AGENDA

1. Constraints
2. DDL Operations
3. DML Statements with Subqueries, Analytics Functions

Data Science Academy

Constraints

Data Science Academy

Constrains

- Used to limit the type of data that can go into a table
- Can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement)
- Type of constraints:
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK
  - DEFAULT

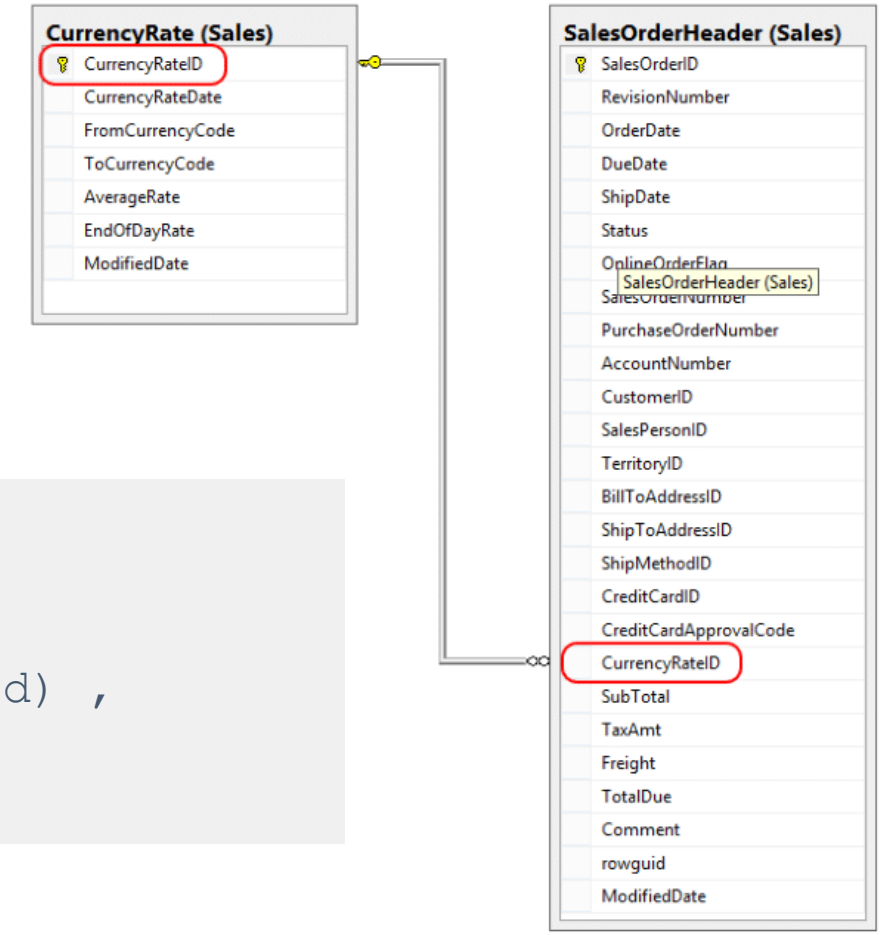
NOT NULL

- Enforce a column to not accept NULL values

```
CREATE TABLE Persons
(
P_Id          INT NOT NULL,
LastName      varchar(255) NOT NULL,
FirstName     varchar(255),
Address        varchar(255),
City          varchar(255)
);
```

Primary Key

- A table contains a primary key that uniquely identifies a row of data. Each record must have a distinct value of primary key. The primary key is used to relate (join) tables. The Primary key consists of one or more columns whose data contained within is used to uniquely identify each row in the table.




```
1 CREATE TABLE regions(
2 region_id       NUMBER
3 CONSTRAINT region_id_nn NOT NULL,
4 CONSTRAINT reg_id_pk PRIMARY KEY (region_id) ,
5 region_name     VARCHAR2(25)
6 )
```

Foreign Key

- **A foreign key** is a column or group of columns in a relational database table that provides a link between data in two tables.

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Foreign key

Foreign Key

- It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them.

```
CREATE TABLE countries    (
country_id       CHAR(2)
CONSTRAINT country_id_nn NOT NULL ,
CONSTRAINT country_c_id_pk
PRIMARY KEY (country_id) ,
country_name  VARCHAR2(40),
region_id NUMBER ,
CONSTRAINT countr_reg_fk
FOREIGN KEY (region_id)
REFERENCES regions (region_id))
```

UNIQUE

- Uniquely identifies each record in a database table
- UNIQUE and PRIMARY KEY both provide a guarantee for uniqueness for a column or set of columns
- A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

```
CREATE TABLE Persona
(
P_Id   INT NOT NULL,
LastName  varchar(255) NOT NULL,
FirstName  varchar(255),
Address      varchar(255),
City   varchar(255),
CONSTRAINT uc_PersonID UNIQUE (P_Id, LastName)
);
```

Check Constraint

- Used to limit the value range of a column

```
CREATE TABLE Personaa (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age>=18)
);
```

II

DDL Operations

CREATE TABLE

- To create a new table in Oracle Database, you use the `CREATE TABLE` statement. The following illustrates the basic syntax of the `CREATE TABLE` statement:

```
CREATE TABLE schema_name.table_name (
    column_1 data_type column_constraint,
    column_2 data_type column_constraint,
    ...
    table_constraint
);
```

Creating a Table

- The following example shows how to create a new table named persons :

```
1 CREATE TABLE customers
2 ( customer_id number(10) NOT NULL,
3   customer_name varchar2(50) NOT NULL,
4   city varchar2(50)
5 );
6
```

CREATE TABLE AS Statement

- You can also use the Oracle CREATE TABLE AS statement to create a table from an existing table by copying the existing table's columns.
- It is important to note that when creating a table in this way, the new table will be populated with the records from the existing table (based on the SELECT Statement).
- The syntax for the CREATE TABLE AS statement :

```
1 CREATE TABLE new_table AS (SELECT * FROM old_table);
```

- The syntax for the CREATE TABLE AS statement that copies the selected columns

```
1 CREATE TABLE customers_new
2     AS (SELECT customer_id, customer_name
3     FROM customers);
```

Introduction to the Oracle DROP TABLE Statement

- To move a table to the recycle bin or remove it entirely from the database, you as the `DROP TABLE` statement

```
1 DROP TABLE schema_name.table_name
2 [CASCADE CONSTRAINTS | PURGE];
```

For example :

```
1 DROP TABLE persons;
```

Introduction to the Oracle TRUNCATE TABLE Statement

- When you have a table with a large number of rows, using the DELETE statement to remove all data is not efficient. Oracle introduced the TRUNCATE TABLE statement that allows you to delete all rows from a big table.

  The following illustrates the syntax of the Oracle TRUNCATE TABLE statement:

```
1 TRUNCATE TABLE schema_name.table_name
2 [CASCADE]
3 [[ PRESERVE | PURGE] MATERIALIZED VIEW LOG ]]
4 [[ DROP | REUSE]] STORAGE ]
```

  For example:

```
1 TRUNCATE TABLE customers_copy;
```

Oracle ALTER TABLE ADD statement

- To modify the structure of an existing table, you use the ALTER TABLE statement.
- To add a new column to a table, the following illustrates the syntax of the ADD function.

```
1 ALTER TABLE table_name
2 ADD new_column TYPE CONSTRAINT;
```

For example:

```
1 ALTER TABLE customers
2 ADD birthdate DATE NOT NULL;
```

Oracle ALTER TABLE MODIFY statement

- To modify the attributes of the structure of an existing table,

```
1 ALTER TABLE table_name
2 MODIFY new_column TYPE CONSTRAINT;
```

For example:

```
1 ALTER TABLE customers
2 MODIFY birthdate DATE NULL;
```

Oracle ALTER TABLE DROP statement

- To remove an existing column from a table, you use the following syntax:

```
1 ALTER TABLE table_name
2 DROP COLUMN new_column ;
```

For example:

```
1 ALTER TABLE customers
2 DROP COLUMN birthdate ;
```

III

DML Statements with Subqueries, Analytics Functions

Data Science Academy

Oracle INSERT Statement

- To insert a new row into a table, you use the Oracle `INSERT` statement as follows:

```
1 INSERT INTO table_name (column_list)
2 VALUES( value_list);
```

- If the value list has the same order as the table columns, you can skip the column list although this is not considered as a good practice:

```
1 INSERT INTO table_name
2 VALUES (value_list);
```

Oracle INSERT Statement Example

- The following statement inserts a new row into the vutable:

```
1 INSERT INTO customers(customer_id,customer_name,city)
2 VALUES('12345','Andrea', 'Paris');
```

CUSTOMER_ID	CUSTOMER_NAME	CITY
12345	Andrea	Paris

INSERT With Subquery

- Overview of Oracle INSERT INTO SELECT statement. Sometimes, you want to select data from a table and insert it into another table. To do it, you use the Oracle INSERT INTO SELECT statement as follows. The Oracle INSERT INTO SELECT statement requires the data type of the source and target tables match. If you want to copy all rows from the source table to the target table, you remove the WHERE clause. Otherwise, you can specify which rows from the source table should be copied to the target table.

```
INSERT INTO target_table (col1, col2, col3)
SELECT col1,
       col2,
       col3
FROM source_table
WHERE condition;
```

INSERT Subquery - Example

- Copy data from the contacts table to the customer_lists table

```
1 create table
2 customers_new(customer_id
3 number(10) NOT NULL,
4 customer_name varchar2(50)
5 NOT NULL);
```

- In this example, in addition to retrieving data from the contacts table, we also used literal 0 as the value for the sent column.

```
1 INSERT INTO
2 customers_new(
3 customer_id,
4 customer_name)
5 SELECT
6 customer_id,
7 customer_name FROM customers;
```

CUSTOMER_ID	CUSTOMER_NAME
12345	Andrea

UPDATE Statement

- To changes existing values in a table, you use the following Oracle UPDATE statement:

```
UPDATE
    table_name
SET
    column1 = value1,
    column2 = value2,
    column3 = value3,
    ...
WHERE
    condition;
```

Warning: If you omit WHERE clause, the UPDATE statement will update all rows of the table.

UPDATE Statement Example

- Change Customer ID to ‘10000’ when Customer names start with ‘And’ in ‘Customers_new’ table

```
1 UPDATE
2 customers_new
3 SET
4  CUSTOMER_ID = '10000'
5 WHERE CUSTOMER_NAME LIKE 'And%';
```

CUSTOMER_ID	CUSTOMER_NAME	CITY
10000	Andrea	-

UPDATE Using Subqueries

- The syntax for the Oracle UPDATE statement when updating one table with data from another table is:

```
UPDATE table1
SET
 column1 = (SELECT expression1
            FROM table2
            WHERE conditions2)
[WHERE conditions1];
```

- Let's look at an Oracle UPDATE example that shows how to update a table with data from another table.

```
UPDATE customers_new
SET customer_id = (SELECT customer_id
FROM customers
WHERE customers.customer_name like 'And%');
```

DELETE Statement

- To delete one or more rows from a table, you use the Oracle DELETE statement as follows:

```
DELETE
FROM
    table_name
WHERE
    condition;
```

In this statement,

- First, you specify the name of the table from which you want to delete data.
- Second, you specify which row should be deleted by using the condition in the WHERE clause.
- If you omit the WHERE clause, Oracle DELETE statement removes all rows from the table.
- Note that it is faster and more efficient to use the TRUNCATE TABLE statement to delete all rows from the large table.

DELETE Statement Example

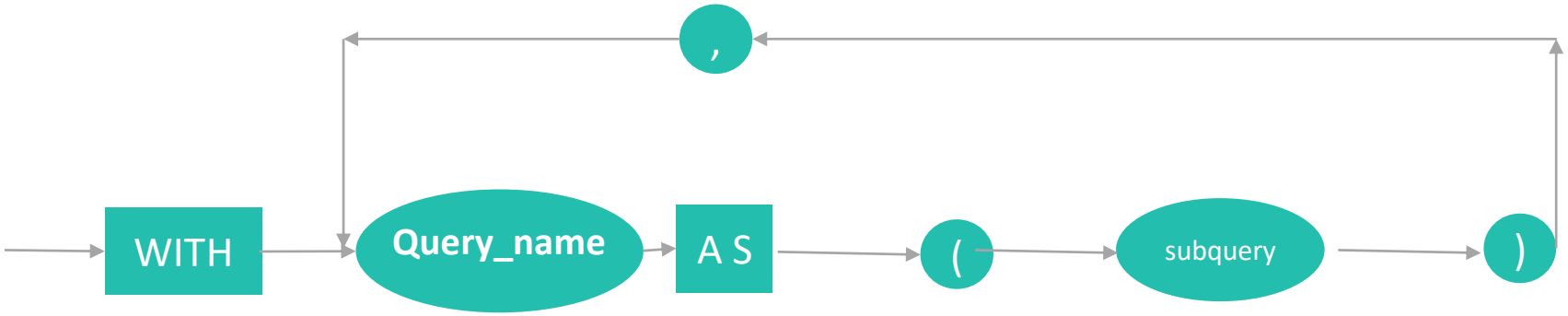
- Delete rows where Customer name starts with ‘And’ from ‘customers_new’ table

```
1 delete from customers_new where customer_name like 'And%'
```

SQL WITH Statement

- SQL WITH clause allows you to give a sub-query block a name (a process also called sub-query refactoring), which can be referenced in several places within the main SQL query.

```
CODE
```



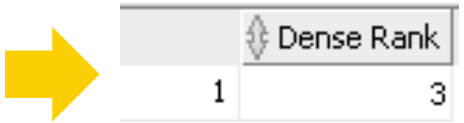
Functions – Analytics Functions

Name		Description
DENSE_RANK		Calculate the rank of a row in an ordered set of rows with no gaps in rank values.
FIRST_VALUE		Get the value of the first row in a specified window frame.
LAG		Provide access to a row at a given physical offset that comes before the current row without using a self-join.
LAST_VALUE		Get the value of the last row in a specified window frame.
LEAD		Provide access to a row at a given physical offset that follows the current row without using a self-join.
ROW_NUMBER		Assigns a unique number to each row to which it is applied

Oracle DENSE_RANK

- DENSE_RANK computes the rank of a row in an ordered group of rows and returns the rank as a NUMBER. The ranks are consecutive integers beginning with 1. The largest rank value is the number of unique values returned by the query. Rank values are not skipped in the event of ties. Rows with equal values for the ranking criteria receive the same rank. This function is useful for top-N and bottom-N reporting.

```
1 SELECT
2 DENSE_RANK (15500, 0.05)
3 WITHIN GROUP
4 (ORDER BY salary DESC, commission_pct) "Dense Rank"
5 FROM employees;
```



	Dense Rank
1	3

- Following statement displays analytic example of DENSE_RANK() function. It The following statement ranks the employees in the sample hr schema in department 60 based on their salaries. Identical salary values receive the same rank.

```
1 SELECT
2 department_id, last_name, salary,
3 DENSE_RANK()
4 OVER (PARTITION BY department_id
5 ORDER BY salary)
6 DENSE_RANK FROM employees
7 WHERE department_id = 60
  ORDER BY DENSE_RANK, last_name;
```

	DEPARTMENT_ID	LAST_NAME	SALARY	DENSE_RANK
1	60	Lorentz	4200	1
2	60	Austin	4800	2
3	60	Pataballa	4800	2
4	60	Ernst	6000	3
5	60	Hunold	9000	4

Oracle FIRST_VALUE

- FIRST_VALUE() returns the first value in an ordered set of values.

```
1 SELECT
2 department_id, last_name, salary,
3 FIRST_VALUE (last_name)
4 OVER (ORDER BY salary) AS lowest_sal
5 FROM employees
```

	DEPARTMENT_ID	LAST_NAME	SALARY	LOWEST_SAL
1	50	Olson	2100	Olson
2	50	Markle	2200	Olson
3	50	Philtanker	2200	Olson
4	50	Gee	2400	Olson
5	50	Landry	2400	Olson

Oracle LAST_VALUE

- LAST_VALUE returns the last value in an ordered set of values. If the last value in the set is null, then the function returns NULL unless you specify IGNORE NULLS. This setting is useful for data densification. If you specify IGNORE NULLS, then LAST_VALUE returns the fist non-null value in the set, or NULL if all values are null.
- The following example returns, for each row, the hire date of the employee earning the highest salary:

```
1 SELECT
2 last_name, salary, hire_date,
3 LAST_VALUE(hire_date)
4 OVER (ORDER BY salary)
5 FROM employees
```

	LAST_NAME	SALARY	HIRE_DATE	LAST_VALUE(
1	Olson	2100	10-APR-07	10-APR-07
2	Philtanker	2200	06-FEB-08	08-MAR-08
3	Markle	2200	08-MAR-08	08-MAR-08
4	Landry	2400	14-JAN-07	12-DEC-07
5	Gee	2400	12-DEC-07	12-DEC-07

Oracle LAG()

- LAG() provides access to more than one row of a table at the same time without a self join. Given a series of rows returned from a query and a position of the cursor, LAG provides access to a row at a given physical offset prior to that position
- The following example provides, for each salesperson in the employees table, the salary of the employee hired just before:

```
1 SELECT
2 last_name, hire_date, salary,
3 LAG(salary, 1, 0)
4 OVER (ORDER BY hire_date)
5 FROM employees
6 WHERE job_id = 'PU_CLERK';
```

	LAST_NAME	HIRE_DATE	SALARY	PREV_SAL
1	Khoo	18-MAY-03	3100	0
2	Tobias	24-JUL-05	2800	3100
3	Baida	24-DEC-05	2900	2800
4	Himuro	15-NOV-06	2600	2900
5	Colmenares	10-AUG-07	2500	2600

Oracle LEAD()

- LEAD() provides access to more than one row of a table at the same time without a self join. Given a series of rows returned from a query and a position of the cursor, LEAD provides access to a row at a given physical offset beyond that position.

- The following example provides, for each employee in Department 30 in the employees table, the hire date of the employee hired just after:

```
1 SELECT hire_date, last_name,
2 LEAD(hire_date, 1)
3 OVER (ORDER BY hire_date) AS "NextHired"
4 FROM employees
5 WHERE department_id = 30;
```

	HIRE_DATE	LAST_NAME	NextHired
1	07-DEC-02	Raphaely	18-MAY-03
2	18-MAY-03	Khoo	24-JUL-05
3	24-JUL-05	Tobias	24-DEC-05
4	24-DEC-05	Baida	15-NOV-06
5	15-NOV-06	Himuro	10-AUG-07
6	10-AUG-07	Colmenares	(null)

Oracle ROW_NUMBER

- ROW_NUMBER is an analytic function. It assigns a unique number to each row to which it is applied (either each row in the partition or each row returned by the query), in the ordered sequence of rows specified in the ORDER_BY_CLAUSE, beginning with 1.
- For each department in the sample table oe.employees, the following example assigns numbers to each row in order of employee's hire date:

```
1  SELECT
2  department_id, last_name, employee_id,
3  ROW_NUMBER()
4  OVER (ORDER BY employee_id) AS emp_id
5     FROM employees;
```

	DEPARTMENT_ID	LAST_N...	EMPLOYEE_ID	EMP_ID
1	90	King	100	1
2	90	Kochhar	101	2
3	90	De Haan	102	3
4	60	Hunold	103	4
5	60	Ernst	104	5

Additional Resources

Data Science Academy

Oracle Index

- Oracle index is one of the effective tools for boost the query performance.
- To create a new index for a table, you use the CREATE INDEX statement as follows:

```
1 CREATE INDEX index_name
2 ON table_name(column1[,column2,...])
```

- When you create a new table with a primary key, Oracle automatically creates a new index for the primary key columns.
- Unlike other database systems, Oracle does not automatically create an index for the foreign key columns.

Oracle Index

In this syntax:

- First, specify the name of the index. The index name should be meaningful and includes table alias and column name(s) where possible, along with the suffix _I such as:

<table_name>_<column_name>_I

- Second, specify the name of the table followed by one or more indexed columns surrounded by parentheses.

Oracle CREATE INDEX examples

- The following statement creates a new table named members that stores members' data:

```
CREATE TABLE members(
  member_id INT GENERATED BY DEFAULT AS IDENTITY,
  first_name   VARCHAR2(100)
  NOT   NULL,      last_name
  VARCHAR2(100)   NOT   NULL,
  gender CHAR(1) NOT NULL,
  dob DATE NOT NULL,
  email VARCHAR2(255) NOT NULL,
  PRIMARY KEY(member_id)
);
```

Oracle CREATE INDEX examples

- The members table has a primary key column, therefore, member_id Oracle created a new index for this column. To view all indexes of a table, you query from the all_indexes view:

```
SELECT
  index_name,
  index_type,
  visibility,
  status
FROM
  all_indexes
WHERE
  table_name = 'MEMBERS';
```

Here is the output:

INDEX_NAME	INDEX_TYPE	VISIBILITY	STATUS
SYS_C007865	NORMAL	VISIBLE	VALID

Oracle CREATE INDEX examples

- Suppose, you often want to look up members by the last name and you find that the query is quite slow. To speed up the lookup, you create an index for the last_name column:

```
1 CREATE INDEX members_last_name_i
2 ON members(last_name);
```

Oracle CREATE INDEX examples

- Now, showing the indexes again, you will find that the members table has two indexes:

```
1  SELECT
2    index_name,
3    index_type,
4    visibility,
5    status
6  FROM
7    all_indexes
8  WHERE
9    table_name = 'MEMBERS';
```

The output is:

INDEX_NAME	INDEX_TYPE	VISIBILITY	STATUS
SYS_C007865	NORMAL	VISIBLE	VALID
MEMBERS_LAST_NAME_I	NORMAL	VISIBLE	VALID

Creating an index on multiple columns example

- The following example creates an index on both last name and first name columns:

```
1 CREATE INDEX members_name_i
2 ON members(last_name, first_name)
```

Removing an INDEX

- To remove an index, you use the DROP INDEX statement:

```
1 DROP INDEX index_name;
```

- For example, to drop the members_last_name_i index, you use the following statement:

```
1 DROP INDEX members_last_name_i;
```

Oracle Sequence

- A sequence is a list of integers in which their orders are important. For example, the (1,2,3,4,5) and (5,4,3,2,1) are totally different sequences even though they have the same members.
- The CREATE STATEMENT statement allows you to create a new sequence object in your own schema.

```
CREATE SEQUENCE id_seq
    INCREMENT BY 10
    START WITH 10
    MINVALUE 10
    MAXVALUE 100
    CYCLE
    CACHE 2;
```

Using a Sequence

- To access the next available value for a sequence, you use the NEXTVAL pseudo-column:

```
1 SELECT id_seq.NEXTVAL
2 FROM dual;
```

- Once, you acquire the sequence number through the NEXTVAL pseudo-column, you can access it repeatedly using the CURRVAL pseudo-column:

```
1 SELECT id_seq.CURRVAL
2 FROM dual;
```

Modifying & Removing a Sequence

- To modify the attributes and behavior of an existing sequence object, you use the ALTER SEQUENCE statement.

```
1 ALTER SEQUENCE item_seq MAXVALUE 100;
```

- To remove an existing sequence from the database, you use the DROP SEQUENCE statement.

```
1 DROP SEQUENCE item_seq;
```

II

Additional Resources

Data Science Academy

What is mySQL?

- Relational database management system
- Stores all data in different tables and databases in an orderly fashion instead of stacking in a single warehouse
- Processes the most common and standard language called SQL, which is used to access databases.
- It is double licensed software . It has both a free software with a General Public License (GPL and a separate license for those who want to use it in areas restricted by the GPL)

What about MariaDB?

- First Sun Microsystems, then **Oracle** moved away from **MySQL**'s community development model. Monty Widenius, the creator of **MySQL**, started to develop **MySQL**'s code, and this time, developed it under the same code base giving the name **MariaDB**, the name of her second daughter.
- **MariaDB** is a fork (drop-in replacement).
- In the first MySQL model, the company was backed with rapid and efficient development with company support. Same commands, same interfaces, same libraries and **API**'s with **MySQL**.
- It can use MySQL databases without converting.
- MariaDB version numbers are fully compatible with MySQL(MariaDB 5.2 >=MySQL 5.2)
- Additional new features and improvements are being made.

Key differences

- Most of the deployments still officially support MySQL , MariaDB needs to be installed separately.
- The awareness of the brand of MySQL is not present in MariaDB.
- MariaDB is not double licensed, you can only use it as GPL (free software).
- While in MariaDB 5.3 series, in MySQL 5.5 series.

Table type differences- I

- InnoDB (MySQL): Standard transaction supported table type with reliable ACID compatible design, advanced MVCC structure.
- XtraDB (MariaDB): Developed by Percona, backwards compatible with InnoDB, it can be used instead. It has more features than InnoDB, it can be fine-tuned and scaled.
- Federated (MySQL): it can use independent database servers to create a single logical database. Suitable for distributed structures. It is no longer developed by Oracle.
- FederatedX (MariaDB): One-to-one backward compatible with Federated. Its development continues, new features are added, existing errors are corrected.

Table Types Differences - II

- IMDB2i (MariaDB): Allows MariaDB to store its data in DB2 tables on IBM i with transaction support. Oracle removed this support from MySQL after purchasing MySQL. It can still be used with MariaDB.
- Aria (MariaDB): The type of table that is intended to become the default choice for both atomic and transaction tables in the future. It has been in development since 2007.
- PBXT(MariaDB): A table type developed by Primebase, ACID compatible, MVCC and similar advanced features.
- OQGraph (MariaDB): Open Query Graph tables allow hierarchical (tree structure) and complex graphs (nodes with multiple links in many directions).
- SphinxSE (MariaDB): A table type that allows using the Sphinx search engine to store and search data.

Additional Features of MariaDB

- Faster and safer replication with group commit.
- No unnecessary character encoding conversions, there is an increase in speed.
- Pluggable verification structure.
- Virtual and dynamic column for each row
- Wider user statistics
- Partial key cache

XAMPP

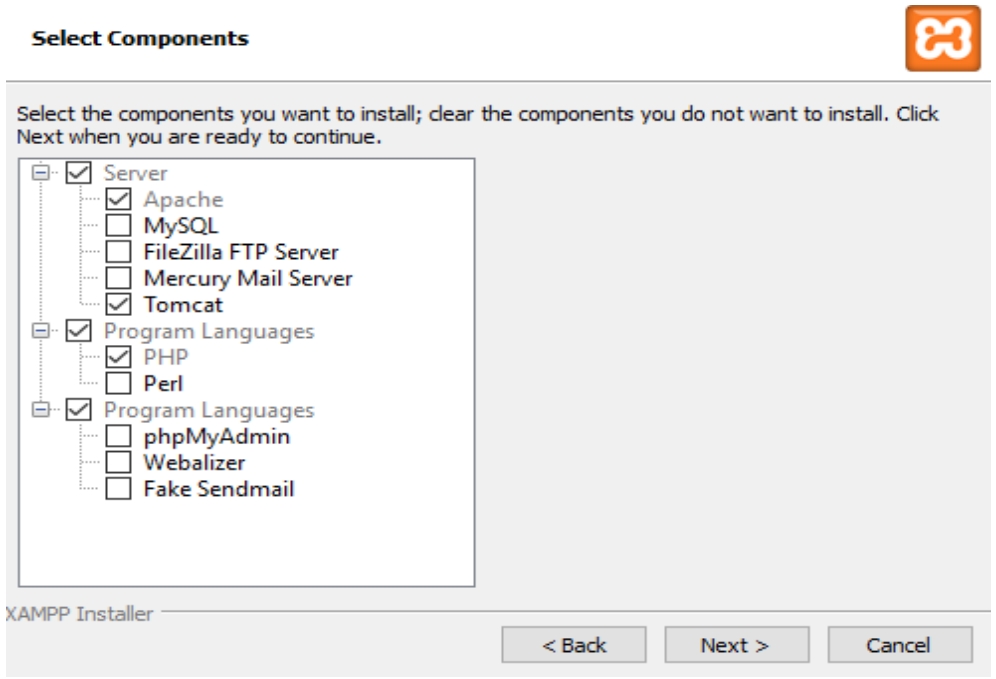
- XAMPP is software packet which contains four key components.
- The software packet contains the web server Apache , the relational database management system MySQL (or MariaDB), and the scripting languages Perl and PHP.
- The initial X stands for the cross-platform compatibility: Linux, Windows, and Mac OS X.

XAMPP INSTALLATION

- Step 1: Download and install XAMPP. XAMPP is a release made available by the non-profit project Apache Friends. Versions with PHP 5.5, 5.6, or 7 are available for download on the https://www.apachefriends.org/download.html

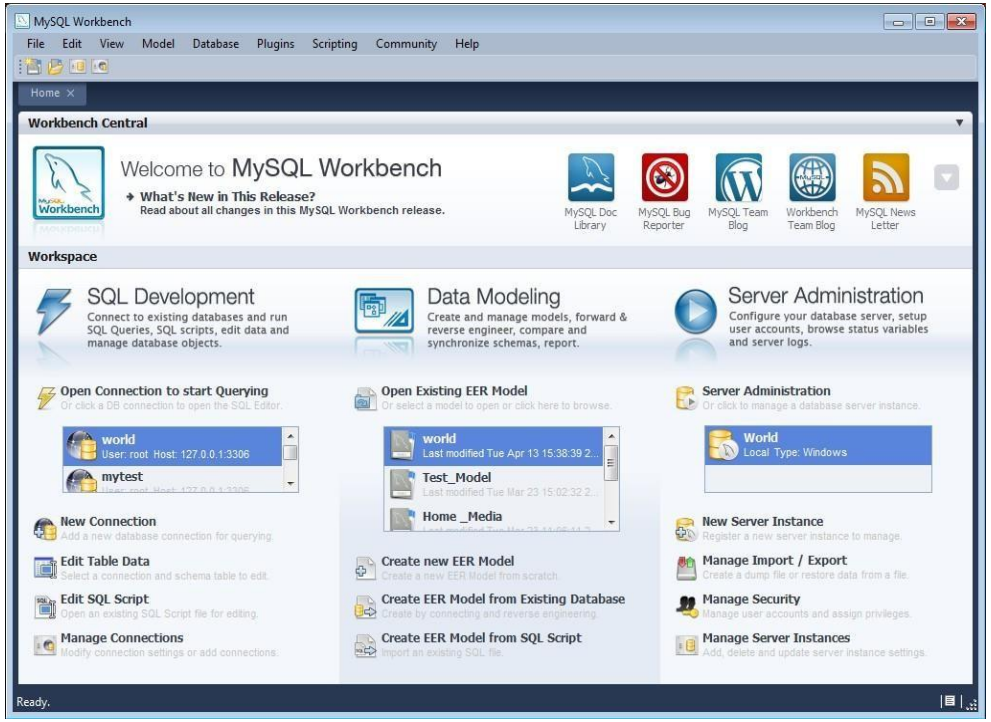
- Step 2: Once the software bundle has been downloaded, you can start the installation by double clicking on the .exe file.

- Step 3: Select Components. Next, you need to check the components which you want to install and can uncheck or leave as it is which you don't want to install.



MySQL Workbench

- MySQL Workbench is a unified visual tool for database architects, developers, and DBAs.
- MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more.
- MySQL Workbench is available on Windows, Linux and Mac OS X.



MySQL Workbench Installation

- MySQL Workbench for Windows can be installed using the MySQL installer that installs and updates all MySQL products on Windows or the standalone Windows MSI Installer package.
- Installation using MySQL installer
- The general MySQL Installer download is available at

https://dev.mysql.com/downloads/windows/installer/

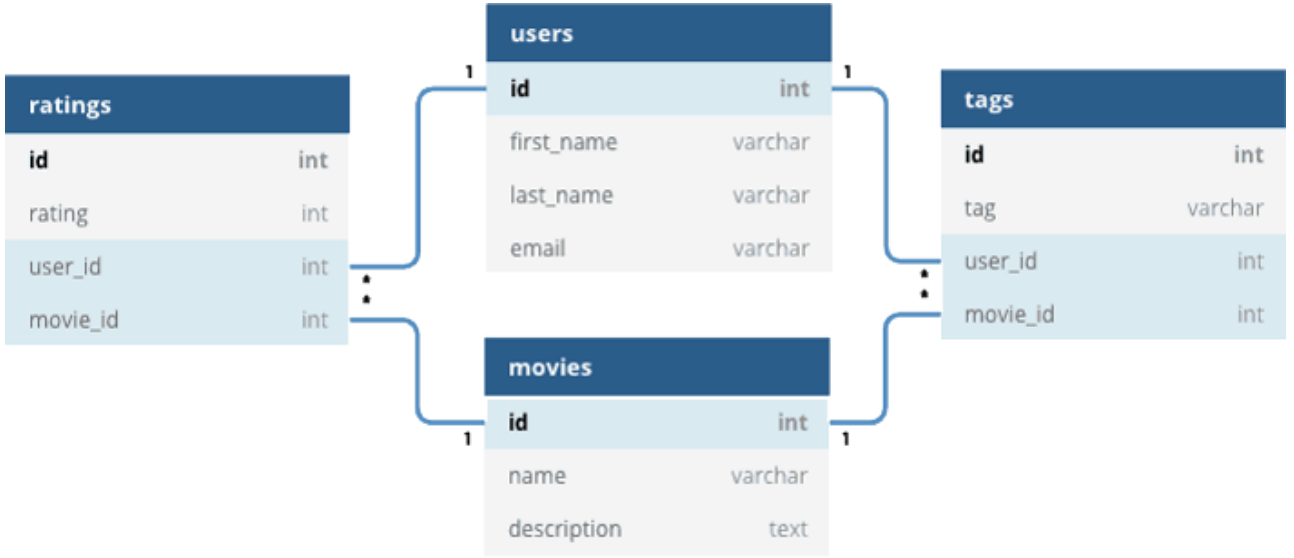
- The MySQL Installer application can install, upgrade, and manage most MySQL products, including MySQL Workbench.
- Managing all of your MySQL products, including Workbench, with MySQL Installer is the recommended approach. It handles all requirements and prerequisites, configurations, and upgrades.
- For additional information see :

https://dev.mysql.com/doc/workbench/en/wb-installing-windows.html

Visual Database Design

In MySQL Workbench

1. Click Home icon 
2. Choose Relational database icon 
3. Select Database to see the visual design of database



Introduction to MySQL Connector/Python

- MySQL Connector/Python enables Python programs to access MySQL databases, using an API that is compliant with the Python Database API Specification v2.0 (PEP 249).

- It is written in pure Python and does not have any dependencies except for the Python Standard Library.

- Installing Connector/Python with pip.
  - Use **pip** to install Connector/ Python on most any operating system:

```
>>> pip install mysql-connector-python
```

Connecting To MySQL Using Python

- Before you can access MySQL databases using Python, you must install one (or more) of the following packages in a virtual environment”
  - MySQL-python: This package contains the MySQLdb module, which is written in C. It is one of the most commonly used Python packages for MySQL
  - mysql-connector-python: This package contains the mysql.connector module, which is written entirely in Python.
  - PyMySQL: This package contains the pymysql modelu, which is written entirely in Python. It is designed to be a drop-in replacement for the MySQL-python package.

Setting up the Python virtual environment and installing a MySQL package

- Log in to your account using SSH.
- To create a virtual environment, type the following commands:

```
>>> cd ~ virtualenv -p /usr/bin/python3 sqlenv
```

- To activate the virtual environment, type of following command:

```
>>> source sqlenv/bin/activate
```

- Type the command for the package you want to install:
- To install the MySQL-python package,type the following command:

```
>>> pip install MySQL-python
```

- To install the mysql-connector-python package,type the following command:

```
>>> pip install mysql-connector-python
```

- To install the pymysql package, type the following command:

```
>>> pip install pymysql
```