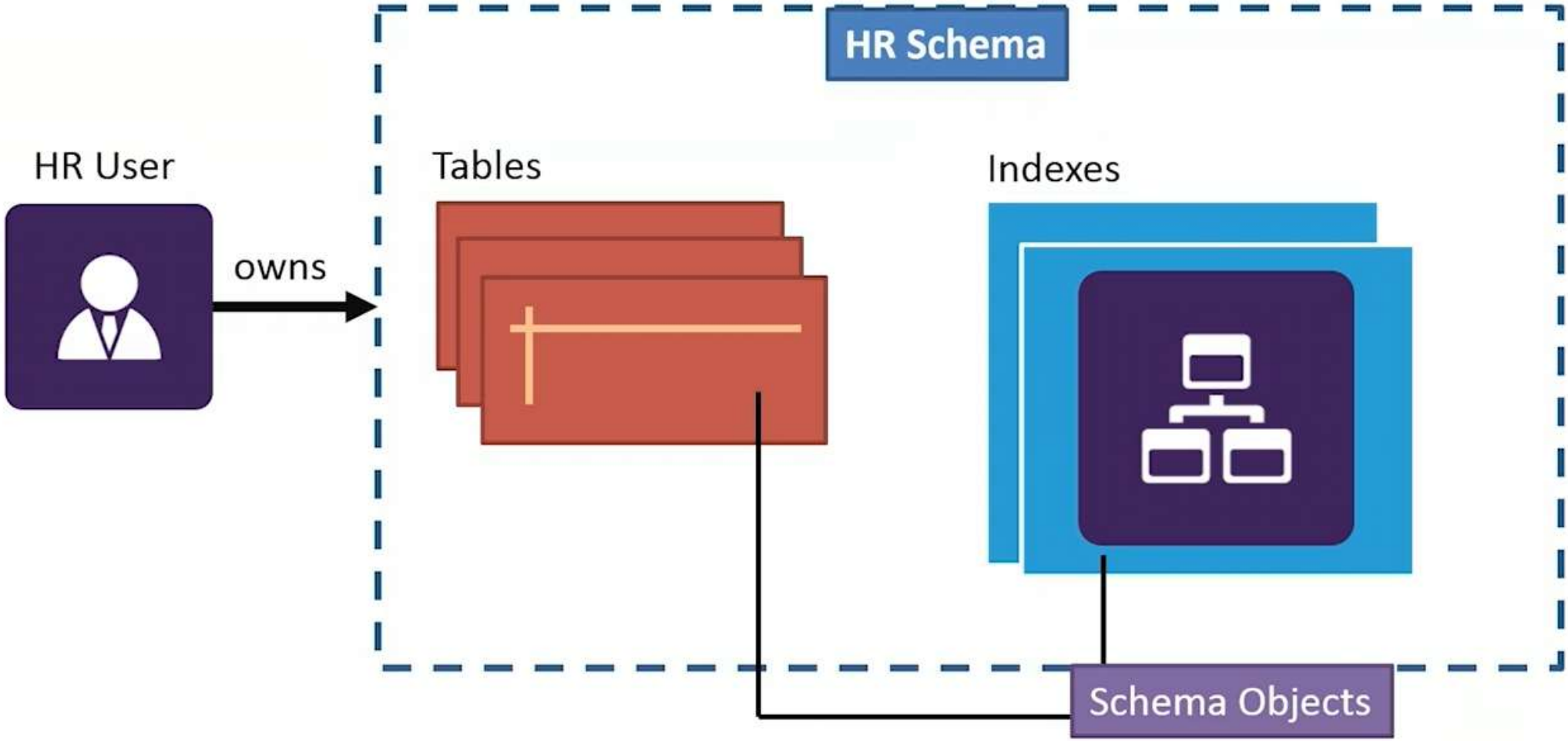
Entity Relationship Model

Entity relationship modeling conventions

Entity	Attribute
• Singular, unique name	• Singular name
• Uppercase	• Lowercase
• Soft box	• Mandatory marked with "*"
• Synonym in parentheses	• Optional marked with "o"

Relational Database Terminology



Relational Database Terminology

Schema object types

- Indexes
- Partitions
- Views
- Sequences
- Synonyms
- PL/SQL subprograms and packages

Using SQL to Query a Database

•SELECT •INSERT •UPDATE •DELETE •MERGE	Data manipulation language (DML)
•CREATE •ALTER •DROP •RENAME •TRUNCATE •COMMENT	Data definition language (DDL)
•GRANT •REVOKE	Data control language (DCL)
•COMMIT •ROLLBACK •SAVEPOINT	Transaction control

Human Resources (HR) Schema

- Sample schemas
  - HR, OE, OC, PM, IX, SH
- Design principles
  - Simplicity and ease of use
  - Relevance for typical users
  - Extensibility

Introduction to SQL Developer

Specifications of SQL Developer

- Is shipped along with Oracle Database 12*c* Release 1
- Is developed in Java
- Supports Windows, Linux, and Mac OS X platforms
- Enables default connectivity using the JDBC thin driver
- Connects to Oracle Database version 9.2.0.1 and later

Oracle SQL Developer

File Edit View Navigate Run Team Tools Window Help

Connections

- Connections
- Oracle NoSQL Connections
- Cloud Connections

Reports

- All Reports
- Data Dictionary Reports
- Data Modeler Reports
- OLAP Reports
- TimesTen Reports
- User Defined Reports

New / Select Database Connection

Connection Name	Connection Details

Connection Name: ORCL
Username: system
Password: ••••••••

Save Password Connection Color

Oracle

Connection Type: Basic Role: default

Hostname: localhost
Port: 1521
SID: xe
Service name:

OS Authentication Kerberos Authentication Advanced...

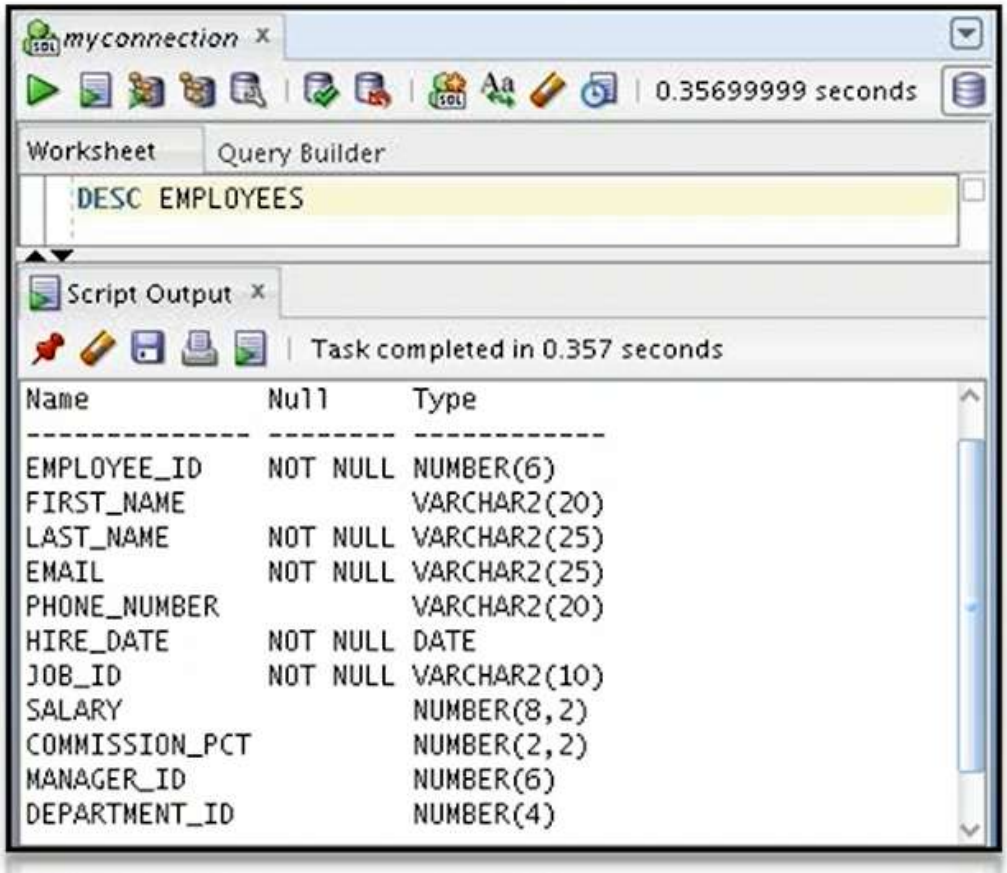
Status :

Help Save Clear Test Connect Cancel

Browsing Database Objects and Structures

Displaying the table structure

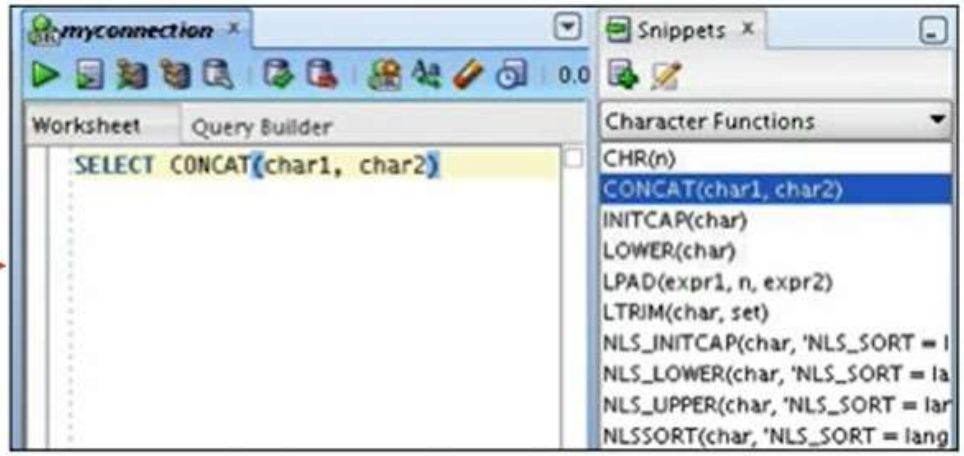
- Use the `DESCRIBE` command to display the structure of a table



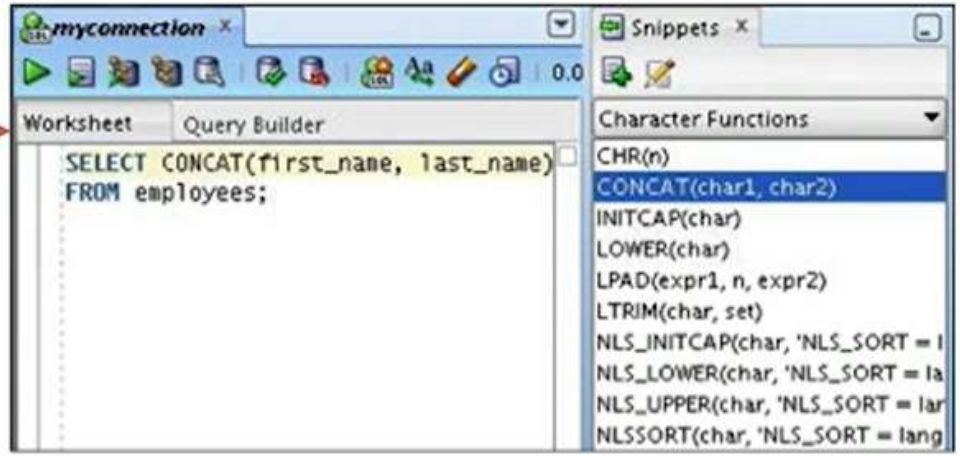
Using Snippets

Examples

Inserting a snippet



Editing the snippet



SQL Statements vs SQL*Plus Commands

SQL

- A language
- ANSI standard
- Keywords can't be abbreviated
- Statements manipulate the data and table definitions in the database

SQL statements

SQL buffer ✓

SQL *Plus

- An environment
- Oracle Proprietary
- Keywords can be abbreviated
- Commands don't allow for the manipulation of the values in the database

SQL *Plus commands

SQL *Plus buffer ✗

SQL*Plus Editing Commands

- A[PPEND] *text*
- C[HANGE] / *old* / *new*
- C[HANGE] / *text* /
- CL[EAR] BUFF[ER]
- DEL
- DEL *n*
- DEL *m n*

SQL *Plus Editing Commands

Airplane mode on

- I[NPUT]
- I[NPUT] *text*
- L[IST]
- L[IST] *n*
- L[IST] *m n*
- R[UN]
- *n*
- *n text*
- O *text*

SQL*Plus File Commands

- SAVE *filename*
- GET *filename*
- START *filename*
- @ *filename*
- EDIT *filename*
- SPOOL *filename*
- EXIT

AUTOTRACE Command

- It displays a report after the successful execution of SQL DML statements such as `SELECT`, `INSERT`, `UPDATE`, or `DELETE`.
- The report can now include execution statistics and the query

```
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]]
    [STAT[ISTICS]]
```

```
SET AUTOTRACE ON
-- The AUTOTRACE report includes both the optimizer
-- execution path and the SQL statement execution
-- statistics
```

Writing SQL Statements

- SQL statements are not case-sensitive.
- SQL statements can be entered on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.
- In SQL Developer, SQL statements can be optionally terminated by a semicolon (;). Semicolons are required when you execute multiple SQL statements.
- In SQL*Plus, you are required to end each SQL statement with a semicolon (;).

Column Heading Default

- SQL Developer:
  - Default heading alignment: Left-aligned
  - Default heading display: Uppercase
- SQL*Plus:
  - Character and Date column headings are left-aligned.
  - Number column headings are right-aligned.
  - Default heading display: Uppercase

Defining a Column Alias

- A column alias:
  - Renames a column heading
  - Is useful with calculations
  - Immediately follows the column name (there can also be the optional `AS` word between the column name and the alias)
  - Requires double quotation marks if it contains spaces or special characters, or if it is case-sensitive

Alternative Quote (q) Operator

- Specify your own quotation mark delimiter
- Select any delimiter
- Increase readability and usability

```
SELECT department_name || q'[ Department's Manager Id: ]'
       || manager_id
       AS "Department and Manager"
FROM departments;
```

	Department and Manager
1	Administration Department's Manager Id: 200
2	Marketing Department's Manager Id: 201
3	Shipping Department's Manager Id: 124
4	IT Department's Manager Id: 103
5	Sales Department's Manager Id: 149
6	Executive Department's Manager Id: 100
7	Accounting Department's Manager Id: 205
8	Contracting Department's Manager Id:

Duplicate Rows

The default display of queries is all rows, including duplicate rows.

1

```
SELECT department_id
FROM   employees;
```

	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60
6	60
7	50
8	50

...

2

```
SELECT DISTINCT department_id
FROM   employees;
```

	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110
5	50
6	80
7	60
8	10

Comparison Operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN ...AND...	Between two values (inclusive)
IN(set)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

Pattern matching using the LIKE operator

- Use the LIKE operator to perform wildcard searches of valid search string values
- Search conditions can contain either literal characters or numbers:
  - % denotes zero or more characters
  - _ denotes one character

```
SELECT    first_name
FROM      employees
WHERE     first_name LIKE 'S%' ;
```

	FIRST_NAME
1	Shelley
2	Steven

Rules of precedence

- You can use parentheses to override rules of precedence

Operator	Meaning
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	`IS [NOT] NULL, LIKE, [NOT] IN`
5	`[NOT] BETWEEN`
6	Not equal to
7	`NOT` logical operator
8	`AND` logical operator
9	`OR` logical operator

SQL row limiting clause

- You can use the row limiting clause to limit the rows that are returned by a query
- You can use this clause to implement Top-N reporting
- You can specify the number of rows or percentage of rows to return with the `FETCH FIRST` keyword
- You can use the `OFFSET` keyword to specify that the returned rows begin with a row after the first row of the full result set
- The `WITH TIES` keyword includes additional rows with the same ordering keys as the last row of the row-limited result set (you must specify `ORDER BY` in the query)

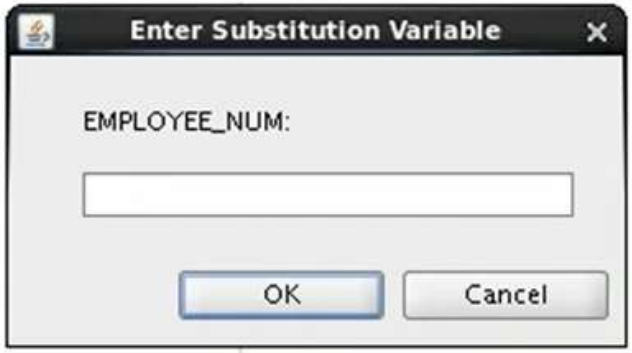
Substitution variables

- Use substitution variables to:
  - Temporarily store values with single-ampersand (`&`) and double-ampersand (`&&`) substitution
- Use substitution variables to supplement the following:
  - `WHERE` conditions
  - `ORDER BY` clauses
  - Column expressions
  - Table names
  - Entire `SELECT` statements

Using the single-ampersand substitution variable

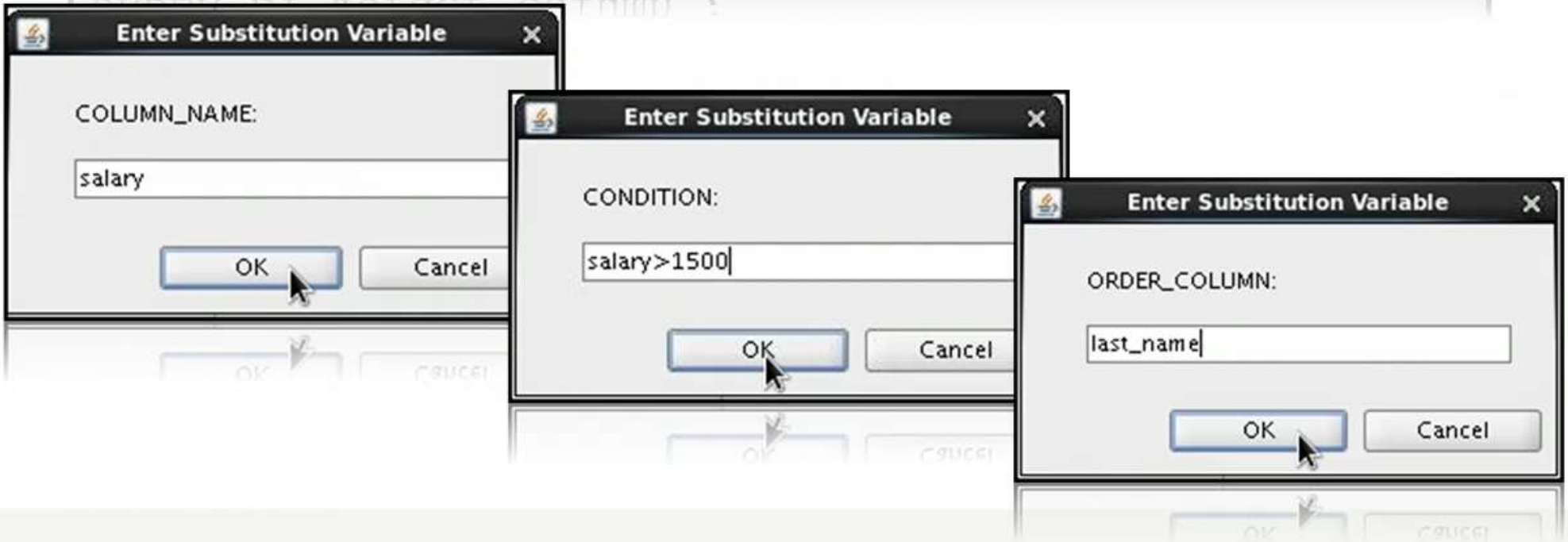
- Use a variable prefixed with an ampersand (&) to prompt the user for a value:

```
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;
```



Specifying column names, expressions, and text

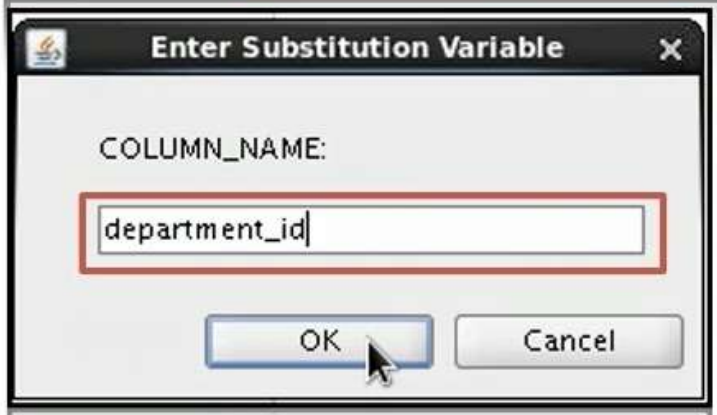
```
SELECT employee_id, last_name, job_id,&column_name
FROM   employees
WHERE  &condition
ORDER BY &order_column ;
```



Using the double-ampersand and substitution variable

- Use double ampersand (&&) if you want to reuse the variable value without prompting the user each time:

```
SELECT   employee_id, last_name, job_id, &&column_name
FROM     employees
ORDER BY &column_name ;
```



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	200	Whalen	AD_ASST	10
2	201	Hartstein	MK_MAN	20
3	202	Fay	MK_REP	20

Using the DEFINE command

- Use the DEFINE command to create and assign a value to a variable
- Use the UNDEFINE command to remove a variable

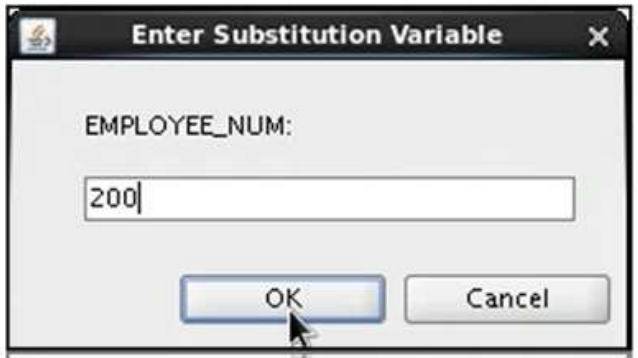
```
DEFINE employee_num = 200

SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;

UNDEFINE employee_num
```

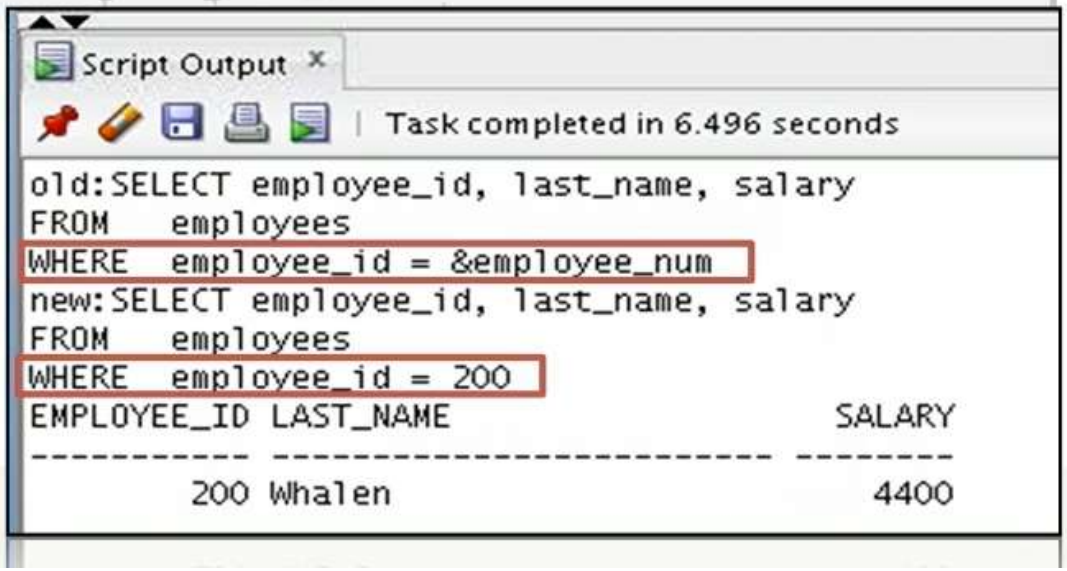
	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	200	Whalen	4400	10

Using the VERIFY command

- Use the VERIFY command to toggle the display of the substitution variable, both before and after SQL Developer replaces substitution variables with values:

```
SET VERIFY ON
SELECT employee_id, last_name, salary
FROM   employees
WHERE  employee_id = &employee_num;
```



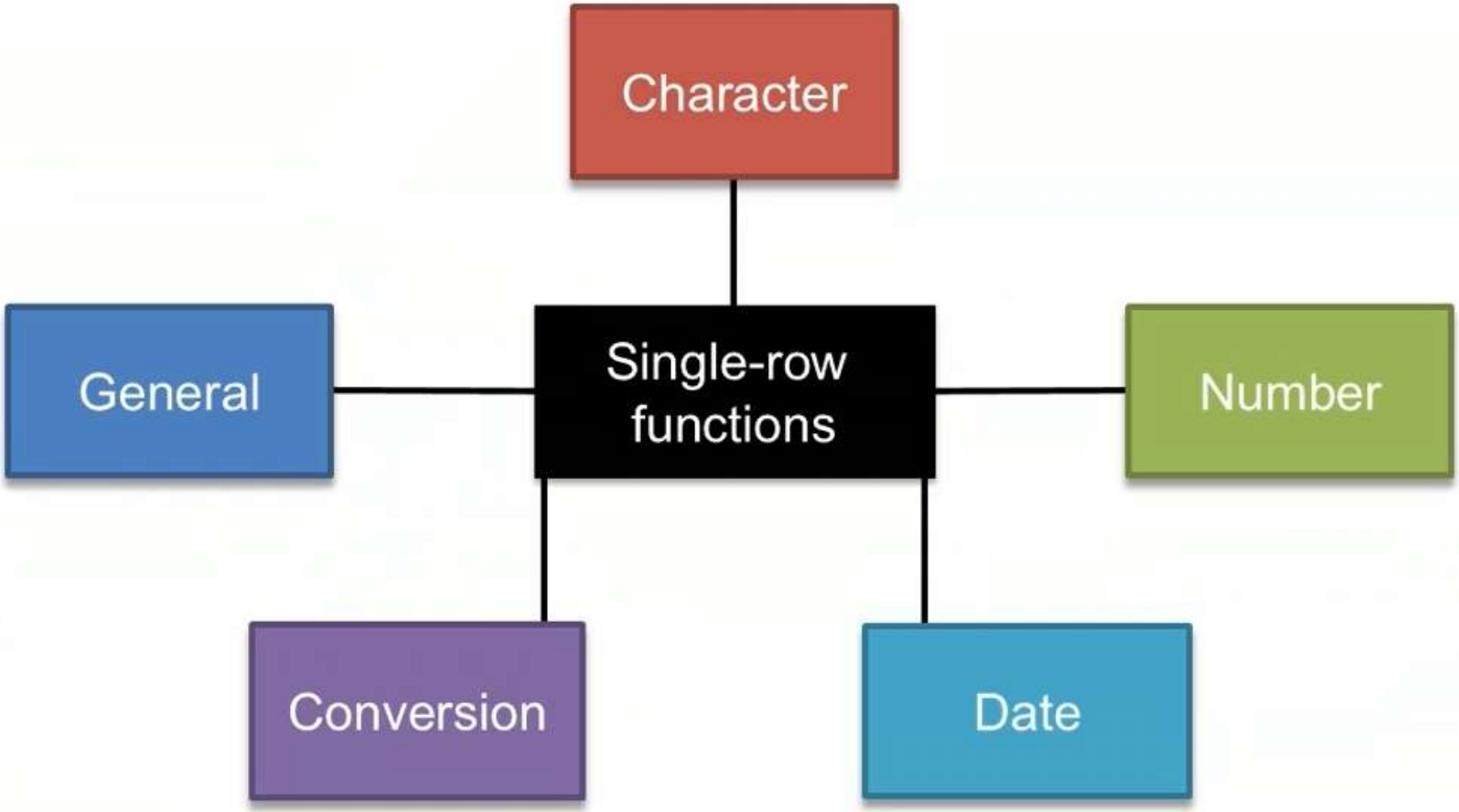


Single-row Functions

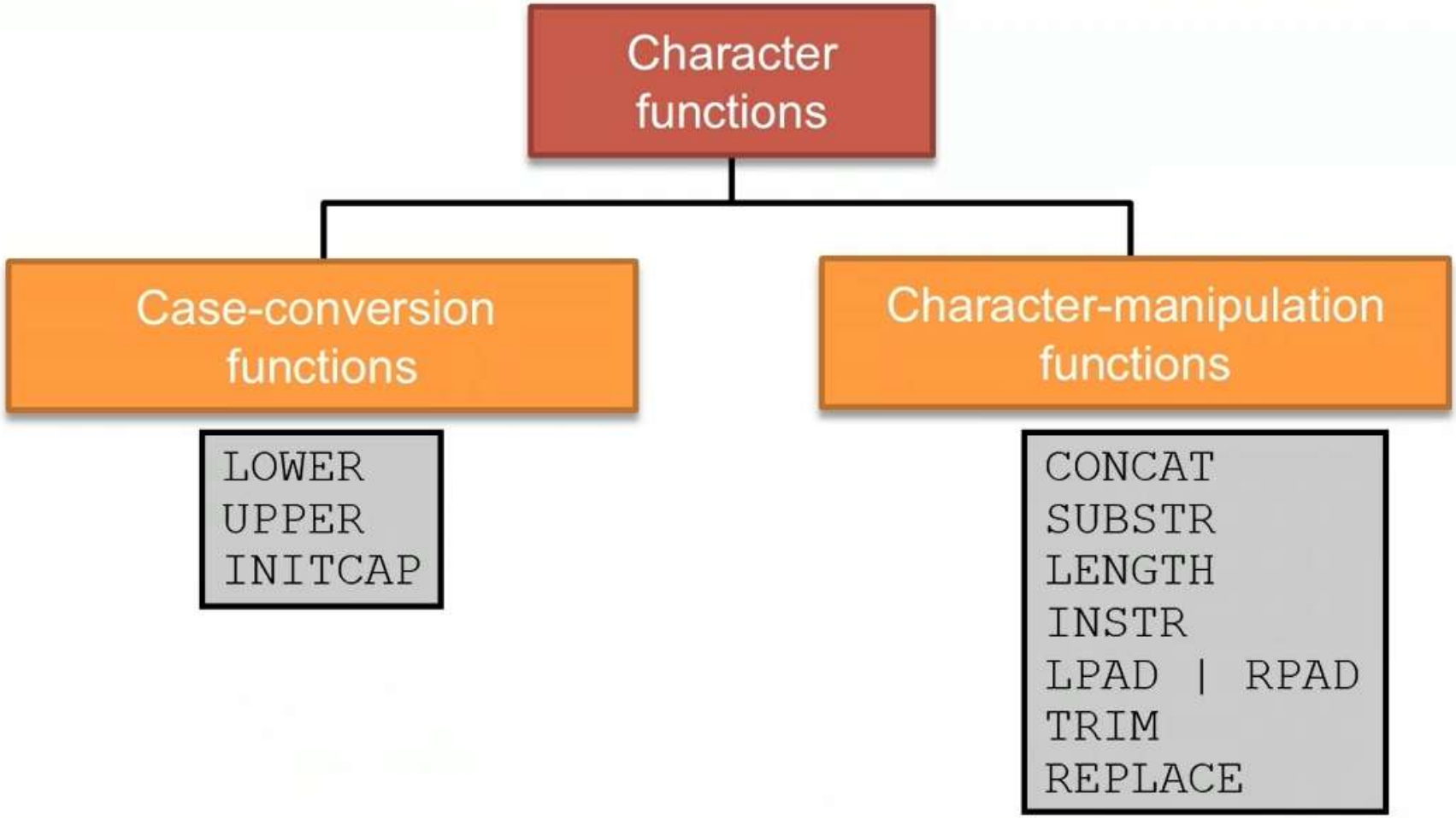
- Manipulate data items
- Accept arguments and return one value
- Act on each row that is returned
- Return one result per row
- May modify the data type
- Can be nested
- Accept arguments that can be a column or an expression

```
function_name [(arg1, arg2,...)]
```

Single-row Functions



Character Functions



Character Functions

Function	Purpose
LOWER(*column*|*expression*)	Converts alpha character values to lowercase
UPPER(*column*|*expression*)	Converts alpha character values to lowercase
INITCAP(*column*|*expression*)	Converts alpha character values to uppercase for the first letter of each word: all other letters in lowercase
CONCAT(*column*|*expression1*, *column2*|*expression2*)	Concatenates the first character value to the second character value: equivalent to concatenation operator (II)
SUBSTR(*column*|*expression*,*m*(,*n*})	Returns specified characters from character value starting at character position *m*, *n* characters long (If *m* is negative, the count starts from the end of the character value. If *n* is omitted, all characters to the end of the string are returned.)

Character Functions

Function	Purpose
LENGTH(column|expression)	Returns the number of characters in the expression
INSTR(*column|expression, 'string', {,m}, {n}*)	Returns the numeric position of a named string, Optionally, you can provide a position *m* to start searching, and the occurrence *n* of the string. *m* and *n* default to 1, meaning start the search at the beginning of the string and report the first occurrence,
LPAD(*column|expression, n, 'string'*} RPAD(*column|expression, n, 'string'*}	Returns an expression left-padded to length of *n* characters with a character expression. Returns an expression right-padded to length of *n* characters with a character expression.
TRIM(*leading|trailer|both, trim_character FROM trim_source*)	Enables you to trim leading or trailing characters (or both) from a character string. If *trim_character* or *trim_source* is a character literal, you must enclose it in a single quotation marks,
REPLACE(text, search_string, Replacement_string)	Searches a text expression for a character string and, if found, replaces it with a specified replacement string.

Case-conversion Functions

- These functions convert the case for character strings:

Function	Result
LOWER('SQL Course')	sql course
UPPER(SQL Course)	SQL COURSE
INITCAP('SQL Course')	Sql Course

Character-manipulation Functions

- These functions manipulate character strings:

Function	Result
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld ,1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(last_name,12,'-')	*****24000
RPAD(first_name, 12, '-')	24000*****

Numeric Functions

- `ROUND` rounds the value to a specified decimal
- `TRUNC` truncates the value to a specified decimal
- `CEIL` returns the smallest whole number greater than or equal to a specified number
- `FLOOR` returns the largest whole number equal to or less than a specified number
- `MOD` returns the remainder of division

Numeric Functions: Examples

Function	Result
ROUND(45,926, 2)	45.93
TRUNC(45.926, 2)	45,92
CEIL (2,83)	3
FLOOR (2,83)	2
MOD (1600, 300)	100

RR Date Format

Current Year	Specified Date	RR Format	YY Format
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

		If the specified two-digit year is:	
		0–49	50–99
If two digits of the current year are:	0–49	The return date is in the current century	The return date is in the century before the current one
	50–99	The return date is in the century after the current one	The return date is in the current century

Date-manipulation Functions

Function	Result
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Week day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

Using Date Functions

Function	Result
MONTHS_BETWEEN ('01-SEP-05','11-JAN-04')	19.6774194
ADD_MONTHS ('31-JAN-04',1	'29-FEB-04'
NEXT_DAY ('01-SEP-05','FRIDAY')	'08-SEP-05'
LAST_DAY ('01-FEB-05')	'28-FEB-05'

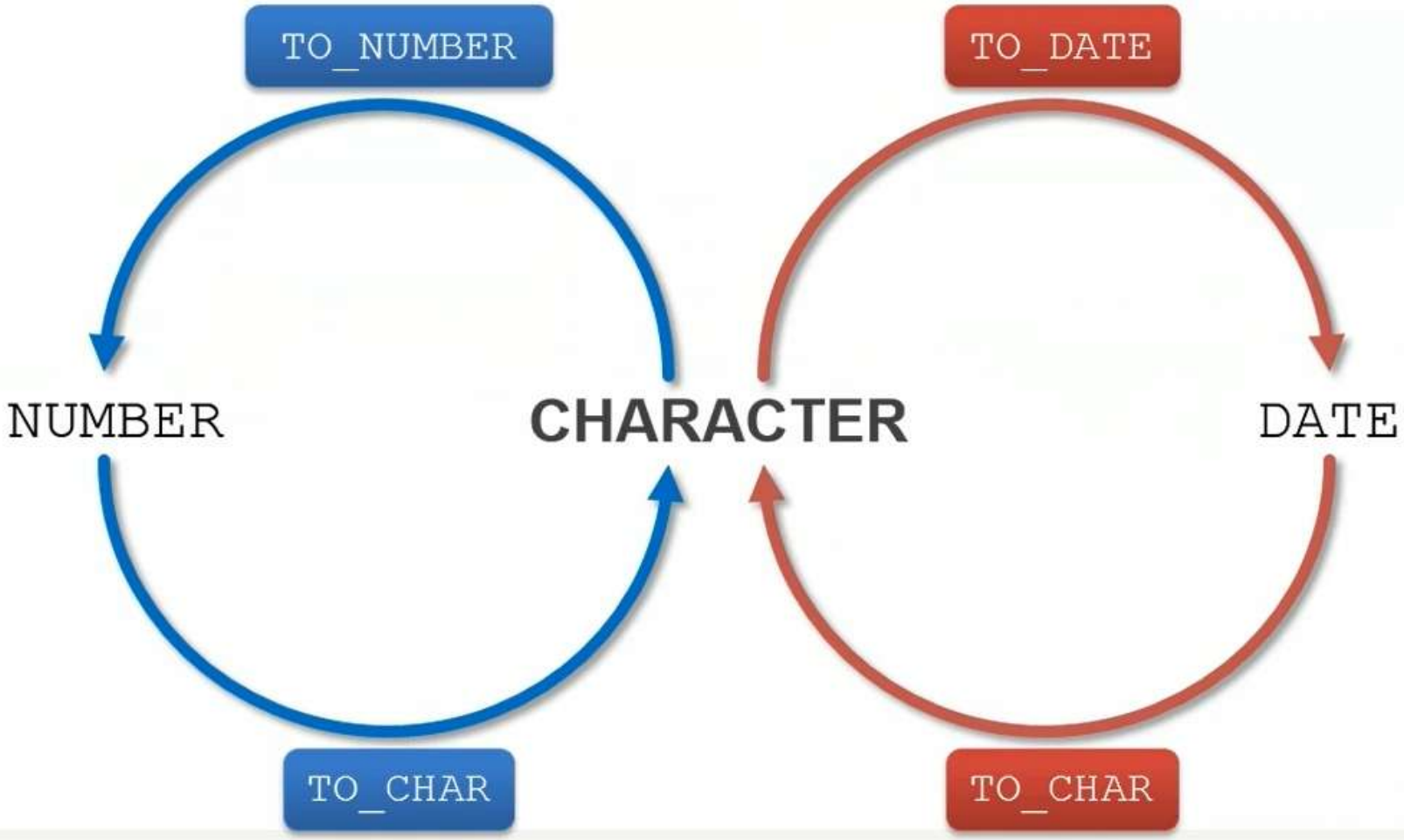
02:31
02:11 / 06:07

Using ROUND and TRUNC Functions with Dates

Function	Result
ROUND(SYSDATE,'MONTH')	01-AUG-03
ROUND(SYSDATE ,'YEAR')	01-JAN-04
TRUNC(SYSDATE ,'MONTH')	01-JUL-03
TRUNC(SYSDATE ,'YEAR')	01-JAN-03

Pause
00:24 / 02:43

Explicit Data Type Conversion



Elements of the Date Format Model

Element	Result
YYYY	Full year in numbers
YEAR	Year spelled out (in English)
MM	Two-digit value for the month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month

Elements of the Date Format Model

Element	Description
AM or PM	Meridian indicator
A.M. or P.M.	Meridian indicator with periods
HH or HH12	12 hour format
HH24	24 hour format
MI	Minute (0–59)
SS	Second (0–59)
SSSSS	Seconds past midnight (0–86399)

General Functions

- The following functions work with any data type and pertain to using nulls:
  - `NVL (expr1, expr2)`
  - `NVL2 (expr1, expr2, expr3)`
  - `NULLIF (expr1, expr2)`
  - `COALESCE (expr1, expr2, ..., exprn)`

Function	Description
NVL	Converts a null value to an actual value
NVL2	If expr1 is not null, NVL2 returns expr2. If expr1 is null, NVL2 returns expr3. The argument expr1 can have any data type.
NULLIF	Compares two expressions and returns null if they are equal; returns the first expression if they are not equal
COALESCE	Returns the first non-null expression in the expression list

Note: For more information about the hundreds of functions available, see the "Functions" section in *Oracle Database SQL Language Reference* for 12*c* database

Using the COALESCE Function

- The advantage of the COALESCE function over the NVL function is that the COALESCE function can take multiple alternative values

- If the first expression is not null, the COALESCE function returns that expression; otherwise, it does a COALESCE of the remaining expressions

Conditional Expressions

- Provide the use of the `IF-THEN-ELSE` logic within a SQL statement

- Use the following methods:
  - `CASE` expression
  - Searched `CASE` expression
  - `DECODE` function

CASE Expression

Press Esc to exit full screen

- Facilitates conditional inquiries by doing the work of an `IF-THEN-ELSE` statement:

```
CASE expr WHEN comparison_expr1 THEN return_expr1
          [WHEN comparison_expr2 THEN return_expr2
           WHEN comparison_exprn THEN return_exprn
           ELSE else_expr]
END
```

00:26 / 04:51

DECODE Function

Press Esc to exit full screen

- Facilitates conditional inquiries by doing the work of a CASE expression or an IF-THEN-ELSE statement:

```
DECODE(col|expression, search1, result1
                       [, search2, result2,...,]
                       [, default])
```

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- SUM
- LISTAGG
- STDDEV
- VARIANCE

Creating Groups of Data: GROUP BY Clause Syntax

- You can divide rows in a table into smaller groups by using the GROUP BY clause

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

Using the GROUP BY Clause on Multiple Columns

```
SELECT    department_id, job_id, SUM(salary)
FROM      employees
WHERE      department_id > 40
GROUP BY department_id, job_id
ORDER BY department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12008

Illegal Queries Using Group Functions

- Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause:

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

```
ORA-00937: not a single-group group function
00937. 00000 - "not a single-group group function"
```

A GROUP BY clause must be added to count the last names for each department_id

```
SELECT department_id, job_id, COUNT(last_name)
FROM   employees
GROUP BY department_id;
```

```
ORA-00979: not a GROUP BY expression
00979. 00000 - "not a GROUP BY expression"
```

Either add job_id in the GROUP BY or remove the job_id column from the SELECT list

Illegal Queries Using Group Functions

- You cannot use the `WHERE` clause to restrict groups
- You use the `HAVING` clause to restrict groups
- You cannot use group functions in the `WHERE` clause

```
SELECT   department_id, AVG(salary)
FROM     employees
WHERE    AVG(salary) > 8000
GROUP BY department_id;
```

```
ORA-00934: group function is not allowed here
00934. 00000 - "group function is not allowed here"
*Cause:
*Action:
Error at Line: 3 Column: 9
```

Using the HAVING Clause

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

Nesting Group Functions

- Display the maximum average salary:

```
SELECT   MAX(AVG(salary))
FROM     employees
GROUP BY department_id;
```

	MAX(AVG(SALARY))
1	19333.3333333333333333333333333333333333

Pause
00:17 / 04:42

Types of Joins

Several joins are compliant with the SQL:1999 standard

- Natural joins with the `NATURAL JOIN` clause
- Joins with the `USING` clause
- Joins with the `ON` clause
- Outer joins
  - `LEFT OUTER JOIN`
  - `RIGHT OUTER JOIN`
  - `FULL OUTER JOIN`
- Cross joins

Creating Natural Joins

- The `NATURAL JOIN` clause is based on all the columns that have the same name in two tables.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned.

```
SELECT * FROM table1 NATURAL JOIN table2;
```

INNER versus OUTER Joins

- In SQL:1999, the join of two tables returning only matched rows is called an `INNER` join
- A join between two tables that returns the results of the `INNER` join – as well as the unmatched rows from the left (or right) table – is called a left (or right) `OUTER` join
- A join between two tables that returns the results of an `INNER` join – as well as the results of a left and right join – is a full `OUTER` join

Creating CROSS Joins

- A `CROSS JOIN` is a `JOIN` operation that produces the Cartesian product of two tables
- To create a Cartesian product, specify the `CROSS JOIN` in your `SELECT` statement

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration

...

158	Vargas	Contracting
159	Whalen	Contracting
160	Zlotkey	Contracting

Using the HAVING Clause with Subqueries

- The Oracle server executes the subqueries first

```
SELECT    department_id, MIN(salary)
FROM      employees
GROUP BY  department_id
HAVING    MIN(salary) >
                        (SELECT MIN(salary)
                         FROM   employees
                         WHERE  department_id = 30);
```

	DEPARTMENT_ID	MIN(SALARY)
1	100	6900
2	(null)	7000
3	90	17000
4	20	6000
5	70	10000
6	110	8300
7	80	6100
8	40	6500
9	60	4200
10	10	4400

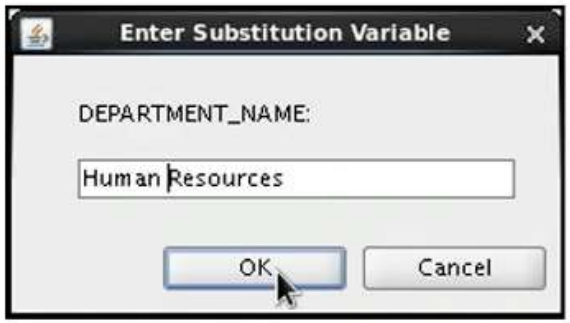
Data Manipulation Language

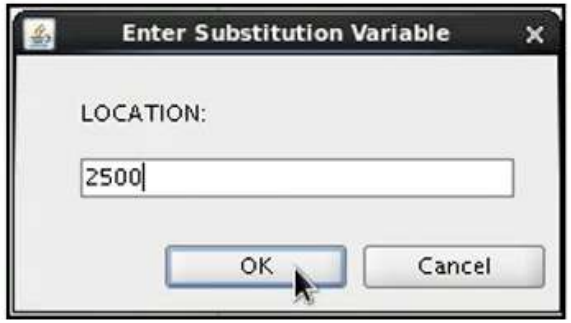
- A DML statement is executed when you
  - add new rows to a table
  - modify existing rows in a table
  - remove existing rows from a table
- A transaction consists of a collection of DML statements that form a logical unit of work

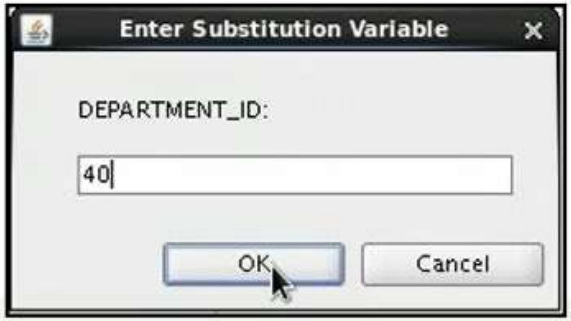
Creating a Script

- Use the ampersand (&) substitution in a SQL statement to prompt for values
- Ampersand (&) is a placeholder for the variable value

```
INSERT INTO departments
            (department_id, department_name, location_id)
VALUES      (&department_id, '&department_name',&location);
```







UPDATE Statement Syntax

- Use the UPDATE statement to modify existing values in a table

```
UPDATE        table
SET           column = value [, column = value, ...]
[WHERE        condition];
```

Deleting Rows from a Table

- Specific rows are deleted if you specify the `WHERE` clause

```
DELETE FROM departments
WHERE  department_name = 'Finance';
```

```
1 row deleted
```

- All rows in the table are deleted if you omit the `WHERE` clause

```
DELETE FROM  copy_emp;
```

```
22 rows deleted
```

The TRUNCATE Statement

Press Esc to exit full screen

- The `TRUNCATE` statement removes all rows from a table, leaving the table empty and the table structure intact
- The `TRUNCATE` statement is a data definition language (DDL) statement rather than a DML statement; it cannot easily be undone
- Syntax

```
TRUNCATE TABLE table_name;
```

Example

```
TRUNCATE TABLE copy_emp;
```

Database Transactions

- A database transaction consists of one of three elements
  - DML statements that constitute one consistent change to the data
  - One DDL statement
  - One data control language (DCL) statement

Database Transactions: Start and End

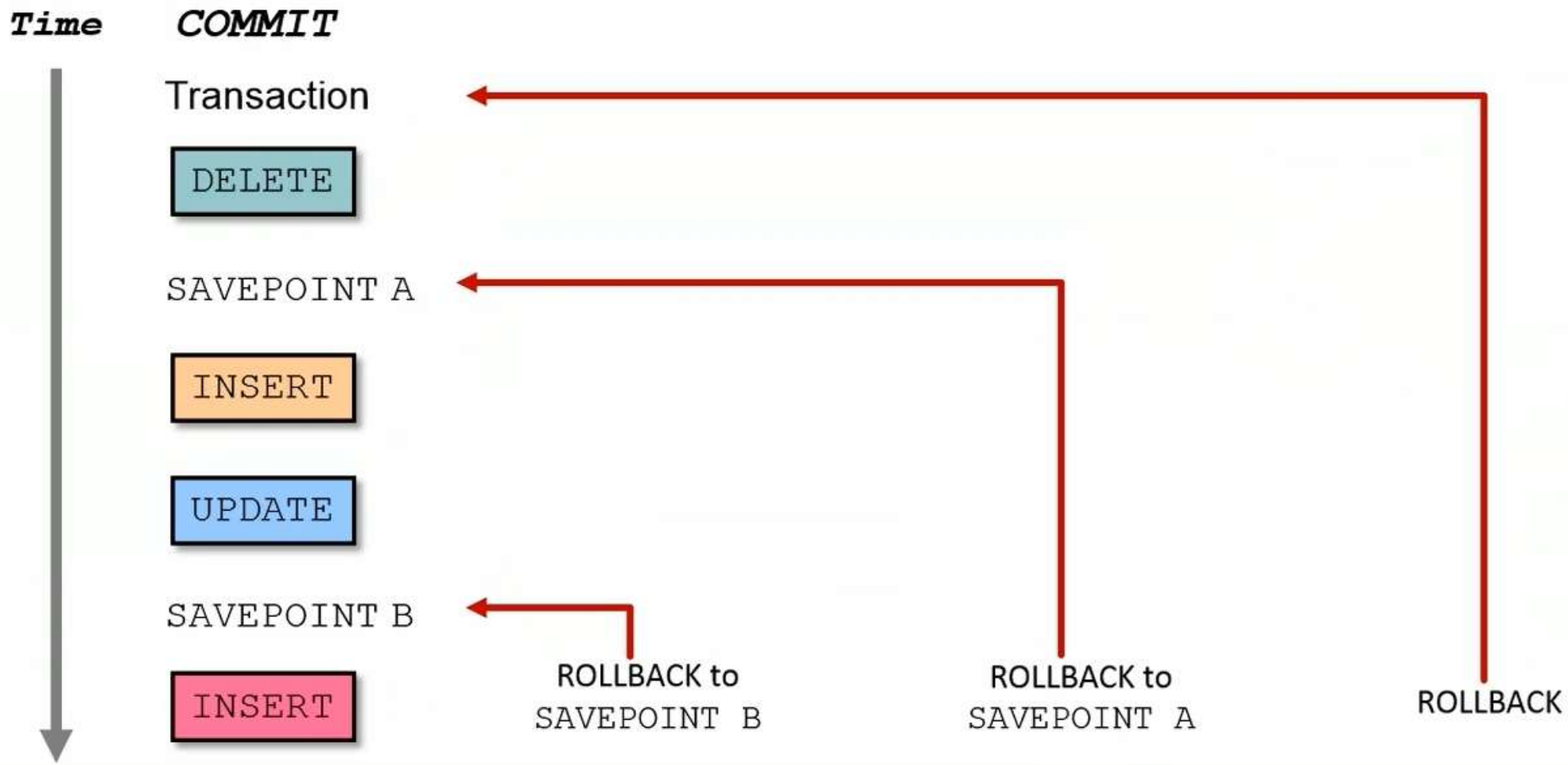
- Starts when the first DML SQL statement is executed

- Ends with one of the following events
  - A `COMMIT` or `ROLLBACK` statement is issued
  - A DDL statement such as CREATE is issued
  - A DDL or DCL statement executes (automatic commit)
  - The user exits SQL Developer or SQL*Plus
  - The system crashes

Advantages of COMMIT and ROLLBACK Statements

- With `COMMIT` and `ROLLBACK` statements, you can
  - ensure data consistency
  - preview data changes before making changes permanent
  - group logically related operations

Explicit Transaction Control Statements



Rolling Back Changes to a Marker

- Create a marker in the current transaction using the `SAVEPOINT` statement
- Roll back to that marker using the `ROLLBACK TO SAVEPOINT` statement

```
UPDATE...
SAVEPOINT update_done;
SAVEPOINT update_done succeeded
INSERT...
ROLLBACK TO update_done;
ROLLBACK TO succeeded
```

Implicit Transaction Processing

- An automatic commit occurs under the following circumstances
  - A DDL statement issued
  - A DCL statement issued
  - Normal exit from SQL Developer or SQL*Plus, without explicitly issuing `COMMIT` or `ROLLBACK` statements

- An automatic rollback occurs when there is an abnormal termination of SQL Developer or SQL*Plus or a system failure

State of Data Before COMMIT or ROLLBACK

- The previous state of the data can be recovered
- The current session can review the results of the DML operations using the `SELECT` statement
- Other sessions cannot view the results of the DML statements issued by the current session
- The affected rows are locked; other sessions cannot change the data in the affected rows

State of Data After COMMIT

- Data changes are saved in the database
- The previous state of the data is overwritten
- All sessions can view the results
- Locks on the affected rows are released; those rows are available for other sessions to manipulate
- All savepoints are erased

State of Data Before COMMIT or ROLLBACK

- The previous state of the data can be recovered
- The current session can review the results of the DML operations using the `SELECT` statement
- Other sessions cannot view the results of the DML statements issued by the current session
- The affected rows are locked; other sessions cannot change the data in the affected rows

State of Data After ROLLBACK

- Use the `ROLLBACK` statement to discard all pending changes
  - Data changes are undone
  - Previous state of the data is restored
  - Locks on the affected rows are released

```
DELETE FROM copy_emp;
ROLLBACK ;
```

Statement-level Rollback

- If a single DML statement fails during execution, only that statement is rolled back
- The Oracle server implements an implicit savepoint
- All other changes are retained
- Execute a `COMMIT` or `ROLLBACK` statement to terminate transactions explicitly

Read Consistency

- Read consistency guarantees a consistent view of the data at all times
- Changes made by one user do not conflict with the changes made by another user
- Read consistency ensures that, on the same data
  - readers do not wait for writers
  - writers do not wait for readers
  - writers wait for writers

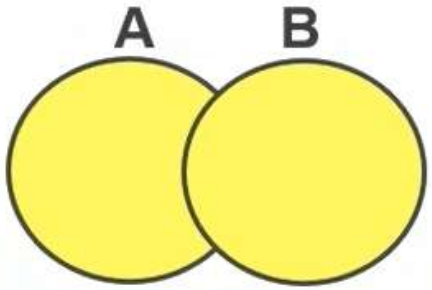
Using the FOR UPDATE Clause in a SELECT Statement

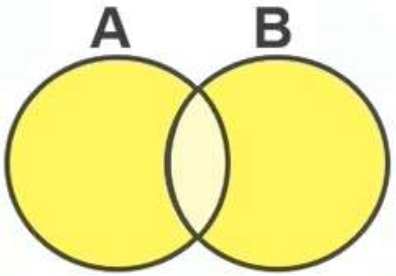
- Locks the rows in the EMPLOYEES table where job_id is SA_REP

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
FOR UPDATE
ORDER BY employee_id;
```

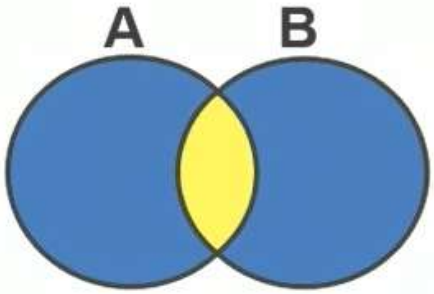
- The lock is released only when you issue a ROLLBACK or a COMMIT
- If the SELECT statement attempts to lock a row that is locked by another user, the database waits until the row is available, and then returns the results of the SELECT statement

Set Operators

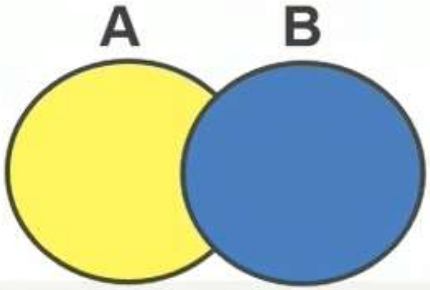




UNION/UNION ALL



INTERSECT



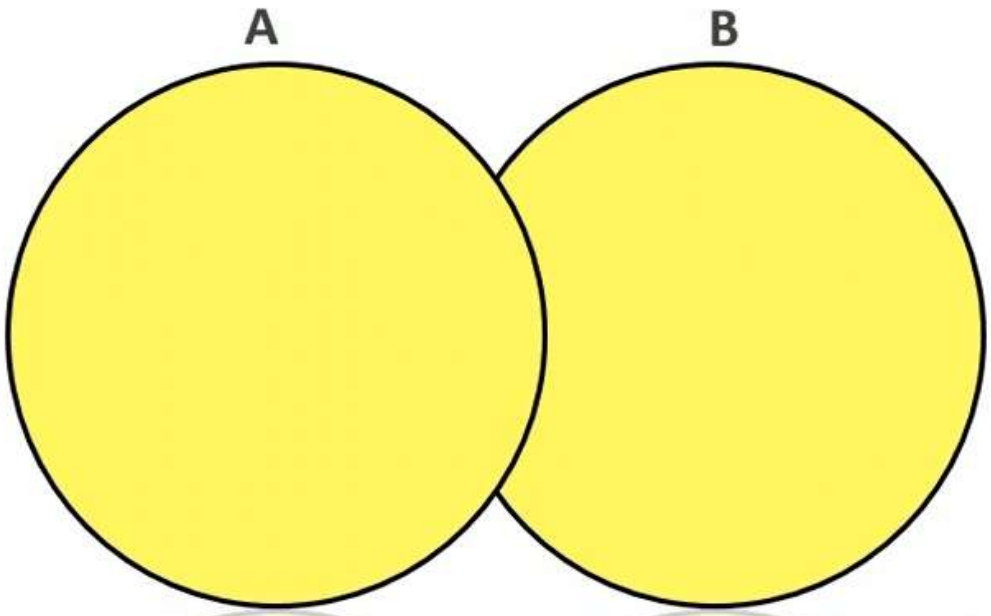
MINUS

Oracle Server and Set Operators

- Duplicate rows are automatically eliminated except in `UNION ALL`
- Column names from the first query appear in the result
- The output is sorted in ascending order by default, except in `UNION ALL`

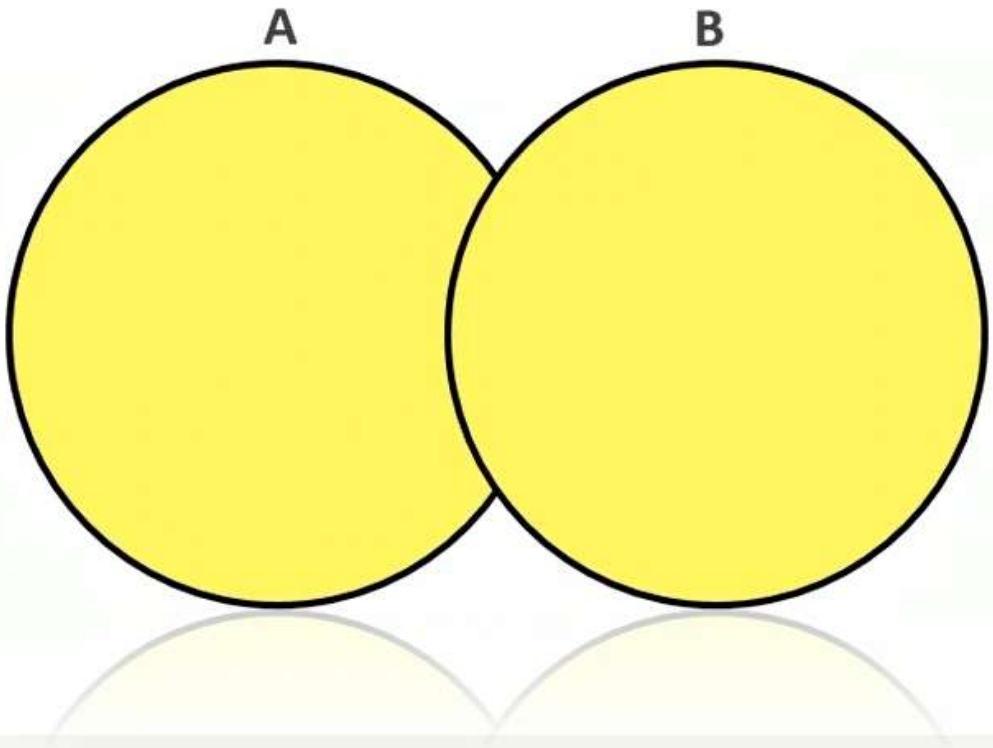
The UNION Operator

- The `UNION` operator returns rows from both queries after eliminating duplications



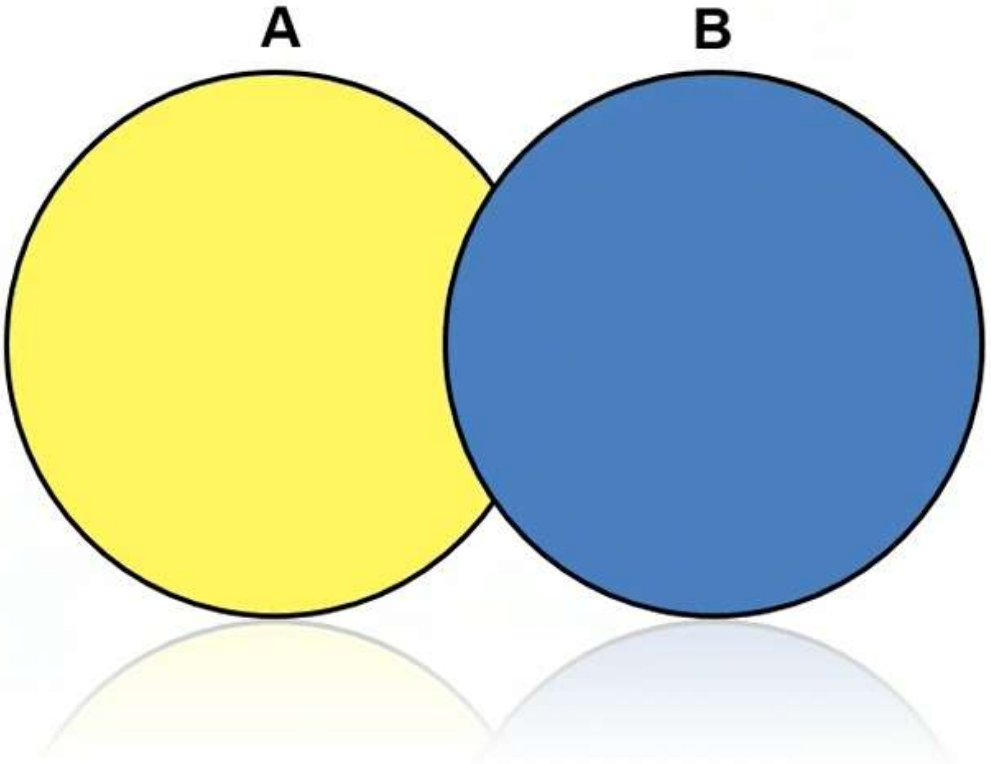
The UNION ALL Operator

- The `UNION ALL` operator returns rows from both queries, including all duplications



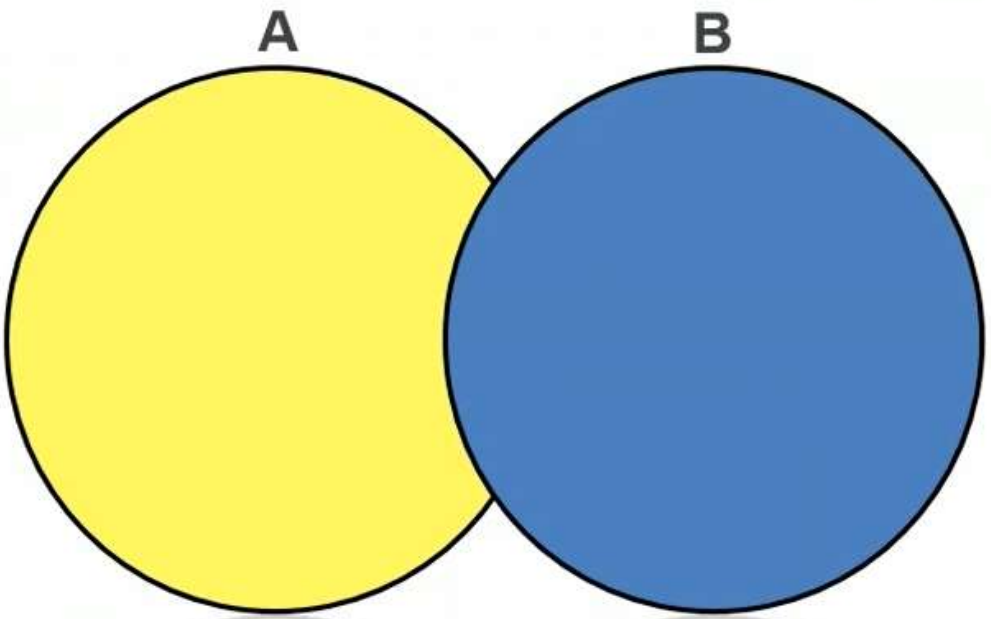
The INTERSECT Operator

- The `INTERSECT` operator returns rows that are common to both queries



The MINUS Operator

- The `MINUS` operator returns all the distinct rows selected by the first query, but not present in the second query result set



Using the ORDER BY Clause in Set Operations

- The `ORDER BY` clause can appear only once at the end of the compound query
- Component queries cannot have individual `ORDER BY` clauses
- The `ORDER BY` clause only recognizes the columns of the first `SELECT` query
- By default, the first column of the first `SELECT` query is used to sort the output in an ascending order

Database Objects

Object	Description
Table	The basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives an alternative name to an object

The CREATE TABLE Statement

- You must have
  - the CREATE TABLE privilege
  - a storage area

```
CREATE TABLE [schema.]table
            (column datatype [DEFAULT expr][, ...]);
```

- You specify
  - the table name
  - the column name, column data type, and column size

Data Types

Data Type	Description
VARCHAR2(size)	Variable-length character data
CHAR(size)	Fixed-length character data
NUMBER(p, s)	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data (up to 2 GB)
CLOB	Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE).
RAW and LONG RAW	Raw binary data
BLOB	Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE initialization parameter (8 TB to 128 TB)).
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

Including Constraints

- Constraints enforce rules at the table level
- Constraints ensure the consistency and integrity of the database
- The following constraint types are valid
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK

The FOREIGN KEY Constraint: Keywords

- `FOREIGN KEY`: Defines the column in the child table at the table-constraint level
- `REFERENCES`: Identifies the table and column in the parent table
- `ON DELETE CASCADE`: Deletes the dependent rows in the child table when a row in the parent table is deleted
- `ON DELETE SET NULL`: Converts dependent foreign key values to null

The ALTER TABLE Statement

- Use the `ALTER TABLE` statement to
  - add a new column
  - modify an existing column definition
  - define a default value for the new column
  - drop a column
  - rename a column
  - change a table to read-only status

The SET UNUSED Option

- Use the SET UNUSED option to mark one or more columns as unused
- Use the DROP UNUSED COLUMNS option to remove the columns marked as unused
- Specify the ONLINE keyword to indicate that DML operations on the table will be allowed while marking the column or columns UNUSED

```
ALTER TABLE <table name>
SET   UNUSED(<column_name> [ , <column_name>]);
OR
ALTER TABLE  <table name>
SET   UNUSED COLUMN <column_name> [,<column_name>];
```

```
ALTER TABLE <table_name>
DROP  UNUSED COLUMNS;
```

Read-Only Tables

- You can use the `ALTER TABLE` syntax to
  - put a table in read-only mode, which prevents DDL or DML changes during table maintenance
  - put a table back into read/write mode

```
ALTER TABLE employees READ ONLY;

-- perform table maintenance and then
-- return table back to read/write mode

ALTER TABLE employees READ WRITE;
```

Dropping a Table

- Moves a table to the recycle bin
- Removes the table and all its data entirely if the `PURGE` clause is specified
- Invalidates dependent objects and removes object privileges on the table

```
DROP TABLE dept80;
```

```
Table DEPT80 dropped
```