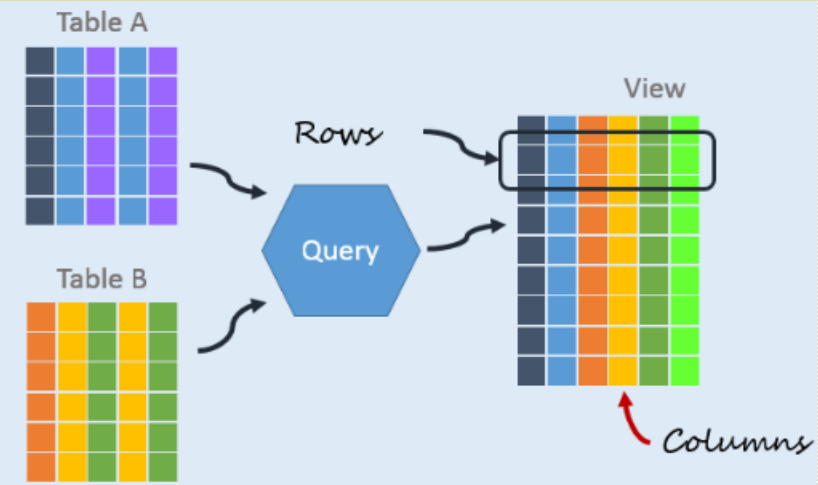


Views, Null Values, and Hierarchical Queries

date	fixed
01-02-2016	(NULL)
01-02-2016	(NULL)
01-03-2016	02-02-2016
01-03-2016	02-03-2016
01-04-2016	(NULL)
01-05-2016	02-04-2016
01-05-2016	(NULL)
01-07-2016	02-05-2016

Outline

- Creating and using views
- Processing queries that reference views
- Null value handling
- Hierarchical queries

Views

- Views are fundamental to application development:
- Purpose of writing queries is to specify data for applications (forms, reports, batch processing)
- Simplification of tasks is the most important benefit of views.
- View: a stored query with intelligent processing
- Simplification of tasks is the most important benefit of views.

View?

- Derived table
  - Not just a subset of base tables: specify with a query
  - Query can be complex and even contain row summaries, not individual rows

View behavior:

  - Use a view like a base table
  - Use a view in a query
  - Use a view in modification statements
  - Some restrictions especially on usage in modification statements

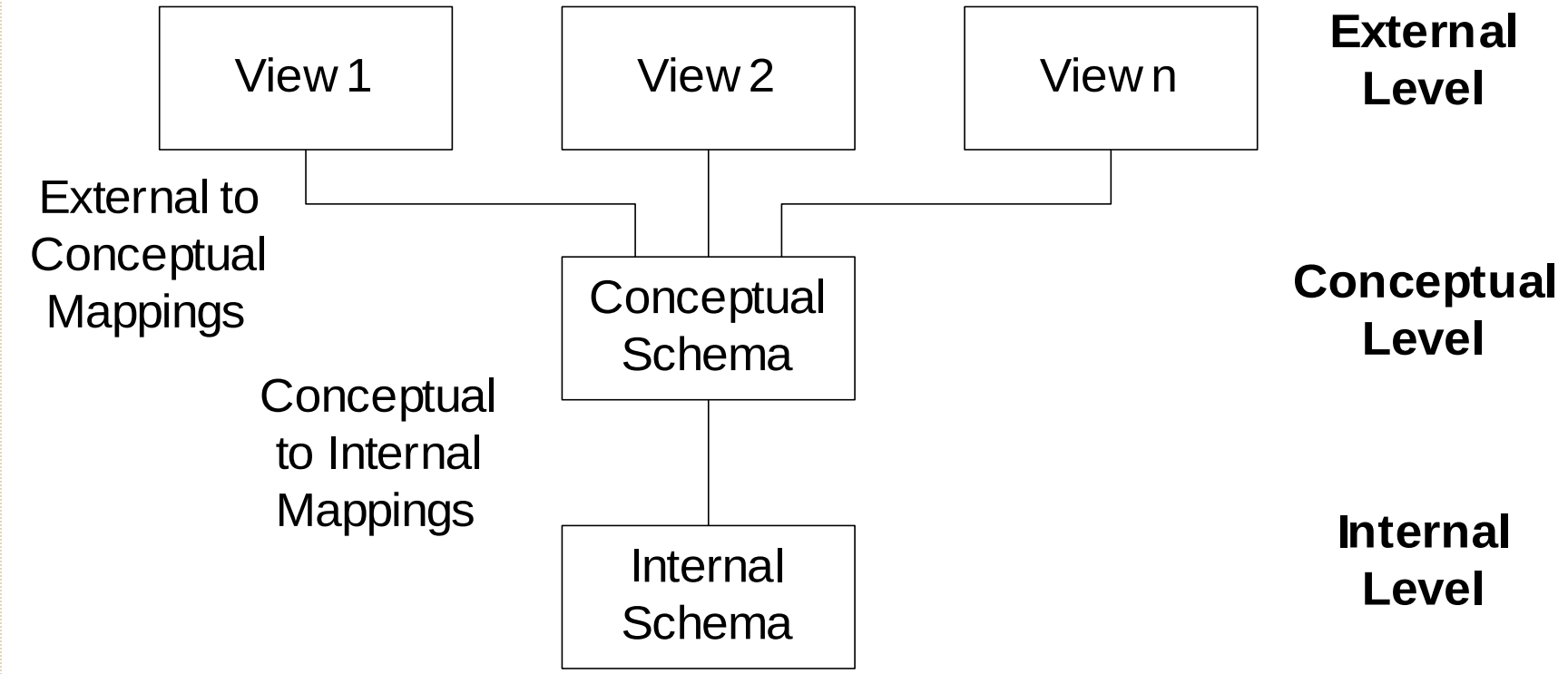
Virtual

  - Seems to be real but is not
  - Performance may suffer: must buffer in anticipation of memory needs (delay otherwise)
  - Performance of views must be close to base tables: otherwise not used

View Advantages

- Reduce impact of database definition changes
- Simplify database usage
- Unit of database security
- Can be a performance penalty on complex views

Three Schema Architecture



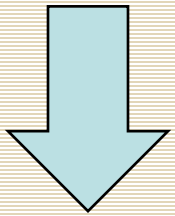
View Definition Example

Defining a view is only slightly more difficult than writing a SELECT statement. SQL provides the CREATE VIEW statement in which a view name and a SELECT statement must be specified,

students majoring in IS

```
CREATE VIEW IS_View AS
    SELECT *
        FROM Student
        WHERE StdMajor = 'IS';
```

Display the rows of the view after creating the view.



StdNo	StdFirst Name	StdLast Name	Std City	Std State	Std Zip	Std Major	Std Class	Std GPA
123-45-6789	HOMER	WELLS	SEATTLE	WA	98121-1111	IS	FR	3.00
345-67-8901	WALLY	KENDALL	SEATTLE	WA	98123-1141	IS	SR	2.80
567-89-0123	MARIAH	DODGE	SEATTLE	WA	98114-0021	IS	JR	3.60

View Definition Example

Defining a view is only slightly more difficult than writing a SELECT statement. SQL provides the CREATE VIEW statement in which a view name and a SELECT statement must be specified,

Create a view consisting of offerings taught by faculty in the MS department.

```
CREATE or REPLACE VIEW MS_View
AS
 SELECT OfferNo Oferta, Course.CourseNo, CrsUnits,
       OffTerm, OffYear, FacFirstName, FacLastName, OffTime, OffDays
  FROM Faculty, Course, Offering
  WHERE FacDept = 'MS'
    AND Faculty.FacNo = Offering.FacNo
    AND Offering.CourseNo = Course.CourseNo;
```

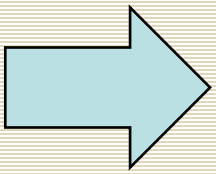
Column Renaming I

The column list is required in Example to rename the aggregate calculation (COUNT(*)) column

Define a multiple table view
create a view containing offering data and the number of enrolled students.

```
CREATE VIEW  Enrollment_View
 ( OfferNo, CourseNo, Term, Year, Instructor,  NumStudents )
AS
 SELECT Offering.OfferNo, CourseNo, OffTerm, OffYear, FacLastName, COUNT(*)
  FROM Offering, Faculty, Enrollment
  WHERE Offering.FacNo  =  Faculty.FacNo
    AND Offering.OfferNo  =  Enrollment.OfferNo
  GROUP BY Offering.OfferNo, CourseNo, OffTerm,
         OffYear, FacFirstName, FacLastName;
```

Display rows of the view
after creating the view.



OfferNo	CourseNo	Term	Year	Instructor	NumStudents
1234	IS320	FALL	2019	VINCE	6
7777	FIN480	SPRING	2020	MACON	3

Column Renaming II

Alternatively, renaming can be selectively done in the SELECT statement as shown in Example

create a view containing offering data and the number of enrolled students.

```
CREATE VIEW  Enrollment_View1
AS
  SELECT Offering.OfferNo, CourseNo, OffTerm, OffYear,
         FacLastName AS Instructor, COUNT(*) AS NumStudents
   FROM Offering, Faculty, Enrollment
   WHERE Offering.FacNo  =  Faculty.FacNo
    AND Offering.OfferNo  =  Enrollment.OfferNo
   GROUP BY Offering.OfferNo, CourseNo, OffTerm,
        OffYear, FacFirstName, FacLastName;
```

Using Views

After creating a view, you can use it in SELECT statements. You simply use the view name in the FROM clause and the view columns in other parts of the statement. You can add other conditions and select a subset of columns

Query using a multiple table view
List the spring 2020 courses in MS_View.

```
SELECT OfferNo, CourseNo, FacFirstName,
       FacLastName, OffTime, OffDays
  FROM MS_View
  WHERE OffTerm = 'SPRING' AND OffYear = 2020;
```

OfferNo	CourseNo	FacFirstName	FacLastName	OffTime	OffDays
9876	IS460	LEONARD	FIBON	13:30 :00	TTH
3333	IS320	LEONARD	VINCE	08:30:00	MW
5679	IS480	CRISTOPHER	COLAN	15:30:00	TTH

Using Views

After creating a view, you can use it in SELECT statements. You simply use the view name in the FROM clause and the view columns in other parts of the statement. You can add other conditions and select a subset of columns

Query using a grouping view
List the spring 2020 offerings of IS courses

```
SELECT OfferNo, CourseNo, Instructor,
NumStudents
  FROM Enrollment_View
  WHERE Term = 'SPRING'
    AND Year = 2020
    AND CourseNo LIKE 'IS%';
```

OfferNo	CourseNo	Instructor	NumStudents
5679	IS480	COLAN	6
9876	IS460	FIBON	7

Using Views

Grouping query using a view derived from a grouping query

List the average number of students by instructor name using Enrollment_View.

```
SELECT Instructor, AVG(NumStudents) AS AvgStdCount
  FROM Enrollment_View
  GROUP BY Instructor;
```

Instructor	AvgStdCount
MACON	2.50
COLAN	6.00
MILLS	3.50
VINCE	6.00
FIBON	7.00

Using Views

Joining a **base table with a view** derived from a grouping query

List the offering number, instructor, number of students, and course units using the Enrollment_View view and the Course table.

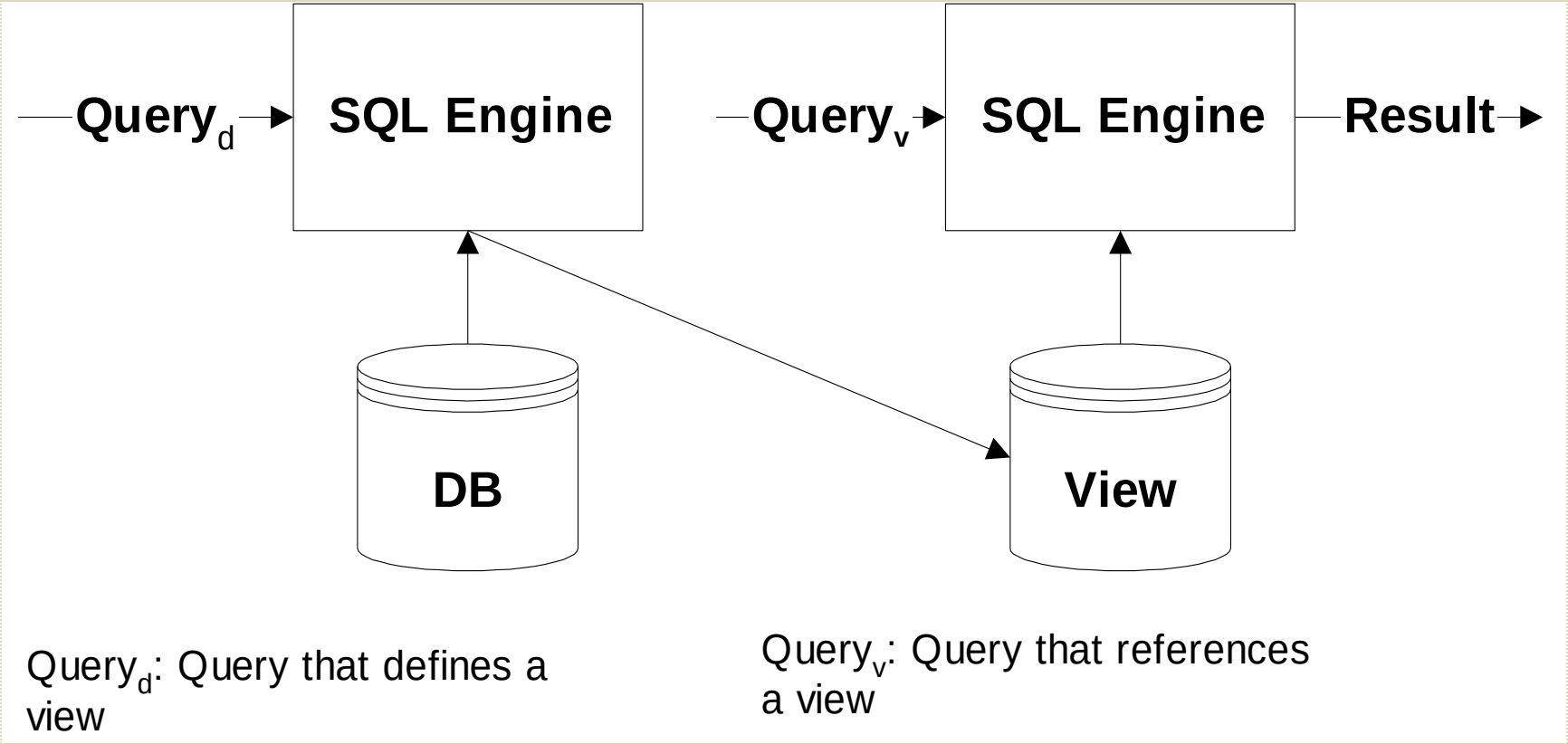
```
SELECT OfferNo, Instructor, NumStudents,
       CrsUnits
  FROM Enrollment_View, Course
  WHERE Enrollment_View.CourseNo = Course.CourseNo
    AND NumStudents < 5;
```

OfferNo	Instructor	NumStudents	CrsUnits
7777	MACON	3	4
5555	MACON	2	4
6666	MILLS	2	4

Processing View Queries

- Materialization
  - a method to process a query on a view by executing the query directly on the stored view. The stored view can be materialized on demand (when the view query is submitted) or periodically rebuilt from base tables.
  - **For databases with a mixture of retrieval and update activity, materialization usually is not an efficient way to process a query on a view.**

View Materialization

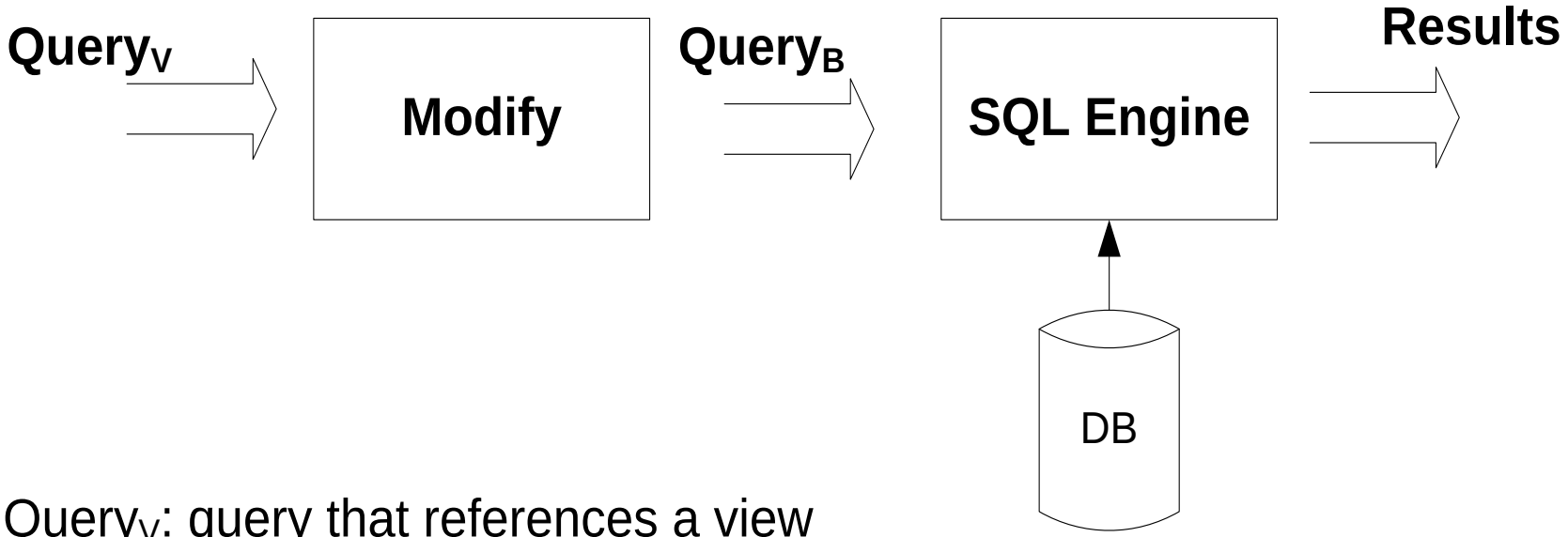


Processing View Queries

Modification

- a method to process a query on a view involving the execution of only one query. A query using a view is translated into a query using base tables by replacing references to the view with its definition.
- **For databases with a mixture of retrieval and update activity, modification provides an efficient way to process a query on a view.**

View Modification



Query$_V$: query that references a view

Query$_B$: modification of Query$_V$ such that references to the view are replaced by references to base tables.

Modification Examples

Query using a view

```
SELECT OfferNo, CourseNo, FacLastName
 FROM MS_View
 WHERE OffYear = 2020;
```

Modified query

```
SELECT OfferNo, Course.CourseNo,
       FacLastName
 FROM Faculty, Course, Offering
 WHERE FacDept = 'MS' AND OffYear = 2020
   AND Faculty.FacNo = Offering.FacNo
   AND Offering.CourseNo = Course.CourseNo;
```

Updatable Views

- Read-only views used in SELECT statements only
- Updatable views also used in modification statements (INSERT, UPDATE, DELETE)
- Simplification of data requirements for data entry forms
- Rules for updatable views, both single table and multiple table

Rules for Single Table Updatable Views

- 1-1 correspondence between view rows and base table rows
- View includes PK of base table
- The view contains all required columns (NOT NULL) of the base table.
- View definition does not have GROUP BY, DISTINCT, or traditional set operators (UNION, INTERSECT, MINUS/EXCEPT).

Updatable View Examples

Create a row and column subset view with the primary key. Display view rows after creating the view.

Single table updatable view

```
CREATE VIEW Fac_View1 AS
 SELECT FacNo, FacFirstName, FacLastName,
        FacRank, FacSalary, FacDept,
        FacCity, FacState, FacZipCode
  FROM Faculty
  WHERE FacDept = 'MS';
```

Fac_View1 is updatable assuming the missing Faculty columns are not required, it can be used in INSERT, UPDATE, and DELETE statements to change the Faculty table. Note that modifications to views are subject to the integrity rules of the underlying base tables.

Updatable View Examples

Give a 10 percent raise to the new row in Fac_View1.

Insert a new faculty row into the MS department.

```
INSERT INTO Fac_View1
  (FacNo, FacFirstName, FacLastName, FacRank,
FacSalary,  FacDept, FacCity, FacState, FacZipCode)
  VALUES ('999-99-8888', 'Joe', 'Smith', 'Prof',
80000,'MS', 'SEATTLE', 'WA', '98011-0011');
```

View update

```
 UPDATE Fac_View1
   SET FacSalary = FacSalary * 1.1
   WHERE FacNo = '999-99-8888';
```

Delete operation on an updatable view

```
DELETE FROM Fac_View1
  WHERE FacNo = ‘999-99-8888’;
```

Read-Only View Examples

Single Table read-only view

Create a row and column subset without the primary key.
Display view rows after creation.

```
CREATE VIEW Fac_View2 AS
    SELECT FacDept, FacRank, FacSalary
        FROM Faculty
        WHERE FacSalary > 50000;
```

View Update with Side Effects

- Some modifications to updatable views can be problematic
- Modify column used in the WHERE condition of a view definition

View update

```
UPDATE Fac_View1
  SET FacDept = 'FIN'
  WHERE FacNo = '999-99-8888';
```

FACNO	FACFIRSTNAME	FACLASTNAME	FACRANK	FACSALARY	FACDEPT	FACCITY	FACSTATE	FACZIPCODE
654-32-1098	LEONARD	FIBON	ASSC	70000	MS	SEATTLE	WA	98121-0094
098-76-5432	LEONARD	VINCE	ASST	35000	MS	SEATTLE	WA	98111-9921
876-54-3210	CRISTOPHER	COLAN	ASST	40000	MS	SEATTLE	WA	98114-1332

View Update with Side Effects

- Use WITH CHECK OPTION clause to prevent

```
CREATE VIEW Fac_View1_Revised AS
 SELECT FacNo, FacFirstName, FacLastName,
        FacRank, FacSalary, FacDept,
        FacCity, FacState, FacZipCode
  FROM Faculty
  WHERE FacDept = 'MS'
 WITH CHECK OPTION;
```

Because this side effect can be confusing to a user, the WITH CHECK OPTION clause can be used to prevent updates with side effects. If the WITH CHECK OPTION is specified in the CREATE VIEW statement, INSERT or UPDATE statements that do not satisfy the WHERE clause are rejected.

```
ORA-01402: violación de la cláusula WHERE en la vista WITH CHECK OPTION
Error at Line: 7 Column: 0
```

Multiple Table Updatable Views

- No industry standard
- More complex rules than single table updatable views
- Limited support in Oracle
- Supplement with triggers for more complex updatable views

Updatable Join Views in Oracle

- One updatable table
- Key preserving table
  - Updates supported for only one key preserving table
- Other restrictions
  - does not contain the WITH CHECK OPTION
  - Single table updatability restrictions
- Write INSTEAD OF triggers to support more operations on multiple table join views

- Updatable join views use the concept of a key preserving table a table in a 1-1 relationship with a view.
- A join view preserves a table if every candidate key of the table is a candidate key of the view.
- This statement means that the rows of an updatable join view can be mapped in a 1-1 manner with each key preserved table.
- In a join involving a 1-M relationship, the child table can be key preserved because each child row is associated with at most one view row.

- An updatable join view supports insert, update, and delete operations on one underlying table per manipulation statement.
- The updatable table is the key preserving table.
- An UPDATE statement can modify (in the SET clause) only columns of one key preserving table.
- An INSERT statement can add a row in the key preserving table if the view includes all required columns without default values.
- An INSERT statement cannot contain columns from nonkey preserving tables.
- Rows can be deleted if the join view contains only one key preserving table.

Updatable View

```
CREATE VIEW Course_Offering_View AS
 SELECT Course.CourseNo AS CCourseNo, CrsDesc,
        CrsUnits, OfferNo, OffTerm, OffYear,
        OffTime, Offering.CourseNo AS OCourseNo,
        OffLocation, FacNo, OffDays
  FROM Course INNER JOIN Offering
        ON Course.CourseNo = Offering.CourseNo;
```

- Key preserving table: *Offering*
- Supports insert, update, and delete operations on *Offering* columns in the view
- Does not support operations on the *Course* table

- The Course-Offering-View supports a limited set of operations that map to operations on the key preserving table.

- The following list summarizes operations supported by the Course-Offering-View:

- Insert operations using all Offering columns result in new Offering rows.
- Insert operations using Course columns are rejected as illegal operations.
- Update operations involving Offering columns change the columns in the associated Offering rows.
- Delete operations remove corresponding Offering rows.

Operations on Course_Offering_View

demonstrates allowable manipulation operations on the Course_Offering_View.

```
INSERT INTO Course_Offering_View
 (OfferNo, OffTerm, OffYear,
  OffLocation,OffTime,FacNo, OffDays, OCourseNo )
 VALUES
  (9977,'FALL',2020,'BLM410','13:30:00',NULL,'MW','IS4
  60');

UPDATE Course_Offering_View
  SET OffYear = 2021
  WHERE OfferNo = 9977;

DELETE FROM Course_Offering_View
  WHERE OfferNo = 9977;
```

Null Value Effects

- Simple conditions
- Compound conditions
- Grouping and aggregate functions
- SQL standard but implementation may vary

Club Table with Null Values

ClubNo	CName	CPurpose	CBudget	CActual
C1	DELTA	SOCIAL	1000.00	1,200.00
C2	BITS	ACADEMIC	500.00	350.00
C3	HELPS	SERVICE	300.00	330.00
C4	SIGMA	SOCIAL	[null]	150.00

Simple Conditions

- Simple condition is null if either left-hand or right-hand side is null.
- Discard rows evaluating to false or null
- Retain rows evaluating to true
- Rows evaluating to null will not appear in the result of the simple condition or its negation

Simple condition using a column with null values

List the clubs with a budget greater than 200. The result omits the club with a null budget (C4) because the condition evaluates as a null value.

ClubNo	CName	CPurpose	CBudget	CActual
C1	DELTA	SOCIAL	1000.00	1,200.00
C2	BITS	ACADEMIC	500.00	350.00
C3	HELPS	SERVICE	300.00	330.00
C4	SIGMA	SOCIAL	[null]	150.00

```
SELECT *
   FROM Club
   WHERE CBudget > 200;
```

Simple condition involving two columns and null values

List clubs with a budget greater than the actual spending. The result omits the club with a null budget (C4) because the condition evaluates to null.

ClubNo	CName	CPurpose	CBudget	CActual
C1	DELTA	SOCIAL	1000.00	1,200.00
C2	BITS	ACADEMIC	500.00	350.00
C3	HELPS	SERVICE	300.00	330.00
C4	SIGMA	SOCIAL	[null]	150.00

```
SELECT *
   FROM Club
   WHERE CBudget > CActual;
```

```
SELECT *
   FROM Club
   WHERE CBudget <= CActual;
```

Compound Conditions

AND	True	False	Null
True	True	False	Null
False	False	False	False
Null	Null	False	Null

OR	True	False	Null
True	True	True	True
False	True	False	Null
Null	True	Null	Null

NOT	True	False	Null
	False	True	Null

A row is selected only if the entire compound condition in the WHERE clause evaluates to true, not false or null.

Evaluation of a compound OR condition with a null value

List the clubs with a budget less than or equal to the actual spending or the actual spending less than 200. The result contains the club with a null budget (C4) because the second condition evaluates to true.

ClubNo	CName	CPurpose	CBudget	CActual
C1	DELTA	SOCIAL	1000.00	1,200.00
C2	BITS	ACADEMIC	500.00	350.00
C3	HELPS	SERVICE	300.00	330.00
C4	SIGMA	SOCIAL	[null]	150.00

```
SELECT *
   FROM Club
   WHERE CBudget <= CActual
         OR CActual < 200;
```

Effect on IN and NOT IN

- Involve implicit OR operations
  - CBudget IN (300,500) to CBudget = 300 OR CBudget = 500

ClubNo	CName	CPurpose	CBudget	CActual
C1	DELTA	SOCIAL	1000.00	1,200.00
C2	BITS	ACADEMIC	500.00	350.00
C3	HELPS	SERVICE	300.00	330.00
C4	SIGMA	SOCIAL	[null]	150.00

Effect on IN and NOT IN

- CBudget NOT IN (300,500) to NOT (CBudget = 300 OR CBudget = 500)

ClubNo	CName	CPurpose	CBudget	CActual
C1	DELTA	SOCIAL	1000.00	1,200.00
C2	BITS	ACADEMIC	500.00	350.00
C3	HELPS	SERVICE	300.00	330.00
C4	SIGMA	SOCIAL	[null]	150.00

Aggregate Functions

- Calculations with aggregate functions ignore null values
- Effects can be subtle
  - COUNT(*) may differ from Count(Column)
  - SUM(Column1) + SUM(Column2) may differ from SUM(Column1 + Column2)

Aggregate Functions

```
SELECT COUNT(*) AS NumRows, COUNT(CBudget) AS NumBudgets
FROM Club;
```

NumRows	NumBudgets
4	3

```
SELECT SUM(CBudget) AS SumBudget,
                    SUM(CActual) AS SumActual,
                    SUM(CBudget)-SUM(CActual) AS SumDifference,
                    SUM(CBudget-CActual) AS SumOfDifferences
  FROM Club;
```

SumBudget	SumActual	SumDifference	SumOfDifferences
1800.00	2030.00	-230.00	-80.00

Difference Problem Effect

Difference problem with NOT NULL condition

```
SELECT FacFirstName, FacLastName, FacDept
 FROM Faculty
 WHERE Faculty.FacNo NOT IN
  ( SELECT FacNo
     FROM Offering
     WHERE OffTerm = 'SUMMER'
       AND FacNo IS NOT NULL ) ;
```

Grouping Effects

- Rows with null values are grouped together
- Grouping column contains null values
- Null group can be placed at beginning or end of the non-null groups

```
SELECT FacNo, COUNT(*) AS NumRows
   FROM Offering
   GROUP BY FacNo;
```

FacNo	NumRows
[null]	2
876-54-3210	1
543-21-0987	1
765-43-2109	2
654-32-1098	2
098-76-5432	3
987-65-4321	2

Hierarchical Queries

- Background
- Hierarchical data
- Proprietary Oracle notation
- SQL standard notation

Background

- Involve self-referencing relationships
- Child row related to at most one parent row
  - Organization chart
  - Part explosion diagram
  - Chart of accounts
- Finding details or summarizing features of employees managed directly or indirectly
- Hierarchical queries not common but important in some business situations

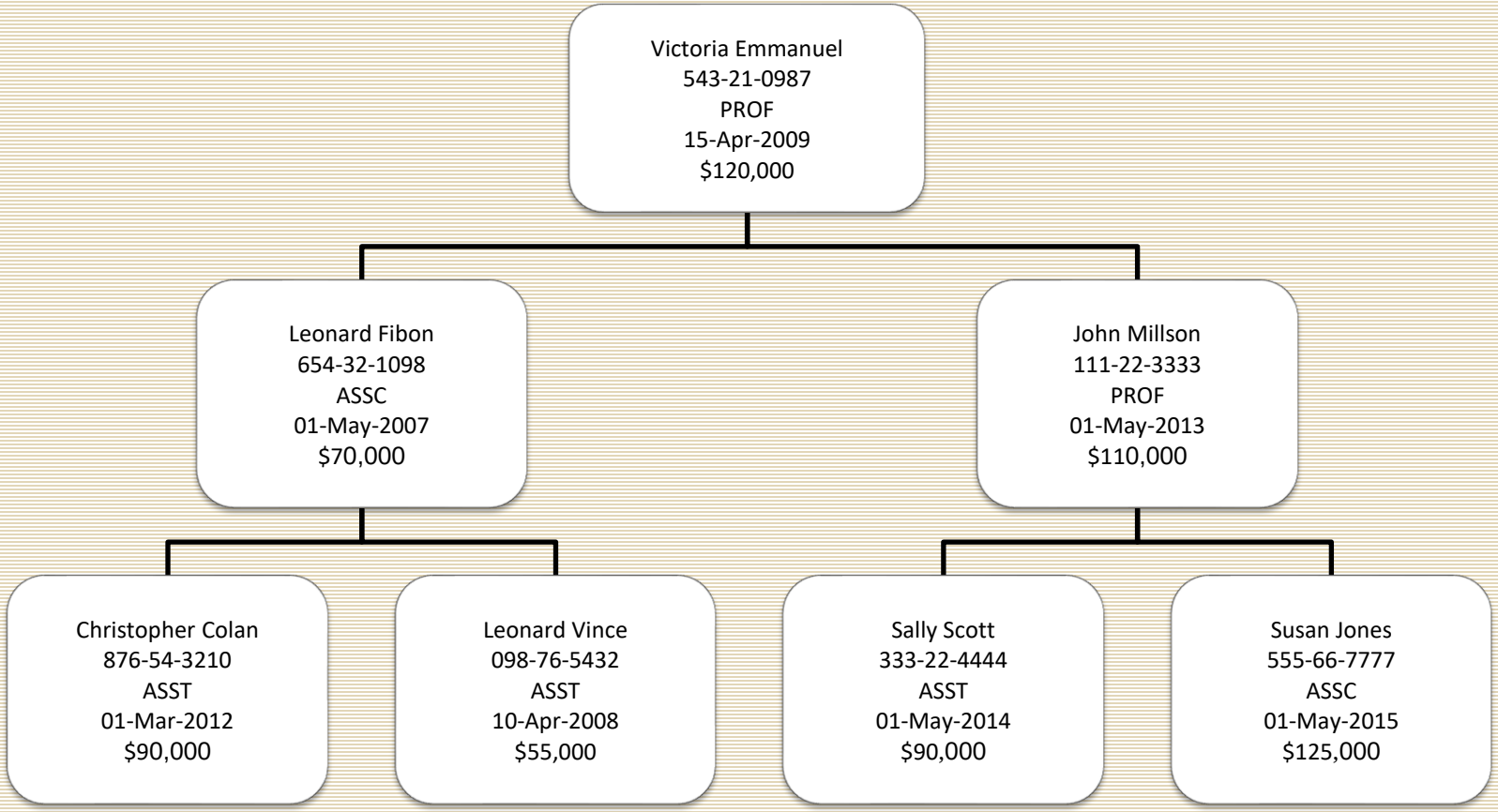
Importance of SQL Extensions

- Self-joins not sufficient due to variable number of self-joins
- Write stored procedure with looping for variable number of joins
- Higher productivity with SQL extension
- SQL compiler optimization with SQL extensions
- Oracle developed a proprietary extension of the SELECT statement using the CONNECT BY PRIOR clause. Standard SQL provides an extension to the SELECT statement involving the WITH clause and recursive common table expressions.

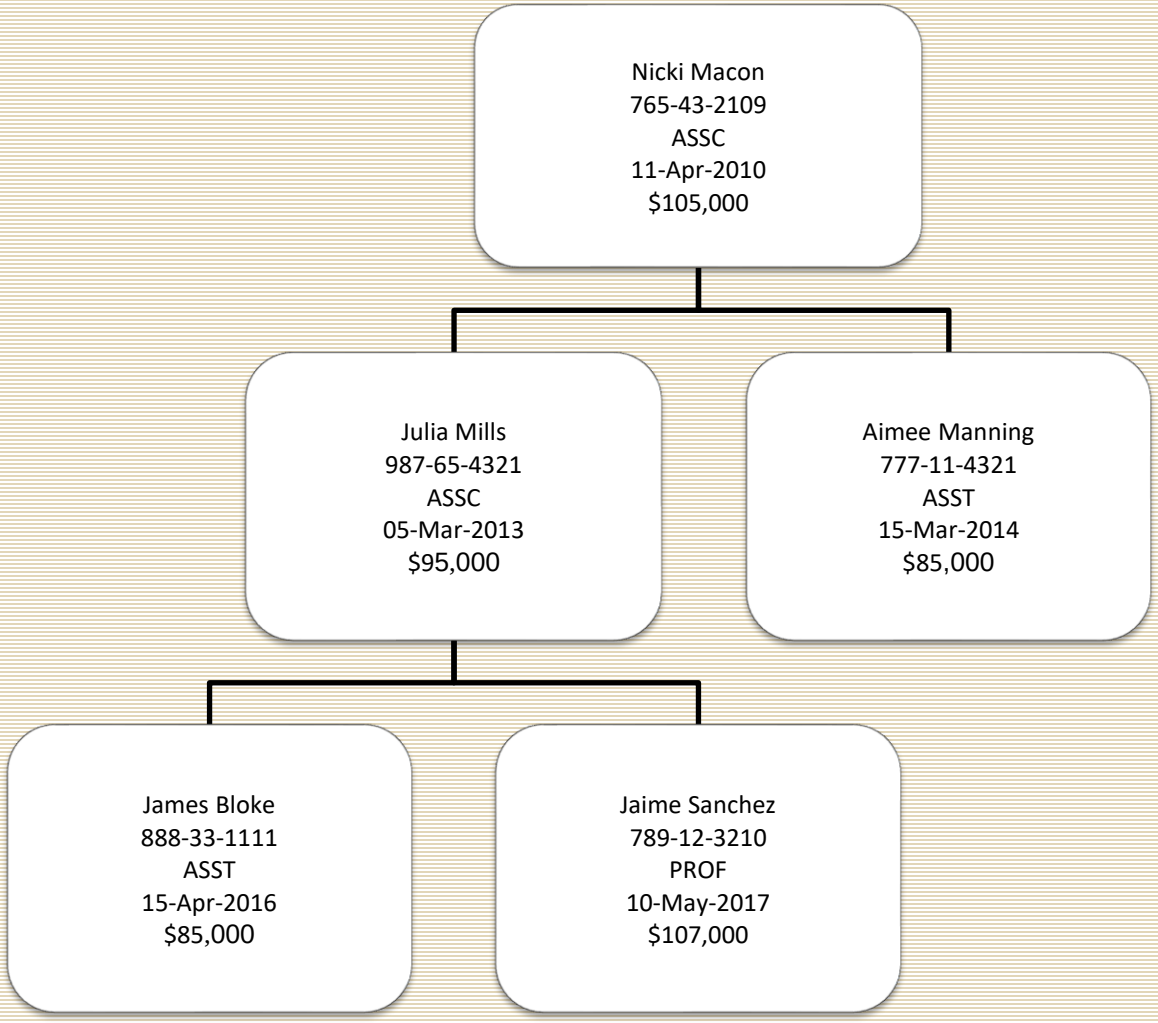
Hierarchical Data Example

FacNo	FacSupervisor	FacFirstName	FacLastName	FacHireDate	FacRank	FacSalary
098-76-5432	654-32-1098	LEONARD	VINCE	10-Apr-2008	ASST	55000
543-21-0987	[null]	VICTORIA	EMMANUEL	15-Apr-2009	PROF	120000
654-32-1098	543-21-0987	LEONARD	FIBON	01-May-2007	ASSC	70000
765-43-2109	[null]	NICKI	MACON	11-Apr-2010	ASSC	105000
876-54-3210	654-32-1098	CRISTOPHER	COLAN	01-Mar-2012	ASST	90000
987-65-4321	765-43-2109	JULIA	MILLS	15-Mar-2013	ASSC	95000
111-22-3333	543-21-0987	JOHN	MILLSON	01-May-2013	PROF	110000
333-22-4444	111-22-3333	SALLY	SCOTT	01-May-2014	ASST	90000
555-66-7777	111-22-3333	SUSAN	JONES	01-May-2015	ASSC	125000
777-11-4321	765-43-2109	AIMEE	MANNING	15-Mar-2014	ASST	85000
888-33-1111	987-65-4321	JAMES	BLOKE	15-Apr-2016	ASST	85000
789-12-3210	987-65-4321	JAIME	SANCHEZ	10-May-2017	PROF	107000

Hierarchical Data Example I



Hierarchical Data Example II



Oracle Proprietary Notation

- CONNECT BY PRIOR clause
- START WITH clause
- Pseudo columns: LEVEL, CONNECT_BY_LEAF
- Operators: CONNECT_BY_ROOT, PRIOR
- SIBLINGS keyword
- Functions: SYS_CONNECT_BY_PATH

Basic Hierarchical Query

- Visit each row one time on a path
- Use basic syntax elements

```
SELECT FacNo, FacSupervisor, FacFirstName,
       FacLastName, FacHireDate, FacSalary,
       FacRank, LEVEL
  FROM Faculty2
  CONNECT BY PRIOR FacNo = FacSupervisor
  ORDER BY FacNo, LEVEL;
```

The CONNECT BY PRIOR clause contains a condition relating parent and child rows, typically the self-join condition.

For example, the row with COLAN appears three times in the result because it resides on level 3

START WITH Clause

- Start with root rows
- Use null value condition on FK

```
SELECT FacNo, FacSupervisor, FacFirstName,
       FacLastName, FacHireDate, FacSalary,
       FacRank, LEVEL
  FROM Faculty2
  START WITH FacSupervisor IS NULL
  CONNECT BY PRIOR FacNo = FacSupervisor
  ORDER BY LEVEL;
```

the rows with null values for FacSupervisor are designated as the starting rows. The START WITH clause eliminates duplicate rows in the result caused by treating each row as a root.

SIBLINGS Keyword

- Rows with the same parent
- Specify sort order for siblings
- LPAD function adds spaces on the left.

```
SELECT FacNo, FacLastName, FacHireDate,
 LPAD(' ',2*(LEVEL-1)) || FacLastName AS LastName,
 FacSalary, FacRank, LEVEL
  FROM Faculty2
  START WITH FacSupervisor IS NULL
  CONNECT BY PRIOR FacNo = FacSupervisor
  ORDER SIBLINGS BY FacLastName;
```

SYS_CONNECT_BY_PATH Function

- Column name parameter
- Separator character
- Alternative to LPAD function

```
SELECT SYS_CONNECT_BY_PATH(FacLastName,'/') AS Path,
 FacHireDate, FacSalary, FacRank, LEVEL
  FROM Faculty2
  START WITH FacSupervisor IS NULL
  CONNECT BY PRIOR FacNo = FacSupervisor
  ORDER SIBLINGS BY FacLastName;
```

SYS_CONNECT_BY_PATH function to show the complete path with a column name and a separator character as parameters.

CONNECT_BY_ROOT Operator

- Reference root row
- Column name parameter

```
SELECT SYS_CONNECT_BY_PATH(FacLastName,'/') AS Path,
 CONNECT_BY_ROOT FacLastName AS Root,
 CONNECT_BY_ISLEAF AS IsLeaf,
 FacHireDate, FacSalary, FacRank, LEVEL
  FROM Faculty2
  START WITH FacSupervisor IS NULL
  CONNECT BY PRIOR FacNo = FacSupervisor
  ORDER SIBLINGS BY FacLastName;
```

The CONNECT_BY_ROOT operator retrieves a column value from a root row. The CONNECT_BY_ISLEAF pseudo column provides a row's leaf status. A row is a leaf if it has no children.

Summary Totals

- CONNECT_BY_ROOT cannot be used in WHERE clause

```
SELECT Root, COUNT(*)-1 AS NumSubordinates,
       SUM(FacSalary) AS FacSalarySum
  FROM
   (  SELECT CONNECT_BY_ROOT FacLastName AS Root,
             FacSalary
     FROM Faculty2
     CONNECT BY PRIOR FacNo = FacSupervisor )
  GROUP BY Root
  ORDER BY COUNT(*) DESC;
```

The CONNECT_BY_ROOT operator can be used indirectly for grouping so that summary totals can be calculated for paths in a hierarchy. To calculate summary totals, the root should not be restricted by the START WITH clause.

SQL Standard Notation

- Supported by major DBMSs including Oracle and PostgreSQL
- Part of standard since SQL:1999
- Recursive common table expressions (CTE)
- More verbose but fewer language elements than Oracle notation

Recursive CTE Template

- CTE definition follows WITH keyword
- Two query blocks connected by UNION
- SELECT statement uses CTE

```
WITH CTEName <ColumnList> -- for Oracle
-- WITH RECURSIVE CTEName <ColumnList> for PostgreSQL
AS
-- Anchor member (AM) referencing the hierarchical table.
 ( <CTEQuery1>
UNION ALL
-- Recursive member (RM) referencing the CTEName.
    <CTEQuery2> )
-- Statement using the CTE
SELECT * FROM CTEName;
```

Recursive CTE Example I

- Path exception query for faculty earning more than a supervisor

```
WITH Faculty2CTE ( FacNo, FacSupNo, FacLastName,
  FacSupLastName, FacSalary, FacSupSalary )
AS
( SELECT F1.FacNo, F1.FacSupervisor, F1.FacLastName,
   F1Sup.FacLastName, F1.FacSalary, F1Sup.FacSalary
  FROM Faculty2 F1 INNER JOIN Faculty2 F1Sup
    ON F1.FacSupervisor = F1Sup.FacNo
UNION ALL
  SELECT F2.FacNo, F2CTE.FacSupNo, F2.FacLastName,
   F2CTE.FacSupLastName, F2.FacSalary, F2CTE.FacSupSalary
  FROM Faculty2 F2 INNER JOIN Faculty2CTE F2CTE
    ON F2.FacSupervisor = F2CTE.FacNo )
-- Statement using the CTE
SELECT *
 FROM Faculty2CTE
 WHERE FacSupNo <> FacNo AND FacSalary > FacSupSalary;
```

Recursive CTE Example II

- Path exception query for faculty with a higher rank than a supervisor

```
WITH Faculty2CTE ( FacNo, FacSupNo, FacLastName,
  FacSupLastName, FacRank, FacSupRank)
AS
( SELECT F1.FacNo, F1.FacSupervisor, F1.FacLastName,
    F1Sup.FacLastName, F1.FacRank, F1Sup.FacRank
  FROM Faculty2 F1 INNER JOIN Faculty2 F1Sup
    ON F1.FacSupervisor = F1Sup.FacNo
UNION ALL
  SELECT F2.FacNo, F2CTE.FacSupNo, F2.FacLastName,
    F2CTE.FacSupLastName, F2.FacRank, F2CTE.FacSupRank
  FROM Faculty2 F2 INNER JOIN Faculty2CTE F2CTE
    ON F2.FacSupervisor = F2CTE.FacNo )
-- Statement using the CTE
SELECT *
 FROM Faculty2CTE
 WHERE FacSupNo <> FacNo
  AND ( ( FacRank = 'PROF' AND FacSupRank = 'ASSC' )
   OR ( FacRank = 'PROF' AND FacSupRank = 'ASST' )
   OR ( FacRank = 'ASSC' AND FacSupRank = 'ASST' ) );
```

Summary

- Extension of query formulation skills
- Simplification and unit of security for views
- Modest performance penalty for views
- Subtle but important effects of null values on conditions, logical expressions, and aggregate functions
- Specialized but important usage of hierarchical data and queries