AWS/Python Training

MySQL

Authorized & published by Summitworks Technologies Inc

SummitWorks™
GLOBAL SOLUTION ARCHITECTS

MySQL – Day 3 Training Plan

Agenda: Day - 3: Introduction to MySQL

- Introduction and MySQL concepts
  - Why to use Database?
  - What is ER (Entity-relationship) – Modeling?
  - Data types
  - Normalization
  - Keys in SQL
  - Database Constraints
- MySQL Installation
- Sub Languages in MySQL
  - Data Definition Language (DDL)
    - Create, Alter, Drop
  - Data Manipulation Language (DML)
    - Insert,Update,Delete
  - Data Query Language (DQL)
    - Select,Show,Help
  - Data Control Language (DCL)
    - Grant and Revoke
- Basic SQL
  - Conditional Statements
  - Comparison Operators
- Joins
  - Inner
  - Outer Join
  - Left Join
  - Right Join
  - Cross Join

Introduction to MySQL concepts

Why to use databases?

- A database is a collection of information that is organized so that it can easily be accessed, managed, and updated.
- Benefits of using database:
  - To eliminate or reduce the data Redundancy
  - To remove or reduce the data Inconsistency
  - For data Standardization
  - Secure all the data in one shell. Performed Role based security
  - To make sure that the Integrity of the data is maintained

Introduction to MySQL concepts (cont.)

What is ER (Entity-relationship) – Modeling?

- ➢ **ER Diagram** is a visual representation of data that describes how data is related to each other.
- ➢ In ER Model, we disintegrate data into **entities**, **attributes** and setup **relationships** between entities, all this can be represented visually using the ER diagram.
- ➢ ER Diagrams are most often used to design or debug relational databases in the fields of software engineering.
- ➢ they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes.

Introduction to MySQL concepts (cont.)

Components of ER Diagram

- ➢ Entity, Attributes, Relationships etc form the components of ER Diagram and there are defined symbols and shapes to represent each one of them.
- ➢ Let's see how we can represent these in our ER Diagram.

  Entity

  Simple rectangular box represents an Entity.

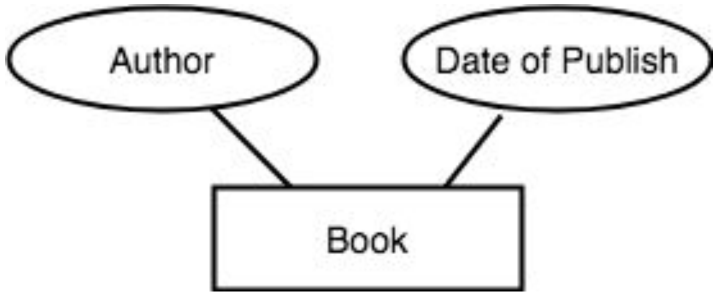
Student
Subject

Introduction to MySQL concepts (cont.)

Components of ER Diagram

Attributes for any Entity

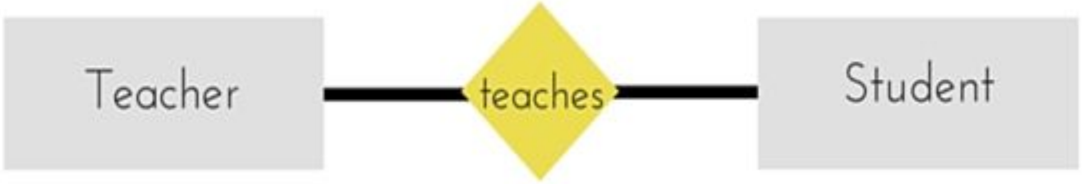
Ellipse is used to represent attributes of any entity. It is connected to the entity



Introduction to MySQL concepts (cont.)

ER Diagram: Relationship

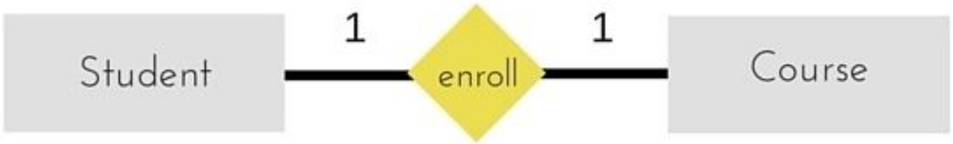
A Relationship describes relation between entities. Relationship is represented using diamonds or rhombus.



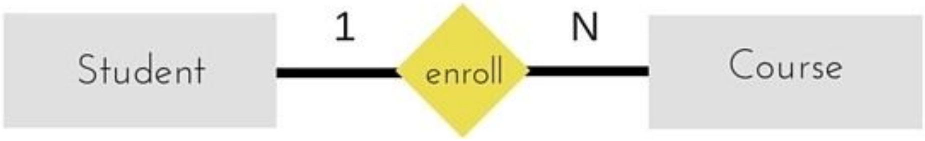
Introduction to MySQL concepts (cont.)

ER Diagram: Binary Relationship Binary Relationship means relation between two Entities. This is further divided into three types.

One to One Relationship: This type of relationship is rarely seen in real world.



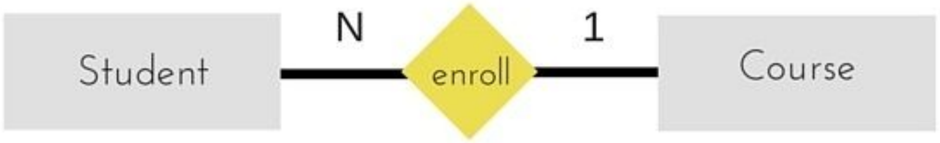
One to Many Relationship: The below example showcases this relationship, which means that 1 student can opt for many courses, but a course can only have 1 student. Sounds weird! This is how it is.



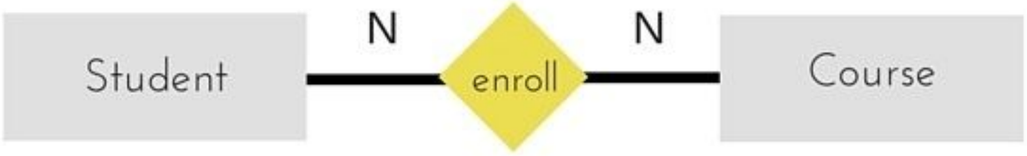
Introduction to MySQL concepts (cont.)

ER Diagram: Binary Relationship

Many to One Relationship: It reflects business rule that many entities can be associated with just one entity. For example, Student enrolls for only one Course but a Course can have many Students.



Many to One Relationship: The below diagram represents that one student can enroll for more than one courses. And a course can have more than 1 student enrolled in it



Data types

- Properly defining the fields in a table is important to the overall optimization of your database. You use only the type and size of field you really need to use. MySQL uses many different data types broken into three categories –

    - **Numeric:** INT , FLOAT
    - **Date and Time:** DATE , DATETIME
    - **String Types:** CHAR, VARCHAR() , VARCHAR2, BLOB or TEXT

Data types (cont.)

- **Numeric**
  - **INT** – A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
  - **FLOAT(M,D)** – A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.
  - Ex: if you insert 999.00009 into a FLOAT(7,4) column, the approximate result is 999.0001.
  - **DOUBLE(M,D)** – A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.
  - **DECIMAL(M,D)** – An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

Data types (cont.)

- **Date and Time**
  - **DATE –** A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.
  - **DATETIME –** A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.

Data types (cont.)

- **String types**
  - **CHAR(M)** – A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.
  - **VARCHAR(M)** – A variable-length string between 1 and 255 characters in length. For example, VARCHAR(25). You must define a length when creating a VARCHAR field.

Keys

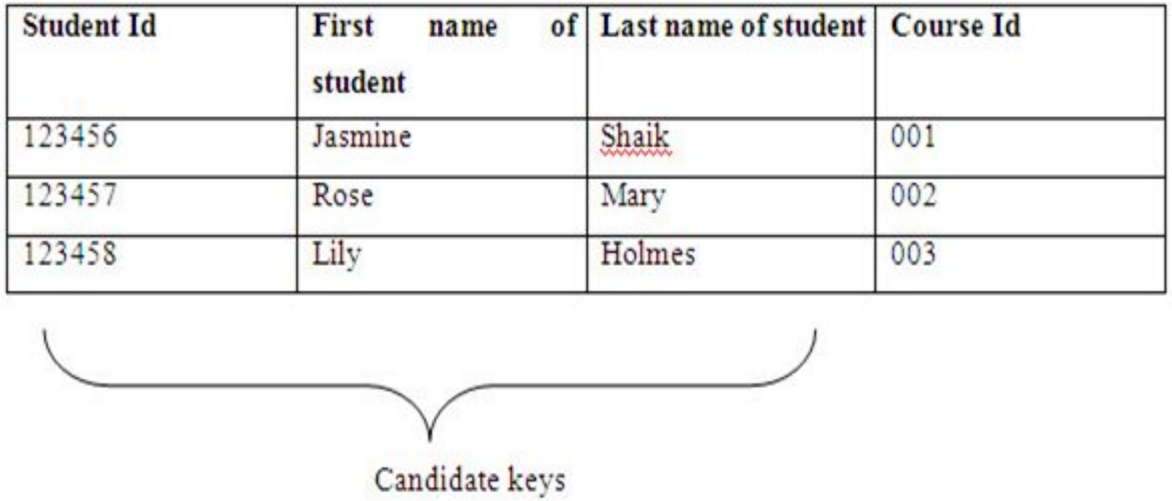
- **Keys** are important part of the arrangement of a table. Keys make sure to uniquely identify a table's each part or record of a field or combination of fields.
  - Type of keys:
    - Natural Key
    - Surrogate/Artificial key
    - Candidate keys
    - Primary key
    - Secondary or Alternative key
    - Composite Key

Keys (cont.)

- **Natural Key:** It is a key that is naturally declared as the Primary key. Natural keys are sometimes called as business or domain keys because these key are based on the real world observation. So it is a key whose attributes or values exists in the real world. These attributes have logical relationship with the table.
  - For Example: Social Security Number (SSN) is a natural key that can be declared as the primary key.
- **Surrogate/Artificial key :** Surrogate key is artificially generated key and its main purpose it to be the primary key of table. Artificial keys do not have meaning to the table. There are few properties of surrogate or artificial keys.
  - They are unique because these just created when you don't have any natural primary key.
  - They are integer values. One cannot find the meaning of surrogate keys in the table.
  - End users cannot surrogate key.Surrogate keys are allowed when no property has the parameter of primary key.
  - The primary key is huge and complex.
  - Example: Table which has the details of the student has primary key but it is large and complex. The addition of row id column to it is the DBA's decision, where the primary key is row id.

Keys (cont.)

- **Candidate keys** are the set of fields; primary key can be selected from these fields. A set of properties or attributes acts as a primary key for a table. Every table must have at least one candidate key or several candidate keys.
  - For example: The fields of a candidate key uniquely identify a student. It has the properties like – Being unique and Parameter of irreducibility**.**

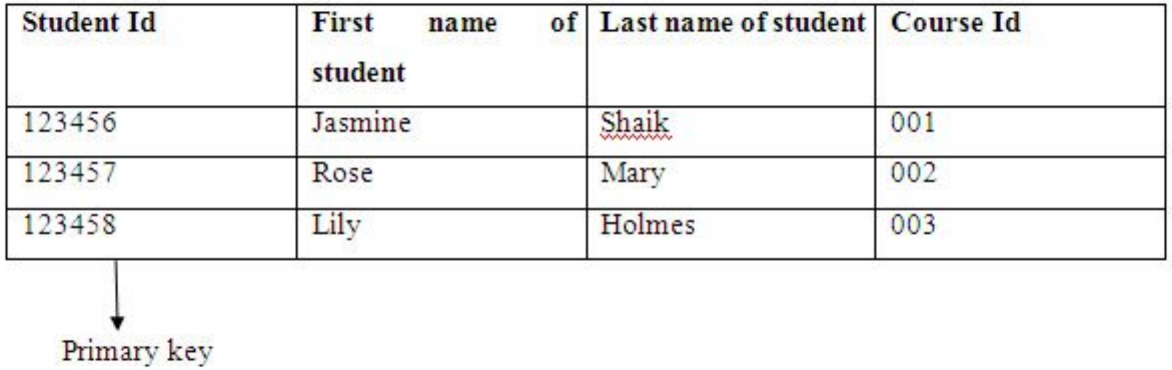


Student Id	First name of student	Last name of student	Course Id
123456	Jasmine	Shaik	001
123457	Rose	Mary	002
123458	Lily	Holmes	003

Candidate keys

Keys (cont.)

- **Primary key:** The candidate key which is very suitable to be the main key of table is a primary key.
- The primary keys are compulsory in every table.
- The properties of a primary key are:
  - Model stability
  - Occurrence of minimum fields
  - Defining value for every record i.e. being definitive
  - Feature of accessibility

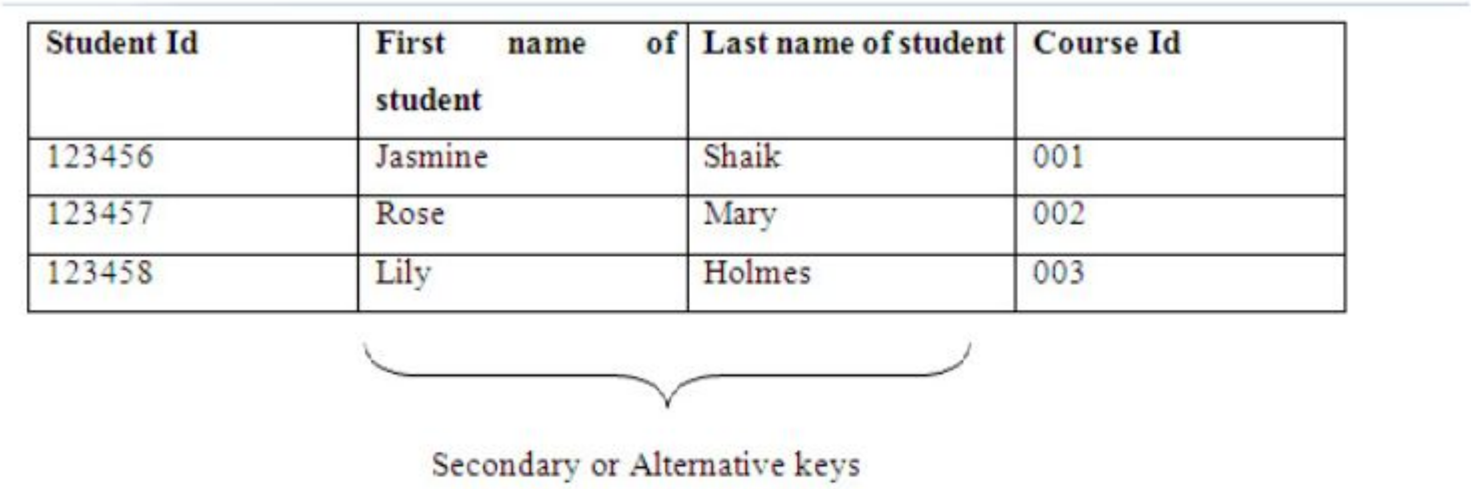


Student Id	First name of student	Last name of student	Course Id
123456	Jasmine	Shaik	001
123457	Rose	Mary	002
123458	Lily	Holmes	003

Primary key

Keys (cont.)

- **Secondary or Alternative key:** The rejected candidate keys as primary keys are called as secondary or alternative keys.

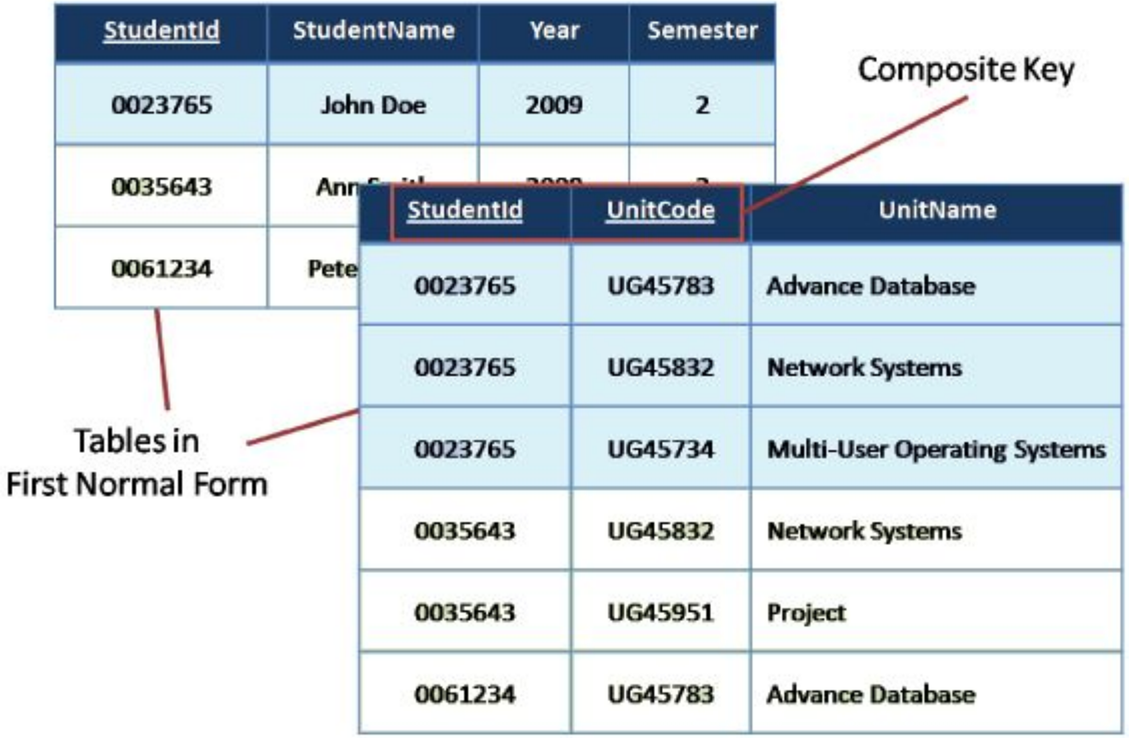


Student Id	First name of student	Last name of student	Course Id
123456	Jasmine	Shaik	001
123457	Rose	Mary	002
123458	Lily	Holmes	003

Secondary or Alternative keys

Keys (cont.)

- **Composite Key** has two or more properties which specially identifies the occurrence of an entity.



StudentId	StudentName	Year	Semester
0023765	John Doe	2009	2
0035643	Ann Smith	2009	2
0061234	Pete		

StudentId	UnitCode	UnitName
0023765	UG45783	Advance Database
0023765	UG45832	Network Systems
0023765	UG45734	Multi-User Operating Systems
0035643	UG45832	Network Systems
0035643	UG45951	Project
0061234	UG45783	Advance Database

Composite Key
Tables in
First Normal Form

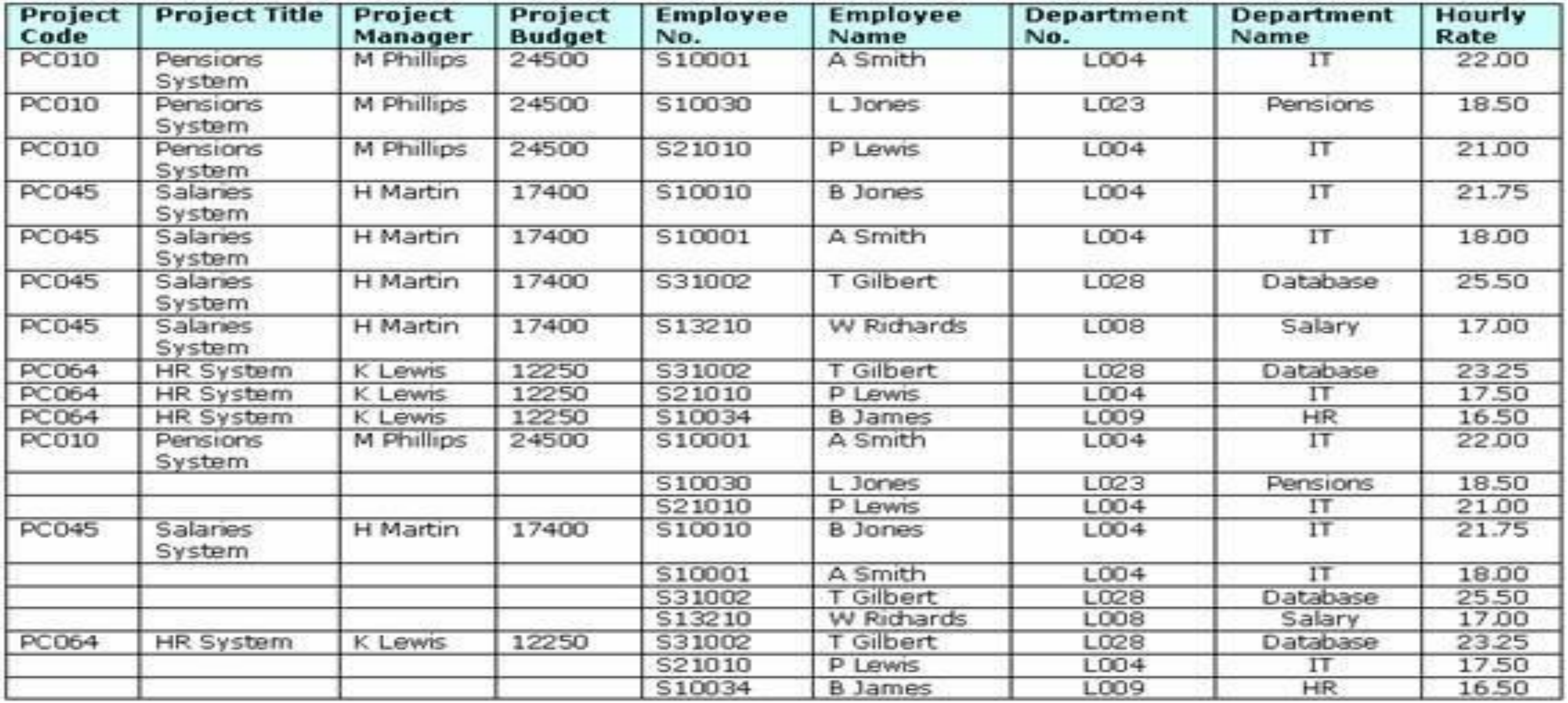
What is Normalization? 1NF, 2NF, 3NF and BCNF

- Database normalization (or normalization) is the process of organizing the columns (attributes) and tables (relations) of a relational database to minimize data redundancy.

<u>UNF</u>	1NF	2NF	3NF
• All the attributes of the database are simply listed, without any sense of order or grouping.	• It has an identifying key • Each table cell should contain a single value.	• It is in 1NF • All non-key attributes are functionally dependent on the whole key (not part of the key	• It is in 2NF • It contains no transitive dependencies

Normalization(Example)

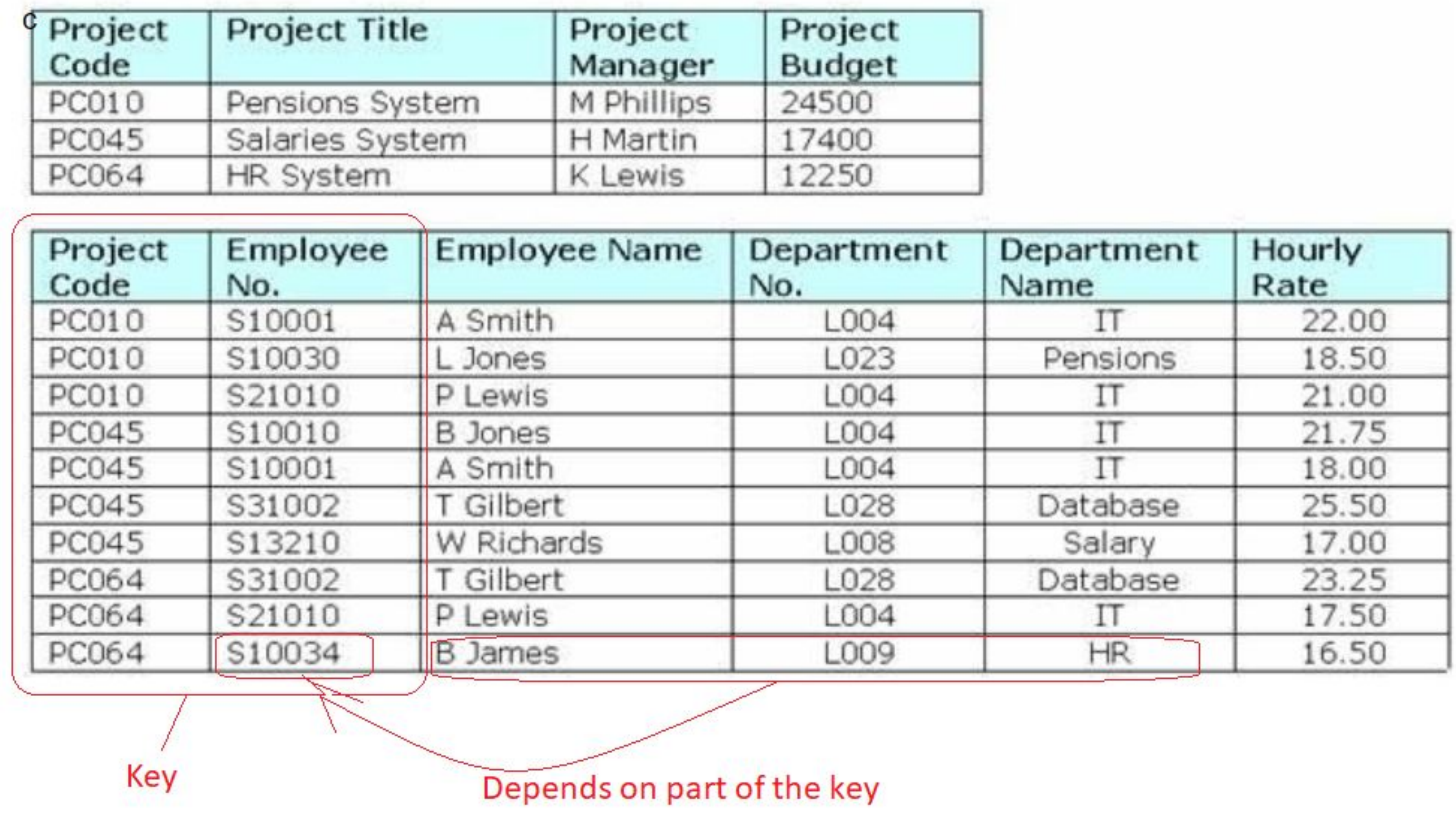
Unnormalised Form



Project Code	Project Title	Project Manager	Project Budget	Employee No.	Employee Name	Department No.	Department Name	Hourly Rate
PC010	Pensions System	M Phillips	24500	S10001	A Smith	L004	IT	22.00
PC010	Pensions System	M Phillips	24500	S10030	L Jones	L023	Pensions	18.50
PC010	Pensions System	M Phillips	24500	S21010	P Lewis	L004	IT	21.00
PC045	Salaries System	H Martin	17400	S10010	B Jones	L004	IT	21.75
PC045	Salaries System	H Martin	17400	S10001	A Smith	L004	IT	18.00
PC045	Salaries System	H Martin	17400	S31002	T Gilbert	L028	Database	25.50
PC045	Salaries System	H Martin	17400	S13210	W Richards	L008	Salary	17.00
PC064	HR System	K Lewis	12250	S31002	T Gilbert	L028	Database	23.25
PC064	HR System	K Lewis	12250	S21010	P Lewis	L004	IT	17.50
PC064	HR System	K Lewis	12250	S10034	B James	L009	HR	16.50
PC010	Pensions System	M Phillips	24500	S10001	A Smith	L004	IT	22.00
				S10030	L Jones	L023	Pensions	18.50
				S21010	P Lewis	L004	IT	21.00
PC045	Salaries System	H Martin	17400	S10010	B Jones	L004	IT	21.75
				S10001	A Smith	L004	IT	18.00
				S31002	T Gilbert	L028	Database	25.50
				S13210	W Richards	L008	Salary	17.00
PC064	HR System	K Lewis	12250	S31002	T Gilbert	L028	Database	23.25
				S21010	P Lewis	L004	IT	17.50
				S10034	B James	L009	HR	16.50

Normalization(Example)

- First Normal Form(1NF)
  - The rule is: remove any repeating attributes to a new table. The process is as follows:



Project Code	Project Title	Project Manager	Project Budget
PC010	Pensions System	M Phillips	24500
PC045	Salaries System	H Martin	17400
PC064	HR System	K Lewis	12250

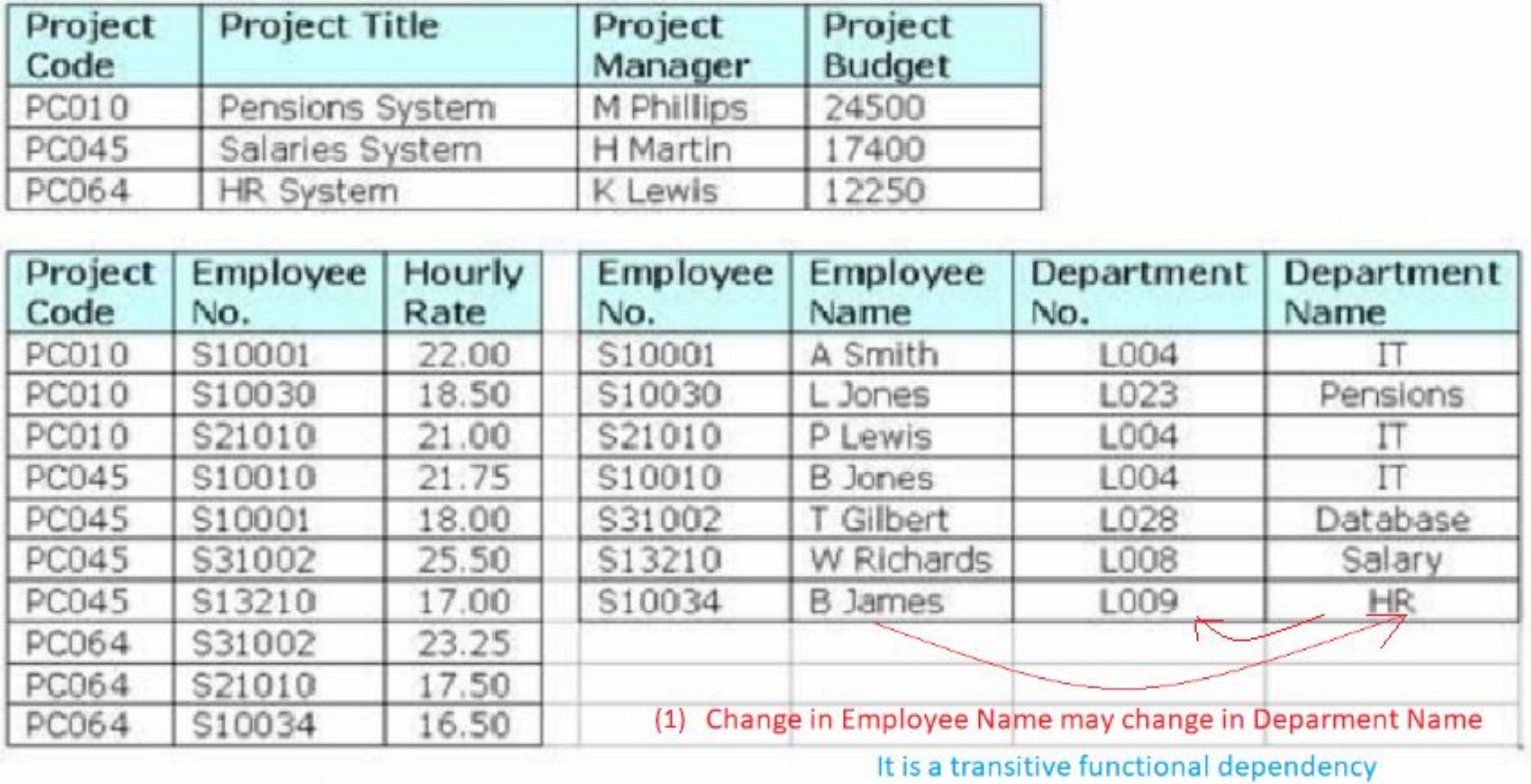
Project Code	Employee No.	Employee Name	Department No.	Department Name	Hourly Rate
PC010	S10001	A Smith	L004	IT	22.00
PC010	S10030	L Jones	L023	Pensions	18.50
PC010	S21010	P Lewis	L004	IT	21.00
PC045	S10010	B Jones	L004	IT	21.75
PC045	S10001	A Smith	L004	IT	18.00
PC045	S31002	T Gilbert	L028	Database	25.50
PC045	S13210	W Richards	L008	Salary	17.00
PC064	S31002	T Gilbert	L028	Database	23.25
PC064	S21010	P Lewis	L004	IT	17.50
PC064	S10034	B James	L009	HR	16.50

Key

Depends on part of the key

Normalization(Example)

- Second Normal Form(2NF)
  - The rule is: remove any non-key attributes that only depend on part of the table key to a new table.
  - A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change



Project Code	Project Title	Project Manager	Project Budget
PC010	Pensions System	M Phillips	24500
PC045	Salaries System	H Martin	17400
PC064	HR System	K Lewis	12250

Project Code	Employee No.	Hourly Rate
PC010	S10001	22.00
PC010	S10030	18.50
PC010	S21010	21.00
PC045	S10010	21.75
PC045	S10001	18.00
PC045	S31002	25.50
PC045	S13210	17.00
PC064	S31002	23.25
PC064	S21010	17.50
PC064	S10034	16.50

Employee No.	Employee Name	Department No.	Department Name
S10001	A Smith	L004	IT
S10030	L Jones	L023	Pensions
S21010	P Lewis	L004	IT
S10010	B Jones	L004	IT
S31002	T Gilbert	L028	Database
S13210	W Richards	L008	Salary
S10034	B James	L009	HR

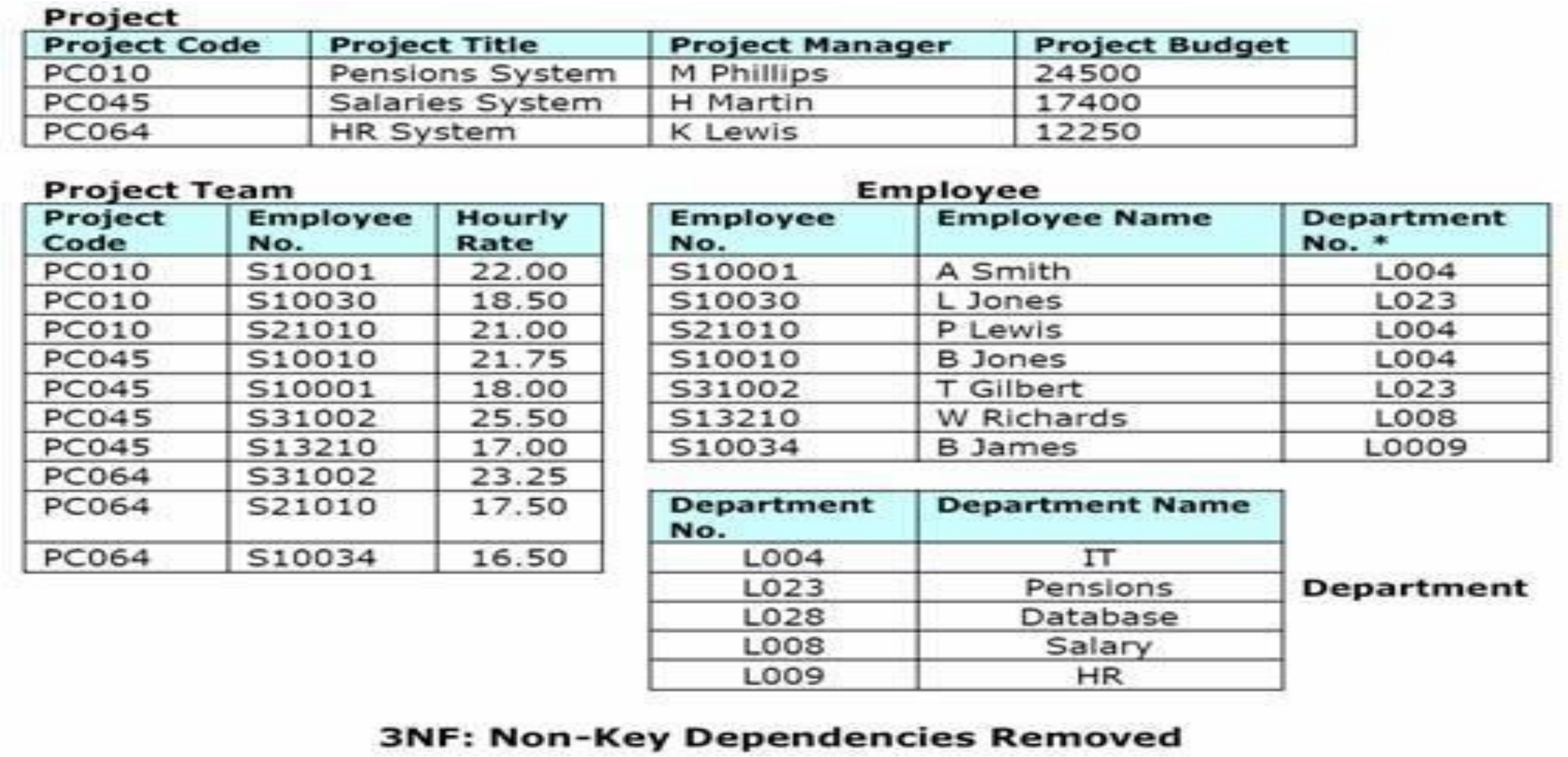
(1) Change in Employee Name may change in Deparment Name

It is a transitive functional dependency

(2) Change in Department name cause change in Deparment No.

Normalization(Example)

- Third Normal Form(3NF)
  - The rule is: remove to a new table any non-key attributes that are more dependent on other non-key attributes than the table key



Project

Project Code	Project Title	Project Manager	Project Budget
PC010	Pensions System	M Phillips	24500
PC045	Salaries System	H Martin	17400
PC064	HR System	K Lewis	12250

Project Team

Project Code	Employee No.	Hourly Rate
PC010	S10001	22.00
PC010	S10030	18.50
PC010	S21010	21.00
PC045	S10010	21.75
PC045	S10001	18.00
PC045	S31002	25.50
PC045	S13210	17.00
PC064	S31002	23.25
PC064	S21010	17.50
PC064	S10034	16.50

Employee

Employee No.	Employee Name	Department No. *
S10001	A Smith	L004
S10030	L Jones	L023
S21010	P Lewis	L004
S10010	B Jones	L004
S31002	T Gilbert	L023
S13210	W Richards	L008
S10034	B James	L0009

Department

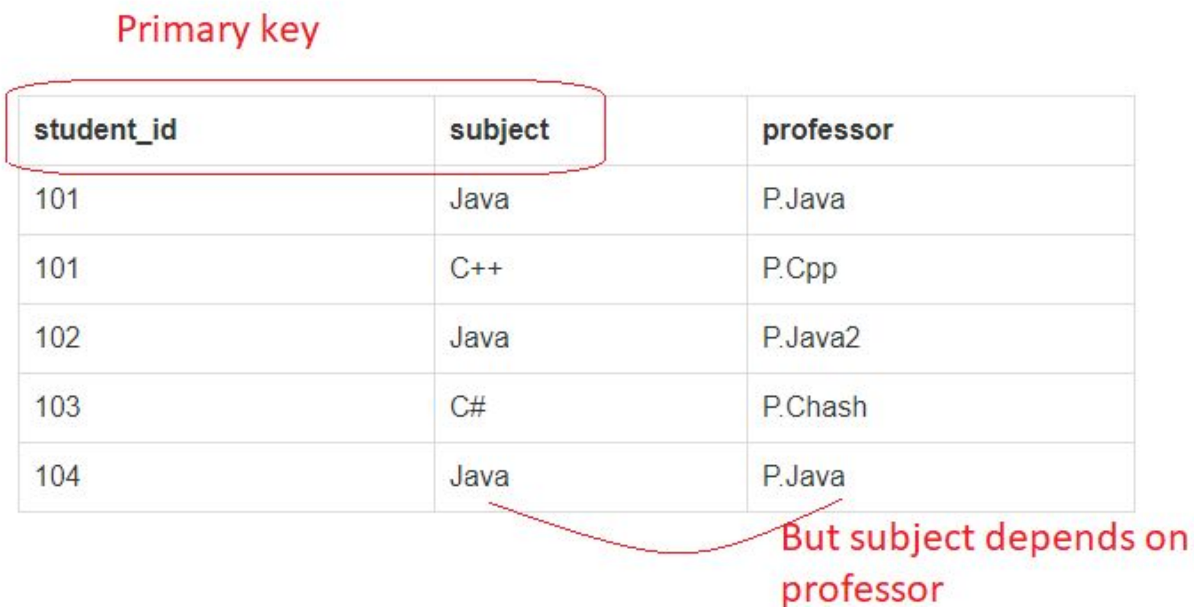
Department No.	Department Name
L004	IT
L023	Pensions
L028	Database
L008	Salary
L009	HR

3NF: Non-Key Dependencies Removed

BCNF (3.5)

- BCNF (Boyce-Codd Normal Form) does **not allow dependencies between attributes that belong to candidate keys**.
- BCNF is a refinement of the third normal form in which it drops the restriction of a non-key attribute from the 3rd normal form.

- For Example: following is 3NF but not BCNF, since
  - *Subject* column is a prime attribute and *professor* is a non-prime attribute,
  - *Subject* is dependent on *professor* *(*since one professor teaches only one subject, but one subject may have two different professors.*)*



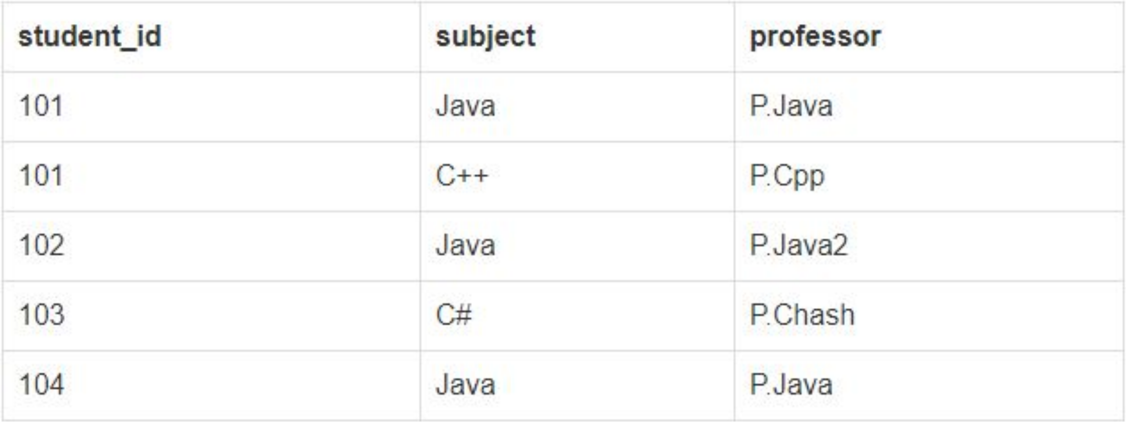
Primary key

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

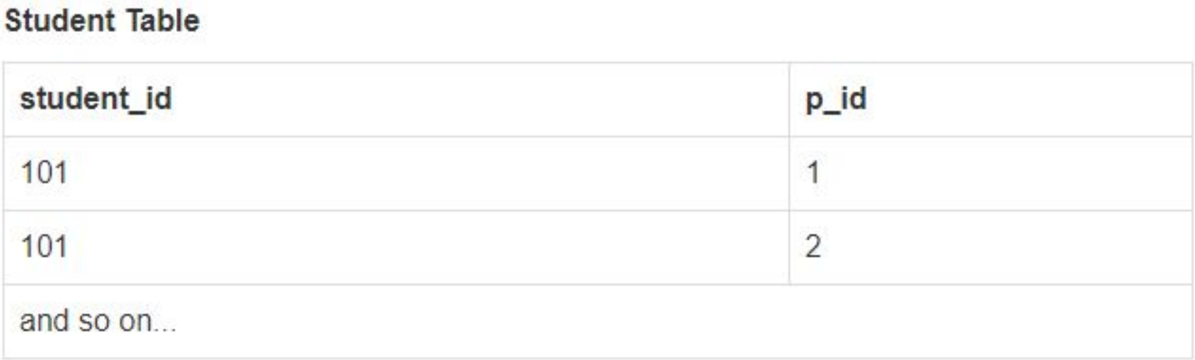
But subject depends on professor

BCNF

- For Example: From 3NF to BCNF.

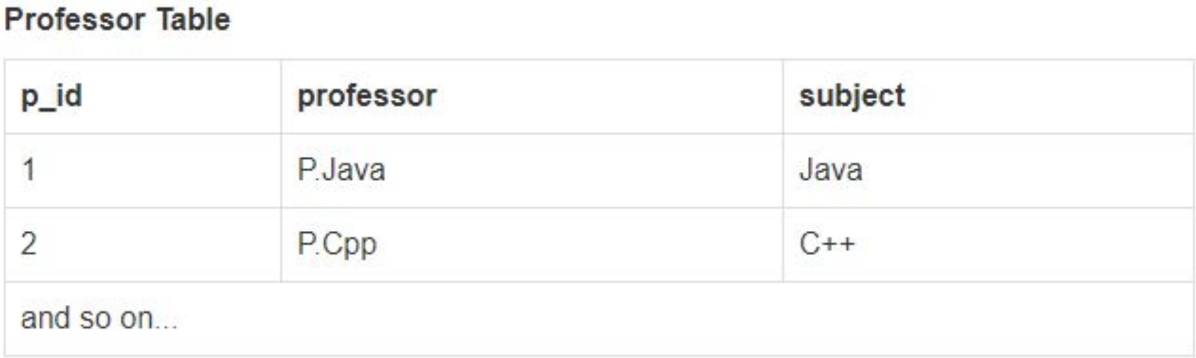


student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java



Student Table

student_id	p_id
101	1
101	2
and so on...	



Professor Table

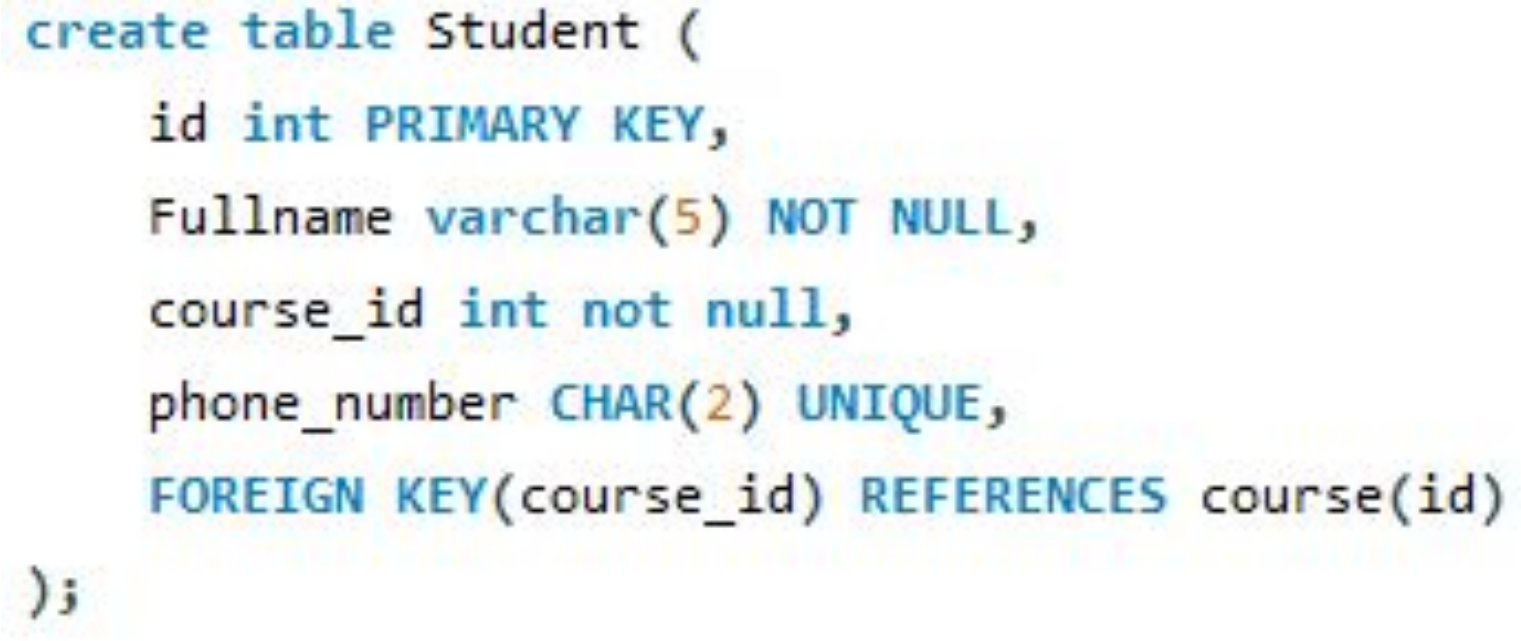
p_id	professor	subject
1	P.Java	Java
2	P.Cpp	C++
and so on...		

Database Constraints

- **SQL constraints** are used to specify rules for the data in a table.
- If there is any violation between the constraint and the data action, the action is aborted by the constraint.
- Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).
- **Constraints can be defined in two ways**
  - 1. The constraints can be specified immediately after the column definition. This is called column- level definition.
  - 2. The constraints can be specified after all the columns are defined. This is called table-level definition.
- **Types of constraints**
  - A NOT NULL constraint
  - A unique key constraint (A unique constraint)
  - A primary key constraint
  - A foreign key constraint (referential constraint or a referential integrity constraint)

Introduction to MySQL concepts (cont.)

Example of constraints:



```
create table Student (
    id int PRIMARY KEY,
    Fullname varchar(5) NOT NULL,
    course_id int not null,
    phone_number CHAR(2) UNIQUE,
    FOREIGN KEY(course_id) REFERENCES course(id)
);
```

Introduction to MySQL concepts (cont.)

MySQL Installation

- ➢ **For Windows**

  Refer to installation guide:
  MySQL all required features installation on Windows.pdf

- ➢ **For Macs**

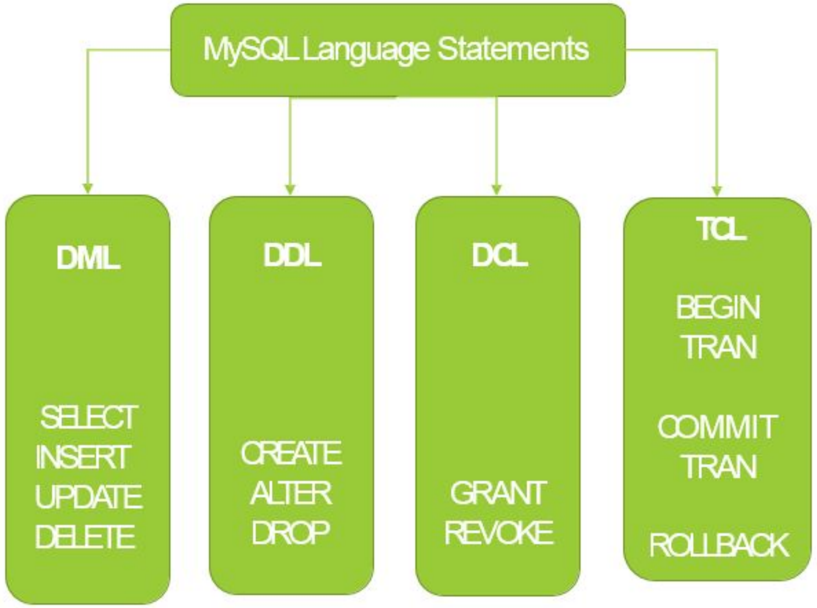
  Refer to installation guide:
  MySQL Server installation on MacOS.pdf
  MySQL Workbench (UI) installation on MacOS.pdf

  Or

  Follow the instruction here:
  https://dev.mysql.com/doc/refman/8.0/en/osx-installation-pkg.html

Sub Languages in MySQL

- **DDL (Data Definition Language):** statements are used to define the database structure or schema
- **DML (Data Manipulation Language):** statements are used for managing data within schema objects DML deals with data manipulation,
- **DCL (Data Control Language):** DCL statements control the level of access that users have on database objects.
  - **GRANT** – allows users to read/write on certain database objects
  - **REVOKE** – keeps users from read/write permission on database objects
- **TCL (Transaction Control Language):** statements allow you to control and manage transactions to maintain the integrity of data.
  - **BEGIN Transaction** – opens a transaction
  - **COMMIT Transaction** – commits a transaction
  - **ROLLBACK Transaction** – ROLLBACK a transaction in case of any error



MySQL Language Statements

DML

SELECT
INSERT
UPDATE
DELETE

DDL

CREATE
ALTER
DROP

DCL

GRANT
REVOKE

TCL

BEGIN
TRAN

COMMIT
TRAN

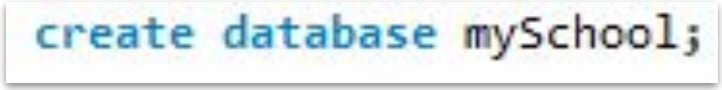
ROLLBACK

DDL Queries

Data Definition Language (DDL) is a vocabulary used to define data structures. Data Definition Language understanding with database schemas and describes how the data should consist in the database, therefore language statements like CREATE TABLE or ALTER TABLE

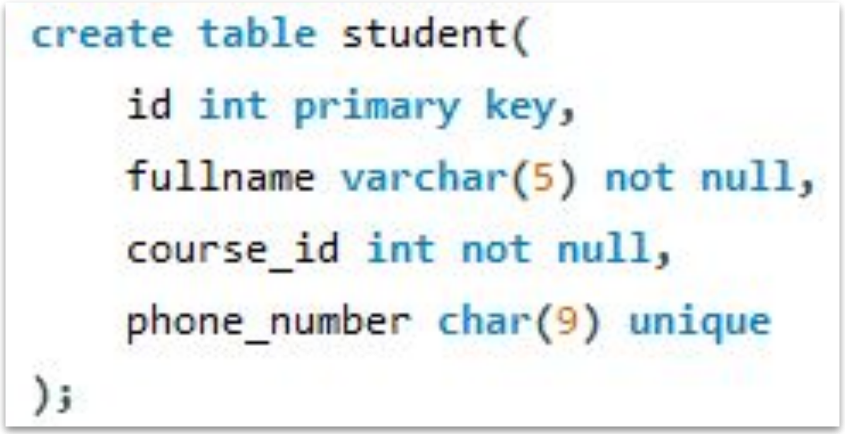
Create Statements: CREATE statements are used to define new entities.

CREATE DATABASE database_name



```
create database mySchool;
```

```
CREATE TABLE table_name
(
        column_name1 data_type(size),
        column_name2 data_type(size),
         column_name3 data_type(size),
        ....
);
```



```
create table student(
    id int primary key,
    fullname varchar(5) not null,
    course_id int not null,
    phone_number char(9) unique
);
```

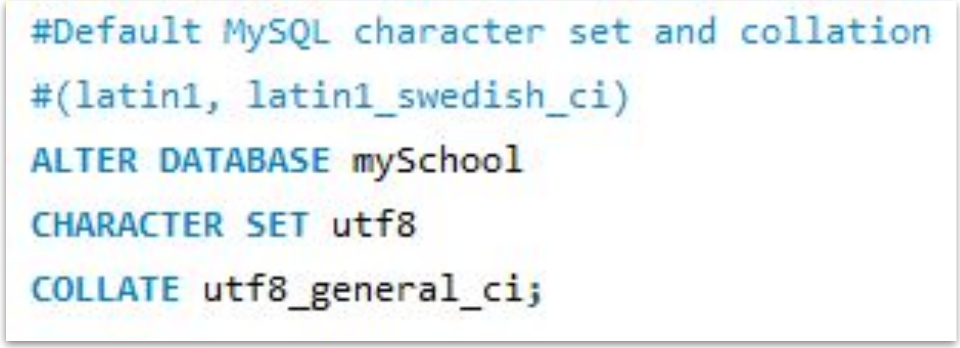
DDL Queries

ALTER Statements

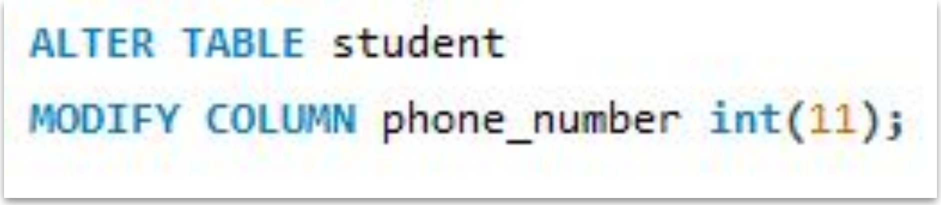
- ALTER statements are used to modify the definition of existing entities

```
ALTER {DATABASE | SCHEMA} [db_name]
   [DEFAULT] CHARACTER SET [=] charset_name
 | [DEFAULT] COLLATE [=] collation_name
```

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```



```
#Default MySQL character set and collation
#(latin1, latin1_swedish_ci)
ALTER DATABASE mySchool
CHARACTER SET utf8
COLLATE utf8_general_ci;
```

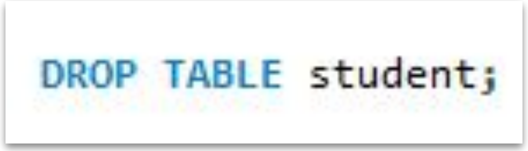


```
ALTER TABLE student
MODIFY COLUMN phone_number int(11);
```

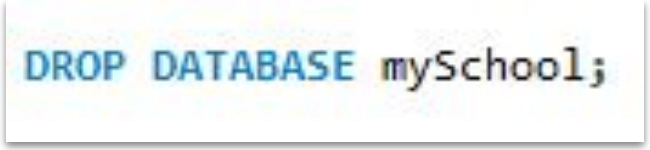
DDL Queries

DROP Statements

- Use DROP statements to remove existing entities.



```
DROP TABLE student;
```



```
DROP DATABASE mySchool;
```

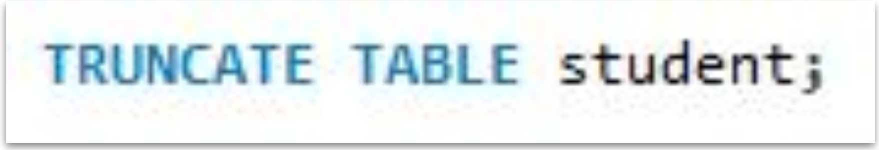
DDL Queries

Truncate table

Removes all rows from a table or specified partitions of a table, without logging the individual row deletions. The TRUNCATE TABLE command is used to delete complete data from an existing table.

You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.

TRUNCATE [TABLE] tbl_name



```
TRUNCATE TABLE student;
```

Data Manipulation Language (DML) Statements

- **DML** statements are used to work with the data IN tables.
- **DML** statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records.
- When you are connected to most multi-user databases (whether in a client program or by a connection from a Web page script), you are in effect working with a private copy of your tables that can't be seen by anyone else until you are finished (or tell the system that you are finished).

INSERT – insert new records
UPDATE – update/Modify existing records
DELETE – delete existing records

Data Manipulation Language (DML) Statements

Adds one or more rows to a table or a view in database

```
INSERT INTO table_name VALUES (value1, value2, value3…)
```

```
INSERT INTO table_name  (column2, column5,column6)  VALUES (value2, value5, value6)
```

For example:

```
INSERT INTO TraineeList VALUES (1, ‘John’ , ‘Smith’)

INSERT INTO TraineeList  (FirstName,LastName)  VALUES (‘Sara’ , ‘Wilson’)
```

Data Manipulation Language (DML) Statements

Update:

Changes existing data in a table or view

```
UPDATE table_name SET
column_name1 = new_value1,
column_name2 = new value2
WHERE some_column = some_value
(condition)
```

Sample:

```
UPDATE TraineeList  SET FirstName = ‘Philip’  WHERE Trainee_ID = 1
```

Data Manipulation Language (DML) Statements

DELETE : Removes one or more rows from a table or view

- ➢ DELETE * FROM table_name
- ➢ DELETE FROM table_name WHERE some_column = some_value (condition)

Sample:

- ➢ DELETE FROM TraineeList WHERE FirstName = ‘sara’

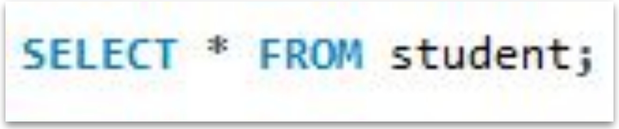
Differences between Truncate and Delete Statements

TRUNCATE	DELETE
TRUNCATE is a **DDL command**	DELETE is a **DML command**
TRUNCATE is executed using a table lock and **whole table is locked** for remove all records.	DELETE is executed using a row lock, **each row in the table is locked** for deletion.
We **can't Rollback** after performing Truncate.	We **can Rollback** after performing DELETE.
TRUNCATE **removes all rows** from a table. (We cannot use Where clause with TRUNCATE.)	The DELETE command is used to **remove rows from a table based on WHERE condition. (** (We can use where clause with DELETE to filter & delete specific records.)
Minimal logging in transaction log, so it is performance wise faster.	It maintain the log, so it slower than TRUNCATE.
TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log.	The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row
Identify column is reset to its seed value if table contains any identity column.	Identity of column keep DELETE retain the identity
To use Truncate on a table you need at least ALTER permission on the table.	To use Delete you need DELETE permission on the table.
Truncate uses the less transaction space than Delete statement.	Delete uses the more transaction space than Truncate statement.
Truncate cannot be used with indexed views	Delete can be used with indexed views
TRUNCATE TABLE can't activate a trigger because the operation does not log individual row deletions. When we run truncate command to remove all rows of table then it actually doesn't removes any row, rather it deallocates the data pages. In case of Truncate triggers will not be fired because no modification takes place, we have just deallocated the data pages not deleted	**Delete activates a trigger** because the operation are logged individually. When we execute Delete command, DELETE trigger will be initiated if present. Delete is a DML command and it deletes the data on row-by-row basis from a table. Which means delete is modifying the data by deleting it from the table. Triggers are fired when a DML statement executed on a table, so trigger will be fired in case of Delete command execution

Basic query Statements

SELECT :
Retrieves rows from the database and enables the selection of one or many rows or columns from one or many tables in MySQL. The full syntax of the SELECT statement is complex

SELECT select_list (* | column_name)
FROM table_name
WHERE search_condition GROUP BY group_by_expression HAVING search_condition
ORDER BY order_expression ASC | DESC



```
SELECT * FROM student;
```

Query statements: Show, Help

show: used to get more details about databases and tables.

Description	Command
List all databases on the sql server.	show databases;
To see all the tables in the db.	show tables;
Returns the columns and column information pertaining to the designated table.	show columns from [table name];

Help: The HELP statement returns online information from the MySQL Reference manual.

Description	Command
use contents to retrieve a list of the top-level help categories	HELP 'contents'
For a list of topics in a given help category, such as Data Types, use the category name	HELP 'data types'
For help on a specific help topic, such as the ASCII() function or the CREATE TABLE statement, use the associated keyword or keywords:	HELP 'ascii' HELP 'create table'

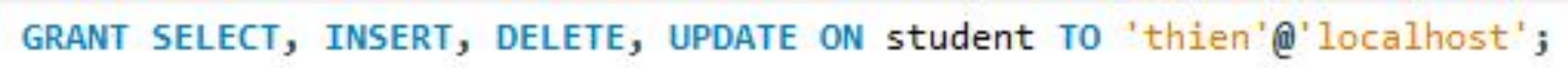
Data Control Language (DCL)

➢ **GRANT**: Used to provide any user access privileges or other privileges for the database.

GRANT : Data Control Language(DCL) is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges.

- Create a user, then you can try to create new connection with this user and password
  CREATE USER 'thien'@'localhost' IDENTIFIED BY '*password*';

- Examples of GRANT permission:
  GRANT ALL ON db1.* TO 'thien'@'localhost';



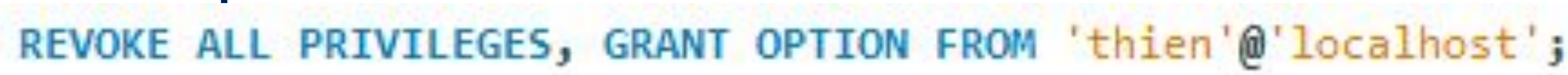
```
GRANT SELECT, INSERT, DELETE, UPDATE ON student TO 'thien'@'localhost';
```

Data Control Language (DCL)

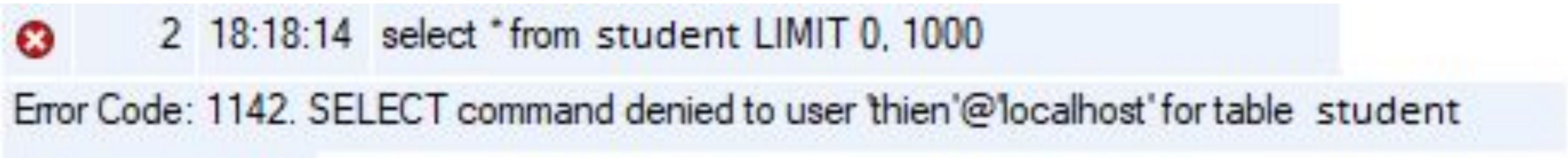
➢ **REVOKE**: Used to take back permissions from any user.

REVOKE ALL PRIVILEGES, GRANT OPTION FROM user_or_role [, user_or_role] ...
REVOKE 'role1', 'role2' FROM 'user1'@'localhost', 'user2'@'localhost';

For example:



```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'thien'@'localhost';
```



```
2   18:18:14   select * from student LIMIT 0, 1000
Error Code: 1142. SELECT command denied to user 'thien'@'localhost' for table  student
```

Basic SQL – Where Clause

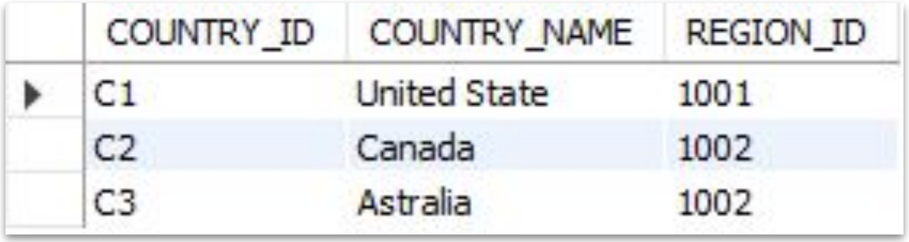
- **Conditional Statements: WHERE Clause**

Select statements:

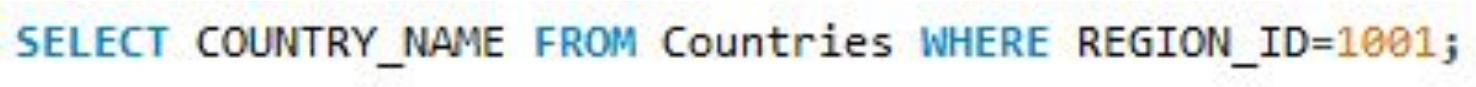
WHERE Clause: Specify an actual value as condition to filter the SELECT statement

```
SELECT columnName FROM tableName WHERE condition;
```

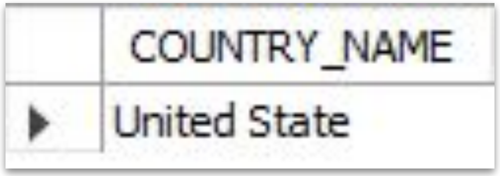
For example:



	COUNTRY_ID	COUNTRY_NAME	REGION_ID
▶	C1	United State	1001
	C2	Canada	1002
	C3	Astralia	1002



```
SELECT COUNTRY_NAME FROM Countries WHERE REGION_ID=1001;
```



	COUNTRY_NAME
▶	United State

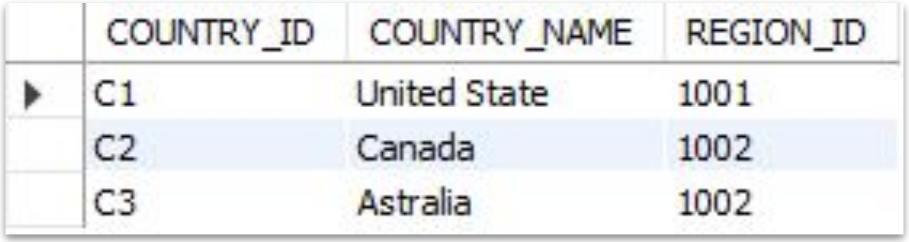
Basic SQL - Sorting

Sorting: When you select rows, the MySQL server is free to return them in any order, unless you instruct it otherwise by saying how to sort the result. But, you sort a result set by adding an ORDER BY clause that names the column or columns which you want to sort.
The following code block is a generic SQL syntax of the SELECT command along with the ORDER BY clause to sort the data from a MySQL table.

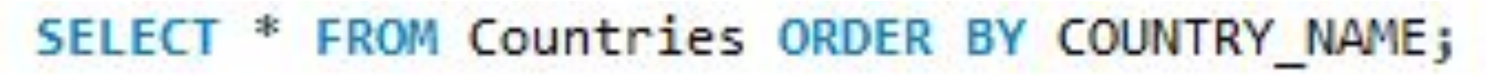
SELECT field1, field2,...fieldN table_name1, table_name2... ORDER BY field1, [field2...] [ASC [DESC]]

- You can sort the returned result on any field, if that field is being listed out.
- You can sort the result on more than one field.
- You can use the keyword ASC or DESC to get result in ascending or descending order. By default, it's the ascending order.
- You can use the WHERE...LIKE clause in the usual way to put a condition.

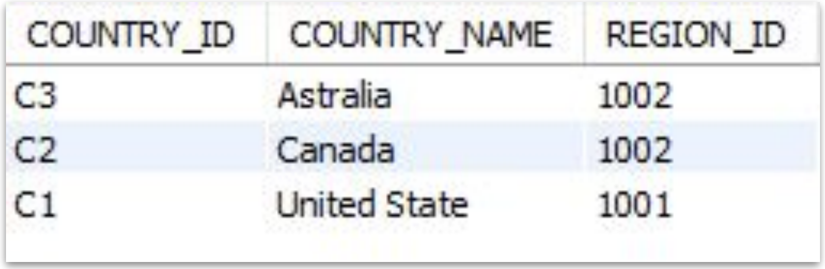
Basic SQL: ORDER BY Example



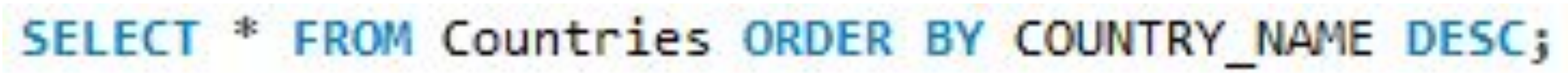
	COUNTRY_ID	COUNTRY_NAME	REGION_ID
►	C1	United State	1001
	C2	Canada	1002
	C3	Astralia	1002



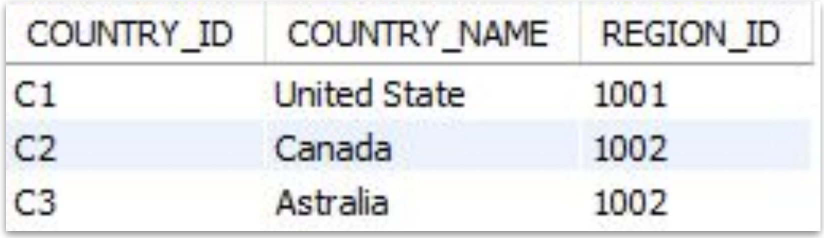
```
SELECT * FROM Countries ORDER BY COUNTRY_NAME;
```



COUNTRY_ID	COUNTRY_NAME	REGION_ID
C3	Astralia	1002
C2	Canada	1002
C1	United State	1001



```
SELECT * FROM Countries ORDER BY COUNTRY_NAME DESC;
```



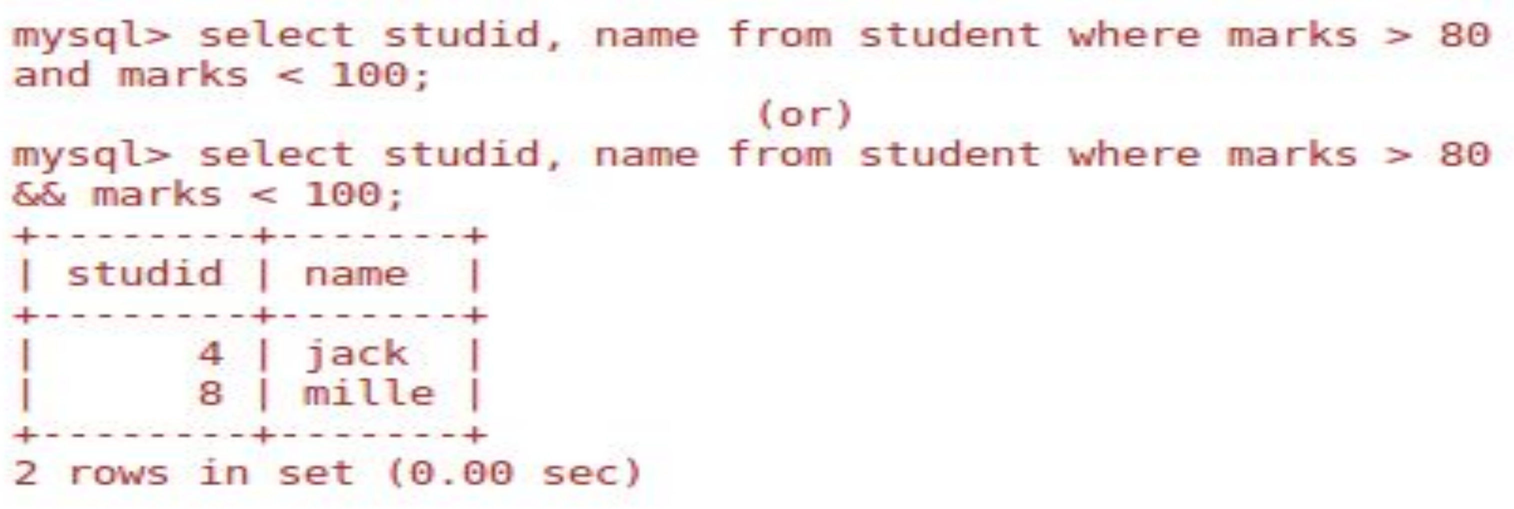
COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C2	Canada	1002
C3	Astralia	1002

Logical Operators

MySQL supports the following logical operations :

- ➢ AND(&&) Operator
- ➢ OR(||) Operator
- ➢ NOT(!) Operator

MySQL AND(&&) Operator : The logical AND(&&) operator indicates whether the both operands are true. Lets see a statement using AND operator.

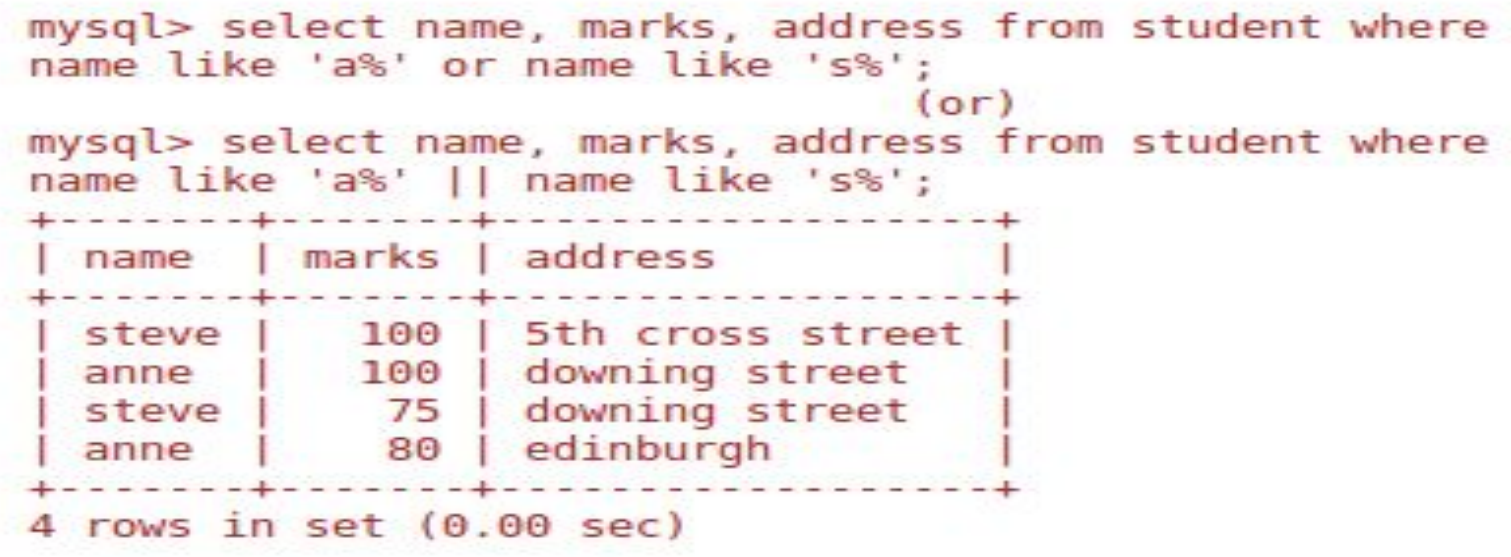


```
mysql> select studid, name from student where marks > 80
and marks < 100;
                                  (or)
mysql> select studid, name from student where marks > 80
&& marks < 100;
+--------+-------+
| studid | name  |
+--------+-------+
|      4 | jack  |
|      8 | mille |
+--------+-------+
2 rows in set (0.00 sec)
```

In the above example it will list the studid and name of the student who have secured more than 80 and less than 100.

Logical Operators

MySQL OR(||) Operator :
The logical OR(||) operator indicates whether either operand is true. Lets see a statement using OR operator.

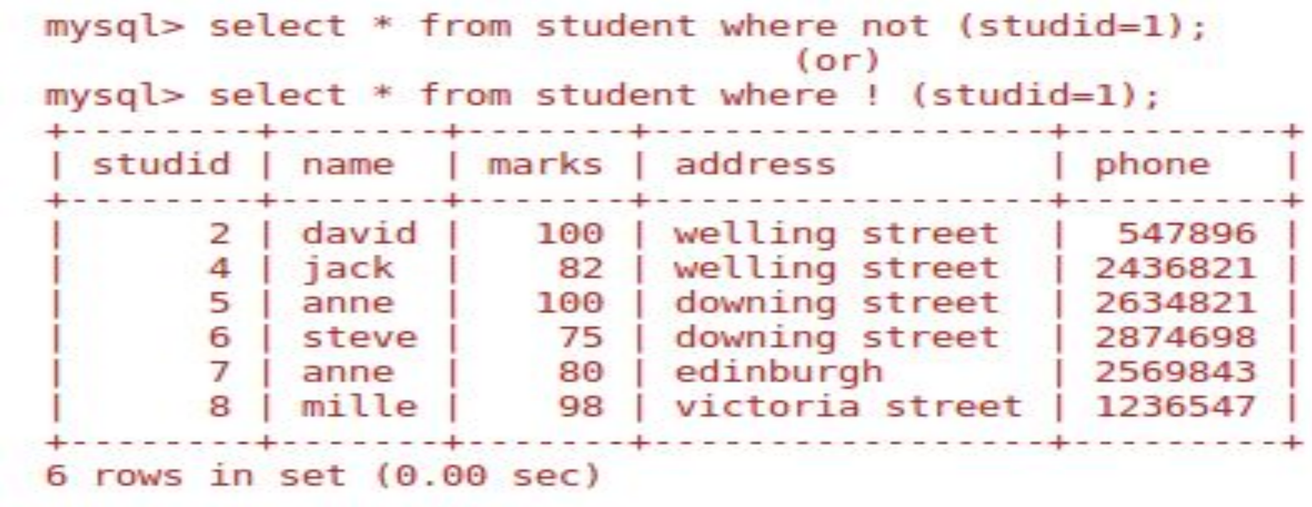


```
mysql> select name, marks, address from student where
name like 'a%' or name like 's%';
                                            (or)
mysql> select name, marks, address from student where
name like 'a%' || name like 's%';
+-------+-------+-------------------+
| name  | marks | address           |
+-------+-------+-------------------+
| steve |   100 | 5th cross street  |
| anne  |   100 | downing street    |
| steve |    75 | downing street    |
| anne  |    80 | edinburgh         |
+-------+-------+-------------------+
4 rows in set (0.00 sec)
```

In the above statement it will list the name, marks and address of the student whose name starts with the letter A and S.

Logical Operators

MySQL NOT(!) Operator : The logical NOT(!) operator have only one operand and it returns the inverse of the value.



```
mysql> select * from student where not (studid=1);
                                  (or)
mysql> select * from student where ! (studid=1);
+--------+-------+-------+------------------+---------+
| studid | name  | marks | address          | phone   |
+--------+-------+-------+------------------+---------+
|      2 | david |   100 | welling street   |  547896 |
|      4 | jack  |    82 | welling street   | 2436821 |
|      5 | anne  |   100 | downing street   | 2634821 |
|      6 | steve |    75 | downing street   | 2874698 |
|      7 | anne  |    80 | edinburgh        | 2569843 |
|      8 | mille |    98 | victoria street  | 1236547 |
+--------+-------+-------+------------------+---------+
6 rows in set (0.00 sec)
```

It will list all the student details except the studid 1.

Comparison Operators

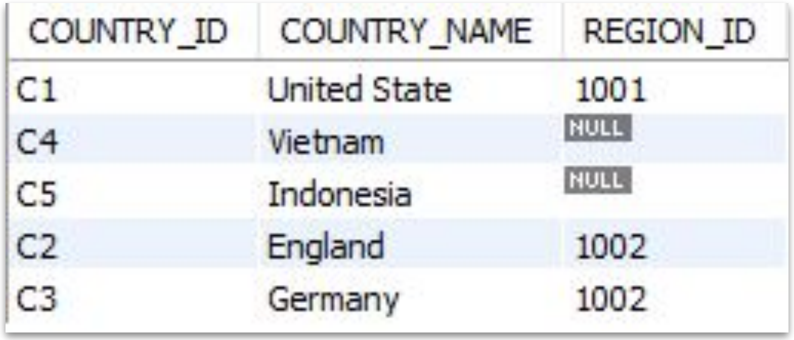
Comparison operators are used in the WHERE clause to determine which records to select. Here is a list of the comparison operators that you can use in MySQL:

Comparison Operator	Description
=	Equal
<=>	Equal (Safe to compare NULL values)
<>	Not Equal
!=	Not Equal
>	Greater Than
>=	Greater Than or Equal
<	Less Than
<=	Less Than or Equal
IN ( )	Matches a value in a list
NOT	Negates a condition
BETWEEN	Within a range (inclusive)
IS NULL	NULL value
IS NOT NULL	Non-NULL value
LIKE	Pattern matching with % and _

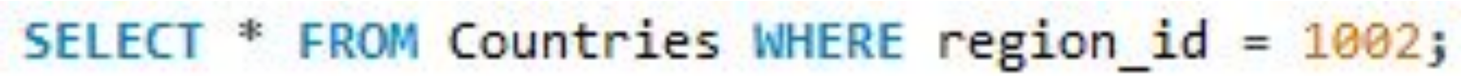
Comparison Operators: Equality Operator

In MySQL, you can use the = operator to test for equality in a query. The = operator can only test equality with values that are not NULL.

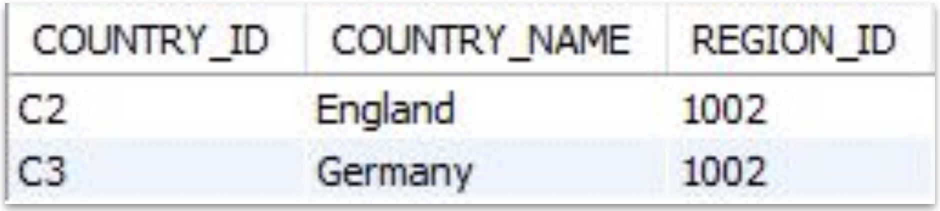
For example:



COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C4	Vietnam	NULL
C5	Indonesia	NULL
C2	England	1002
C3	Germany	1002



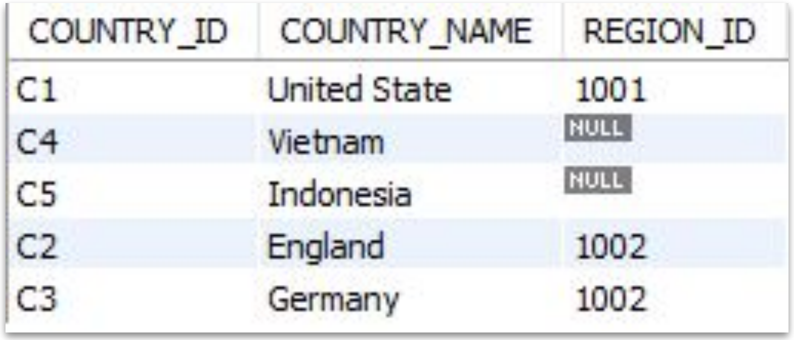
```
SELECT * FROM Countries WHERE region_id = 1002;
```



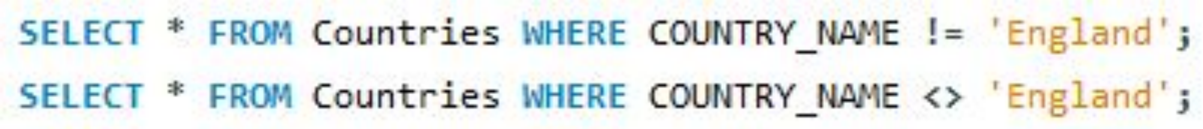
COUNTRY_ID	COUNTRY_NAME	REGION_ID
C2	England	1002
C3	Germany	1002

Comparison Operators: Inequality Operator

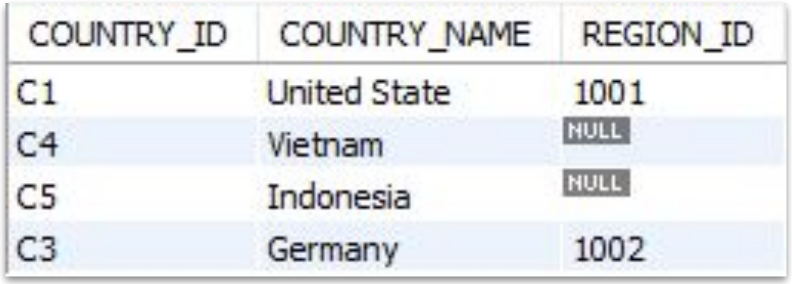
In MySQL, you can use the <> or != operators to test for inequality in a query. For example, we could test for inequality using the <> operator, as follows:



COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C4	Vietnam	NULL
C5	Indonesia	NULL
C2	England	1002
C3	Germany	1002



```
SELECT * FROM Countries WHERE COUNTRY_NAME != 'England';
SELECT * FROM Countries WHERE COUNTRY_NAME <> 'England';
```

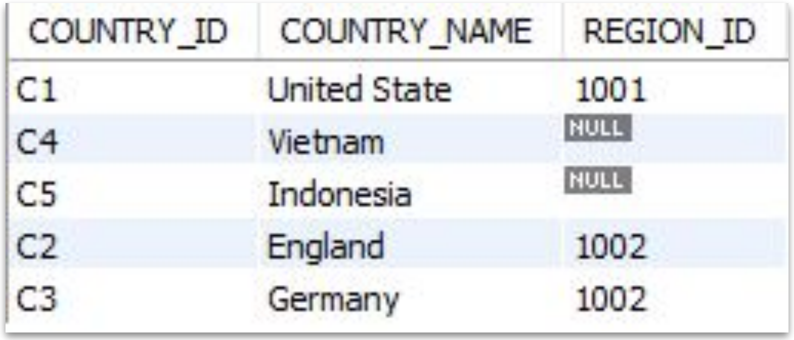


COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C4	Vietnam	NULL
C5	Indonesia	NULL
C3	Germany	1002

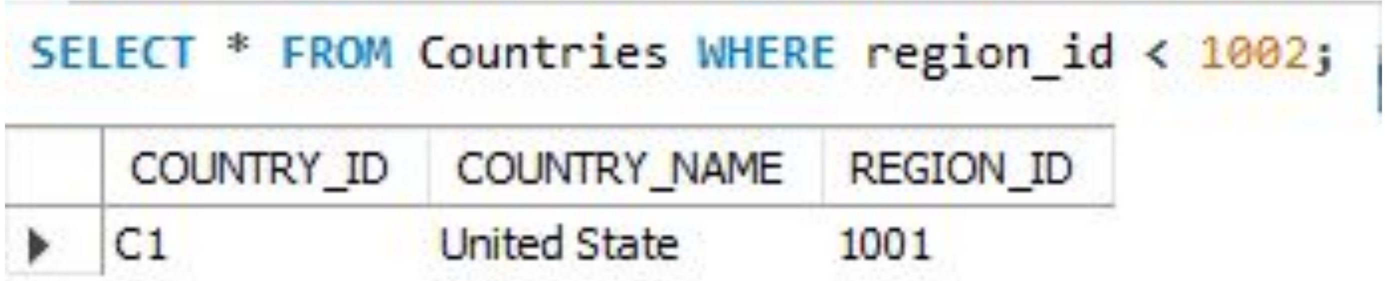
Both of these queries would return the same results.

Comparison Operators: Comparison Operator

You can use the < operator in MySQL to test for an expression less than.



COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C4	Vietnam	NULL
C5	Indonesia	NULL
C2	England	1002
C3	Germany	1002



```
SELECT * FROM Countries WHERE region_id < 1002;
```

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
▶	C1	United State	1001

Comparison Operators: IN Condition

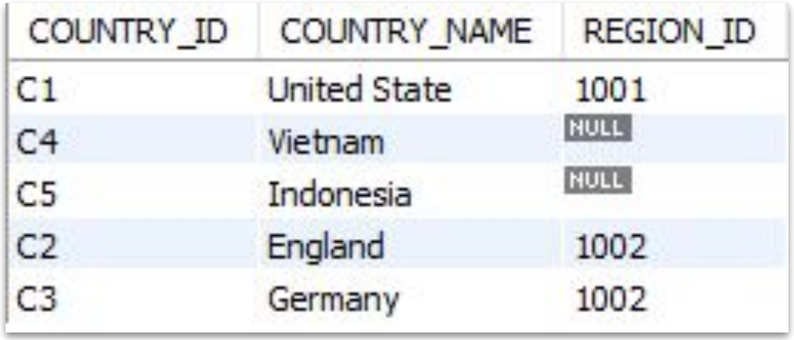
The MySQL IN condition is used to help reduce the need to use multiple OR conditions in a SELECT, INSERT, UPDATE, or DELETE statement.

Syntax:

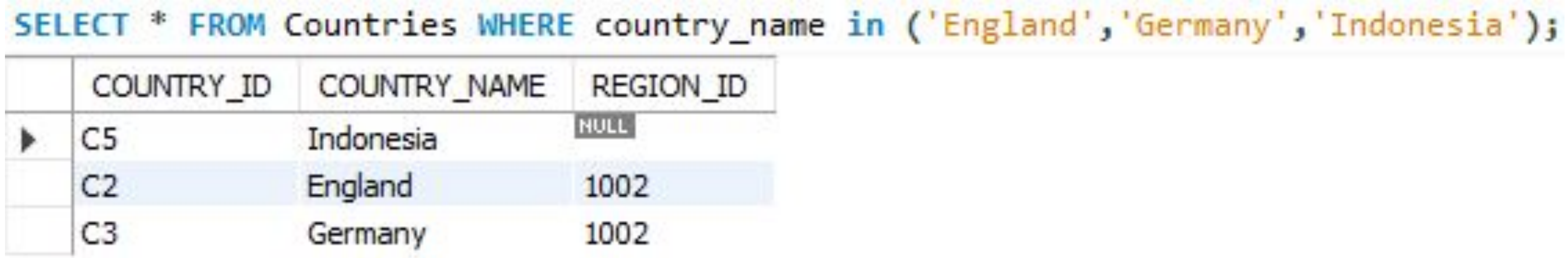
expression IN (value1, value2, ... value_n);

Expression: The value to test.

value1, value2, ... or value_n: These are the values to test against expression. If any of these values matches expression, then the IN condition will evaluate to true. This is a quick method to test if any one of the values matches expression.



COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C4	Vietnam	NULL
C5	Indonesia	NULL
C2	England	1002
C3	Germany	1002



```
SELECT * FROM Countries WHERE country_name in ('England','Germany','Indonesia');
```

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
▶	C5	Indonesia	NULL
	C2	England	1002
	C3	Germany	1002

This MySQL IN condition example would return all rows from the countries table where the name is either England, Germany or United State

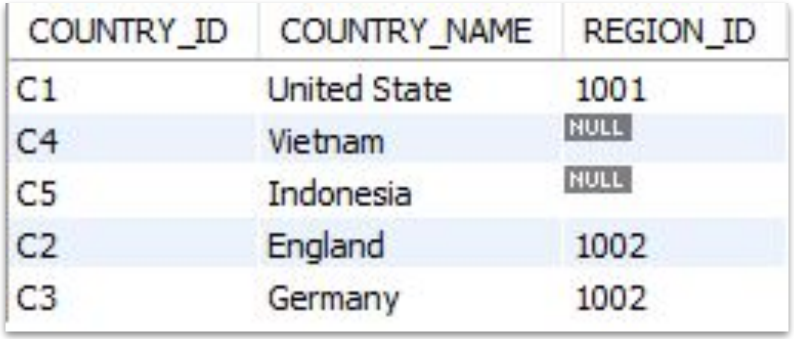
Comparison Operators: NOT

The MySQL NOT Condition (also called the NOT Operator) is used to negate a condition in a SELECT, INSERT, UPDATE, or DELETE statement.

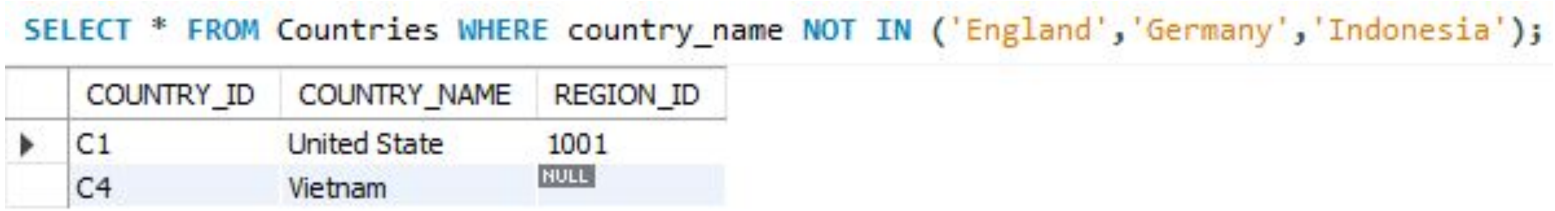
Syntax

NOT condition

For example:



COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C4	Vietnam	NULL
C5	Indonesia	NULL
C2	England	1002
C3	Germany	1002



```
SELECT * FROM Countries WHERE country_name NOT IN ('England','Germany','Indonesia');
```

COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C4	Vietnam	NULL

This MySQL NOT example would return all rows from the country table where the name is not Joseph, Andrew, or Brad.

Comparison Operators: BETWEEN Condition

The MySQL BETWEEN Condition is used to retrieve values within a range in a SELECT, INSERT, UPDATE, or DELETE statement.

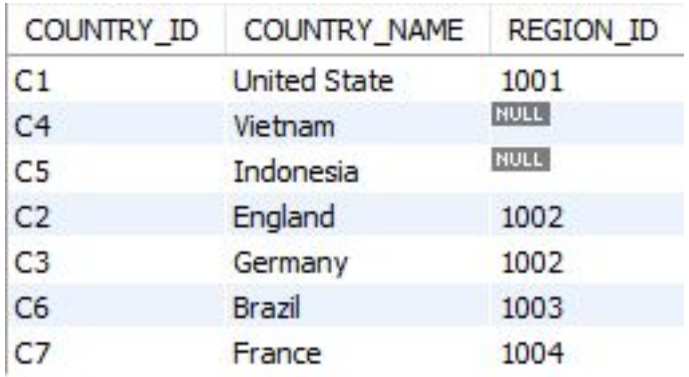
Syntax:

expression BETWEEN value1 AND value2;

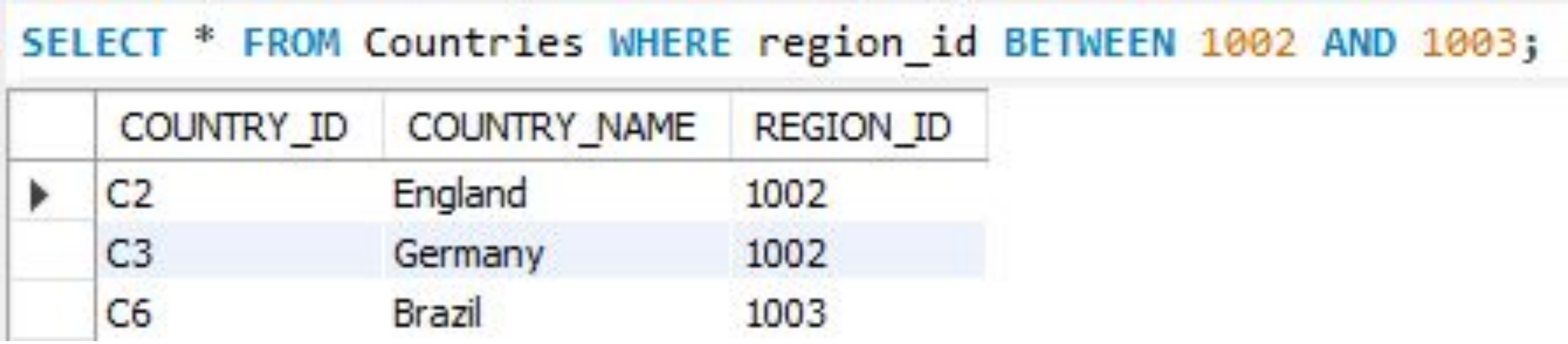
Expression: A column or calculation.

value1 and value2: These values create an inclusive range that expression is compared to.

For example:



COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C4	Vietnam	NULL
C5	Indonesia	NULL
C2	England	1002
C3	Germany	1002
C6	Brazil	1003
C7	France	1004



```
SELECT * FROM Countries WHERE region_id BETWEEN 1002 AND 1003;
```

COUNTRY_ID	COUNTRY_NAME	REGION_ID
C2	England	1002
C3	Germany	1002
C6	Brazil	1003

This MySQL BETWEEN example would return all rows from the contacts table where the region_id is between 1002 and 1002 (inclusive)

Comparison Operators: IS NULL Condition

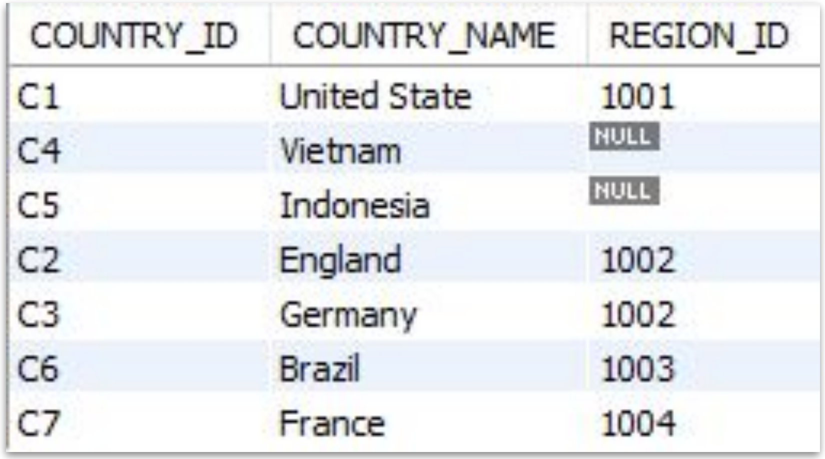
The MySQL IS NULL Condition is used to test for a NULL value in a SELECT, INSERT, UPDATE, or DELETE statement.

Syntax

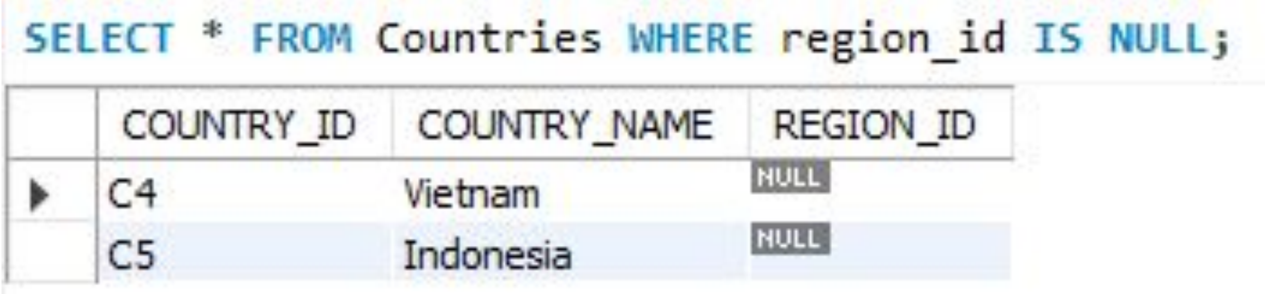
expression IS NULL

Expression: The value to test if it is a NULL value.

Let's look at an example of how to use MySQL IS NULL in a SELECT statement:



COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C4	Vietnam	NULL
C5	Indonesia	NULL
C2	England	1002
C3	Germany	1002
C6	Brazil	1003
C7	France	1004



```
SELECT * FROM Countries WHERE region_id IS NULL;
```

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
▶	C4	Vietnam	NULL
	C5	Indonesia	NULL

This MySQL IS NULL example will return all records from the contacts table where the region_id contains a NULL value.

Comparison Operators: IS NOT NULL

The MySQL IS NOT NULL condition is used to test for a NOT NULL value in a SELECT, INSERT, UPDATE, or DELETE statement.

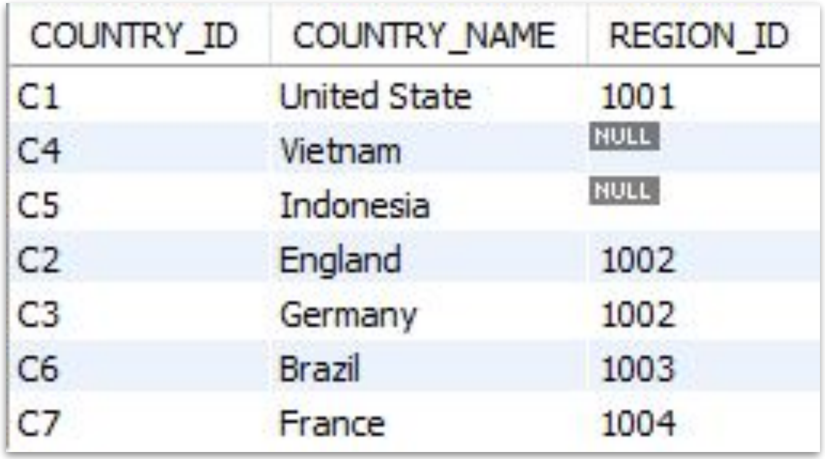
Syntax

expression IS NOT NULL

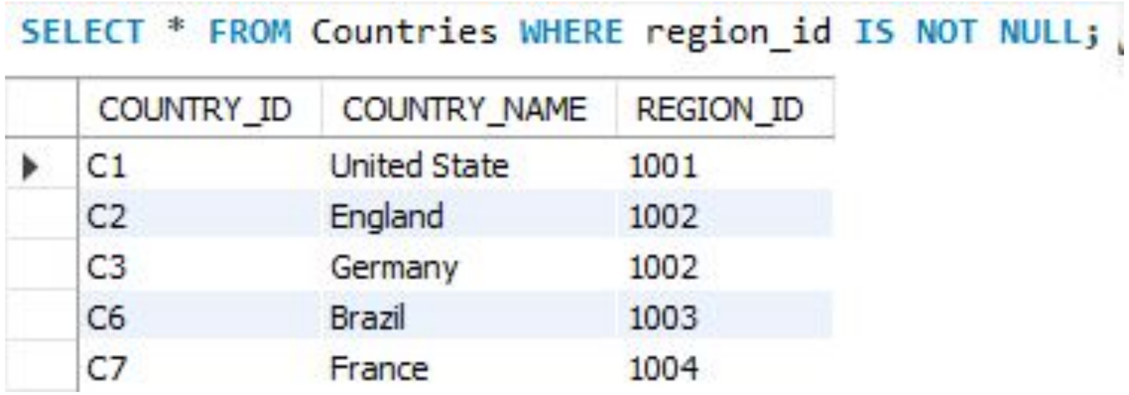
Expression: The value to test if it is a not NULL value.

Example - With SELECT Statement

Here is an example of how to use the MySQL IS NOT NULL condition in a SELECT statement:



COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C4	Vietnam	NULL
C5	Indonesia	NULL
C2	England	1002
C3	Germany	1002
C6	Brazil	1003
C7	France	1004



```sql
SELECT * FROM Countries WHERE region_id IS NOT NULL;
```

COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C2	England	1002
C3	Germany	1002
C6	Brazil	1003
C7	France	1004

This MySQL IS NOT NULL example will return all records from the countries table where the region_id does not contain a null value.

Comparison Operators: LIKE Condition

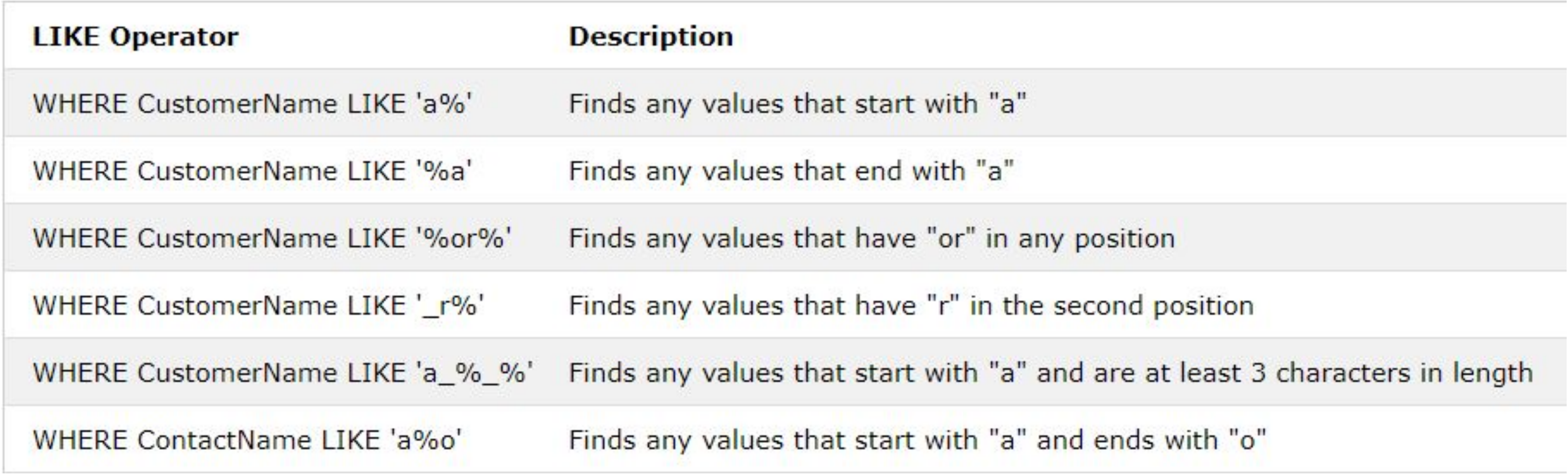
The MySQL LIKE operator is used in a WHERE clause to search for a specified pattern in a column

Pattern: There are two wildcards often used in conjunction with the LIKE operator:

Wildcard	Explanation
%	Allows you to match any string of any length (including zero length)
_	Allows you to match on a single character

Comparison Operators: LIKE Condition

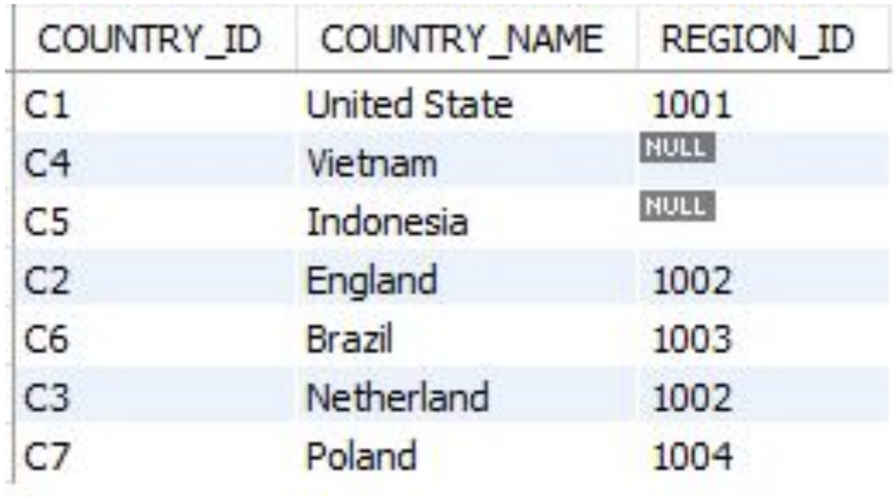
Some examples of using like operator



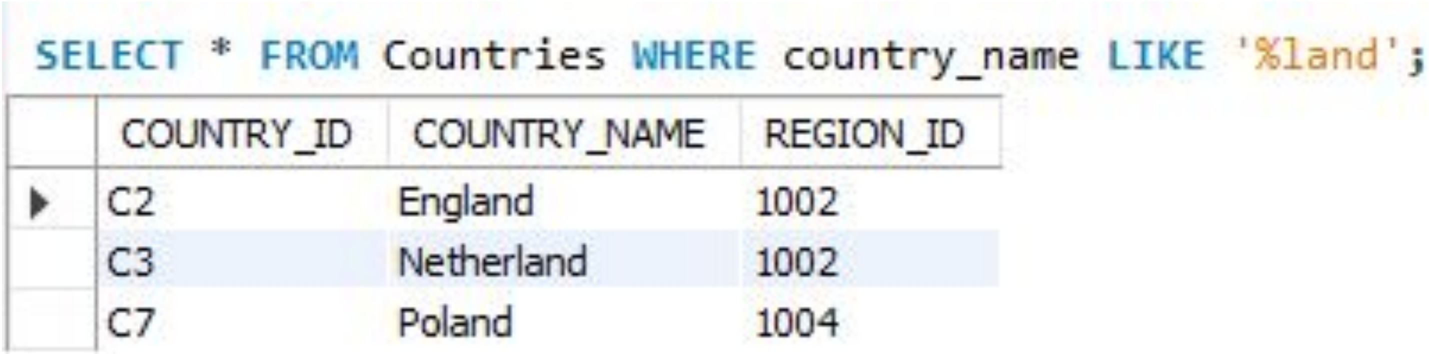
LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Comparison Operators: LIKE Condition

Example: Display all country names that ends with land



COUNTRY_ID	COUNTRY_NAME	REGION_ID
C1	United State	1001
C4	Vietnam	NULL
C5	Indonesia	NULL
C2	England	1002
C6	Brazil	1003
C3	Netherland	1002
C7	Poland	1004



```
SELECT * FROM Countries WHERE country_name LIKE '%land';
```

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
▶	C2	England	1002
	C3	Netherland	1002
	C7	Poland	1004

Joins

MySQL joins are used to combine columns from two or more tables, based on a common field between them.

Types of Joins:

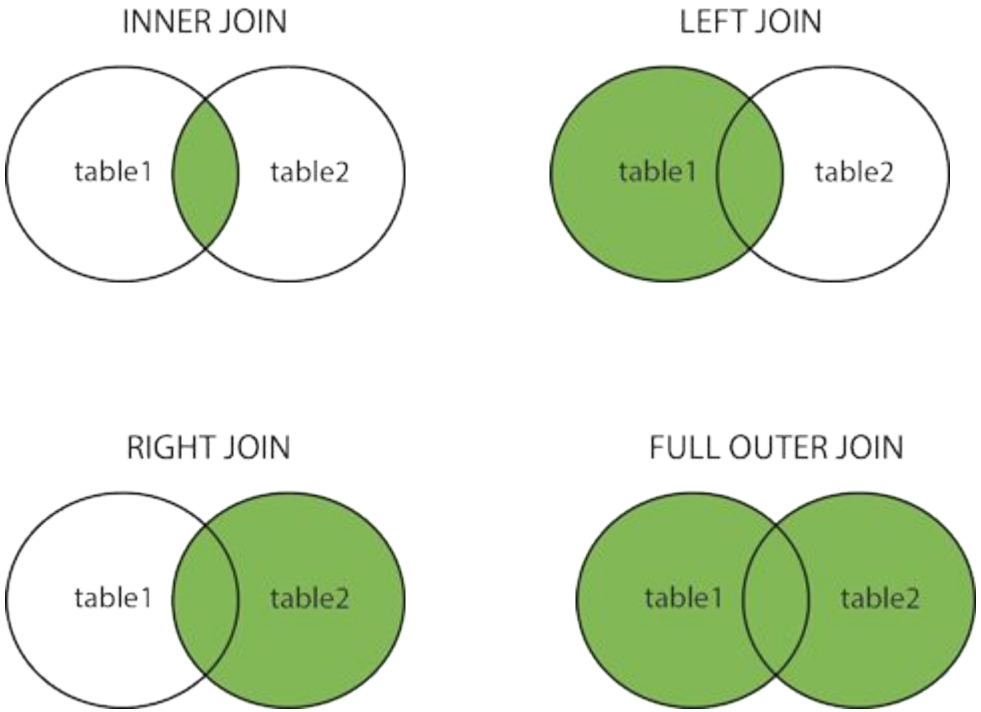
- Inner join
- Outer join
  - Left outer join
  - Right outer join
  - Full outer join
- Cross join
- Self join

Joins - Types of Joins:

Inner join

- The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables
- Outer join
  - Left outer join: LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.
  - Right outer join: RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match
  - Full outer join : The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

    The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins
- Cross join
- Self join

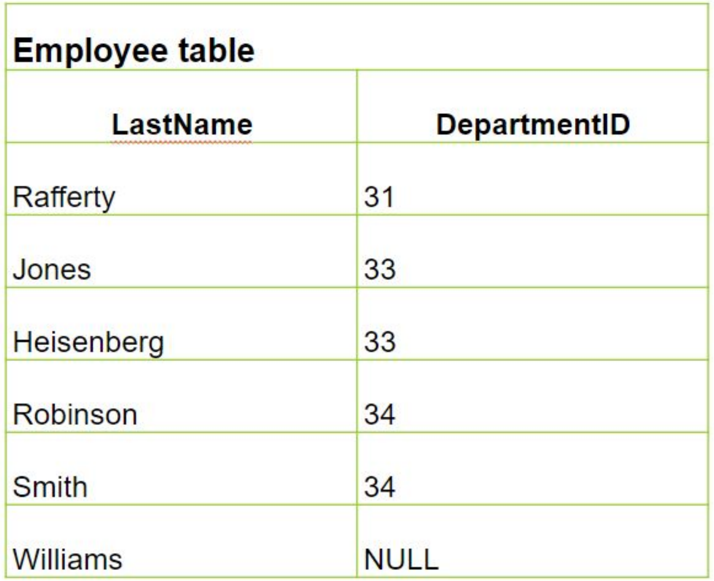


Joins - Sample Table

Employee table	
LastName	**DepartmentID**
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

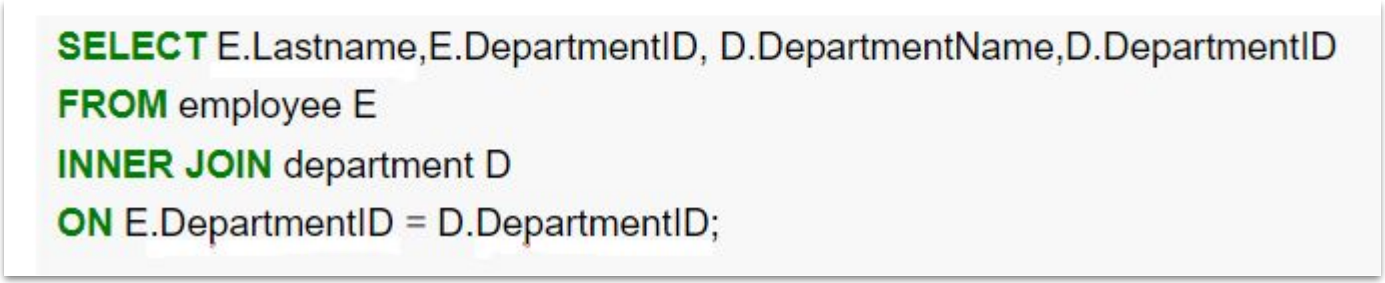
Department table	
DepartmentID	**DepartmentName**
31	Sales
33	Engineering
34	Clerical
35	Marketing

Joins - Inner Join

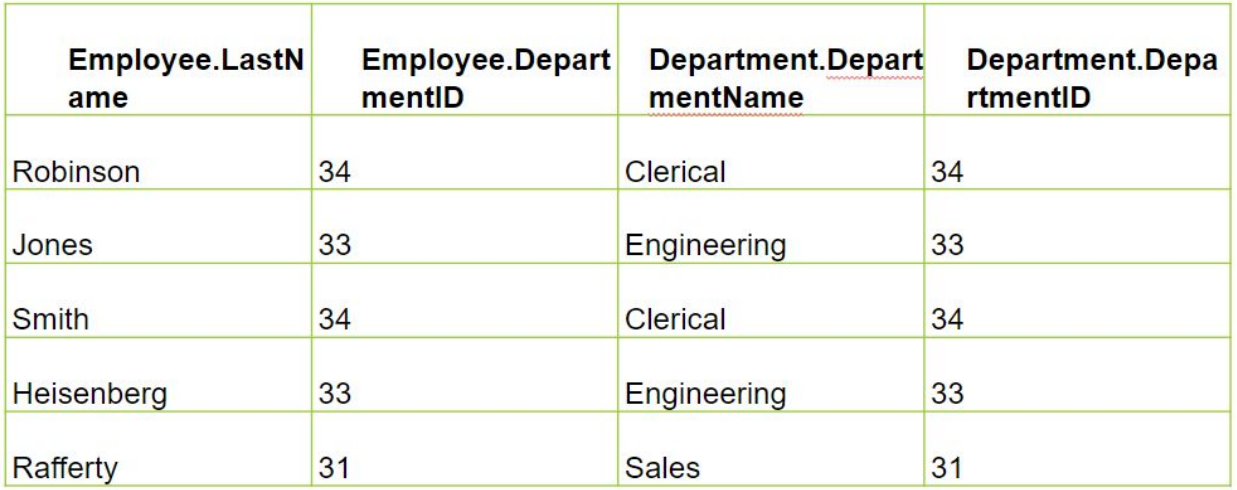


Employee table

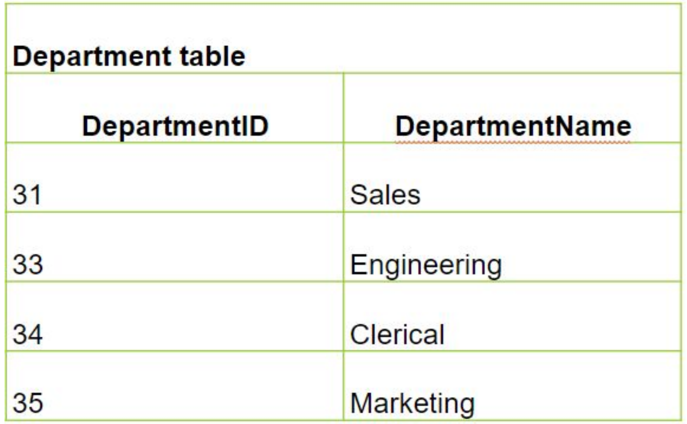
LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL



```
SELECT E.Lastname,E.DepartmentID, D.DepartmentName,D.DepartmentID
FROM employee E
INNER JOIN department D
ON E.DepartmentID = D.DepartmentID;
```



Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Robinson	34	Clerical	34
Jones	33	Engineering	33
Smith	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31

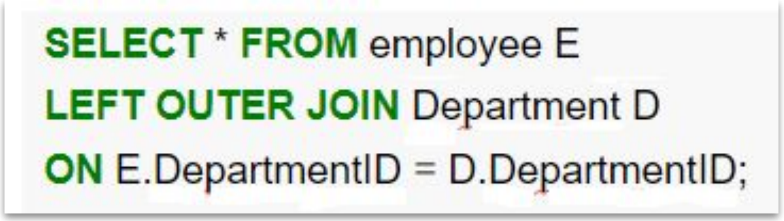


Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Joins - Left Outer Join

Employee table	
LastName	**DepartmentID**
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

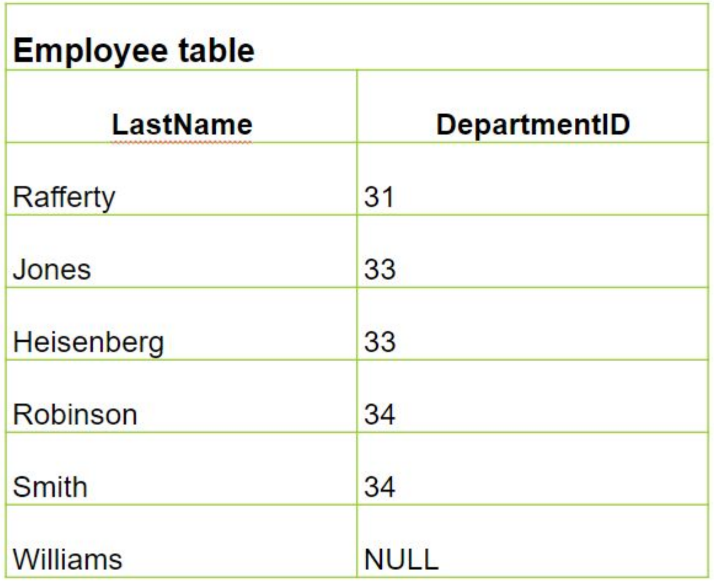


```
SELECT * FROM employee E
LEFT OUTER JOIN Department D
ON E.DepartmentID = D.DepartmentID;
```

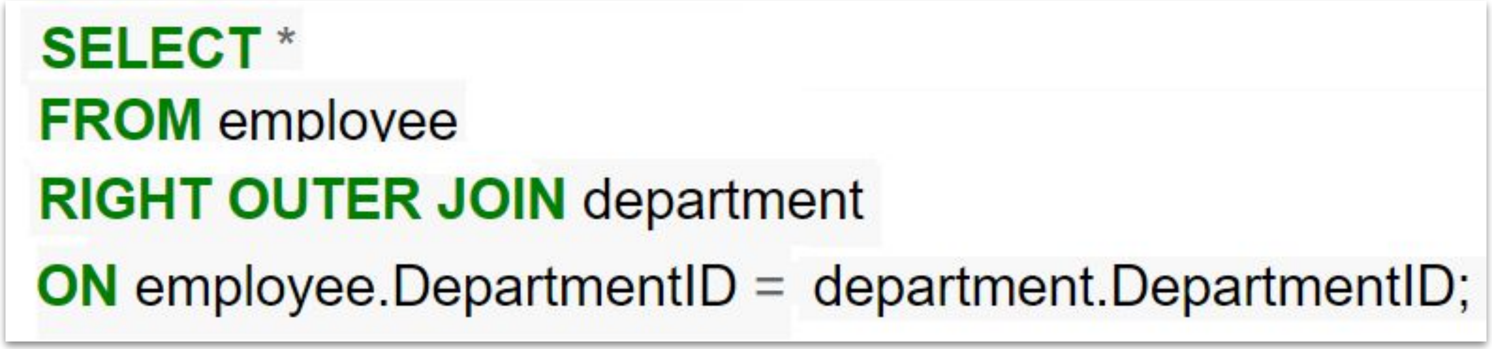
Department table	
DepartmentID	**DepartmentName**
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
Williams	NULL	NULL	NULL
Heisenberg	33	Engineering	33

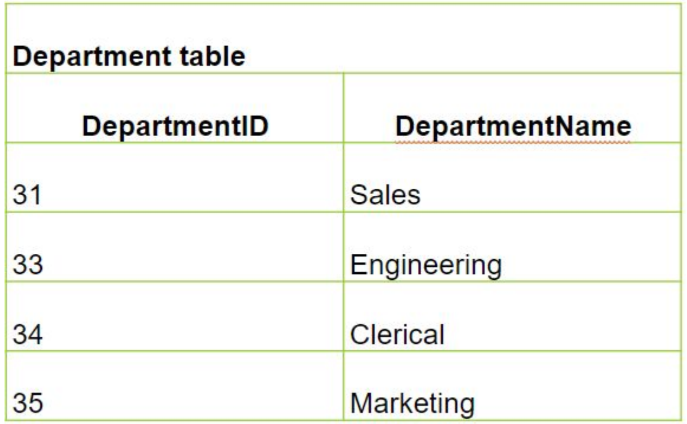
Joins - Right Outer Join



Employee table	
LastName	**DepartmentID**
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL



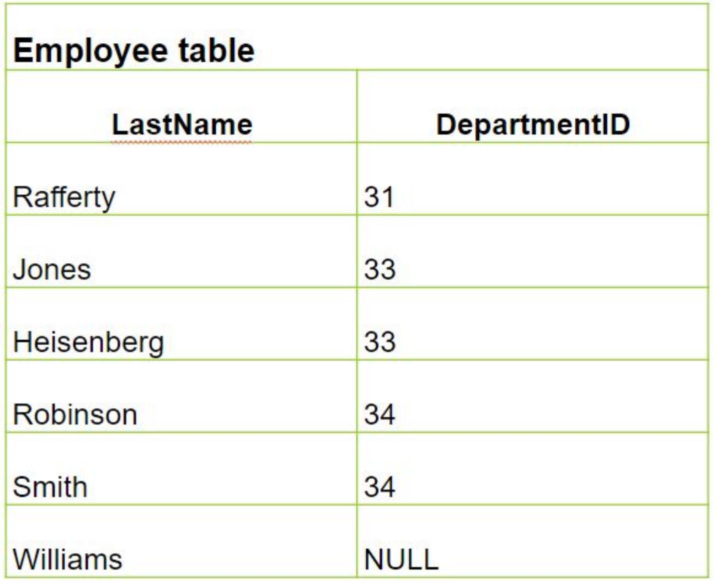
```
SELECT *
FROM employee
RIGHT OUTER JOIN department
ON employee.DepartmentID = department.DepartmentID;
```



Department table	
DepartmentID	**DepartmentName**
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	Marketing	35

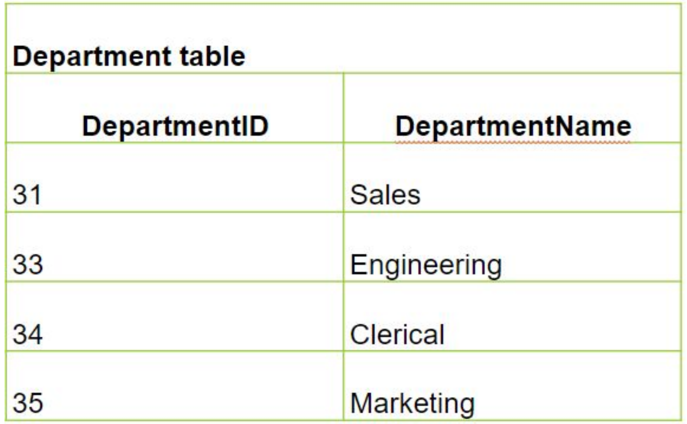
Joins - Full Outer Join



Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

```
SELECT * FROM employee
FULL OUTER JOIN department
ON employee.DepartmentID = department.DepartmentID;
```



Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.Department Name	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Williams	NULL	NULL	NULL
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	Marketing	35

Joins - CROSS JOIN

Returns the Cartesian product of rows from tables in the join. In other words, it will produce rows which combine each row from the first table with each row from the second table

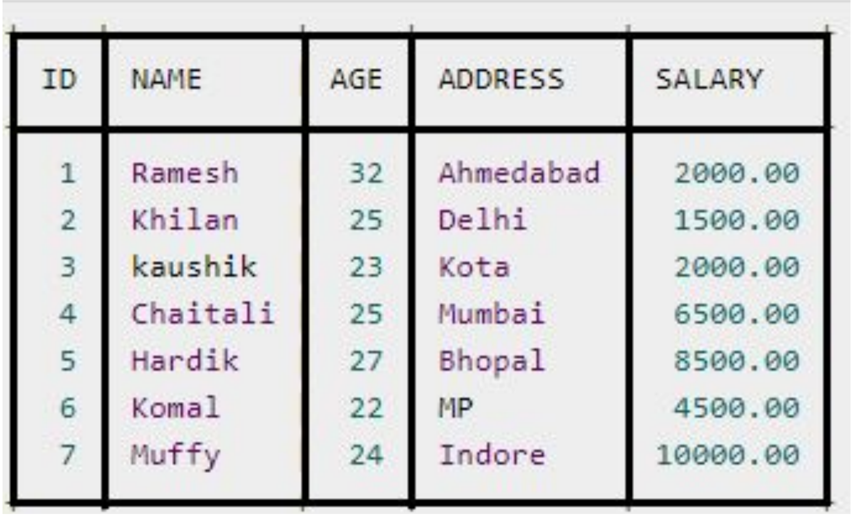
```
SELECT * FROM employee
CROSS JOIN department;
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Rafferty	31	Sales	31
Jones	33	Sales	31
Heisenberg	33	Sales	31
Smith	34	Sales	31
Robinson	34	Sales	31
Williams	NULL	Sales	31
Rafferty	31	Engineering	33
Jones	33	Engineering	33
Heisenberg	33	Engineering	33
Smith	34	Engineering	33
Robinson	34	Engineering	33
Williams	NULL	Engineering	33
Rafferty	31	Clerical	34
Jones	33	Clerical	34
Heisenberg	33	Clerical	34
Smith	34	Clerical	34
Robinson	34	Clerical	34
Williams	NULL	Clerical	34
Rafferty	31	Marketing	35
Jones	33	Marketing	35
Heisenberg	33	Marketing	35
Smith	34	Marketing	35
Robinson	34	Marketing	35
Williams	NULL	Marketing	35

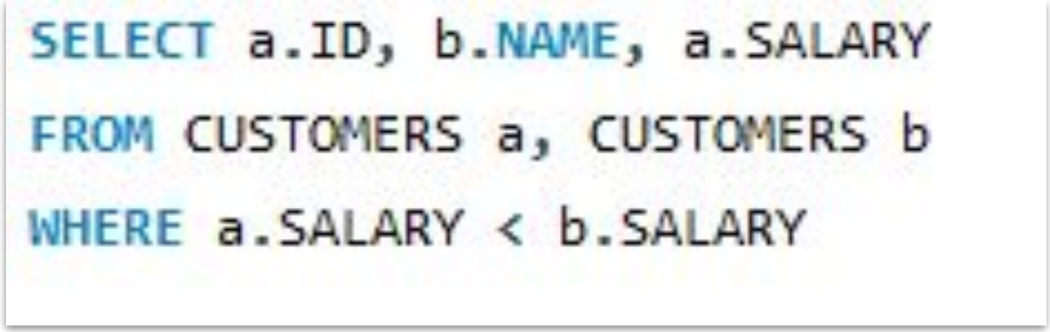
Joins - Self Join

A self-join is joining a table to itself

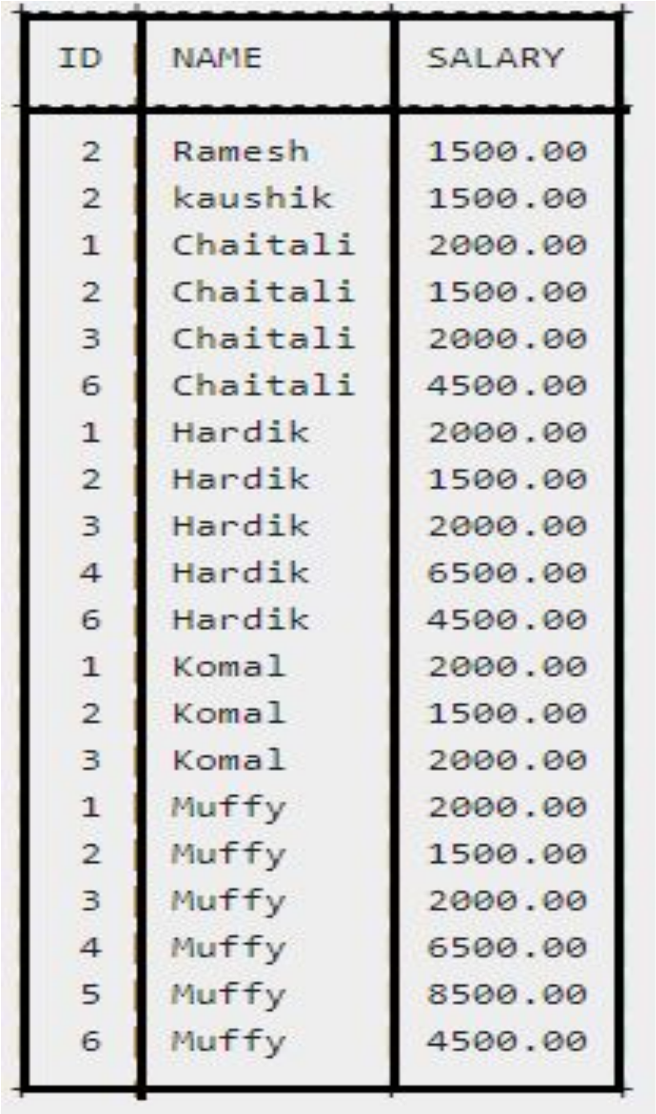
CUSTOMERS Table is as follows.



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



```
SELECT a.ID, b.NAME, a.SALARY
FROM CUSTOMERS a, CUSTOMERS b
WHERE a.SALARY < b.SALARY
```



ID	NAME	SALARY
2	Ramesh	1500.00
2	kaushik	1500.00
1	Chaitali	2000.00
2	Chaitali	1500.00
3	Chaitali	2000.00
6	Chaitali	4500.00
1	Hardik	2000.00
2	Hardik	1500.00
3	Hardik	2000.00
4	Hardik	6500.00
6	Hardik	4500.00
1	Komal	2000.00
2	Komal	1500.00
3	Komal	2000.00
1	Muffy	2000.00
2	Muffy	1500.00
3	Muffy	2000.00
4	Muffy	6500.00
5	Muffy	8500.00
6	Muffy	4500.00

Q&A