MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Syllabus

- **Introduction to SQL**: Characteristics and advantages of SQL, SQL Data Types
- **DDL Commands, DCL Commands.**
- **SQL Queries**: DML Queries with Select Query Clauses, Creating, Modifying, Deleting.
- **Views**: Creating, Dropping, Updating, Indexes,
- Set Operations, Predicates and Joins, Set membership, Grouping and Aggregation, Aggregate Functions, Nested Queries

Characteristics of SQL

- SQL stands for Structured Query Language
- SQL is an ANSI and ISO standard computer language for creating and manipulating databases.
- SQL allows the user to create, update, delete, and retrieve data from a database.
- SQL is very simple and easy to learn.
- SQL works with database programs like DB2, Oracle, MS Access, Sybase, MS SQL Sever etc.
- SQL is a declarative language, not a procedural language.
- All keywords of SQL are case insensitive.

Advantages of SQL

- **High Speed:** SQL Queries can be used to retrieve large amounts of records from a database quickly and efficiently.
- **Well Defined Standards Exist:** SQL databases use long-established standard, which is being adopted by ANSI & ISO. Non-SQL databases do not adhere to any clear standard.
- **No Coding Required:** Using standard SQL it is easier to manage database systems without having to write substantial amount of code.
- **Easy to learn and understand**
- **Portable:** SQL can be run on any platform, Databases using SQL can be moved from a device to another without any problems.

SQL Data Types and Literals

char(n). Fixed length character string, with user-specified length *n*.

varchar(n). Variable length character strings, with user-specified maximum length *n*.

Boolean. Accepts value true or false.

int. Integer (a finite subset of the integers that is machine-dependent).

smallint. Small integer (a machine-dependent subset of the integer domain type).

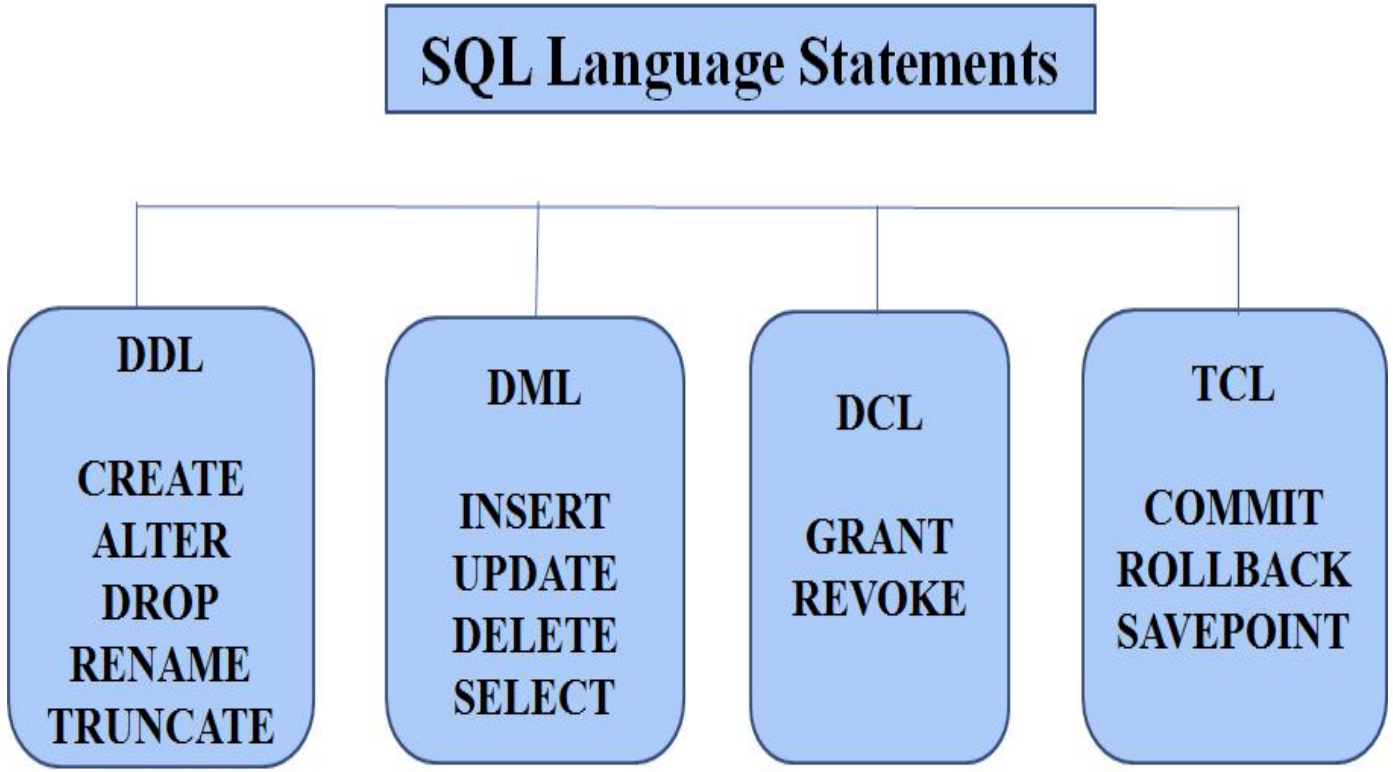
decimal(p,d). Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point. (ex., **decimal(3,1)**, allows 44.5 to be stored exactly, but not 444.5 or 0.32)

Double(p,d). Floating point and double-precision floating point numbers, with machine-dependent precision. Decimal precision can go to 53 places for a DOUBLE.

float(p,d). Floating point number, with user-specified precision of at least *n* digits. Decimal precision can go to 24 places for a FLOAT.

SQL language statements

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Transaction Control Language (TCL)



Data Definition Language (DDL)

- The SQL data-definition language (DDL) allows
  - Database tables to be created or deleted
  - Define indexes (keys)
  - Specify links between tables
  - Impose Integrity constraints between database tables

Some of the most commonly used DDL statements in SQL are

- **CREATE TABLE** : creates a new database tables
- **ALTER TABLE** : Alters(changes) a database tables
- **DROP TABLE** : Deletes a database table
- **RENAME TABLE** : Renames a database table
- **TRUNCATE TABLE :** Deletes all the records in the table.

Create Table Construct

- An SQL relation is defined using the **create table** command:

create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
(integrity-constraint$_1$),
...,
(integrity-constraint$_k$));

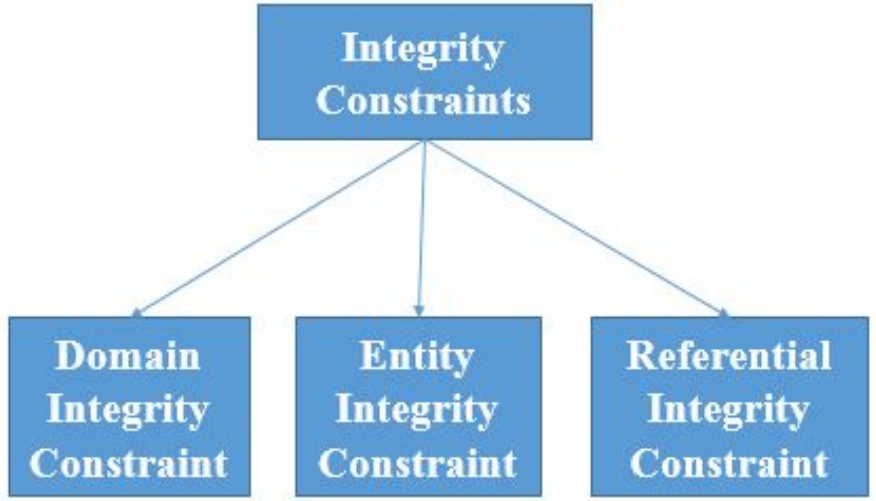
Example:

create table *instructor* (
ID **char**(5),
n*ame* **varchar**(20),
dept_name **varchar**(20),
salary **decimal**(8,2));

ID	*name*	*dept_name*	*salary*

Integrity Constraints

- Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.
- Constraints could be either column level or table level.
  1. **Column Level**: Column level constraints are applied to only one column.
  2. **Table Level**: Table level constraints are applied to the whole table.
- There are 4 types of Integrity Constraints:



Domain Integrity Constraints

- Domain Integrity constraints can be defined as the definition of a valid set of values for an attribute.
  1. **NOT NULL Constraint**:
  2. **Unique Constraint** :
  3. **Default Constraint** :
  4. **Check Constraint** :

1. NOT NULL :

- Ensures that a column cannot have NULL value.
- E.g. Roll_no int *not null*,
  Name varchar(20)

NULL value is not allowed

Roll_No	Name
1	ABC
2	XYZ
3	

Domain Integrity Constraints (Cont..)

2. Unique Constraint :

- Ensures that all values in a column are different.
- E.g. Emp_ID varchar(20) not null unique,

Not allowed as Emp_ID has unique constraint

Emp_ ID	Name	Salary
E101	ABC	20000
E102	XYZ	20000
E102	PQR	18000

3. Default Constraint:

- Provides a default value for a column when none is specified.
- **E.g. Marks int default NULL,**

Roll_No	Name	Marks
1	ABC	NULL
2	XYZ	NULL

Domain Integrity Constraints (Cont..)

4. Check Constraint:

- The CHECK constraint ensures that all the values in a column satisfies certain conditions.

```
CREATE TABLE student (
Roll_No int NOT NULL,
Name varchar(255) NOT NULL,
Age int CHECK (Age>=18)
);
```

Roll_No	Name	Age
1	ABC	18
2	XYZ	20
3	PQR	25
4	MNP	10

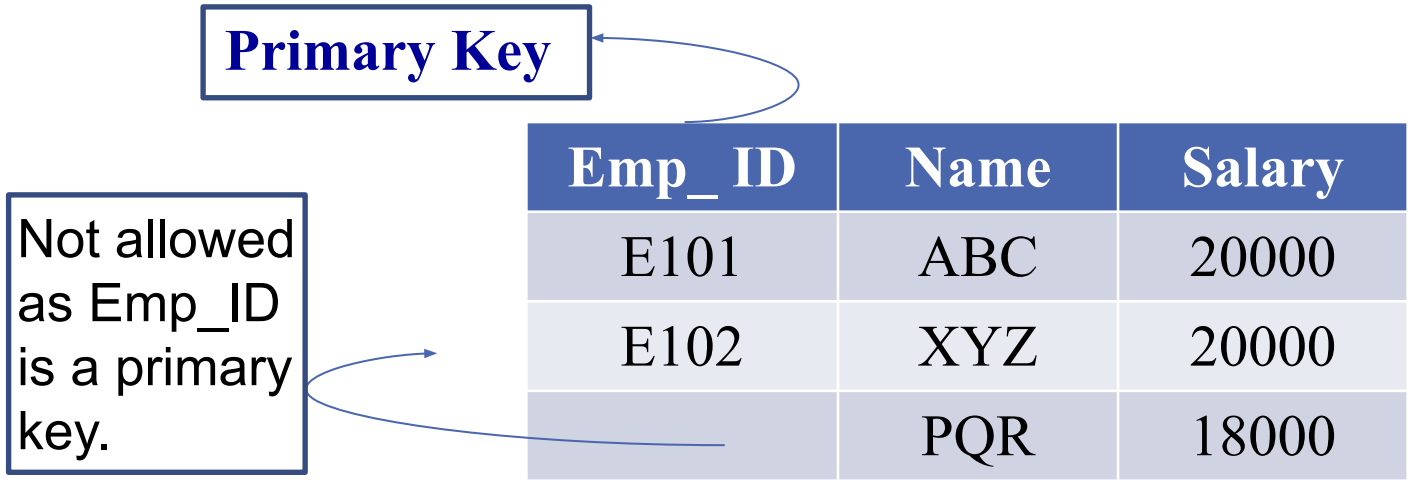
Domain Constraint

(Age>=18)

Not Allowed

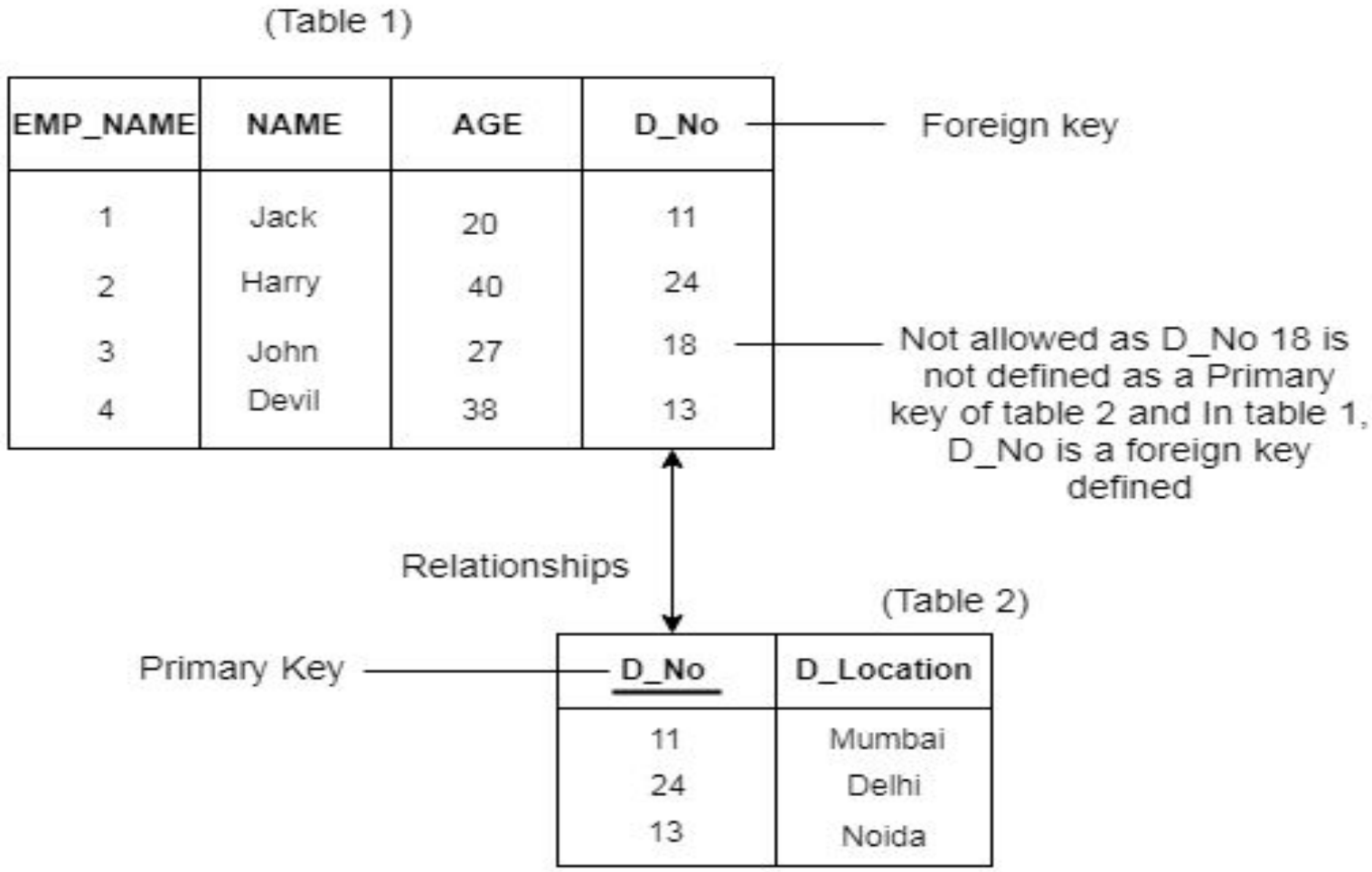
Entity Integrity Constraints

- **Primary Key constraint:**
  - states that primary key value can't be null.
  - Because primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
  - A table can contain a null value other than the primary key field.



Referential Integrity Constraints

- **Foreign Key constraint:**
  - A foreign key is a key used to link two tables together.
  - A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
  - The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.



Referential Integrity Constraints(Cont..)

- There are two type foreign key integrity constraints:
  1. cascade delete
  2. cascade update

1. Cascade Delete :

A foreign key with cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted.

Syntax:

```
CREATE TABLE child_table(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ], ...
  CONSTRAINT fk_name
  FOREIGN KEY (child_col1, child_col2, ...
child_col_n)
   REFERENCES parent_table (parent_col1, parent_col2,
...   parent_col_n)
    ON DELETE CASCADE
    [ ON DELETE { NO ACTION | CASCADE | SET
NULL | SET DEFAULT } ] );
```

Referential Integrity Constraints(Cont..)

2. Cascade Update :

A foreign key with cascade update means that if a record in the parent table is updates, then the corresponding records in the child table will automatically be updated.

- **DROP a FOREIGN KEY Constraint**

***ALTER TABLE** ORDERS*
***DROP** FOREIGN KEY;*

Syntax:

```
CREATE TABLE child_table(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ], ...
  CONSTRAINT fk_name
  FOREIGN KEY (child_col1, child_col2, ... child_col_n)
   REFERENCES parent_table (parent_col1, parent_col2,
... parent_col_n)
   ON UPDATE CASCADE
   [ ON UPDATE { NO ACTION | CASCADE | SET
NULL | SET DEFAULT } ] );
```

Integrity Constraints in Create Table

- **not null**
- **primary key** (A_1, ..., A_n)
- **Foreign key** (A_m, ..., A_n) **references** *r*

Example:

create table *instructor* (
ID **char**(5),
name **varchar**(20) **not null,**
dept_name **varchar**(20),
salary **numeric**(8,2),
primary key (*ID*),
foreign key *(dept_name*) **references** *department);*

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

- **primary key** declaration on an attribute automatically ensures **not null.**

Alter Command

Alter command is used for altering the table structure, such as,

1. to add a column to existing table
2. to rename any existing column
3. to change datatype of any column or to modify its size.
4. to drop a column from the table.

2. To add a column

alter table <table name> add <column name> datatype

- All exiting tuples in the relation are assigned *null* as the value for the new attribute, if default value is not specified.
- E.g. ***ALTER TABLE*** *Customers*
  ADD *Email varchar(255);*

Alter Command (Cont..)

- **By setting default value for new column**

```
ALTER TABLE table_name  ADD(column-name1 datatype1
      DEFAULT some_value);
```

E.g. ***ALTER TABLE student ADD(***
dob DATE DEFAULT '01-Jan-99');

2.To modify a column

```
ALTER TABLE table_name modify Column(
   column_name datatype);
```

Alter Command (Cont..)

E.g. *ALTER TABLE student MODIFY Column(address varchar(300));*

3. To Rename a Column

***ALTER TABLE** table_name **RENAME Column** old_column_name **TO** new_column_name;*

E.g. ALTER TABLE student RENAME column address TO location;

Alter Command (Cont..)

4. To drop a column

- **Dropping of attributes not supported by many databases.**

ALTER TABLE table_name DROP Column(column_name);

- *E.g.* ***ALTER TABLE*** *Customers*
  DROP COLUMN *Email;*

Drop Command

DROP TABLE command is used to drop an existing table in a database.

DROP TABLE *table_name;*

E.g. ***DROP TABLE Customers;***

Rename Command

RENAME command is used to rename a table.

***RENAME TABLE** {tbl_name} **TO** {new_tbl_name};*

E.g. ***RENAME TABLE Customers TO Customers _new;***

Truncate Command

TRUNCATE TABLE command is used to delete complete data from an existing table.

TRUNCATE TABLE *table_name;*

E.g. ***TRUNCATE TABLE Customers;***

Data Control Language (DCL)

DCL commands control the level of access that users have on database objects.

1) GRANT – provides access privileges to the users on the database objects. The privileges could be select, delete, update and insert on the tables and views. On the procedures, functions and packages it gives select and execute privileges.

- In DCL we have two commands,

1. **GRANT**: Used to provide any user access privileges or other privileges for the database.
2. **REVOKE**: Used to take back permissions from any user.

GRANT Command

- **Syntax for the GRANT command :**

  GRANT privilege_name ON object_name

  TO {user_name | PUBLIC | role_name} [with GRANT option];

- Allow User to create table:

  To allow a user to create tables in the database, we can use the below command,

  GRANT **CREATE TABLE** TO username;
- Grant Select privileges to user on customer table:

  GRANT **SELECT** ON **customer** TO username;
- Grant permission to drop any table:

  GRANT **DROP ANY TABLE** TO username;
- To GRANT ALL privileges to a user

  GRANT **ALL PRIVILEGES** ON database_name TO username

DCL Example

mysql> CREATE USER 'finley'@'localhost‘ IDENTIFIED BY 'password';

mysql> GRANT ALL ON *.* TO 'finley'@'localhost’; (Databasename.tablename)

mysql> SHOW GRANTS FOR 'finley'@'localhost';

from cmd prompt change to folder

C:\Program Files\MySQL\MySQL Server 8.0\bin> mysql -u finley -p

Enter password: ******** (password)

mysql> create database a;

mysql> use a;

mysql> create table abc(a1 int);

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Continued…….

From Root

```
mysql> REVOKE ALL ON *.* FROM 'finley'@'localhost';

mysql> REVOKE CREATE,DROP  ON *.*   FROM 'finley'@'localhost';
```

REVOKE Command

- **Syntax for the REVOKE command:**

  REVOKE privilege_name ON object_name

  FROM {User_name | PUBLIC | Role_name}

- To take back Permissions from user

  REVOKE CREATE TABLE FROM username;

- Revoke SELECT privilege on employee table from user1.

  REVOKE SELECT ON employee FROM user1;

Transaction Control Language (TCL)

- **TCL commands are used to** manage transactions in the database.
- These are used to manage the changes made to the data in a table by DML statements.

1) Commit
2) Rollback
3) Savepoint

TCL (Cont..)

1) Commit Command:

- used to permanently save any transaction into the database.
- When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.
- To avoid that, we use the COMMIT command to mark the changes as permanent.
- Syntax:

COMMIT;

TCL (Cont..)

2) ROLLBACK Command:

- restores the database to last committed state.
- Can be used to cancel the last update made to the database, if those changes are not committed using the COMMIT command.
- **Syntax:**

ROLLBACK TO savepoint_name;

3) SAVEPOINT command:

- used to temporarily save a transaction so that we can rollback to that point whenever required.
- Syntax:

SAVEPOINT savepoint_name;

Data Manipulation Language (DML)

DML commands are used to make modifications of the Database like,

- Deletion of tuples from a given relation.
- Insertion of new tuples into a given relation
- Updation of values in some tuples in a given relation

INSERT Query

- Add a new tuple to *course*

insert into *course*

values (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

- or equivalently

insert into *course* (*course_id*, *title*, *dept_name*, *credits*)

values (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

- Add a new tuple to *student* with *tot_creds* set to null

insert into *student*

values (’3003’, ’Green’, ’Finance’, *null*);

DELETE Query

- Delete all instructors from the Finance department

  delete from *instructor*
  where *dept_name*= ’Finance’;

- Delete all tuples in the *student* relation.

  delete from *student;*

UPDATE Query

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two **update** statements:

```
update instructor
    set salary = salary * 0.03
    where salary > 100000;

update instructor
    set salary = salary * 0.05
    where salary <= 100000;
```

  - Can be done better using the **case** statement (next slide)

Arithmetic Operations in SELECT Query

- The **select** clause can contain arithmetic expressions involving the operation, +, –, *, and /, and operating on constants or attributes of tuples.
  - The Query:

    select *ID, name, salary/12*

    from *instructor;*

    would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.
  - Can rename "*salary/12*" using the **as** clause:

    select *ID, name, salary/12* **as** *monthly_salary;*

SELECT Query

- The SELECT statement is used to select data from a database tables.

Select ------ ← Mandatory Clause
From
Where
Group by ← Clauses(Use as per need)
Having
Order by

E.g. ***SELECT * FROM** Student**;***

- The result of an SQL query is a relation.

SELECT Query (Cont..)

- An attribute can be a literal with **from** clause

  select 'A' from instructor

  - Result is a table with one column and *N* rows (number of tuples in the *instructors* table), each row with value "A".

SQL Functions

- Single Row Functions : Operates on each row and out for each row
  - Date Functions, String Functions, Number Functions, Conversion Functions
- Multirow Function : Aggregate Function/Group Row : Operates on Group of row, provide o/p per group

  Min, Max, Count, Sum, Avg etc
- SQL Single Row Functions can be used in Select Clause, Where Clause, Group By Clause, Order By clause
- SQL Multi Row Functions can be used in Select Clause, Group By Clause, Having Clause

String Function : Use in Select, Where, group by , having , order by Clause

Function	Meaning
Char_length(string)	Return number of characters in argument
Concat(expr1,expr2)	Return concatenated string
Instr(expr1,expr2)	Return the index of the first occurrence of substring
Lower(expr1)	Return the argument in lowercase
Left(expr1,count)	Return the leftmost number of characters from string
Lpad(expr1,length,expr2)	left-pads a string with another string, to a certain length
Ltrim()	Remove leading spaces
Substr(string,startpos,length)	extracts a substring from a string (starting at any position).
LOCATE(*substring*, *string*, *start*)	returns the position of the first occurrence of a substring in a string
STRCMP(*string1*, *string2*)	compares two strings. Returns 0,1,-1
Upper(string)	Convert the text to upper-case
Trim(string)	removes leading and trailing spaces from a string.

DATE Function : Use in Select, Where, group by having Clause, order by clause

Function	Meaning
ADDDATE(date, INTERVAL value addunit)	adds a time/date interval to a date and then returns the date
CURDATE() function	returns the current date. as "YYYY-MM-DD" (string)
DATEDIFF(date1, date2)	returns the number of days between two date values
DATE_SUB(date, INTERVAL value interval)	subtracts a time/date interval from a date and then returns the date
DAY(date)	returns the day of the month for a given date
DAYNAME(date)	returns the weekday name for a given date.
SYSDATE()	returns the current date and time.

The WHERE Clause

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

- Comparison results can be combined using the logical connectives **and, or,** and **not**
  - To find all instructors in Comp. Sci. dept with salary > 80000

```
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 80000
```

- Comparisons can be applied to results of arithmetic expressions.

The FROM Clause

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*

```
select *
from instructor, teaches;
```

  - ;generates every possible instructor – teaches pair, with all attributes from both relations.
  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

Renaming table in Select clause

- The SQL allows renaming relations and attributes using the **as** clause:

  old-name **as** *new-name*

- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.

  select distinct *T.name*
  from *instructor* **as** *T, instructor* **as** *S*
  where *T.salary* > *S.salary* **and** *S.dept_name* = *'Comp. Sci.'*

- Keyword **as** is optional and may be omitted

  instructor **as** *T* ≡ *instructor T*

Ordering the Display of Tuples

- List in alphabetic order the names of all instructors

```
select distinct name
from   instructor
order by name
```

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.

  Example: **order by** *name* **desc**

- Can sort on multiple attributes

  Example: **order by** *dept_name, name*

Where Clause Predicates

- SQL includes a **between AND** comparison operator
- Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, >= $90,000 and <= $100,000)

```
select name
from instructor
where salary between 90000 and 100000
```

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- The predicate **is null** can be used to check for null values.
  - Example: Find all instructors whose salary is null.

```
select name
from instructor
where salary is null
```

Views : Uses and Importance

- In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database.)
- Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

```
select ID, name, dept_name
from instructor
```

- A **view** provides a mechanism to hide certain data from the view of certain users thus providing security.
- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**.

View -Syntax

- A view is defined using the **create view** statement which has the form

  create view *v* **as** < query expression >

  view name — any legal SQL expression.

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- Can provide huge time savings in writing queries by already having a group of frequently accessed tables joined together in a view .

Example of Views

- A view of instructors without their salary

  create view *faculty* **as**
  select *ID*, *name*, *dept_name*
  from *instructor*

- Find all instructors in the Biology department

  select *name*
  from *faculty*
  where *dept_name* = 'Biology'

- Create a view of department salary totals

  create view *departments_total_salary*(*dept_name*, *total_salary*) **as**
  select *dept_name*, **sum** (*salary*)
  from *instructor*
  group by *dept_name*;

Inserting a new tuple into a View

- Add a new tuple to *faculty* view which we defined earlier

  insert into *faculty* **values** (’30765’, ’Green’, ’Music’);

- This insertion must be represented by the insertion of the tuple

  (’30765’, ’Green’, ’Music’, null)

  into the *instructor* relation

Update of a View

- Update query is used to Update the tuples of *a view.*

UPDATE faculty
set dept_name=“Biology”
where name=“ABC”

- Updation in view reflects the original table . Means the changes will be done in the original table.

Updatable View

- VIEWs are either **Updatable or Read-only**, but not both.
- INSERT, UPDATE, and DELETE operations are allowed on updatable VIEWs and base tables, subject to any other constraints.
- INSERT, UPDATE, and DELETE are not allowed on read-only VIEWs, but we can change their base tables, as we would expect.
- An updatable VIEW is one that can have each of its rows associated with exactly one row in an underlying base table.
- When the VIEW is changed, the changes pass unambiguously through the VIEW to that underlying base table.

Non-Updatable View (cont..)

- In Non-updatable view, the SELECT statement **contain** any of the following elements:
  - Aggregate functions such as MIN, MAX, SUM, AVG, and COUNT.
  - DISTINCT
  - GROUP BY clause.
  - HAVING clause.
  - UNION or UNION ALL clause.
  - Left join or outer join.
  - Subquery in the SELECT clause or in the WHERE clause that refers to the table appeared in the FROM clause.
  - Reference to non-updatable view in the FROM clause.
  - Reference only to literal values.
  - Multiple references to any column of the base table.

Dropping a View

- DROP query is used to delete a view.

Syntax:

DROP view view_name;

Example:

DROP view faculty;

Index

- Indices are data structures used to speed up access of records with specified values for index attributes.
- Indexes are used to find rows with specific column values quickly.
- Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. (Sequential Scan)
- If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data.
- This is much faster than reading every row sequentially
- MySQL create default indexes on PRIMARY KEY, UNIQUE KEY
- User defined index can be created using CREATE INDEX COMMAND
- MySQL indices are stored in B-trees.

SQL Joins

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the **from** clause
- **Join condition** – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

SQL Joins : Cross Join

Cross JOIN is a simplest form of JOINs which matches each row from one database table to all rows of another. In other words it gives us combinations of each row of first table with all records in second table as cartesian product

SELECT * FROM `movies` CROSS JOIN `members`

OR

SELECT * FROM `movies` ,`members`

members

id	first_name	last_name	movie_id
1	Adam	Smith	1
2	Ravi	Kumar	2

movie

movie_id	title	category
1	ASSASSIN'S CREED:	Animations
2	Real Steel(2012)	Animations

SQL Joins : Inner Join

The inner JOIN is used to return rows from both tables that satisfy the given condition(join condition on common column).

SELECT * FROM movies INNER JOIN `members` on movies.movie_id` = members.movie_id

OR

SELECT * FROM movies ,members WHERE movies.movie_id = members.movie_id

SQL Joins : Outer Join

MySQL Outer JOINs return all records matching from both tables .It can detect records having no match in joined table. It returns **NULL** values for records of joined table if no match is found.

```
SELECT A.title , B.first_name , B.last_name
FROM movies "A" LEFT OUTER JOIN members " B"
ON B.`movie_id` = A. 'movie_id'
# Some SQL Support keyword : Left join/natural left outer join
```

OR

```
SELECT A.title , B.first_name , B.last_name
FROM movies "A" LEFT OUTER JOIN members " B" USING ( `movie_id` )
```

The LEFT JOIN returns all the rows from the table on the left even if no matching rows have been found in the table on the right.

Where no matches have been found in the table on the right, NULL is returned.

What will Right Outer return?

What will full outer return?

SQL Joins

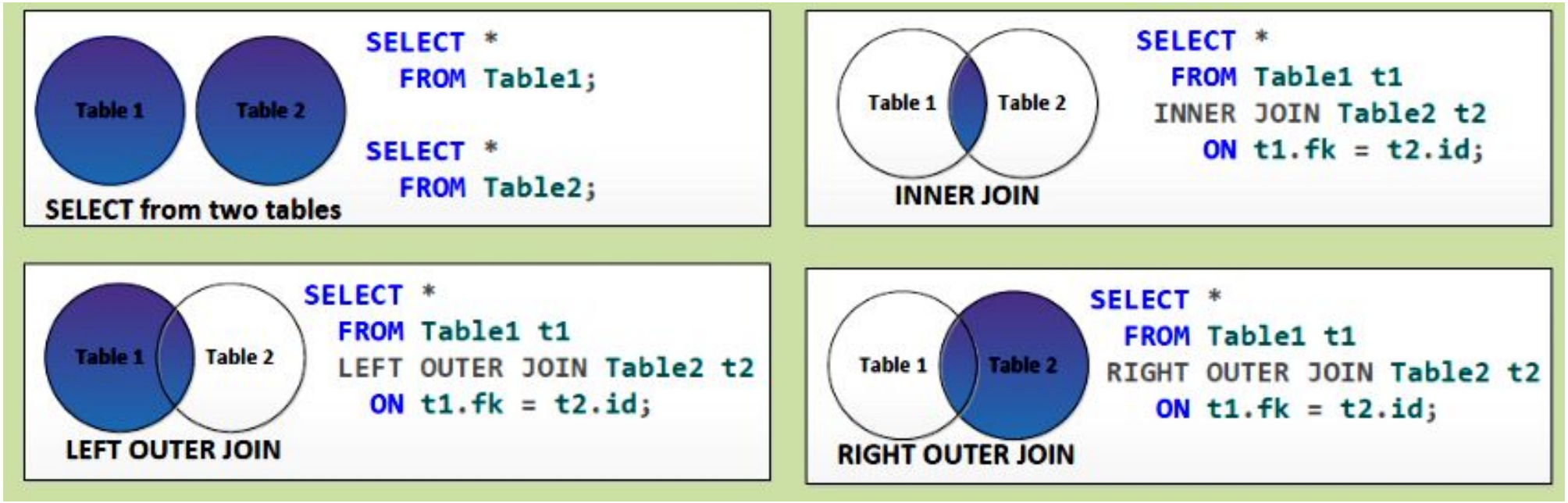


Table 1
Table 2
SELECT from two tables
SELECT *
FROM Table1;
SELECT *
FROM Table2;
Table 1
Table 2
INNER JOIN
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.fk = t2.id;
Table 1
Table 2
LEFT OUTER JOIN
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id;
Table 1
Table 2
RIGHT OUTER JOIN
SELECT *
FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
ON t1.fk = t2.id;

SQL Joins

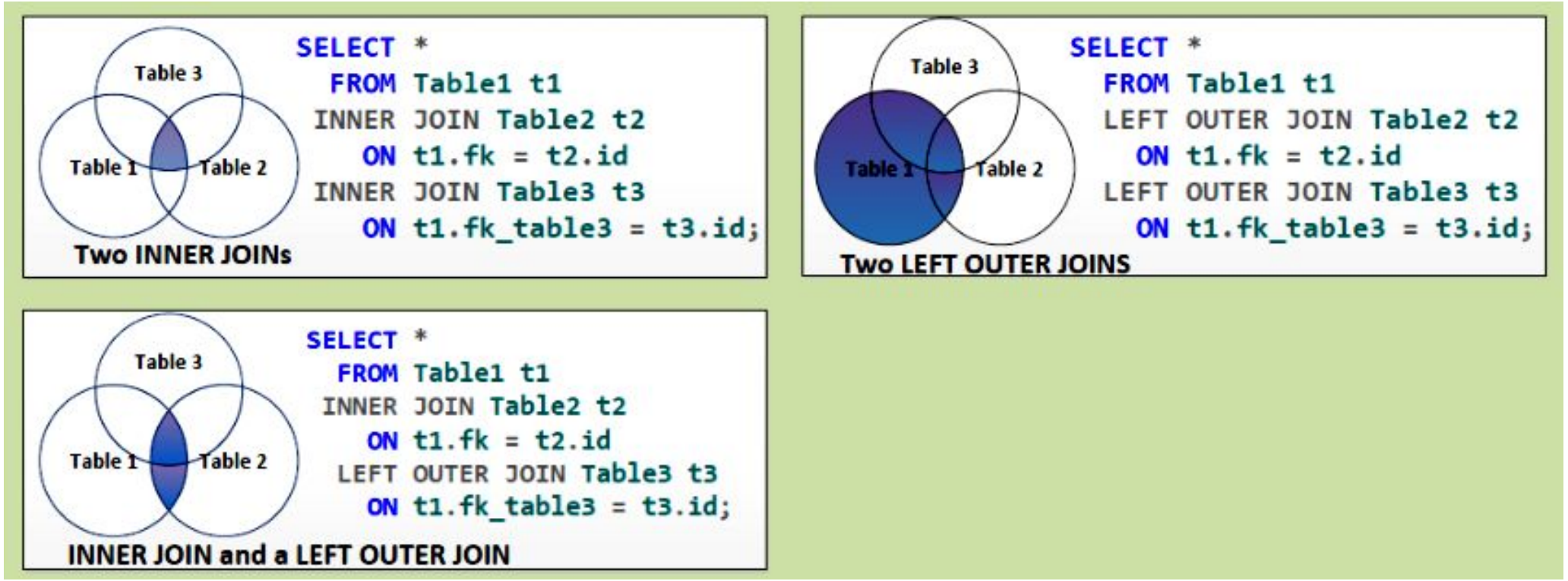


Table 3
Table 1
Table 2
Two INNER JOINs
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.fk = t2.id
INNER JOIN Table3 t3
ON t1.fk_table3 = t3.id;
Table 3
Table 1
Table 2
Two LEFT OUTER JOINS
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id
LEFT OUTER JOIN Table3 t3
ON t1.fk_table3 = t3.id;
Table 3
Table 1
Table 2
INNER JOIN and a LEFT OUTER JOIN
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.fk = t2.id
LEFT OUTER JOIN Table3 t3
ON t1.fk_table3 = t3.id;

Join operations – Example

Relation *course*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

Relation *prereq*

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that

  prereq information is missing for CS-315 and

  course information is missing for CS-437

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values.

Left Outer Join And Right Outer Join

course **natural left outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

- *course* **natural right outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

cours

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

Full Outer Join

- *course* **natural full outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

cours

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

Joined Relations – Examples

*Select * from course* **inner join** *prereq* **on** *course.course_id = prereq.course_id*

course_id	title	dept_name	credits	prereq_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

What is the difference between the above, and a natural join?

- *Select * from course* **left outer join** *prereq* **on** *course.course_id = prereq.course_id*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

course

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

prereq

course_id	title	dept_name	credit
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

Joined Relations – Examples

- *course* **full outer join** *prereq* **using** (*course_id*)

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

prereq_id

Aggregate Functions

- Multiple –row functions are called as aggregate functions
- Act on a multiple row in the relation returning single value as an output

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

Aggregate Functions Examples

Find the average salary of instructors in the Computer Science department

- **select avg** (*salary*),min(salary), max(salary),sum(salary)
  from *instructor*
  where *dept_name*= 'Comp. Sci.';

Find the number of tuples in the *course* relation

- **select count** (*) **from instructor**;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Aggregate Functions – Group By

Find the average salary of instructors in each department

- **select** *dept_name*, **avg** (*salary*) **as** *avg_salary*
  from *instructor*
  group by *dept_name*;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregation (Cont.)

Attributes in **select** clause outside of aggregate functions must appear in **group by** list

- /* erroneous query */
  select *dept_name*, *ID*, **avg** (*salary*)
  from *instructor*
  group by *dept_name*;

Discuss why query is erroneous, [Hint :refer last table]

Aggregate Functions – Having Clause

Find the names and average salaries of all departments whose average salary is greater than 42000

select *dept_name*, **avg** (*salary*) **as** *avg_salary*
from *instructor*
group by *dept_name*
having avg (*salary*) > 42000;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

Like Operator

- **Percent (%):** The % character matches any substring.
- **Underscore (_):** The character matches any character.
- Patterns are case sensitive; that is, uppercase characters do not match lowercase characters, or vice-versa.
- To illustrate pattern matching, we consider the following examples:
- 'Intro%' matches any string beginning with "Intro".
- '%Comp%' matches any string containing "Comp" as a substring, for example, 'Intro. to Computer Science', and 'Computational Biology'.
- '---' matches any string of exactly three characters.
- '---%' matches any string of at least three characters.

Queries with Like Operator

Using % (percent) Wildcard:

Syntax: select name from table_name where column_name not like 'pattern';

- SELECT name FROM student WHERE city LIKE 'Luck%';
- SELECT name FROM student WHERE city NOT LIKE 'Luck%';
- SELECT * FROM student WHERE firstname = 'Ajeet' AND id > 3;
- SELECT * FROM student WHERE firstname = 'Ajeet' OR id > 100;
- SELECT * FROM student WHERE student_nm IN ('Ajeet', 'Vimal', 'Deepika');
- SELECT * FROM student WHERE id BETWEEN 1 AND 3;

Null Values and Aggregates

Total all salaries

select sum (*salary*)
from *instructor*

- Above statement ignores null amounts
- Result is *null* if there is no non-null amount

All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes

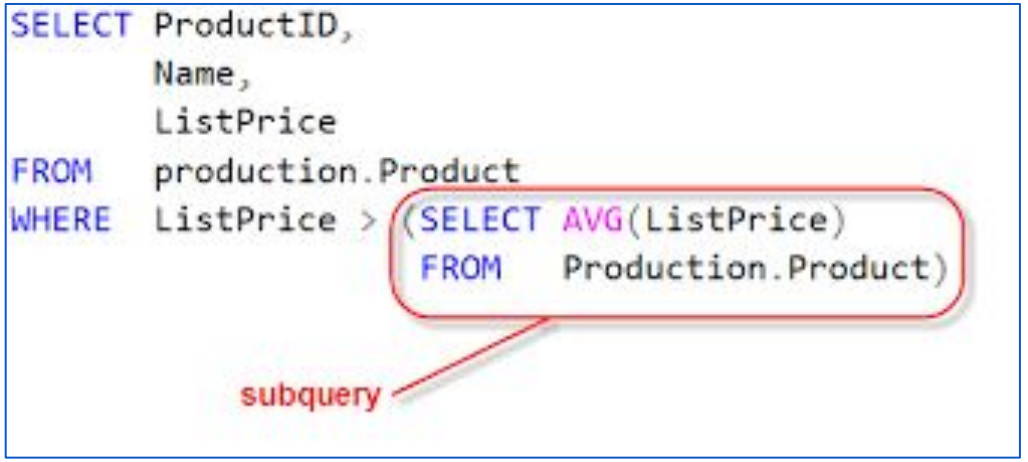
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

Subqueries (Nested Query)

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.



Examples of Subquery in DML and Select

SQL> SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID FROM CUSTOMERS WHERE SALARY > 4500) ;

SQL> INSERT INTO CUSTOMERS_BKP SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID FROM CUSTOMERS) ;

SQL> UPDATE CUSTOMERS SET SALARY = SALARY * 0.25 WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP WHERE AGE >= 27);

SQL> DELETE FROM CUSTOMERS WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP WHERE AGE >= 27);

Subqueries in the From Clause

- Find the average instructors' salaries of those departments where the average salary is greater than $42,000."

```
select dept_name, avg_salary
from ( select dept_name, avg (salary) as avg_salary
        from instructor
        group by dept_name)
where avg_salary > 42000;
```

Note that we do not need to use the having clause

Another way to write above query

```
select dept_name, avg_salary
from ( select dept_name, avg (salary)
        from instructor
        group by dept_name)
        as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Set Operations

Set operations are **union, intersect,** and **minus**

- Each of the above operations automatically eliminates duplicates

To retain all duplicates use the keyword all

- **union all,**
- **intersect all**
- **Minus**

- Select name from table1 union select name from table2;

table1

ID	NAME
1	ABHI
2	SAMEER
3	SAMEER

table2

ID	NAME
3	JAVED
4	SAMEER

Set Operations -examples

- SELECT name FROM table1

  UNION
  SELECT name FROM table2
  ORDER BY name;

  Note: The number of columns in the two selected tables or queries of a union query must match and also with similar data types.

- SELECT id,name FROM table1
  WHERE name-=‘sameer’
  UNION
  SELECT id,name FROM table2
  WHERE name=‘sameer’
  ORDER BY name;

table1

ID	NAME
1	ABHI
2	SAMEER
3	SAMEER

table2

ID	NAME
3	JAVED
4	SAMEER

Set Membership

Find courses offered in Fall 2017 and in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
        course_id in (select course_id
                      from section
                      where semester = 'Spring' and year=
2018);
```

Find courses offered in Fall 2017 but not in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
        course_id  not in (select course_id
                           from section
                      where semester = 'Spring' and year= 2018);
```

```
section (course_id , sec_id , semester ,
      year , building , room number   ,
time_slot_id varchar (4),   primary key
(course_id,   sec_id,semester,year),
      foreign key (course id)
references course);
```

Set Membership (Cont.)

- Name all instructors whose name is neither “Mozart” nor Einstein”

select distinct *name*
from *instructor*
where *name* **not in** ('Mozart', 'Einstein')

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Test for Empty Relations

- EXISTS and NOT EXISTS are used with a subquery in WHERE clause to examine if the result the subquery returns is TRUE or FALSE.
- The true or false value is then used to restrict the rows from outer query select.
- As EXISTS and NOT EXISTS only return TRUE or FALSE in the subquery, the SELECT list in the subquery does not need to contain actual column name(s).

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Continued…

- SELECT * FROM customers WHERE EXISTS (SELECT * FROM order_details

  WHERE customers.customer_id = order_details.customer_id);

- SELECT * FROM customers WHERE NOT EXISTS (SELECT * FROM order_details WHERE customers.customer_id = order_details.customer_id);

Insert,update,delete commands can also be used with EXISTS commands

- INSERT INTO contacts (contact_id, contact_name) SELECT supplier_id, supplier_name FROM suppliers WHERE EXISTS (SELECT * FROM orders WHERE suppliers.supplier_id = orders.supplier_id);

- Delete from contacts SELECT supplier_id, supplier_name FROM suppliers WHERE EXISTS (SELECT * FROM orders WHERE suppliers.supplier_id = orders.supplier_id);

The End