UNISYS
imagine it. done.

Relational Database Management System

Agenda

- RDBMS Concepts
- ER Modeling
- Database design
- SQL Lab

Structured Query Language (SQL)

- Basic DDL statements
- DML statements
- Aggregate functions
- Grouped Results
- Relational Algebra
- Joins

SQL

- SQL is used to make a request to retrieve data from a Database.
- The DBMS processes the SQL request, retrieves the requested data from the Database, and returns it.
- This process of requesting data from a Database and receiving back the results is called a Database Query and hence the name Structured Query Language.

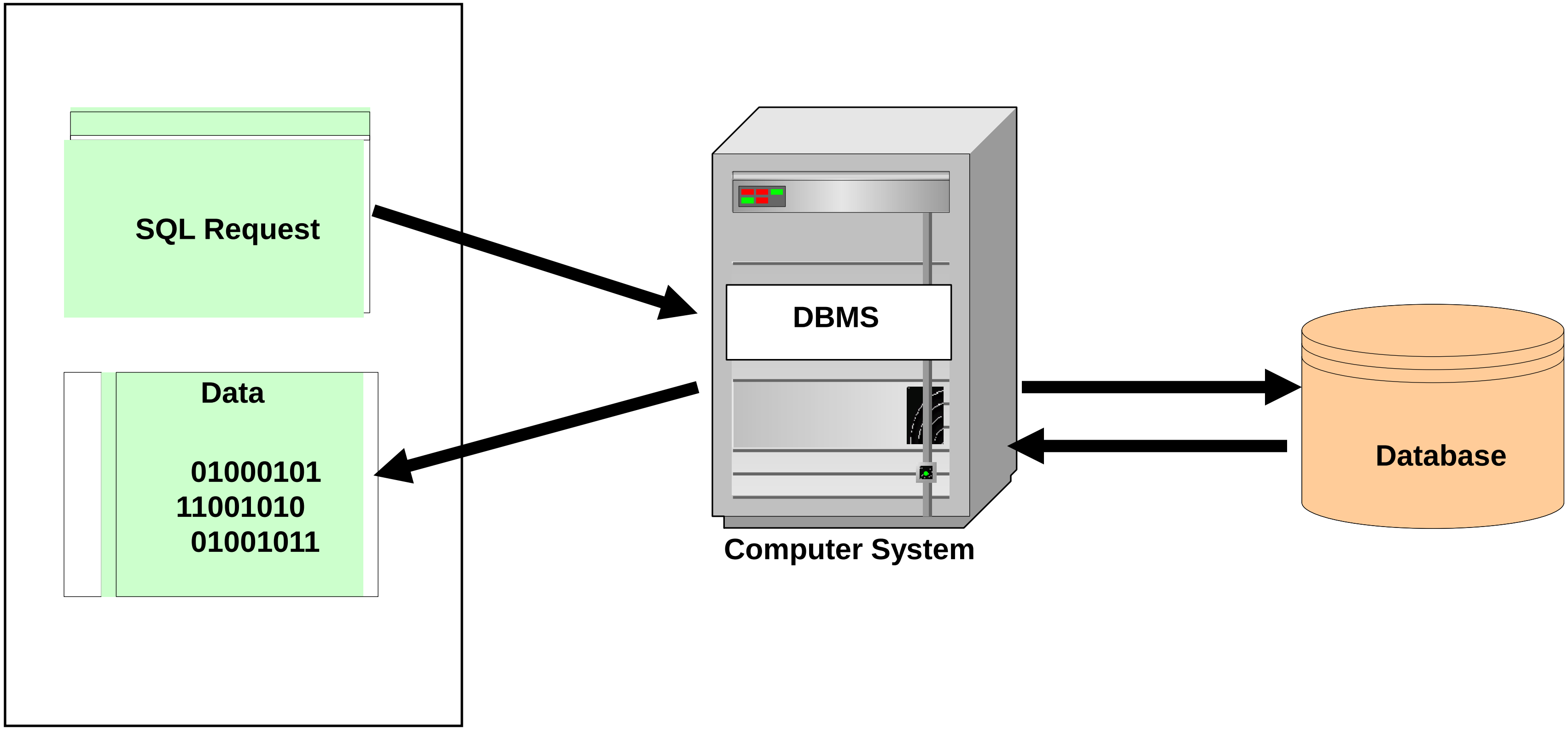
SQL

- SQL is a language that all commercial RDBMS implementations understand.

- SQL is a non-procedural language

- We would be discussing SQL with respect to **oracle** syntax

Structured Query Language (SQL)



Structured Query Language (SQL)

- **1979** Oracle Corporation introduces the first commercial RDBMS
- **1982** ANSI (American National Standards Institute) forms SQL Standards Committee
- **1983** IBM (International Business Machine) announces DB2 (a Database)
- **1986** ANSI (American National Standards Institute) SQL1 standard is approved
- **1987** ISO (International Organization for Standardization)   SQL1 standard is approved
- **1992** ANSI (American National Standards Institute) SQL2 standard is approved
- **2000** Microsoft Corp introduces SQL Server 2000, aimed at enterprise applications2002
- **2004** SQL: 2003 standard is published

Statements

- **DDL (Data Definition Language)**
  - Create
  - Alter
  - Drop
  - Truncate
- **DML (Data Manipulation Language)**
  - Insert
  - Update
  - Delete
  - Select
- **DCL (Data Control Language)**
  - Grant
  - Revoke
  - Commit
  - Rollback

Data types

- Number
- Char
- Varchar2
- Long
- date

Operators

- Arithmetic operators: +, -, *, /
- Logical operators: AND, OR, NOT
- Relational operators: =,<=,>=, < >, <, >

NULL

- Missing/unknown/inapplicable data represented as a **null** value
- NULL is not a data value. It is just an indicator that the value is unknown

SQL - Data Definition Language

UNISYS
imagine it. done.

SQL - CREATE TABLE

Syntax:

```
CREATE TABLE tablename
 (
            column_name                   data_ type                      constraints, …
)
```

Create Table (Contd…)

- **Implementing NOT NULL and Primary Key**

EXAMPLE :

```
CREATE TABLE Customer_Details
(
    Cust_ID           Number(5)         CONSTRAINT Nnull1 NOT NULL,
    Cust_Last_Name    VarChar2(20)      CONSTRAINT Nnull2 NOT NULL,
    Cust_Mid_Name     VarChar2(4),
    Cust_First_Name   VarChar2(20),
    Account_No        Number(5)         CONSTRAINT Pkey1 PRIMARY KEY,
    Account_Type      VarChar2(10)      CONSTRAINT Nnull3 NOT NULL,
    Bank_Branch       VarChar2(25)      CONSTRAINT Nnull4 NOT NULL,
    Cust_Email        VarChar2(30)
);
```

Create Table (Contd…)

- **Implementing Composite Primary Key**

EXAMPLE :

```
CREATE TABLE Customer_Details
(
    Cust_ID           Number(5) CONSTRAINT Nnull7 NOT NULL,
    Cust_Last_Name    VarChar2(20)         CONSTRAINT Nnull8 NOT NULL,
    Cust_Mid_Name     VarChar2(4),
    Cust_First_Name   VarChar2(20),
    Account_No        Number(5) CONSTRAINT Nnull9 NOT NULL,
    Account_Type      VarChar2(10)         CONSTRAINT Nnull10 NOT NULL,
    Bank_Branch       VarChar2(25)         CONSTRAINT Nnull11 NOT NULL,
    Cust_Email        VarChar2(30),
                      CONSTRAINT PKey3 PRIMARY KEY(Cust_ID,Account_No)
);
```

Create Table (Contd…)

- **Implementation of Unique Constraint**

```
Create Table UnqTable(
ECode Number(6) Constraint PK11 Primary Key,
EName Varchar2(25) Constraint NNull18 NOT NULL,
EEmail Varchar2(25) Constraint Unq1 Unique
);
```

Create Table (Contd…)

- **Implementation of Primary Key and Foreign Key Constraints**

```
CREATE TABLE EMPLOYEE_MANAGER
(
Employee_ID                 Number(6) CONSTRAINT Pkey2 PRIMARY KEY,
Employee_Last_Name          VarChar2(25),
Employee_Mid_Name           VarChar2(5),
Employee_First_Name         VarChar2(25),
Employee_Email              VarChar2(35),
Department                  VarChar2(10),
Grade                       Number(2),
MANAGER_ID                  Number(6) CONSTRAINT Fkey2
                 REFERENCES EMPLOYEE_MANAGER(Employee_ID)
);
```

Create Table (Contd…)

- **Implementation of Check Constraint**

EXAMPLE :

```
CREATE TABLE  EMPLOYEE
(
EmpNo                NUMBER(5) CONSTRAINT PKey4 Primary Key,
EmpName              Varchar(25) NOT NULL,
EmpSalary            Number(7)
                     Constraint chk Check (EmpSalary > 0 and
                     EmpSalary < 1000000)
);
```

Create Table (Contd…)

- **Implementation of Default**

```
CREATE TABLE  TABDEF(
    Ecode Number(4)  Not Null,
    Ename Varchar2(25) Not Null,
    ECity  char(10) DEFAULT ‘Mysore’
    );
```

SQL - ALTER TABLE

- **Add/Drop Column**

Syntax:
ALTER TABLE *tablename* (ADD/MODIFY/DROP column_name)

```
ALTER TABLE Customer_Details
ADD Contact_Phone  Char(10);
```

```
ALTER TABLE Customer_Details
MODIFY Contact_Phone Char(12);
```

```
ALTER TABLE Customer_Details
DROP (Contact_Phone);
```

SQL - ALTER TABLE

- **Add/Drop Primary key**

```
ALTER TABLE Customer_Details
ADD CONSTRAINT Pkey1 PRIMARY KEY (Account_No);
```

```
ALTER TABLE Customer_Details
ADD CONSTRAINT Pkey2 PRIMARY KEY (Account_No, Cust_ID);
```

```
ALTER TABLE Customer_Details
DROP PRIMARY KEY;
```

Or

```
ALTER TABLE Customer_Details
DROP CONSTRAINT Pkey1;
```

SQL - ALTER TABLE

- **Add/Drop Foreign key**

```
ALTER TABLE Customer_Transaction
ADD CONSTRAINT Fkey1 FOREIGN KEY (Account_No)            REFERENCES
Customer_Details (Account_No);
```

```
ALTER TABLE Customer_Transaction
DROP CONSTRAINT Fkey1
```

SQL - DROP TABLE

- **DROP TABLE**
  - Deletes table structure
  - Cannot be recovered
  - Use with caution

```
DROP TABLE UnqTable;
```

SQL – Truncate Table

- **Deleting All Rows of a table**

TRUNCATE TABLE Customer_Details ;

Index

- Indexing involves forming a two dimensional matrix completely independent of the table on which index is created.

- Here one column will hold the sorted data of the column which is been indexed

- Another column called the address field identifies the location of the record i.e. Row ID.

- Row Id indicates exactly where the record is stored in the table.

Index

- **Syntax**

CREATE [UNIQUE] INDEX index-name on table-name (column-name) **[ASC / DESC]**

- Index on a single column

CREATE UNIQUE INDEX Cust_Idx

    ON Customer_Details (Cust_ID);

- Index on Multiple Column

CREATE UNIQUE INDEX ID_AccountNo_Idx

    ON Customer_Details (Cust_ID, Account_No);

- Drop a Index

DROP INDEX ID_AccountNo_Idx;

Index

- **Advantages of having an INDEX:**
  - Greatly speeds the execution of SQL statements with search conditions that refer to the indexed column(s)
  - It is most appropriate when retrieval of data from tables are more frequent than inserts and updates

- **Disadvantages of having an INDEX:**
  - It consumes additional disk space
  - Additional Overhead on DML Statements

SQL - Data Manipulation Language

SQL - INSERT INTO

Syntax: INSERT INTO *tablename (Columnlist)* VALUES (*value list*)

- Single-row insert with values for all Columns

```
INSERT INTO Customer_Details

VALUES (106, 'Costner', 'A.', 'Kevin', 3350, 'Savings', 'Indus
   Bank', 'Costner_Kevin@times.com');
```

- Inserting one row, few columns at a time

```
 INSERT INTO Customer_Details
(Cust_ID,  Cust_Last_Name,  Cust_Mid_Name,  Cust_First_Name, Account_No,
    Account_Type,  Bank_Branch)
VALUES (107, 'Robert', 'B.', 'Dan', 3351, 'Savings', 'Indus Bank');
```

UNISYS
imagine it. done.

SQL - INSERT INTO

- Inserting NULL Value into a Column

```
 INSERT INTO Customer_Details
(Cust_ID,  Cust_Last_Name,  Cust_Mid_Name,  Cust_First_Name,
   Account_No, Account_Type,  Bank_Branch)
VALUES (108, 'Robert', 'B.', 'Dan', 3352, 'Savings', 'Indus Bank');
```

Or

```
 INSERT INTO Customer_Details
(Cust_ID,  Cust_Last_Name,  Cust_Mid_Name,  Cust_First_Name,
   Account_No, Account_Type,  Bank_Branch,Cust_Email)
VALUES (108, 'Robert', 'B.', 'Dan', 3352, 'Savings', 'Indus Bank‘,NULL);
```

SQL - INSERT INTO

- Inserting Many rows from a Different Table

```
INSERT INTO OldCust_details
(Account_No, Transaction_Date,Total_Available_Balance_in_Dollars)
SELECT Account_No, Transaction_Date, Total_Avail_Balance_in_Dollars
From Customer_Transaction
WHERE Total_Available_Balance_in_Dollars > 10000.00;
```

SQL - UPDATE

Syntax:

UPDATE *tablename* SET *column_name* =*value* [WHERE *condition*]

Updating All Rows

```
UPDATE Customer_Fixed_Deposit
SET Rate_of_Interest_in_Percent = NULL;
```

Updating Particular rows

```
UPDATE Customer_Fixed_Deposit
SET Rate_of_Interest_in_Percent = 7.3
WHERE Amount_in_Dollars > 3000;
```

SQL - UPDATE

- Updating Multiple Columns

```
UPDATE Customer_Fixed_Deposit
SET
    Cust_Email = ‘Quails_Jack@rediffmail.com’ ,
    Rate_of_Interest_in_Percent = 7.3
 WHERE Cust_ID = 104;
```

SQL - DELETE FROM

- With or without WHERE clause

Syntax:
DELETE FROM *tablename* WHERE *condition*

Deleting All Rows
DELETE FROM Customer_Details;

Deleting Specific Rows
DELETE
FROM Customer_Details
WHERE Cust_ID = 102;

Difference Between Delete and Truncate

DELETE	TRUNCATE
Data can be recovered	Data cannot be recovered
DML statement	DDL statement
DELETE does not release the memory occupied by the records of the table	TRUNCATE releases the memory occupied by the records of the table

Retrieving All columns from a table

To select set of column names,

```
SELECT column1, column2,… FROM TableName
```

Example

```
SELECT  *
FROM Customer_Details;
```

Or

```
SELECT Cust_ID, Cust_Last_Name, Cust_Mid_Name, Cust_First_Name,
Account_No, Account_Type, Bank_Branch, Cust_Email
FROM Customer_Details;
```

Retrieving Few Columns

```
SELECT  Cust_ID, Account_No
  FROM Customer_Details;
```

Implementing Customized Columns Names

```
SELECT Account_No  AS  “Customer Account No.”,
          Total_Available_Balance_in_Dollars   AS  “Total Balance”
                    FROM Customer_Transaction;
```

SQL - ALL, DISTINCT

Get all Customers Name:

```
SELECT ALL Cust_Last_Name
           FROM Customer_Details;
Or
SELECT Cust_Last_Name
           FROM Customer_Details;
```

Get all distinct Customer Name

```
SELECT DISTINCT Cust_Last_Name
           FROM Customer_Details;
```

Retrieving a subset of rows

- For retrieval of rows based on some condition, the syntax is

```
SELECT  COL1,COL2,.........

            FROM   TABLE NAME

                        WHERE  < SEARCH CONDITION>
```

Relational operators

- *List all customers with an account balance > $10000*

```
SELECT Account_No, Total_Available_Balance_in_Dollars
            FROM Customer_Transaction
                        WHERE Total_Available_Balance_in_Dollars > 10000.00;
```

- *List the Cust_ID, Account_No of ‘Graham’*

```
SELECT Cust_ID, Account_No
            FROM Customer_Details
                        WHERE Cust_First_Name = ‘Graham’;
```

Relational operator

= , < , > , <= , >= , != or < >

Relational operators

- *List all Account_No where Total_Available_Balance_in_Dollars is atleast $10000.00*

```
SELECT Account_No
           FROM Customer_Transaction
                      WHERE Total_Available_Balance_in_Dollars >= 10000.00;
```

Logical operators

- *List all Cust_ID, Cust_Last_Name where Account_type is ‘Savings’ and Bank_Branch is ‘Capital Bank’.*

```
SELECT Cust_ID, Cust_Last_Name
   FROM Customer_Details
       WHERE Account_Type = ‘Savings’ AND Bank_Branch = ‘Capital Bank’;
```

- *List all Cust_ID, Cust_Last_Name where neither Account_type is ‘Savings’ and nor Bank_Branch is ‘Capital Bank’*

```
SELECT Cust_ID, Cust_Last_Name
              FROM Customer_Details
                            WHERE NOT Account_Type = ‘Savings’ AND
                                          NOT Bank_Branch = ‘Capital Bank’;
```

Logical operators

- *List all Cust_ID, Cust_Last_Name where either Account_type is ‘Savings’*

  or Bank_Branch is ‘Capital Bank’.

```
SELECT Cust_ID, Cust_Last_Name
    FROM Customer_Details
         WHERE Account_Type = ‘Savings’  OR  Bank_Branch = ‘Capital Bank’;
```

Logical operator: AND, OR, and NOT

Retrieval using BETWEEN

test-expression [**NOT**] **BETWEEN** low-expression **AND** high-expression

List all Account_Nos with balance in the range $10000.00 to $20000.00.

```
SELECT Account_No
              FROM Customer_Transaction
                            WHERE Total_Available_Balance_in_Dollars >= 10000.00
                            AND Total_Available_Balance_in_Dollars <= 20000.00;
Or
SELECT Account_No
              FROM Customer_Transaction
                            WHERE Total_Available_Balance_in_Dollars
                             BETWEEN 10000.00 AND 20000.00;
```

Retrieval using IN

test-expression [**NOT**] **IN** (constant1, constant2...........)

List all customers who have account in Capital Bank or Indus Bank.

```
SELECT Cust_ID
          FROM Customer_Details
                    WHERE Bank_Branch = ‘Capital Bank’
                    OR Bank_Branch =  ‘Indus Bank’;
Or
SELECT Cust_ID
          FROM Customer_Details
                    WHERE Bank_Branch IN (‘Capital Bank’, ‘Indus Bank’);
```

Retrieval using LIKE

Column-name [**NOT**] **LIKE** pattern **ESCAPE** escape-character

List all Accounts where the Bank_Branch begins with a ‘C’ and has ‘a’ as the second character

```
SELECT Cust_ID, Cust_Last_Name, Account_No
          FROM Customer_Details
                    WHERE Bank_Branch LIKE ‘Ca%’;
```

List all Accounts where the Bank_Branch column has ‘a’ as the second character.

```
SELECT Cust_ID, Cust_Last_Name, Account_No
          FROM Customer_Details
                    WHERE Bank_Branch LIKE ‘_a%’;
```

SQL - Retrieval using IS NULL

column-name **IS** [**NOT**] **NULL**

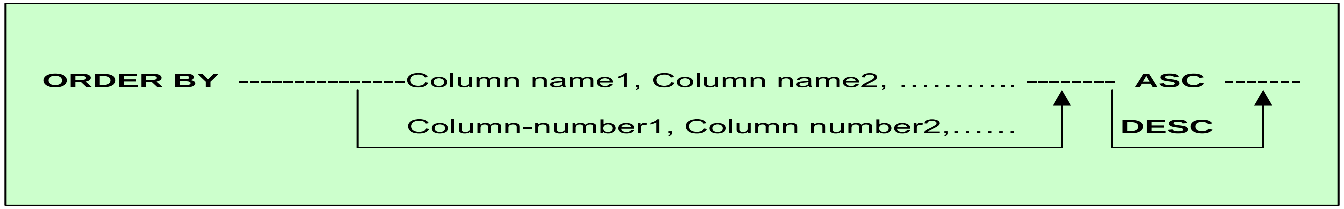
List employees who have not been assigned a Manager yet.

```
SELECT Employee_ID
            FROM Employee_Manager
                        WHERE Manager_ID IS NULL;
```

List employees who have been assigned to some Manager.

```
SELECT Employee_ID
            FROM Employee_Manager
                        WHERE Manager_ID IS NOT NULL;
```

SQL - Sorting your results (ORDER BY)



ORDER BY ---------------Column name1, Column name2, ... -------- ASC -------

Column-number1, Column number2,......

DESC

List the customers account numbers and their account balances, in the increasing order of the balance

```
SELECT Account_No, Total_Available_Balance_in_Dollars
          FROM Customer_Transaction
                    ORDER BY Total_Available_Balance_in_Dollars;
```

- *by default the order is ASCENDING*

Retrieval using ORDER BY

List the customers and their account numbers in the decreasing order of the account numbers.

```
SELECT  Cust_Last_Name, Cust_First_Name, Account_No
            FROM Customer_Details
                        ORDER BY 3 DESC;
```

Retrieval using ORDER BY

List the customers and their account numbers in the decreasing order of the Customer Last Name and increasing order of account numbers.

```
SELECT  Cust_Last_Name, Cust_First_Name, Account_No
            FROM Customer_Details
                        ORDER BY Cust_Last_Name DESC, Account_No;
Or
SELECT  Cust_Last_Name, Cust_First_Name, Account_No
            FROM Customer_Details
                        ORDER BY 1 DESC, 3;
```

Aggregate Functions

SQL - Aggregate functions

- Used when information you want to extract from a table has to do with the data in the entire table taken as a set.
- Aggregate functions are used in place of column names in the SELECT statement
- The aggregate functions in sql are :
  SUM( ) , AVG( ) , MAX( ) , MIN( ), COUNT( )

SUM ([**DISTINCT**] column-name / expression)

AVG ([**DISTINCT**] column-name / expression)

MIN (expression)

MAX (expression)

COUNT ([**DISTINCT**] column-name)

COUNT (*)

Aggregate function - MIN

- Returns the smallest value that occurs in the specified column
- Column need not be numeric type

List the minimum account balance.

```
SELECT MIN (Total_Available_Balance_in_Dollars)
          FROM Customer_Transaction;
```

Aggregate function - MAX

- Returns the largest value that occurs in the specified column
- Column need not be numeric type

Example:

List the maximum account balance.

```
SELECT MAX (Total_Available_Balance_in_Dollars)
           FROM Customer_Transaction;
```

Aggregate function - AVG

- Returns the average of all the values in the specified column
- Column must be numeric data type

Example:

List the average account balance of customers.

```
SELECT AVG (Total_Available_Balance_in_Dollars)
            FROM Customer_Transaction;
```

Aggregate function - SUM

- Adds up the values in the specified column
- Column must be numeric data type
- Value of the sum must be within the range of that data type
- **Example:**

List the minimum and Sum of all account balance.

```
SELECT MIN (Total_Available_Balance_in_Dollars),
            SUM (Total_Available_Balance_in_Dollars)
                      FROM Customer_Transaction;
```

Aggregate function - COUNT

- Returns the number of rows in the table

List total number of Employees.

```
SELECT COUNT (*)
           FROM Employee_Manager;
```

List total number of Employees who have been assigned a Manager.

```
SELECT COUNT (Manager_ID)
           FROM Employee_Manager;
```

Count(*)	=	No of rows
Count(ColumnName)	=	No. of rows that do not have NULL Value

Aggregate function - COUNT

List total number of account holders in the ‘Capital Bank’ Branch.

```
SELECT COUNT (*)
          FROM Customer_Details
                    WHERE Bank_Branch = ‘Capital Bank’;
```

List total number of unique Customer Last Names.

```
SELECT COUNT (DISTINCT Cust_Last_Name)
          FROM Customer_Details;
```

Count(*)	=	No of rows
Count(ColumnName)	=	No. of rows that do not have NULL Value

Grouped Results

SQL - Using GROUP BY

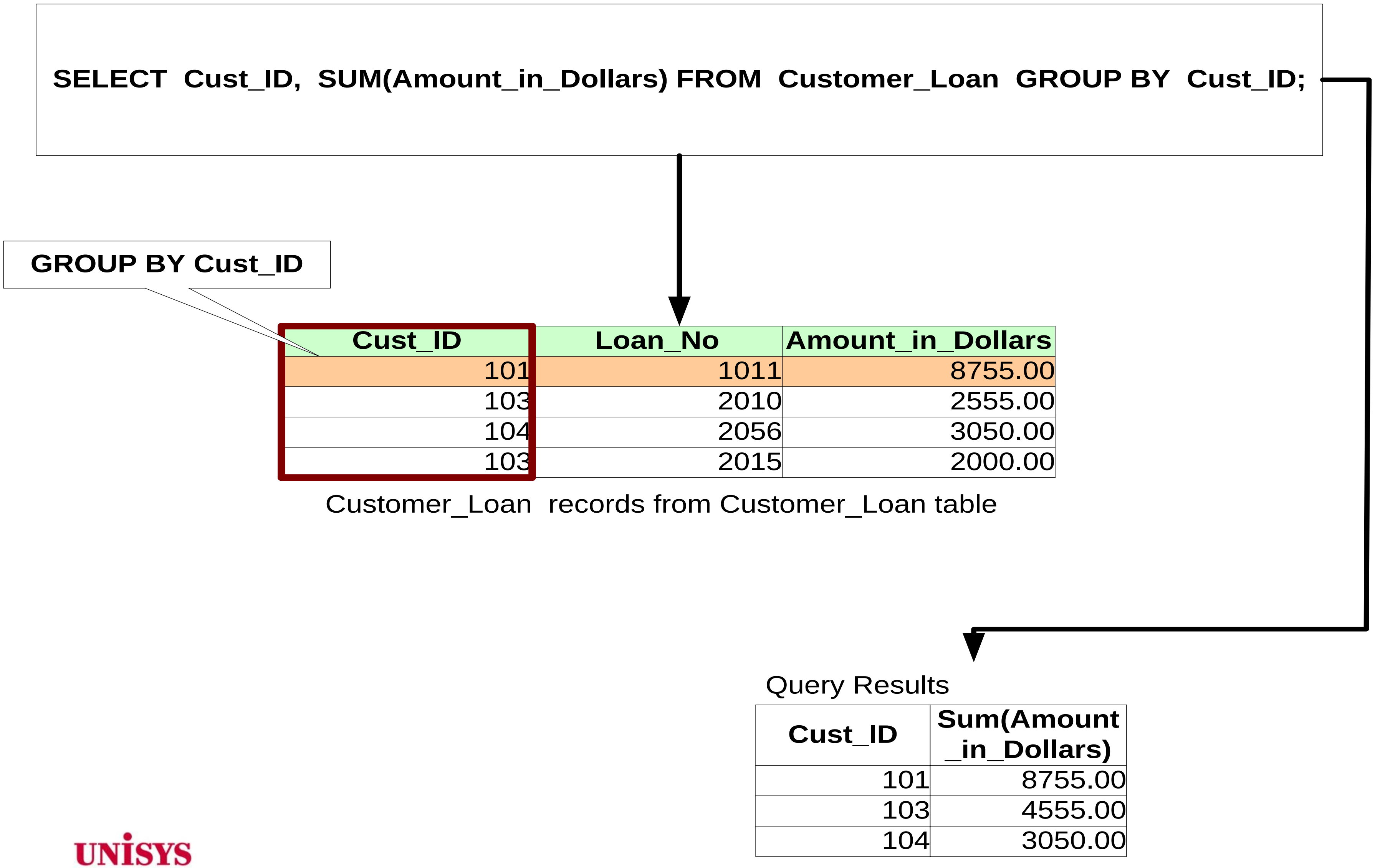
- Related rows can be grouped together by **GROUP BY** clause by specifying a column as a grouping column.

- **GROUP BY** is associated with an aggregate function

- *To retrieve the total loan-amount of all loans taken by each Customer.*

```
SELECT  Cust_ID,  SUM(Amount_in_Dollars)
            FROM  Customer_Loan
                        GROUP BY  Cust_ID;
```

SQL – Group By



```
SELECT  Cust_ID,  SUM(Amount_in_Dollars) FROM  Customer_Loan  GROUP BY  Cust_ID;
```

GROUP BY Cust_ID

Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Customer_Loan records from Customer_Loan table

Query Results

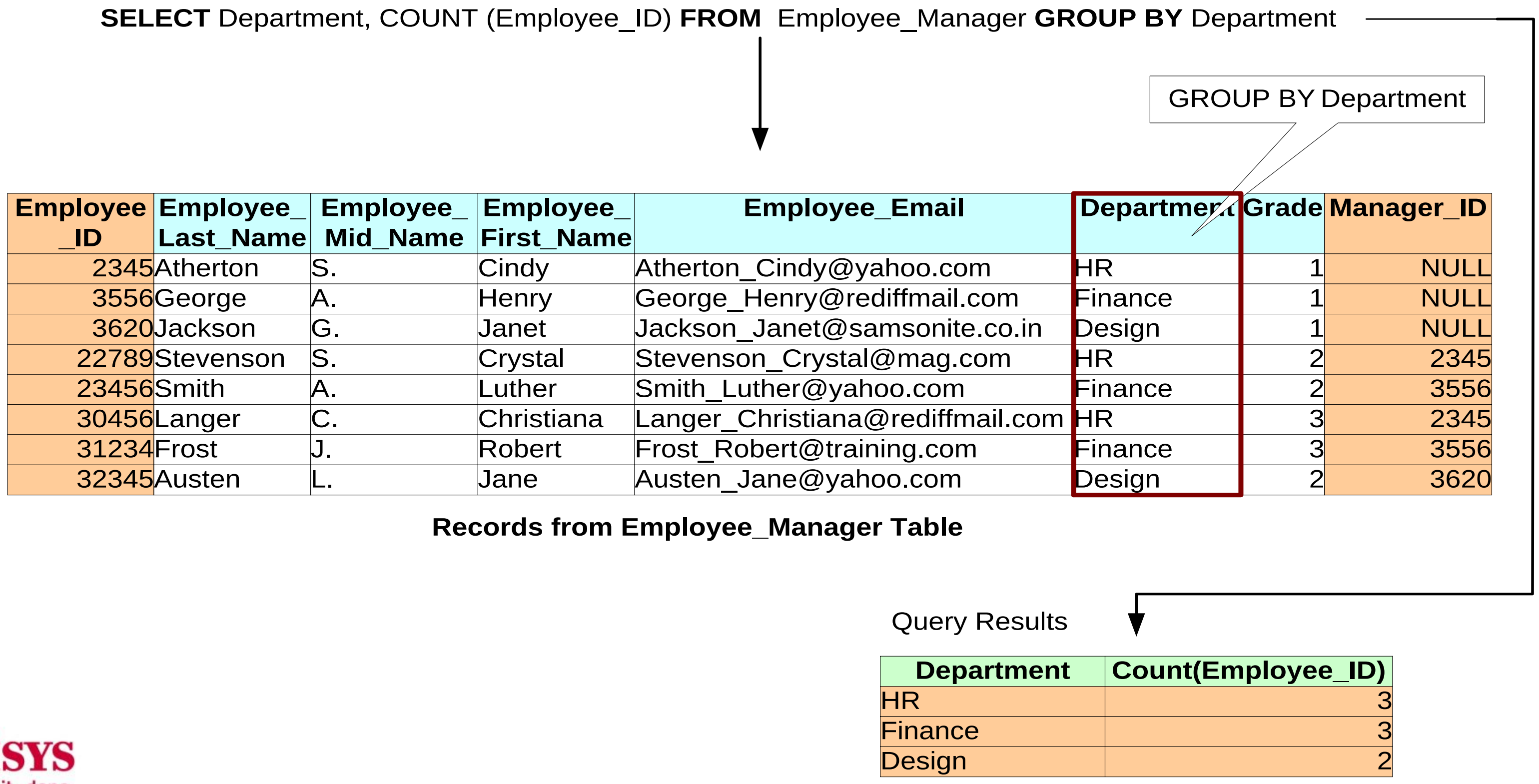
Cust_ID	Sum(Amount _in_Dollars)
101	8755.00
103	4555.00
104	3050.00

UNISYS
imagine it. done.

SQL – Group BY

- *To retrieve Number of Employees in each Department*

```
SELECT  Department, COUNT (Employee_ID)
          FROM  Employee_Manager
                    GROUP BY Department
```



SELECT Department, COUNT (Employee_ID) **FROM** Employee_Manager **GROUP BY** Department

GROUP BY Department

Employee _ID	Employee_ Last_Name	Employee_ Mid_Name	Employee_ First_Name	Employee_Email	Department	Grade	Manager_ID
2345	Atherton	S.	Cindy	Atherton_Cindy@yahoo.com	HR	1	NULL
3556	George	A.	Henry	George_Henry@rediffmail.com	Finance	1	NULL
3620	Jackson	G.	Janet	Jackson_Janet@samsonite.co.in	Design	1	NULL
22789	Stevenson	S.	Crystal	Stevenson_Crystal@mag.com	HR	2	2345
23456	Smith	A.	Luther	Smith_Luther@yahoo.com	Finance	2	3556
30456	Langer	C.	Christiana	Langer_Christiana@rediffmail.com	HR	3	2345
31234	Frost	J.	Robert	Frost_Robert@training.com	Finance	3	3556
32345	Austen	L.	Jane	Austen_Jane@yahoo.com	Design	2	3620

Records from Employee_Manager Table

Query Results

Department	Count(Employee_ID)
HR	3
Finance	3
Design	2

UNISYS
imagine it. done.

Retrieval using GROUP BY

Example:

Invalid SQL statement

```
SELECT Department, Manager_ID, COUNT(Employee_ID)
           FROM Employee_Manager
                      GROUP BY Manager_ID;
```

✗

Valid SQL Statement

```
SELECT Department, Manager_ID, COUNT(Employee_ID)
           FROM Employee_Manager
                      GROUP BY Manager_ID, Department;
```

✓

SQL – Group By

SELECT Department, Manager_ID, COUNT (Employee_ID) **FROM** Employee_Manager
GROUP BY Manager_ID, Department

Group By Manager_ID,Department

Employee_ID	Employee_Last_Name	Employee_Mid_Name	Employee_First_Name	Employee_Email	Department	Grade	Manager_ID
2345	Atherton	S.	Cindy	Atherton_Cindy@yahoo.com	HR	1	NULL
3556	George	A.	Henry	George_Henry@rediffmail.com	Finance	1	NULL
3620	Jackson	G.	Janet	Jackson_Janet@samsonite.co.in	Design	1	NULL
22789	Stevenson	S.	Crystal	Stevenson_Crystal@mag.com	HR	2	2345
23456	Smith	A.	Luther	Smith_Luther@yahoo.com	Finance	2	3556
30456	Langer	C.	Christiana	Langer_Christiana@rediffmail.com	HR	3	2345
31234	Frost	J.	Robert	Frost_Robert@training.com	Finance	3	3556
32345	Austen	L.	Jane	Austen_Jane@yahoo.com	Design	2	3620

Records from Employee_Manager Table

Query Results

Department	Manager_ID	Count(Employee_ID)
HR	2345	2
Finance	3556	2
Design	3620	1
HR	NULL	1
Finance	NULL	1
Design	NULL	1

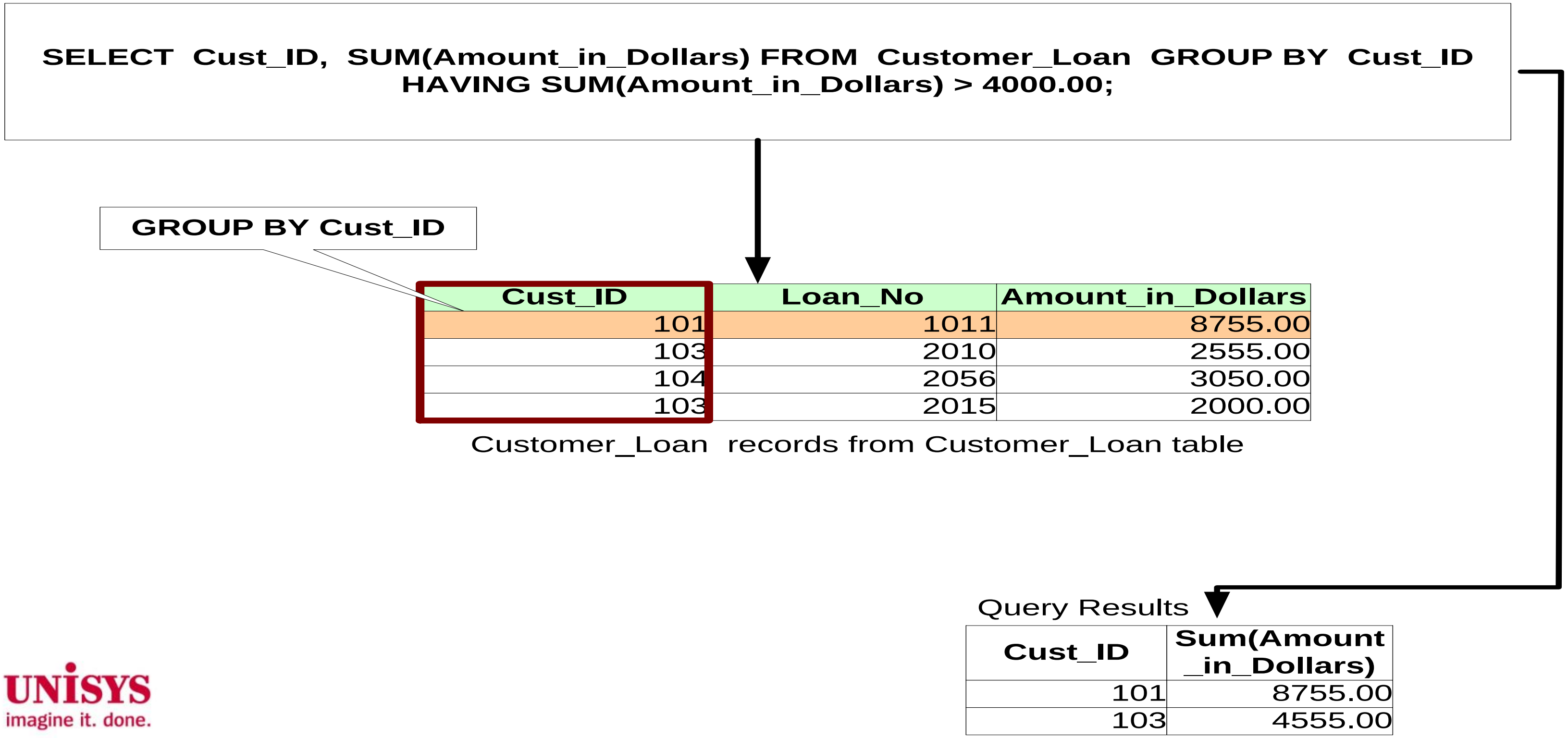
UNISYS
imagine it. done.

Retrieval using HAVING

- Used to specify condition on group

List all customers who are having loans greater than 4000

```
Select Cust_ID,SUM(Amount_in_Dollars)
            From Customer_Loan
               Group By Cust_ID   Having SUM(Amount_in_Dollars) > 4000.00;
```



```
SELECT  Cust_ID,  SUM(Amount_in_Dollars) FROM  Customer_Loan  GROUP BY  Cust_ID
HAVING SUM(Amount_in_Dollars) > 4000.00;
```

GROUP BY Cust_ID

Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Customer_Loan records from Customer_Loan table

Query Results

Cust_ID	Sum(Amount _in_Dollars)
101	8755.00
103	4555.00

Can you identify any error…?

```
Select Cust_ID,SUM(Amount_in_Dollars)
From Customer_Loan
Group By Cust_ID  Having LOAN_NO > 4000.00;
```

Ans:
The Having condition has to be based on some column that appears in the select list

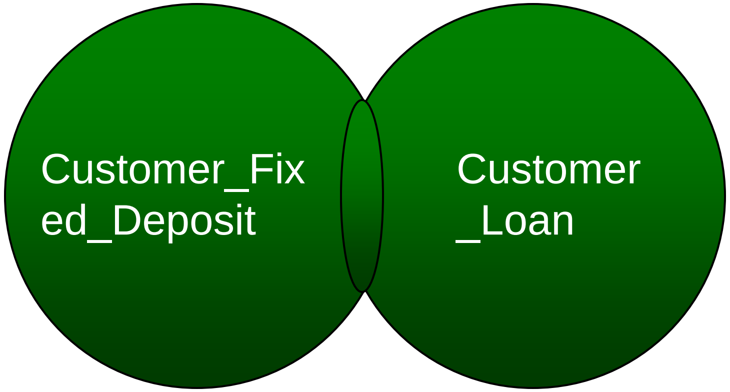
Relational Algebra Operations

Set Operations

Retrieval using UNION

List all the customer who has either Fixed Deposit or Loan or Both

```
SELECT Cust_ID
FROM Customer_Fixed_Deposit
          UNION
SELECT Cust_ID
FROM    Customer_Loan;
```



The UNION operation

- Combines the rows from two sets of query results.
- By default, the UNION operation eliminates duplicate rows as part of its processing.

Union (Contd…)

Cust_ID	Cust_Last_Name	Cust_Mid_Name	Cust_First_Name	Cust_Email	Fixed_Deposit_No	Amount_in_Dollars	Rate_of_Interest_in_Percent
101	Smith	A.	Mike	Smith_Mike@yahoo.com	2011	8055.00	6.5
103	Langer	G.	Justin	Langer_Justin@yahoo.com	2015	2060.00	6.5
104	Quails	D.	Jack	Quails_Jack@yahoo.com	3010	3050.00	6.5

Customer_Fixed_Deposit records from Customer_Fixed_Deposit table

Cust_ID
101
103
104

Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Customer_Loan records from Customer_Loan table

Cust_ID
101
103
104
103

UNION

Query Results

101
103
104

Union All

SELECT Cust_ID **FROM** Customer_Fixed_Deposit

UNION ALL

SELECT Cust_ID **FROM** Customer_Loan;

Cust_ID	Cust_Last_Name	Cust_Mid_Name	Cust_First_Name	Cust_Email	Fixed_Deposit_No	Amount_in_Dollars	Rate_of_Interest_in_Percent
101	Smith	A.	Mike	Smith_Mike@yahoo.com	2011	8055.00	6.5
103	Langer	G.	Justin	Langer_Justin@yahoo.com	2015	2060.00	6.5
104	Quails	D.	Jack	Quails_Jack@yahoo.com	3010	3050.00	6.5

Customer_Fixed_Deposit records from Customer_Fixed_Deposit table

Cust_ID
101
103
104

Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Customer_Loan records from Customer_Loan table

Cust_ID
101
103
104
103

UNION ALL

Query Results

101
103
104
103
101
103
104

Union - Restrictions

- The SELECT statements must contain the **same number of columns**

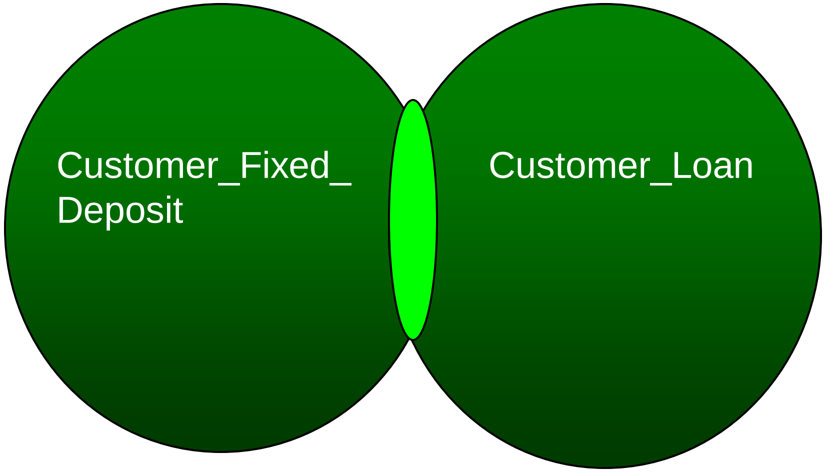
- Data type
  - Each column in the first table must be the same as the **data type** of the corresponding column in the second table.
  - Data width and column name can differ

- Neither of the two tables can be sorted with the **ORDER BY** clause.
  - **Combined query results can be sorted**

Retrieval using INTERSECT

List all the customer who have both Fixed Deposit and Loan.

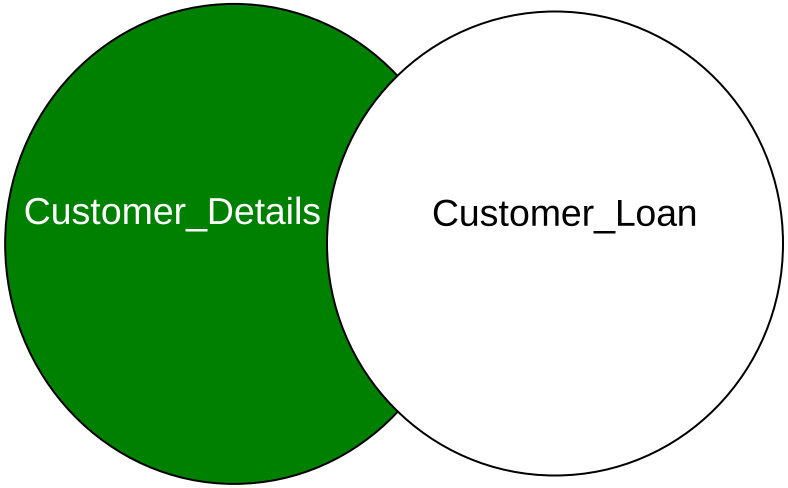
```
SELECT Cust_ID
FROM Customer_Fixed_Deposit
          INTERSECT
SELECT Cust_ID
FROM Customer_Loan;
```



Minus

- Get All the Customer who have not taken loan

```
Select Cust_ID from Customer_details
Minus
Select Cust_Id from Customer_loan;
```



Other RA operations

- Restriction
- Projection
- Join

Restriction

- Restricts the rows that can be chosen from a relation using a **WHERE** clause

- Takes a **horizontal subset of values** from the original relation

  - Example: select * from employee where salary > 10000;

Projection

- Projection is projecting a set of attributes of a relation so that rows of values corresponding to those columns will figure in the output

- This takes a **vertical subset** of the relation

- Example: select empid, name, salary from employee;

Join

JOIN

- Cartesian Product
- Inner join
- Equi join
- Outer join
  - Left-outer join
  - Right-outer join
- Self join

Cartesian Product Or Cross Join

- Returns **All** rows from first table, Each row from the first table is combined with all rows from the second table

Example

Select * from Table1,Table2;

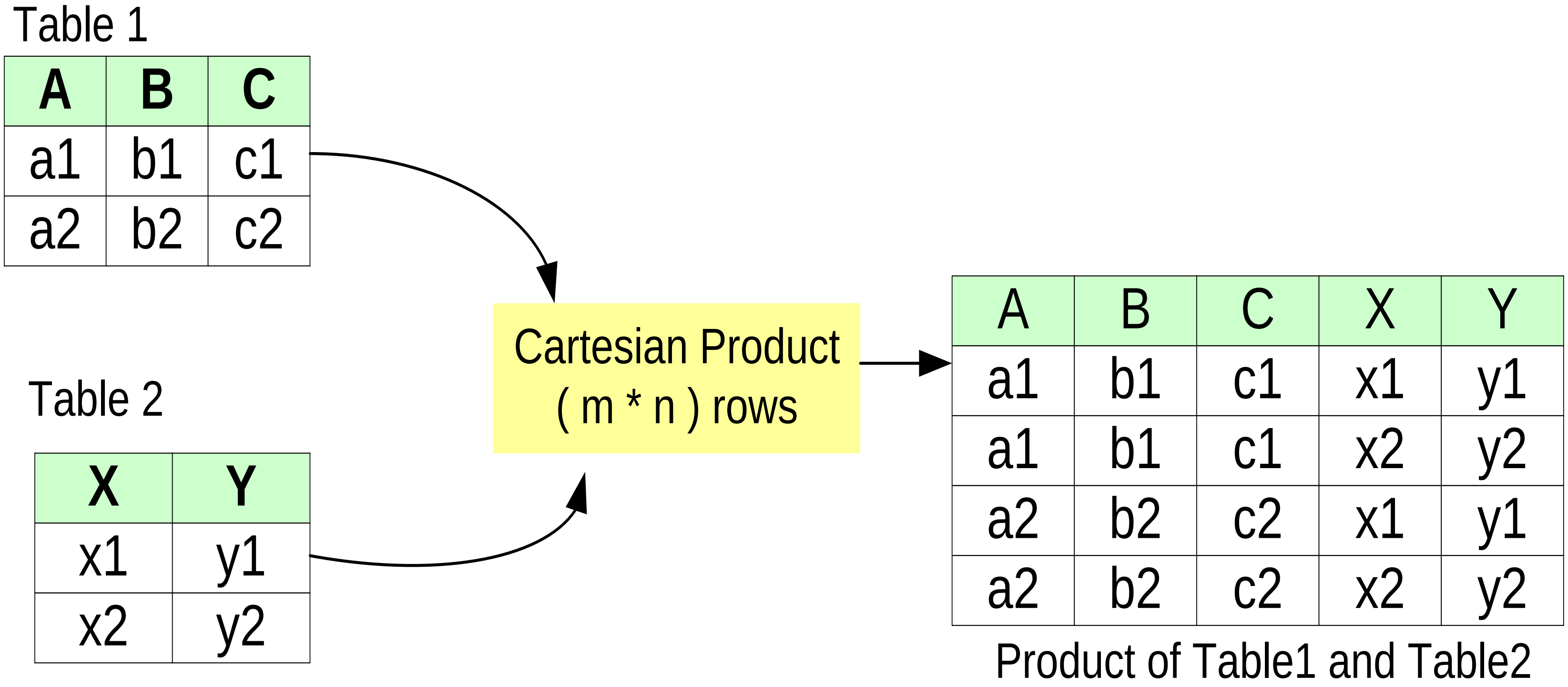


Table 1

A	B	C
a1	b1	c1
a2	b2	c2

Table 2

X	Y
x1	y1
x2	y2

Cartesian Product
(m * n) rows

A	B	C	X	Y
a1	b1	c1	x1	y1
a1	b1	c1	x2	y2
a2	b2	c2	x1	y1
a2	b2	c2	x2	y2

Product of Table1 and Table2

Inner Joins

- Common type of join

- An inner join between two (or more) tables is the Cartesian product that **satisfies the join condition in the WHERE clause**

Retrieval from Multiple tables-Equi join

Get all combinations of emp and cust information such that the emp and cust are co-located.

```
SELECT Table1.Emp_ID, Table1.City, Table2.Cust_ID, Table2.City
          FROM Table1, Table2
                    WHERE Table1.City = Table2.City;
```

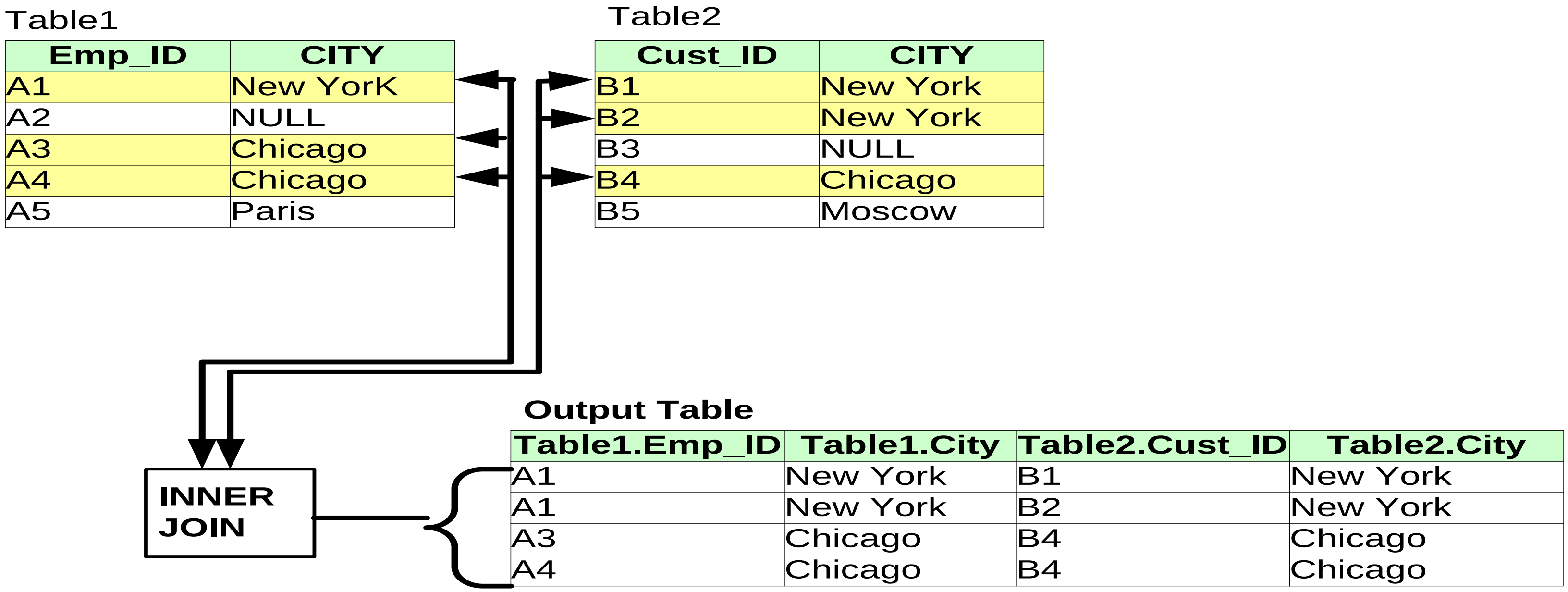


Table1

Emp_ID	CITY
A1	New York
A2	NULL
A3	Chicago
A4	Chicago
A5	Paris

Table2

Cust_ID	CITY
B1	New York
B2	New York
B3	NULL
B4	Chicago
B5	Moscow

Output Table

Table1.Emp_ID	Table1.City	Table2.Cust_ID	Table2.City
A1	New York	B1	New York
A1	New York	B2	New York
A3	Chicago	B4	Chicago
A4	Chicago	B4	Chicago

UNISYS
imagine it. done.

Retrieval from Multiple tables- Equi join

Display the First and Last Name of Customer who have taken Loan

```
Select a.Cust_Id,b.Cust_First_Name,b.Cust_Last_Name
           from Customer_loan a, customer_details b
                      where a.cust_id = b.cust_id;
```

Outer join

- Retrieve all rows that match the **WHERE** clause and also those that have a **NULL** value in the column used for join.

Left/Right-Outer join

- Left outer joins include all records from the first (left) of two tables,

  A = B **(+)**

- Right outer joins include all records from the second (right) of two tables,

  A **(+)** = B

Example of left-join

List all cities of Table1 if there is match in cities in Table2 & also unmatched Cities from Table1

```
SELECT Table1.Emp_ID, Table1.City, Table2.Cust_ID, Table2.City
FROM Table1, Table2
WHERE Table1.City  = Table2.City (+);
```

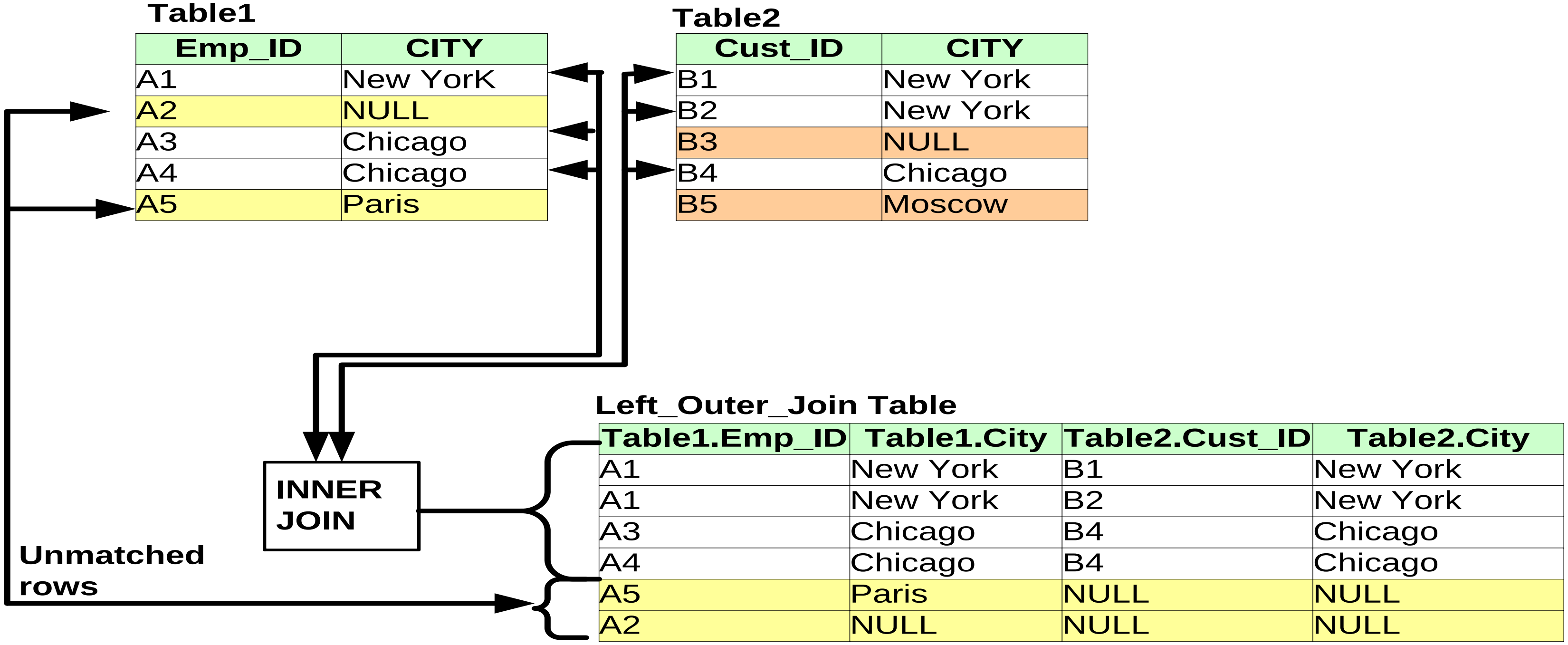


Table1

Emp_ID	CITY
A1	New YorK
A2	NULL
A3	Chicago
A4	Chicago
A5	Paris

Table2

Cust_ID	CITY
B1	New York
B2	New York
B3	NULL
B4	Chicago
B5	Moscow

Left_Outer_Join Table

Table1.Emp_ID	Table1.City	Table2.Cust_ID	Table2.City
A1	New York	B1	New York
A1	New York	B2	New York
A3	Chicago	B4	Chicago
A4	Chicago	B4	Chicago
A5	Paris	NULL	NULL
A2	NULL	NULL	NULL

Example of Left Outer Join

- List **all** customer details and loan details if they have availed loans.

Select Customer_details.Cust_id,Cust_Last_name,Loan_no,Amount_in_dollars

from Customer_details,Customer_loan

where Customer_details.Cust_id = Customer_loan.Cust_id **(+)**;

Example of right outer join

SELECT Table1.Emp_ID, Table1.City, Table2.Cust_ID, Table2.City

FROM Table1, Table2

WHERE Table1.City (**+**) = Table2.City;

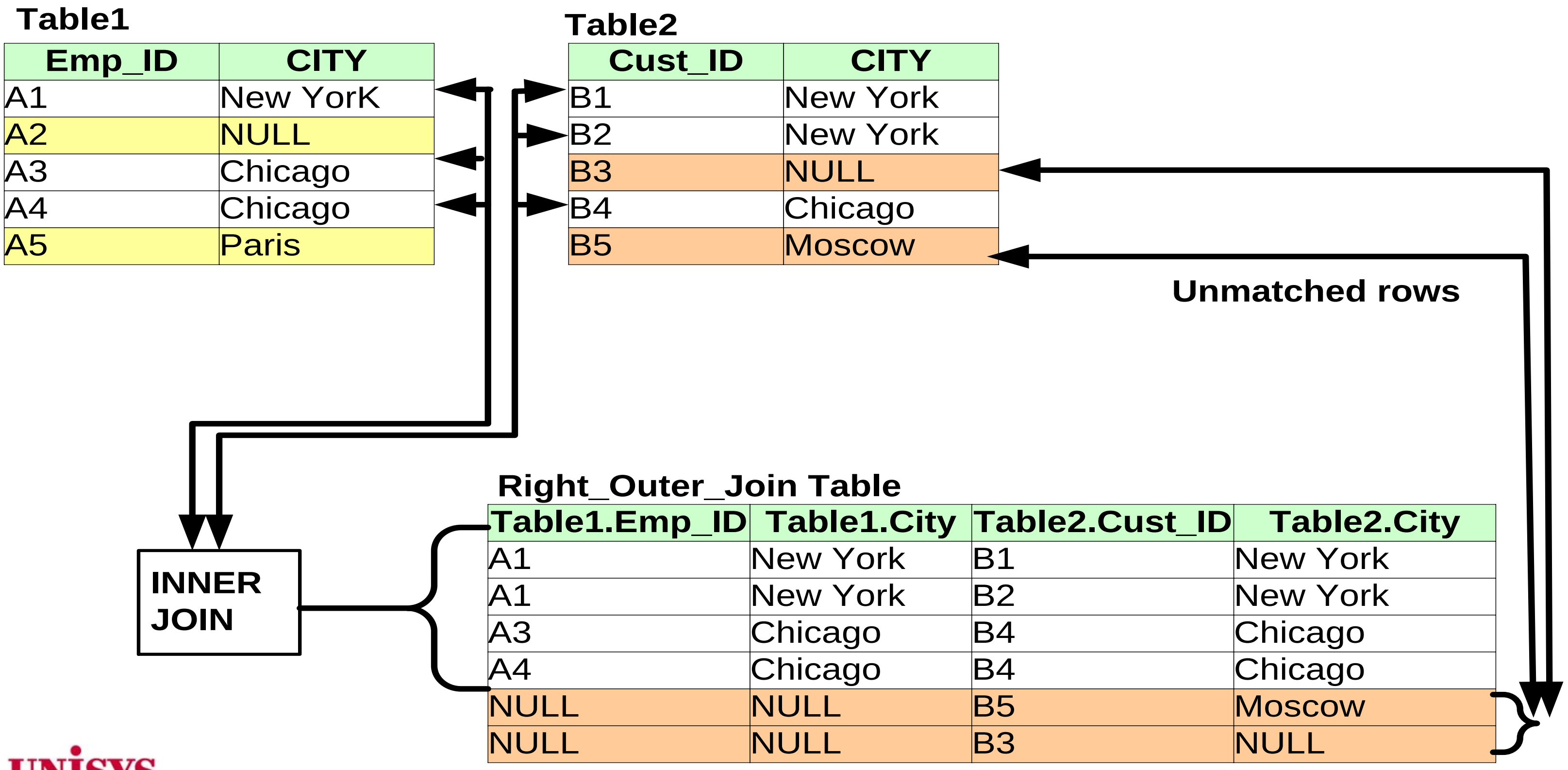


Table1

Emp_ID	CITY
A1	New YorK
A2	NULL
A3	Chicago
A4	Chicago
A5	Paris

Table2

Cust_ID	CITY
B1	New York
B2	New York
B3	NULL
B4	Chicago
B5	Moscow

Unmatched rows

INNER JOIN

Right_Outer_Join Table

Table1.Emp_ID	Table1.City	Table2.Cust_ID	Table2.City
A1	New York	B1	New York
A1	New York	B2	New York
A3	Chicago	B4	Chicago
A4	Chicago	B4	Chicago
NULL	NULL	B5	Moscow
NULL	NULL	B3	NULL

Self join-Joining a table with itself

To list all the Employees along with their Managers

```
Select
        Emp.Employee_ID                  as      "Employee ID",
        Emp.Employee_Last_Name           as      "Employee Last Name",
        Emp.Employee_first_Name          as      "Employee First Name",
        Emp.Manager_Id                   as      "Manager ID",
        Manager.Employee_Last_Name       as      "Manager Last Name",
        Manager.Employee_first_Name      as      "Manager first Name"

From    employee_Manager   Emp ,  employee_Manager   Manager

Where Emp.Manager_ID  =  Manager.Employee_ID;
```

Self Join (Contd…)

Employee_ID	Employee_Last_Name	Employee_Mid_Name	Employee_First_Name	Employee_Email	Department	Grade	Manager_ID
2345	Atherton	S.	Cindy	Atherton_Cindy@yahoo.com	HR	1	NULL
3556	George	A.	Henry	George_Henry@rediffmail.com	Finance	1	NULL
3620	Jackson	G.	Janet	Jackson_Janet@samsonite.co.in	Design	1	NULL
22789	Stevenson	S.	Crystal	Stevenson_Crystal@mag.com	HR	2	2345
23456	Smith	A.	Luther	Smith_Luther@yahoo.com	Finance	2	3556
30456	Langer	C.	Christiana	Langer_Christiana@rediffmail.com	HR	3	2345
31234	Frost	J.	Robert	Frost_Robert@training.com	Finance	3	3556
32345	Austen	L.	Jane	Austen_Jane@yahoo.com	Design	2	3620

```
SELECT Emp.Employee_ID as “Employee ID”, Emp.Employee_Last_Name as “Employee Last Name”, Emp.Employee_First_Name as “Employee First Name”,Emp.Manager_ID as “Manager ID”, Manager.Employee_Last_Name as “Manager Last Name” , Manager.Employee_First_Name as “Manager First Name”

FROM Employee_Manager  Emp, Employee_Manager  Manager

WHERE Emp.Manager_ID = Manager.Employee_ID ;
```

Self Join

Query Results

Employee ID	Employee Last Name	Employee First Name	Manager ID	Manager Last Name	Manager First Name
22789	Stevenson	Crystal	2345	Atherton	Cindy
23456	Smith	Luther	3556	George	Henry
30456	Langer	Christiana	2345	Atherton	Cindy
31234	Frost	Robert	3556	George	Henry
32345	Austen	Jane	3620	Jackson	Janet

Summary of basic DDL and DML

- Create , Alter and Drop are the DDL commands
- Update, Insert, Delete are basic DML commands to add/ remove data
- Various flavors of Select statement, used to retrieve information from the table
- Aggregate functions work on all the rows of the table taken as a group (based on some condition optionally)
- The result of a query can be grouped based on a grouping column
- To check for conditions after grouping by a column, Having is used instead of where
- Grouped queries help look at data category wise

Independent Sub-queries

Independent sub-queries

- Inner query is independent of outer query.

- Inner query is executed first and the results are stored.

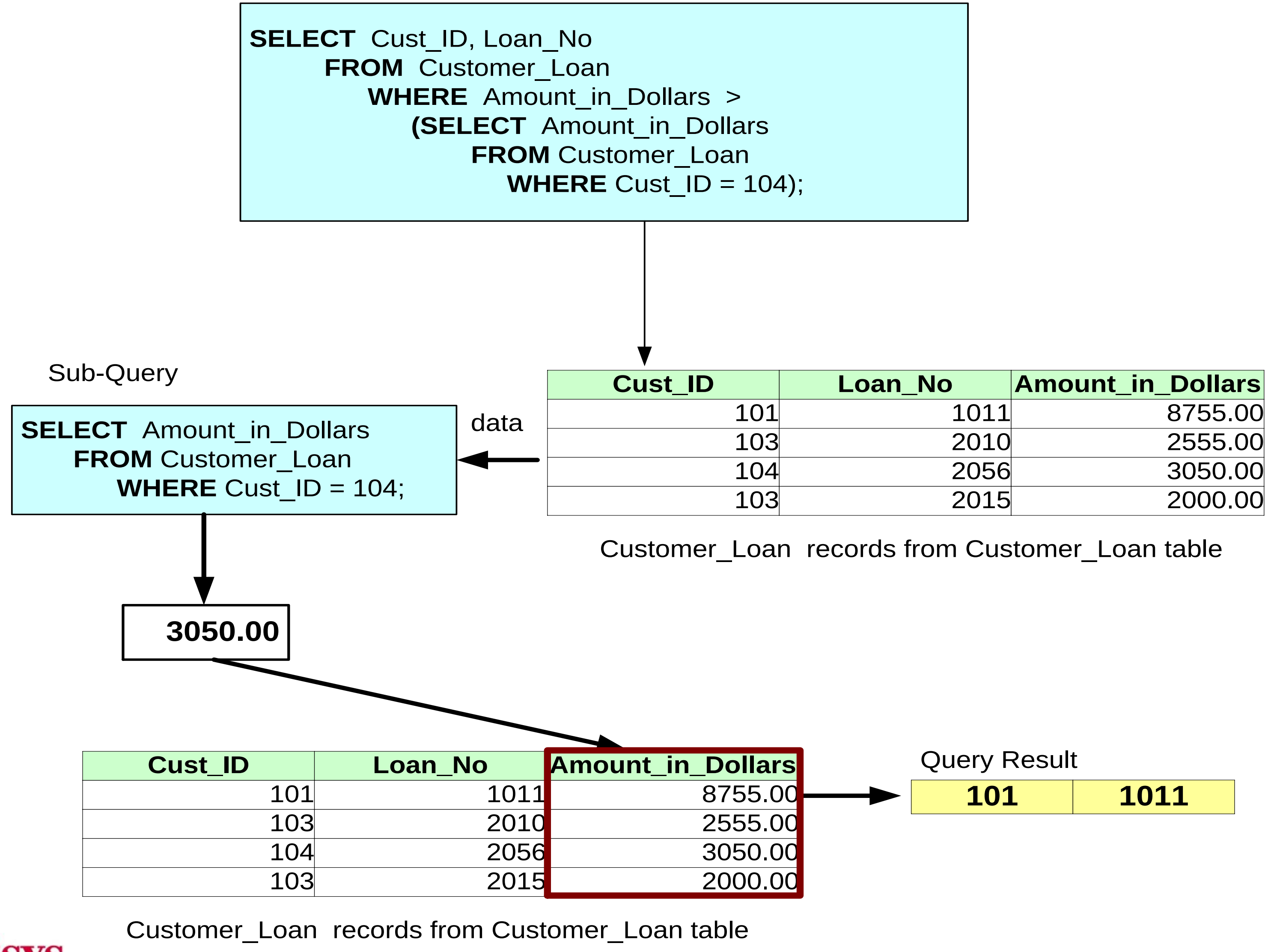
- Outer query then runs on the stored results.

Retrieval using SUB QUERIES

To list the Cust_ID and Loan_No for all Customers who have taken a loan of amount greater than the loan amount of Customer (Cust_ID = 104).

```
Select cust_ID, Loan_no
          From Customer_Loan
                    Where amount_in_dollars  >
                              (Select amount_in_dollars
                                        From Customer_Loan
                                                  Where Cust_ID = 104);
```

Sub Query (Contd…)



```
SELECT  Cust_ID, Loan_No
        FROM  Customer_Loan
             WHERE  Amount_in_Dollars  >
                  (SELECT  Amount_in_Dollars
                         FROM Customer_Loan
                             WHERE Cust_ID = 104);
```

Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Customer_Loan records from Customer_Loan table

data

Sub-Query

```
SELECT  Amount_in_Dollars
      FROM Customer_Loan
           WHERE Cust_ID = 104;
```

3050.00

Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Customer_Loan records from Customer_Loan table

Query Result

101	1011

UNISYS
imagine it. done.

Retrieval using SUB QUERIES

List customer names of all customers who have taken a loan > $3000.00.

```
SELECT Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
       FROM Customer_Details
               WHERE Cust_ID
                      IN
                      ( SELECT Cust_ID
                              FROM Customer_Loan
                                 WHERE Amount_in_Dollars > 3000.00);
```

Retrieval using SUB QUERIES

List customer names of all customers who have the same Account_type as Customer ‘Jones Simon’ .

```
SELECT Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
            FROM Customer_Details
                        WHERE Account_Type
                                    =
                                    ( SELECT Account_Type
                                         FROM Customer_Details
                                              WHERE Cust_Last_Name = ‘Jones’
                                              AND Cust_First_Name = ‘Simon’);
```

Retrieval using SUB QUERIES

List customer names of all customers who do not have a Fixed Deposit.

```
SELECT Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
          FROM Customer_Details
                    WHERE Cust_ID
                              NOT IN
                                        ( SELECT Cust_ID
                                               FROM Customer_Fixed_Deposit );
```

Retrieval using SUB QUERIES

List customer names of all customers who have either a Fixed Deposit or a loan but not both at any of Bank Branches. The list includes customers who have no fixed deposit and loan at any of the bank branches.

```
SELECT Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
        FROM Customer_Details
                WHERE Cust_ID
                        NOT IN
                        ( SELECT Cust_ID
                            FROM Customer_Loan
                                WHERE Cust_ID
                                        IN
                                   (SELECT Cust_ID
                                      FROM Customer_Fixed_Deposit ));
```

Correlated Sub Queries

- You can refer to the table in the FROM clause of the outer query in the inner query using Correlated sub-queries.

- The inner query is executed separately for each row of the outer query.

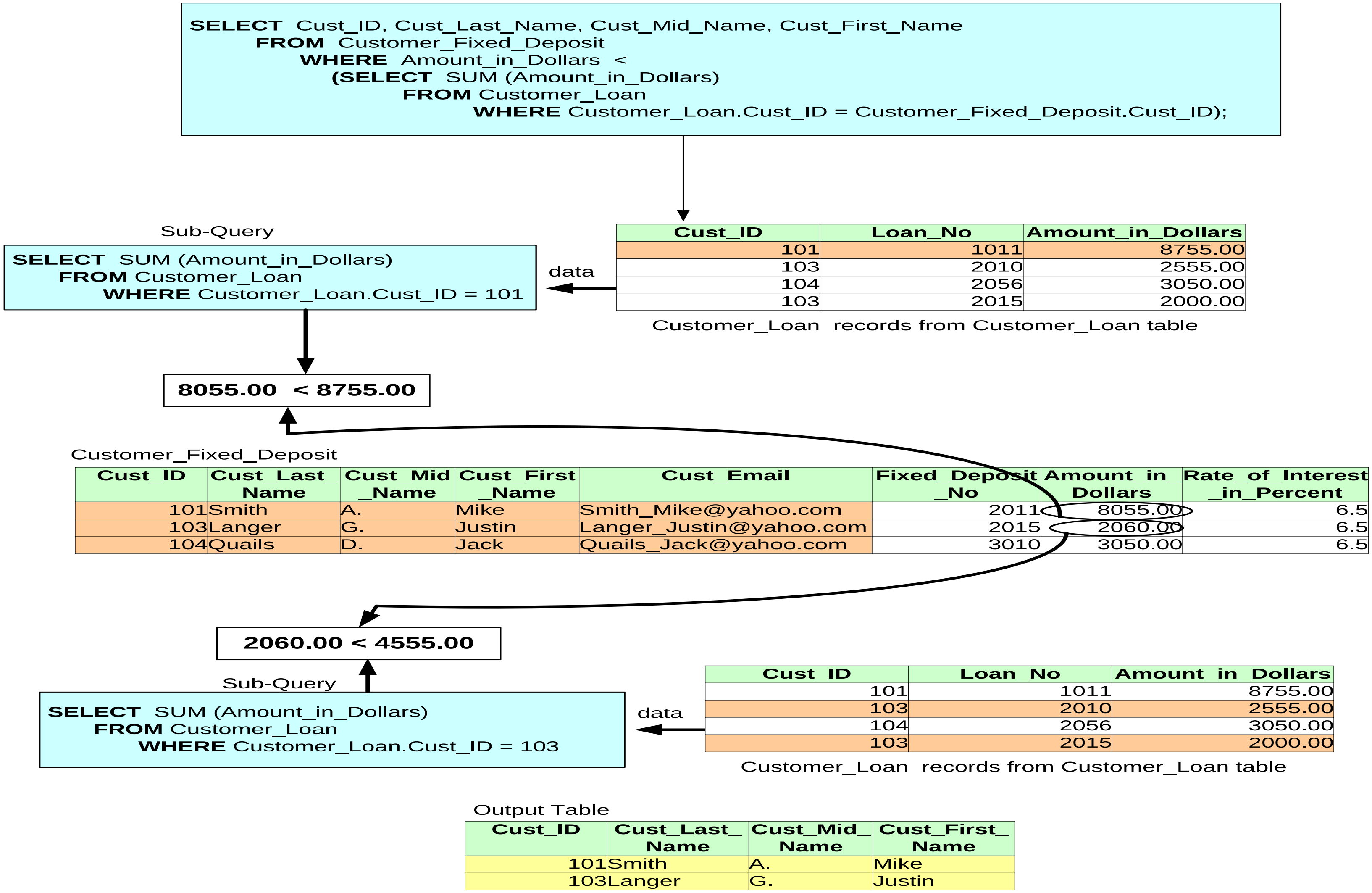
  (i.e. In Co-Related Sub-queries, SQL performs a sub-query over and over again – once for each row of the main query.)

Correlated Sub Queries

To list all Customers who have a fixed deposit of amount less than the sum of all their loans.

```
Select Cust_Id, Cust_Last_Name, cust_Mid_Name, cust_First_Name
From Customer_fixed_Deposit
Where amount_in_dollars
         <
        (Select sum(amount_in_dollars)
        From Customer_Loan
        Where Customer_Loan.Cust_Id = Customer_Fixed_Deposit.Cust_ID);
```

Correlated Sub Queries (Contd…)



```
SELECT  Cust_ID, Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
        FROM  Customer_Fixed_Deposit
             WHERE  Amount_in_Dollars  <
                 (SELECT  SUM (Amount_in_Dollars)
                          FROM Customer_Loan
                                   WHERE Customer_Loan.Cust_ID = Customer_Fixed_Deposit.Cust_ID);
```

Sub-Query

```
SELECT  SUM (Amount_in_Dollars)
     FROM Customer_Loan
           WHERE Customer_Loan.Cust_ID = 101
```

Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Customer_Loan records from Customer_Loan table

Customer_Fixed_Deposit

Cust_ID	Cust_Last_ Name	Cust_Mid _Name	Cust_First _Name	Cust_Email	Fixed_Deposit _No	Amount_in_ Dollars	Rate_of_Interest _in_Percent
101	Smith	A.	Mike	Smith_Mike@yahoo.com	2011	8055.00	6.5
103	Langer	G.	Justin	Langer_Justin@yahoo.com	2015	2060.00	6.5
104	Quails	D.	Jack	Quails_Jack@yahoo.com	3010	3050.00	6.5

Sub-Query

```
SELECT  SUM (Amount_in_Dollars)
     FROM Customer_Loan
           WHERE Customer_Loan.Cust_ID = 103
```

Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Customer_Loan records from Customer_Loan table

Output Table

Cust_ID	Cust_Last_ Name	Cust_Mid_ Name	Cust_First_ Name
101	Smith	A.	Mike
103	Langer	G.	Justin

Correlated Sub Queries

List customer IDs of all customers who have both a Fixed Deposit and a loan at any of Bank Branches

```
SELECT Cust_ID
FROM Customer_Details
WHERE Cust_ID
    IN
   (SELECT Cust_ID
    FROM Customer_Loan
    WHERE Customer_Loan.Cust_ID  = Customer_Details.Cust_ID)
    AND Cust_ID IN
        (SELECT Cust_ID
         FROM Customer_Fixed_Deposit
       WHERE Customer_Fixed_Deposit.Cust_ID  = Customer_Details.Cust_ID);
```

Correlated Sub Queries ...

Get S# for suppliers supplying some project with P1 in a quantity greater than the average qty of P1 supplied to that project

```
SELECT  DISTINCT  S#
          FROM  Shipments  X
                    WHERE  P# = ‘P1’ AND  QTY >
                              (SELECT  AVG(QTY)
                                       FROM  Shipments Y
                                            WHERE  P# = ‘P1’ AND  X.J# = Y.J#)
```

Correlated Sub Queries

Get P# for all parts supplied by more than one supplier

```
SELECT  P#
          FROM  Shipment  X
                    WHERE  P#  IN
                              (SELECT P#
                                        FROM  Shipment Y
                                                  WHERE  Y.S#  <> X.S#)
```

Exists versus Not Exists

Retrieval using EXISTS

List all Customers who have at least one Fixed Deposit more than $3000.00.

```
SELECT Cust_ID, Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
      FROM Customer_Details S
           WHERE EXISTS
    (SELECT *
         FROM Customer_Fixed_Deposit O
               WHERE O.Amount_in_Dollars > 3000.00 AND O.Cust_ID = S.Cust_ID);
```

Retrieval using EXISTS

List all Customers who have both a Fixed Deposit and a Loan at the Bank

```
SELECT Cust_ID
FROM Customer_Fixed_Deposit
WHERE EXISTS
    (SELECT *
    FROM Customer_Loan
    WHERE Customer_Loan.Cust_ID = Customer_Fixed_Deposit.Cust_ID);
```

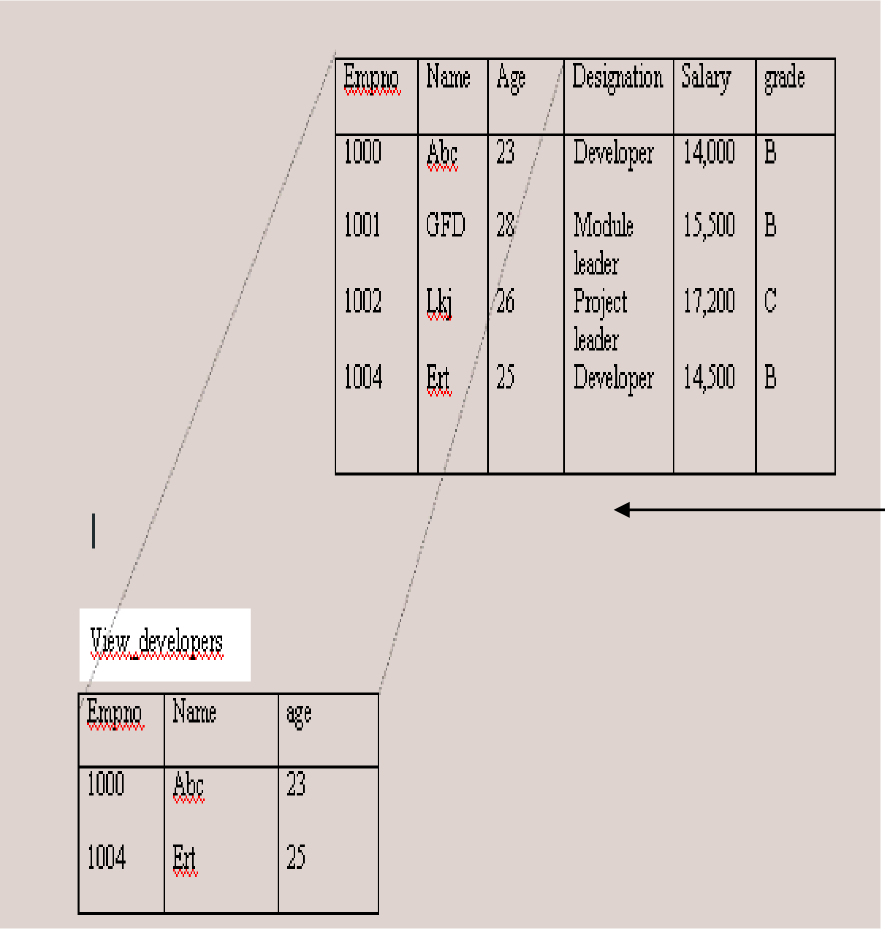
Retrieval using NOT EXISTS

List all Customers who don’t have a single Fixed Deposit over $3000.00.

```
SELECT Cust_ID, Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
FROM Customer_Details S
WHERE NOT EXISTS
    (SELECT *
     FROM Customer_Fixed_Deposit O
     WHERE O.Amount_in_Dollars > 3000.00 AND O.Cust_ID = S.Cust_ID);
```

Views

What is a view?



Empno	Name	Age	Designation	Salary	grade
1000	Abc	23	Developer	14,000	B
1001	GFD	28	Module leader	15,500	B
1002	Lkj	26	Project leader	17,200	C
1004	Ert	25	Developer	14,500	B

View_developers

Empno	Name	age
1000	Abc	23
1004	Ert	25

- A view is a kind of "virtual table"
- Contents are defined by a query like:

  Select Empno, Name, age

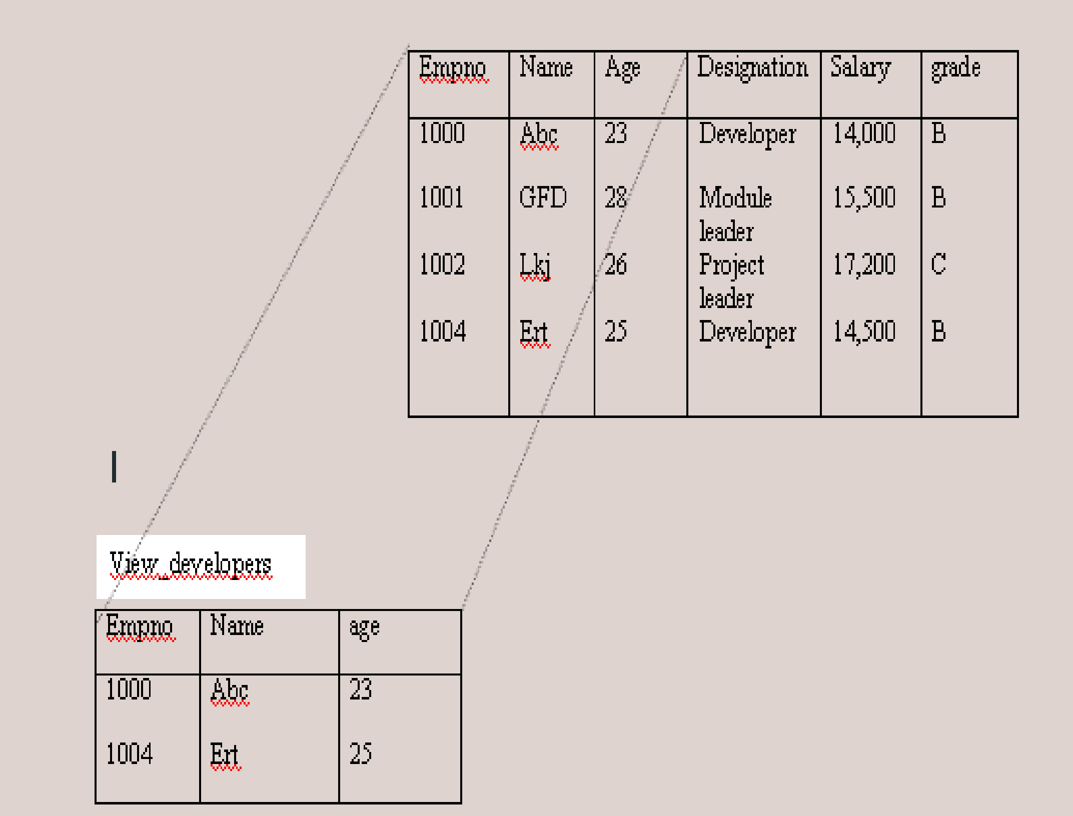
  from Employee

  Where designation='developer';

  As shown in the figure

What is a view to the DBMS

- We can use views in select statements like
- **Select** * **from** view_employees **where** age > 23;
- DBMS translates the request to an equivalent request to the source table



Empno	Name	Age	Designation	Salary	grade
1000	Abc	23	Developer	14,000	B
1001	GFD	28	Module leader	15,500	B
1002	Lkj	26	Project leader	17,200	C
1004	Ert	25	Developer	14,500	B

View_developers

Empno	Name	age
1000	Abc	23
1004	Ert	25

Create a view

```
CREATE  VIEW   view-name  column-name1, column-name2, ---------------  AS  query
```

```
CREATE VIEW ViewCustomerDetails

   AS

            SELECT  *

                        FROM Customer_Details;
```

Assigning names to columns

```
Create view vwCustDetails (CustCode,CustLname,CustFName)
   As  Select Cust_Id,Cust_Last_Name,Cust_First_Name
           From Customer_details;
```

Types of views

- Horizontal views
- Vertical views
- Row/column subset views
- Grouped views
- Joined views

Horizontal views

Horizontal view restricts a user’s access to only selected rows of a table.

```
CREATE VIEW view_cust AS

        SELECT   *

                 FROM Customer_Details

                          WHERE  Cust_ID in (101,102,103);
```

Vertical views

- A view which selects only few columns of a table:
- Vertical view restricts a user's access to only certain columns of a table

```
CREATE VIEW view_cust AS

        SELECT Cust_ID, Account_No, Account_Type

                FROM Customer_Details;
```

Row/column subset views

```
CREATE VIEW View_Cust_VertHor
          AS  SELECT  Cust_Id,Account_No,Account_Type
                    FROM Customer_Details
                               WHERE CUST_ID IN (101,102,103);
```

Views with Group By clause

- The query contains a group by clause

```
Create view View_GroupBY(Dept,NoofEmp)
            As Select Department, count(Employee_ID)
                        FROM Employee_Manager
                                    GROUP BY Department;
```

Views with Joins

- Created by specifying a two-table or three-table query in the view creation command

```
Create view View_Cust_Join
 as  select a.Cust_Id,b.Cust_First_Name,b.Cust_Last_Name,Amount_in_dollars
 from Customer_loan a, customer_details b
 where a.cust_id = b.cust_id;
```

Updating a VIEW

A view can be modified by the DML command.

```
CREATE VIEW View_Cust
          AS   SELECT  *
                    FROM Customer_Details
                              WHERE CUST_ID IN (101,102,103);


--Insert Statement
insert into view_cust values(103,'Langer','G.','Justin',3421,'Savings',' Global
Commerce Bank','Langer_Justin@Yahoo.com');


--Delete Statement
delete view_cust  where cust_id=103;


--Update Statament
Update view_cust set Cust_last_name = 'Smyth‘ where cust_id=101;
```

Updating View

A view can be updated if the query that defines the view meets all of these restrictions:

- DISTINCT must not be specified; that is, duplicate rows must not be eliminated from the query results
- The FROM clause must specify only one updateable table; the view must have a single underlying source table
- The SELECT list cannot contain expressions, calculated columns, or column functions
- The WHERE clause must not include a sub query; only simple row-by-row search conditions may appear

Dropping Views

Views are dropped similar to the way in which the tables are dropped. However, you must own a view in order to drop it.

```
DROP VIEW  <view name>;

DROP VIEW View_Cust;
```

Checking View Updates – Check Option

```
CREATE  VIEW  view_customer AS
        SELECT Cust_ID, Cust_Last_Name, Account_No, Account_Type, Bank_Branch
                FROM Customer_Details
                        WHERE Bank_Branch = ‘Downtown’ ;
```

```
                INSERT INTO view_customer
        VALUES (115, ’Costner’, 107, ‘Savings’, ‘Bridgewater’);
```

Will it prevent insertion into Customer_details ?

```
SELECT  Cust_ID, Cust_Last_Name, Bank_Branch

                  FROM view_customer;
```

Solution is :

```
CREATE  VIEW  view_customer AS

        SELECT Cust_ID, Cust_Last_Name, Account_No, Account_Type, Bank_Branch

                        FROM Customer_Details

                                WHERE Bank_Branch = ‘Downtown’

                                        With CHECK OPTION;
```

Advantages of views

- Security
- Query simplicity
- Structural simplicity

Disadvantages of views

- Performance
- Restrictions

SQL – Data Control Language

GRANT …. Tables or views

```
GRANT {
            [ALTER[, ]]
            [DELETE[, ]]
            [INDEX[, ]]
            [INSERT[, ]]
            [SELECT[, ]]
            [UPDATE [(column-name[,...])][, ]]
            | ALL [PRIVILEGES]]     }
ON [TABLE] {table-name[,...] | view-name[,...]}
TO [AuthID][,...]
[WITH GRANT OPTION]
```

GRANT

```
GRANT SELECT, INSERT
            ON Customer_Details
                        TO Edwin ;
GRANT ALL PRIVILEGES
            ON Customer_Loan
                        TO JACK ;
GRANT ALL
            ON Customer_Loan
                        TO PUBLIC ;
```

GRANT

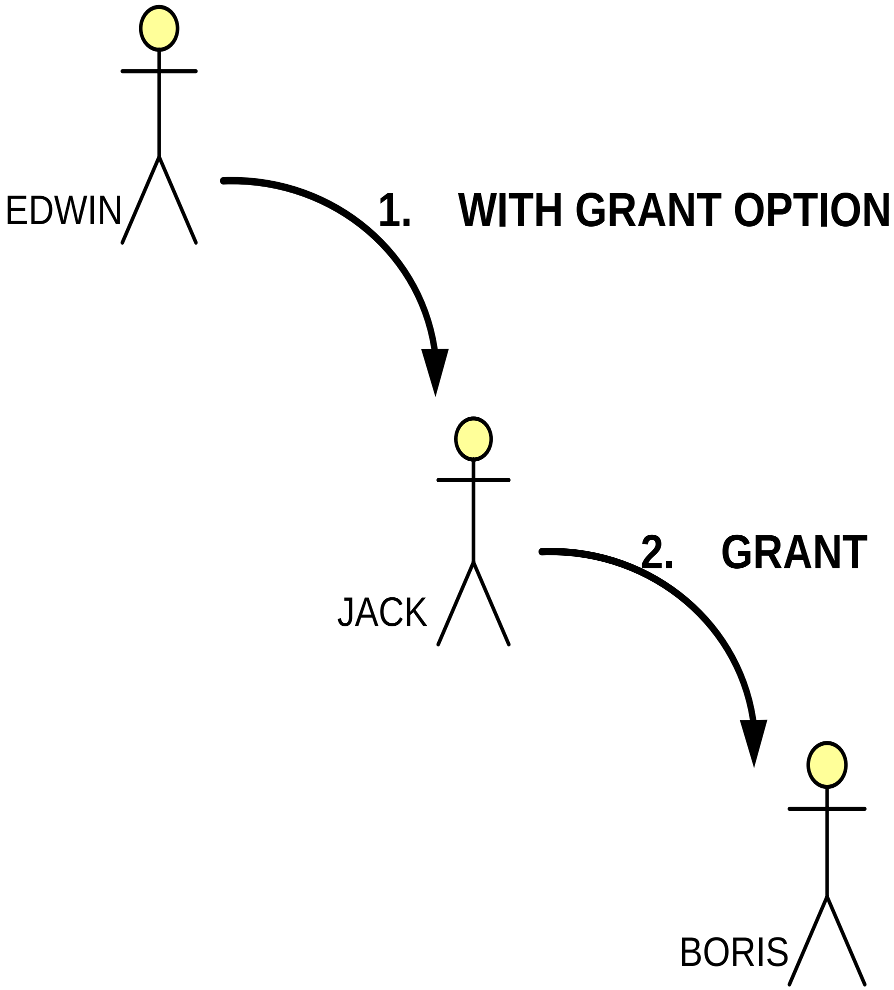
- With Grant Option

GRANT SELECT

ON Customer_Loan

TO EDWIN

With GRANT OPTION;



Taking PRIVILIGES away

The syntax of REVOKE command is patterned after GRANT, but with a reverse meaning.

```
REVOKE{
            [ALTER[, ]]
            [DELETE[, ]]
            [INDEX[, ]]
            [INSERT[, ]]
            [SELECT[, ]]
            [UPDATE [(column-name[,...])][, ]]
            |  ALL [PRIVILEGES] }
ON [TABLE] {table-name[,...] | view-name [,...]}
FROM AuthID[,...]
```

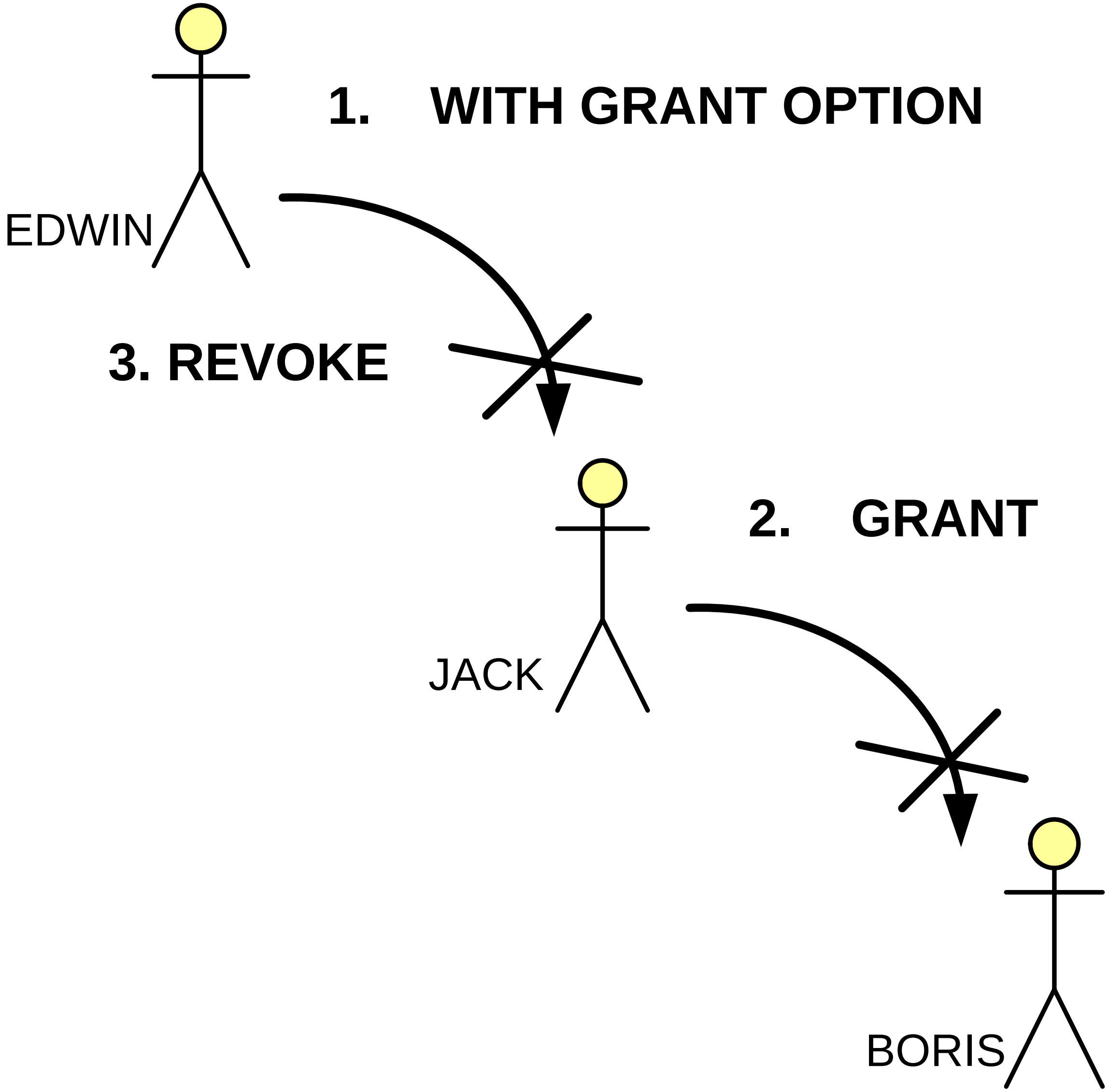
Revoke

```
REVOKE SELECT, INSERT
   ON Customer_Details
            FROM Edwin ;

REVOKE ALL PRIVILEGES
   ON Customer_Loan
            FROM JACK ;

REVOKE ALL
   ON Customer_Loan
            FROM PUBLIC ;
```

Revoke



Summary

- When the query consists of more than one component, it is implemented in the form of a nested query depending on the nature of the query
- Sub queries help split a problem involving different levels of data
- Relational algebra operations like union, intersect, difference, restriction, projection and join help us get different combinations of data from more than one table
- Views create a window into the table thereby allowing restricted access
- Grant statement is used to grant access privileges on database objects like table, view etc.
- Revoke statement is used to take back access privileges on database objects like table, view etc.

Thank you

UNISYS
imagine it. done.