CSE 302
Database Management Systems Sessional

JOIN

Objective

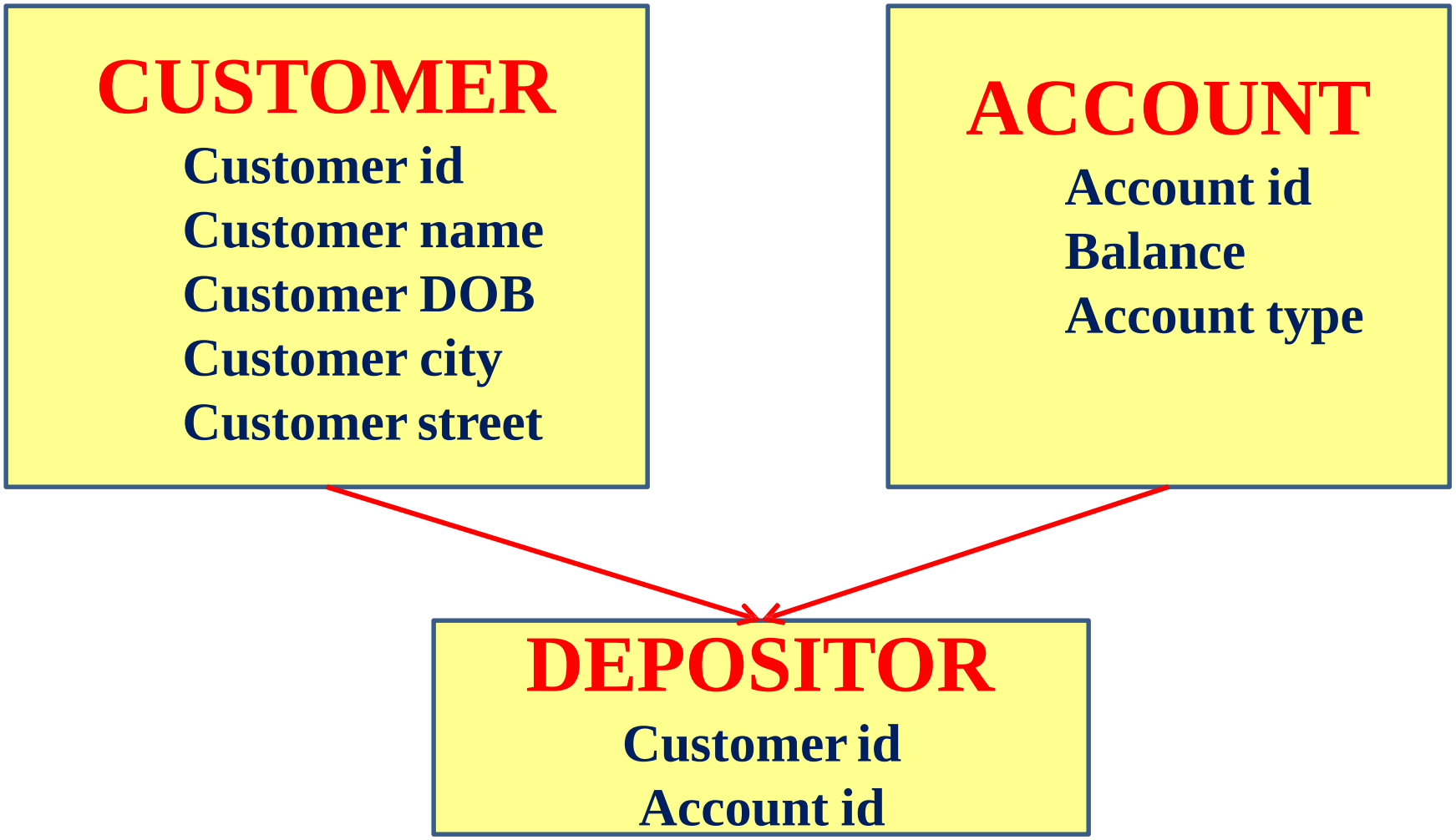
- Displaying Data from Multiple Tables

CUSTOMER

- Customer id
- Customer name
- Customer DOB
- Customer city
- Customer street

ACCOUNT

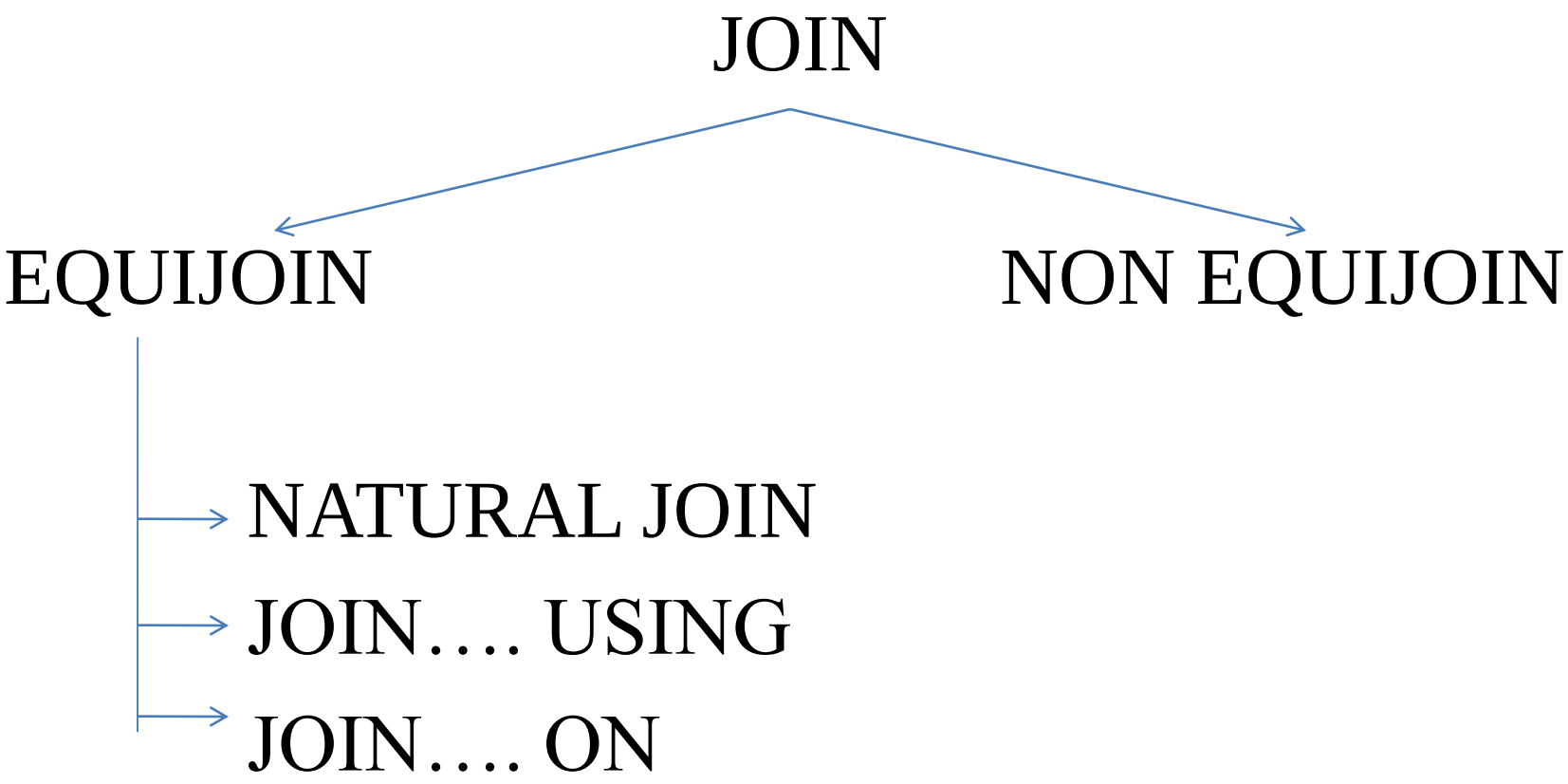
- Account id
- Balance
- Account type


CUSTOMER
Customer id
Customer name
Customer DOB
Customer city
Customer street
ACCOUNT
Account id
Balance
Account type
DEPOSITOR
Customer id
Account id

Joining Multiple Tables

- Use a Join to query data from more than one table
  - Cartesian Product OR Cross Join
  - Equality Join

Types of JOIN



Some Additional JOIN Methods

- Outer Joins
- Self Joins

Cartesian Product / Cross Join

- Also referred to as Cartesian Product or Cross Join
- A Cartesian Join is useful for performing certain statistical procedures for data analysis. Otherwise, it is very rarely used. You probably will have no reason to create it
- In most cases, a Cartesian join is the result of accidentally not including the joining condition in either the WHERE or FROM clause.

Cartesian Product / Cross Join

- If one table has 20 records and the other table has 5 records, the number of rows returned is the product of 20 * 5. This results in the display of 100 records!!

```
SELECT table1.col, table2.col
FROM table1,table2
```

Cartesian Product / Cross Join

- SELECT *cust_name*, *account_id* FROM CUSTOMER , DEPOSITOR;

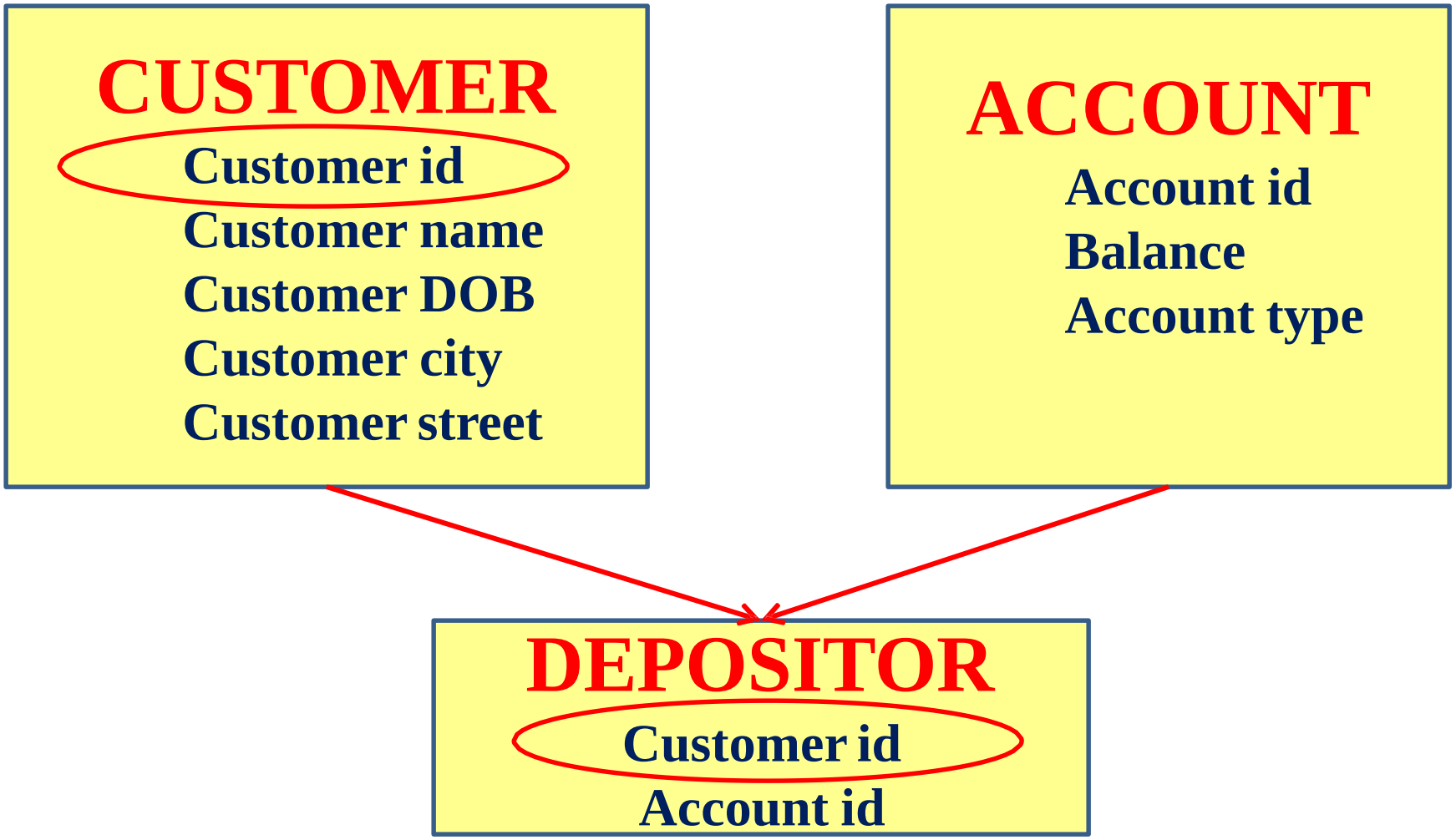
OR

- SELECT *cust_name*, *account_id* FROM CUSTOMER **CROSS JOIN** DEPOSITOR;

Cartesian Product / Cross Join

84 rows.. !!

CUST_NAME	ACCOUNT_ID
Jones	A-101
Smith	A-101
Hayes	A-101
Curry	A-101
Lindsay	A-101
Turner	A-101
Williams	A-101
Adams	A-101
Johnson	A-101
Glenn	A-101
Brooks	A-101
Green	A-101
Jones	A-215
Smith	A-215
Hayes	A-215
Curry	A-215
Lindsay	A-215
Turner	A-215
Williams	A-215
Adams	A-215
Johnson	A-215
Glenn	A-215
Brooks	A-215
Green	A-215
Jones	A-102
Smith	A-102
Hayes	A-102
Curry	A-102
Lindsay	A-102


CUSTOMER
Customer id
Customer name
Customer DOB
Customer city
Customer street
ACCOUNT
Account id
Balance
Account type
DEPOSITOR
Customer id
Account id

Equijoin / Equality Join

- Also known as Equijoins, Inner Joins, or Simple Joins
- Based upon two (or more) tables having equivalent data stored in a "**common column**"
- Example:

  Both the Customer table and the Depositor table have *Cust_ID* as a common column.

  Cust_ID contains an identification code that is assigned to each Customer.

Equijoin / Equality Join

- Two (or more) tables having equivalent data stored in a "**common column“.**

SELECT *Column Names*

FROM *Multiple Table Names*

WHERE *Condition / Access Path*

Equijoin / Equality Join - Example

Entities:
Customer (Cust_id, Cust_name, Cust_dob, Cust_street, Cust_city)
Account (Account_id, Balance, Type)
Depositor (Cust_id, Account_id)

- Display a list showing the **Customer Name from Customer Table and the Account No from Depositor Table**

SELECT *Cust_name, Account_id*

FROM CUSTOMER, DEPOSITOR

Equijoin / Equality Join - Example

- Display a list showing the **Customer Name from Customer Table and the Account No from Depositor Table**

SELECT *Cust_name, Account_id*

FROM CUSTOMER, DEPOSITOR

WHERE ***CUSTOMER.Cust_id = DEPOSITOR.Cust_id;***

Equijoin / Equality Join - Example

- To include the *Cust_id* field in the displayed rows:

SELECT *Cust_name*, ***Cust_id***, *Account_id*
FROM CUSTOMER, DEPOSITOR
WHERE *CUSTOMER.Cust_id = DEPOSITOR.Cust_id*;

Equijoin / Equality Join - Example

- To include the *Cust_id* field in the displayed rows:

SELECT *Cust_name*, ***Cust_id***, *Account_id*
FROM CUSTOMER, DEPOSITOR
WHERE *CUSTOMER.Cust_id = DEPOSITOR.Cust_id*;

The Cust_id in the SELECT clause is ambiguous because it appears in both the Customer table and the Depositor table.

Equijoin / Equality Join - Example

- To include the *Cust_id* field in the displayed rows:

  SELECT *Cust_name*, ***Customer.Cust_id***, *Account_id*
  FROM CUSTOMER, DEPOSITOR
  WHERE *CUSTOMER.Cust_id = DEPOSITOR.Cust_id*;

- Add **Customer or Depositor** **as a column qualifier**

Table Aliases

```
SELECT CUST_NAME, ACCOUNT_ID
FROM CUSTOMER C, DEPOSITOR D
WHERE C.CUST_ID = D.CUST_ID;
```

Additional Search Conditions in JOIN

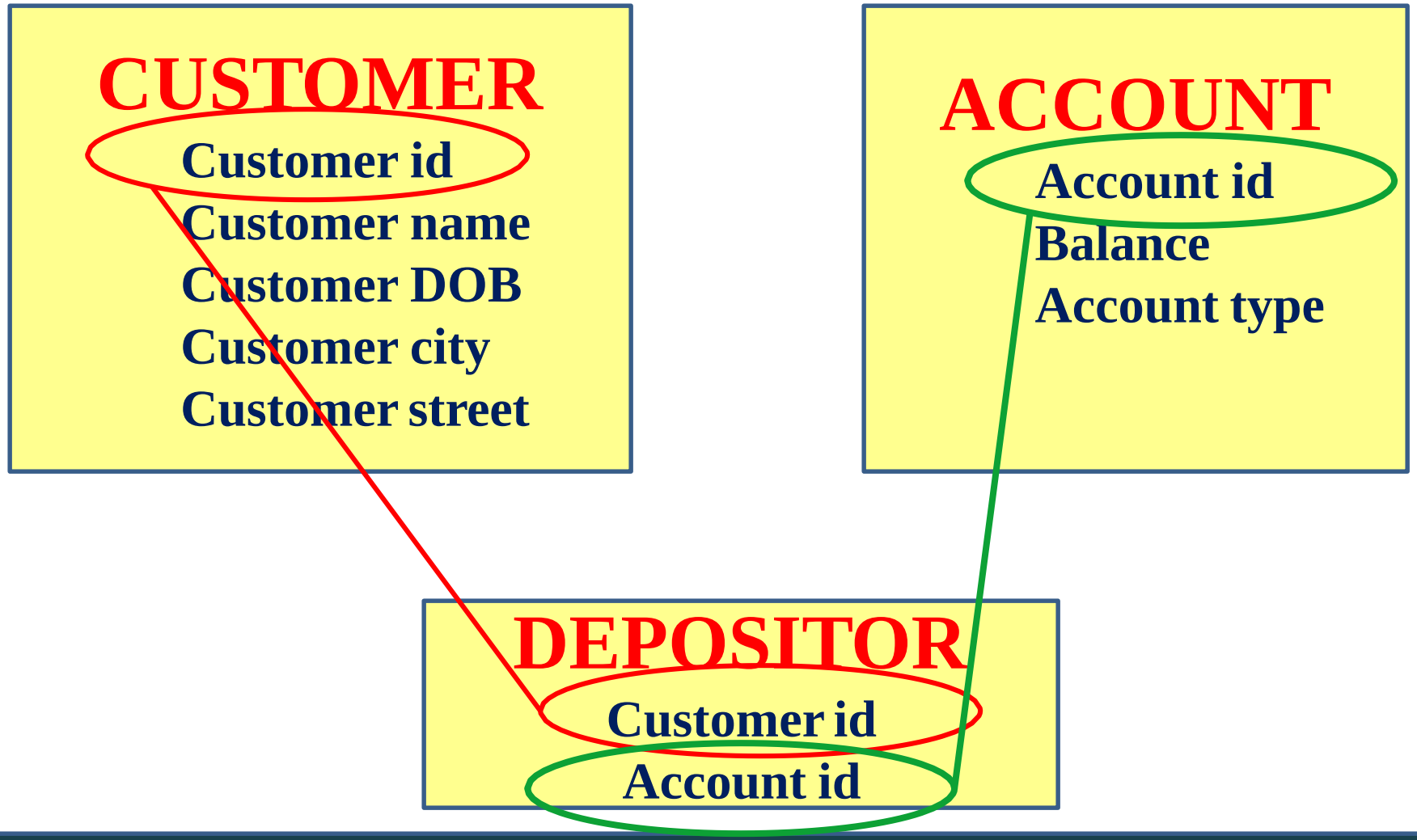
- *Display a list showing the Customer Names and Account Numbers for the customers who live in Harrison city.*

```
SELECT CUST_NAME, ACCOUNT_ID
FROM CUSTOMER C, DEPOSITOR D
WHERE C.CUST_ID = D.CUST_ID AND
C.CUST_CITY=’Harrison’;
```

Practice

- Display the **Customer Name, Account ID and Balance**

Joining More Than Two Tables



JOINING MORE THAN TWO TABLES

- *Display the Customer Name, Account ID and Balance.*

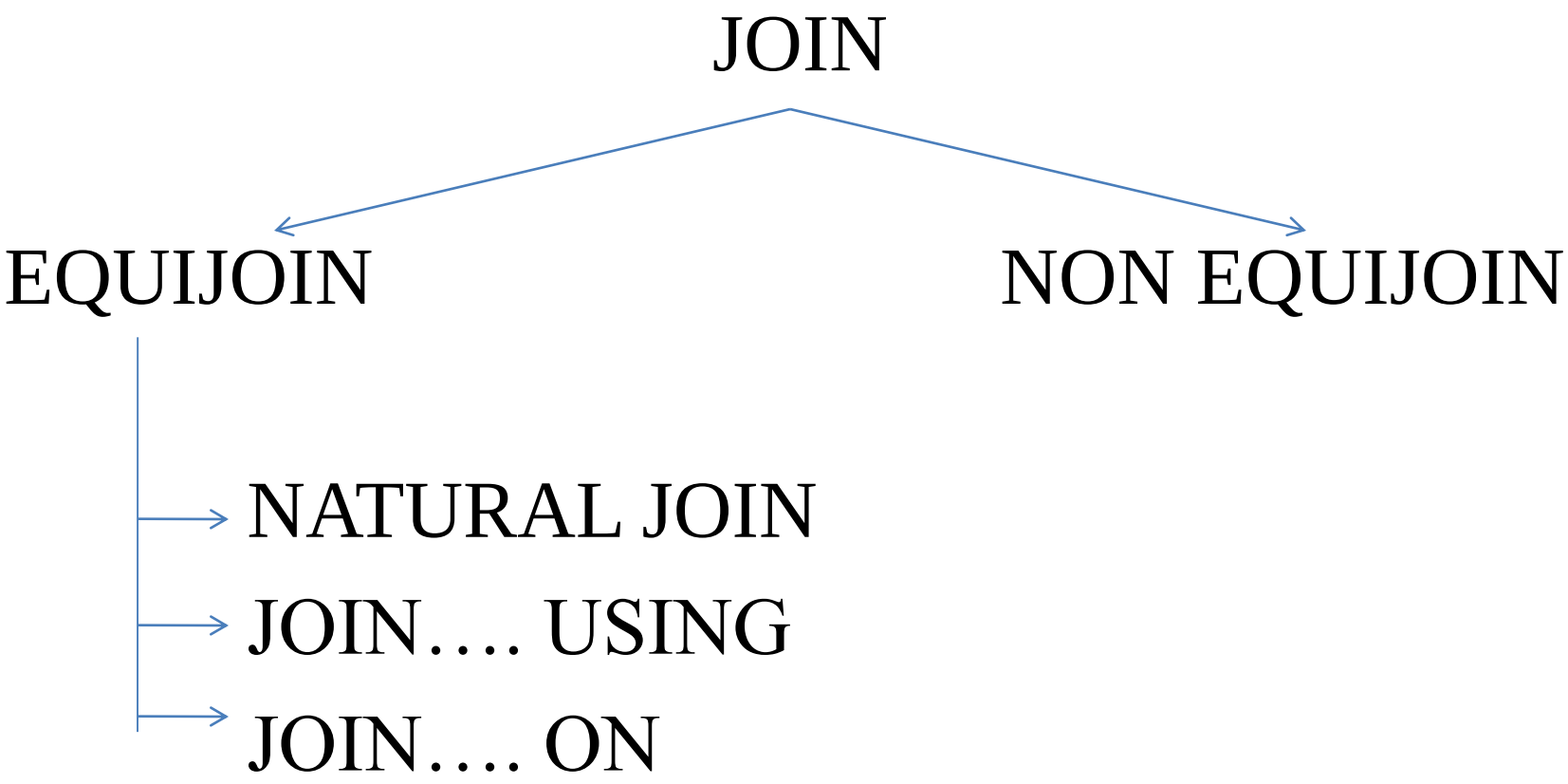
- You need to join the
  - CUSTOMER,
  - DEPOSITOR &
  - ACCOUNT.

JOINING MORE THAN TWO TABLES

- *Display the Customer Name, Account ID and Balance.*

```
SELECT C.CUST_NAME, D.CUST_ID, A.ACCOUNT_ID,
    BALANCE
FROM CUSTOMER C, DEPOSITOR D, ACCOUNT A
WHERE C.CUST_ID=D.CUST_ID AND
            D.ACCOUNT_ID=A. ACCOUNT_ID;
```

Types of JOIN



NATURAL JOIN

- Automatically joins the two tables that have a commonly named and defined field.

- SELECT …
  FROM Table 1 NATURAL JOIN Table 2;

For more than 2 tables,

- SELECT ….
  FROM T1 NATURAL JOIN T2 NATURAL JOIN T3;

NATURAL JOIN

- *Display a list showing the Customer Names and Account Numbers.*

```
SELECT CUST_NAME, CUST_ID, ACCOUNT_ID
FROM CUSTOMER NATURAL JOIN DEPOSITOR;
```

- *Display the Customer Name, Account ID and Balance.*

```
SELECT CUST_NAME, CUST_ID, ACCOUNT_ID,BALANCE
FROM CUSTOMER NATURAL JOIN DEPOSITOR NATURAL JOIN
ACCOUNT;
```

JOIN….USING

- Joins based on a column that has the same name and definition in both tables can be created with the USING clause.

```
SELECT ….
FROM T1 JOIN T2
USING (Common Column Name);
```

- USING clause can only contain the name of the common column (enclosed in parentheses).

JOIN….USING - EXAMPLE

- *Display a list showing the Customer Names and Account Numbers.*

```
SELECT CUST_NAME, CUST_ID, ACCOUNT_ID
FROM CUSTOMER JOIN DEPOSITOR USING(CUST_ID);
```

JOIN….USING - EXAMPLE

- *Display a list showing the Customer Names, Account Numbers, Balances.*

```
SELECT CUST_NAME, CUST_ID, ACCOUNT_ID , BALANCE
FROM CUSTOMER JOIN DEPOSITOR USING(CUST_ID)
JOIN ACCOUNT USING (ACCOUNT_ID);
```

JOIN….USING - EXAMPLE

- Additional condition
- *Display a list showing the Customer Names, Account Numbers, Balances of the people live in Harrison city.*

```
SELECT CUST_NAME, CUST_ID, ACCOUNT_ID , BALANCE
FROM CUSTOMER JOIN DEPOSITOR USING(CUST_ID)
JOIN ACCOUNT USING (ACCOUNT_ID)
WHERE CUST_CITY = ‘Harrison’;
```

JOIN….ON

- When tables have related (common) columns with different names, ON clause is useful.

  SELECT *Column names*
  FROM Table1 A JOIN Table2 B
  ON A.Col1=B.Col2;

- Add the table alias before the column names

JOIN….ON

- *Display a list showing the Customer Names and Account Numbers.*

```
SELECT CUST_NAME, ACCOUNT_ID
FROM CUSTOMER C JOIN DEPOSITOR D
ON C.CUST_ID=D.CUST_ID;
```

JOIN….ON

- *Display a list showing the Customer Names and Account Numbers for the customers who live in Harrison city.*

```
SELECT CUST_NAME, C.CUST_ID, ACCOUNT_ID
FROM CUSTOMER C JOIN DEPOSITOR D
ON C.CUST_ID=D.CUST_ID
WHERE C.CUST_CITY= 'Harrison';
```

JOIN….ON - PRACTICE

- *Display a list showing the Customer Names, Account Numbers, Balances.*

NON-EQUALITY JOINS

NON-EQUALITY JOINS

- With an equality join, the two tables must have exactly the same value in their common columns.
- But that's not always possible.

- A non-equality join is used when the related columns cannot be joined through the use of an equal sign.

Example

- Employee (Employee_id, Employee_name, Employee_dob, Employee_street, Employee_city, Employee_startdate, Salary, Manager_id);
- SALGRADE (GRADE,LOSAL,HISAL);

- The relationship between the EMPLOYEE and the SALGRADE table is a non-equijoin.

- No column in the EMPLOYEE table corresponds directly to a column in the SALGRADE TABLE.

Example

- *Display a list showing the Employee Names, their Salaries and Grades.*

```
SELECT E.EMPLOYEE_NAME, E.SALARY,S.GRADE
FROM EMPLOYEE E, SALGRADE S
WHERE E.SALARY BETWEEN S.LOSAL AND S.HISAL;
```

Some Additional JOINS

OUTER JOINS

- The Outer Join operator is a plus sign enclosed in parenthesis(+) and it is placed on the side of the join that is deficient in information.
- This operator has the effect of creating one or more null rows, to which one or more rows from the non deficient table can be joined.
- The outer join can appear on only one side of the expression-the side that has information missing.
- A condition involving an outer join can’t use the IN operator or be linked to another condition by the OR operator.

OUTER JOINS

- *Display a list showing the Customer Names and Account Numbers. Show all of the customers, regardless of which customers have ACCOUNT.*

```
SELECT CUST_NAME, C.CUST_ID, ACCOUNT_ID
FROM CUSTOMER C, DEPOSITOR D
WHERE C.CUST_ID= D.CUST_ID(+)
ORDER BY C.CUST_ID;
```

SELF-JOINS

- Data in a table references other data in the same table.

- Sometimes you need to join a table to itself.

SELF-JOINS

- *Find the name of each employee's manager.*

```
SELECT  WORKER.EMPLOYEE_NAME ||
            ' WORKS FOR ' || MANAGER.EMPLOYEE_NAME
             "WORKER AND MANAGER"
FROM EMPLOYEE WORKER, EMPLOYEE MANAGER
WHERE
WORKER.MGR=MANAGER.EMPLOYEE_ID;
```

Practice Problems - JOINS

- **Find all the customers who have an account as well as a loan .**
- **Find all the customers who have an account but not a loan.**
- **Display the Employee name and Employee Id along with their manager's name and manager ID.**
- **Display the list of Customer name, Customer DOB and Account Type.**

References

1. Oracle_Database_11g_The_Complete Reference
2. Oracle Built-in Datatypes
3. https://docs.oracle.com/cd/B28359_01/server.111/b28286/queries006.htm#SQLRF52331
4. Book: Database System Concepts written by Avi Silberschatz, Henry F. Korth, S. Sudarshan