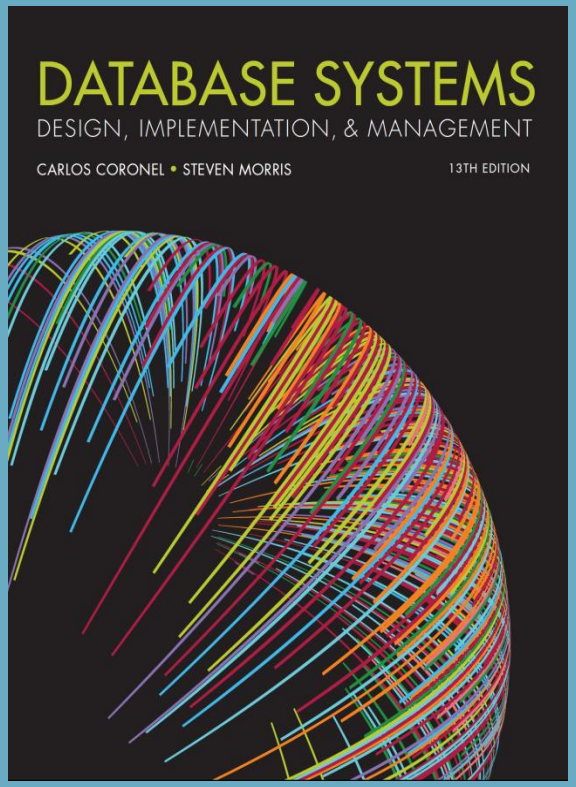


Chapter 3

Advanced SQL

Learning Objectives

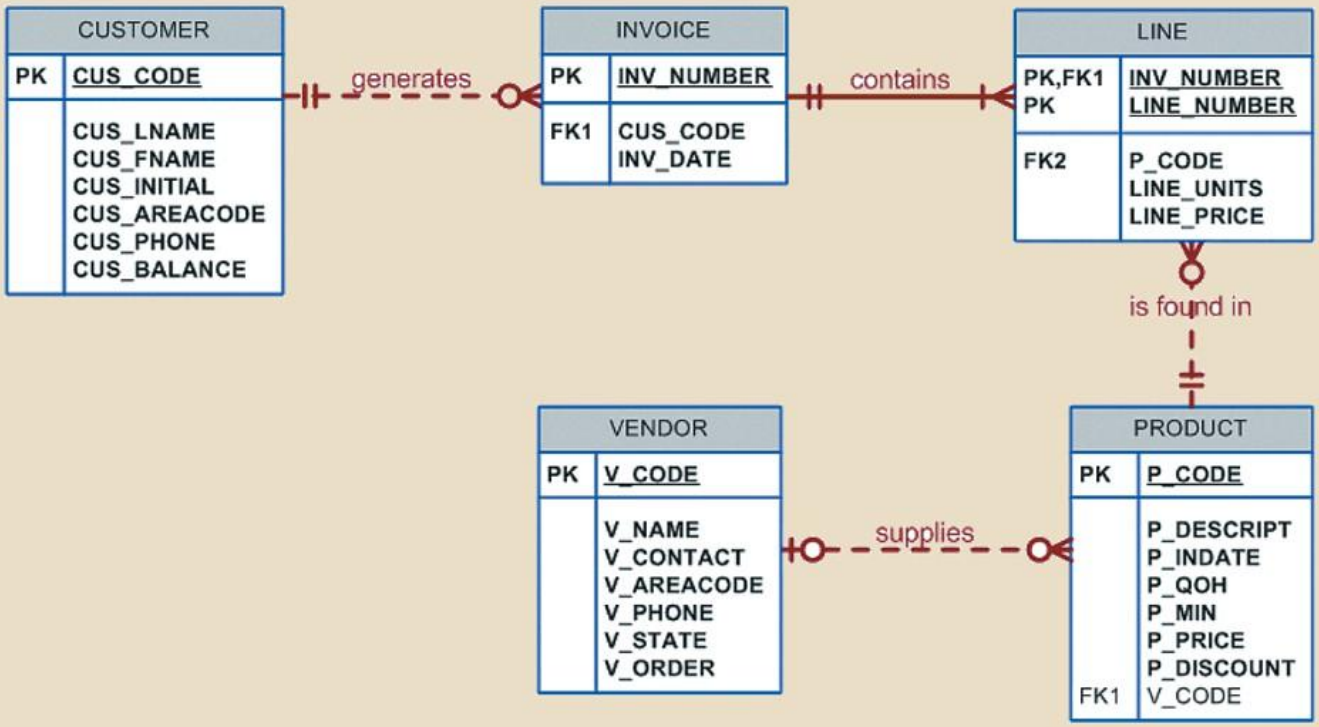
- After completing this chapter, you will be able to:
  - Use SQL to create a table manually
  - Use SQL to create a copy of a table using a subquery
  - Manipulate the structure of existing tables to add, modify, and remove columns and constraints
  - Use SQL to do data manipulation (insert, update, and delete rows of data)
  - Use SQL to create database views, including updatable views
  - Use Procedural Language SQL (PL/SQL) to create triggers, stored procedures, and PL/SQL functions
  - Create embedded SQL

Data Definition Commands (1 of 3)

- Starting database model
  - Refer to Figure 8.1
- Creating the database
  - Before a new RDBMS can be used, the database structure and the tables that will hold the end-user data must be created
- The database schema
  - Logical group of database objects—such as tables and indexes—that are related to each other
- Data types
  - Character, numeric, and date

Data Definition Commands (2 of 3)

FIGURE 8.1 DATABASE MODEL



Data Definition Commands (3 of 3)

TABLE 8.1: Some Common SQL Data Types		
Data Type	**Format**	**Comments**
Numeric	NUMBER(L,D) or NUMERIC(L,D)	The declaration NUMBER(7,2) or NUMERIC(7,2) indicates that numbers will be stored with two decimal places and may be up to seven digits long, including the sign and the decimal place (for example, 12.32 or −134.99).
	INTEGER	May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.
	SMALLINT	Like INTEGER but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.
	DECIMAL(L,D)	Like the NUMBER specification, but the storage length is a minimum specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.
Character	CHAR(L)	Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as Smith and Katzenjammer are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes.
	VARCHAR(L) or VARCHAR2(L)	Variable-length character data. T he designation VARCHAR2(25) or VARCHAR(25) will let you store characters up to 25 characters long. However, unlike CHAR, VARCHAR will not leave unused spaces. Oracle automatically converts VARCHAR to VARCHAR2.
Date	DATE	Stores dates in the Julian date format.

Creating Table Structures (1 of 2)

- CREATE TABLE command

```
CREATE TABLE tablename (
column1          data type          [constraint] [,
column2          data type          [constraint] ] [,
PRIMARY KEY      (column1           [, column2]) ] [,
FOREIGN KEY      (column1           [, column2]) REFERENCES tablename] [,
CONSTRAINT       constraint ] );
```

- SQL constraints
  - FOREIGN KEY
  - NOT NULL
  - UNIQUE
  - DEFAULT
  - CHECK

Creating Table Structures (2 of 2)

- Create a table with a SELECT statement
  - Rapidly creates a new table based on selected columns and rows of an existing table using a subquery
  - Automatically copies all of the data rows returned
- SQL indexes
  - CREATE INDEX improves the efficiency of searches and avoids duplicate column values
  - DROP Index deletes an index

Altering Table Structures (1 of 3)

- All changes in the table structure are made by using the ALTER TABLE command followed by a keyword that produces the specific change you want to make
  - ADD, MODIFY, and DROP
- Changing a column's data type
  - ALTER
- Changing a column's data characteristics
  - If the column to be changed already contains data, you can make changes in the column's characteristics if those changes do not alter the data type
- Adding a column
  - You can alter an existing table by adding one or more columns
  - Be careful not to include the NOT NULL clause for the new column

Altering Table Structures (2 of 3)

- Adding primary key, foreign key, and check constraints
  - Primary key syntax:

```
ALTER TABLE    PART
ADD            PRIMARY KEY (PART_CODE);
```

  - Foreign key syntax:

```
ALTER TABLE    PART
ADD            FOREIGN KEY (V_CODE) REFERENCES
               VENDOR;
```

  - Check constraint syntax:

```
ALTER TABLE    PART
ADD            CHECK (PART_PRICE >= 0);
```

Altering Table Structures (3 of 3)

- Dropping a column
  - Syntax:

```
ALTER TABLE VENDOR
   DROP COLUMN V_ORDER;
```

- Deleting a table from the database
  - Syntax:

```
DROP TABLE PART;
```

Data Manipulation Commands (1 of 2)

- Adding table rows
  - INSERT command syntax:

    INSERT INTO *tablename* VALUES (*value1, value2, …, valuen*)
  - Inserting rows with null attributes: use NULL entry
  - Inserting rows with optional attributes: indicate attributes that have required values
- Inserting table rows with a SELECT subquery
  - Add multiple rows to a table, using another table as the source, at the same time
  - SELECT syntax:

    INSERT INTO *target_tablename*[(*target_columnlist*)]
    SELECT *source_columnlist*
    FROM *source_tablename*;
- Saving table changes
  - COMMIT command syntax:

    COMMIT [WORK]

Data Manipulation Commands (2 of 2)

- Updating table rows
  - UPDATE command is used to modify data in a table
  - UPDATE syntax:

```
UPDATE      tablename
SET         columnname = expression [, columnname = expression]
[WHERE      conditionlist ];
```

- Deleting table rows
  - DELETE statement syntax:

```
DELETE FROM   tablename
[WHERE        conditionlist ];
```

- Restoring table contents
  - ROLLBACK command is used restore the database to its previous condition

```
ROLLBACK;
```

Virtual Tables: Creating a **View**

- View: virtual table based on a SELECT query
  - Base tables: tables on which the view is based
  - ***Dynamic result*** of one or more relational operations operating on ***base relations*** to produce another relation.
  - ***Virtual relation*** that **does not necessarily actually exist** in the database but is produced upon request, at time of request.
  - Contents of a view are defined as a ***query*** on one or more base relations.
- With ***view resolution***, any operations on view are automatically translated into operations on relations from which it is derived.
- With ***view materialization***, the view is stored as a ***temporary table***, which is maintained as the underlying base tables are updated.

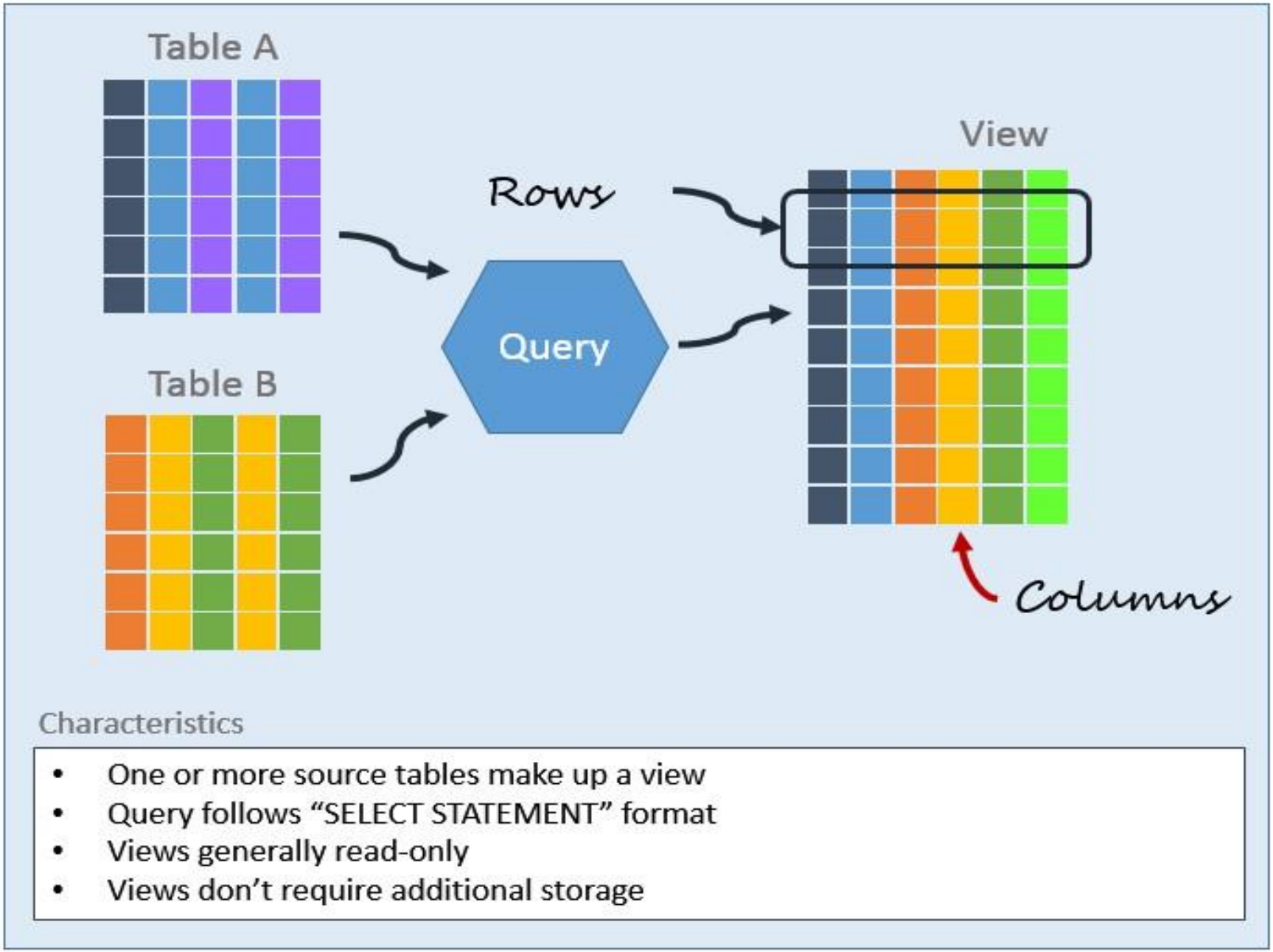
Views

- CREATE VIEW statement: data definition command that stores the subquery specification in the data dictionary
  - CREATE VIEW command syntax:

    CREATE VIEW *viewname* AS SELECT *query*

14

Anatomy of a View



Characteristics

- One or more source tables make up a view
- Query follows “SELECT STATEMENT” format
- Views generally read-only
- Views don’t require additional storage

Views – Creating View

CREATE VIEW ***view_name*** **[(*newColumnName* [,...])]**
AS subselect [WITH [CASCADED | LOCAL] CHECK OPTION]

- Can assign a name to each column in view.
- If list of column names is specified, it must have same number of items as number of columns produced by ***subselect***.
- If omitted, each column takes name of corresponding column in ***subselect***.
- List must be specified if there is any ambiguity in a column name.
- The ***subselect*** is known as the ***defining query***.

16

Views – Creating View

- **WITH CHECK OPTION** ensures that if a row fails to satisfy **WHERE** clause of defining query, it is not added to underlying base table.

- Need **SELECT** privilege on all tables referenced in ***subselect*** and **USAGE** privilege on any domains used in referenced columns.

Views – Creating Horizontal View

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

- Restrict a user's access to ***selected rows*** of one or more tables
- Create view so that manager at branch B003 can only see details for staff who work in his or her office.

```
CREATE VIEW Manager3Staff
AS  SELECT *
      FROM Staff
      WHERE branchNo = 'B003';
```

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

18

Views – Creating Vertical View

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

- Restrict a user's access to ***selected columns*** of one or more tables
- Create view of staff details at branch B003 excluding salaries.

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
FROM Staff
WHERE branchNo = 'B003';
```

staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

19

Views - Grouped and Joined Views

Create view of staff who manage properties for rent, including branch number they work at, staff number, and number of properties they manage.

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS SELECT s.branchNo, s.staffNo, COUNT(*)
        FROM Staff s, PropertyForRent p
        WHERE s.staffNo = p.staffNo
        GROUP BY s.branchNo, s.staffNo;
```

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Views – Removing View

DROP VIEW ***view_name*** **[RESTRICT | CASCADE]**

- Causes definition of view to be ***deleted*** from database.
- For example:

DROP VIEW Manager3Staff;

- With **CASCADE**, all related dependent objects are ***deleted***; i.e. any views defined on view being dropped.
- With **RESTRICT** (default), if any other objects depend for their existence on continued existence of view being dropped, ***command is rejected***.

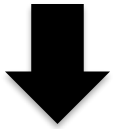
21

Views – View Resolution

- How a query on a view is handled
- Count number of properties managed by each member at branch B003.

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

```
SELECT staffNo, cnt
FROM StaffPropCnt
WHERE branchNo = 'B003'
ORDER BY staffNo;
```



staffNo	cnt
SG14	1
SG37	2

Views – View Resolution

(a) View column names in **SELECT** list are translated into their corresponding column names in the defining query:

```
SELECT s.staffNo As staffNo, COUNT(*) As cnt
```

(b) View names in **FROM** are replaced with corresponding **FROM** lists of defining query:

```
FROM Staff s, PropertyForRent p
```

Views – View Resolution

(c) **WHERE** from user query is combined with **WHERE** of defining query using **AND**:

```
WHERE s.staffNo = p.staffNo AND branchNo = 'B003'
```

(d) **GROUP BY** and **HAVING** clauses copied from defining query:

```
GROUP BY s.branchNo, s.staffNo
```

(e) **ORDER BY** copied from query with view column name translated into defining query column name

```
ORDER BY s.staffNo
```


24

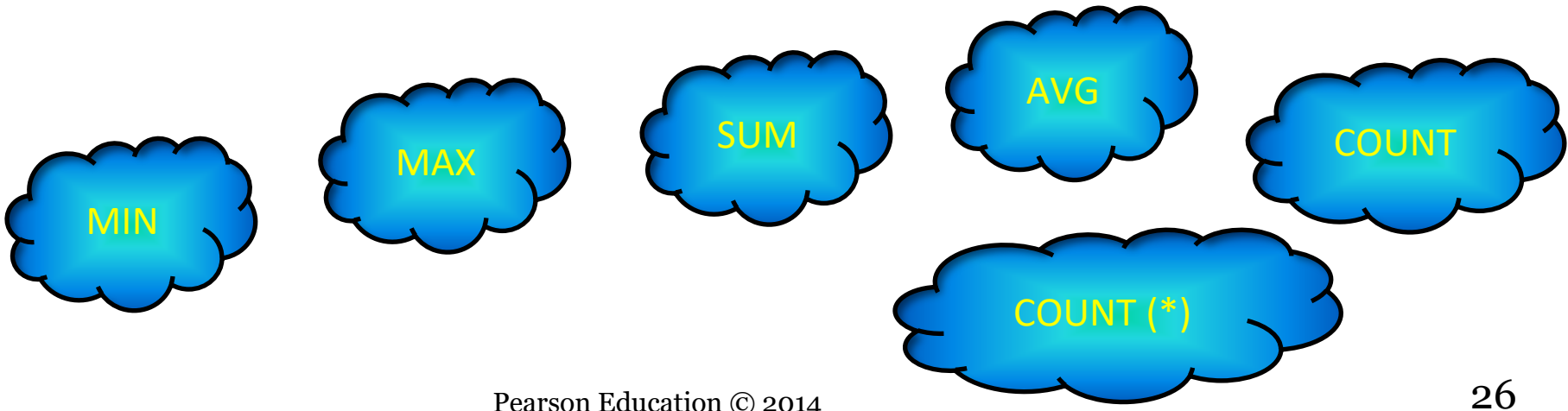
Views – View Resolution

(f) Final merged query is now executed to produce the result:

```
SELECT s.staffNo AS staffNo, COUNT(*) AS cnt
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo AND
            branchNo = 'B003'
GROUP BY s.branchNo, s.staffNo
ORDER BY s.staffNo;
```

Views – Restrictions on Views

- SQL imposes several **restrictions** on creation and use of views.
- If column in view is based on an ***aggregate function***:
  - Column may appear **only** in **SELECT** and **ORDER BY** clauses of queries that access view.
  - Column may not be used in **WHERE** nor be an argument to an aggregate function in any query based on view.



26

Views – Restrictions on Views

- For example, following query would fail:

```
SELECT COUNT(cnt)
FROM StaffPropCnt;
```

- Similarly, following query would also fail:

```
SELECT *
FROM StaffPropCnt
WHERE cnt > 2;
```

Restrictions on Views

- Grouped view may never be joined with a ***base table*** or a ***view***.
  - For example, **StaffPropCnt** view is a grouped view, so any attempt to join this view with another table or view ***fails***.

28

Views - View Updatability

- All updates to ***base table*** reflected in all views that encompass base table.

- Similarly, may expect that if ***view*** is updated then base table(s) will reflect change.

29

Views - View Updatability

- However, consider again view **StaffPropCnt**.

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

- If we tried to insert record showing that at branch B003, SG5 manages 2 properties:

```
INSERT INTO StaffPropCnt
VALUES ('B003', 'SG5', 2);
```

- Have to insert 2 records into **PropertyForRent** showing which properties SG5 manages. However, do not know which properties they are; i.e. do not know **primary keys!**

30

Views - View Updatability

- If the definition of the view is changed and replace count with actual property numbers:

```
CREATE VIEW StaffPropList (branchNo, staffNo, propertyNo)
AS  SELECT s.branchNo, s.staffNo, p.propertyNo
    FROM Staff s, PropertyForRent p
    WHERE s.staffNo = p.staffNo;
```

- Now try to insert the record:

```
INSERT INTO StaffPropList
VALUES ('B003', 'SG5', 'PG19');
```

  - Problem in insertion because in **PropertyForRent** all columns except **postcode/staffNo** are not allowed to be nulls.
  - No way of giving the remaining non-null columns with values.

Views - View Updatability

- ISO specifies that a view is updatable if and only if:
  - **DISTINCT** is not specified.
  - Every element in **SELECT** list of the defining query is a column name and no column appears more than once.
  - The **FROM** clause specifies only one source table, excluding any views based on a join, union, intersection or difference.
  - **WHERE** clause does not include any nested **SELECT** that reference the table in the **FROM** clause.
  - No **GROUP BY** or **HAVING** clause.
  - Also, every row added through view **must not** violate integrity constraints of base table.

32

Views - WITH CHECK OPTION

- Rows exist in a view because they satisfy the **WHERE** condition of the defining query.
- If a row changes and no longer satisfies the condition, it disappears from the view.
- New rows will appear within view when insert/update on the view causes them to satisfy **WHERE** condition.
- Rows that enter or leave a view are called ***migrating rows***.
- **WITH CHECK OPTION** prohibits a row from migrating out of the view.

33

Views - WITH CHECK OPTION

- **LOCAL/CASCADED** are applicable to view hierarchies.
- With **LOCAL**, any row insert/update on the view and on any view directly or indirectly defined on this view **must not** cause the row to disappear from view unless the row also disappears from derived view/table.
- With **CASCADED** (default), any row insert/ update on the view and on any view directly or indirectly defined on this view **must not** cause row to disappear from the view.

https://www.mysqltutorial.org/mysql-view

34

Views - WITH CHECK OPTION (Example)

```
CREATE VIEW Manager3Staff
    AS  SELECT *
    FROM Staff
    WHERE branchNo = 'B003'
    WITH CHECK OPTION;
```

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

```
UPDATE Manager3Staff
  SET branchNo = 'B005'
  WHERE staffNo = 'SG37'
```

- **Cannot update** branch number of row B003 to B005 as this would cause row to migrate from view.

```
INSERT INTO Manager3Staff
VALUES ('SL15, 'Mary', 'Black', 'Assistant', 'F', '1967-06-21', 8000, 'B002)
```

- **Cannot insert** a row into view with a branch number that does not equal B003.

Views - WITH CHECK OPTION (Example)

- Now consider the following:

```
CREATE VIEW LowSalary
    AS    SELECT *
    FROM Staff
    WHERE salary > 9000;
```

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

```
CREATE VIEW HighSalary
    AS    SELECT *
    FROM LowSalary
    WHERE salary > 10000
    WITH LOCAL CHECK OPTION;
```

```
CREATE VIEW Manager3Staff
    AS    SELECT *
    FROM HighSalary
    WHERE branchNo = 'B003';
```

Views - WITH CHECK OPTION (Example)

```
UPDATE Manager3Staff
SET salary = 9500
WHERE staffNo = 'SG37';
```

- This update would ***fail***: although update would cause row to disappear from **HighSalary**, row would not disappear from **LowSalary**.

- However, if update tried to set salary to 8000, update would succeed as row would no longer be part of LowSalary.

Views - WITH CHECK OPTION (Example)

- If **HighSalary** had specified **WITH CASCADED CHECK OPTION**, setting salary to 9500 or 8000 would be ***rejected*** because row would disappear from **HighSalary**.
- To prevent anomalies like this, each view should be created using **WITH CASCADED CHECK OPTION**.

38

Views - Advantages of Views

- Data independence
- Currency
- Improved security
- Reduced complexity
- Convenience
- Customization
- Data integrity

39

Views - Disadvantages of Views

- Update restriction
- Structure restriction
- Performance

40

Views - View Materialization

- View resolution mechanism might slow, particularly if view is **accessed frequently**.
- View materialization stores view as ***temporary table*** when view is first queried.
- Thereafter, queries based on materialized view can be **faster** than recomputing view each time.
- The difficulty of this approach is to maintain the currency of view while base tables(s) are being updated.

Views - View Materialization

- View maintenance aims to apply only those changes necessary to keep view current.
- Consider following view:

```
CREATE VIEW StaffPropRent(staffNo)
    AS SELECT DISTINCT staffNo
        FROM PropertyForRent
        WHERE branchNo = 'B003' AND rent > 400;
```

staffNo
SG37
SG14

42

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

staffNo
SG37
SG14

Views - View Materialization

- If insert row into **PropertyForRent** with rent £400 then view would be **unchanged**.
- If insert row for property **PG24** at branch **B003** with staffNo = **SG19** and rent = **550**, then row would ***appear*** in materialized view.
- If insert row for property **PG54** at branch **B003** with staffNo = **SG37** and rent = **450**, then no new row would need to be added to materialized view.
- If delete property **PG24**, row should be deleted from materialized view.
- If delete property **PG54**, then row for **SG37** should not be deleted (because of existing property **PG21**).

44

Updatable Views

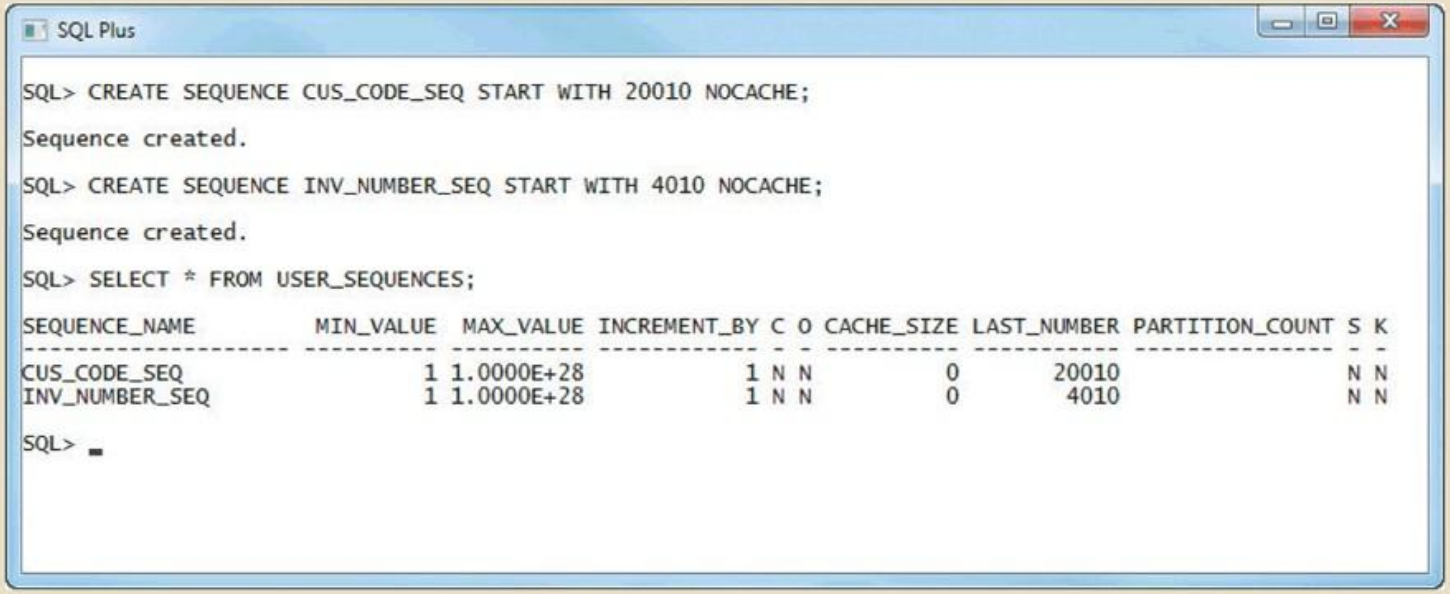
- Used to update attributes
  - Batch update routine: pools multiple transactions into a single batch to update a master table field in a single operation
- Updatable view restrictions
  - GROUP BY expressions or aggregate functions cannot be used
  - Set operators cannot be used
  - Most restrictions are based on the use of JOINs or group operators in views

Sequences (1 of 2)

- Many similarities in the use of sequences across these DBMSs
  - Independent object in the database
  - Have a name and can be used anywhere a value expected
  - Not tied to a table or column
  - Generate a numeric value that can be assigned to any column in any table
  - Table attribute with an assigned value can be edited and modified

Sequences (2 of 2)

FIGURE 8.10 ORACLE SEQUENCE



Procedural SQL (1 of 3)

- Performs a conditional or looping operation by isolating critical code and making all application programs call the shared code
  - Better maintenance and logic control
- Persistent stored module (PSM): block of code
  - Contains standard SQL statements and procedural extensions that is stored and executed at the DBMS server

Procedural SQL (2 of 3)

- Procedural Language SQL (PL/SQL)
  - Use and storage of procedural code and SQL statements within the database
  - Merging of SQL and traditional programming constructs
- Procedural code is executed as a unit by DBMS when invoked by end user
  - Anonymous PL/SQL blocks
  - Triggers
  - Stored procedures
  - PL/SQL functions

Procedural SQL (3 of 3)

Table 8.4 PL/SQL BASIC DATA TYPES	
DATA TYPE	**DESCRIPTION**
CHAR	Character values of a fixed length; for example: W_ZIP CHAR(5)
VARCHAR2	Variable-length character values; for example: W_FNAME VARCHAR2(15)
NUMBER	Numeric values; for example: W_PRICE NUMBER(6,2)
DATE	Date values; for example: W_EMP_DOB DATE
%TYPE	Inherits the data type from a variable that you declared previously or from an attribute of a database table; for example: W_PRICE PRODUCT.P_PRICE%TYPE Assigns W_PRICE the same data type as the P_PRICE column in the PRODUCT table

Triggers (1 of 2)

- Procedural SQL code automatically invoked by RDBMS when given data manipulation event occurs
- Parts of a trigger definition
  - Triggering timing: indicates when trigger's PL/SQL code executes
  - Triggering event: statement that causes the trigger to execute
    - Triggering level: statement- and row-level
    - Triggering action: PL/SQL code enclosed between the BEGIN and END keywords

Triggers (2 of 2)

- DROP TRIGGER trigger_name command
  - Deletes a trigger without deleting the table
- Trigger action based on conditional DML predicates
  - Actions depend on the type of DML statement that fires the trigger

Stored Procedures

- Named collection of procedural and SQL statements
  - Stored in the database
  - Can be used to encapsulate and represent business transactions
- Advantages
  - Reduce network traffic and increase performance
  - Decrease code duplication by means of code isolation and code sharing

PL/SQL Processing with Cursors (1 of 3)

- Cursor: special construct used to hold data rows returned by a SQL query
  - Implicit cursor: automatically created when SQL statement returns only one value
  - Explicit cursor: holds the output of a SQL statement that may return two or more rows
  - Syntax:

    CURSOR *cursor_name* IS *select-query*;
- Cursor-style processing involves retrieving data from the cursor one row at a time
  - Current row is copied to PL/SQL variables

PL/SQL Processing with Cursors (2 of 3)

Table 8.5 **Cursor Processing Commands**	
Cursor Command	Explanation
OPEN	Opening the cursor executes the SQL command and populates the cursor with data, opening the cursor for processing. The cursor declaration command only reserves a named memory area for the cursor; it does not populate the cursor with the data. Before you can use a cursor, you need to open it. For example: OPEN *cursor_name*
FETCH	Once the cursor is opened, you can use the FETCH command to retrieve data from the cursor and copy it to the PL/SQL variables for processing. T he syntax is: FETCH cursor_name INTO variable1 [, variable2, …] The PL/SQL variables used to hold the data must be declared in the DECLARE section and must have data types compatible with the columns retrieved by the SQL command. If the cursor's SQL statement returns five columns, there must be five PL/SQL variables to receive the data from the cursor. This type of processing resembles the one-record-at-a-time processing used in previous database models. The first time you fetch a row from the cursor, the first row of data from the cursor is copied to the PL/SQL variables; the second time you fetch a row from the cursor, the second row of data is placed in the PL/SQL variables; and so on.
CLOSE	The CLOSE command closes the cursor for processing.

PL/SQL Processing with Cursors (3 of 3)

Table 8.6 Cursor Attributes	
Attribute	**Description**
%ROWCOUNT	Returns the number of rows fetched so far. If the cursor is not OPEN, it returns an error. If no FETCH has been done but the cursor is OPEN, it returns 0.
%FOUND	Returns TRUE if the last FETCH returned a row, and FALSE if not. If the cursor is not OPEN, it returns an error. If no FETCH has been done, it contains NULL.
%NOT FOUND	Returns TRUE if the last FETCH did not return any row, and FALSE if it did. If the cursor is not OPEN, it returns an error. If no FETCH has been done, it contains NULL.
%ISOPEN	Returns TRUE if the cursor is open (ready for processing) or FALSE if the cursor is closed. Remember, before you can use a cursor, you must open it.

PL/SQL Stored Functions

- Stored function: named group of procedural and SQL statements that returns a value
  - Indicated by a RETURN statement in its program code
- Can be invoked only from within stored procedures or triggers
  - Cannot be invoked from SQL statements unless the function follows some very specific compliance rules

Embedded SQL (1 of 4)

- SQL statements contained within an application programming language
  - Host language: any language that contains embedded SQL statements
- Differences between SQL and procedural languages
  - Run-time mismatch
    - SQL is executed one instruction at a time
    - Host language runs at client side in its own memory space
  - Processing mismatch
    - Conventional programming languages process one data element at a time
    - Newer programming environments manipulate data sets in a cohesive manner
  - Data type mismatch
    - Data types provided by SQL might not match data types used in different host languages

Embedded SQL (2 of 4)

- Embedded SQL framework defines:
  - Standard syntax to identify embedded SQL code within the host language
  - Standard syntax to identify host variables
  - Communication area used to exchange status and error information between SQL and host language

Embedded SQL (3 of 4)

Table 8.7 SQL Status and Error Reporting Variables		
Variable Name	**Value**	**Explanation**
SQLCODE		Old-style error reporting supported for backward compatibility only; returns an integer value (positive or negative)
	0	Successful completion of command
	100	No data; the SQL statement did not return any rows and did not select, update, or delete any rows
	-999	Any negative value indicates that an error occurred
SQLSTATE		Added by SQL-92 standard to provide predefined error codes; defined as a character string (5 characters long)
	00000	Successful completion of command
		Multiple values in the format XXYYY where: XX-> represents the class code YYY-> represents the subclass code

Embedded SQL (4 of 4)

- Static SQL: programmer uses predefined SQL statements and parameters
  - SQL statements will not change while application is running
- Dynamic SQL: SQL statement is generated at run time
  - Attribute list and condition are not known until end user specifies them
  - Slower than static SQL
  - Requires more computer resources
  - Inconsistent levels of support and incompatibilities among DBMS vendors

Summary (1 of 2)

- The ANSI standard data types are supported by all RDBMS vendors in different ways
- The basic data definition commands allow you to create tables and indexes
- Data manipulation commands allow you to add, modify, and delete rows from tables
- Views can be created to expose subsets of data to end users primarily for security and privacy reasons
- In Oracle and SQL Server, sequences may be used to generate values to be assigned to a record
- Procedural Language SQL (PL/SQL) can be used to create triggers, stored procedures, and PL/SQL functions

Summary (2 of 2)

- A stored procedure is a named collection of SQL statements
- When SQL statements are designed to return more than one value inside the PL/SQL code, a cursor is needed
- Embedded SQL refers to the use of SQL statements within an application programming language such as Visual Basic .NET, C#, COBOL, or Java