Databases Concepts

Objectives

- Why RDBMS?
- Different models (hierarchy, network, OO, relational)
- RDBMS concepts: tables, keys, relationships
- ER model concepts: entities, attributes, relationships, cardinality
- Mapping entity relationships to tables
- Normalization
- Using MySQL
- Data types in MySQL
- Create tables, basic DDL
- DDL commands (tables, relationships, keys, constraints)

Objectives

- DML (Insert, update, delete)
- Queries (select, predicates, identity)
- Complex queries
- Built-in MySQL functions
- Grouping and ordering
- Joins
- Subqueries

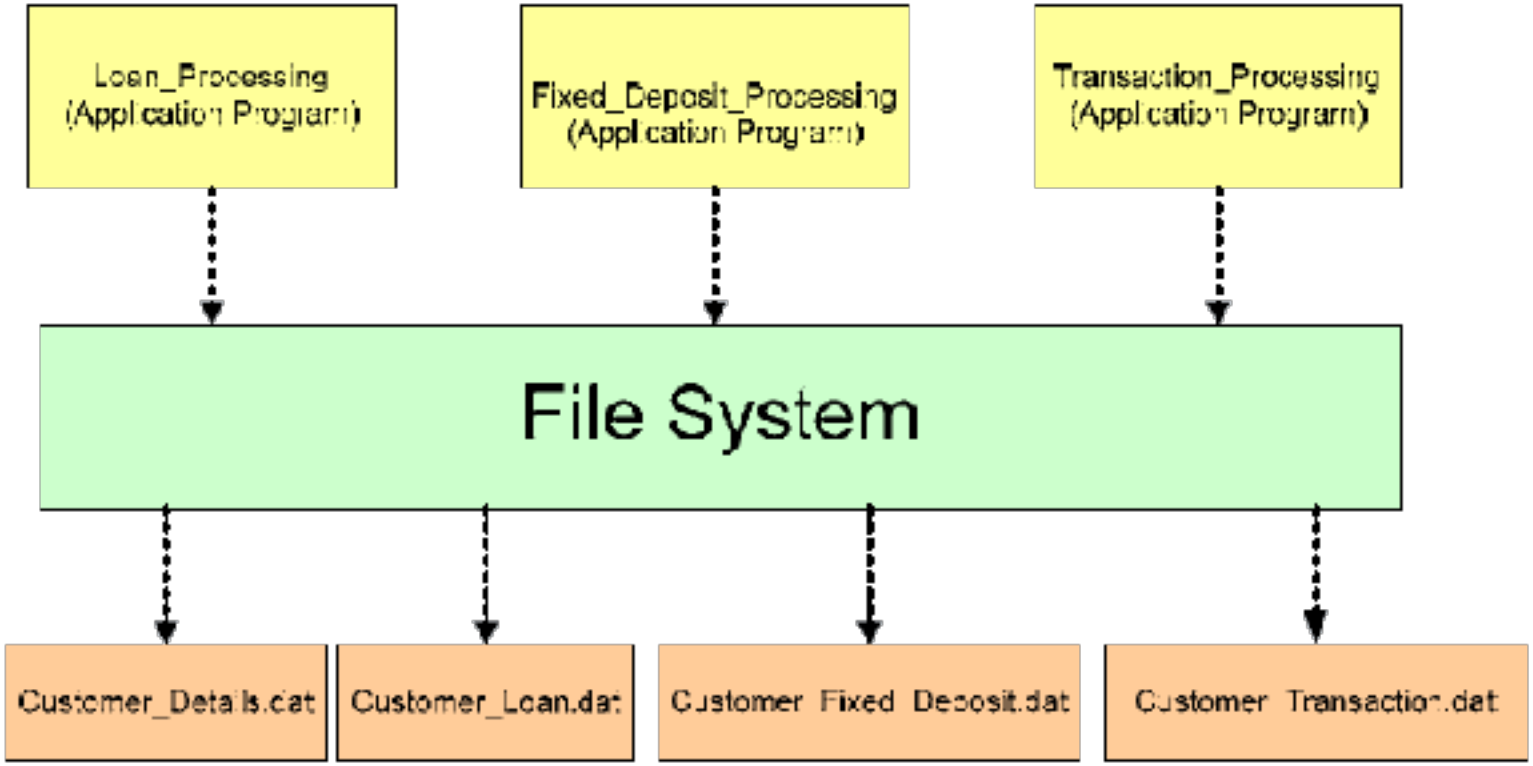
What is

- Data?
  - Meaningful facts, text, graphics, images, sound, video segments.
- Database?
  - An organized collection of logically related data.
- Information?
  - Data processed to be useful in decision making.
- Metadata?
  - Data that describes data.

Database

- Represents some aspect of the real world
  - Also known as the miniworld
- Changes to miniworld are reflected in the database
- Database has some source
  - some degree of interaction with events in real world

Traditional method of data storage



Limitations of traditional approach

- Data Dependence
- Data Redundancy (Duplication of data)
- Limited Data Sharing
- Lengthy Development Times
- Excessive Program Maintenance

Why DBMS?

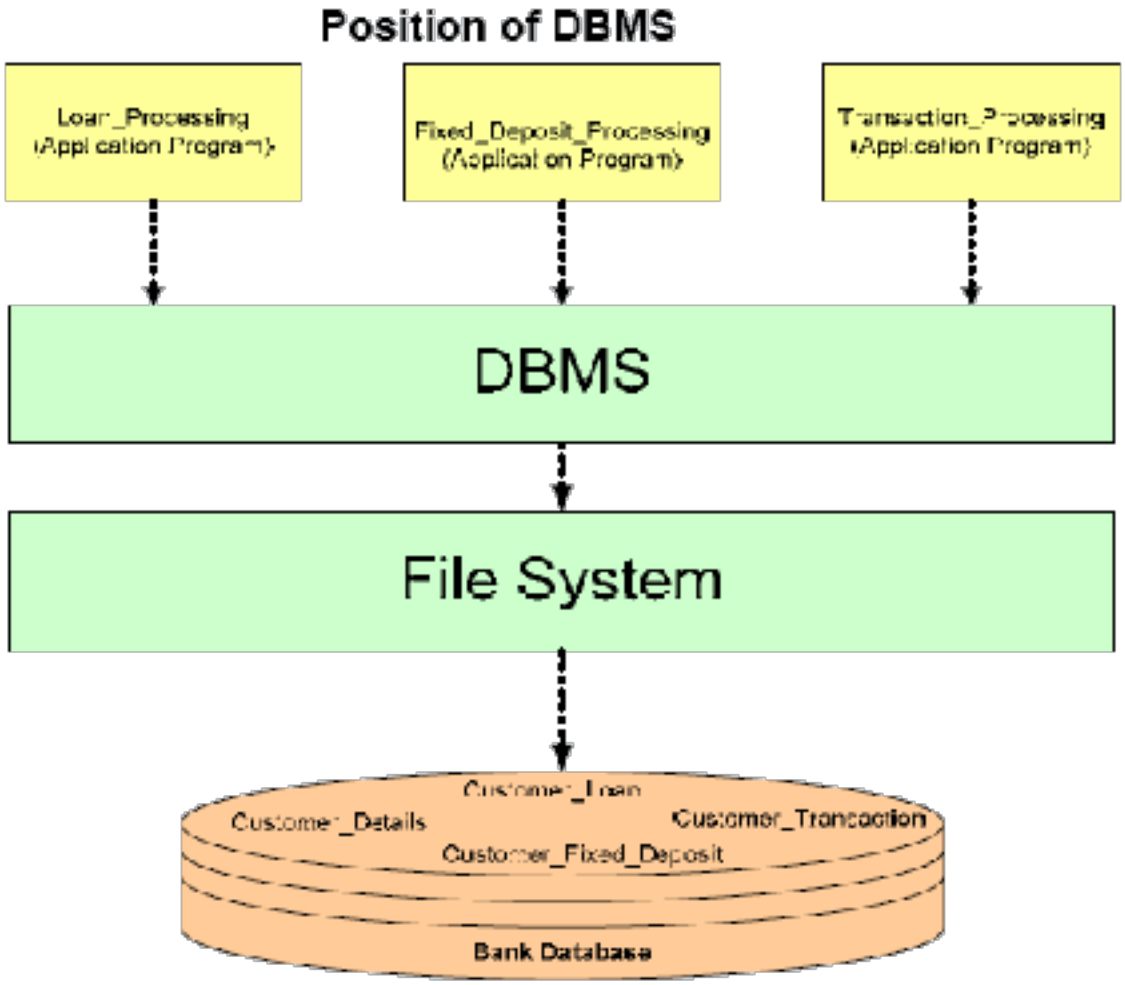
- Controlling redundancy
- Sharing of data
- Restricting unauthorized access
- Providing multiple interfaces
- Enforcing integrity constraints
- Providing backup & recovery
- Potential for enforcing standards
- Reduced application development time
- Availability of up-to-date Information

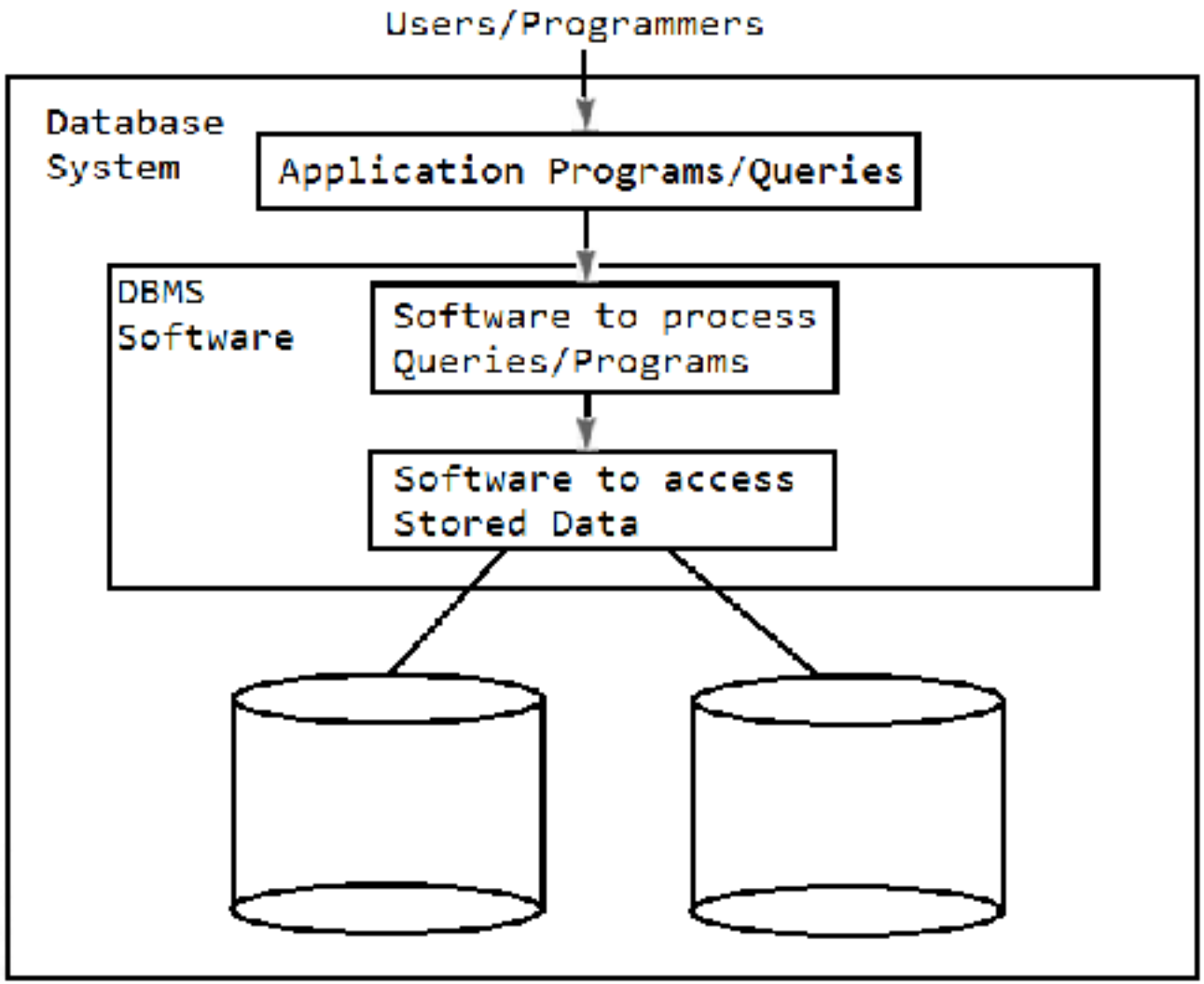
Database Management System

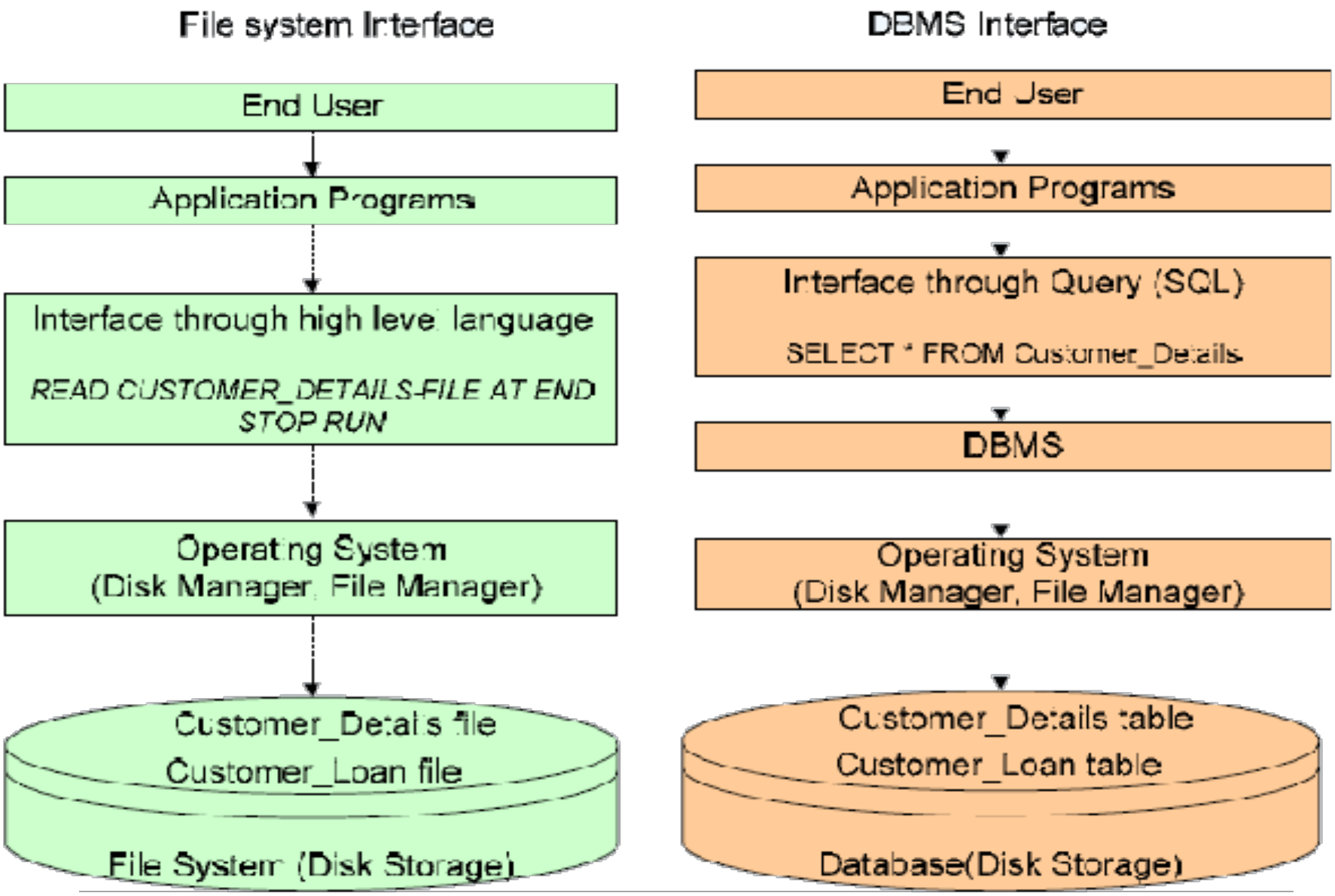
- Database Management System
  - A computerized record-keeping system
  - A data storage and retrieval system which permits data to be stored non-redundantly while making it appear to the user as if the data is well-integrated.
  - General purpose software system
  - Facilitates the processes of
    - Defining, constructing, and manipulating databases for various applications

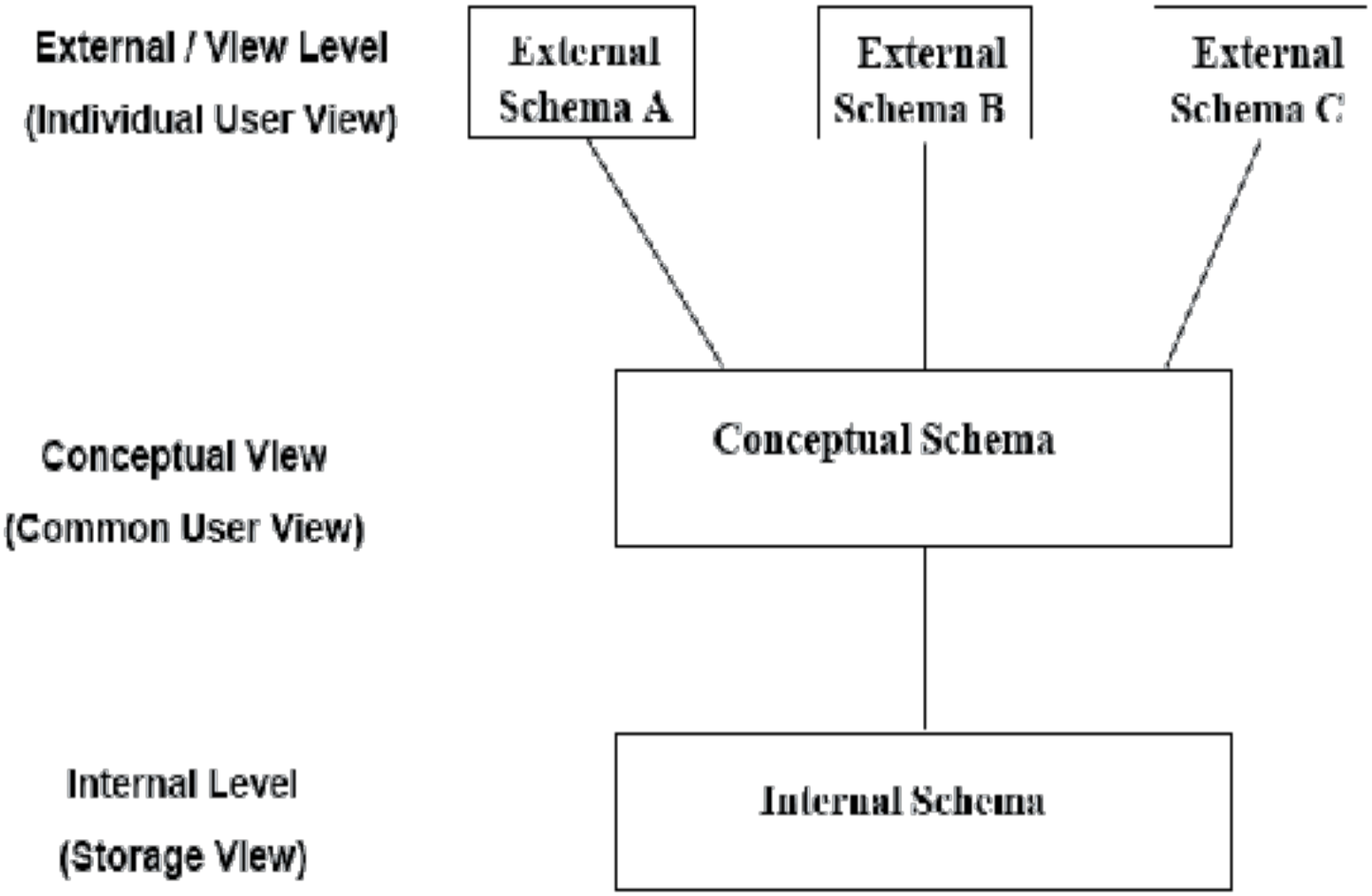
Database Management System

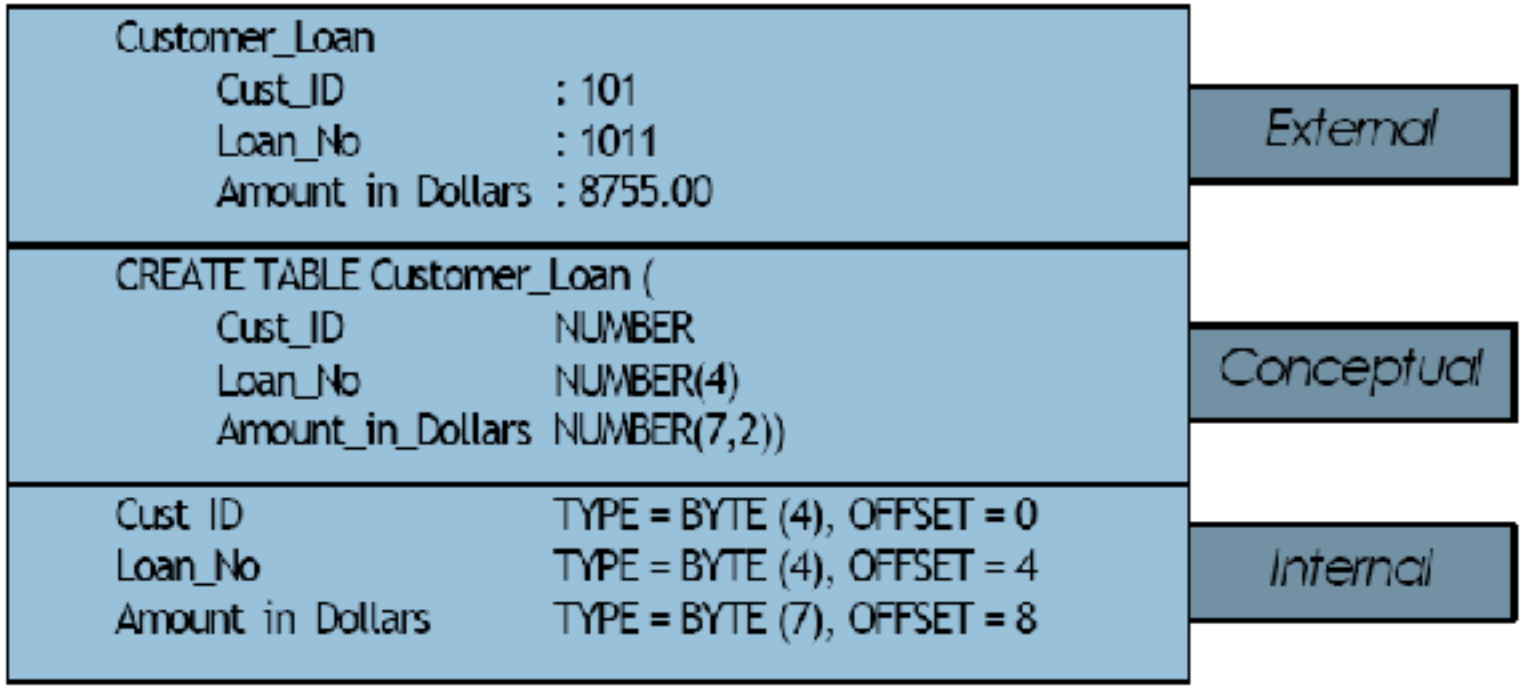
- Defining
  - specifying types of data and detailed description of each type
- Constructing
  - storing the data itself on some storage medium
- Manipulating:
  - Querying
  - Updating
  - Generating reports


Position of DBMS
Loan_Processing
(Application Program)
Fixed_Deposit_Processing
(Application Program)
Transaction_Processing
(Application Program)
DBMS
File System
Customer_Details
Customer_Fixed_Deposit
Customer_Trancaction
Bank Database


Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to process Queries/Programs
Software to access Stored Data


File system Interface
End User
Application Programs
Interface through high level language
READ CUSTOMER_DETAILS-FILE AT END
STOP RUN
Operating System
(Disk Manager, File Manager)
Customer_Details file
Customer_Loan file
File System (Disk Storage)
DBMS Interface
End User
Application Programs
Interface through Query (SQL)
SELECT * FROM Customer_Details
DBMS
Operating System
(Disk Manager, File Manager)
Customer_Details table
Customer_Loan table
Database(Disk Storage)


External / View Level
(Individual User View)
External Schema A
External Schema B
External Schema C
Conceptual View
(Common User View)
Conceptual Schema
Internal Level
(Storage View)
Internal Schema


Customer_Loan
Cust_ID : 101
Loan_No : 1011
Amount in Dollars : 8755.00
External
CREATE TABLE Customer_Loan (
Cust_ID NUMBER
Loan_No NUMBER(4)
Amount_in_Dollars NUMBER(7,2))
Conceptual
Cust ID TYPE = BYTE (4), OFFSET = 0
Loan_No TYPE = BYTE (4), OFFSET = 4
Amount in Dollars TYPE = BYTE (7), OFFSET = 8
Internal

Data Model (Self Learning)

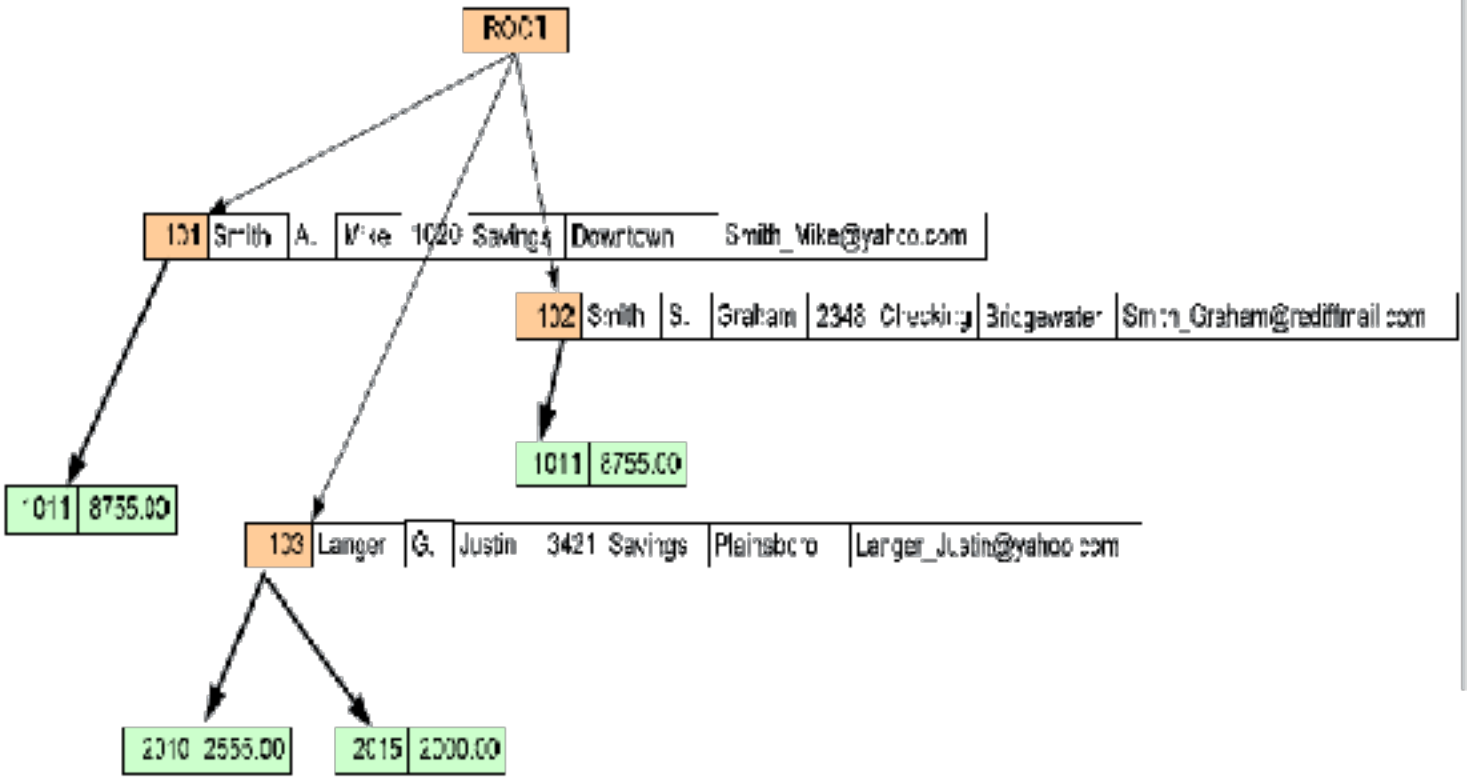
- Is a collection of concepts that can be used to describe the structure of a database
- High-level/Conceptual Model-provides concepts that are close to the way many users perceive data
- Low-level/physical Models-provides concepts that describe the details of how data is stored on computer
- Representational/implementation Models-provides concepts understood by end users , hides few details on haw data is stored

Evolution of Database Systems (Implementation Model)

- Hierarchical and Network (Legacy)
- Relational

(Self Learning)

Record based data model – Hierarchical data model



Drawbacks of Hierarchical Model (Self Learning)

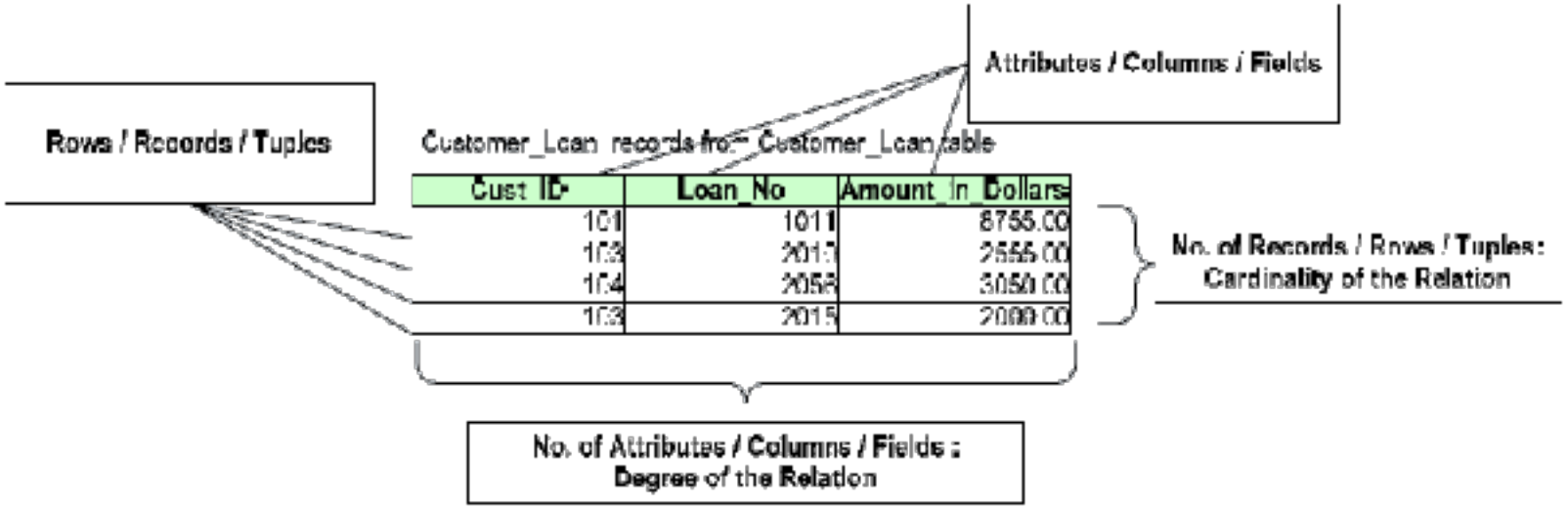
- Knowledge of the structure and its physical representation is required.
- Does not support for modifying the structure
- Time consuming as the processing is record at a time.

Record based data model – Network data model

Drawbacks of Network Model (Self Learning)

- Structure is inherently complex.
- Knowledge of the structure and its physical representation is required.
- Does not support for modifying the structure
- time consuming as the processing is record at a time.

Record based data model – Relational data model



Cust_ID	Cust_Last_Name	Cust_Mid_Name	Cust_First_Name	Account_No	Account_Type	Bank_Branch	Cust_Email
101	Smith	A.	Mike	1020	Savings	Downtown	Smith_Mike@yahoo.com
102	Smith	S.	Graham	2345	Checking	Bridgewater	Smith_Graham@rediffmail.com
103	Langer	C.	Justin	3471	Savings	Plainsboro	Langer_Justin@yahoo.com
104	Quails	D.	Jack	2337	Checking	Downtown	Quails_Jack@yahoo.com
105	Jones	E.	Simon	2309	Checking	Brighton	Jones_Simon@rediffmail.com

Customer_Detail records from Customer_Details table

RDBMS concepts: tables, keys, relationships

- Relation, Tuple, Attribute
  - Every relation has a unique name.
  - Every attribute value is atomic.
  - Every row/tuple is unique.
  - Attributes in tables have unique names.
  - Sometimes the value for a particular attribute may be unknown, or it may have no value. This is represented by a null
  - Null is not the same as zero, blank or an empty string

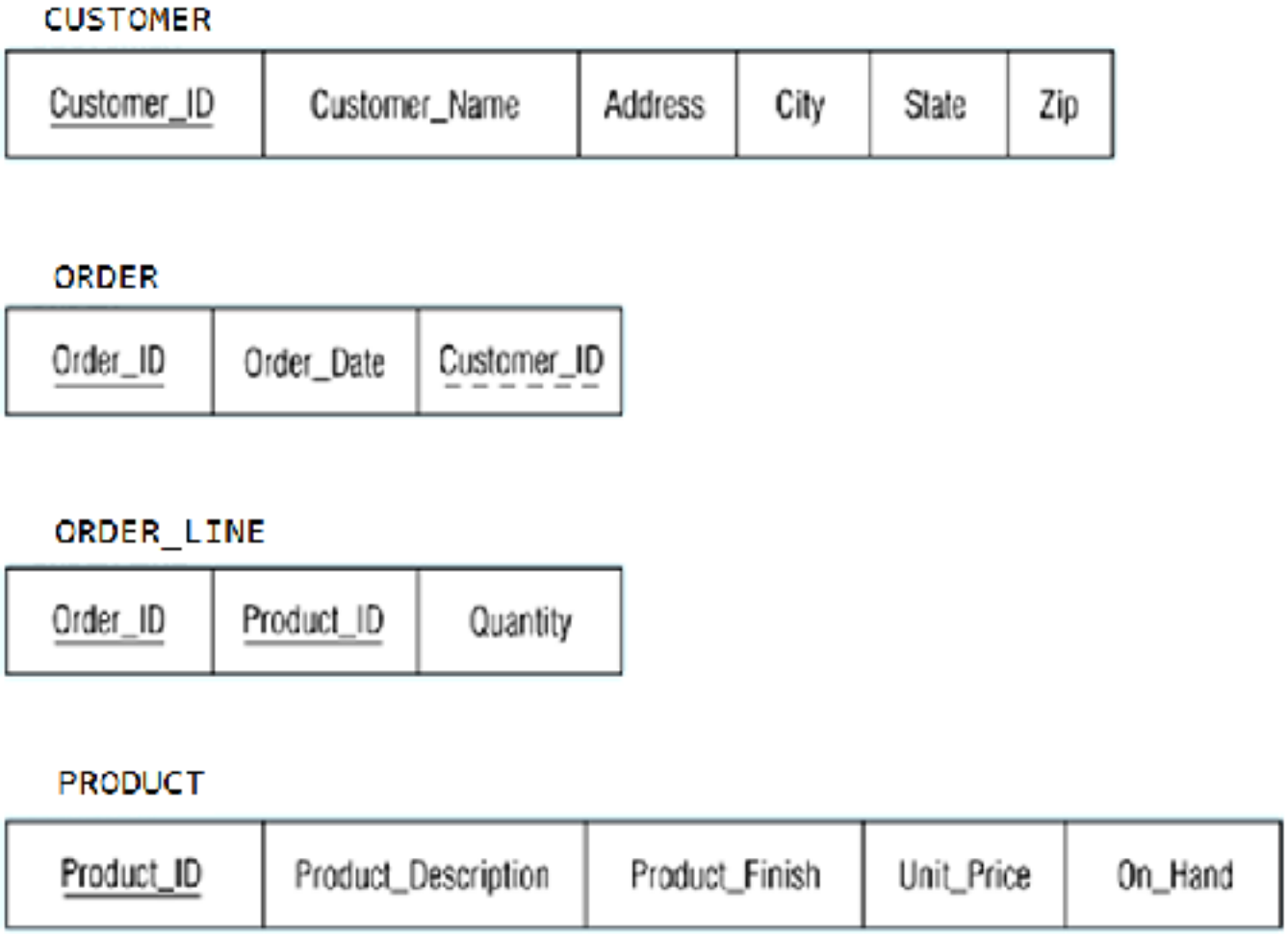
RDBMS concepts: tables, keys, relationships

- Relational Keys and Structures
  - Keys
    - Candidate Key is a set of one or more attributes that can uniquely identify a row in a given table.
    - Primary Key is an attribute of an entity which can uniquely identify a row in a given table
    - Composite Key is a primary key which is a combination of more than one attribute is called a composite primary key
    - Foreign Key is an attribute of an entity which refers to the domain of the primary key in the same entity or a different entity

Integrity Constraints

- Domain Integrity
  - Allowable values for an attribute. (not null,check)
- Entity Integrity
  - No primary key attribute may be null. (unique, not null)
- Referential Integrity
  - For example: Delete Rules
    - Restrict, Cascade, Set-to-Null
    - To be discussed more in SQL

Schema for Relations



Conceptual Data Modeling

The Entity-Relationship Model

E-R Model Constituents

- Entity
  - Person, Place, Object, Event, Concept…
- Attributes
  - Name, Age, City, Salary…
- Relationships
  - Student *registers* for a Course
  - Employee *applies* for a Leave

Simple Vs composite attribute

- Simple Attribute:
  - Can not be divided into simpler components
    - For example, *age* of a Person
- Composite Attribute:
  - Can be split into components
    - *Address* of a Person can be split into street, location, city, state, zip
    - *Date of birth* of an Employee can be split into day, month, and year

Single Vs Multi-values Attributes

- Single Valued:
  - Can take only a single value for each entity instance
    - For example, Salary of an employee (only one value)
- Multi-valued:
  - Can take many values
    - For example, Skill-Set of an employee can be
      “Java, Servlet, JSP” or
      “C#, ASP.Net, WCF, WPF”

Stored Vs Derived Attributes

- Stored Attribute:
  - Attribute that need to be stored permanently
    - For example, *name* of an employee
- Derived Attribute:
  - Attribute that can be calculated based on other attributes
    - For example, *years-of-service* of an employee can be calculated from the date of hire and the current date

Regular Vs Week Entity

- Regular Entity:
  - Entity that has its own key attribute
    - For example, Employee, Student, Customer etc.
- Weak Entity:
  - Entity that depends on other entity for its existence and does not have key attribute of its own
    - For example, *Nominee* of an Insurance policy holder.

Relationships

- A relationship type between two entity types defines the set of all associations between these entity types
- Each instance of the relationship between the members of these entity types is called a relationship instance

Degree of a Relationship

- Degree: The number of entity types involved
  - Unary One
  - Binary Two
  - Ternary Three
  - For example,
    - *Employee* and *Manager-of-Employee* is Unary
    - *Employee* works for *Department* is Binary
    - *Customer* purchases *Item* from a *Shop* is Ternary

Cardinality

- Relationships can have different connectivity
  - One-To-One (1:1)
  - One-To-Many (1:N)
  - Many-To-One (N:1)
  - Many-To-Many (N:M)

Cardinality

- One-To-One
  - One employee is given only one laptop and one laptop has only one owner
  - One department has only one head-of-department and each HOD heads only one department

Cardinality

- One-To-Many
  - One department has many employees
- Many-To-One
  - Many employees belong to one department

Cardinality

- Many-To-Many
  - One project has many employees, and one employee has worked in many projects
  - One customer has purchased many products, and one product is purchased by many customers

Relationship Participation

- Total: Every entity instance must be connected through the relationship to another instance of the other participating entity types
- Partial: All instance need not participate
  - For example,
    Employee and Head-Of-Department
    Employee – Partial
    Department - Total

All employees will not be head-of some department. So only few instances of employee entity participate in the above relationship. But each department will be headed by some employee. So department entity's participation is total and employee entity's participation is partial in the above relationship

An Entity is an object or concept about which business user wants to store information.

A weak Entity is dependent on another Entity to exist. Example Order Item depends upon Order Number for its existence. Without Order Number it is impossible to identify Order Item uniquely.

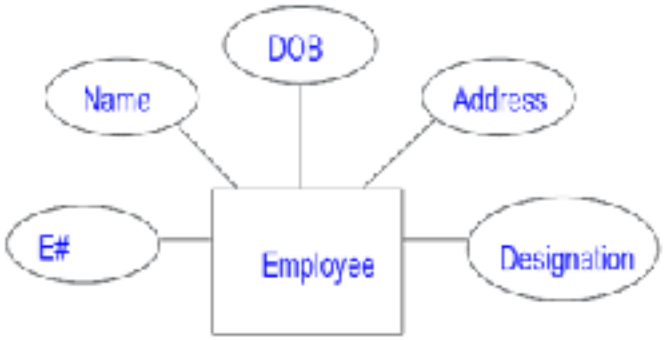
Attributes are the properties or characteristics of an Entity

A key attribute is the unique, distinguishing characteristic of the Entity

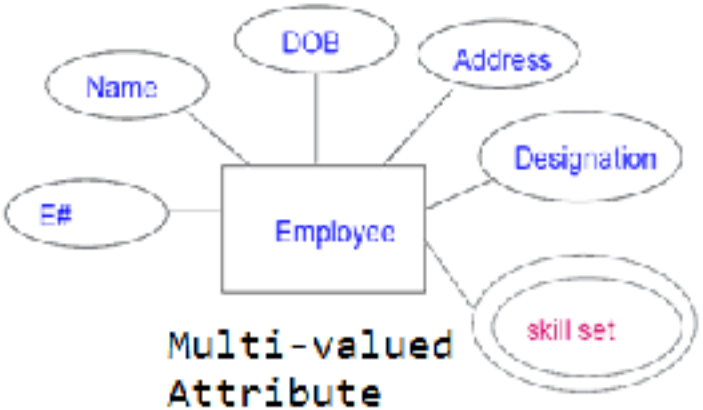
A multivalued attribute can have more than one value. For example, an employee Entity can have multiple skill values.

A derived attribute is based on another attribute. For example, an employee's monthly salary is based on the employee's basic salary and House rent allowance.

Relationships illustrate how two entities share information in the database structure.



Attributes



Multi-valued Attribute

Key Attribute

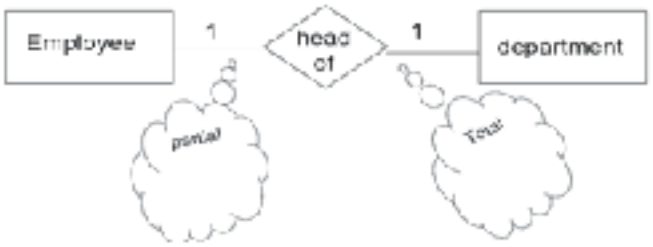
Composite Attribute

Relationship

Unary Relationship

Binary Relationship

Ternary Relationship



Relationship Participation

Weak Entity

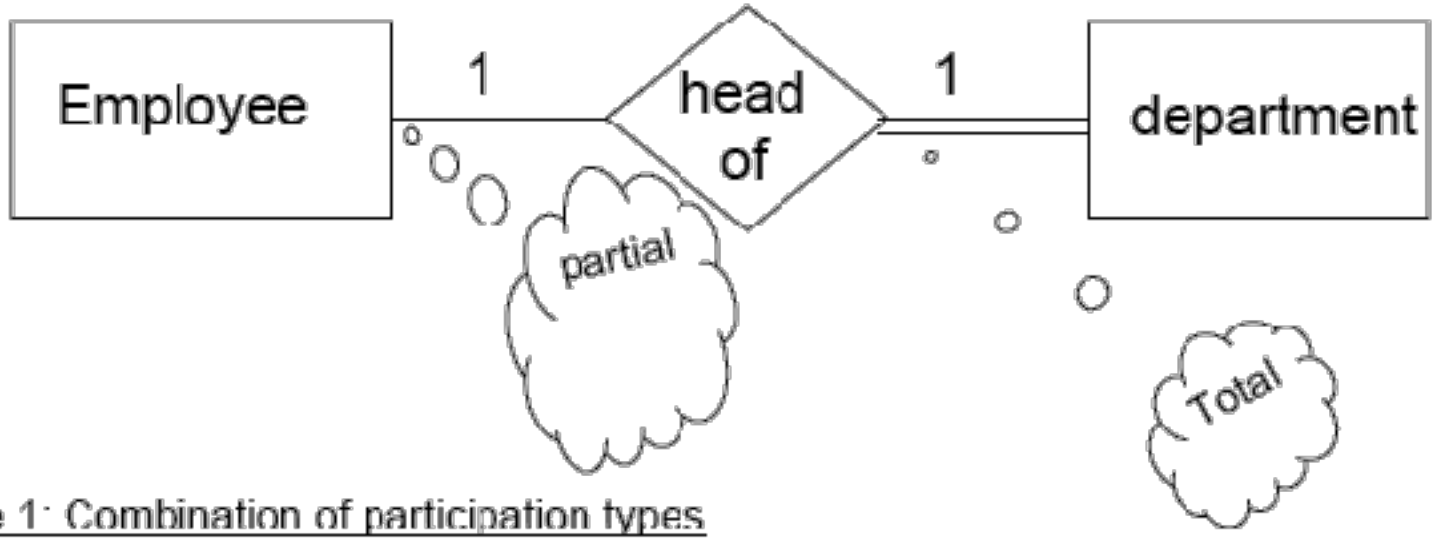
Attributes Of Relationship

Transforming E-R Diagrams into Relations

- Map Regular Entities to Relations
- Composite attributes: Use only their simple, component attributes
- Multi-valued Attribute - Becomes a separate relation with a foreign key taken from the superior entity
- Map Weak Entities
  - Becomes a separate relation with a foreign key taken from the superior entity

Map Binary Relationships

Binary 1:1



- Case 1: Combination of participation types

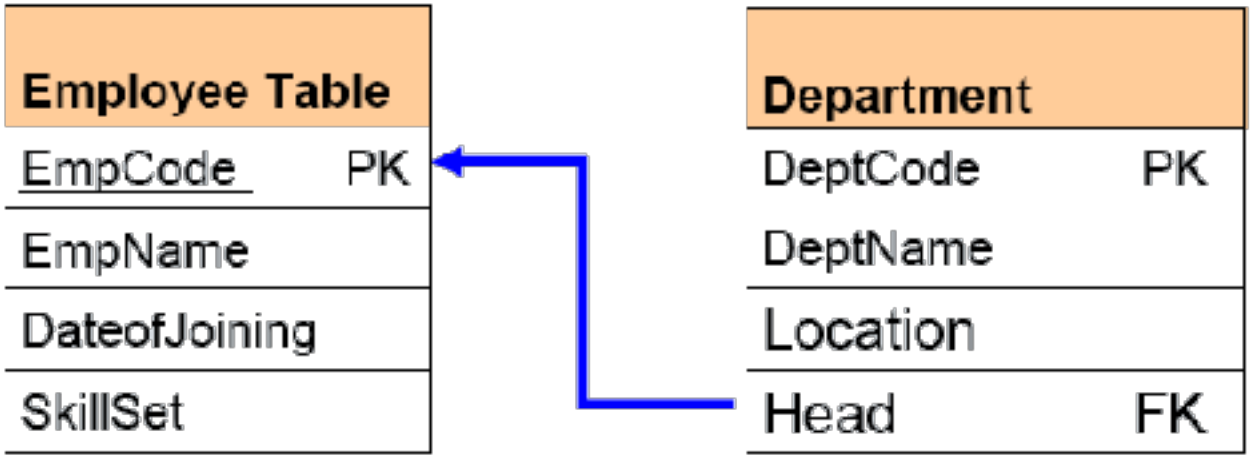
  The primary key of the partial participant will become the foreign key of the total participant

  Employee(E#, Name,…)

  Department (Dept#, Name…,Head)

Binary 1:1

Binary 1 : 1



Map Binary Relationships

Binary 1:N

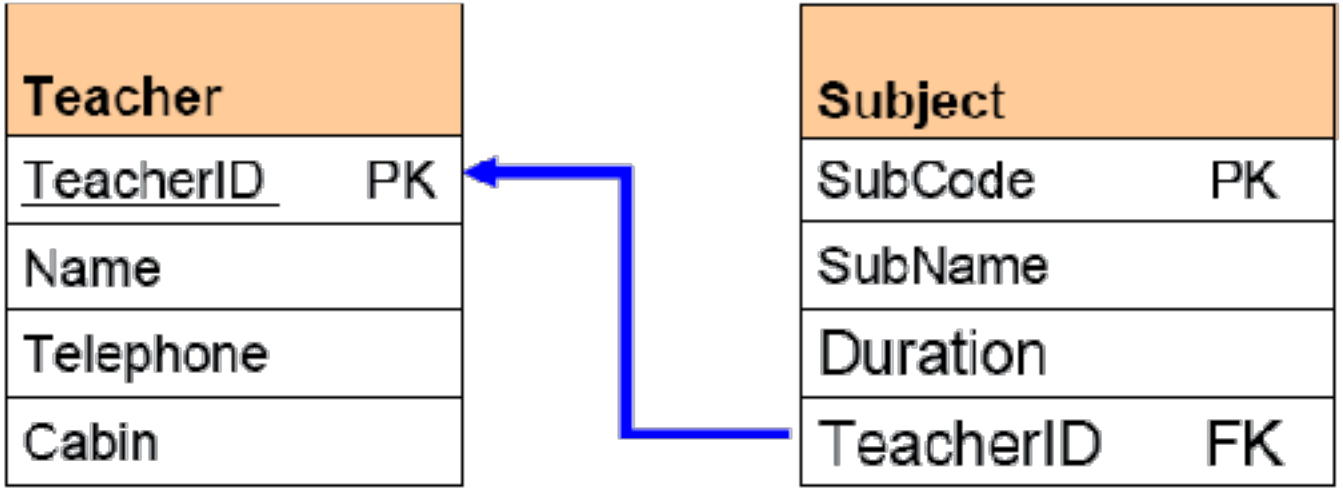
The primary key of the relation on the "1" side of the relationship becomes a foreign key in the relation on the "N" side

Teacher (ID, Name, Telephone, ...)

Subject (Code, Name, ..., Teacher)

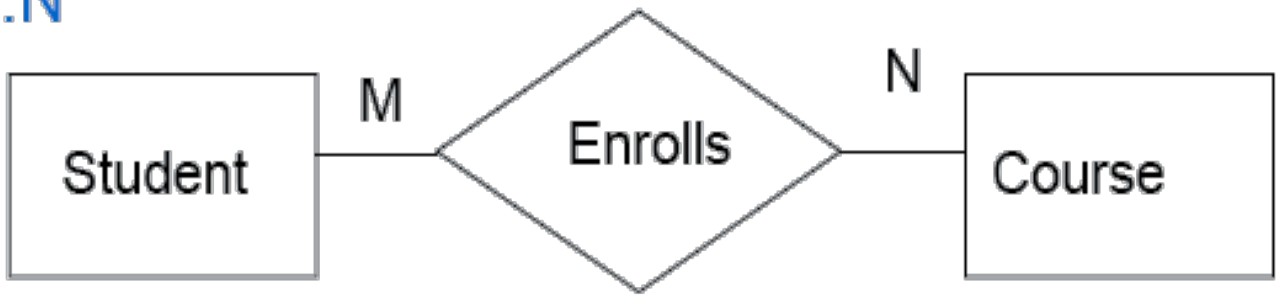
Binary 1:N

Binary 1 : N



Binary M:N

Binary M:N



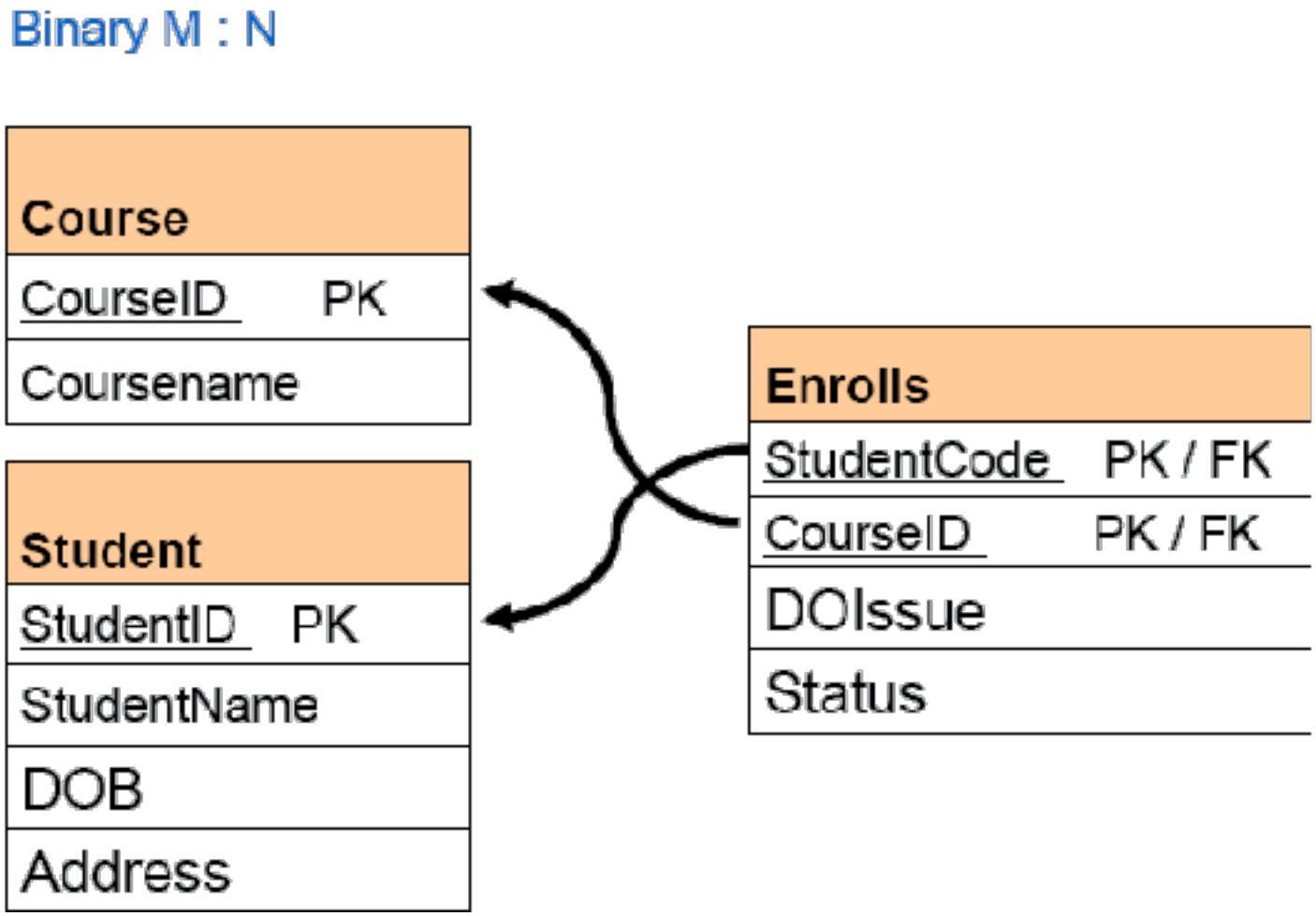
- A new table is created to represent the relationship
- Contains two foreign keys - one from each of the participants in the relationship
- The primary key of the new table is the combination of the two foreign keys

Student (Sid#, Title…)

Course(C#, CName,…)

Enrolls (Sid#, C#)

Binary M:N


Binary M : N
Course
CourseID PK
Coursename
Student
StudentID PK
StudentName
DOB
Address
Enrolls
StudentCode PK / FK
CourseID PK / FK
DOIssue
Status

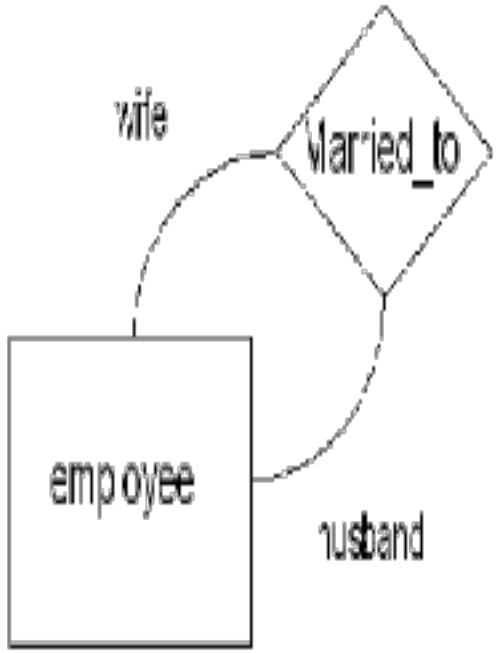
Unary

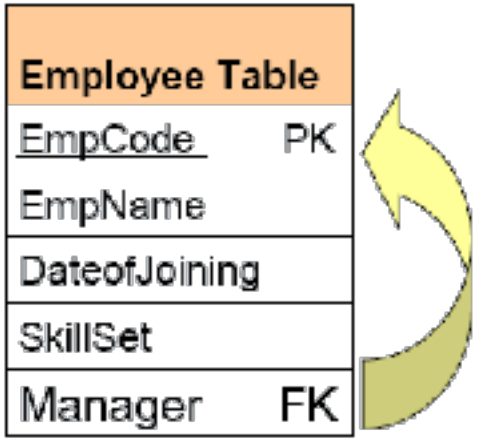
Unary 1:1

- Consider employees who are also a couple
- The primary key field itself will become foreign key in the same table

Employee(E#, Name,... **Spouse**)

Unary 1 : N

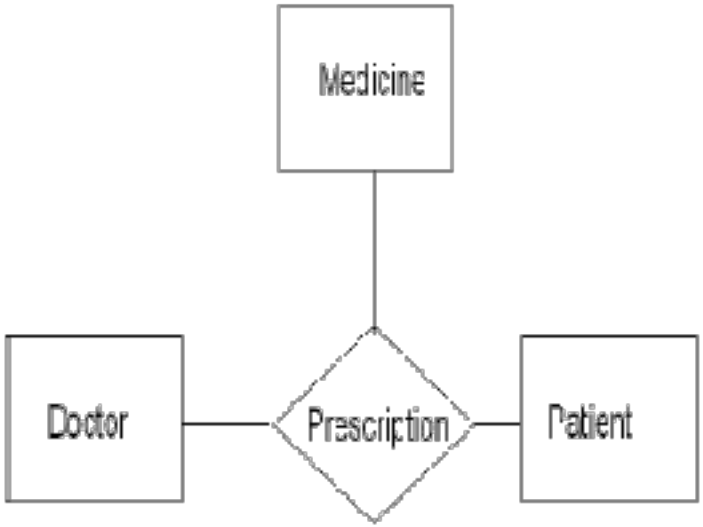




(Self Learning)

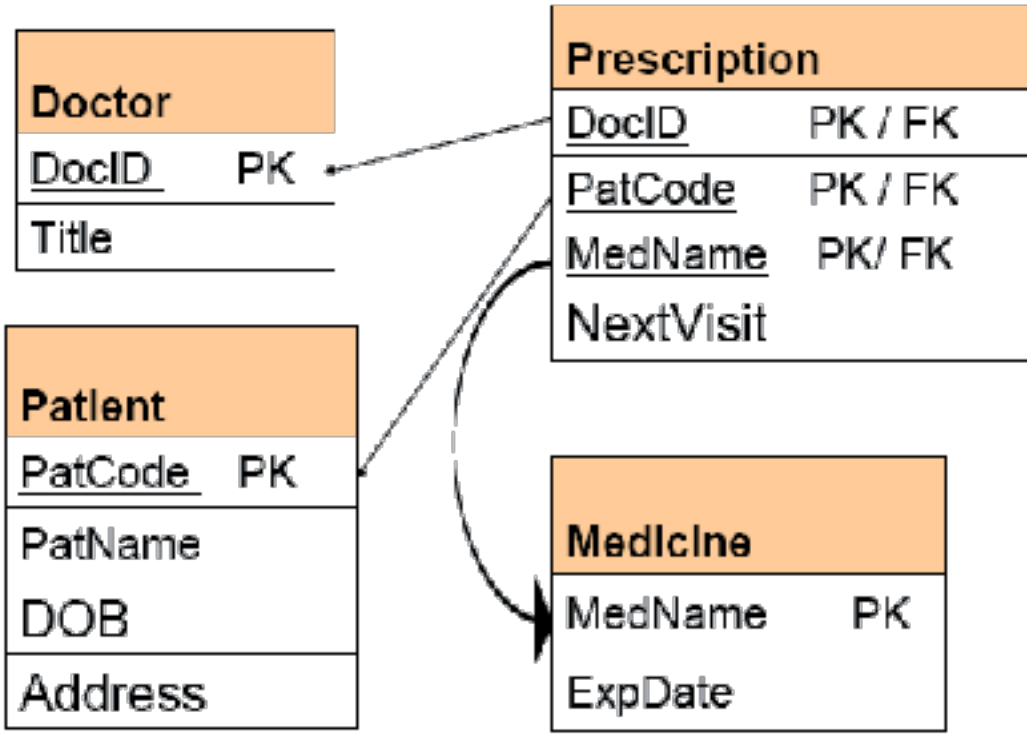
Ternary relationship

- Represented by a new table
- The new table contains three foreign keys - one from each of the participating Entities
- The primary key of the new table is the combination of all three foreign keys
- Prescription (Doctor#, Patient #, Medicine_Name)



(Self Learning)

Ternary



Redundancy and normalization

- The Evils of Redundancy
  - Redundancy is at the root of several problems associated with relational schemas:
    - redundant storage, insert/delete/update anomalies
  - Integrity constraints, in particular functional dependencies, can be used to identify schemas with such problems and to suggest refinements.
  - Main refinement technique: decomposition
    - replacing ABCD with, say, AB and BCD, or ACD and ABD.
  - Decomposition should be used judiciously:
    - Is there reason to decompose a relation?
    - What problems (if any) does the decomposition cause?

Redundancy and normalization

What is Normalization?

- Database designed based on the E-R model may have some amount of
  - Inconsistency
  - Uncertainty
  - Redundancy

To eliminate these draw backs some **refinement** has to be done on the database.

- **Refinement** process is called **Normalization**
- Defined as a step-by-step process of decomposing a complex relation into a simple and stable data structure.
- The formal process that can be followed to achieve a good database design
- Also used to check that an existing design is of good quality
- The different stages of normalization are known as "normal forms"
- To accomplish normalization we need to understand the concept of Functional Dependencies.

Redundancy and normalization

Student_Course_Result Table

Student_Details			Course_Details				Result_Details		
101	Davis	11/4/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	82	A
102	Daniel	11/6/1987	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	62	C
101	Davis	11/4/1986	H6	American History		4	11/22/2004	79	B
103	Sandra	10/2/1988	C3	Bio Chemistry	Basic Chemistry	11	11/16/2004	65	B
104	Evelyn	2/22/1986	B3	Botany		8	11/26/2004	77	B
102	Daniel	11/6/1987	P3	Nuclear Physics	Basic Physics	13	11/12/2004	68	B
105	Susan	8/31/1985	P3	Nuclear Physics	Basic Physics	13	11/12/2004	89	A
103	Sandra	10/2/1988	B4	Zoology		5	11/27/2004	54	D
105	Susan	8/31/1985	H6	American History		4	11/22/2004	87	A
104	Evelyn	2/22/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	65	B

Functional Dependencies and Keys

- In a given relation R, X and Y are attributes. Attribute Y is **functionally dependent** on attribute X if each value of X determines **EXACTLY ONE** value of Y, which is represented as X -> Y (X can be composite in nature).

- We say here "x determines y" *or* "y is functionally dependent on x"
  X→Y does not imply Y→X

- If the value of an attribute "Marks" is known then the value of an attribute "Grade" is determined since Marks→ Grade

- Types of functional dependencies:

  – Full Functional dependency
  – Partial Functional dependency
  – Transitive dependency

(Self Learning)

Consider the following Relation

REPORT (STUDENT#,COURSE#, CourseName, IName, Room#, Marks, Grade)

- **STUDENT#** - Student Number
- **COURSE#** - Course Number
- **CourseName** - Course Name
- **IName** - Name of the Instructor who delivered the course
- **Room#** - Room number which is assigned to respective Instructor
- **Marks** - Scored in Course COURSE# by Student STUDENT#
- **Grade** - obtained by Student STUDENT# in Course COURSE#

(Self Learning)

Functional Dependencies- From the previous example

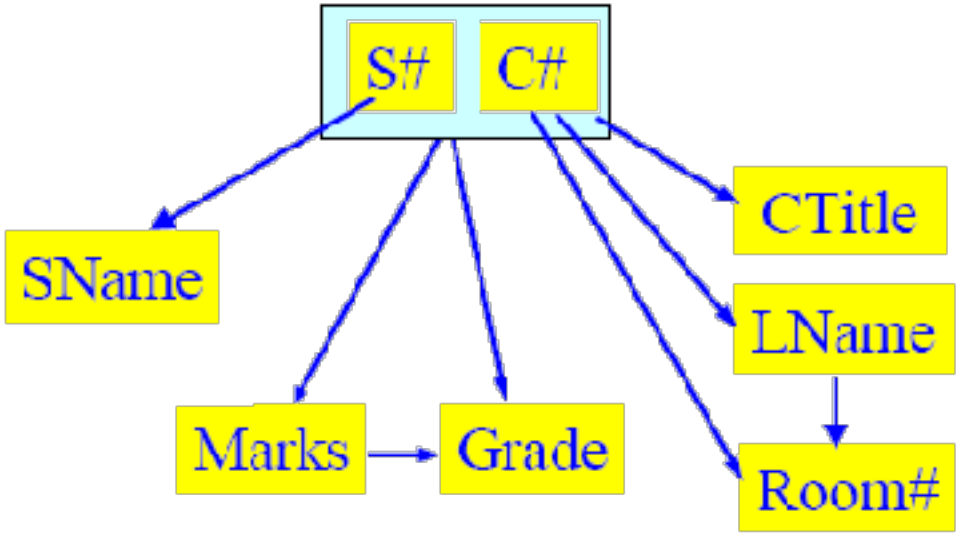
- **STUDENT# COURSE# → Marks**
- **COURSE# → CourseName,**
- **COURSE# → IName (Assuming one course is taught by one and only one Instructor)**
- **IName → Room# (Assuming each Instructor has his/her own and non-shared room)**
- **Marks → Grade**

(Self Learning)

Dependency diagram

Report(S#,C#,SName,CTitle,LName,Room#,Marks,Grade)

- **S# → SName**
- **C# → CTitle,**
- **C# → LName**
- **LName → Room#**
- **C# → Room#**
- **S# C# → Marks**
- **Marks → Grade**
- **S# C# → Grade**



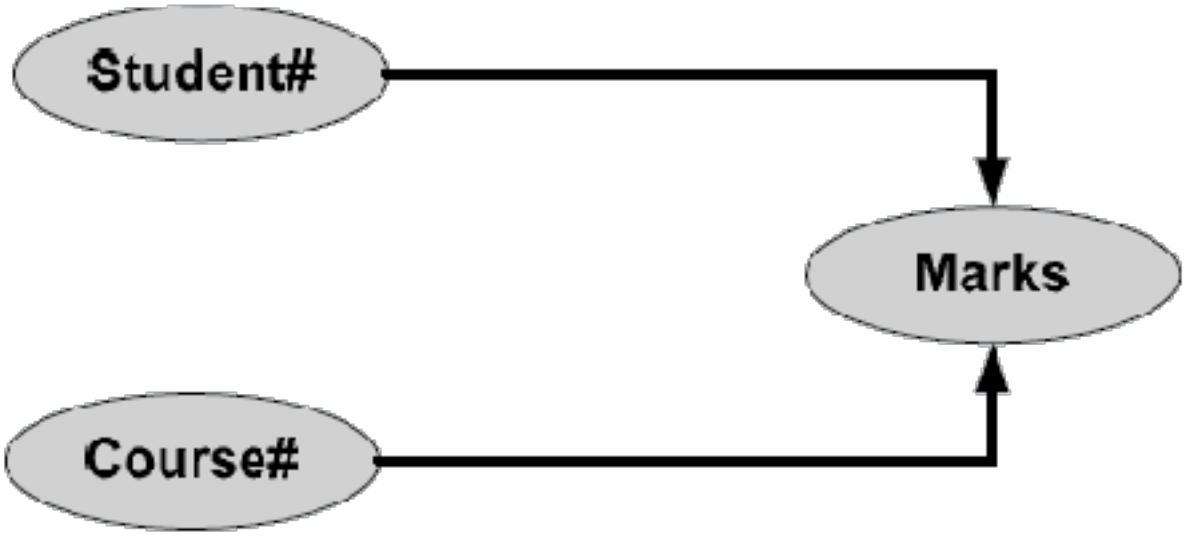
Assumptions:

- Each course has only one lecturer and each lecturer has a room.
- Grade is determined from Marks.

(Self Learning)

Full dependencies

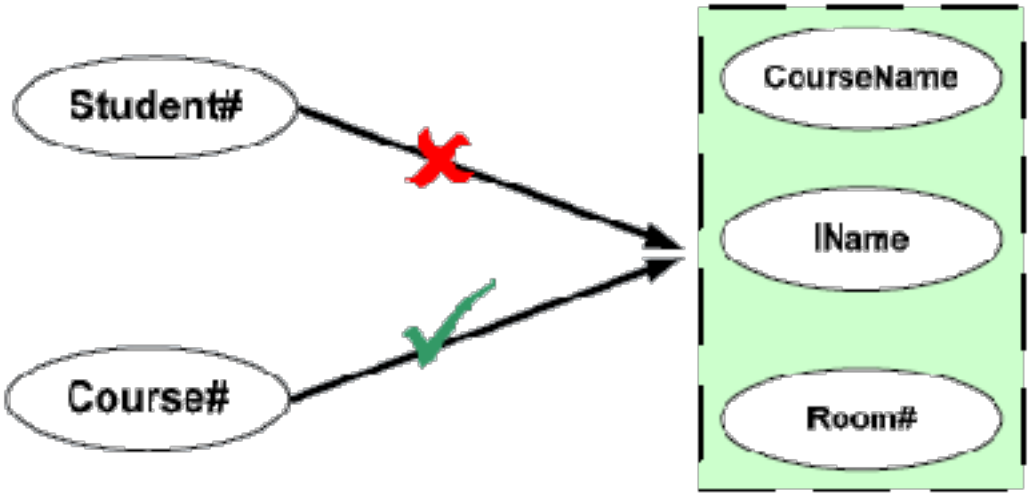
X and Y are attributes.
X Functionally determines Y
Note: Subset of X should not functionally determine Y



(Self Learning)

Partial dependencies

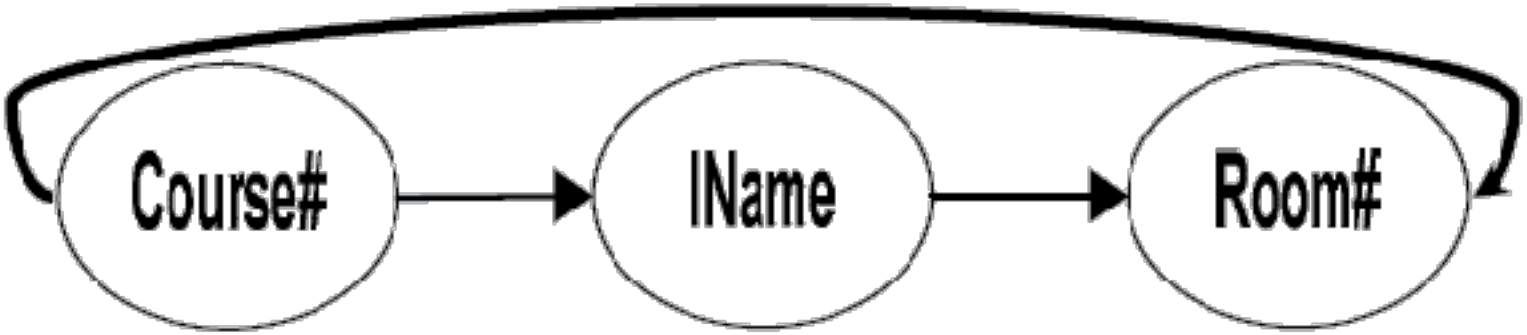
X and Y are attributes.
Attribute Y is partially dependent on the attribute X only if it is dependent on a sub-set of attribute X.



(Self Learning)

Transitive dependencies

X Y and Z are three attributes.
X -> Y
Y-> Z
=> X -> Z



Normal Forms - 1st Normal Form (1NF)

- A table (relation) is in 1NF if
  - There are no duplicated rows in the table.
  - Each cell is single-valued (i.e.,there are no repeating groups or arrays).
  - Entries in a column (attribute, field) are of the same kind.

Recall

Student_Course_Result Table

Student_Details			Course_Details				Results		
101	Davis	11/4/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	82	A
102	Daniel	11/6/1987	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	62	C
101	Davis	11/4/1986	H6	American History		4	11/22/2004	70	B
103	Sandra	10/2/1988	C3	Bio Chemistry	Basic Chemistry	11	11/15/2004	65	B
104	Evelyn	2/22/1986	R3	Botany		8	11/26/2004	77	B
102	Daniel	11/6/1987	P3	Nuclear Physics	Basic Physics	13	11/12/2004	63	B
105	Susan	8/31/1985	P3	Nuclear Physics	Basic Physics	13	11/12/2004	89	A
103	Sandra	10/2/1988	R4	Zoology		5	11/27/2004	54	D
105	Susan	8/31/1985	H6	American History		4	11/22/2004	87	A
104	Evelyn	2/22/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	65	B

Normal Forms – 2nd Normal Form (2NF)

- ***A Relation is said to be in Second Normal Form if and only if :***
  - ***It is In the First normal form, and***
  - ***No partial dependency exists between non-key attributes and key attributes.***

- An attribute of a relation R that belongs to any key of R is said to be a prime attribute and that which doesn't is a **non-prime attribute**

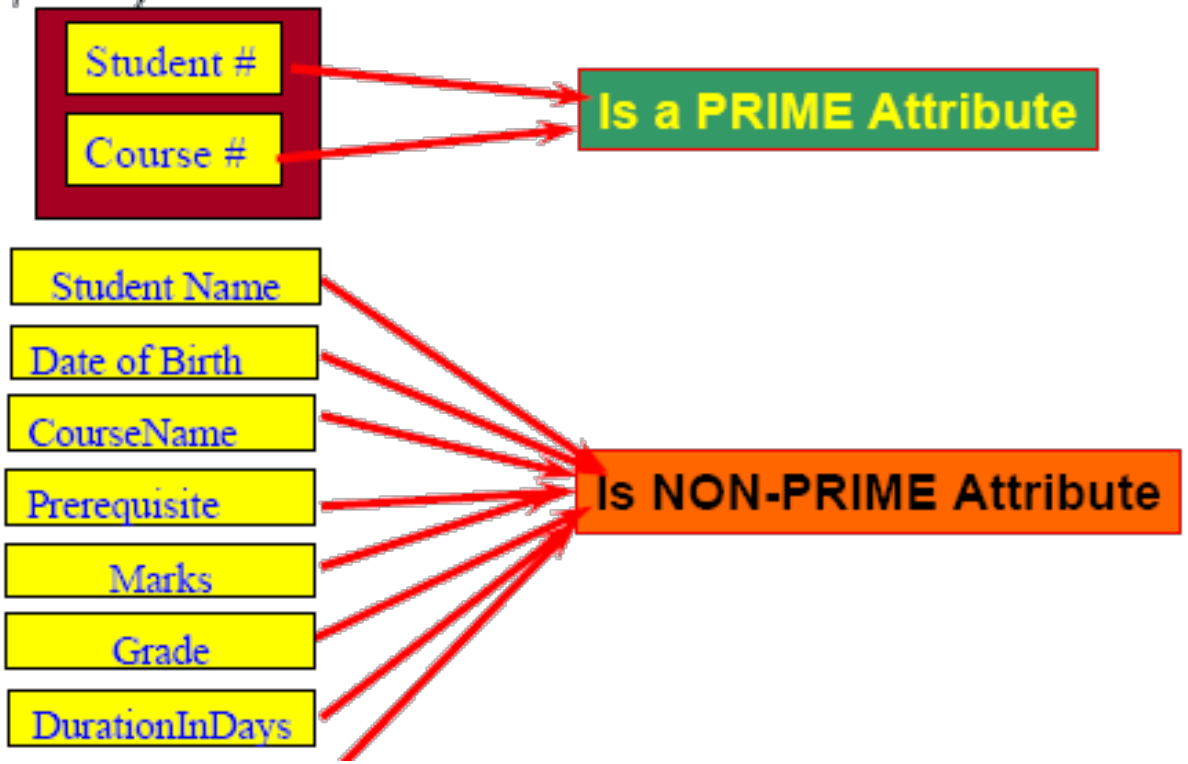
To make a table 2NF compliant, we have to remove all the partial dependencies

Note : - All partial dependencies are eliminated

Prime Vs Non-prime attributes

- An attribute of a relation R that belongs to any key of R is said to be a **prime** attribute and that which doesn't is a **non-prime attribute**

Report(**S#,C#**,StudentName,DateOfBirth,CourseName,PreRequisite,DurationInDays,DateOfExam,Marks,Grade)



Normal Forms – 2nd Normal Form (2NF)

- STUDENT# is key attribute for Student,
- COURSE# is key attribute for Course
- STUDENT# COURSE# together form the composite key attributes for Results relationship.
- Other attributes like StudentName (Student Name), DateofBirth, CourseName, PreRequisite, DurationInDays, DateofExam, Marks and Grade are non-key attributes.

To make this table 2NF compliant, we have to remove all the partial dependencies.

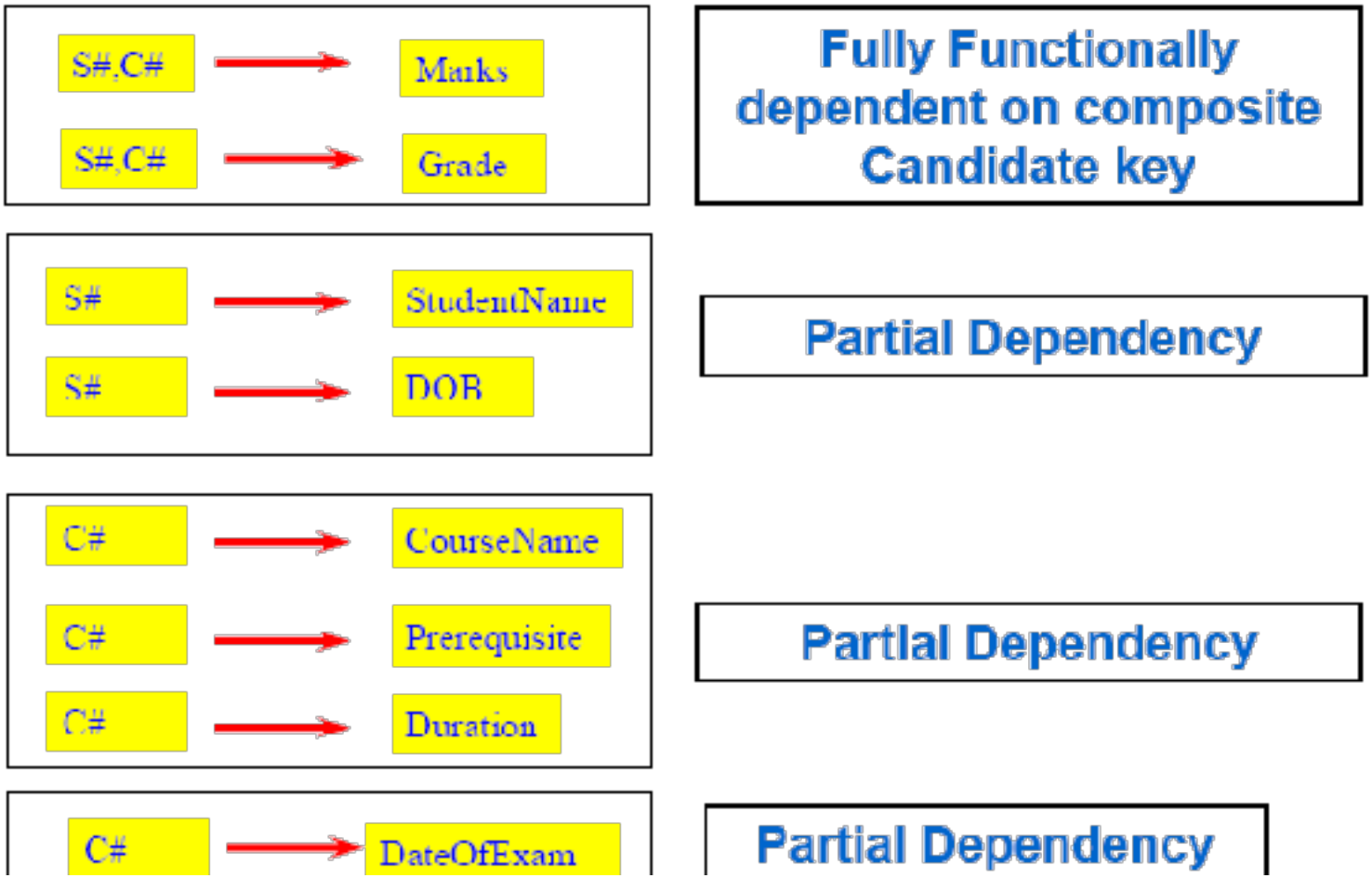
Student #, Course# -> Marks, Grade

Student# -> StudentName, DOB,

Course# -> CourseName, Prerequiste, DurationInDays

Course# -> Date of Exam

Normal Forms – 2nd Normal Form (2NF)



Normal Forms – 2nd Normal Form (2NF)

STUDENT TABLE

Student#	StudentName	DateofBirth
101	Davis	04-Nov-1986
102	Daniel	06-Nov-1987
103	Sandra	02-Oct-1988
104	Evelyn	22-Feb-1986
105	Susan	31-Aug-1985
106	Mike	04-Feb-1987
107	Juliet	09-Nov-1986
108	Tom	07-Oct-1986

COURSE TABLE

Course#	Course Name	Pre Requisite	Duration InDays
M1	Basic Mathematics		11
M4	Applied Mathematics	M1	7
H6	American History		4
C1	Basic Chemistry		5
C3	Bio Chemistry	C1	11
B3	Botany		8
P1	Basic Physics		8

Normal Forms – 2nd Normal Form (2NF)

Student-Marks

Student#	Course#	Marks	Grade
101	M4	82	A
102	M4	62	C
101	H6	79	B
103	C3	65	B
104	B3	77	B
102	P3	68	B
105	P3	89	A
103	B4	54	D
105	H6	87	A
104	M4	65	B

Exam-Date

Course#	DateOfExam
M4	11-Nov-04
H6	22-Nov-04
C3	16-Nov-04
B3	26-Nov-04
P3	12-Nov-04
B4	27-Nov-04

Normal Forms – 3rd Normal Form (3NF)

A relation R is said to be in the Third Normal Form (3NF) if and only if

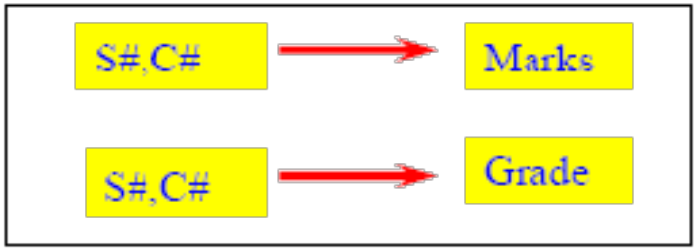
— ***It is in 2NF and***

— ***No transitive dependency exists between non-key attributes and key attributes.***

• STUDENT# and COURSE# are the key attributes.

• All other attributes, except grade are non-partially, non-transitively

dependent on key attributes.



- **Student#, Course# - >** Marks
- **Marks ->** Grade

Note : - All transitive dependencies are eliminated

Normal Forms – 3rd Normal Form (3NF)

3NF Tables

Student#	Course#	Marks
101	M4	82
102	M4	62
101	H6	79
103	C3	65
104	B3	77
102	P3	68
105	P3	89
103	B4	54
105	H6	87
104	M4	65

Third Normal Form – Tables in 3 NF

MARKSGRADE TABLE

UpperBound	LowerBound	Grade
100	95	A+
94	85	A
84	70	B
69	65	B-
64	55	C
54	45	D

MySQL - Session 1

Objectives

- After completing this lesson, you should be able to:
  - Describe the main database objects
  - Create tables
  - Describe the data types that can be used when specifying column definition
  - Alter table definitions
  - Drop, rename, and truncate tables

Database Objects in MySQL

Object	Description
Table	Basic unit of storage; composed of rows and columns
View	Logically represents subsets of data from one or more tables
Index	Improves the performance of some queries

Naming Rules For Identifiers

- Identifiers are names of table, view, index, coulumn, constraint etc.
- An identifier may be quoted or unquoted
  - Permitted characters in unquoted identifiers:
    - ASCII: [0-9, a-z, A-Z $ _] (basic Latin letters, digits 0-9, dollar, underscore)
    - Extended: U+0080 .. U+FFFF
  - Identifiers may begin with a digit but unless quoted may not consist solely of digits.
  - Database, table, and column names cannot end with space characters.
  - Database and table names cannot contain “/”, “\”, “.”, or characters that are not permitted in file names.
- The identifier quote character is the backtick (“`”)

Table creation

```
CREATE TABLE [IF NOT EXISTS] tbl_name(
create_definition,
...
);
```

create_definition may be one or more of the following:

- col_name *column_definition*
- [CONSTRAINT [*constraint_name*]] PRIMARY KEY (col_name)
- [CONSTRAINT [*constraint_name*]] UNIQUE (col_name)
- [CONSTRAINT [*constraint_name*]] FOREIGN KEY (col_name) *reference_definition*
- CHECK (expr)

Note: the CHECK constraint is parsed, but not implemented by the MySQL engines.

For a complete syntax/features refer:

http://dev.mysql.com/doc/refman/5.1/en/create-table.html

Table creation

To define a column in a table, use the following syntax:

col_name *column_definition*

where

column_definition:

```
    data_type
      [NOT NULL | NULL]
      [DEFAULT default_value]
      [AUTO_INCREMENT]
      [UNIQUE | PRIMARY KEY]
      [reference_definition]
```

reference_definition:

```
    REFERENCES tbl_name (index_col_name, ...)
      [ON DELETE reference_option]
      [ON UPDATE reference_option]
```

reference_option:

```
    RESTRICT | CASCADE | SET NULL | NO ACTION
```

Examples of column definition

- age int
- emp_id int primary key
- emp_id int primary key auto_increment
- city varchar(20) not null default ‘Bangalore’
- email_id varchar(50) not null unique

- dept_id int references tbl_departments (dept_id)
- dept_id int references tbl_departments(dept_id) on delete cascade
- age int check (age>0)

Note: The last 3 statements will not give any syntax errors, but do not work as expected.

Datatypes

- Datatypes in MySQL are broadly divided into the following categories:
  - Numeric (integers and real)
  - Date and time
  - String

For more details about the datatypes in MySQL, please visit

http://dev.mysql.com/doc/refman/5.1/en/data-types.html

Datatypes – Numeric (integers)

Type	Storage (Bytes)	Minimum Value (Signed/ Unsigned)	Maximum Value Signed/ Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

Datatypes – Numeric (real)

- Floating Point
  - FLOAT (uses 4 bytes to store the data)
  - DOUBLE (uses 8 bytes to store the data)
- Fixed Point
  - DECIMAL (precision, scale)

Datatypes – Date and time

- DATE
  - Used for values with a date part but no time part.
  - MySQL retrieves and displays DATE values in'YYYY-MM-DD' format.
  - The supported range is '1000-01-01' to '9999-12-31'.
- DATETIME
  - Used for values that contain both date and time parts.
  - MySQL retrieves and displaysDATETIME values in 'YYYY-MM-DD HH:MM:SS' format.
  - The supported range is '1000-01-01 00:00:00' to'9999-12-31 23:59:59'
- TIMESTAMP
  - Used for values that contain both date and time parts.
  - The supported range is **'1970-01-01 00:00:01' to '2038-01-19 03:14:07'**

Datatypes – Date and time

- TIME
  - MySQL retrieves and displays TIME values in 'HH:MM:SS' format
  - May range from '-838:59:59' to '838:59:59‘
  - The hours part may also represent elapsed time or a time interval between two events

Datatypes - String

- CHAR
- VARCHAR
- BLOB
- TEXT
- ENUM

Datatypes – String (CHAR)

- CHAR
  - Length is fixed
  - Length can range from 0 to 255 characters
  - When CHAR values are stored, they are right-padded with spaces to the specified length.
  - When CHAR values are retrieved, trailing spaces are removed unless the PAD_CHAR_TO_FULL_LENGTH SQL mode is enabled.
    - SET SQL_MODE='PAD_CHAR_TO_FULL_LENGTH'

Datatypes – String (VARCHAR)

- VARCHAR
  - Values in VARCHAR columns are variable-length strings.
  - The length can be specified as a value from 0 to 65535.

Datatypes – String (BLOB, TEXT)

- BLOB (TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB)
  - Binary Large OBject that can hold a variable amount of data
  - Ususally used for storing non-textual file content such as images, audio or video
  - BLOB values are treated as binary strings (byte strings).
  - They have no character set, and sorting and comparison are based on the numeric values of the bytes in column values.
  - Maximum size of a BLOB field is 64 KB.
    - If you need to store more than the above mentioned size, then use MEDIUMBLOB (16 MB) or LONGBLOB (4GB)
- TEXT (TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT)
  - Same as BLOB, except that it can hold text data (non-binary string)
  - Sizes are similar to their BLOB counterparts

Datatypes – String (ENUM)

- An ENUM is a string object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification at table creation time.

```
CREATE TABLE QUESTIONS (
ID INT PRIMARY KEY,
QUESTION TEXT,
DIFFICULTY_LEVEL ENUM ('VERY EASY', 'EASY',
'MODERATE', 'DIFFICULT', 'VERY DIFFICULT')
);
```

- By default, if an invalid value is inserted for this column, MySQL will insert a NULL
  - This can be controlled by setting the SQL_MODE to ‘TRADITIONAL’, which results in an error, if an invalid value is inserted

Datatypes – String (ENUM)

```
mysql> SET SQL_MODE='';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO QUESTIONS VALUES (1, 'WHAT IS THE CAPITAL CITY OF INDIA', 'SO EASY');
Query OK, 1 row affected, 1 warning (0.04 sec)

mysql> SELECT * FROM QUESTIONS;
+----+-----------------------------------+-----------------+
| ID | QUESTION                          | DIFFICULTY_LEVEL |
+----+-----------------------------------+-----------------+
|  1 | WHAT IS THE CAPITAL CITY OF INDIA |                 |
+----+-----------------------------------+-----------------+
1 row in set (0.00 sec)

mysql> SET SQL_MODE='TRADITIONAL';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO QUESTIONS VALUES (2, 'WHAT IS THE CAPITAL CITY OF KARNATAKA', 'SO EASY');
ERROR 1265 (01000): Data truncated for column 'DIFFICULTY_LEVEL' at row 1
mysql> SELECT * FROM QUESTIONS;
+----+-----------------------------------+-----------------+
| ID | QUESTION                          | DIFFICULTY_LEVEL |
+----+-----------------------------------+-----------------+
|  1 | WHAT IS THE CAPITAL CITY OF INDIA |                 |
+----+-----------------------------------+-----------------+
1 row in set (0.00 sec)

mysql>
```

With SQL_MODE=''
The record was inserted with NULL value for DIFFICULTY_LEVEL

With SQL_MODE='TRADITIONAL'
The record was not inserted.

Table - examples

```
CREATE TABLE BOOKS(
        BOOK_ID INT AUTO_INCREMENT PRIMARY KEY,
        TITLE VARCHAR(30) NOT NULL,
        PRICE DOUBLE
);
```

Table - examples

```
CREATE TABLE DEPTS (
        DEPT_ID INT,
        DNAME VARCHAR(20) NOT NULL,
        PRIMARY KEY (DEPT_ID)
);

CREATE TABLE EMPS (
        EMP_ID INT PRIMARY KEY,
        ENAME VARCHAR(20),
        DEPT_NO INT,
        SALARY DECIMAL(10, 2),
        FOREIGN KEY (DEPT_NO) REFERENCES DEPTS(DEPT_ID)
                ON DELETE SET NULL
);
```

Constraints

- The MySQL Server uses constraints to prevent invalid data entry into tables.
- You can use constraints to do the following:
  - Enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table.
  - Prevent the deletion of a table if there are dependencies from other tables
- Constraints to a table can be added during the table creation itself, or later using the ALTER TABLE command
- Constraints can be given at table or column level

Constraints

- NOT NULL
  - Does not allow blanks for a given column
  - Note that ‘’ or “” is not same as NULL
- UNIQUE
  - Does not allow duplicates
  - MySQL allows more than one NULL entry to a column with UNIQUE constraint (unlike Oracle)
    - Since NULL is not a value, repetition of NULL values for a given column is not a violation of this constraint. To avoid this, use UNIQUE with NOT NULL constraint.

Constraints

- PRIMARY KEY
  - One or more columns together identify a row uniquely.
  - Only one PRIMARY KEY is allowed per table
  - If more than one column are involved, then the constraint must be specified as a table level constraint.
    - This kind of primary key is also known as COMPOSITE KEY
- FOREIGN KEY
  - Establishes a PARENT-CHILD relationship between two tables
  - The table being referenced is considered as a PARENT table
  - The table which is referencing (where the constraint is specified), is considered as a CHILD table

Constraints - examples

```
create table person (
id int,
name varchar(20) not null,
city varchar(20),
constraint PK_PERSON primary key (id));
```

Valid inserts:

```
insert into person values(1, 'SCOTT', 'DALLAS');
insert into person (id, name) values (2, 'MILLER');
insert into person values (3, 'ALLEN', NULL);
```

Invalid inserts:

```
insert into person values (2, 'SMITH', 'CHICAGO');
```

➔ duplicate value for primary key

```
insert into person (id, city) values (4, 'BANGALORE');
```

➔ name can not contain null

```
insert into person values (4, NULL, 'MUMBAI');
```

➔ name can not contain null

Constraints - examples

Column level constraint:

```
CREATE TABLE BOOKS(
        BOOK_ID INT PRIMARY KEY,
        TITLE VARCHAR(30) NOT NULL,
        PRICE DOUBLE
);
```

While specifying the column-level constraint, a constraint name can not be given.

Hence, it is a good practice to use the table level constraint specification.

Table level constraint:

```
CREATE TABLE BOOKS(
        BOOK_ID INT,
        TITLE VARCHAR(30) NOT NULL,
        PRICE DOUBLE,
        CONSTRAINT PK_BOOKS PRIMARY KEY (BOOK_ID)
);
```

Constraints - examples

Column level constraint:

```
CREATE TABLE EMPS (
        EMP_ID INT PRIMARY KEY,
        ENAME VARCHAR(20),
        DEPT_NO INT REFERENCES DEPTS(DEPT_ID),
        SALARY DECIMAL(10, 2)
);
```

Table level constraint:

```
CREATE TABLE EMPS (
        EMP_ID INT PRIMARY KEY,
        ENAME VARCHAR(20),
        DEPT_NO INT,
        SALARY DECIMAL(10, 2),
        CONSTRAINT FK_EMPS_DEPT
        FOREIGN KEY (DEPT_NO) REFERENCES DEPTS(DEPT_ID)
);
```

The **column-level foreign key** constraint specification is successfully parsed by MySQL, but may not enforce the constraint.

Hence, always use the table level constraint specification for **foreign keys**.

Constraints – Example on foreign keys

```
CREATE TABLE CATEGORIES (
ID INT PRIMARY KEY,
CATEGORY_NAME VARCHAR(20));

INSERT INTO CATEGORY VALUES(10, ‘COMPUTERS’);
INSERT INTO CATEGORY VALUES(20, ‘STATIONARIES’);

CREATE TABLE PRODUCTS (
ID INT PRIMARY KEY,
PRODUCT_NAME VARCHAR(20),
CATEGORY_ID INT,
CONSTRAINT FK_PRD_CAT
FOREIGN KEY (CATEGORY_ID) REFERENCES CATEGORIES(ID)
);

INSERT INTO PRODUCTS VALUES (1, ‘KEYBOARD’, 10); -- Works fine
INSERT INTO PRODUCTS VALUES (2, ‘PURSE’, 50); -- Fails, No Parent Key

DELETE FROM CATEGORIES WHERE ID = 20; -- Works fine
DELETE FROM CATEGORIES WHERE ID = 10; -- Fails, child records exist
```

Constraints – Foreign key UPDATE/DELETE options

- In a parent-child relationship using the foreign key constraint, when a record in the parent table is deleted,
  - it will be successful if there is no child record corresponding to the parent record being deleted.
  - it will fail if there is a child record corresponding to the parent record being deleted.
    - This can be controlled using the ON DELETE option of the constraint.

```
ALTER TABLE PRODUCTS DROP FOREIGN KEY FK_PRD_CAT;

ALTER TABLE PRODUCTS
        ADD CONSTRAINT FOREIGN KEY FK_PRD_CAT (CATEGORY_ID)
        REFERENCES CATEGORIES (ID)
        ON DELETE CASCADE;
```

Constraints – Foreign key DELETE/UPDATE options

- The different values for ON DELETE option are:
  - CASCADE
    - Dangerous, since this will delete all the corresponding child records.
  - SET NULL
    - Safe, allows the parent record to be deleted, while preserving the child records.
    - For this to work, the foreign key column should not have NOT NULL constraint on it.
  - RESTRICT
    - Default and should be prefered in most cases to ensure that the data is not accidentally deleted.
- The same values apply for ‘ON UPDATE’ option of the constraint.

Altering the table structure

- Alter table allows us to do the following:
  - Change the name of the column
  - Change the column definition
  - Add a new column
  - Delete an existing column
  - Add/delete primary key
  - Add/delete foreign key
  - Rename a table

Syntax: Alter table

- ALTER TABLE tbl_name
- [alter_specification [, alter_specification] ...]
- Where alter_specification can be:
- ADD [COLUMN] col_name column_definition
- ADD [CONSTRAINT [con_name]] PRIMARY KEY (col_name)
- ADD [CONSTRAINT [con_name]] UNIQUE (col_name)
- ADD [CONSTRAINT [con_name]] FOREIGN KEY (col_name) reference_definition
- ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
- CHANGE [COLUMN] old_col_name new_col_name column_definition
- MODIFY [COLUMN] col_name column_definition
- ..contd

Syntax: Alter table

- ..cont.

- DROP [COLUMN] col_name
- DROP PRIMARY KEY
- DROP FOREIGN KEY fk_symbol
- RENAME [TO] new_tbl_name

Alter table - examples

```
ALTER TABLE QUESTIONS
ADD COLUMN MARKS INT;

ALTER TABLE QUESTIONS
       CHANGE COLUMN MARKS POINTS INT;

ALTER TABLE QUESTIONS
       DROP PRIMARY KEY;

ALTER TABLE QUESTIONS
       ADD PRIMARY KEY(ID);

ALTER TABLE EMPLOYEES
       ALTER CITY SET DEFAULT ‘BANGALORE’

ALTER TABLE EMPLOYEES
       ALTER CITY DROP DEFAULT;
```

Data Manipulation Language

- A DML statement corresponds to
  - adding a new row to a table
  - modifying existing rows in a table
  - deleting existing rows from a table
- A transaction consists of a collection of DML statements that form a logical unit of work.

The INSERT statement - Syntax

```
INSERT [INTO] tbl_name [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
```

The above syntax allows us to insert one or more rows to a given table.

Examples:

```
INSERT CATEGORIES VALUES (10, 'COMPUTERS');
INSERT INTO CATEGORIES VALUE (20, 'STATIONARIES');

-- Insert multiple rows
INSERT CATEGORIES VALUES (30, 'BEVERAGES'), (40, 'CONDIMENTS');

-- Selected columns only
INSERT PRODUCTS (ID, PRODUCT_NAME)
VALUES (1, 'KEYBOARD'), (2, 'OPTICAL MOUSE'), (3, 'SCANNER');
```

In the above example, the omitted columns will be inserted with either NULL or DEFAULT (if specified in column definition)

The INSERT statement

- While inserting text or date/time values, you can use either single quot or double qout.
  - Note: Other database softwares may allow only single qouts as text qualifiers and hence, it is a good practice to use the same in MySQL also.
- You can use functions during a record insertion.
  - For example, to insert a registration date as current date, you may use INSERT INTO MEMBERS VALUES (9812, ‘SCOTT’, SYSDATE(), ‘DALLAS’);
- While inserting date values, use the YYYY-MM-DD format
  - INSERT INTO MEMBERS VALUES (1988, ‘MIKE’, ‘2012-07-22’, ‘UTAH’);

The INSERT statement

- You can also copy values from another table using the INSERT statement:
  - INSERT INTO TOP_10_EMPS
    SELECT EMPNO, ENAME, SALARY FROM EMPLOYEES
    ORDER BY SALARY DESC LIMIT 10
- The column types in the query should match with the column types in the target table.
- Do not use the VALUE/VALUES during this operation.

Modifying existing rows – UPDATE statement

- You can change the value of one or more columns of one or more rows in a table using the UPDATE statement.

```
UPDATE table_name
SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}]
...
[WHERE where_condition]
[ORDER BY column_name [{ASC|DESC}]]
[LIMIT row_count]
```

- If you do not specify the WHERE clause, then all the rows in the table will be updated

UPDATE - Examples

Increase the product price by 10% for all products:

```
UPDATE PRODUCTS
        SET PRICE = PRICE + PRICE * 10 / 100;
UPDATE PRODUCTS
        SET PRICE = PRICE * 1.1;
```

Increase the price by $5 for those products which are priced below $10:

```
UPDATE PRODUCTS
        SET PRICE = PRICE + 5
        WHERE PRICE < 10;
```

Convert the CITY, STATE, and COUNTRY into upper case letters:

```
UPDATE CUSTOMERS
        SET CITY=UPPER('CITY'),
        STATE = UPPER('STATE'),
        COUNTRY = UPPER('COUNTRY');
```

UPDATE - Examples

Change the CITY of the CUSTOMER whose ID is 1245 to the DEFAULT value:

```
UPDATE CUSTOMERS
        SET CITY = DEFAULT
        WHERE CUSTOMER_ID = 1245;
```

Change the DESIGNATION of TOP 3 salaried employees as ‘DIRECTOR’:

```
UPDATE EMPLOYEES
        SET DESIGNATION = ‘DIRECTOR’
        ORDER BY SALARY DESC
        LIMIT 3;
```

UPDATE - Examples

Change employee 114's job and salary to match that of employee 205.

```
UPDATE EMPLOYEES AS E1
          JOIN EMPLOYEES AS E2
          ON E2.EMPLOYEE_ID=205
          SET E1.JOB_ID = E2.JOB_ID,
          E1.SALARY = E2.SALARY
          WHERE
          E1.EMPLOYEE_ID = 114;
```

The source data and the corresponding condition.

The columns whose values need to be updated.

The rows in the target table to be updated.

In other database softwares, such as Oracle, the same thing can be achieved using a subquery as shown below:

```
UPDATE EMPLOYEES
SET JOB = (SELECT JOB FROM EMPLOYEES WHERE EMPLOYEE_ID = 205),
SALARY = (SELECT SALARY FROM EMPLOYEES WHERE EMPLOYEE_ID = 205)
WHERE EMPLOYEE_ID = 114;
```

However, in MySQL, this is not allowed.

You can not modify the same table which you are using in the SELECT part.

Updating rows – Integrity constraint violations

Trying to change the parent row's primary key value, while there are child records existing for this parent row:

```
UPDATE CATEGORIES SET ID=15 WHERE ID=10;
```

```
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`mydb`.`products`, CONSTRAINT `fk_prd_cat` FOREIGN KEY (`CATEGORY_ID`) REFERENCES `categories` (`ID`) ON DELETE CASCADE)
```

Trying to change the child row's foreign key value, while the new foreign key value is not present in the parent table:

```
UPDATE PRODUCTS SET CATEGORY_ID=15 WHERE ID = 3;
```

```
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`mydb`.`products`, CONSTRAINT ` fk_prd_cat` FOREIGN KEY (`CATEGORY_ID`) REFERENCES `categories` (`ID`) ON DELETE CASCADE)
```

Deleting Rows from a Table - the DELETE Statement

- You can remove existing rows from a table by using the DELETE statement.

```
DELETE [FROM] <table_name>
[WHERE <condition> ]
```

Deleting Rows from a Table - the DELETE Statement

```
mysql> select category_id, category_name from categories;
+-------------+---------------+
| category_id | category_name |
+-------------+---------------+
|           1 | Beverages     |
|           2 | Condiments    |
|           3 | Confections   |
|           4 | Dairy Products |
|           5 | Grains/Cereals |
|           6 | Meat/Poultry  |
|           7 | Produce       |
|           8 | Seafood       |
|          15 | Softdrinks    |
|          17 | Electronics   |
+-------------+---------------+
10 rows in set (0.00 sec)
```

```
mysql> delete from categories where category_id in (15, 17);
Query OK, 2 rows affected (0.09 sec)
```

Specific rows are deleted if you specify the **WHERE** clause.

```
mysql> select category_id, category_name from categories;
+-------------+---------------+
| category_id | category_name |
+-------------+---------------+
|           1 | Beverages     |
|           2 | Condiments    |
|           3 | Confections   |
|           4 | Dairy Products |
|           5 | Grains/Cereals |
|           6 | Meat/Poultry  |
|           7 | Produce       |
|           8 | Seafood       |
+-------------+---------------+
8 rows in set (0.00 sec)
```

Deleting Rows from a Table - the DELETE Statement

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

```
mysql> delete from categories;
Query OK, 8 rows affected (0.00 sec)

mysql> select category_id, category_name from categories;
Empty set (0.00 sec)
```

Database Transactions

- A transaction comprises a unit of work performed within a database management system against a database, and treated in a coherent and reliable way independent of other transactions.
- Transactions in a database environment have two main purposes:
- A database transaction, by definition, must be atomic, consistent, isolated and durable.
  - Database practitioners often refer to these properties of database transactions using the acronym ACID.

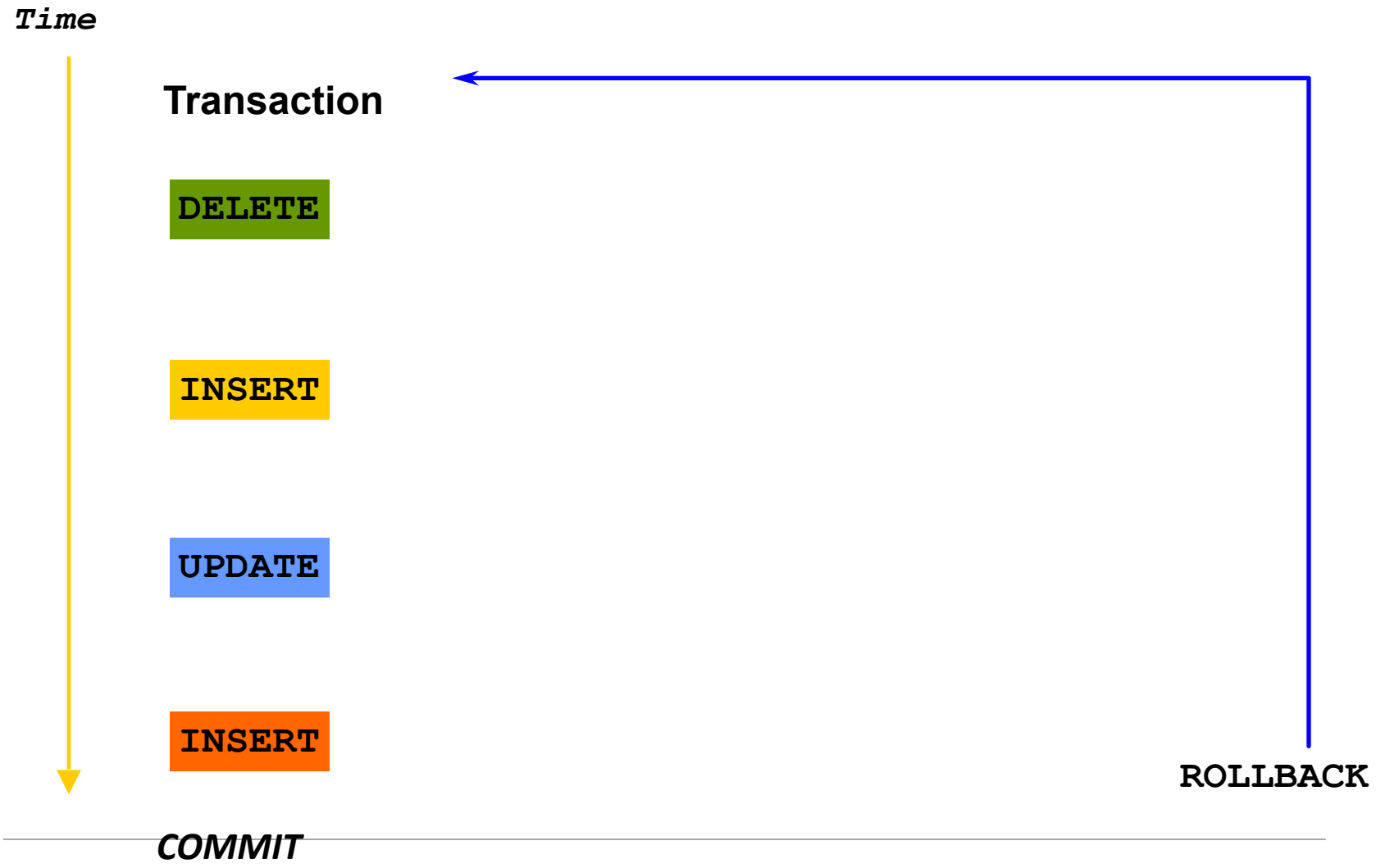
Database Transactions

- Transaction begins when the first DML SQL statement is executed.
- Ends with one of the following events:
  - A COMMIT or ROLLBACK statement is issued
  - A DDL statement executes (automatic commit)
  - The system crashes

Advantages of COMMIT and ROLLBACK Statements

- With COMMIT and ROLLBACK statements, you can:
  - Ensure data consistency
  - Preview data changes before making changes permanent
  - Group logically related operations

Controlling Transactions



Implicit Transaction Processing

- An automatic commit occurs under the following circumstances:
  - DDL statement is issued
  - The autocommit option is turned on
    - SET AUTOCOMMIT=1;
- By default, the autocommit is turned on by MySQL, and if you want it to be turned off, then issute this command:
  - SET AUTOCOMMIT=0;

Rolling back the data changes

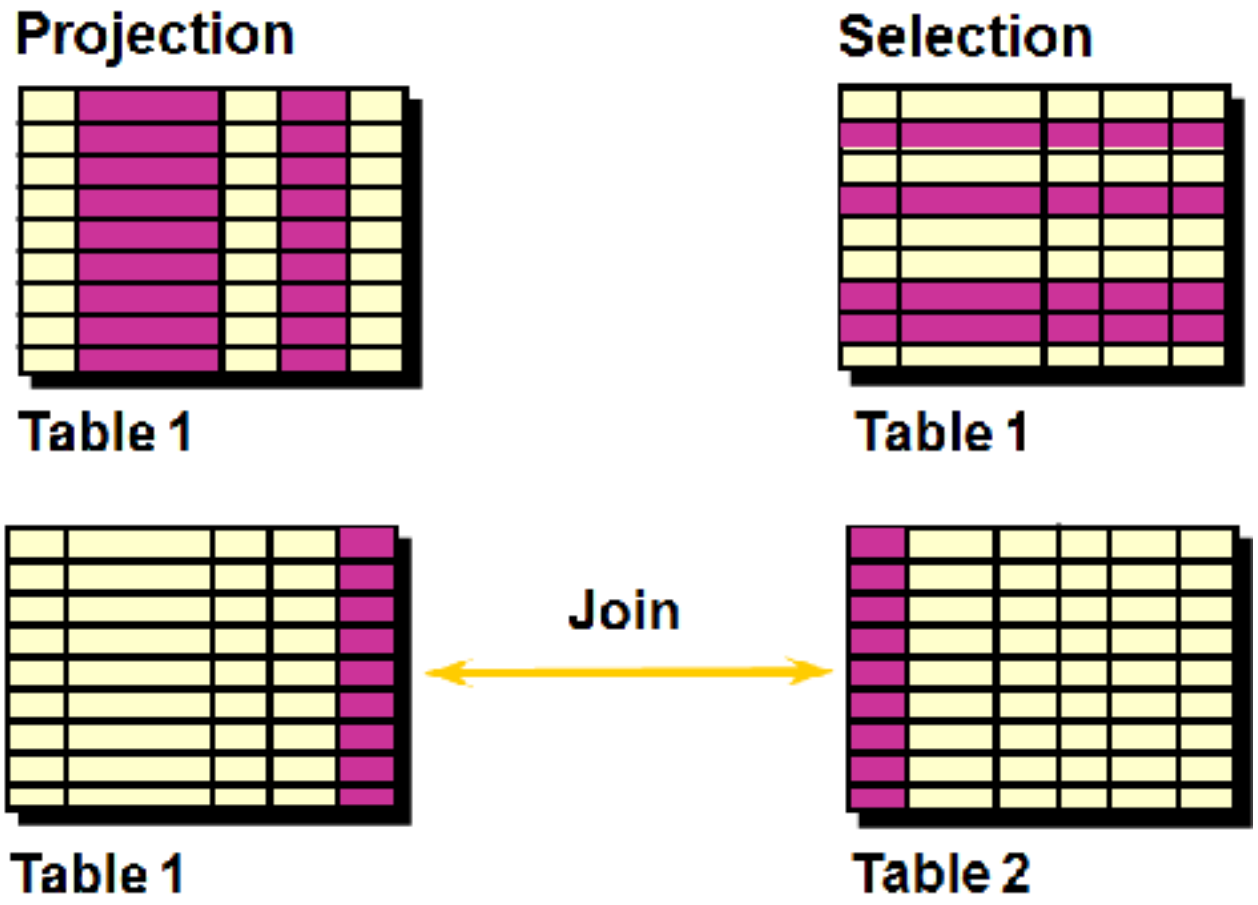
- Discard all pending changes by using the ROLLBACK statement:
  - Data changes are undone.
  - Previous state of the data is restored.
  - Locks on the affected rows are released.

Rolling back the data changes

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into categories values (60, 'SEA FOODS');
Query OK, 1 row affected (0.00 sec)

mysql> select * from categories;
+----+---------------+
| ID | CATEGORY_NAME |
+----+---------------+
| 10 | COMPUTERS     |
| 20 | STATIONARIES  |
| 30 | BEVERAGES     |
| 40 | CONDIMENTS    |
| 50 | CONFECTIONS   |
| 60 | SEA FOODS     |
+----+---------------+
6 rows in set (0.00 sec)
mysql> rollback;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from categories;
+----+---------------+
| ID | CATEGORY_NAME |
+----+---------------+
| 10 | COMPUTERS     |
| 20 | STATIONARIES  |
| 30 | BEVERAGES     |
| 40 | CONDIMENTS    |
| 50 | CONFECTIONS   |
+----+---------------+
5 rows in set (0.00 sec)
```

Queries

Different operations on a table



SELECT - Syntax

SELECT [DISTINCT] *col_name* [, col_name...]
[FROM *table_references*]
[WHERE *where_condition*]
[GROUP BY {*col_name*} [HAVING *where_condition*]]
[ORDER BY {*col_name*} [ASC | DESC], ...]
[LIMIT *row_count*]
[OFFSET *offset*]

- SELECT identifies which columns to retrieve
- FROM identifies which tables to retrieve the data from
- WHERE identifies which rows to filter (or retrieve)
- GROUP BY identifies how to summarize the data
- ORDER BY identifies how the data need to be arranged
- LIMIT identifies how many rows to be retrieved within the result
- OFFSET identifies which record to start retrieving the result from

You can refer the following link for the detailed syntax and usage:
http://dev.mysql.com/doc/refman/5.1/en/select.html

Selecting all columns

```
SELECT *
FROM SHIPPERS;
```

```
mysql> SELECT * FROM SHIPPERS;
+------------+------------------+----------------+
| SHIPPER_ID | COMPANY_NAME     | PHONE          |
+------------+------------------+----------------+
|          1 | Speedy Express   | (503) 555-9831 |
|          2 | United Package   | (503) 555-3199 |
|          3 | Federal Shipping | (503) 555-9931 |
+------------+------------------+----------------+
3 rows in set (0.07 sec)
```

Selecting specific columns (Projection)

```
SELECT CATEGORY_NAME, DESCRIPTION
FROM CATEGORIES;
```

```
+---------------+-----------------------------------------------------------+
| CATEGORY_NAME | DESCRIPTION                                               |
+---------------+-----------------------------------------------------------+
| Beverages     | Soft drinks, coffees, teas, beers, and ales               |
| Condiments    | Sweet and savory sauces, relishes, spreads, and seasonings |
| Confections   | Desserts, candies, and sweet breads                       |
| Dairy Products | Cheeses                                                   |
| Grains/Cereals | Breads, crackers, pasta, and cereal                      |
| Meat/Poultry  | Prepared meats                                            |
| Produce       | Dried fruit and bean curd                                 |
| Seafood       | Seaweed and fish                                          |
+---------------+-----------------------------------------------------------+
8 rows in set (0.07 sec)
```

Writing SQL Statements

- SQL statements are not case sensitive
- SQL statements can be on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.

Arithmetic Expressions

- Create expressions with number and date data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulus

Using arithmetic operators

```
SELECT PRODUCT_NAME, UNIT_PRICE,
UNIT_PRICE * 0.5 AS PROFIT
FROM PRODUCTS LIMIT 10;
```

```
+----------------------------------+------------+----------+
| PRODUCT_NAME                     | UNIT_PRICE | PROFIT   |
+----------------------------------+------------+----------+
| Chai                             |    18.0000 |  9.00000 |
| Chang                            |    19.0000 |  9.50000 |
| Aniseed Syrup                    |    10.0000 |  5.00000 |
| Chef Anton's Cajun Seasoning     |    22.0000 | 11.00000 |
| Chef Anton's Gumbo Mix           |    21.3500 | 10.67500 |
| Grandma's Boysenberry Spread     |    25.0000 | 12.50000 |
| Uncle Bob's Organic Dried Pears  |    30.0000 | 15.00000 |
| Northwoods Cranberry Sauce       |    40.0000 | 20.00000 |
| Mishi Kobe Niku                  |    97.0000 | 48.50000 |
| Ikura                            |    31.0000 | 15.50000 |
+----------------------------------+------------+----------+
10 rows in set (0.14 sec)
```

Operator precedence

- Multiplication and division take priority over addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritized evaluation and to clarify statements.

Operator precedence - Example

```
SELECT PRODUCT_NAME,
UNIT_PRICE * UNITS_IN_STOCK - REORDER_LEVEL AS EXPR1,
UNIT_PRICE * (UNITS_IN_STOCK - REORDER_LEVEL) AS EXPR2
FROM PRODUCTS LIMIT 10;
```

PRODUCT_NAME	EXPR1	EXPR2
Chai	692.0000	522.0000
Chang	298.0000	-152.0000
Aniseed Syrup	105.0000	-120.0000
Chef Anton's Cajun Seasoning	1166.0000	1166.0000
Chef Anton's Gumbo Mix	0.0000	0.0000
Grandma's Boysenberry Spread	2975.0000	2375.0000
Uncle Bob's Organic Dried Pears	440.0000	150.0000
Northwoods Cranberry Sauce	240.0000	240.0000
Mishi Kobe Niku	2813.0000	2813.0000
Ikura	961.0000	961.0000

10 rows in set (0.03 sec)

NULL values

- A null is a value that is unavailable, unassigned, unknown, or inapplicable.
- A null is not the same as zero or a blank space.

```
SELECT FIRST_NAME, CITY,
REGION FROM EMPLOYEES;
```

FIRST_NAME	CITY	REGION
Nancy	Seattle	WA
Andrew	Tacoma	WA
Janet	Kirkland	WA
Margaret	Redmond	WA
Steven	London	NULL
Michael	London	NULL
Robert	London	NULL
Laura	Seattle	WA
Anne	London	NULL

NULL values in expressions

- Arithmetic expressions containing a null value evaluate to null.

```
SELECT FIRST_NAME,
CONCAT(CITY, ' ', REGION)
FROM EMPLOYEES;
```

FIRST_NAME	CONCAT(CITY, ' ', REGION)
Nancy	Seattle WA
Andrew	Tacoma WA
Janet	Kirkland WA
Margaret	Redmond WA
Steven	NULL
Michael	NULL
Robert	NULL
Laura	Seattle WA
Anne	NULL

NULL values in expressions

- If any column value in an arithmetic expression is null, the result is null.
  - For example, if you attempt to perform division with zero, you get an error. However, if you divide a number by null, the result is a null or unknown.

Defining a column alias

- A column alias:
  - Renames a column heading
  - Is useful with calculations
  - Immediately follows the column name
    - There can also be the optional '**AS'** keyword between the column name and alias
  - If the alias contains a space, then it must be quoted.

Column alias - Example

```
SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME) AS 'EMPLOYEE NAME' , CITY
FROM EMPLOYEES;
```

EMPLOYEE NAME	CITY
Nancy Davolio	Seattle
Andrew Fuller	Tacoma
Janet Leverling	Kirkland
Margaret Peacock	Redmond
Steven Buchanan	London
Michael Suyama	London
Robert King	London
Laura Callahan	Seattle
Anne Dodsworth	London

Concatenation

- Unlike other DB softwares, MySQL does not use the double-pipe for concatenation.
  - For example, the following SQL statement would result in undesired output:
    SELECT FIRST_NAME||' '||LAST_NAME AS 'EMPLOYEE NAME', CITY FROM EMPLOYEES;
  - The reason for this is, MySQL considers double-pipe as ‘logical OR’ operator and not as concatenation operator.
  - You may use this operator by setting the SQL_MODE to ‘ANSI’

EMPLOYEE NAME	CITY
0	Seattle
0	Tacoma
0	Kirkland
0	Redmond
0	London
0	London
0	London
0	Seattle
0	London

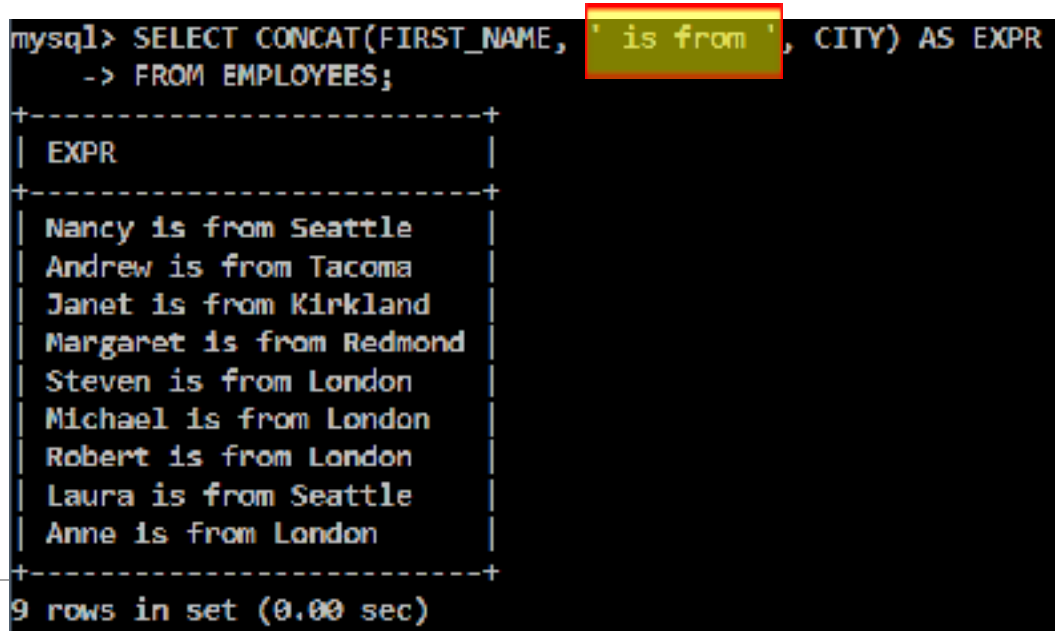
Concatenation

```
mysql> SET SQL_MODE='ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT FIRST_NAME||' '||LAST_NAME AS 'EMPLOYEE NAME',
    -> CITY FROM EMPLOYEES;
+------------------+----------+
| EMPLOYEE NAME    | CITY     |
+------------------+----------+
| Nancy Davolio    | Seattle  |
| Andrew Fuller    | Tacoma   |
| Janet Leverling  | Kirkland |
| Margaret Peacock | Redmond  |
| Steven Buchanan  | London   |
| Michael Suyama   | London   |
| Robert King      | London   |
| Laura Callahan   | Seattle  |
| Anne Dodsworth   | London   |
+------------------+----------+
9 rows in set (0.00 sec)
```

A better and preferred way of concatenating is to use the CONCAT function. You can supply multiple arguments to this function.

Literal Character Strings

- A literal is a character, a number, or a date included in the SELECT list.
- Date and character literal values must be enclosed within single quotation marks.
- Each character string is output once for each row returned.



Restricting and Sorting Data

- The combination of SELECT and FROM retrieves all the rows.
- You can limit the number of rows based on a condition using the WHERE clause.

```
Retrieve all employees from ‘London’ city
```

```
mysql> SELECT FIRST_NAME, LAST_NAME FROM EMPLOYEES
    -> WHERE CITY = 'LONDON';
+------------+-----------+
| FIRST_NAME | LAST_NAME |
+------------+-----------+
| Steven     | Buchanan  |
| Michael    | Suyama    |
| Robert     | King      |
| Anne       | Dodsworth |
+------------+-----------+
4 rows in set (0.00 sec)
```

Character Strings and Dates

- Character strings and date values are enclosed in single quotation marks.
  - However, in the WHERE clause, strings and dates can be given in single or double quotes
- Character values are NOT case sensitive
  - Unless you are using MySQL on Unix/Linux platform
- The default date format is YYYY-MM-DD and datetime format is YYYY-MM-DD hh:mm:ss

```
mysql> SELECT LAST_NAME, FIRST_NAME, BIRTH_DATE, HIRE_DATE
    -> FROM EMPLOYEES
    -> WHERE LAST_NAME = 'KING';
+-----------+------------+---------------------+---------------------+
| LAST_NAME | FIRST_NAME | BIRTH_DATE          | HIRE_DATE           |
+-----------+------------+---------------------+---------------------+
| King      | Robert     | 1960-05-29 00:00:00 | 1994-01-02 00:00:00 |
+-----------+------------+---------------------+---------------------+
1 row in set (0.00 sec)
```

Operators for WHERE clause

Operator	Description
=	Equals to
>	Greater than
>=	Greater than or equals to
<	Less than
<=	Less than or equals to
!=	Not equals to
<>	Not equals to
BETWEEN, NOT BETWEEN	Compare to a range of values
IN, NOT IN	Match (or do not) any of the value in the list
LIKE, NOT LIKE	Match (or do not) a character pattern
IS NULL, IS NOT NULL	Compare with NULL value

WHERE - Examples

```
Retrieve the id, name, and price of all products that belong to
category 1

SELECT PRODUCT_ID, PRODUCT_NAME, UNIT_PRICE
FROM PRODUCTS
WHERE CATEGORY_ID = 1;
```

```
Retrieve the id, name, price in stock for all products that are
not in stock

SELECT PRODUCT_ID, PRODUCT_NAME, UNIT_PRICE
FROM PRODUCTS
WHERE UNITS_IN_STOCK = 0;
```

WHERE - Examples

```
Retrieve the details of all the products cost less than $20

SELECT * FROM PRODUCTS
WHERE UNIT_PRICE < 20;
```

```
Retrieve the details of all the products that are priced between
$20 and $50

SELECT * FROM PRODUCTS
WHERE UNIT_PRICE BETWEEN 20 AND 50;
```

WHERE - Examples

Retrieve the name and city of all customers who live in ‘London’, ‘Paris’, or ‘Madrid’

```
SELECT CUSTOMER_NAME, CITY
FROM CUSTOMERS
WHERE CITY = 'LONDON' OR CITY = 'PARIS' OR CITY = 'MADRID';
```

```
SELECT CUSTOMER_NAME, CITY
FROM CUSTOMERS
WHERE CITY IN ('LONDON', 'PARIS', 'MADRID');
```

WHERE - Examples

Retrieve the names of the suppliers who do not have a website.

```
SELECT COMPANY_NAME FROM SUPPLIERS
WHERE HOME_PAGE IS NULL;
```

Retrieve the names of the suppliers who have a website address along with their homepage.

```
SELECT COMPANY_NAME, HOME_PAGE FROM SUPPLIERS
WHERE HOME_PAGE IS NOT NULL;
```

WHERE - Examples

Retrieve the name and city of the all customers whose name ends with ‘EN’

```
SELECT CUSTOMER_NAME, CITY FROM CUSTOMERS
WHERE CUSTOMER_NAME LIKE ‘%EN’;
```

Retrieve the name and joining date of all employees who joined in the year 1993

```
SELECT FIRST_NAME, LAST_NAME, HIRE_DATE FROM EMPLOYEES
WHERE HIRE_DATE LIKE '1993-%';
```

A better and preferred method is using the year() function as shown below:

```
SELECT FIRST_NAME, LAST_NAME, HIRE_DATE FROM EMPLOYEES
WHERE YEAR(HIRE_DATE) = 1993;
```

Logical Operators

Operator	Description
AND	Returns TRUE if both conditions are true
OR	Returns TRUE if either of the conditions is true
NOT	Returns TRUE if the following condition is false

Using the AND operator

- AND requires both conditions to be true

```
SELECT PRODUCT_NAME, CATEGORY_ID, UNIT_PRICE
FROM PRODUCTS
WHERE UNIT_PRICE BETWEEN 10 AND 50
AND
CATEGORY_ID = 1;
```

PRODUCT_NAME	CATEGORY_ID	UNIT_PRICE
Chai	1	18.0000
Chang	1	19.0000
Sasquatch Ale	1	14.0000
Steeleye Stout	1	18.0000
Chartreuse verte	1	18.0000
Ipoh Coffee	1	46.0000
Laughing Lumberjack Lager	1	14.0000
Outback Lager	1	15.0000
Lakkalik╠╣╠╣ri	1	18.0000

Using the OR operator

- OR requires any one of the conditions to be true

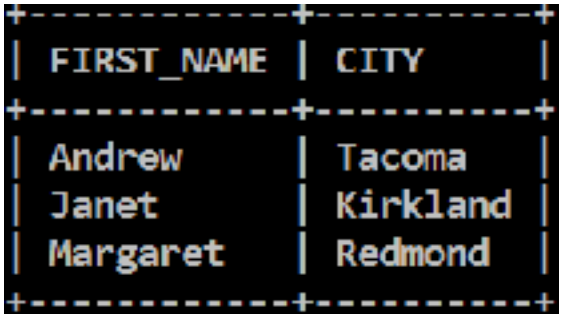
```
SELECT PRODUCT_NAME, UNIT_PRICE, UNITS_IN_STOCK
FROM PRODUCTS
WHERE
UNITS_IN_STOCK = 0
OR
UNIT_PRICE > 100;
```

PRODUCT_NAME	UNIT_PRICE	UNITS_IN_STOCK
Chef Anton's Gumbo Mix	21.3500	0
Alice Mutton	39.0000	0
Th╜ringer Rostbratwurst	123.7900	0
Gorgonzola Telino	12.5000	0
C┼te de Blaye	263.5000	17
Perth Pasties	32.8000	0

Using the OR operator

- NOT is used for complementing the result of the condition

```
SELECT FIRST_NAME, CITY FROM EMPLOYEES
WHERE CITY NOT IN ('SEATTLE', 'LONDON');
```



FIRST_NAME	CITY
Andrew	Tacoma
Janet	Kirkland
Margaret	Redmond

- You may try NOT BETWEEN, NOT LIKE, IS NOT NULL also.

Precedence control

- Note that the result does not match the expected output due to the default precedence and flow of evaluation (left to right)

```
mysql> select product_name, supplier_id, unit_price
    -> from products
    -> where unit_price between 10 and 20 and supplier_id=1 or supplier_id=2;
+----------------------------------+-------------+------------+
| product_name                     | supplier_id | unit_price |
+----------------------------------+-------------+------------+
| Chai                             |           1 |    18.0000 |
| Chang                            |           1 |    19.0000 |
| Aniseed Syrup                    |           1 |    10.0000 |
| Chef Anton's Cajun Seasoning     |           2 |    22.0000 |
| Chef Anton's Gumbo Mix           |           2 |    21.3500 |
| Louisiana Fiery Hot Pepper Sauce |           2 |    21.0500 |
| Louisiana Hot Spiced Okra        |           2 |    17.0000 |
+----------------------------------+-------------+------------+
7 rows in set (0.00 sec)
```

Precedence control

- Use the parentheses to control the precedence

```
mysql> select product_name, supplier_id, unit_price
    -> from products
    -> where unit_price between 10 and 20 and (supplier_id=1 or supplier_id=2);
+--------------------------+-------------+------------+
| product_name             | supplier_id | unit_price |
+--------------------------+-------------+------------+
| Chai                     |           1 |    18.0000 |
| Chang                    |           1 |    19.0000 |
| Aniseed Syrup            |           1 |    10.0000 |
| Louisiana Hot Spiced Okra |           2 |    17.0000 |
+--------------------------+-------------+------------+
4 rows in set (0.00 sec)
```

Sorting the results

- Sort rows with the ORDER BY clause
  - ASC: ascending order, default
  - DESC: descending order
- The ORDER BY clause comes at the end in the SELECT statement.

Sorting the results

```
SELECT PRODUCT_NAME, CATEGORY_ID, SUPPLIER_ID, UNIT_PRICE
FROM PRODUCTS
WHERE UNIT_PRICE > 40
ORDER BY UNIT_PRICE DESC;
```

PRODUCT_NAME	CATEGORY_ID	SUPPLIER_ID	UNIT_PRICE
C├┤te de Blaye	1	18	263.5000
Th├╝ringer Rostbratwurst	6	12	123.7900
Mishi Kobe Niku	6	4	97.0000
Sir Rodney's Marmalade	3	8	81.0000
Carnarvon Tigers	8	7	62.5000
Raclette Courdavault	4	28	55.0000
Manjimup Dried Apples	7	24	53.0000
Tarte au sucre	3	29	49.3000
Ipoh Coffee	1	20	46.0000
R├╢ssle Sauerkraut	7	12	45.6000
Schoggi Schokolade	3	11	43.9000
Vegie-spread	2	7	43.9000

Sorting by Column Alias

```
SELECT PRODUCT_NAME,
UNIT_PRICE * UNITS_IN_STOCK AS STOCK_VALUE
FROM PRODUCTS
ORDER BY STOCK_VALUE DESC LIMIT 10;
```

PRODUCT_NAME	STOCK_VALUE
C┼te de Blaye	4479.5000
Raclette Courdavault	4345.0000
Queso Manchego La Pastora	3268.0000
Sir Rodney's Marmalade	3240.0000
Sirop d'┼rable	3220.5000
Grandma's Boysenberry Spread	3000.0000
Mishi Kobe Niku	2813.0000
P┼ót┼ chinois	2760.0000
Carnarvon Tigers	2625.0000
Boston Crab Meat	2263.2000

Sorting by multiple columns

- The second column in the sort-list will be used only if the values in the first column being used for sorting has duplicate values.

```
SELECT PRODUCT_NAME, CATEGORY_ID, UNIT_PRICE
FROM PRODUCTS WHERE UNIT_PRICE > 40
ORDER BY CATEGORY_ID, UNIT_PRICE DESC;
```

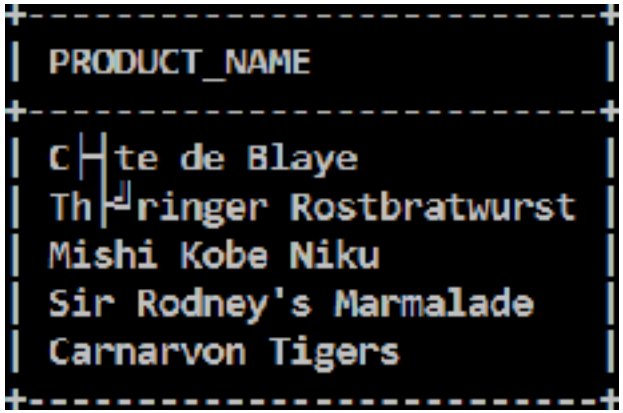
PRODUCT_NAME	CATEGORY_ID	UNIT_PRICE
C╪te de Blaye	1	263.5000
Ipoh Coffee	1	46.0000
Vegie-spread	2	43.9000
Sir Rodney's Marmalade	3	81.0000
Tarte au sucre	3	49.3000
Schoggi Schokolade	3	43.9000
Raclette Courdavault	4	55.0000
Th╝ringer Rostbratwurst	6	123.7900
Mishi Kobe Niku	6	97.0000
Manjimup Dried Apples	7	53.0000
R╪ssle Sauerkraut	7	45.6000
Carnarvon Tigers	8	62.5000

Note that the data is initially sorted based on category id and then within each category group, the data is sorted based on the price in descending order

Sorting and Projection

- The column used for sorting need not be in the column list beging retrieved.
  - For example,
    Retrieve the top 5 costliest product names:

```
SELECT PRODUCT_NAME FROM PRODUCTS
ORDER BY UNIT_PRICE DESC
LIMIT 5;
```



Conditional Expressions – CASE statement

- CASE statement
- Syntax:
  - CASE value
    WHEN [compare_value] THEN result
    [WHEN [compare_value] THEN result ...]
    [ELSE result]
    END

Conditional Expressions – CASE statement

```
SELECT PRODUCT_NAME,
CASE CATEGORY_ID
        WHEN 1 THEN 'BEVERAGES'
        WHEN 2 THEN 'CONDIMENTS'
        WHEN 3 THEN 'CONFECTIONS'
        ELSE 'GENERAL'
END AS CATEGORY
FROM PRODUCTS
ORDER BY CATEGORY;
```

PRODUCT_NAME	CATEGORY
Chai	BEVERAGES
Sasquatch Ale	BEVERAGES
Steeleye Stout	BEVERAGES
C┼te de Blaye	BEVERAGES
Ipoh Coffee	BEVERAGES
Guaran├í Fant├ística	BEVERAGES
Chartreuse verte	BEVERAGES
Laughing Lumberjack Lager	BEVERAGES
Outback Lager	BEVERAGES
Rh┼nbr├u Klosterbier	BEVERAGES
Chang	BEVERAGES
Lakkalik┼┼ri	BEVERAGES
Aniseed Syrup	CONDIMENTS
Chef Anton's Cajun Seasoning	CONDIMENTS
Chef Anton's Gumbo Mix	CONDIMENTS
Grandma's Boysenberry Spread	CONDIMENTS
Gula Malacca	CONDIMENTS
Vegie-spread	CONDIMENTS
Original Frankfurter gr╝ne So├ße	CONDIMENTS
Sirop d'├rable	CONDIMENTS
Northwoods Cranberry Sauce	CONDIMENTS
Louisiana Hot Spiced Okra	CONDIMENTS
Louisiana Fiery Hot Pepper Sauce	CONDIMENTS
Genen Shouyu	CONDIMENTS
Tarte au sucre	CONFECTIONS
Chocolade	CONFECTIONS
Zaanse koeken	CONFECTIONS
Maxilaku	CONFECTIONS
Scottish Longbreads	CONFECTIONS

Conditional Expressions – IF statement

- IF statement
- Syntax:
  - IF(expr1,expr2,expr3)
  - If expr1 is TRUE (expr1 <> 0 and expr1 <> NULL)
    - then IF() returns expr2;
    - otherwise it returns expr3.
  - IF() returns a numeric or string value, depending on the context in which it is used.

Conditional Expressions – IF statement

```
SELECT PRODUCT_NAME, UNIT_PRICE,
IF(UNIT_PRICE >= 25, 'COSTLY', 'CHEAP') AS 'IS COSTLY'
FROM PRODUCTS
LIMIT 15;
```

PRODUCT_NAME	UNIT_PRICE	IS COSTLY
Chai	18.0000	CHEAP
Chang	19.0000	CHEAP
Aniseed Syrup	10.0000	CHEAP
Chef Anton's Cajun Seasoning	22.0000	CHEAP
Chef Anton's Gumbo Mix	21.3500	CHEAP
Grandma's Boysenberry Spread	25.0000	COSTLY
Uncle Bob's Organic Dried Pears	30.0000	COSTLY
Northwoods Cranberry Sauce	40.0000	COSTLY
Mishi Kobe Niku	97.0000	COSTLY
Ikura	31.0000	COSTLY
Queso Cabrales	21.0000	CHEAP
Queso Manchego La Pastora	38.0000	COSTLY
Konbu	6.0000	CHEAP
Tofu	23.2500	CHEAP

Conditional Expressions – IFNULL statement

- IFNULL statement
- Syntax:
  - IFNULL(expr1,expr2)
  - If expr1 is not NULL,
    - IFNULL() returns expr1;
    - otherwise it returns expr2.
  - IFNULL() returns a numeric or string value, depending on the context in which it is used

Conditional Expressions – IFNULL statement

```
SELECT FIRST_NAME, CITY,
IFNULL(REGION, '--N/A--') AS REGION
FROM EMPLOYEES;
```

FIRST_NAME	CITY	REGION
Nancy	Seattle	WA
Andrew	Tacoma	WA
Janet	Kirkland	WA
Margaret	Redmond	WA
Steven	London	--N/A--
Michael	London	--N/A--
Robert	London	--N/A--
Laura	Seattle	WA
Anne	London	--N/A--

Conditional Expressions – NULLIF statement

- NULLIF statement
- Syntax:
  - NULLIF(expr1,expr2)
    - Returns NULL if expr1 = expr2 is true,
    - otherwise returns expr1.

Conditional Expressions – NULLIF statement

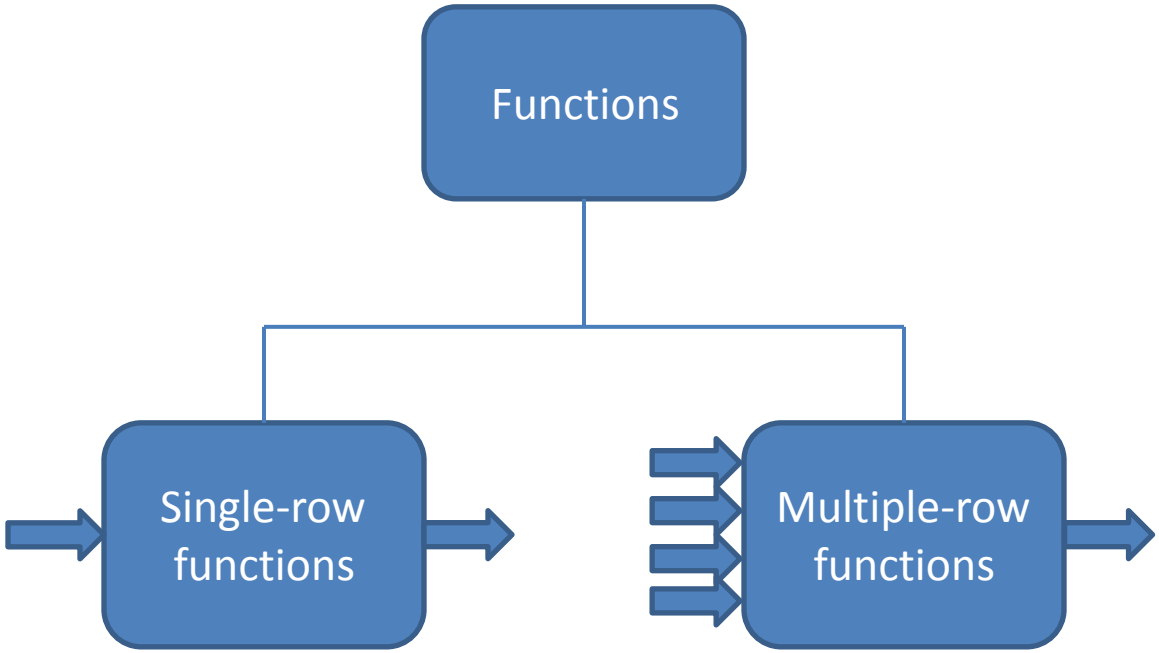
```
SELECT PRODUCT_NAME, UNIT_PRICE,
NULLIF(UNITS_IN_STOCK, 0) AS UNITS_IN_STOCK FROM PRODUCTS;
```

PRODUCT_NAME	UNIT_PRICE	UNITS_IN_STOCK
Chai	18.0000	39
Chang	19.0000	17
Aniseed Syrup	10.0000	13
Chef Anton's Cajun Seasoning	22.0000	53
Chef Anton's Gumbo Mix	21.3500	NULL
Grandma's Boysenberry Spread	25.0000	120
Uncle Bob's Organic Dried Pears	30.0000	15
Northwoods Cranberry Sauce	40.0000	6
Mishi Kobe Niku	97.0000	29
Ikura	31.0000	31
Queso Cabrales	21.0000	22
Queso Manchego La Pastora	38.0000	86
Konbu	6.0000	24
Tofu	23.2500	35
Genen Shouyu	15.5000	39
Pavlova	17.4500	29
Alice Mutton	39.0000	NULL
Carnarvon Tigers	62.5000	42

Functions

- A function is a callable unit of code to perform a task
- A function may take 0 or more arguments (parameters), which can be fed as input to the job to be done
- A function returns a value, which usually represents the result of the job being done
- For example,
  - The sysdate() function does not take any parameter, but returns the current system date and time:
    SELECT SYSDATE();
  - The length() function takes one string parameter and returns the no.of characters in it:
    SELECT LENGTH('MYSQL IS FUN');

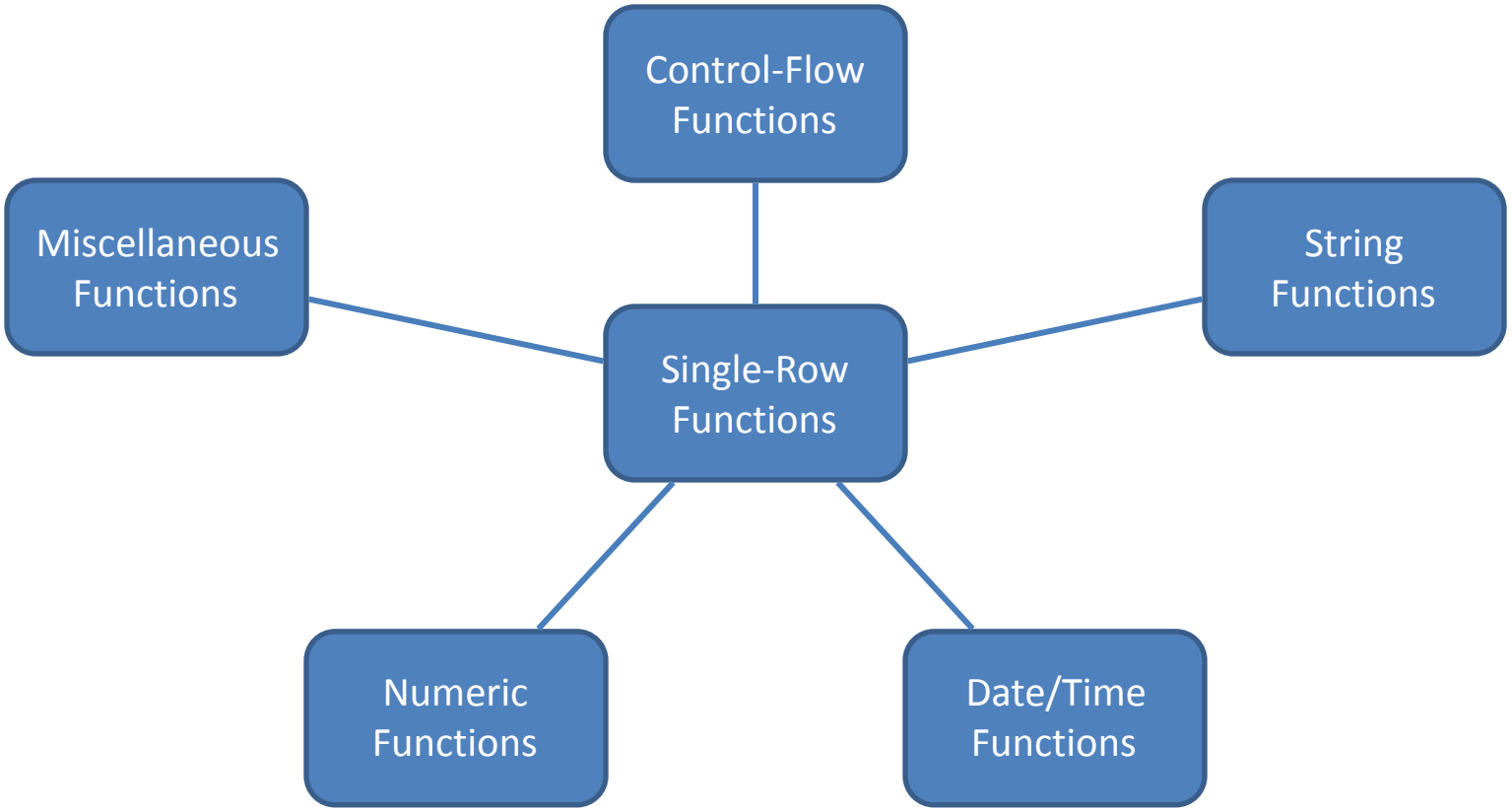
Two types of functions



Single-Row functions

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

Single-Row functions



Control-Flow functions

- IF()
  - Used for If/Else construct
- IFNULL()
  - Null-if/else construct
  - Used for NULL value substitution
- NULLIF()
  - Return NULL if expr1 = expr2
- This topic has already been discussed in the last session.

Complete Reference: http://dev.mysql.com/doc/refman/5.1/en/control-flow-functions.html

String functions

Function	Description
CHAR()	Return the character for each integer passed
CHAR_LENGTH() CHARACTER_LENGTH()	Returns the length of the string str, measured in characters. A multi-byte character counts as a single character.
CONCAT() CONCAT_WS()	CONCAT_WS() stands for Concatenate With Separator and is a special form of CONCAT().
FORMAT()	Return a number formatted to specified number of decimal places
LOWER() LCASE()	Return the argument in lowercase
LEFT() RIGHT()	Return the leftmost (rightmost) number of characters as specified
LENGTH()	Return the length of a string in bytes
LOAD()	Load the named file. Used for inserting into BLOB fields.
LOCATE() POSITION()	Return the position of the first occurrence of substring
LPAD() RPAD()	Return the string argument, left (or right) padded with the specified string

String functions

Function	Description
LTRIM() RTRIM() TRIM()	Removes extra spaces (left, right or both)
SPACE()	Return a string of the specified number of spaces
SUBSTR()	Return the substring as specified
UPPER() UCASE()	Convert to uppercase

Complete reference: http://dev.mysql.com/doc/refman/5.1/en/string-functions.html

String functions - Examples

```
SELECT UPPER(CUSTOMER_NAME) FROM CUSTOMERS LIMIT 10;
SELECT LOWER(CUSTOMER_NAME) FROM CUSTOMERS LIMIT 10;

-- Case insensitive comparison
SELECT * FROM CUSTOMERS
        WHERE UPPER(CITY) = ‘LONDON’;
```

```
SELECT
CONCAT(FIRST_NAME, LAST_NAME, 'lives in', CITY) AS INFO1,
CONCAT_WS(' ', FIRST_NAME, LAST_NAME, 'lives in', CITY) AS INFO2
FROM EMPLOYEES;
```

INFO1	INFO2
NancyDavoliolives inSeattle	Nancy Davolio lives in Seattle
AndrewFullerlives inTacoma	Andrew Fuller lives in Tacoma
JanetLeverlinglives inKirkland	Janet Leverling lives in Kirkland
MargaretPeacocklives inRedmond	Margaret Peacock lives in Redmond
StevenBuchananlives inLondon	Steven Buchanan lives in London
MichaelSuyamalives inLondon	Michael Suyama lives in London
RobertKinglives inLondon	Robert King lives in London
LauraCallahanlives inSeattle	Laura Callahan lives in Seattle
AnneDodsworthlives inLondon	Anne Dodsworth lives in London

Date/time functions

Function	Description
CURDATE() CURRENT_DATE() CURRENT_DATE	Return the current date
CURTIME() CURRENT_TIME() CURRENT_TIME	Return the current time
CURRENT_TIMESTAMP() CURRENT_TIMESTAMP() NOW() SYSDATE() LOCALTIME() LOCALTIME LOCALTIMESTAMP() LOCALTIMESTAMP	Return the current date and time
DATE_ADD()	Add time values (intervals) to a date value
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
MAKEDATE()	Create a date from the year and day of year

Date/time functions

Function	Description
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)

Complete reference: http://dev.mysql.com/doc/refman/5.1/en/date-and-time-functions.html

Date/Time functions - Examples

```
SELECT SYSDATE(), DAYOFYEAR(SYSDATE()), DAYNAME(SYSDATE());
```

```
+---------------------+-----------------------+--------------------+
| SYSDATE()           | DAYOFYEAR(SYSDATE()) | DAYNAME(SYSDATE()) |
+---------------------+-----------------------+--------------------+
| 2012-07-18 17:27:13 |                   200 | Wednesday          |
+---------------------+-----------------------+--------------------+
```

```
SELECT MAKEDATE(2012, 200);
```

```
+--------------------+
| MAKEDATE(2012, 200) |
+--------------------+
| 2012-07-18         |
+--------------------+
```

Date/Time functions - Examples

```
SELECT DATE_ADD('2012-01-01', INTERVAL 365 DAY);
```

```
+-----------------------------------------+
| DATE_ADD('2012-01-01', INTERVAL 365 DAY) |
+-----------------------------------------+
| 2012-12-31                              |
+-----------------------------------------+
```

```
SELECT NOW(), DATE_ADD(NOW(), INTERVAL 10 HOUR);
```

```
+---------------------+-------------------------------------+
| NOW()               | DATE_ADD(NOW(), INTERVAL 10 HOUR) |
+---------------------+-------------------------------------+
| 2012-07-18 17:31:22 | 2012-07-19 03:31:22                 |
+---------------------+-------------------------------------+
```

For more information about INTERVAL,
Visit http://dev.mysql.com/doc/refman/5.1/en/date-and-time-functions.html#function_date-add

Date/Time functions - Examples

```
SELECT
CONCAT_WS(' ', FIRST_NAME, LAST_NAME, 'has',
        DATEDIFF(SYSDATE(), HIRE_DATE) DIV 365,
        'years experience') AS INFO
FROM EMPLOYEES;
```

```
+----------------------------------------+
| INFO                                   |
+----------------------------------------+
| Nancy Davolio has 20 years experience  |
| Andrew Fuller has 19 years experience  |
| Janet Leverling has 20 years experience|
| Margaret Peacock has 19 years experience|
| Steven Buchanan has 18 years experience|
| Michael Suyama has 18 years experience |
| Robert King has 18 years experience    |
| Laura Callahan has 18 years experience |
| Anne Dodsworth has 17 years experience |
+----------------------------------------+
```

Numeric functions

Function	Description
ABS()	Return the absolute value
ROUND()	Round the argument
RAND()	Return a random floating-point value
POW(), POWER()	Return the argument raised to the specified power
SIGN()	Return the sign of the argument
TRUNCATE()	Truncate to specified number of decimal places
CEIL()	Return the smallest integer value not less than the argument
FLOOR()	Return the largest integer value not greater than the argument
MOD()	Return the remainder
CONV()	Convert numbers between different number bases

Complete reference: http://dev.mysql.com/doc/refman/5.1/en/mathematical-functions.html

Numeric functions - Examples

```
SELECT CEIL(2.16), CEIL(2.96),
         FLOOR(2.16), FLOOR(2.96);
```

```
+-----------+-----------+-----------+-----------+
| CEIL(2.16) | CEIL(2.96) | FLOOR(2.16) | FLOOR(2.96) |
+-----------+-----------+-----------+-----------+
|          3 |          3 |          2 |          2 |
+-----------+-----------+-----------+-----------+
```

```
SELECT ROUND(2.16, 1), ROUND(2.96, 1),
         TRUNCATE(2.16, 1), TRUNCATE(2.96, 1);
```

```
+---------------+---------------+-----------------+-----------------+
| ROUND(2.16, 1) | ROUND(2.96, 1) | TRUNCATE(2.16, 1) | TRUNCATE(2.96, 1) |
+---------------+---------------+-----------------+-----------------+
|           2.2 |           3.0 |               2.1 |               2.9 |
+---------------+---------------+-----------------+-----------------+
```

```
SELECT CONV(1974, 10, 2), CONV(1974, 10, 8), CONV(1974, 10, 16);
```

```
+-----------------+-----------------+-----------------+
| CONV(1974, 10, 2) | CONV(1974, 10, 8) | CONV(1974, 10, 16) |
+-----------------+-----------------+-----------------+
| 11110110110       | 3666              | 7B6               |
+-----------------+-----------------+-----------------+
```

```
SELECT CONV(11110110110, 2, 10), CONV(3666, 8, 10), CONV('7B6', 16, 10);
```

```
+-----------------------+-----------------+-----------------+
| CONV(11110110110, 2, 10) | CONV(3666, 8, 10) | CONV('7B6', 16, 10) |
+-----------------------+-----------------+-----------------+
| 1974                  | 1974              | 1974              |
+-----------------------+-----------------+-----------------+
```

Numeric functions - Examples

```
SELECT 12 MOD 5, MOD(12, 5);
```

```
+----------+------------+
| 12 MOD 5 | MOD(12, 5) |
+----------+------------+
|        2 |          2 |
+----------+------------+
```

```
SELECT RAND(), RAND(), RAND();
```

```
+-------------------+-------------------+-------------------+
| RAND()            | RAND()            | RAND()            |
+-------------------+-------------------+-------------------+
| 0.6117330664878088 | 0.653700472464506 | 0.43330319359274877 |
+-------------------+-------------------+-------------------+
```

```
SELECT POW(2, 10), POWER(2, 10);
```

```
+-----------+-------------+
| POW(2, 10) | POWER(2, 10) |
+-----------+-------------+
|      1024 |        1024 |
+-----------+-------------+
```

Miscellaneous functions

Function	Description
CURRENT_USER() CURRENT_USER SESSION_USER() SYSTEM_USER() USER()	The user name and host name provided by the client
DATABASE() SCHEMA()	Return the default (current) database name
VERSION()	Returns a string that indicates the MySQL server version
DEFAULT()	Return the default value for a table column
ENCODE(), DECODE()	Encode/Decode a string
MD5()	Calculate MD5 checksum
PASSWORD()	Calculate and return a password string (one-way)

Complete reference: http://dev.mysql.com/doc/refman/5.1/en/mathematical-functions.html

Miscellaneous functions - Examples

```
SELECT USER(), DATABASE(), VERSION();
```

```
+----------------+------------+-----------+
| USER()         | DATABASE() | VERSION() |
+----------------+------------+-----------+
| root@localhost | _prep      | 5.5.15    |
+----------------+------------+-----------+
```

```
SELECT * FROM PERSONS;
```

```
+----+--------+-----------+
| ID | NAME   | CITY      |
+----+--------+-----------+
|  1 | VINOD  | BANGALORE |
|  2 | SCOTT  | DALLAS    |
|  3 | BANU   | BANGALORE |
|  4 | MILLER | CHICAGO   |
+----+--------+-----------+
```

```
SELECT * FROM PERSONS
WHERE CITY = DEFAULT(CITY);
```

```
+----+-------+-----------+
| ID | NAME  | CITY      |
+----+-------+-----------+
|  1 | VINOD | BANGALORE |
|  3 | BANU  | BANGALORE |
+----+-------+-----------+
```

```
SELECT * FROM USERS;
```

```
+----------+-------------------------------------------+
| USERNAME | PASSWD                                    |
+----------+-------------------------------------------+
| VINOD    | *9D2D1415F6A0D138D693228417BA3ED2B7D06C9E |
| SCOTT    | *746FC1265EFA3E5BC598A1F2122C3377D74E933D |
| ROOT     | *B481E6BC13CBA73BC4C2B05F6F1393D16F2EE840 |
+----------+-------------------------------------------+
```

```
SELECT USERNAME FROM USERS
WHERE PASSWD = PASSWORD('SECRET');
```

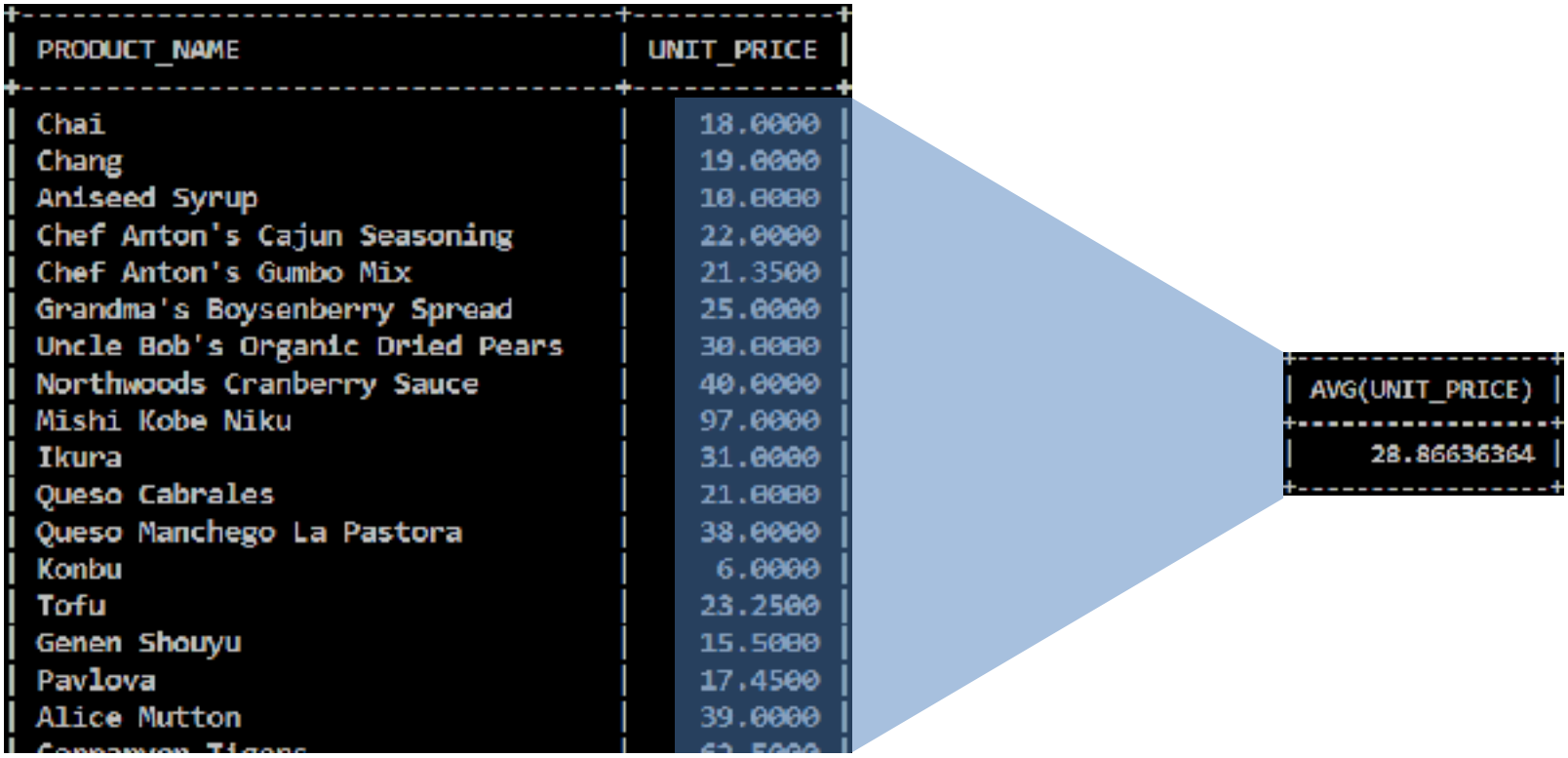
```
+----------+
| USERNAME |
+----------+
| VINOD    |
+----------+
```

Aggregating data

Using Group Functions

Group Functions

- Group functions operate on sets of rows to give one result per group.



PRODUCT_NAME	UNIT_PRICE
Chai	18.0000
Chang	19.0000
Aniseed Syrup	10.0000
Chef Anton's Cajun Seasoning	22.0000
Chef Anton's Gumbo Mix	21.3500
Grandma's Boysenberry Spread	25.0000
Uncle Bob's Organic Dried Pears	30.0000
Northwoods Cranberry Sauce	40.0000
Mishi Kobe Niku	97.0000
Ikura	31.0000
Queso Cabrales	21.0000
Queso Manchego La Pastora	38.0000
Konbu	6.0000
Tofu	23.2500
Genen Shouyu	15.5000
Pavlova	17.4500
Alice Mutton	39.0000

AVG(UNIT_PRICE)
28.86636364

Group Functions

- Some of the most commonly used group functions:
  - AVG
  - COUNT
  - MAX
  - MIN
  - SUM

Group Functions

```
SELECT [column,] group_function(column), ...
FROM table
[WHERE condition]
[GROUP BY column [HAVING {condition}]]
[ORDER BY column];
```

Group Functions

```
SELECT MAX(UNIT_PRICE), AVG(UNIT_PRICE), MIN(UNIT_PRICE), MAX(UNIT_PRICE)
FROM PRODUCTS;
```

MAX(UNIT_PRICE)	AVG(UNIT_PRICE)	MIN(UNIT_PRICE)	MAX(UNIT_PRICE)
263.5000	28.86636364	2.5000	263.5000

You can use AVG and SUM for numeric data only,
where as, MIN and MAX can be used even on any data type.

```
SELECT MIN(HIRE_DATE), MAX(HIRE_DATE) FROM EMPLOYEES;
```

MIN(HIRE_DATE)	MAX(HIRE_DATE)
1992-04-01 00:00:00	1994-11-15 00:00:00

Group Functions

When used on strings, MIN and MAX compare the ASCII value of the characters

```
SELECT MIN(FIRST_NAME), MAX(FIRST_NAME)
FROM EMPLOYEES;
```

MIN(FIRST_NAME)	MAX(FIRST_NAME)
Andrew	Steven

Group Functions

- COUNT(*) returns the number of rows in a table.
- COUNT(expr) returns the number of rows with non-null values for the expr.
- COUNT(DISTINCT expr) returns the number of distinct non-null values of the expr.

```
SELECT REGION
FROM EMPLOYEES;
```

REGION
WA
WA
WA
WA
NULL
NULL
NULL
WA
NULL

```
SELECT COUNT(*), COUNT(REGION),
COUNT(DISTINCT REGION) FROM EMPLOYEES;
```

COUNT(*)	COUNT(REGION)	COUNT(DISTINCT REGION)
9	5	1

Group Functions

- Group functions ignore null values in the column.
- For example, the following statement would ignore any NULL values in the UNIT_PRICE column, and perform the calculation

  SELECT AVG(UNIT_PRICE) FROM PRODUCTS

- It is same as:
  SUM_OF_NON_NULL_VALUES / COUNT_OF_NON_NULL_VALUES

Group Functions

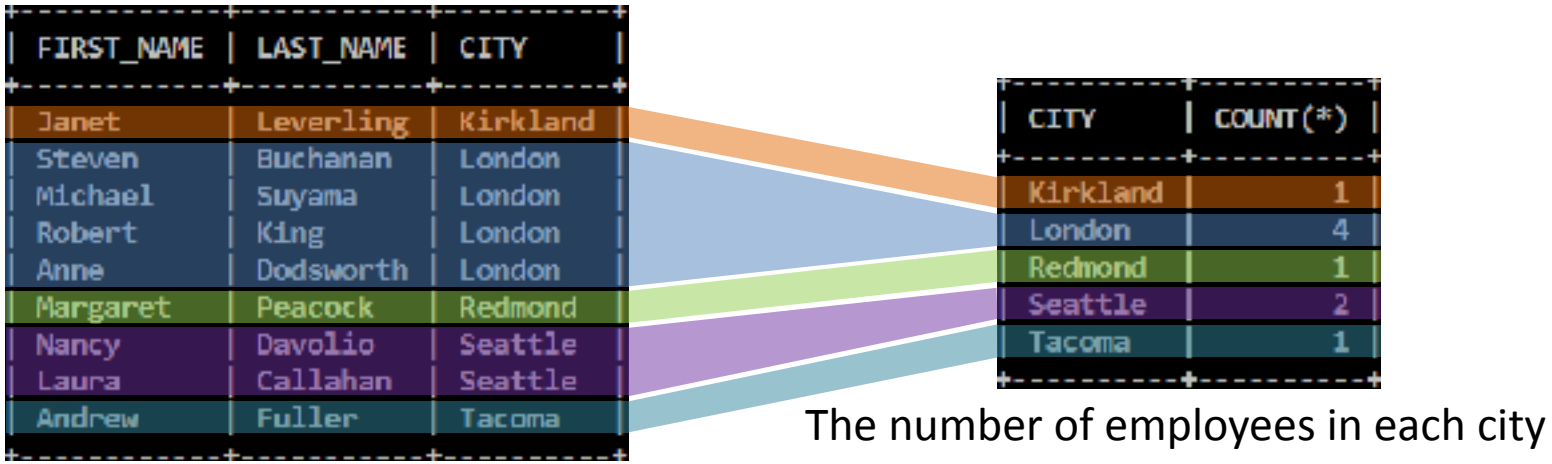
- If you don’t want to ignore the NULL values, you may use the IFNULL() function as shown below:

  SELECT AVG(IFNULL(UNIT_PRICE, 0)) FROM PRODUCTS;

Creating Groups Of Data

```
SELECT FIRST_NAME, LAST_NAME, CITY
FROM EMPLOYEES ORDER BY CITY;
```

```
SELECT CITY, COUNT(*)
FROM EMPLOYEES
GROUP BY CITY;
```



FIRST_NAME	LAST_NAME	CITY
Janet	Leverling	Kirkland
Steven	Buchanan	London
Michael	Suyama	London
Robert	King	London
Anne	Dodsworth	London
Margaret	Peacock	Redmond
Nancy	Davolio	Seattle
Laura	Callahan	Seattle
Andrew	Fuller	Tacoma

All employees

CITY	COUNT(*)
Kirkland	1
London	4
Redmond	1
Seattle	2
Tacoma	1

The number of employees in each city

Creating Groups Of Data

- Divide rows in a table into smaller groups by using the GROUP BY clause.

```
SELECT [column,] group_function(column), ...
FROM table
[WHERE condition]
[GROUP BY column [HAVING {condition}]]
[ORDER BY column];
```

Creating Groups Of Data

- All columns in the SELECT list that are not group functions must be in the GROUP BY clause.

```
SELECT FIRST_NAME, CITY, COUNT(*)
FROM EMPLOYEES GROUP BY CITY;
```

FIRST_NAME	CITY	COUNT(*)
Janet	Kirkland	1
Steven	London	4
Margaret	Redmond	1
Nancy	Seattle	2
Andrew	Tacoma	1

Invalid query. Unlike other DBMS, MySQL does not give error for this. Instead, it just picks the first employee name in each group and outputs.

```
SELECT CITY, COUNT(*)
FROM EMPLOYEES GROUP BY CITY;
```

CITY	COUNT(*)
Kirkland	1
London	4
Redmond	1
Seattle	2
Tacoma	1

Valid query.

Creating Groups Of Data

- The GROUP BY column does not have to be in the SELECT list.

```
SELECT COUNT(*) FROM PRODUCTS
GROUP BY CATEGORY_ID;
```

COUNT(*)
12
12
13
10
7
6
5
12

- However, such queries are very rarely used.

Grouping by More Than One Column

- Example – Total stock value of all products of a **particular category**, supplied by a **particular supplier**

```
SELECT CATEGORY_ID, SUPPLIER_ID, SUM(UNIT_PRICE * UNITS_IN_STOCK) AS INFO
FROM PRODUCTS GROUP BY CATEGORY_ID, SUPPLIER_ID;
```

ME	CATEGORY_ID	SUPPLIER_ID	UNIT_PRICE	UNITS_IN_STOCK
	1	1	18.0000	39
	1	1	19.0000	17
ger	1	7	15.0000	15
ant Kistica	1	10	4.5000	20
Klosterbier	1	12	7.7500	125
tout	1	16	18.0000	20
Ale	1	16	14.0000	111
umberjack Lager	1	16	14.0000	52
verte	1	18	18.0000	69
layo	1	18	263.5000	17
	1	20	46.0000	17
Hri	1	23	18.0000	57
rup	2	1	10.0000	13
Fiery Hot Pepper Sauce	2	2	21.0500	76
Hot Spiced Okra	2	2	17.0000	4
's Cajun Seasoning	2	2	22.0000	53
's Gumbo Mix	2	2	21.3500	0
Boysenberry Spread	2	3	25.0000	120

CATEGORY_ID	SUPPLIER_ID	INFO
1	1	1025.0000
1	7	225.0000
1	10	90.0000
1	12	968.7500
1	16	2642.0000
1	18	5721.5000
1	20	782.0000
1	23	1026.0000
2	1	130.0000
2	2	2833.8000
2	3	3240.0000
2	6	604.5000

Restricting the groups using HAVING clause

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

```
mysql> SELECT CATEGORY_ID, SUPPLIER_ID, SUM(UNIT_PRICE * UNITS_IN_STOCK)
    -> FROM PRODUCTS
    -> WHERE SUM(UNIT_PRICE*UNITS_IN_STOCK) >=2500
    -> GROUP BY CATEGORY_ID, SUPPLIER_ID;
ERROR 1111 (HY000): Invalid use of group function
```

Restricting the groups using HAVING clause

- You use the HAVING clause to restrict groups.

```
SELECT [column,] group_function(column), ...
FROM table
[WHERE condition]
[GROUP BY column [HAVING {condition}]]
[ORDER BY column];
```

```
SELECT CATEGORY_ID, SUPPLIER_ID, SUM(UNIT_PRICE * UNITS_IN_STOCK)
FROM PRODUCTS GROUP BY CATEGORY_ID, SUPPLIER_ID
HAVING SUM(UNIT_PRICE*UNITS_IN_STOCK) >=2500;
```

CATEGORY_ID	SUPPLIER_ID	SUM(UNIT_PRICE * UNITS_IN_STOCK)
1	16	2642.0000
1	18	5721.5000
2	2	2833.8000
2	3	3240.0000
2	29	3220.5000
3	8	3575.0000
3	11	3683.5500
4	5	3730.0000
4	28	4991.0000
5	9	2733.0000
6	4	2813.0000

Restricting the groups using HAVING clause

- You can use the alias in the HAVING clause
  - This may not work in other RDBMS

```
SELECT CATEGORY_ID, SUPPLIER_ID, SUM(UNIT_PRICE * UNITS_IN_STOCK) AS INFO
FROM PRODUCTS
GROUP BY CATEGORY_ID, SUPPLIER_ID
HAVING INFO >=2500;
```

CATEGORY_ID	SUPPLIER_ID	INFO
1	16	2642.0000
1	18	5721.5000
2	2	2833.8000
2	3	3240.0000
2	29	3220.5000
3	8	3575.0000
3	11	3683.5500
4	5	3730.0000
4	28	4991.0000
5	9	2733.0000
6	4	2813.0000
6	25	2916.4500

Displaying Data From Multiple Tables

SQL JOINS

PRORUCTS

PRODUCT_ID	PRODUCT_NAME	CATEGORY_ID
1	Chai	1
2	Chang	1
3	Aniseed Syrup	2
4	Chef Anton's Cajun Seasoning	2
5	Chef Anton's Gumbo Mix	2
6	Grandma's Boysenberry Spread	2
7	Uncle Bob's Organic Dried Pears	7
8	Northwoods Cranberry Sauce	2
9	Mishi Kobe Niku	6
10	Ikura	8
11	Queso Cabrales	4

CATEGORIES

CATEGORY_ID	CATEGORY_NAME	DESCRIPTIC
1	Beverages	Soft drink
2	Condiments	Sweet and
3	Confections	Desserts,
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, cr
6	Meat/Poultry	Prepared m
7	Produce	Dried frui
8	Seafood	Seaweed ar

PRODUCTS JOINED WITH CATEGORIES

PRODUCT_ID	PRODUCT_NAME	CATEGORY_NAME
1	Chai	Beverages
2	Chang	Beverages
3	Aniseed Syrup	Condiments
4	Chef Anton's Cajun Seasoning	Condiments
5	Chef Anton's Gumbo Mix	Condiments
6	Grandma's Boysenberry Spread	Condiments
7	Uncle Bob's Organic Dried Pears	Produce
8	Northwoods Cranberry Sauce	Condiments
9	Mishi Kobe Niku	Meat/Poultry
10	Ikura	Seafood
11	Queso Cabrales	Dairy Products

Cartesian Product

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
- All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition in a WHERE clause.

Cartesian Product

```
mysql> SELECT COUNT(*) FROM PRODUCTS;
+----------+
| COUNT(*) |
+----------+
|       77 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM CATEGORIES;
+----------+
| COUNT(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM PRODUCTS, CATEGORIES;
+----------+
| COUNT(*) |
+----------+
|      616 |
+----------+
1 row in set (0.00 sec)
```

77 Products X 8 Categories

= 616 Records

Cartesian Product

```
SELECT PRODUCT_NAME, CATEGORY_NAME
FROM PRODUCTS, CATEGORIES;
```

As you can see, each product is displayed with each categories, since there is no mapping between the product and the category it belongs to.

PRODUCT_NAME	CATEGORY_NAME
Chai	Beverages
Chai	Condiments
Chai	Confections
Chai	Dairy Products
Chai	Grains/Cereals
Chai	Meat/Poultry
Chai	Produce
Chai	Seafood
Chang	Beverages
Chang	Condiments
Chang	Confections
Chang	Dairy Products
Rh nbr nu Klosterbier	Seafood
Lakkalik ri	Beverages
Lakkalik ri	Condiments
Lakkalik ri	Confections
Lakkalik ri	Dairy Products
Lakkalik ri	Grains/Cereals
Lakkalik ri	Meat/Poultry
Lakkalik ri	Produce
Lakkalik ri	Seafood
Original Frankfurter gr ne So fe	Beverages
Original Frankfurter gr ne So fe	Condiments
Original Frankfurter gr ne So fe	Confections
Original Frankfurter gr ne So fe	Dairy Products
Original Frankfurter gr ne So fe	Grains/Cereals
Original Frankfurter gr ne So fe	Meat/Poultry
Original Frankfurter gr ne So fe	Produce
Original Frankfurter gr ne So fe	Seafood

616 rows in set (0.00 sec)

Avoiding Cartesian Product

- Use a join to query data from more than one table.

```
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1.column1 = table2.column2;
```

- Write the join condition in the WHERE clause.
- Prefix the column name with the table name when the same column name appears in more than one table.

```
SELECT PRODUCT_ID, PRODUCT_NAME, CATEGORY_NAME
FROM PRODUCTS, CATEGORIES
WHERE PRODUCTS.CATEGORY_ID = CATEGORIES.CATEGORY_ID;
```

```
SELECT PRODUCT_ID, PRODUCT_NAME, CATEGORY_NAME
FROM PRODUCTS AS P, CATEGORIES AS C
WHERE P.CATEGORY_ID = C.CATEGORY_ID;
```

What is Equi-Join?

- Foreign key of child table is compared with primary key of parent table using the “=” operator
  - WHERE PRODUCTS.CATEGORY_ID = CATEGORIES.CATEGORY_ID

PRODUCTS – CHILD TABLE

PRODUCT_ID	PRODUCT_NAME	CATEGORY_ID
1	Chai	1
2	Chang	1
3	Aniseed Syrup	2
4	Chef Anton's Cajun Seasoning	2
5	Chef Anton's Gumbo Mix	2
6	Grandma's Boysenberry Spread	2
7	Uncle Bob's Organic Dried Pears	7
8	Northwoods Cranberry Sauce	2
9	Mishi Kobe Niku	6
10	Ikura	8
11	Queso Cabrales	4

Foreign key

CATEGORIES – PARENT TABLE

CATEGORY_ID	CATEGORY_NAME	DESCRIPTIC
1	Beverages	Soft drink
2	Condiments	Sweet and
3	Confections	Desserts,
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, cr
6	Meat/Poultry	Prepared m
7	Produce	Dried frui
8	Seafood	Seaweed ar

Primary key

Using the ANSI JOIN syntax

- ANSI standardizes the way a join has to be performed using the following syntax:

```
SELECT column_list
FROM table_1 JOIN table_2 ON join_condition
```

- Use the WHERE clause only to filter the rows and not to do the JOIN operation.

```
SELECT column_list
FROM table_1 JOIN table_2 ON join_condition
WHERE filter_condition
```

- Non-Standard way of joining:

```
SELECT PRODUCT_ID, PRODUCT_NAME, CATEGORY_NAME
FROM PRODUCTS AS P, CATEGORIES AS C
WHERE P.CATEGORY_ID = C.CATEGORY_ID;
```

- ANSI style (standard):

```
SELECT PRODUCT_ID, PRODUCT_NAME, CATEGORY_NAME
FROM PRODUCTS AS P JOIN CATEGORIES AS C
ON P.CATEGORY_ID = C.CATEGORY_ID;
```

Joining more than two tables

- You can join any number of tables using the following syntax:

```
SELECT column_list
FROM table_1 JOIN table_2 ON join_condition
JOIN table_3 ON join_condition
JOIN table_3 ON join_condition
JOIN table_3 ON join_condition
...
```

Joining more than two tables - Example

- Retrieve the name of the product along with its category name and supplier name for all products

```
SELECT PRODUCT_NAME, CATEGORY_NAME, COMPANY_NAME
FROM PRODUCTS AS P JOIN CATEGORIES AS C ON P.CATEGORY_ID = C.CATEGORY_ID
JOIN SUPPLIERS AS S ON P.SUPPLIER_ID = S.SUPPLIER_ID;
```

PRODUCT_NAME	CATEGORY_NAME	COMPANY_NAME
Chai	Beverages	Exotic Liquids
Chang	Beverages	Exotic Liquids
Guaran├í Fant├ística	Beverages	Refrescos Americanas LTDA
Sasquatch Ale	Beverages	Bigfoot Breweries
Steeleye Stout	Beverages	Bigfoot Breweries
C├┤te de Blaye	Beverages	Aux joyeux eccl├⌐siastiques
Chartreuse verte	Beverages	Aux joyeux eccl├⌐siastiques
Ipoh Coffee	Beverages	Leka Trading
Laughing Lumberjack Lager	Beverages	Bigfoot Breweries
Outback Lager	Beverages	Pavlova, Ltd.
Rh├╢nbr├╢u Klosterbier	Beverages	Plutzer Lebensmittelgro├ƒm├ñrkte AG
Lakkalik├╢├╢ri	Beverages	Karkki Oy
Aniseed Syrup	Condiments	Exotic Liquids
Chef Anton's Cajun Seasoning	Condiments	New Orleans Cajun Delights
Chef Anton's Gumbo Mix	Condiments	New Orleans Cajun Delights
Grandma's Boysenberry Spread	Condiments	Grandma Kelly's Homestead

Joining more than two tables - Example

- Retrieve the name of the product along with its category name and supplier name for all products priced above $50

```
SELECT PRODUCT_NAME, CATEGORY_NAME, COMPANY_NAME, UNIT_PRICE
FROM PRODUCTS AS P JOIN CATEGORIES AS C ON P.CATEGORY_ID = C.CATEGORY_ID
JOIN SUPPLIERS AS S ON P.SUPPLIER_ID = S.SUPPLIER_ID
WHERE UNIT_PRICE > 50;
```

PRODUCT_NAME	CATEGORY_NAME	COMPANY_NAME	UNIT_PRICE
C┼te de Blaye	Beverages	Aux joyeux eccl├-siastiques	263.5000
Sir Rodney's Marmalade	Confections	Specialty Biscuits, Ltd.	81.0000
Raclette Courdavault	Dairy Products	Gai p├ôturage	55.0000
Mishi Kobe Niku	Meat/Poultry	Tokyo Traders	97.0000
Th╝ringer Rostbratwurst	Meat/Poultry	Plutzer Lebensmittelgro├fm├ñrkte AG	123.7900
Manjimup Dried Apples	Produce	G'day, Mate	53.0000
Carnarvon Tigers	Seafood	Pavlova, Ltd.	62.5000

Outer Join

- Consider the tables EMPS and DEPTS

EMPS

EMPNO	ENAME	DEPTNO
1	SCOTT	1
2	MILLER	2
3	ALLEN	2
4	ADAMS	4
5	SMITH	4

DEPTNO	DNAME
1	ADMIN
2	ACCOUNTING
3	MARKETING
4	OPERATIONS
5	SALES

DEPTS

- As you may have observed, there are no employees in the departments ‘MARKETING’ and ‘SALES’
- The join statement does not get all the departments, since the join condition can not find match for DEPTNO 4 and 5

```
SELECT ENAME, DNAME
FROM EMPS JOIN DEPTS
ON EMPS.DEPTNO = DEPTS.DEPTNO;
```

ENAME	DNAME
SCOTT	ADMIN
MILLER	ACCOUNTING
ALLEN	ACCOUNTING
ADAMS	OPERATIONS
SMITH	OPERATIONS

Outer Join

- This default behavior is called INNER join

```
SELECT ENAME, DNAME
FROM EMPS JOIN DEPTS
ON EMPS.DEPTNO = DEPTS.DEPTNO;
```

Same As

```
SELECT ENAME, DNAME
FROM EMPS INNER JOIN DEPTS
ON EMPS.DEPTNO = DEPTS.DEPTNO;
```

- If you want those rows from either of the tables that do not have a corresponding row in the other table, then use OUTER JOIN.

```
SELECT ENAME, DNAME
FROM EMPS RIGHT OUTER JOIN DEPTS
ON EMPS.DEPTNO = DEPTS.DEPTNO;
```

Note that DEPTS table, which has excess rows, appears on the RIGHTSIDE of the JOIN keyword

or

```
SELECT ENAME, DNAME
FROM DEPTS LEFT OUTER JOIN EMPS
ON DEPTS.DEPTNO = EMPS.DEPTNO;
```

Note that DEPTS table, which has excess rows, appears on the LEFTSIDE of the JOIN keyword

Result of OUTER join

ENAME	DNAME
SCOTT	ADMIN
MILLER	ACCOUNTING
ALLEN	ACCOUNTING
NULL	MARKETING
ADAMS	OPERATIONS
SMITH	OPERATIONS
NULL	SALES

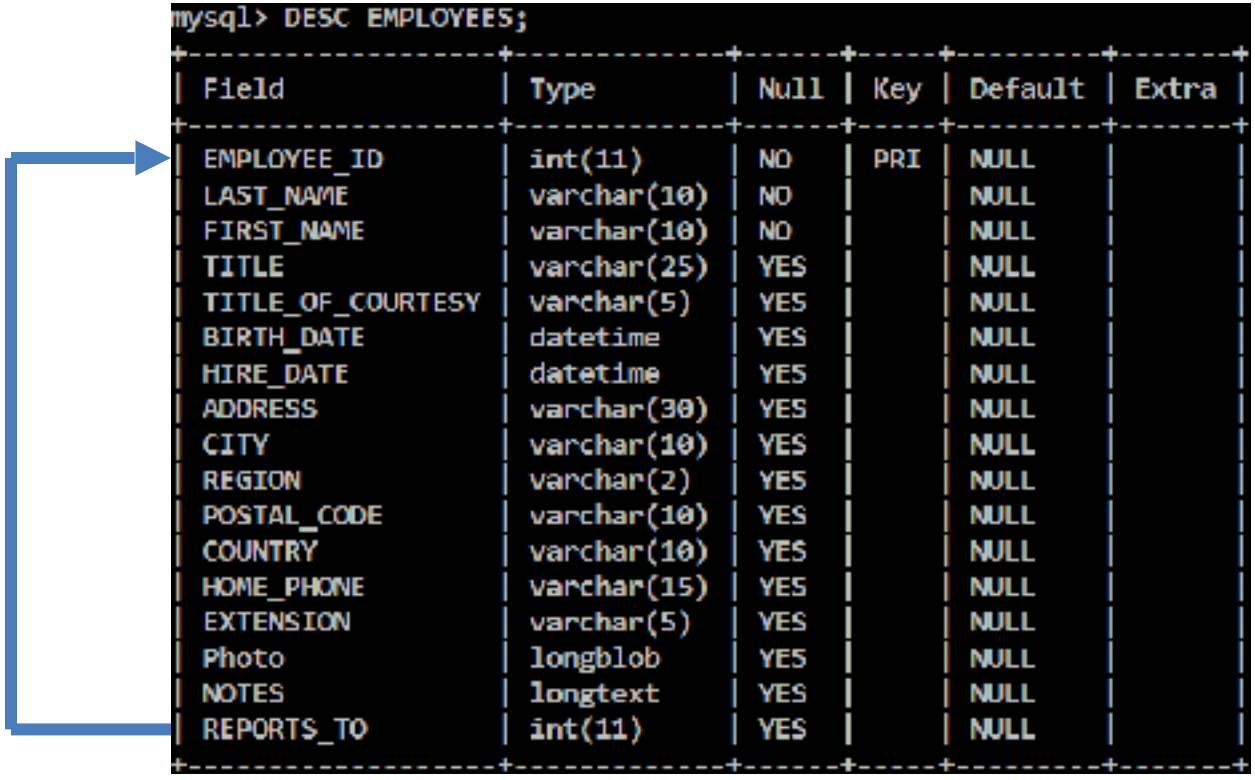
- You can replace the NULL values using IFNULL() function

```
SELECT IFNULL(ENAME, '--N/A--') AS ENAME, DNAME
FROM EMPS RIGHT OUTER JOIN DEPTS
ON EMPS.DEPTNO = DEPTS.DEPTNO;
```

ENAME	DNAME
SCOTT	ADMIN
MILLER	ACCOUNTING
ALLEN	ACCOUNTING
--N/A--	MARKETING
ADAMS	OPERATIONS
SMITH	OPERATIONS
--N/A--	SALES

Self Join

- If a table has a self referencing foreign key, then self join can be used to get relevant data



```
mysql> DESC EMPLOYEES;
+------------------+------------+------+-----+---------+-------+
| Field            | Type       | Null | Key | Default | Extra |
+------------------+------------+------+-----+---------+-------+
| EMPLOYEE_ID      | int(11)    | NO   | PRI | NULL    |       |
| LAST_NAME        | varchar(10)| NO   |     | NULL    |       |
| FIRST_NAME       | varchar(10)| NO   |     | NULL    |       |
| TITLE            | varchar(25)| YES  |     | NULL    |       |
| TITLE_OF_COURTESY| varchar(5) | YES  |     | NULL    |       |
| BIRTH_DATE       | datetime   | YES  |     | NULL    |       |
| HIRE_DATE        | datetime   | YES  |     | NULL    |       |
| ADDRESS          | varchar(30)| YES  |     | NULL    |       |
| CITY             | varchar(10)| YES  |     | NULL    |       |
| REGION           | varchar(2) | YES  |     | NULL    |       |
| POSTAL_CODE      | varchar(10)| YES  |     | NULL    |       |
| COUNTRY          | varchar(10)| YES  |     | NULL    |       |
| HOME_PHONE       | varchar(15)| YES  |     | NULL    |       |
| EXTENSION        | varchar(5) | YES  |     | NULL    |       |
| Photo            | longblob   | YES  |     | NULL    |       |
| NOTES            | longtext   | YES  |     | NULL    |       |
| REPORTS_TO       | int(11)    | YES  |     | NULL    |       |
+------------------+------------+------+-----+---------+-------+
```

Self Join

EMPLOYEES (AS EMPS)

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	REPORTS_TO
1	Nancy	Davolio	2
2	Andrew	Fuller	NULL
3	Janet	Leverling	2
4	Margaret	Peacock	2
5	Steven	Buchanan	2
6	Michael	Suyama	5
7	Robert	King	5
8	Laura	Callahan	2
9	Anne	Dodsworth	5

EMPLOYEES (AS MANAGERS)

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
1	Nancy	Davolio
2	Andrew	Fuller
3	Janet	Leverling
4	Margaret	Peacock
5	Steven	Buchanan
6	Michael	Suyama
7	Robert	King
8	Laura	Callahan
9	Anne	Dodsworth

- From the above data, we can observe that Nancy(1), Janet(3), Margaret(4), Steven(5), and Laura(8) report to Andrew(2). Also, Michael(6), Robert(7), and Anne(9) report to Steven(5).

Self Join

EMPLOYEES (AS EMPS)

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	REPORTS_TO
1	Nancy	Davolio	2
2	Andrew	Fuller	NULL
3	Janet	Leverling	2
4	Margaret	Peacock	2
5	Steven	Buchanan	2
6	Michael	Suyama	5
7	Robert	King	5
8	Laura	Callahan	2
9	Anne	Dodsworth	5

EMPLOYEES (AS MANAGERS)

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
1	Nancy	Davolio
2	Andrew	Fuller
3	Janet	Leverling
4	Margaret	Peacock
5	Steven	Buchanan
6	Michael	Suyama
7	Robert	King
8	Laura	Callahan
9	Anne	Dodsworth

FIRST_NAME (EMPLOYEE)	REPORTS_TO / EMPLOYEE_ID	FIRST_NAME (MANAGER)
Nancy Janet Margaret Steven Laura	2	Andrew
Michael Robert Anne	5	Steven

```
SELECT EMPS.FIRST_NAME, MANAGERS.FIRST_NAME
FROM EMPLOYEES AS EMPS JOIN EMPLOYEES AS MANAGERS
ON EMPS.REPORTS_TO = MANAGERS.EMPLOYEE_ID;
```

Self Join - Examples

- Retrieve the PRODUCT_NAME, UNIT_PRICE of all the products that are costlier than the product with PRODUCT_ID 62

```
SELECT P1.PRODUCT_NAME, P1.UNIT_PRICE
FROM PRODUCTS P1 JOIN PRODUCTS P2
ON P1.UNIT_PRICE > P2.UNIT_PRICE
WHERE P2.PRODUCT_ID = 62;
```

PRODUCT_NAME	UNIT_PRICE
Mishi Kobe Niku	97.0000
Carnarvon Tigers	62.5000
Sir Rodney's Marmalade	81.0000
Th╝ringer Rostbratwurst	123.7900
C┼te de Blaye	263.5000
Manjimup Dried Apples	53.0000
Raclette Courdavault	55.0000

Self Join - Examples

- Retrieve the EMPLOYEE_ID, FIRST_NAME, LAST_NAME, TITLE of all employees who are from the same city as the employee with FIRST_NAME ‘STEVEN’

```
SELECT E1.EMPLOYEE_ID, E1.FIRST_NAME, E1.LAST_NAME, E1.TITLE
FROM EMPLOYEES E1 JOIN EMPLOYEES E2
ON E1.CITY = E2.CITY
WHERE E2.FIRST_NAME = 'STEVEN';
```

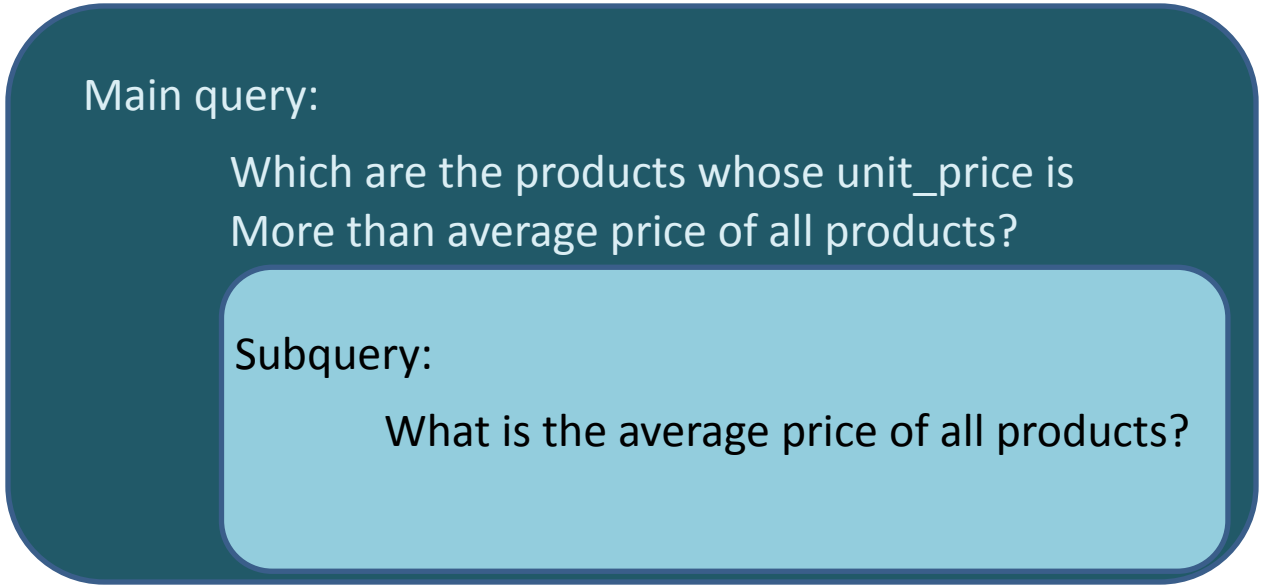
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	TITLE
5	Steven	Buchanan	Sales Manager
6	Michael	Suyama	Sales Representative
7	Robert	King	Sales Representative
9	Anne	Dodsworth	Sales Representative

Using

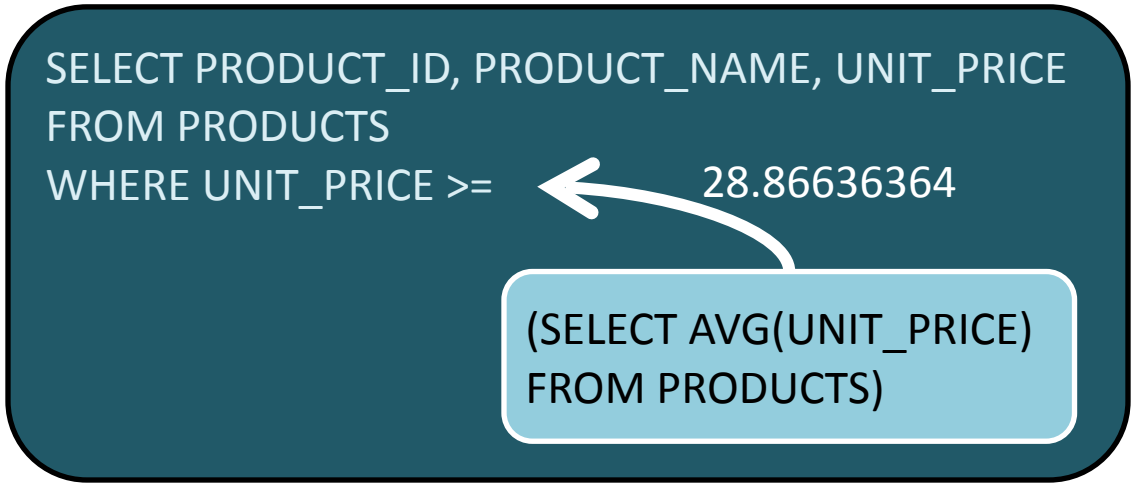
Subqueries

Using a subquery to solve a problem

- Which are the products costlier than average price of all products?



Using a subquery to solve a problem



- A subquery is a SELECT statement embedded in the WHERE cluase of another SELECT statement
- The subquery is executed once before the main query is executed
- The result of the subquery is substitutes the subquery and is used by the main query for every row

Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition.
- The ORDER BY clause in the subquery is not needed unless you are performing Top-N analysis
- Use single-row operators with single-row subqueries and use multiple-row operators with multiple-row subqueries.

Multi-row subqueries

- If we have to retrieve all the products that belong to those category of products supplied by a supplier with SUPPLIER_ID 6
  - Subquery:
    Get the category ids from products table where supplier id is 6:

```
SELECT DISTINCT CATEGORY_ID
FROM PRODUCTS
WHERE SUPPLIER_ID=6;
```

  - This may result in the category ids – 2, 7, 8
  - This type of subqueries are known as multi-row subqueries, and an outer query (or the main query) must use ‘IN’ or ‘NOT IN’ to compare them

```
SELECT PRODUCT_ID, PRODUCT_NAME, CATEGORY_ID
FROM PRODUCTS WHERE CATEGORY_ID IN
(SELECT DISTINCT CATEGORY_ID FROM PRODUCTS WHERE SUPPLIER_ID=6)
ORDER BY CATEGORY_ID;
```

Multi-row subqueries

- Using the equals operator to do the same would result in an error
  - One CATEGORY_ID from each row can not be compared with a bunch of CATEGORY_ID values given by the subquery

```
mysql> SELECT PRODUCT_ID, PRODUCT_NAME, CATEGORY_ID
    -> FROM PRODUCTS WHERE CATEGORY_ID =
    -> (SELECT DISTINCT CATEGORY_ID FROM PRODUCTS WHERE SUPPLIER_ID=6)
    -> ORDER BY CATEGORY_ID;
ERROR 1242 (21000): Subquery returns more than 1 row
```

  - Alternatively, you can use
    = ANY
    operator to do the same

```
SELECT PRODUCT_ID, PRODUCT_NAME, CATEGORY_ID
FROM PRODUCTS WHERE CATEGORY_ID = ANY
(SELECT DISTINCT CATEGORY_ID FROM PRODUCTS WHERE SUPPLIER_ID=6)
ORDER BY CATEGORY_ID;
```

Multi-Column Subqueries

- Get the details of the products that belong to the same category and are supplied by the same supplier as that of the product with id ‘6’

PRODUCT_ID = 6

SUPPLIER_ID	CATEGORY_ID
3	2

- The outer/main query can use two columns together to compare the result of the subquery

```
SELECT PRODUCT_ID, PRODUCT_NAME, CATEGORY_ID, SUPPLIER_ID
FROM PRODUCTS WHERE (SUPPLIER_ID, CATEGORY_ID) =
(SELECT SUPPLIER_ID, CATEGORY_ID FROM PRODUCTS WHERE PRODUCT_ID = 6);
```

PRODUCT_ID	PRODUCT_NAME	CATEGORY_ID	SUPPLIER_ID
6	Grandma's Boysenberry Spread	2	3
8	Northwoods Cranberry Sauce	2	3

Multi-Column Subqueries

```
SELECT PRODUCT_ID, PRODUCT_NAME, CATEGORY_ID, SUPPLIER_ID
FROM PRODUCTS WHERE (SUPPLIER_ID, CATEGORY_ID) =
(SELECT SUPPLIER_ID, CATEGORY_ID FROM PRODUCTS WHERE PRODUCT_ID = 6);
```

Using a Subquery in the FROM Clause

- Retrieve the CATEGORY_NAME, DESCRIPTION, NUMBER_OF_PRODUCTS, AVG_PRICE_OF_PRODUCT for each category in the CATEGORIES table

```
SELECT CATEGORY_NAME, DESCRIPTION, T1.NUMBER_OF_PRODUCTS, T1.AVG_PRICE
FROM CATEGORIES C1 JOIN
        (SELECT CATEGORY_ID, COUNT(*) NUMBER_OF_PRODUCTS,
        AVG(UNIT_PRICE) AVG_PRICE
        FROM PRODUCTS GROUP BY CATEGORY_ID) AS T1
ON C1.CATEGORY_ID = T1.CATEGORY_ID;
```

CATEGORY_NAME	DESCRIPTION	NUMBER_OF_PRODUCTS	AVG_PRICE
Beverages	Soft drinks, coffees, teas, beers, and ales	12	37.97915667
Condiments	Sweet and savory sauces, relishes, spreads, and seascnings	12	23.06250000
Confections	Desserts, candies, and sweet breads	13	25.16000000
Dairy Products	Cheeses	10	28.73000000
Grains/Cereals	Breads, crackers, pasta, and cereal	7	20.25000000
Meat/Poultry	Prepared meats	6	54.00666667
Produce	Dried fruit and bean curd	5	32.37000000
Seafood	Seaweed and fish	12	20.68250000

Note: This is only for illustration. The same can be done using a simple joining of CATEGORIES table with PRODUCTS table.

Correlated Subqueries

- A correlated subquery is a subquery that contains a reference to a table that also appears in the outer query

```
SELECT * FROM t1 WHERE column1 =
       (SELECT column1 FROM t2 WHERE t2.column2 = t1.column2);
```

  - Notice that the subquery contains a reference to a column of t1, even though the subquery's FROM clause does not mention a table t1.
  - So, MySQL looks outside the subquery, and finds t1 in the outer query.
  - MySQL evaluates the inner query for each of the row in the outer query, unlike the rest of the cases, where the inner query is evaluated only once for all the rows of the outer query.

Correlated Subqueries

- For certain cases, a correlated subquery is optimized
- Otherwise, they are inefficient and likely to be slow.
- Rewriting the query as a join might improve performance.

Correlated Subqueries - Example

- Retrieve the costliest product in each category. Display these fields: Category name, Product Name, Unit price

```
SELECT CATEGORY_NAME, PRODUCT_NAME, UNIT_PRICE
FROM CATEGORIES C JOIN PRODUCTS P
ON C.CATEGORY_ID = P.CATEGORY_ID
WHERE UNIT_PRICE =
       (SELECT MAX(UNIT_PRICE) FROM PRODUCTS
       WHERE CATEGORY_ID = P.CATEGORY_ID)
ORDER BY CATEGORY_NAME;
```

CATEGORY_NAME	PRODUCT_NAME	UNIT_PRICE
Beverages	C┼te de Blaye	263.5000
Condiments	Vegie-spread	43.9000
Confections	Sir Rodney's Marmalade	81.0000
Dairy Products	Raclette Courdavault	55.0000
Grains/Cereals	Gnocchi di nonna Alice	38.0000
Meat/Poultry	Th╝ringer Rostbratwurst	123.7900
Produce	Manjimup Dried Apples	53.0000
Seafood	Carnarvon Tigers	62.5000

Views

Database View

- A view consists of a stored query accessible as a virtual table in a relational database.
- Unlike ordinary tables (base tables) in a relational database, a view does not form part of the physical schema:
  - It is a dynamic, virtual table computed or collated from data in the database.
- Changing the data in a table alters the data shown in subsequent invocations of the view.

Advantages of a view (1 of 2)

- Views can represent a subset of the data contained in a table
- Views can join and simplify multiple tables into a single virtual table
- Views can act as aggregated tables, where the database engine aggregates data (sum, average etc.) and presents the calculated results as part of the data
- Views can hide the complexity of data; for example a view could appear as Sales2000 or Sales2001, transparently partitioning the actual underlying table

Advantages of a view (2 of 2)

- Views take very little space to store; the database contains only the definition of a view, not a copy of all the data it presents
- Depending on the SQL engine used, views can provide extra security
- Views can limit the degree of exposure of a table or tables to the outer world

Creating a view in MySQL

Syntax:

```
CREATE [OR REPLACE]
VIEW view_name [(column_list)]
AS select_statement
```

Example:

```
CREATE VIEW order_amounts_view AS
SELECT o.order_id,
format(sum(od.unit_price*od.quantity*(1-od.discount)),2)
AS order_amount
FROM orders as o JOIN order_details AS od
ON o.order_id = od.order_id
GROUP BY o.order_id;
```

Output from a view

```
mysql> select * from order_amounts_view limit 10;
+----------+--------------+
| order_id | order_amount |
+----------+--------------+
|    10248 | 440.00       |
|    10250 | 1,912.60     |
|    10251 | 654.06       |
|    10252 | 3,597.90     |
|    10253 | 1,444.80     |
|    10254 | 556.62       |
|    10255 | 2,490.50     |
|    10256 | 517.80       |
|    10258 | 1,614.88     |
|    10260 | 1,504.65     |
+----------+--------------+
10 rows in set (0.08 sec)
```

Database Index

Database index

- A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of slower writes and increased storage space.
- Indices can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.
- Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows.

Database index

Syntax:

```
CREATE INDEX index_name
ON tbl_name (index_col_name,...)
```

Example:

```
CREATE INDEX idx_products_unit_price
ON products (unit_price);
```

To delete the index, use the DROP INDEX command:

```
DROP INDEX idx_products_unit_price ON products;
```

Database index

- In MySQL, the following are automatically indexed:
  - Primary key column
  - Unique column
  - Fulltext column