Database Management

Slides

Organization of the Course

- Subject n°1 during the two first sessions
- Only “sql_tutorial.pdf” is needed for now
- Quizzes will be used at the beginning of the next sessions to evaluate your knowledge
- Final exam at the end

The Tools

- Introduction

How to install Oracle Express ?

- Download the software from www.oracle.com
  - **Oracle Database Express Edition 11g Release 2**
    - **Windows x64**
    - **Windows x32**
    - **Linux x64**
- Unzip the file
- Run the DISK1\setup.exe
- Choose a password for SYS & SYSTEM users

How to install SQL Developer ?

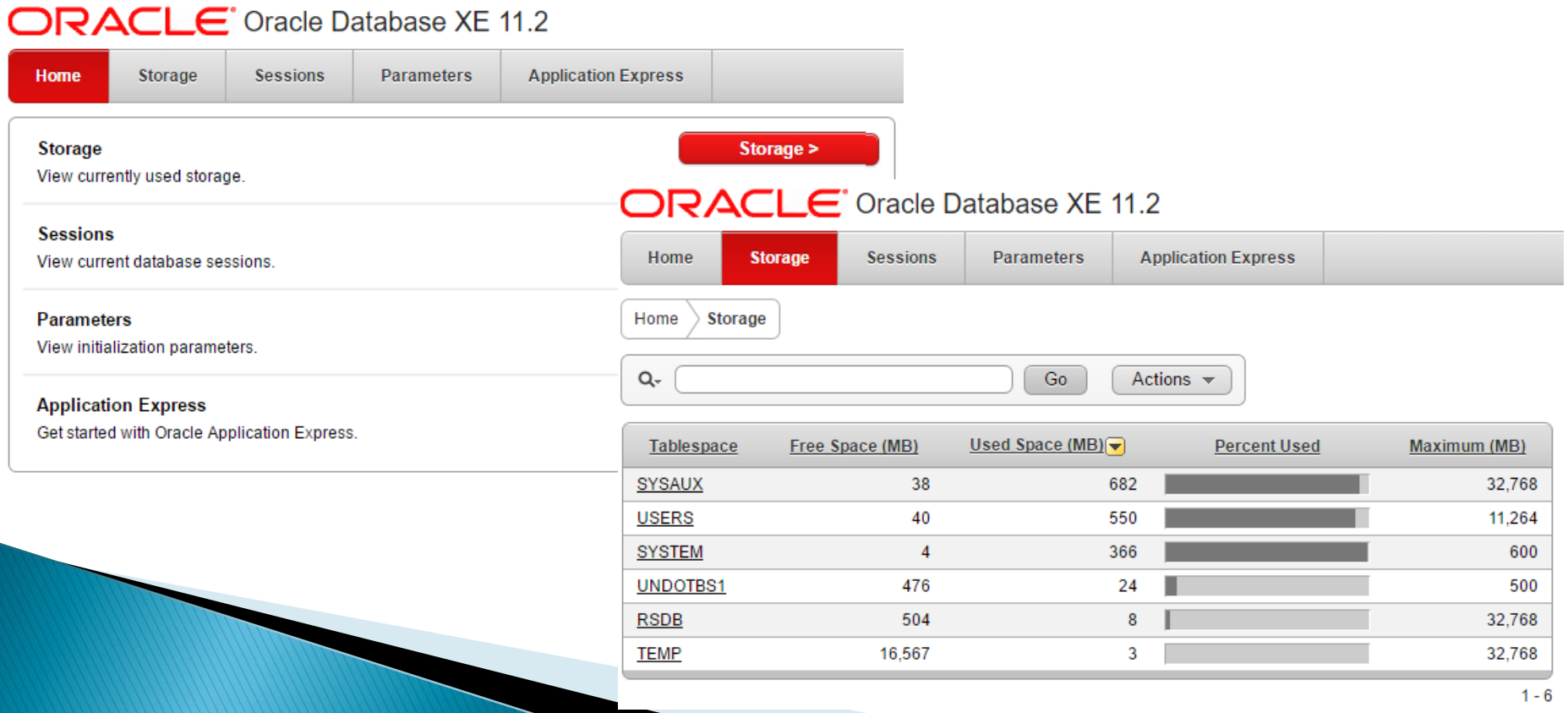
- Download the software from www.oracle.com
  - SQL Developer 18.2
    - Windows 64-bit with JDK 8 included
    - Windows 32-bit/64-bit
    - Linux RPM
- Unzip the file in a destination folder
- Run the sqldeveloper\sqldeveloper.exe

Configure Oracle Services

- Open the “Services” Control Panel
- Select each one of the following services :
  - OracleXETNSListener
  - OracleServiceXE
- Set the “startup type” to “manual”
- Start the service (if necessary)

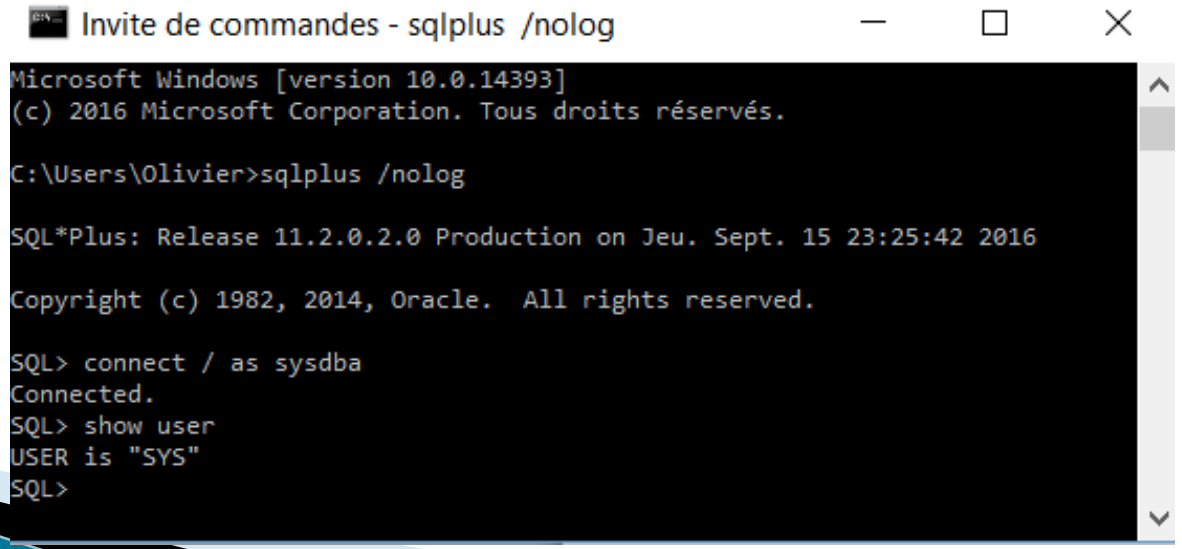
Oracle Express DB Control

- Open a Web Browser
- Use the following URL : http://127.0.0.1:8080/apex/f?p=4950



Oracle SQL*Plus

- Open a Windows Shell (cmd)
- Run SQL*Plus in nolog mode :
  - sqlplus /nolog
- Connect as SYS
  - connect / as SYSDBA
- Use SQL*Plus to administer the database



DataBase Management System

- Introduction

What is a DBMS ?

- A **D**ata**B**ase **M**anagement **S**ystem is a software for creating and managing databases.
- Its main functions are :
  - Data Storage Management
  - Data Dictionary Management
  - Data Transformation and Presentation
  - Data Integrity Management
  - Security Management
  - Multi-user Access Control
  - Backup and Recovery Management
  - Database Access Languages and APIs

Types of DBMS

- Flat Files
- Hierarchical
- Network
- Relational
- Object-Oriented
- Object-Relational
- XML
- NoSQL
- NewSQL

What about Oracle ?

- v7 : RDBMS
- v8i : ORDBMS
- v9i : + XML
- v12c : + JSON

Relational Data Model

- The basis

Relational Data Model

- Invented by CODD in 1970
- It defines :
  - Data Representation
  - Integrity Constraints
  - Normal Forms
  - Algebraic Operations

Data Representation

- A Relation and its attributes :
  - EMPLOYEES (id, first_name, last_name, hire_date)

- Each attribute is defined by a domain :
  - id : number [1000; 2000]
  - first_name : characters string (30)
  - last_name : characters string (30)
  - hire_date : date

Integrity Constraints

- Define rules to avoid data inconsistancy
- Several kinds of constraints :
  - PRIMARY KEY
    - Uniquely identify each row
    - All columns of the primary key are mandatory
  - UNIQUE KEY
    - Secondary unique constraints
    - Some columns may be optionnal
  - FOREIGN KEY
    - Ensures the consistancy between foreign columns and the referenced primary key columns
  - CHECK
    - Other checks about data in one or several columns of the same row

First Normal Form

- Rules :
  - Define the data items required for each relation
  - Ensure that there is no repeating groups of data
  - Define the primary key for each relation
- Example :

1NF

CUSTOMERS

Id	Name	Phone
101	Charles	316-636-5555
102	Barbara	316-111-1234 316-689-5555

CUSTOMERS

Id	Name
101	Charles
102	Barbara

PHONE_NUMBERS

Id	Phone
101	316-636-5555
102	316-111-1234
102	316-689-5555

Second Normal Form

- Rules :
  - Respect the 1NF
  - There must be no partial dependences of any of the primary key attributes
- Example :

1NF 2NF 2NF

ADDRESSES

Street	Zip	City
123 Main St.	67226	Wichita
237 Ash Ave.	67226	Wichita
111 Inwood St.	60606	Fort Dodge

ADDRESSES

Street	Zip
123 Main St.	67226
237 Ash Ave.	67226
111 Inwood St.	60606

CITIES

Zip	City
67226	Wichita
60606	Fort Dodge

Third Normal Form

- Rules :
  - Respect the 2NF
  - All non primary key attributes are dependent on the primary key
- Example :

2NF 3NF 3NF

EMPLOYEE

EmpId	Name	DptId	DptName
1000	Jones	101	Sales
1001	Smith	101	Sales
1002	Brown	102	Marketing

EMPLOYEE

EmpId	Name	DptId
1000	Jones	101
1001	Smith	101
1002	Brown	102

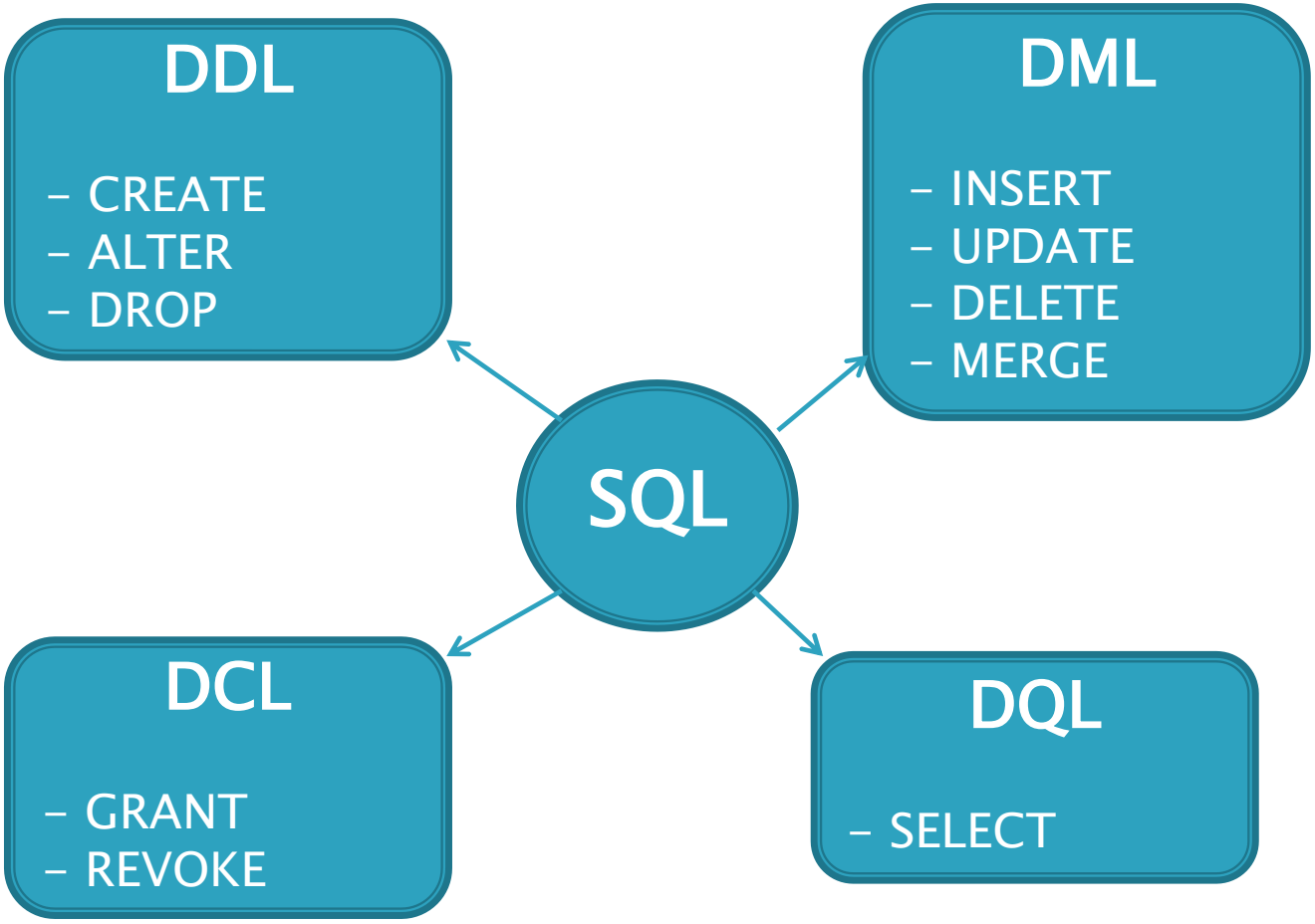
DEPARTMENT

DptId	Name
101	Sales
102	Marketing

SQL

- Structured Query Language

SQL : Commands



SQL : The SELECT Clause

- Definition :
  - Defines each column to be retrieved (projection)
- Examples :

```
SELECT *
  FROM departments
```

DPT_ID	NAME
10	Executive
20	Sales
30	Research
40	Finance

```
SELECT name AS Dpt_Name
  FROM departments
```

DPT_NAME
Executive
Sales
Research
Finance

SQL : The FROM Clause

- Definition :
  - Defines the data source (Table, View or Synonym)
- Examples :

```
SELECT *
  FROM departments
```

DPT_ID	NAME
10	Executive
20	Sales
30	Research
40	Finance

```
SELECT DPT.name
  FROM departments DPT
```

NAME
Executive
Sales
Research
Finance

SQL : The WHERE Clause (1 /2)

- Definition :
  - Defines the conditions to select the rows
- Examples :

```
SELECT *
  FROM departments
 WHERE name = 'Sales'
```

DPT_ID	NAME
20	Sales

```
SELECT dpt_id, name
  FROM departments
 WHERE dpt_id > 10
   AND name LIKE '%E%E%'
```

DPT_ID	NAME
30	Research

SQL : The WHERE Clause (2/2)

- Examples :

```
SELECT *
  FROM departments
 WHERE dpt_id != 20
   AND dpt_id BETWEEN 10 AND 40
```

DPT_ID	NAME
10	Executive
30	Research
40	Finance

```
SELECT name
  FROM departments
 WHERE name > 'R'
    OR name IN ('Sales', 'Finance')
```

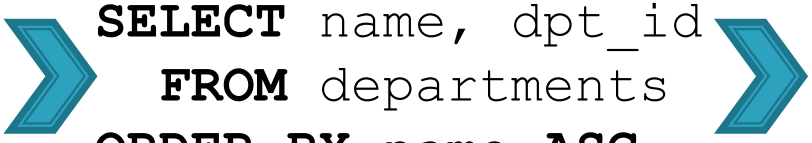
NAME
Sales
Research
Finance

SQL : Columns and Rows Order

- Definition :
  - Columns and Rows are not ordered in a table
- Example :

DEPARTMENTS (storage)

DPT_ID	NAME
10	Executive
20	Sales
30	Research
40	Finance



```
SELECT name, dpt_id
  FROM departments
ORDER BY name ASC
```

DEPARTMENTS (result)

NAME	DPT_ID
Executive	10
Finance	40
Research	30
Sales	20

SQL : Aggregates (1 /4)

- Definition :
  - A way to compute aggregates on numeric data
- Functions :
  - SUM
  - AVG
  - MIN
  - MAX
  - COUNT
- Note :
  - The query returns less rows

SQL : Aggregates (2/4)

- Globally on all rows :

EMPLOYEES

EMP_ID	LNAME	DPT_ID	SALARY
100	King	10	24000
101	Austin	20	12000
103	Chen	20	7500

```
SELECT SUM(salary)
  FROM employees
```

SUM
43500

SQL : Aggregates (3/4)

- Locally on groups of rows :

EMPLOYEES

EMP_ID	LNAME	DPT_ID	SALARY
100	King	10	24000
101	Austin	20	12000
103	Chen	20	7500

GROUP BY

DPT_ID	SALARY	
10	24000	G1
20	12000	G2
20	7500	

```
SELECT dpt_id,
       SUM(salary)
  FROM employees
GROUP BY dpt_id
```

DPT_ID	SUM
10	24000
20	19500

SQL : Aggregates (4/4)

- Apply a restriction on the aggregate result :

EMPLOYEES

EMP_ID	LNAME	DPT_ID	SALARY
100	King	10	24000
101	Austin	20	12000
103	Chen	20	7500

GROUP BY

DPT_ID	SALARY	
10	24000	G1
20	12000	G2
20	7500	

HAVING

DPT_ID	SUM	
10	24000	G1
		G2

```
SELECT dpt_id,
       SUM(salary)
  FROM employees
GROUP BY dpt_id
HAVING SUM(salary) > 20000
```

SQL : Cartesian Product (1/2)

- Definition :
  - Associate each row of T1 with all rows of T2
- Old Syntax :

```
SELECT *
  FROM employees, departments
```

- New Syntax :

```
SELECT *
  FROM employees CROSS JOIN departments
```

SQL : Cartesian Product (2/2)

- Example :

EMPLOYEES

EMP_ID	LNAME	DPT_ID
100	King	10
101	Austin	20
103	Chen	20

DEPARTMENTS

DPT_ID	NAME
10	Executive
20	Sales
30	Research

3x3 rows

CROSS JOIN 3+2 columns

EMP_ID	LNAME	DPT_ID	DPT_ID	NAME
100	King	10	10	Executive
101	Austin	20	10	Executive
103	Chen	20	10	Executive
100	King	10	20	Sales
101	Austin	20	20	Sales
103	Chen	20	20	Sales
100	King	10	30	Research
101	Austin	20	30	Research
103	Chen	20	30	Research

SQL : Join (1 / 6)

- Definition :
  - Restrict the rows of a product cartesian by applying a join condition
- Old Syntax :

```
SELECT *
  FROM employees E, departments D
 WHERE E.dpt_id = D.dpt_id
```

- New Normalized Syntax :

```
SELECT *
  FROM employees E JOIN departments D
       ON (E.dpt_id = D.dpt_id)
```

SQL : Join (2/6)

- Example :

EMPLOYEES

EMP_ID	LNAME	DPT_ID
100	King	10
101	Austin	20
103	Chen	20

DEPARTMENTS

DPT_ID	NAME
10	Executive
20	Sales
30	Research

EMPLOYEES E JOIN DEPARTMENTS D
ON (E.DPT_ID = D.DPT_ID)

EMP_ID	LNAME	DPT_ID		DPT_ID	NAME
100	King	10	=	10	Executive
		20	!=	10	
		20	!=	10	
		10	!=	20	
101	Austin	20	=	20	Sales
103	Chen	20	=	20	Sales
		10	!=	30	
		20	!=	30	
		20	!=	30	

SQL : Join (3/6)

- New Syntax Options :

```
SELECT *
  FROM employees E JOIN departments D
                   ON (E.dpt_id = D.dpt_id)

SELECT *
  FROM employees JOIN departments
                 USING(dpt_id)

SELECT *
  FROM employees NATURAL JOIN departments
```

SQL : Join (4/6)

- Kinds of Joins :
  - INNER :

```
SELECT *
  FROM employees INNER JOIN departments
                 USING(dpt_id)
```

  - OUTER :

```
SELECT *
  FROM employees LEFT OUTER JOIN departments
                 USING(dpt_id)
SELECT *
  FROM employees RIGHT OUTER JOIN departments
                 USING(dpt_id)
SELECT *
  FROM employees FULL OUTER JOIN departments
                 USING(dpt_id)
```

SQL : Join (5/6)

- Outer Join Example :

EMPLOYEES

EMP_ID	LNAME	DPT_ID
100	King	10
101	Austin	20
103	Chen	20

DEPARTMENTS

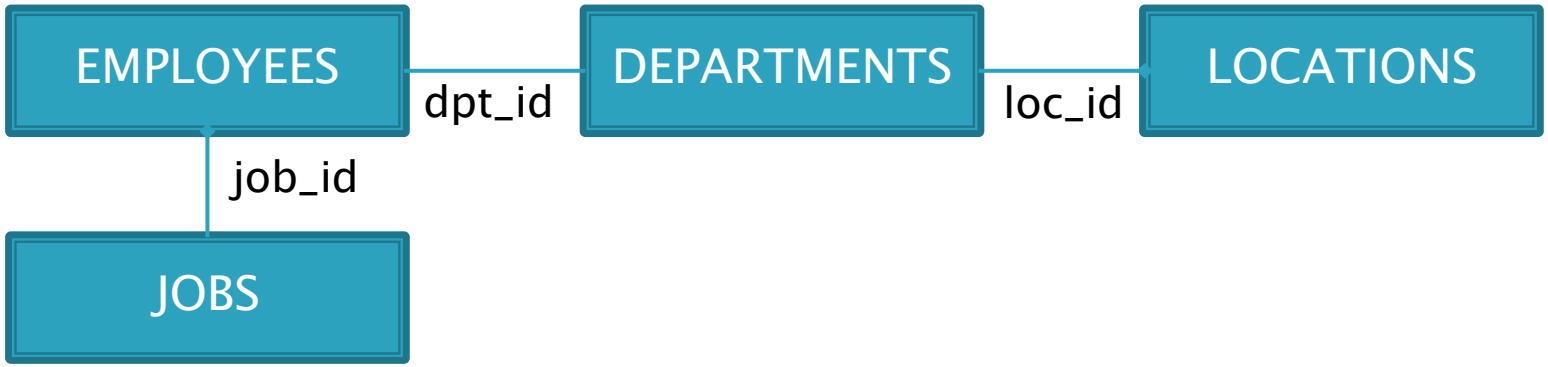
DPT_ID	NAME
10	Executive
20	Sales
30	Research

EMPLOYEES E RIGHT OUTER JOIN DEPARTMENTS D
ON (E.DPT_ID = D.DPT_ID)

EMP_ID	LNAME	DPT_ID		DPT_ID	NAME
100	King	10	=	10	Executive
		20	!=	10	
		20	!=	10	
		10	!=	20	
101	Austin	20	=	20	Sales
103	Chen	20	=	20	Sales
		10	!=	30	
		20	!=	30	
		20	!=	30	
NULL	NULL	NULL		30	Research

SQL : Join (6/6)

- You can join more than 2 tables together :
  - Data Model :



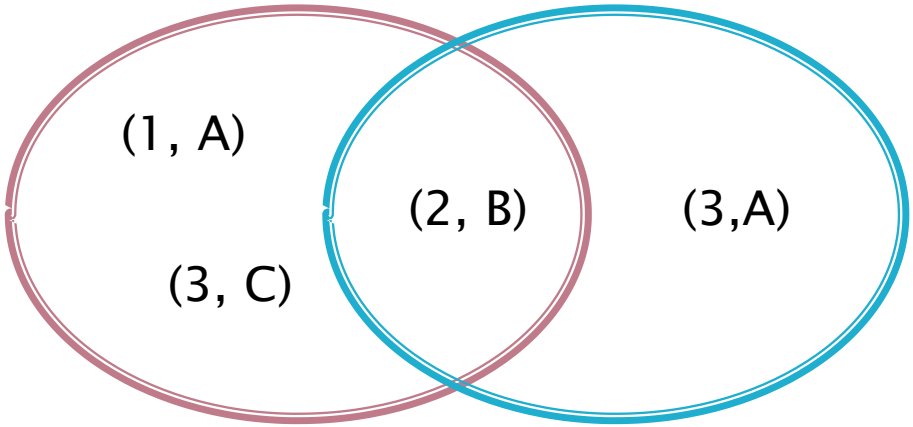
  - Name and function of employees working in Chicago

```
SELECT name, function
  FROM employees JOIN departments USING(dpt_id)
                 JOIN locations USING (loc_id)
                 JOIN jobs USING (job_id)
 WHERE city = 'CHICAGO'
```

SQL : Set Operators (1 / 5)

- An other way to combine 2 data sets
- General syntax :

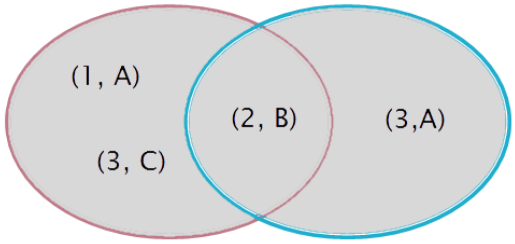
```
SELECT c1, c2
  FROM t1
<SET OPERATOR>
SELECT c1, c2
  FROM t2
```



- Rules
  - Same number of expressions in all subqueries
  - Each expression has the same type in all subqueries
  - ORDER BY only allowed after the last subquery

SQL : Set Operators (2/5)

- UNION



T1

C1	C2
1	A
2	B
3	C

```
SELECT c1, c2
  FROM t1
UNION
SELECT ca, cb
  FROM t2
```

T2

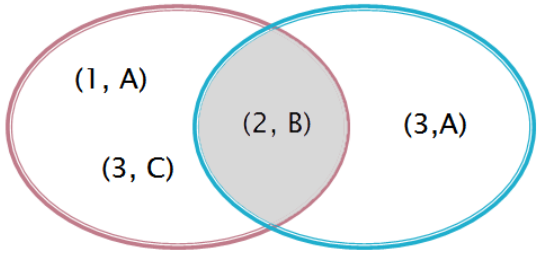
CA	CB
3	A
2	B

T1_∪_T2

C1	C2
1	A
2	B
3	C
3	A

SQL : Set Operators (3 / 5)

- INTERSECT



T1

C1	C2
1	A
2	B
3	C

```
SELECT c1, c2
  FROM t1
INTERSECT
SELECT ca, cb
  FROM t2
```

T2

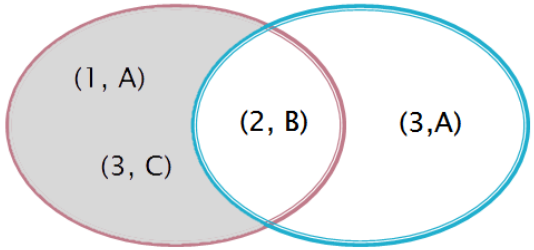
CA	CB
3	A
2	B

T1_∩_T2

C1	C2
2	B

SQL : Set Operators (4/5)

- MINUS



T1

C1	C2
1	A
2	B
3	C

```
SELECT c1, c2
  FROM t1
MINUS
SELECT ca, cb
  FROM t2
```

T2

CA	CB
3	A
2	B

T1_-_T2

C1	C2
1	A
3	C

SQL : Set Operators (5/5)

- A more complex use …

```
SELECT c1, c2
  FROM t1
MINUS
( SELECT ca, cb
    FROM t2
  INTERSECT
  SELECT d1, d2
    FROM t3 )
ORDER BY c1, c2
```

SQL : Nested Query (1 / 5)

- An other way to join data from 2 tables
- A subquery can be mainly defined in :
  - SELECT have to return only one value
  - FROM can be used as a new data source
  - WHERE generally used with EXISTS or IN
- Rule :
  - Data of a subquery inside a WHERE clause can't be used in the main query
- Advice :
  - Prefer using a join when it is possible

SQL : Nested Query (2/5)

- In the WHERE clause
  - Uncorrelated subquery :

```
SELECT lname
  FROM employees
 WHERE dpt_id IN (SELECT dpt_id
                    FROM departments
                   WHERE name = 'Research')
```

  - Correlated subquery :

```
SELECT lname
  FROM employees e
 WHERE EXISTS (SELECT 1
                 FROM departments
                WHERE name = 'Research'
                  AND dpt_id = e.dpt_id)
```

SQL : Nested Query (3/5)

- In the WHERE clause
  - Other set operators :

```
SELECT lname
  FROM employees
 WHERE salary > ALL (SELECT salary
                       FROM employees
                      WHERE dpt_id = 30)

SELECT lname
  FROM employees e
 WHERE salary <= ANY (SELECT salary
                        FROM employees
                       WHERE dpt_id = 30)
```

SQL : Nested Query (4/5)

- In the FROM clause

```
SELECT lname, revenue
  FROM (
    SELECT lname,
           salary + COALESCE(commission,0) AS revenue
      FROM employees
     WHERE lname LIKE 'A%'
       )
ORDER BY lname
```

SQL : Nested Query (5/5)

- In the WITH clause
  - The best way to write complex queries !

```
WITH
   -- First subquery on Employees table
   emp AS (SELECT dpt_id, lname
             FROM employees
            WHERE salary > 2000),
   -- Second subquery on Departments table
   dpt AS (SELECT dpt_id, name
             FROM departments
            WHERE name = 'Research')
-- Now, the main query can use both subqueries
SELECT *
   FROM emp INNER JOIN dpt USING (dpt_id)
```

SQL : Analytic Functions (1 / 5)

- Definition :
  - Another way to compute aggregates
- Functions :
  - SUM, AVG, MIN, MAX, COUNT
  - RANK, DENSE_RANK
  - RATIO_TO_REPORT
  - LEAD, LAG
  - FIRST_VALUE, LAST_VALUE
  - NTILE, PERCENTILE_DISC, PERCENTILE_CONT
  - …
- Note :
  - The query returns as many rows as the source table

SQL : Analytic Functions (2/5)

- General Syntax :

```
SELECT …,
       <function> OVER ([PARTITION BY c1, …]
                        [ORDER BY c2, …] )
  FROM t1
```

  - PARTITION BY : defines the groups of rows
  - ORDER BY : defines how to sort the rows
- Note :
  - This kind of expression can only be used in the SELECT clause

SQL : Analytic Functions (3/5)

- Example #1:

EMPLOYEES

EMP_ID	LNAME	DPT_ID	SALARY	SUM_SAL
100	King	10	24000	43500
101	Austin	20	12000	43500
103	Chen	20	7500	43500

```
SELECT employees.*,
       SUM(salary) OVER () AS sum_sal
  FROM employees
```

SQL : Analytic Functions (4/5)

- Example #2:

EMPLOYEES

EMP_ID	LNAME	DPT_ID	SALARY	SBD	RTR	RKT
100	King	10	24000	24000	0.55	1
101	Austin	20	12000	19500	0.28	2
103	Chen	20	7500	19500	0.17	3

```
SELECT employees.*,
       SUM(salary) OVER (PARTITION BY dpt_id) AS sbd,
       RATIO_TO_REPORT(salary) OVER () AS rtr,
       RANK() OVER (ORDER BY salary DESC) AS rkt
  FROM employees
```

SQL : Analytic Functions (5/5)

- Example #3:

EMPLOYEES

EMP_ID	LNAME	DPT_ID	SALARY	LGS	LDS	MSBD
100	King	10	24000	NULL	12000	24000
101	Austin	20	12000	24000	7500	7500
103	Chen	20	7500	12000	NULL	7500

```
SELECT employees.*,
       LAG(salary)  OVER (ORDER BY salary DESC) AS lgs,
       LEAD(salary) OVER (ORDER BY salary DESC) AS lds
       MIN(salary)  OVER (PARTITION BY dpt_id)  AS msbd
  FROM employees
```

Directory

- Definition :
  - A directory is a logical object that references a physical folder on the database host.
- Syntaxes :

```
CREATE DIRECTORY dir_esilv AS 'c:\temp\esilv';

DROP DIRECTORY dir_esilv;
```

- Privileges :

```
GRANT READ, EXECUTE ON DIRECTORY dir_esilv
   TO flights;
```

External Table (1 /2)

- Definition :
  - A table that references an external file containing the data.
- Rules :
  - The file must be located in an existing directory
  - Only for structured text files (csv, fixed)
  - Read only access

External Table (2/2)

- Example :

```
create table ext_aircraft   (
  aid              varchar2(30),
  aname            varchar2(30),
  cruisingrange varchar2(30))
organization external
( type oracle_loader
  default directory dir_esilv
  access parameters
  ( records delimited by newline
    skip 1
    fields terminated by ';'
  )
  location ('aircraft.csv')
)
reject limit unlimited;
```

TD n°2

- e) Find the names of pilots certified for some Boeing aircraft.
- h) Print the enames of pilots who can operate planes with cruising range greater than 3000 miles but are not certified on any Boeing aircraft.
- c) Find the names of pilots whose salary is less than the price of the cheapest route from Los Angeles to Honolulu.
- f) Find the “aids” of all aircraft that can be used on routes from Los Angeles to Chicago.
- d) For all aircrafts with cruising range over 1000 miles, find the name of the aircraft and the average salary of all pilots certified for this aircraft.

TD n°2 : Flights

- j) Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots).
- k) Print the name and salary of every non pilot whose salary is more than the average salary for pilots.
- m) Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles, but on at least two such aircrafts.
- l) Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles.
- n) Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles and who are certified on some Boeing aircraft.

TD n°2 : Emp

- a) Print the names of each employee who works in both the IT department and the Research department.
- b) For each department with more than 3 full-time equivalent employees, print the did together with the number of employees that work in that department.
- c) Print the name of each employee whose salary exceeds the budget of all of the departments that he or she works in.
- d) Find the manager ids of managers who manage only departments with budgets greater than 100000.
- e) Find the names of managers who manage the departments with the largest budgets.