RDBMS Concepts

Team Emertxe

What is RDBMS?

Definition

- RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

- A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

- The data in RDBMS is stored in database objects called tables.

Different RDBMS Technology

DBMS & RDBMS

- DBMS applications store data as file.
- DBMS does not support client/server architecture.
- DBMS does not allow normalization.
- DBMS does not impose integrity constraints.

- RDBMS applications store data in a tabular form.
- RDBMS supports client/server architecture.
- RDBMS allows normalization.
- RDBMS imposes integrity constraints.

Creating Table

Primary Keys

StudentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042
L0023487	Peter	Murray	P301
L0018453	Anne	Norris	S042

Creating Table

- The data in RDBMS is stored in database objects called tables. The table is a collection of related data entries and it consists of columns and rows.
- Remember, a table is the most common and simplest form of data storage in a relational database.

Field, Record or Row

- Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.
- A record, also called a row of data, is each individual entry that exists in a table. For example there are 7 records in the above CUSTOMERS table.

Column

- A column is a vertical entity in a table that contains all information associated with a specific field in a table.

Null Value

- A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

SQL

SQL

- SQL is a standard language for accessing databases.
- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL is an ANSI (American National Standards Institute) standard

Benefit of SQL

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database

Using SQL in your Project

To build an application that shows data from a database, you will need:

- An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
- To use a server-side scripting language, like Java, PHP or ASP
- To use SQL to get the data you want
- To use HTML / CSS

SQL Data Types

- Number Data types :-

  - INTEGER :- The Integer data type is used to specify an integer value.

  - SMALLINT :- The SMALLINT data type is used to specify small integer value.

  - NUMERIC(P,S) :-It specifies a numeric value. Here 'p' is precision value and 's' is a scale value.

SQL Data Types

- Float Types :

  - FLOAT(P) - It specifies floating-point value. Here 'p' is precision value.

  - DOUBLE PRECISION - It specifies double precision floating point number.

  - REAL - The real integer is used to specify a single precision floating point number.

  - DECIMAL(P,S) - It specifies a decimal value. Here 'p' is precision value and 's' is scale value.

SQL Data Types

- String Types :-

  - CHAR(X) - Here 'x' is the character's number to store. Here char should be used for storing fix length strings.

  - VARCHAR2(X) - Here 'x' is the character's number to store. varchar is used to store variable length strings. The String value's length will be stored on disk with the value itself.

SQL Data Types

- Date & Time Type

  - DATE :- It stores year , month and days values.

  - TIME : -It stores hour,minute and second values.

  - TIMESTAMP :- The timestamp data type is used to year , month ,day ,hour ,minute and second values.

MYSQL
Data Types

- In MySQL there are three main types : text, number, and Date/Time types.

MySQL Datatypes

- Text Types:
  - CHAR(size):(can contain letters, numbers, and special characters).Can store up to 255 characters
  - VARCHAR(size):can contain letters, numbers, and special characters).Can store up to 255 characters.
  - TEXT: Holds a string with a maximum length of 65,535 characters
  - BLOB: For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data

MySQL Data Types

- Number types:
  - INT(size)
  - FLOAT(size,d)
  - DOUBLE(size,d)

MySQL Data Types

- Date types:
  - DATE(): A date. Format: YYYY-MM-DD
  - DATETIME(): A date and time combination. Format: YYYY-MM-DD HH:MM:SS
  - TIME(): A time. Format: HH:MM:SS
  - YEAR(): A year in two-digit or four-digit format.

EMERTXE

Important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database

Important SQL Commands

- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table

SQL Syntax

- CREATE DATABASE
  - CREATE DATABASE dbname;
- CREATE TABLE
  - CREATE TABLE Persons(PersonID int,LastName varchar(255),FirstName varchar(255),Address varchar(255),City varchar(255)

    );

SQL Syntax

- SQL SELECT Statement
  - SELECT * FROM table_name;
- SELECT Column Example
  - SELECT CustomerName,City FROM Customers;
- SELECT DISTINCT Statement
  - SELECT DISTINCT column_name,column_name FROM table_name;

SQL Syntax

- SQL WHERE Clause
  - SELECT * FROM Customers
    WHERE Country='Mexico';;

SQL Syntax

- Operators in the where clause:

  =, >,<,>=,<=, BETWEEN, LIKE, IN
- SQL AND Operators
  - SELECT * FROM Customers

    WHERE Country='Germany'

    AND City='Berlin';
- OR Operator Example
  - SELECT * FROM Customers
  - WHERE City='Berlin'
  - OR City='Munchen';

EMERTXE

SQL Syntax

- Combining AND & OR
  - SELECT * FROM Customers
    WHERE Country='Germany'
    AND (City='Berlin' OR City='München');

ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set by one or more columns.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.

ORDER BY Keyword

- SELECT column_name,column_name

  FROM table_name

  ORDER BY column_name,column_name ASC|DESC;
- SELECT * FROM Customers ORDER BY Country;
- SELECT * FROM Customers ORDER BY Country DESC;
- SELECT * FROM Customers ORDER BY Country,CustomerName;

EMERTXE

INSERT INTO Statement

- INSERT INTO table_name

  VALUES (value1,value2,value3,...);

- INSERT INTO table_name (column1,column2,column3,...)

  VALUES (value1,value2,value3,...);

- INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)

  VALUES ('Cardinal','Tom B. Erichsen','Skagen 21','Stavanger','4006','Norway');

Update Statement

The UPDATE statement is used to update existing records in a table.

- UPDATE table_name

  SET column1=value1,column2=value2,...

  WHERE some_column=some_value;

- ***UPDATE Customers SET ContactName='Alfred Schmidt', City='Hamburg' WHERE CustomerName='Alfreds Futterkiste';***

EMERTXE

DELETE Statement

- The DELETE statement is used to delete records in a table.
- DELETE FROM table_name WHERE some_column=some_value;
- DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste' AND ContactName='Maria Anders';

DELETE Statement

- Delete All Data
- DELETE FROM table_name;

  or
- DELETE * FROM table_name;

SQL Injection

An SQL Injection can destroy the database. SQL injection is a technique where malicious users can inject SQL commands into an SQL statement ,via web page input.

Injected SQL commands can alter SQL statements and compromise the security of a web application.

SQL Web Page

When SQL is used to display data on a web page,it is common to let web user input their own search value.

Since SQL statements are text only,it is easy,with a little piece of computer code, to dynamically change SQL statements to provide the user with selected data.

SQL WEB Page

```
txtUserId =getRequestString(“UserId”);

txtSQL =”SELECT * FROM Users Where
UserId=”+txtUserId
```

- The example creates a select statement by adding a variable (txtUserId) to a select string. The variable is fetched from the user input (Request) to the page.

SQL Injection

SELECT * FROM Users Where UserId =105 or 1=1

- SQL above is valid. It will return all rows from the table Users ,since Where 1=1 is always true.

SELECT TOP

- SELECT TOP clause is used to specify the number of records to return.
- SELECT TOP clause can be very useful on large tables with thousands of records. Returning a large number of records can impact on performance.

SELECT TOP

- MySQL Syntax :

  SELECT column_name(s) from table_name LIMIT number;

- Example

  SELECT * FROM Students LIMIT 5;

SELECT TOP

- Oracle Syntax :

  SELECT column_name(s) FROM table_name WHERE ROWNUM <= number;

- Example

  SELECT * FROM Students WHERE ROWNUM <=5;

SELECT TOP

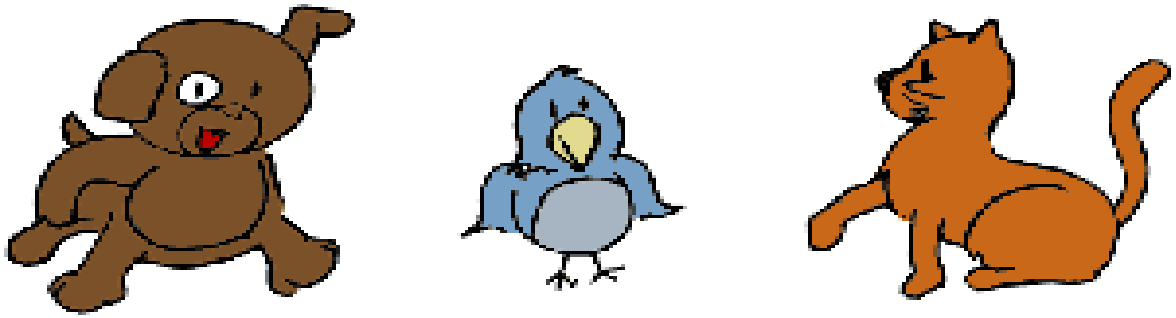
- Oracle Syntax :

  SELECT column_name(s) FROM table_name WHERE ROWNUM <= number;

- Example

  SELECT * FROM Students WHERE ROWNUM <=5;

Between Operators



BETWEEN Operator

- SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;

  SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;

- NOT BETWEEN

  SELECT * FROM Products WHERE Price NOT BETWEEN 10 AND 20;

- BETWEEN Operator with IN

  SELECT * FROM Products WHERE (Price BETWEEN 10 AND 20) AND NOT CategoryID IN (1,2,3);

EMERTXE

BETWEEN Operator

- BETWEEN Operator with Text Value
  - SELECT * FROM Products WHERE ProductName BETWEEN 'C' AND 'M';
- NOT BETWEEN Operator with Text Value
  - SELECT * FROM Products WHERE ProductName NOT BETWEEN 'C' AND 'M';
- BETWEEN Operator with Date Value
  - SELECT * FROM Orders WHERE OrderDate BETWEEN #07/04/1996# AND #07/09/1996#;

LIKE Operator

The Like operator is used to search for a specified pattern in a column .

SQL LIKE Syntax :

SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern;

LIKE Operator

- Select all Customers starting with a City starting with the letter “m”

  SELECT * FROM Customers WHERE City LIKE 'm%';

- Select all Customers with a City ending with the letter “m”

  SELECT * FROM Customers WHERE City LIKE '%m';

- Select all customers with a City not containing the pattern “nadu”

  SELECT * FROM Customers WHERE City NOT LIKE '%nadu%';

Wildcards

A wildcard character can be used to substitute for any other character(s) in a string.

Two types of Wildcard characters :

- Percent sign(%) : Matches one or more Characters
- Underscore (_) : Matches one Character

Wildcards

- Find any value start with 50

  SELECT * FROM Product where price LIKE '50%';

- Find any value that have 50 in any position.

  SELECT * FROM Product where price LIKE '%50%';

- Find any value that have 00 in the second and third positions

  SELECT *FROM Product where price LIKE '_00%';

- Finds any values in a five-digit number that start with 2 and end with 3

  SELECT * FROM Product where price LIKE '2___3';

IN Operator

In operator allows to specify multiple values in a Where clause.

- IN Operator Syntax :

  SELECT column_name(s) FROM table_name WHERE column_name IN (value1,value2,....);

IN Operator

- Select all Customer with a City of “Manchester” and “London”

  SELECT * FROM Customers WHERE City IN ('Manchester','London');

- Select all Customers Except City of “Manchester” and “London”

  SELECT * FROM Customers WHERE City NOT IN ('Manchester','London');

SQL Functions

SQL has many built-in functions for performing calculations on data.

Aggregate Functions

SQL aggregate functions return a single value , calculated from values in a column.

Aggregate Functions

- SUM() - Returns the sum
- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- MAX() - Returns the Largest value
- MIN() - Returns the Smallest value
- FIRST() - Returns the first value
- LAST() - Returns the last value

SUM()

- Syntax :

  SELECT SUM(expression) FROM tables WHERE conditions;

- Example :

  SELECT SUM(salary) AS “Total Salary” FROM Employee WHERE salary >20000;

Group By

- The GROUP BY clause is a SQL command that is used to group rows that have the same values.
- It is used in conjunction with aggregate functions to produce summary reports from the database.
- GROUP BY clause are called grouped queries and only return a single row for every grouped item.

Group By

- Syntax :

  SELECT column_name , aggregate_function(column_name) FROM table_name WHERE column_name operator value

  GROUP BY column_name;

- Example :

  SELECT Department ,SUM(Salary) AS “Total Salary “ FROM Employee

  GROUP BY Department ;

SQL Scalar Functions

SQL Scalar function return a single value, based on the input value.

SQL Scalar Functions

- UCASE() - Converts a field to upper case.
- LCASE() - Converts a field to lower case.
- LEN() - Returns the length of a text field.
- MID() - Extract characters from a text field.
- ROUND() - Rounds a numeric field to the number of decimal specified.
- NOW() - Returns the current date and time.
- FORMAT() - Formats how a field is to be displayed.

UCASE()

The UCASE() function converts the value of a field to uppercase.

- Syntax

  SELECT UCASE(column_name) FROM table_name;

- Syntax for SQL Server

  SELECT UPPER(column_name) FROM table_name;

LEN()

The LEN() function returns the length of the value in a text field.

- Syntax

  SELECT LEN(column_name) FROM table_name;

- Syntax for Oracle

  SELECT LENGTH(column_name) FROM table_name;

MID()

The MID() function is used to extract characters from a text field.

- Syntax

  SELECT MID(column_name,start,length) AS some_name FROM table_name;

- column_name - The field to extract characters
- Start - Specifies the starting position
- Length - The number of characters to return.

ROUND()

The ROUND() function is used to round a numeric field to the number of decimal specified.

- Syntax :

  SELECT ROUND(column_name ,decimals) FROM table_name;

- column_name - The field to round.
- Decimals - Specifies the number of decimals to be returned.

NOW()

The Now() function returns the current date and time.

- Syntax :

  SELECT NOW() FROM table_name;

- Example :

  SELECT Product_Name ,Price ,NOW() AS “ Per Date” from Products;

FORMAT()

The FORMAT() function is used to format how a field is to be displayed.

- Syntax :

  SELECT FORMAT(column_name,format) FROM table_name;

- Example :

  SELECT Product_Name ,Price, FORMAT(NOW(),'YYYY-MM-DD') AS 'Per Date' FROM Products;

SQL Aliases

SQL Aliases are used to temporarily rename a table or column heading.

Syntax

- **SELECT column_name AS alias_name FROM table_name;**

Table Alias

Syntax :

SELECT column_name AS alias_name
FROM table_name;

Example

SELECT C.ID ,C.NAME ,C.AGE ,O.AMOUNT
FROM Customer As C , Order As O WHERE
C.ID = O.Customer _ID;

Column Alias

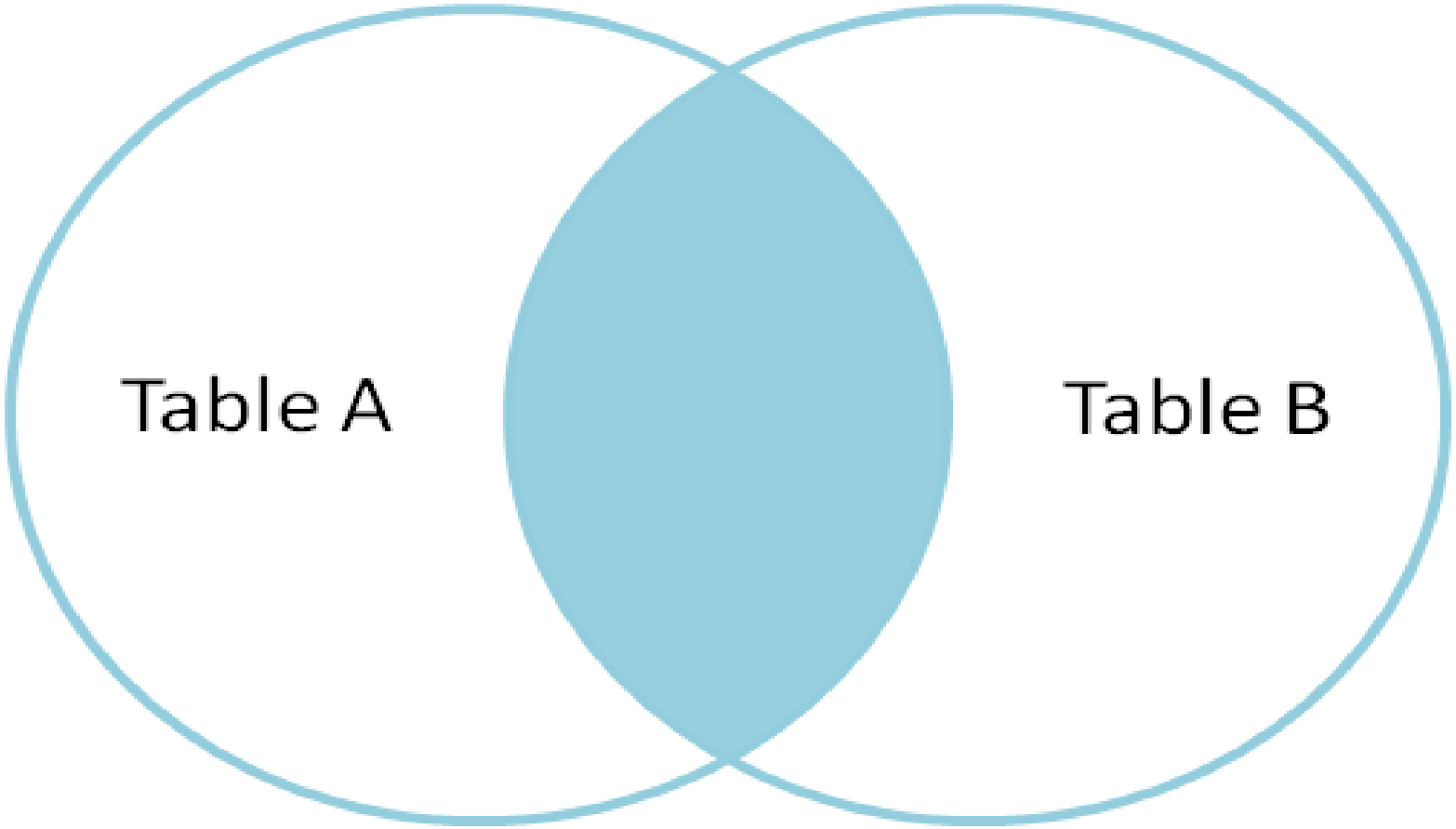
- Syntax :

  SELECT column_name(s) FROM table_name AS alias_name;

- Example

  SELECT ID AS CUSTOMER_ID , NAME AS CUSTOMER_NAME FROM CUSTOMERS WHERE SALARY IS NOT NULL;

SQL Joins



SQL Joins

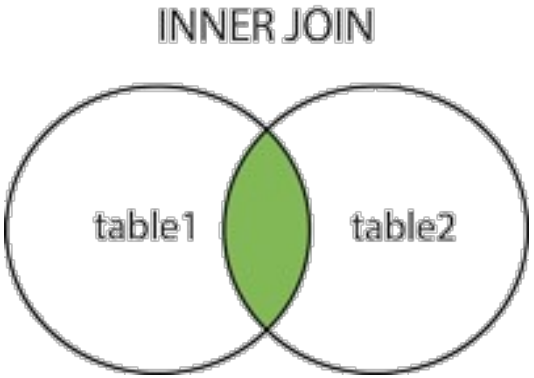
- An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

SQL Joins

- The different SQL JOINs you can use:
  - INNER JOIN: Returns all rows when there is at least one match in BOTH tables
  - LEFT JOIN: Return all rows from the left table, and the matched rows from the right table. The result is null in the right side if there is no match.
  - RIGHT JOIN: Return all rows from the right table, and the matched rows from the left table. The result is null in the right side if there is no match.
  - FULL JOIN: Return all rows from the left table and right table. It combines the result of both LEFT and RIGHT join.

SQL Joins

- The most common type of join is: SQL INNER JOIN (simple join). An SQL INNER JOIN return all rows from multiple tables where the join condition is met.



SQL Joins

Let's look at a selection from the "Orders" table:

- OrderID CustomerID OrderDate

  10308 2 1996-09-18

  10309 37 1996-09-19

  10310 77 1996-09-20

SQL Joins

Have a look at a selection from the "Customers" table:

- CustomerID CustomerName ContactName Country

  1 **Alfreds Futterkiste** Maria Anders **Germany**

  2 **Ana Trujillo** Trujillo **Mexico**

  3 **Antonio Moreno** Moreno **Mexico**

- *Notice that the "CustomerID" column in the "Orders" table refers to the customer in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.*

SQL Joins

Example:

- SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

  FROM Orders

  INNER JOIN Customers

  ON Orders.CustomerID=Customers.CustomerID;

- It will produce the following:

  OrderID CustomerName OrderDate

  10308 Ana Trujillo 9/18/1996

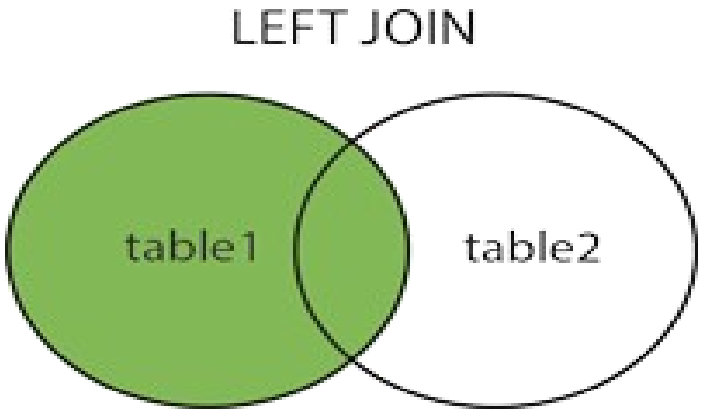
EMERTXE

LEFT JOIN

SYNTAX :

SELECT column_name(s) FROM table1 LEFT JOIN table2

ON table1.column_name = table2.column_name ;

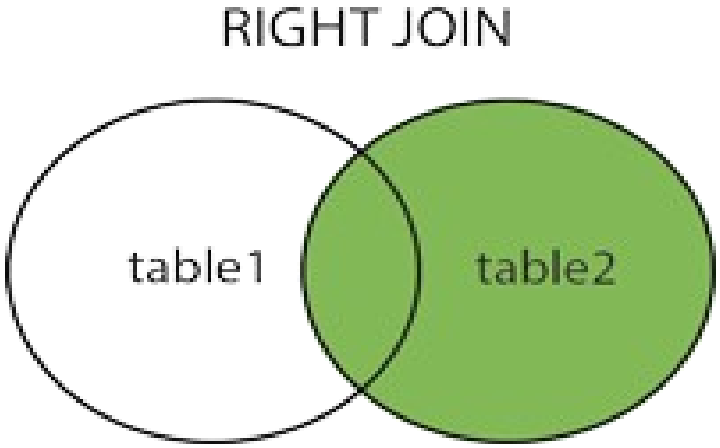


RIGHT JOIN

SYNTAX :

SELECT column_name(s) FROM table1 RIGHT JOIN table2

ON table1.column_name = table2.column_name ;

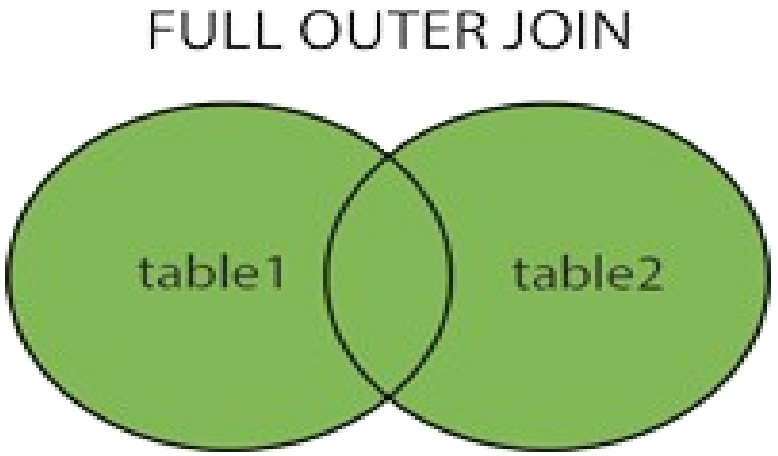


FULL JOIN

SYNTAX :

SELECT column_name(s) FROM table1 FULL JOIN table2

ON table1.column_name = table2.column_name ;



EMERTXE

Union Operator

The SQL Union operator combines the result of two or more select statement. Union Operator Select only distinct values by default . To allow duplicate values use ALL keyword with UNION .

Syntax :

```
SELECT Column_name(s) FROM table1

UNION

SELECT Column_name(s) FROM table2;
```

Select Into

The SELECT INTO statement copies data from one table and insert it into a new Table.

Syntax :

- SELECT * INTO new table [IN External db]
  FROM table1

OR

- SELECT column_name (s) INTO new table [IN external db]
  FROM table1;

Insert into Select

The INSERT INTO SELECT statement copies data from one table and insert it into existing table.

Syntax

- Copy data from one table to another existing table.

  INSERT INTO table2

  SELECT * FROM table1;

- Copy only columns into another existing table

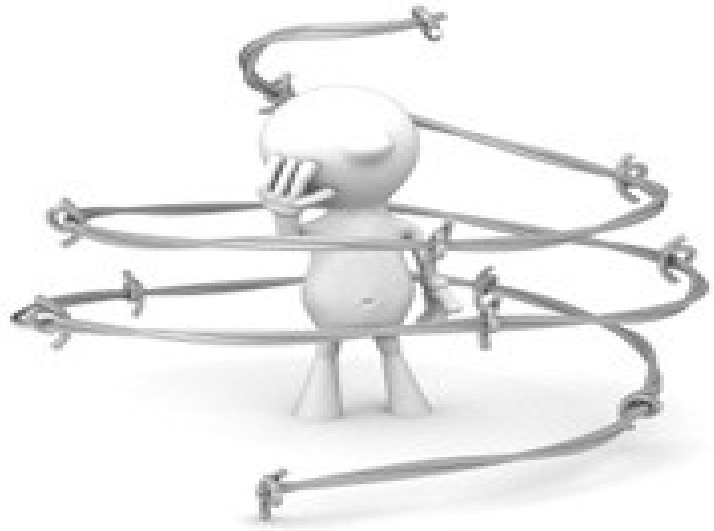
  INSERT INTO table2

  (column_name (s))

  SELECT column_name(s) FROM table1;

SQL Constraints

- SQL constraints are used to specify rules for the data in a table.
- Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).



SQL Constraints

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

NOT NULL Constraint

NOT NULL Constraint enforces a column to not accept a null values.

- Example

```
CREATE TABLE Student
(Stud_ID   int  (10)  NOT NULL,
 Name      varchar (20)  NOT NULL,
 Address  varchar(20)
 Course     varchar  (20));
```

Unique Constraint

The UNIQUE Constraint uniquely identifies each record in a database table.

Example

```
CREATE TABLE  Student
(Stud_id            int NOT NULL UNIQUE,
 Name        varchar (255)  NOT NULL,
 Address     varchar (255),
 City        varchar (255) );
```

PRIMARY KEY Constraint

- PRIMARY KEY Constraint uniquely identifies each record in a database table.
- PRIMARY KEY must contain UNIQUE values.
- PRIMARY KEY column cannot contain NULL values.
- Each table can have only one PRIMARY KEY.

Primary Key Constraint

```
CREATE  TABLE  Student
(Stud_Id   int   PRIMARY  KEY,
 Name     varchar (20)  NOT NULL,
 Address  varchar (20) ,
 City       varchar(35));
```

Primary Key Constraint

- Adding Primary key Already created Table

  ALTER TABLE STUDENT

  ADD PRIMARY_KEY (STUD_ID) ;

- Defining Primary Key Constraints on Multiple Columns

  ALTER TABLE STUDENT

  ADD CONSTRAINT S_ID PRIMARY_KEY(STUD_ID, FIRSTNAME) ;

Primary Key Constraint

- To DROP a Primary KEY

ALTER TABLE STUDENT

DROP PRIMARY KEY

Foreign Key Constraint

- A Foreign Key in one table points to a PRIMARY KEY in another table.

- A Foreign key is a field or a column that is used to establish a link between two tables.

- The Foreign Key constraint also prevents invalid data from being inserted into foreign key column ,because it has to be one of the values contained in the table it points to.

Foreign Key Constraint

- CREATE TABLE Student_Detail
  (Stud_Id int PRIMARY KEY,
  Name varchar (20) NOT NULL,
  Address varchar (20) ,
  City varchar(35));
- CREATE TABLE Marks_Detail
  (Stud_Id int PRIMARY KEY,
  Name varchar (20) NOT NULL,
  Address varchar (20) ,
  City varchar(35)
  FOREIGN KEY (Stud_Id) REFERENCES STUDENT_DETAILS(Stud_Id)
  );

Check Constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.

```
CREATE TABLE Student
( Stud_Id int PRIMARY KEY,
   Name   varchar(20) NOT NULL,
   Address  varchar(20) NOT NULL,
   City       varchar(20)  NOT NULL CHECK (City
='Banglore'));
```

Default Constraint

Default constraint is used to insert a default value into a column. Default value will be added to all new records ,if no other value is specified.

```
CREATE TABLE Student
( Stud_Id int PRIMARY KEY,
   Name     varchar(20) NOT NULL,
   Address  varchar(20) NOT NULL,
   City       varchar(20)  DEFAULT 'Banglore');
```

Indexes

- Indexes allows the database application to find data fast , without reading the whole table.
- An Index can be created in a table to find data more quickly and efficiently
- The users cannot see the indexes, they are just used to speed up searches/queries.

Indexes

- Create Index Syntax

  CREATE INDEX index_name

  ON table_name (column_name)

- Create Unique Index

  CREATE UNIQUE INDEX index_name

  ON table_name (column_name)

EMERTXE

Index Example

- Creating Index on a single column

  CREATE INDEX ID1 ON Student(Name);

- Creating Index on a multiple Column

  CREATE INDEX ID2

  ON Student (FirstName , LastName);

SQL Drop

- DROP TABLE
  - The DROP TABLE statement is used to delete a table.
  - DROP TABLE table_name
- DROP DATABASE
  - The DROP DATABASE statement is used to delete a database.
  - DROP DATABASE database_name
- ALTER TABLE
  - The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
  - ALTER TABLE table_name ADD column_name datatype

SQL Drop

- TRUNCATE TABLE
  - What if we only want to delete the data inside the table, and not the table itself?
  - TRUNCATE TABLE table_name

SQL AUTO INCREMENT

- Very often we would like the value of the primary key field to be created automatically every time a new record is inserted.
- We would like to create an auto-increment field in a table.
- ***CREATE TABLE Persons(ID int NOT NULL AUTO_INCREMENT,LastName varchar(255) NOT NULL,FirstName varchar(255),Address varchar(255),***

  City varchar(255),PRIMARY KEY (ID))

SQL AUTO INCREMENT

- By default, the starting value for AUTO_INCREMENT is 1,and it will increment by 1 for each new record.
- To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:
  - ALTER TABLE Persons AUTO_INCREMENT=100
- To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):
  - INSERT INTO Persons (FirstName,LastName) VALUES ('Lars','Monsen')

Views

- View is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns , just like a real table.
- The fields in a view are fields from one or more real tables in the database.

Views

- Updating View

  CREATE OR REPLACE VIEW view_name AS SELECT column_name(s) FROM table_name WHERE condition;

- Dropping a View

  DROP VIEW view_name;

View

View Syntax :

CREATE VIEW view_name AS SELECT column_name(s) FROM table_name WHERE condition

Example :

CREATE VIEW StudentList AS SELECT Stud_Id ,Name FROM Students WHERE City ='Banglore' ;

We can query the view as follows :

SELECT * FROM StudentList ;

Views

- Updating View

  CREATE OR REPLACE VIEW view_name AS SELECT column_name(s) FROM table_name WHERE condition;

- Dropping a View

  DROP VIEW view_name;

SQL DATES

Assume we have following “Orders” table

- OrderID CustomerID OrderDate

  10308 2 2011-09-18

  10309 37 2009-09-19

  10310 77 2012-09-20

  10311 78 2012-09-21

SELECT *FROM Orders WHERE OrderDate='2011-09-18'

SQL DATE Function

- GETDATE() returns the current date & Time from SQL Server.

  Example

  SELECT GETDATE() AS CurrentDateTime

- DATEDIFF() function returns the time between two functions.

  DATEDIFF(datepart, startdate, enddate)

  Example

  SELECT DATEDIFF(day,'2016-08-05' ,'2016-08-06') AS DiffDate

SQL NULL Values

- NULL values represent missing unknown data.
- By default ,a table can hold NULL values.
- NULL values are treated differently from other values.

  Example

  SELECT OrderId ,CustomerId from Orders where OrderId IS NULL

NULL Functions

- ISNULL() :- function is used to specify how we want to treat NULL values.
- NVL(), IFNULL(), and COALESCE() functions can also be used to achieve the same result.

Subquery

- Subquery or Inner query or Nested query is a query in query is usually added in the WHERE clause of the SQL Statement.
- Subqueries are an alternate way of returning data from multiple tables.
- Subqueries can be used with the SELECT ,INSERT ,UPDATE and DELETE statements along with the operators like = ,<,>,>=,<=,IN ,BETWEEN etc.

Subquery

Syntax :

SELECT select_list FROM table WHERE expr operator (SELECT select_list FROM table);

- The subquery (inner query) executes once before the main query (outer query) executes.
- The main query(outer query) use the subquery result.

Subquery Example

- We have the following two tables student and marks.
- Student Table

StudentID	Name	City
A001	John	Paris
A002	Smith	London
A003	Mac	Berlin
A004	Ivan	London

Subquery Example

- Marks Table

StudentID	Total_Marks
A001	95
A002	80
A003	75
A004	81

Subquery Example

- We want to write a query to identify all students who get better marks than that of the students who's StudentID is A002,but we don't know the marks of A002.

  Here two query one query return the the marks stored in Total_marks field another query identifies the students who get better marks than the result of the first query.

Subquery Example

SELECT a.StudentID , a.name , b.Total_marks FROM Student a , Marks b WHERE a.StudentID =b.StudentID AND b.Total_marks > (SELECT Total_Marks FROM Marks WHERE StudentId='A002');

- Output :

StudentID	Name	Total_Marks
A001	John	95
A004	Ivan	81

Types of Subqueries

- Single Row Subquery
- Multiple Row Subquery
- Correlated Subquery

Single Row Subquery

- Subquery which returns single row output. They mark the usage of single row comparison operators when used in where condition.

Agent Table

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO	COUNTRY
A007	Ramasundar	Bangalore	0.15	077-25814763	
A003	Alex	London	0.13	075-12458969	
A008	Alford	New York	0.12	044-25874365	
A011	Ravi Kumar	Bangalore	0.15	077-45625874	
A010	Santakumar	Chennai	0.14	007-22388644	
A012	Lucida	San Jose	0.12	044-52981425	
A005	Anderson	Brisban	0.13	045-21447739	
A001	Subbarao	Bangalore	0.14	077-12346674	
A002	Mukesh	Mumbai	0.11	029-12358964	
A006	McDen	London	0.15	078-22255588	
A004	Ivan	Torento	0.15	008-22544166	
A009	Benjamin	Hampshair	0.11	008-22536178	

Single Row Subquery Example

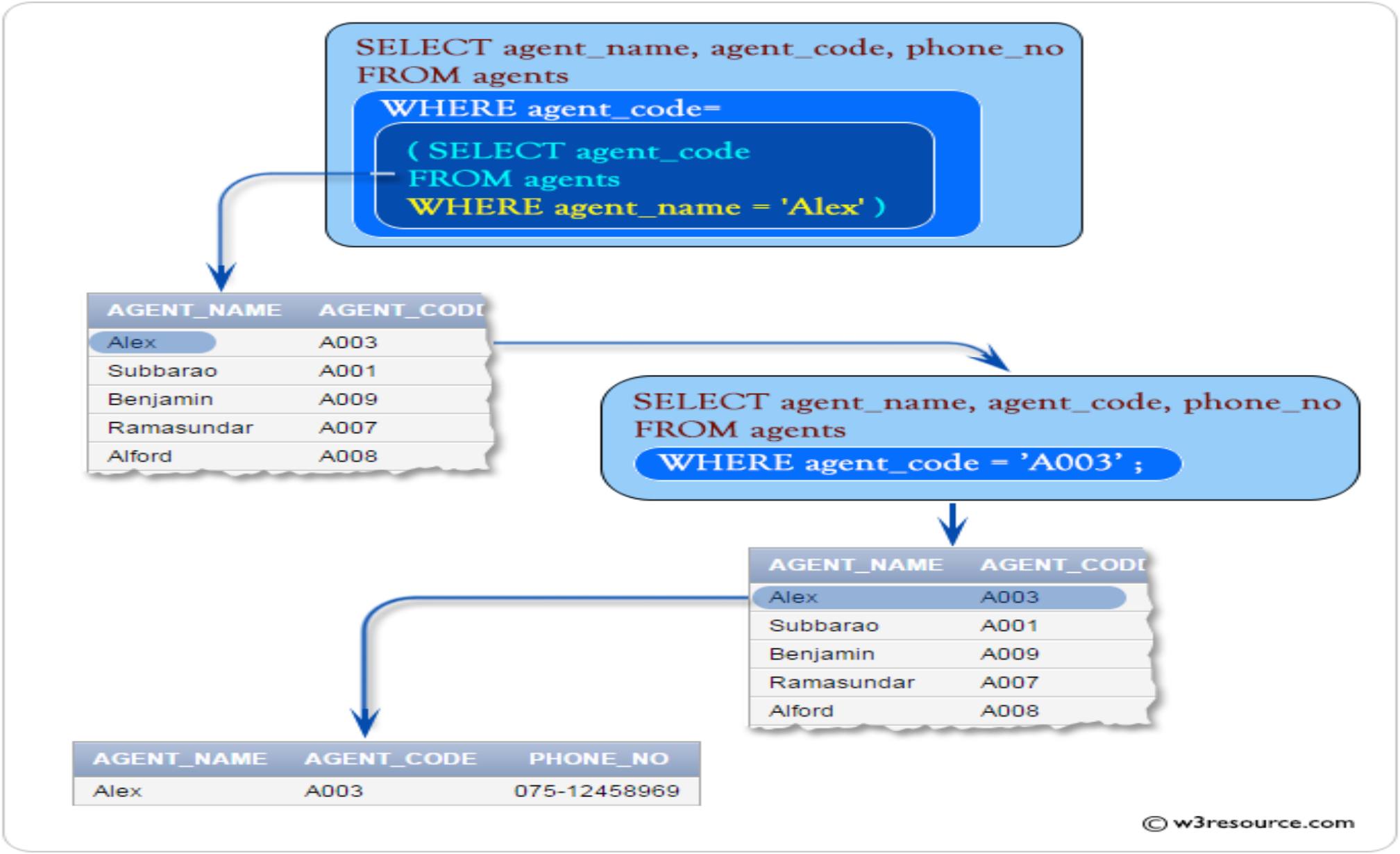
SQL Statement :

```
SELECT agent_name ,agent_code,phone_no FROM
agents WHERE agent_code=(SELECT agent_code
FROM agents WHERE agent_name ='Alex';
```

- Output

AGENT_NAME	AGENT_CODE	PHONE_NO
Alex	A003	075-12458969

Single Row Subquery Example



Multiple Row Subquery

- Multiple row subquery returns one or more rows to the outer SQL statement. We can use IN, ANY, or ALL operator in outer query to handle a subquery that returns multiple rows.

Multiple Row Subquery[Using IN]

Order Table

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200114	3500	2000	15-AUG-08	C00002	A008	
200122	2500	400	16-SEP-08	C00003	A004	
200118	500	100	20-JUL-08	C00023	A006	
200119	4000	700	16-SEP-08	C00007	A010	
200121	1500	600	23-SEP-08	C00008	A004	
200130	2500	400	30-JUL-08	C00025	A011	
200134	4200	1800	25-SEP-08	C00004	A005	
200108	4000	600	15-FEB-08	C00008	A004	
200103	1500	700	15-MAY-08	C00021	A005	
200105	2500	500	18-JUL-08	C00025	A011	
200109	3500	800	30-JUL-08	C00011	A010	
200101	3000	1000	15-JUL-08	C00001	A008	
200111	1000	300	10-JUL-08	C00020	A008	
200104	1500	500	13-MAR-08	C00006	A004	
200106	2500	700	20-APR-08	C00005	A002	
200125	2000	600	10-OCT-08	C00018	A005	
200117	800	200	20-OCT-08	C00014	A001	
200123	500	100	16-SEP-08	C00022	A002	
200120	500	100	20-JUL-08	C00009	A002	
200116	500	100	13-JUL-08	C00010	A009	
200124	500	100	20-JUN-08	C00017	A007	
200126	500	100	24-JUN-08	C00022	A002	
200129	2500	500	20-JUL-08	C00024	A006	
200127	2500	400	20-JUL-08	C00015	A003	
200128	3500	1500	20-JUL-08	C00009	A002	
200135	2000	800	16-SEP-08	C00007	A010	
200131	900	150	26-AUG-08	C00012	A012	
200133	1200	400	29-JUN-08	C00009	A002	
200100	1000	600	08-JAN-08	C00015	A003	

Multiple Row Subquery

SQL Statement :

```
SELECT ord_num,ord_amount ,ord_date ,cust_code
,agent_code FROM orders  WHERE agent_code IN (SELECT
agent_code FROM agents WHERE working_area
='Banglore');
```

- Output :

ORD_NUM	ORD_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE
200130	25000	30-JUL-08	C00025	A011
200105	2500	18-JUL-08	C00025	A011

Correlated Subquery

SQL Correlated Subqueries are used to select data from a table referenced in the outer query. The subquery is known as a correlated because the subquery is related to the outer query.

Correlated Subquery

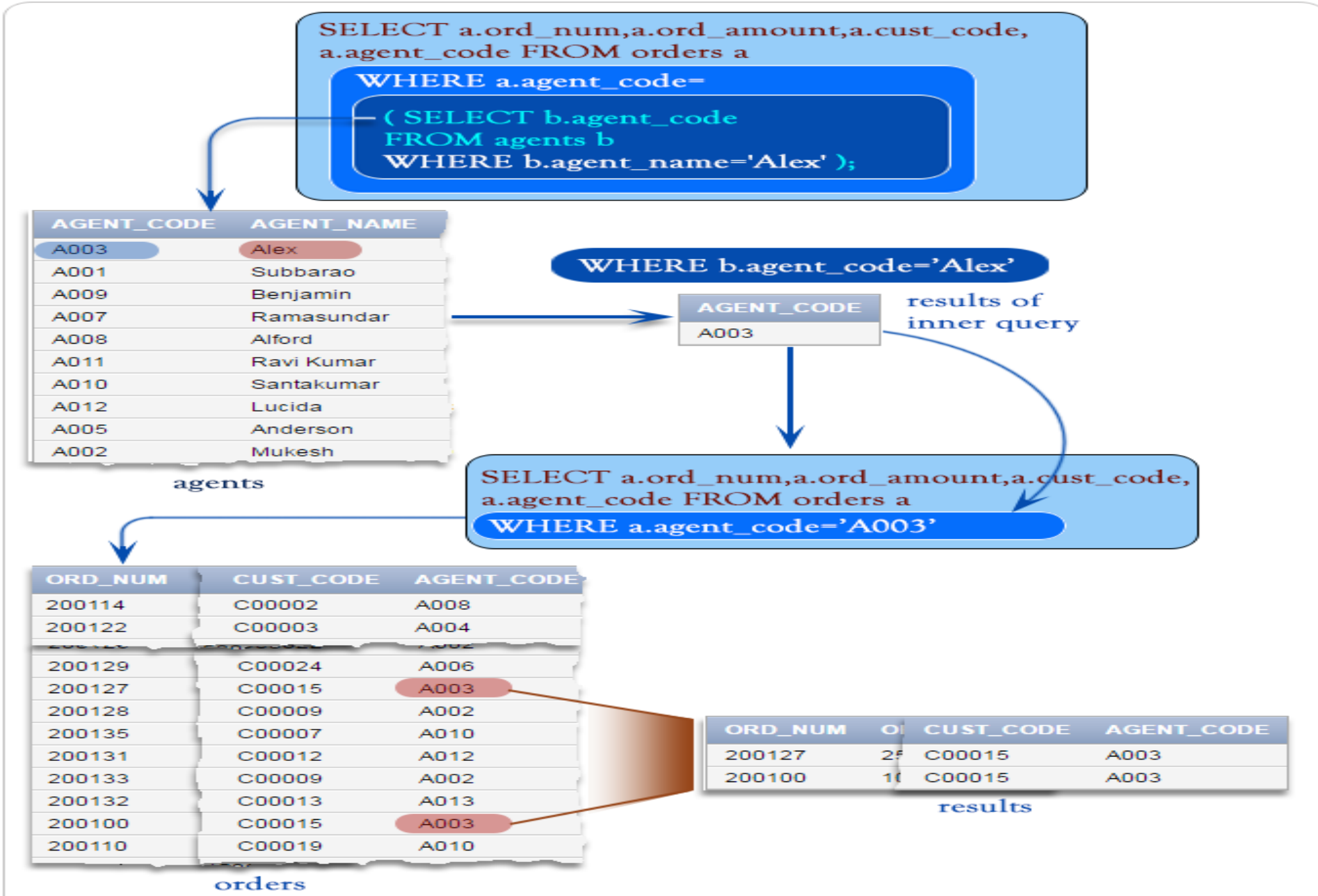
SQL Statement :-

```
SELECT  a.ord_num  ,a.ord_amount ,a.cust_code ,a.agent_code
FROM orders a WHERE  a.agent_code =(SELECT b.agent_code
FROM agents b WHERE b.agent_name='Alex');
```

Output :-

ORD_NUM	ORD_AMOUNT	CUST_CODE	AGENT_CODE
200127	2500	C00015	A003
200100	1000	C00015	A003

Correlated Subquery



Thank You