SQL Refresher

Dr. Villanes

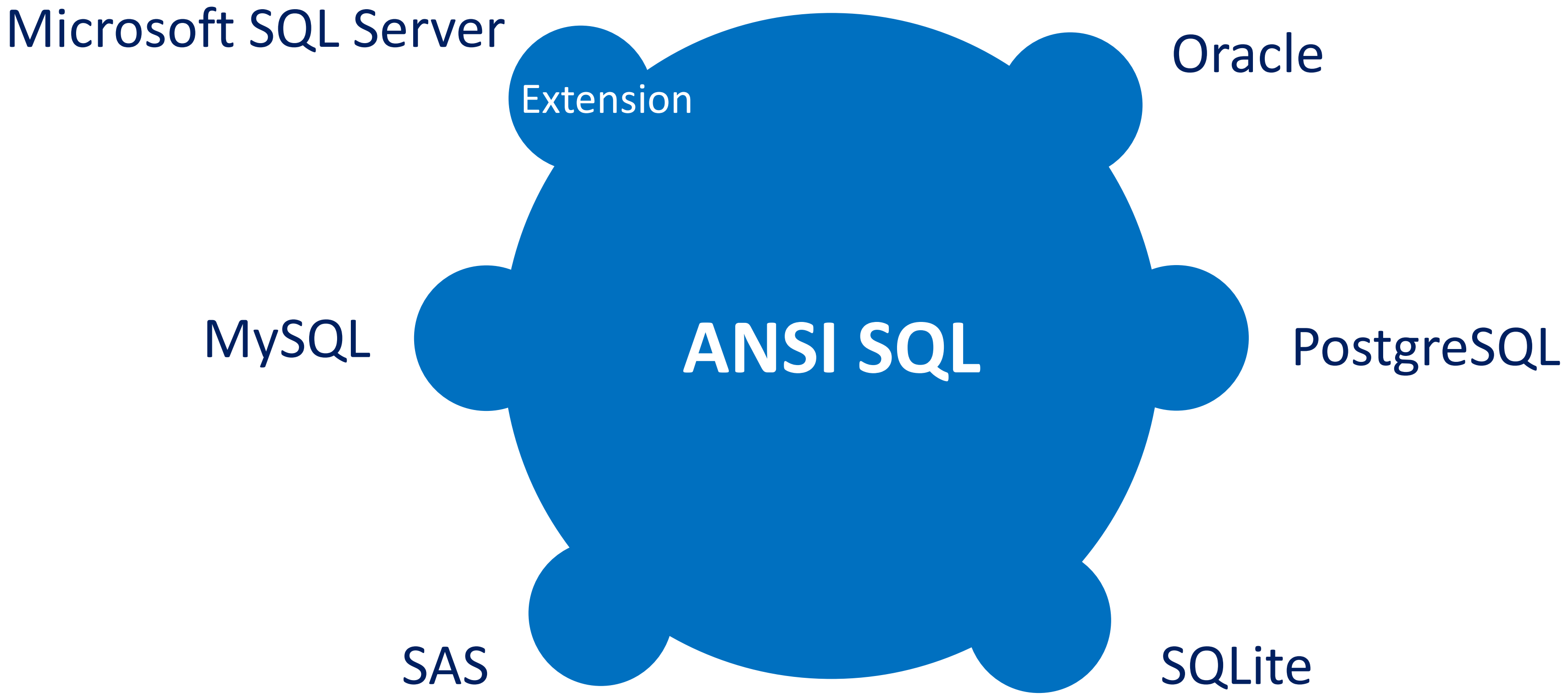
What is SQL? ((Actual interview question))

- *Structured Query Language* (SQL) is a **standardized** language originally designed as a relational database query tool.
- SQL became a **standard** of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.
- SQL is currently used in **many** software products to retrieve and update data.
- Despite the existence of the standards, most SQL code is **not completely portable** among different database systems without adjustments.

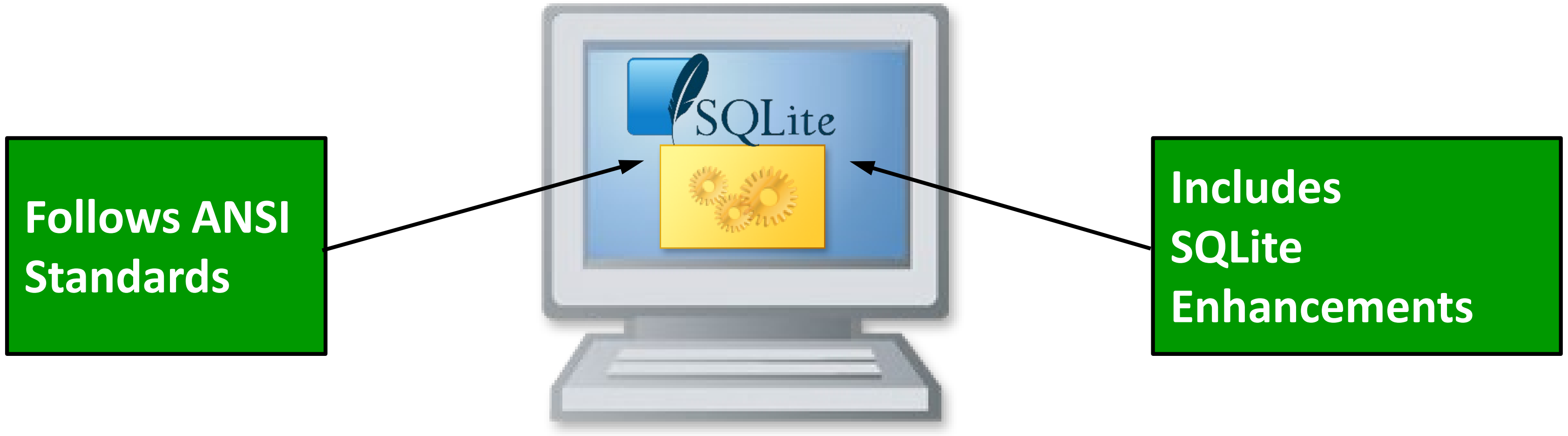
How is SQL currently being used in **many** software products?

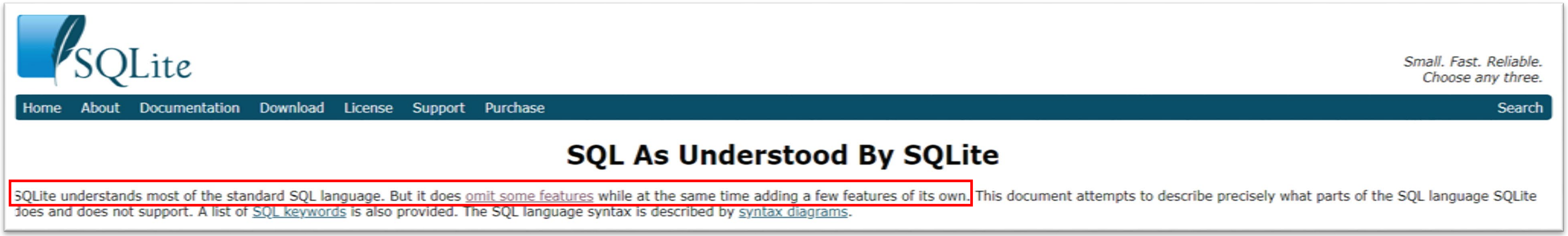
- Although most database systems use SQL, **most of them also have their own additional proprietary extensions** that are usually only used on their system.
- However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

SQL is a standard… BUT…



For example…





SQL As Understood By SQLite

SQLite understands most of the standard SQL language. But it does omit some features while at the same time adding a few features of its own. This document attempts to describe precisely what parts of the SQL language SQLite does and does not support. A list of SQL keywords is also provided. The SQL language syntax is described by syntax diagrams.

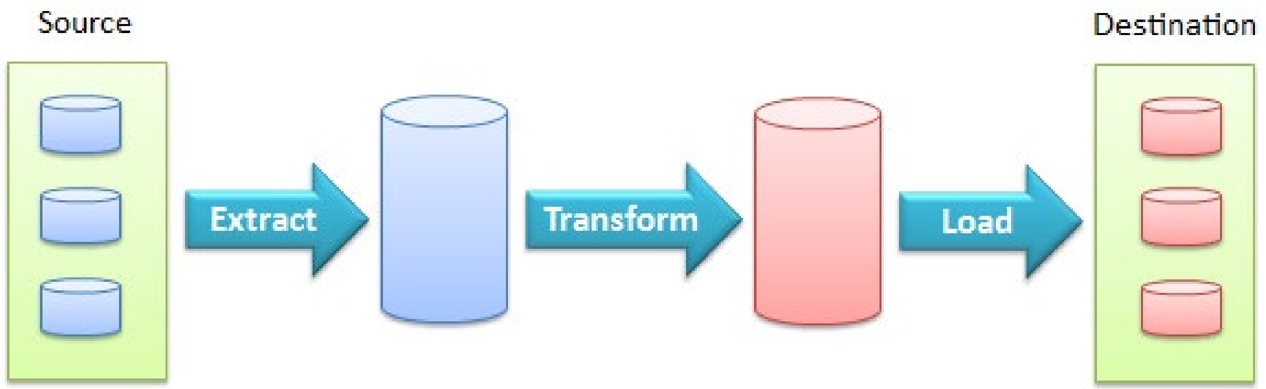
https://www.sqlite.org/lang.html

Where is SQL mostly used?

ETL: extract, transform, and load

Interview question: "Describe the ETL process."

Challenge: Data resides in multiple locations and in many formats.



SQL is very useful for the ETL process

Overview of SQL

Select Statement: Required Clauses

A SELECT statement contains smaller building blocks called clauses.

```
SELECT column1, column2, ...
FROM table_name;
```

- The **SELECT** clause specifies the columns and column order.
- The **FROM** clause specified the data sources

Select Statement: Optional Clauses

```
SELECT column1, column2, ...
FROM table_name
WHERE sql-expression
GROUP BY column_name
HAVING sql-expression
ORDER BY column_name <DESC>;
```

- The **WHERE** clause specifies data that meets certain conditions.
- The **GROUP BY** clause groups data for processing.
- The **HAVING** clause specifies groups that meet certain conditions.
- The **ORDER BY** clause specifies an order for the data.

The specified order of the above clauses within the SELECT statement is required.

Specifying Rows

Subsetting with the WHERE Clause

Use a WHERE clause to specify a condition that the data must satisfy **before being selected**.

```
SELECT Department
FROM employee_information
WHERE salary > 30000;
```

A WHERE clause is evaluated before the SELECT clause.

Summarizing Data

Summary Functions: COUNT Function

The *COUNT function* counts the number of rows returned by a query.

```
select count(*)  as Count
   from employee_information;
```

COUNT(*argument*)

Argument value	Counts
* (asterisk)	All rows in a table or group
A column name	The number of nonmissing values in that column

Commonly Used Summary Functions

ANSI SQL	Description
AVG	Returns the mean (average) value.
COUNT	Returns the number of nonmissing values.
MAX	Returns the largest value.
MIN	Returns the smallest nonmissing value.
SUM	Returns the sum of nonmissing values.

Grouping Data

You can use the GROUP BY clause to do the following:

- classify the data into groups based on the values of one or more columns
- calculate statistics for each unique value of the grouping columns

```
select Employee_Gender as Gender,
       avg(Salary) as Average
from employee_information
group by Employee_Gender;
```

GROUP BY *group-by-item*<,…, *group-by-item*>

Selecting Groups with the HAVING Clause

The ***HAVING clause*** **subsets groups based on the expression value.**

```
select Department, count(*) as Count
   from employee_information
   group by Department
   having Count ge 25
   order by Count desc;
```

GROUP BY *group-by-item* <,…,*group-by-item*>
HAVING *sql-expression*

WHERE Clause versus HAVING Clause

The WHERE clause is evaluated ***before*** a row is available for processing and determines which individual rows are available for grouping.

WHERE *sql-expression*

The HAVING clause is processed ***after*** the GROUP BY clause and determines which groups are displayed.

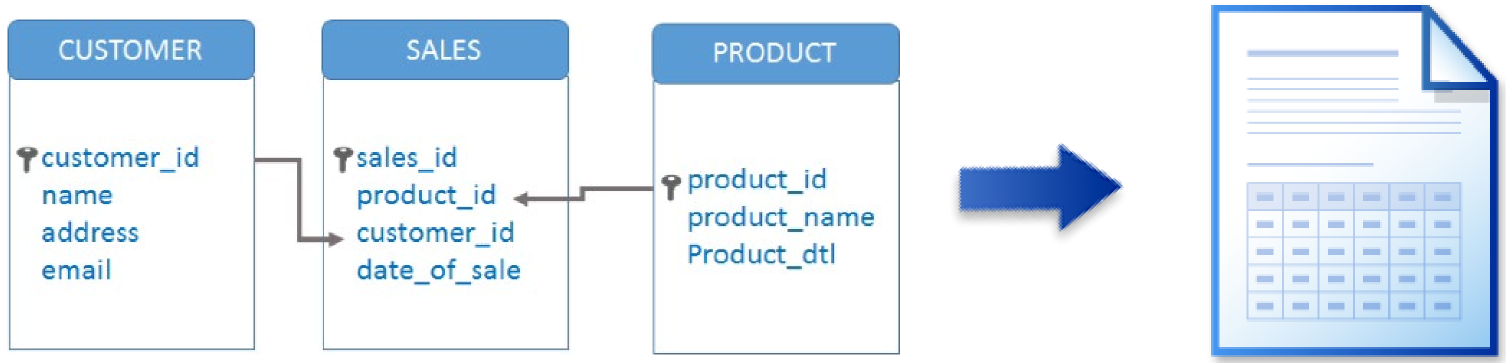
HAVING *sql-expression*

Interview question: "Difference between "where" and "having" in SQL?"

SQL Joins

Combining Tables

SQL uses *joins* to combine tables horizontally. Requesting a join involves matching data from one row in one table with a corresponding row in a second table. Matching is typically performed on one or more columns in the two tables.

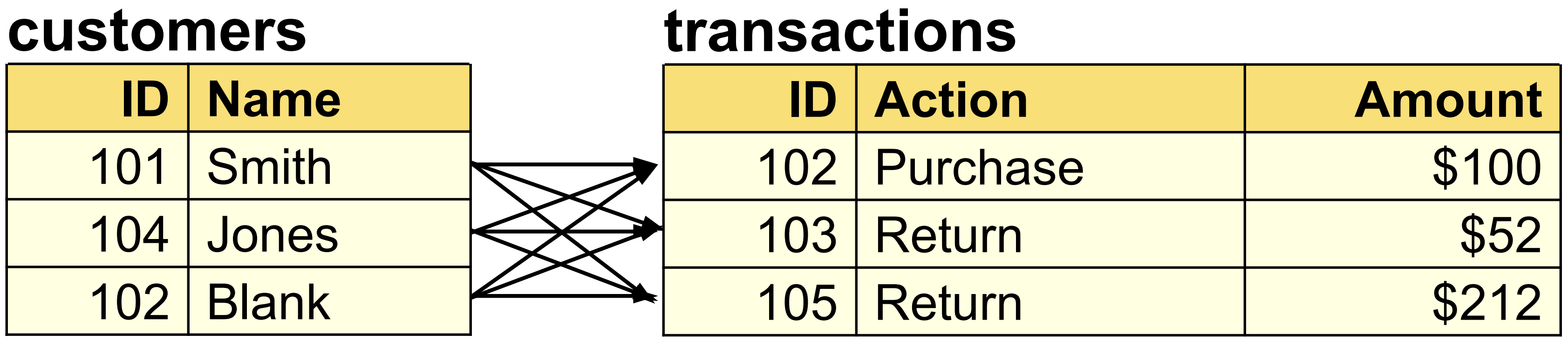


Cartesian Product

A query that lists multiple tables in the FROM clause without a WHERE clause produces all possible combinations of rows from all tables. This result is called a *Cartesian product*.

```
select *
from customers, transactions;
```

Cartesian Product



customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

ID	Action	Amount
102	Purchase	$100
103	Return	$52
105	Return	$212

Result Set

Non-matching IDs

```
 ID   Name      ID   Action          Amount
--------------------------------------------
101   Smith    102   Purchase          $100
101   Smith    103   Return             $52
101   Smith    105   Return            $212
104   Jones    102   Purchase          $100
104   Jones    103   Return             $52
104   Jones    105   Return            $212
102   Blank    102   Purchase          $100
102   Blank    103   Return             $52
102   Blank    105   Return            $212
```

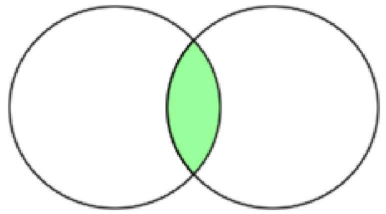
9 rows

The Cartesian Product is rarely what we want to produce…

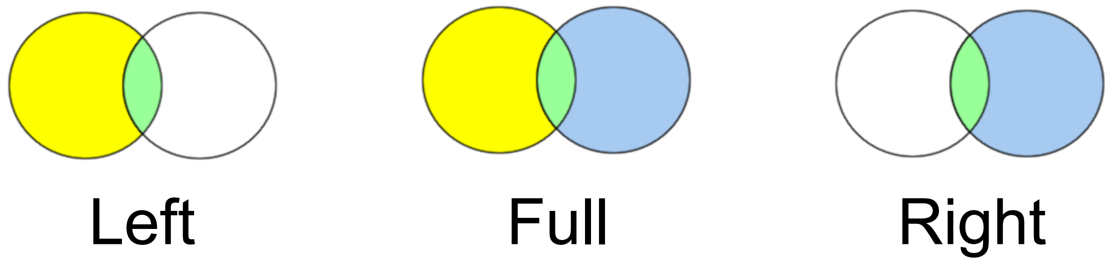
Inner Joins

Types of Joins: two types

- *Inner joins* return only matching rows.



- *Outer joins* return all matching rows, plus nonmatching rows from one or both tables.

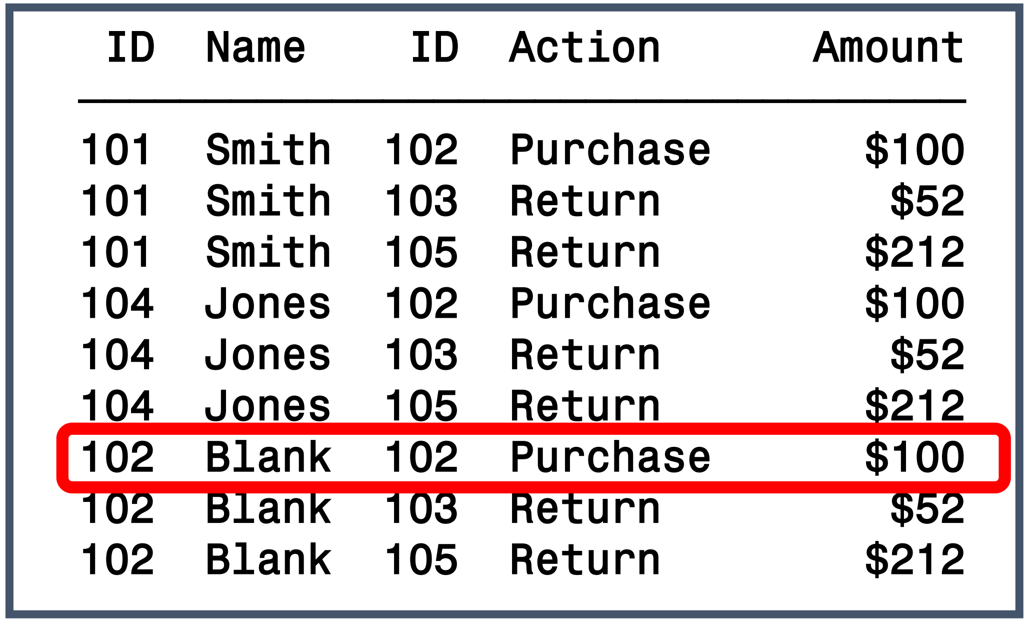


Interview questions regarding joins

- “What is the difference between a SQL Left join and inner join?”
- “What is the difference between an inner and outer join in SQL?”
- “What is an inner join? Outer? Left? Right?”

Inner Join

Generate a report showing all valid order information:



```
 ID  Name     ID  Action        Amount
--------------------------------------
101  Smith   102  Purchase        $100
101  Smith   103  Return           $52
101  Smith   105  Return          $212
104  Jones   102  Purchase        $100
104  Jones   103  Return           $52
104  Jones   105  Return          $212
102  Blank   102  Purchase        $100
102  Blank   103  Return           $52
102  Blank   105  Return          $212
```

Inner Join

The inner join clause links two (or more) tables by a relationship between two columns.

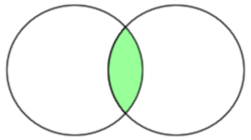
```
select *
from customers, transactions
where customers.ID=transactions.ID;
```

SELECT *object-item<, …object-item>*
    FROM *table-name, … table-name*
    WHERE *join condition*
            <**AND** *sql-expression*>
    <other clauses>;

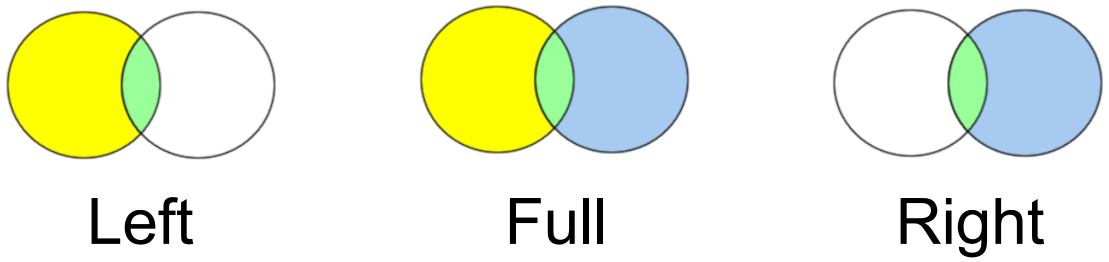
Outer Joins

Outer Joins

- *Inner joins* return only matching rows.



- *Outer joins:* you can retrieve both non-matching and matching rows using an outer join. Many tables can be referenced in outer joins. The tables are processed two tables at a time.



Left Join

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

ID	Action	Amount
102	Purchase	$100
103	Return	$52
105	Return	$212

```
select *
   from customers c left join transactions t
   on c.ID = t.ID;
```

```
  ID   Name      ID   Action         Amount
 -------------------------------------------
 101   Smith      .                       .
 102   Blank    102   Purchase         $100
 104   Jones      .                       .
```

Includes all rows from the left table, even if there are no matching rows in the right table.

Right Join

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

ID	Action	Amount
102	Purchase	$100
103	Return	$52
105	Return	$212

```
select *
   from customers c right join transactions t
   on c.ID = t.ID;
```

```
  ID   Name      ID   Action        Amount
 ------------------------------------------
 102   Blank    102   Purchase        $100
   .            103   Return           $52
   .            105   Return          $212
```

Includes all rows from the right table, even if there are no matching rows in the left table.

Full Join

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

ID	Action	Amount
102	Purchase	$100
103	Return	$52
105	Return	$212

```
select *
   from customers c full join transactions t
   on c.ID = t.ID;
```

```
  ID   Name     ID   Action        Amount
 ----------------------------------------
 101   Smith     .                      .
 102   Blank   102   Purchase        $100
   .           103   Return           $52
 104   Jones     .                      .
   .           105   Return          $212
```

Includes all rows from both tables, even if there are no matching rows in either table

Interview questions regarding joins

- “If you were to join two tables using an outer join in SQL but the two tables don’t share any IDs, what would the resulting table look like?”

Subqueries

(also known as inner queries or nested queries)

What is a Subquery?

A subquery:

- is a query within another SQL query
- returns values to be used:
  - SQLite: You can use a subquery in the SELECT, WHERE, or JOIN clause.
  - Postgres: You can use a subquery in the SELECT or WHERE clause
  - SAS: You can use a subquery in the WHERE or HAVING clause
- must return only a single column
- can return multiple values or a single value.

Example of a Subquery

outer query

subquery

```
SELECT trackid, name, albumid
FROM tracks
WHERE albumid = (SELECT albumid
                 FROM albums
                 WHERE title =
         'Let There Be Rock');
```

Two Types of Subqueries

There are two types of subqueries:

- A *noncorrelated subquery* is a self-contained query. It executes independently of the outer query.
- A *correlated subquery* requires a value or values to be passed to it by the outer (main) query before it can be successfully resolved.

Non-correlated query example

The subquery is resolved before the outer query can be resolved. The following query generates a report that displays **Job_Title** for job groups with an average salary greater than the average salary of the company as a whole.

```
select Job_Title,
       avg(Salary) as MeanSalary
   from staff
   group by Job_Title
   having avg(Salary) >
      (select avg(Salary)
         from staff);
```

Evaluate the subquery first.

Correlated query example

A *correlated subquery* requires a value or values to be passed to it by the outer (main) query before it can be successfully resolved.

```
select Employee_ID, avg(Salary) as MeanSalary
   from employee_addresses
   where 'AU'=
      (select Country
         from supervisors
         where employee_addresses.Employee_ID=
               supervisors.Employee_ID)
group by 1;
```

This query is not stand-alone.
It needs additional information from the main query.

Returning multiple rows from the subquery

A subquery can return multiple values or a single value.

However, subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.

```
select Employee_Name, City, Country
from employee_addresses
where Employee_ID IN
      (select Employee_ID
          from employee_payroll
          where Birth_Month=2)
order by 1;
```

The NOT IN operator displays a record if the condition(s) is NOT TRUE.

In-Line Views

(also known as subquery)

What is an In-Line View?

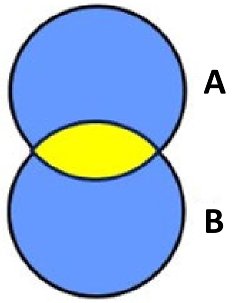
An *in-line view* is a query expression (SELECT statement) that resides in a FROM clause:

- It acts as a virtual table, used in place of a physical table in a query.
- An in-line view can return more than just one column

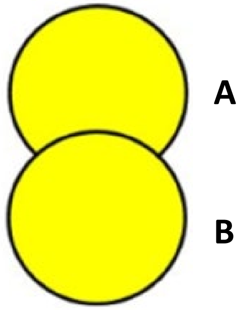
```
SELECT sub.*
FROM (SELECT date, location, resolution
         FROM tutorial.sf_crime_incidents
         WHERE day_of_week = 'Friday')as
         sub
WHERE sub.resolution = 'NONE'
```

Set Operators

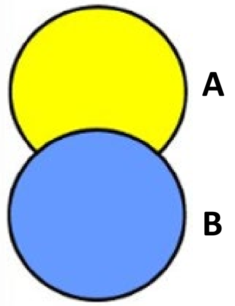
Set Operators



INTERSECT

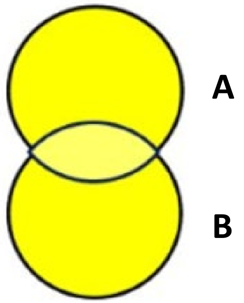


UNION



EXCEPT

MINUS (Oracle)



UNION ALL (SQLite, Postgres)

OUTER UNION (SAS)

The UNION clause removes duplicate rows that exist, while the UNION ALL (or OUTER JOIN) clause does not.

Interview question: "What is the difference between a SQL union and union all?"

Using Set Operators

```
select …
UNION | UNION ALL | EXCEPT | INTERSECT
select … ;
```

Operator	Returns
UNION	All distinct rows selected by either query
UNION ALL	All rows selected by either query, including all duplicates
INTERSECT	All distinct rows selected by both queries
EXCEPT	All distinct rows selected by the first query but not the second

Scenario: Two tables

Partial **train_a**

ID	Name	End_Date
11	Bob	15JUN2012
16	Sam	5JUN2012
14	Pete	21JUN2012

Training class A is completed in a single session. End_Date represents the date of training.

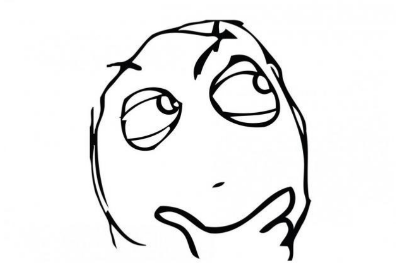
Partial **train_b**

Name	ID	SDate	EDate
Bob	11	9JUL2012	13JUL2012
Pam	15	25JUL2012	27JUL2012
Kyle	19	12JUL2012	20JUL2012
Chris	21	29JUL2012	.

Training class B is a multi-session class. SDate is recorded on the first training day. EDate is recorded when the course is complete.

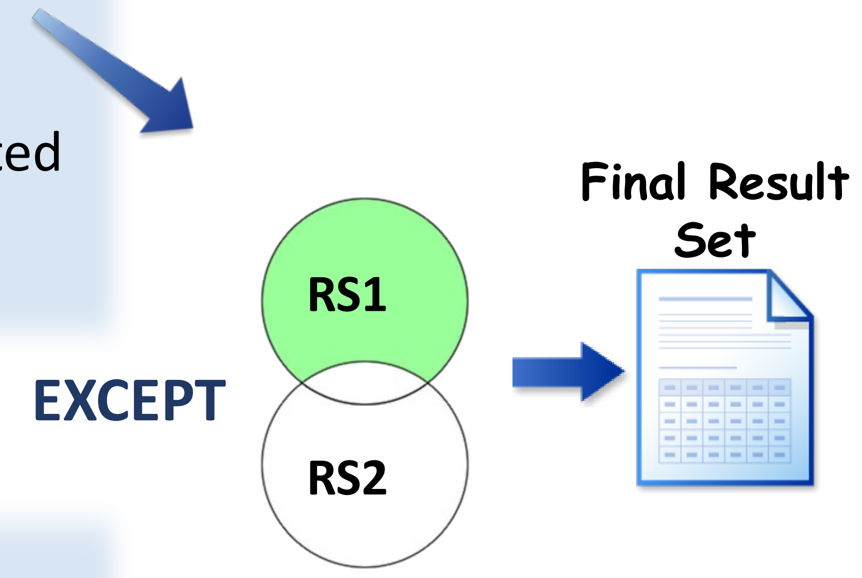
EXCEPT Operator

Which employees have completed training A, but not training B?



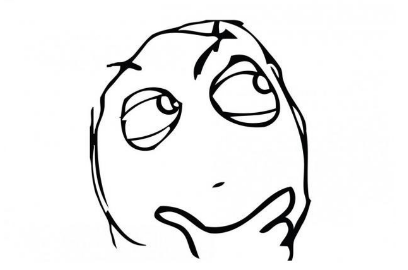
Query 1:
List employees that have completed **train_a**.

Query 2:
List employees that have completed **train_b**.



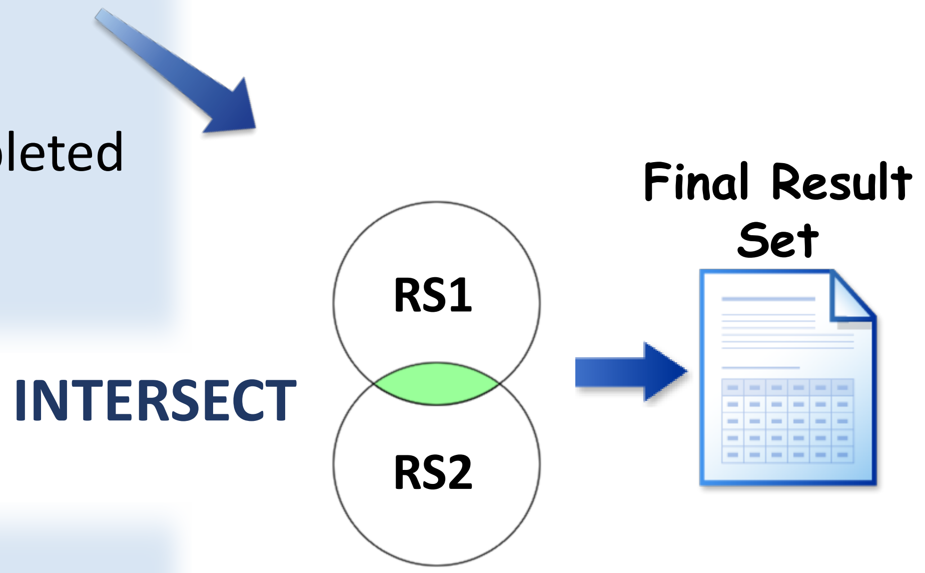
INTERSECT Operator

Which employees have completed both classes?



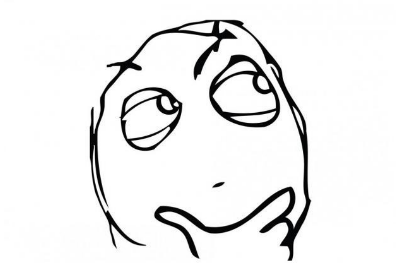
Query 1:
List employees
that have completed
train_a.

Query 2:
List employees
that have completed
train_b.



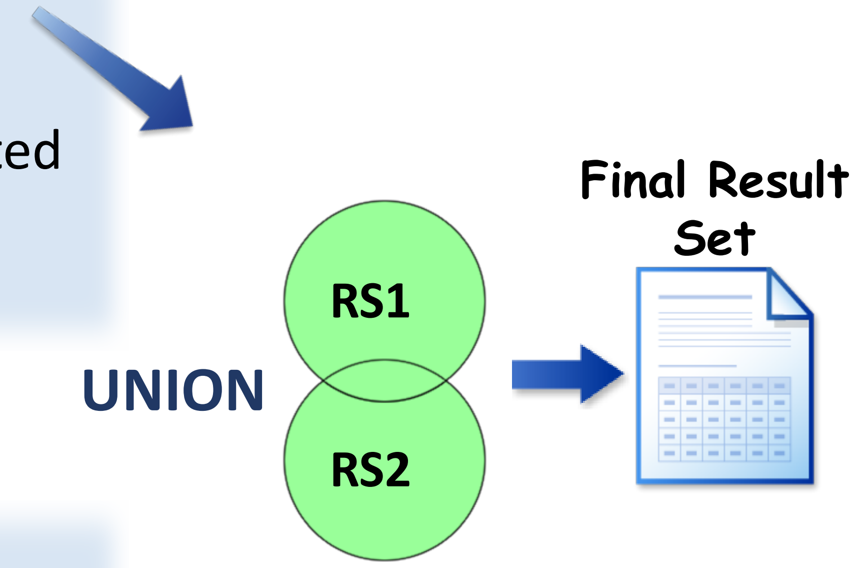
UNION Operator

Which employees have completed training A or B?



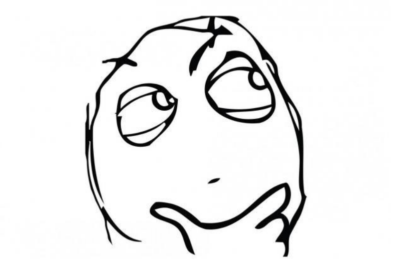
Query 1:
List employees
that have completed
train_a.

Query 2:
List employees
that have completed
train_b.

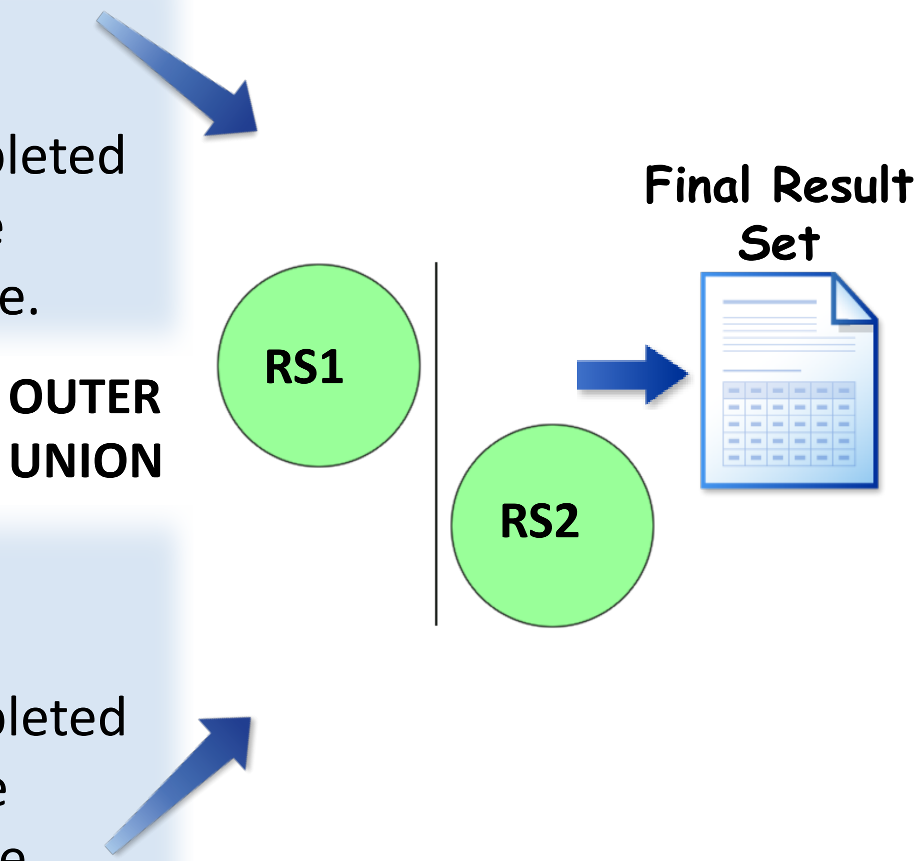


UNION ALL (or OUTER JOIN) Operator

Which employees have completed training A and/or B and on what dates?



Query 1:
List employees that have completed **train_a** and the completion date.



Query 2:
List employees that have completed **train_b** and the completion date.