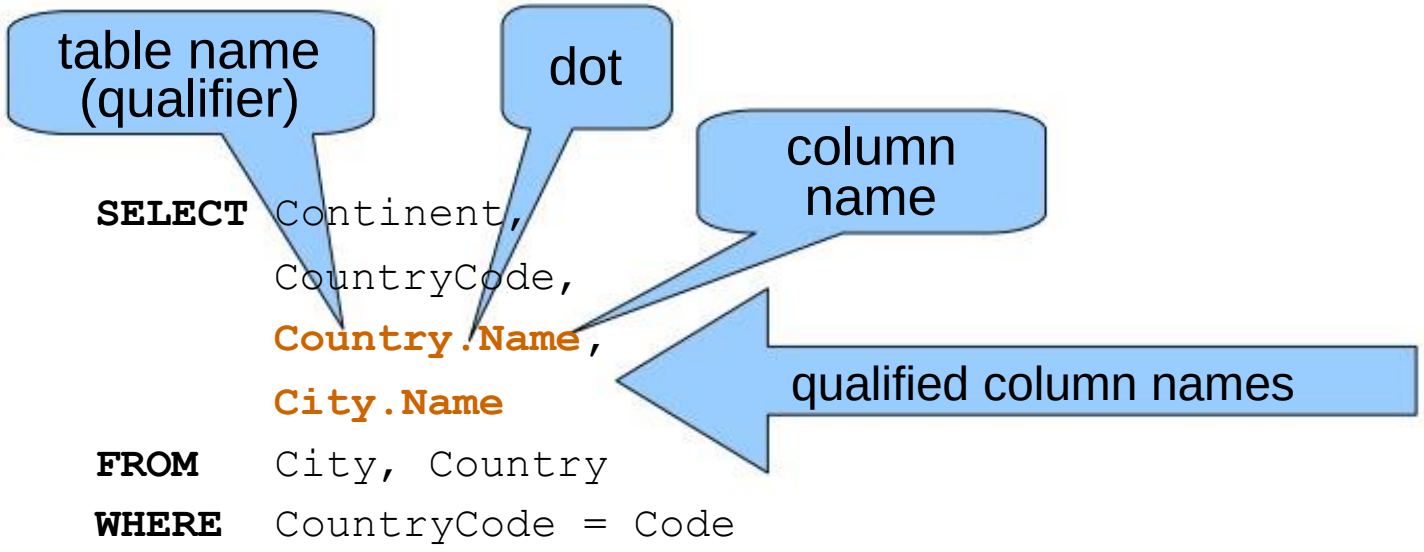
Ambiguous Column Names (1/2)

- Potential ambiguity when joining tables
  - A joined table may contain a column that has a name identical to that of a column in the table it is joined with
- Column name alone may not be enough to identify a column:



Ambiguous Column Names (2/2)

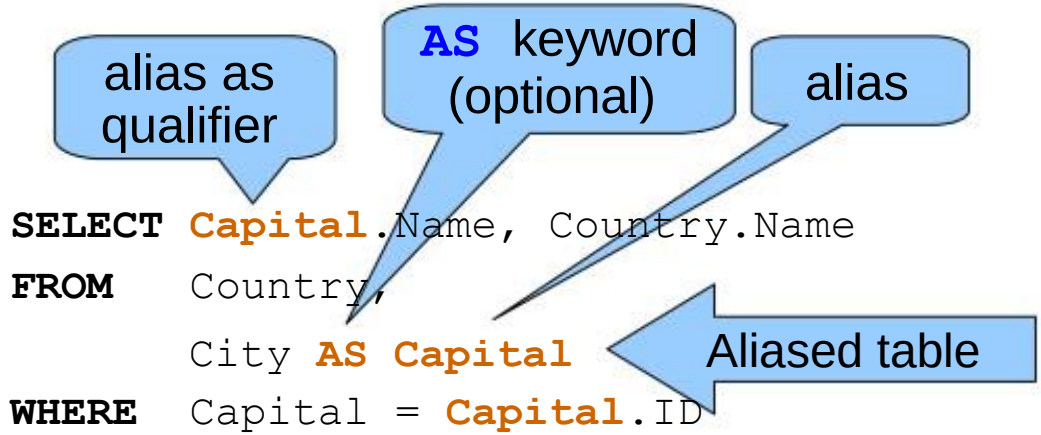
- Avoid ambiguity by ***qualifying*** column names
- Separate table name and column name with a dot



- Qualified columns can appear almost anywhere
- You may also qualify unambiguous columns

Table Aliases

- In SQL statements, tables can be given an ***alias***
  - Alternative name for local use in the statement
- When qualifying a column of an aliased table, the alias must be used as qualifier - not the table name
- Alias follows after the table name
- Optionally, separate table name and alias with the keyword **AS**: *<table-reference>* [**AS**] *<alias>*



Basic Join Syntax

- SQL offers the **`JOIN`** syntax
  - Allows separation of the join condition from other conditions
- Syntax:

```
<table-ref> [<join-type>] JOIN <table-ref>
 ON <join-condition>
```

- Example:

```
SELECT *
 FROM   SimpleCity JOIN SimpleCountry
 ON     CountryCode = Code
 WHERE  CityPop < 1000000
```

Join condition

Non-join condition

- **`ON`** clause still allows non-join conditions
  - Better to put those in the **`WHERE`**

INNER JOIN

- The inner join operation is characterized by the fact that its result contains only rows for which the join condition is satisfied
  - Previous comma join and **JOIN** examples are all inner joins
- Explicit syntax for the inner join operation:
  - Use **INNER** keyword before the **JOIN** keyword
  - If the *<join-type>* is omitted, **INNER** is implied
- Example:

```
SELECT *
FROM   SimpleCity INNER JOIN SimpleCountry
ON     CountryCode = Code
```

Inner Join to Find the Capital City

```
SELECT CountryName, CityName
FROM   SimpleCountry INNER JOIN SimpleCity
ON     Capital = CityID;
```

```
+----------------+----------+
| CountryName    | CityName |
+----------------+----------+
| United Kingdom | London   |
+----------------+----------+
```

The row for `'Canada'` is missing

Inner Join Discards the Unmatched Row

- The `Capital` column for the `'Canada'` row in `SimpleCountry` does not match any `CityID` column in `SimpleCity`
  - The join condition is not satisfied
  - The `'Canada'` row is discarded and does not appear in the join result

SimpleCountry			SimpleCity		
Code	CountryName	Capital	CityID	CityName	CountryCode
CAN	Canada	1822	456	London	GBR
GBR	United Kingdom	456	456	London	GBR
CAN	Canada	1822	1820	London	CAN
GBR	United Kingdom	456	1820	London	CAN

Outer Join Operation

- What if we want a list of *all* countries, and if possible, the capital city?
  - Retain the row from `SimpleCountry` even if no corresponding capital was found in `SimpleCity`
- An outer join operation achieves exactly that

The LEFT OUTER JOIN Syntax

- Syntax:

```
<left-table> LEFT [OUTER] JOIN <right-table>
ON <join-condition>
```

- Note that the **OUTER** keyword is optional
  - Usually omitted
- The **LEFT OUTER JOIN**:
  - Returns all rows that match the join condition
  - Retains unmatched rows from *<left-table>*
  - Substitutes **NULL** for *<right-table>* columns for each unmatched row from *<left-table>*

LEFT OUTER JOIN Example

- Example query:

```
SELECT     CountryName, CityName
FROM       SimpleCountry
LEFT JOIN  SimpleCity
ON         Capital = CityID;
```

```
+-----------------+----------+
| CountryName     | CityName |
+-----------------+----------+
| Canada          | NULL     |
| United Kingdom  | London   |
+-----------------+----------+
```

RIGHT OUTER JOIN Syntax

- Syntax:

```
<left-table> RIGHT [OUTER] JOIN <right-table>
ON <join-condition>
```

- Same as **LEFT OUTER JOIN** syntax except that the keyword **RIGHT** is used instead of **LEFT**
- The **RIGHT OUTER JOIN**:
  - Returns all rows that match the join condition
  - Returns unmatched rows from *<right-table>*
  - Substitutes **NULL** for *<left-table>* columns for each unmatched row in *<right-table>*

RIGHT OUTER JOIN Example

- Example query:

```
SELECT     CountryName, CityName
FROM       SimpleCountry
RIGHT JOIN SimpleCity
ON         Capital = CityID;
```

```
+-----------------+----------+
| CountryName     | CityName |
+-----------------+----------+
| NULL            | New York |
| United Kingdom  | London   |
+-----------------+----------+
```

FULL OUTER JOIN Example

- Example query:

```
SELECT      CountryName, CityName
FROM        SimpleCountry
RIGHT JOIN SimpleCity
ON          Capital = CityID;

UNION

SELECT      CountryName, CityName
FROM        SimpleCountry

ON          Capital = CityID;
```

Equijoin and Non-equijoin

- Equijoin:
  - join condition contains only column comparisons using the equals operator
- Non-equijoin
  - Anything that is not an equijoin
- **`BETWEEN...AND`** join


```
SELECT Employee.ID, Bonus.Amount
FROM   Employee INNER JOIN Bonus
ON     Employee.Salary
       BETWEEN Bonus.LowerSalaryBound
       AND     Bonus.UpperSalaryBound
```

Subquery

Subquery Overview

- Query Nested Inside Another Query
- Enclosed in Parenthesis ()
- Example

```
SELECT  Language                        -- outer SELECT expression
FROM    CountryLanguage
WHERE   CountryCode = (                 -- left parenthesis - starts subquery
            SELECT Code                 -- subquery SELECT expression
            FROM   Country
            WHERE  Name = 'Finland'
        )                               -- right parenthesis - ends subquery
+----------+
| Language |
+----------+
| Estonian |
| Finnish  |
| Russian  |
| Saame    |
| Swedish  |
+----------+
```

Table Subqueries

- Subqueries in the FROM clause
  - The result set of a subquery in the FROM clause is treated in the same way as results retrieved from base tables or views that are referred to in the FROM clause

```
SELECT * FROM (
                SELECT Code, Name FROM Country
                 WHERE  IndepYear IS NOT NULL
               ) AS IndependentCountries;
```

  - Table alias is required for all subqueries that appear in the FROM clause
    - Omitting the alias will result in an error:

```
ERROR 1248 (42000): Every derived table must
have its own alias
```

IN Operator

- Evaluates to true if there is at least one occurrence in the result set derived from the subquery that is equal to the left hand operand

```
SELECT * FROM City

WHERE CountryCode  IN
                (SELECT Code
                 FROM Country
                 WHERE Continent = 'Asia');
```

Views

What are Views?

- View descriptions
  - Database Object Defined in Terms of a SELECT Statement
  - Virtual Table
  - Selected from Base Tables or Views
  - Updatable
- Benefits
  - Access to data becomes simplified
    - Can be used to perform a calculation and display its result
    - Can be used to select a restricted set of rows
    - Can be used for selecting data from multiple tables

The CREATE VIEW Statement

- General syntax

```
CREATE [OR REPLACE] VIEW view_name [(column_list)]

    AS select_statement
    [WITH CHECK OPTION]
```

- Optional parts of a CREATE VIEW statement
  - OR REPLACE
  - ALGORITHM
  - WITH CHECK OPTION

CREATE VIEW with SELECT

- Example

```
CREATE VIEW CityView
AS
SELECT ID, Name FROM City;
```

```
SELECT * FROM CityView;
+------+-----------------------------------+
| ID   | Name                              |
+------+-----------------------------------+
|    1 | Kabul                             |
|    2 | Qandahar                          |
|    3 | Herat                             |
```

WITH CHECK OPTION (1/2)

- Checks the WHERE conditions for updates

```
CREATE VIEW LargePop AS

SELECT Name, Population FROM Country

WHERE Population >= 100000000

WITH CHECK OPTION;
```

```
SELECT * FROM LargePop;
+--------------------+------------+
| Name               | Population |
+--------------------+------------+
| Bangladesh         |  129155000 |
| Brazil             |  170115000 |

| Russian Federation |  146934000 |
| United States      |  278357000 |
+--------------------+------------+
10 rows in set (#.## sec)
```

WITH CHECK OPTION (2/2)

- Update examples

```
UPDATE LargePop SET Population = Population + 1
WHERE Name = 'Nigeria';
Query OK, 1 row affected (#.## sec)
Rows matched: 1  Changed: 1  Warnings: 0

SELECT * FROM LargePop WHERE Name = 'Nigeria';
+---------+------------+
| Name    | Population |
+---------+------------+
| Nigeria |111506001 |
+---------+------------+
1 row in set (#.## sec)

UPDATE LargePop SET Population = 99999999
WHERE Name = 'Nigeria';
ERROR 1369 (HY000): CHECK OPTION failed 'world.LargePop'
```

Altering Views

- Changing the definition of an existing view
- Use ALTER VIEW statement
- Example

```
ALTER VIEW LargePop AS

SELECT Name, Population FROM Country
WHERE Population >= 100000000;
```

- Can also use CREATE VIEW to change a view

Dropping Views

- Deletes one or more views
- Use DROP VIEW statement
  - IF EXISTS clause
- Example

```
DROP VIEW IF EXISTS v1, v2;
Query OK, 0 rows affected, 1 warning (#.## sec)

SHOW WARNINGS;
+-------+------+--------------------------+
| Level | Code | Message                  |
+-------+------+--------------------------+
| Note  | 1051 | Unknown table 'world.v2' |
+-------+------+--------------------------+
1 row in set (#.## sec)
```