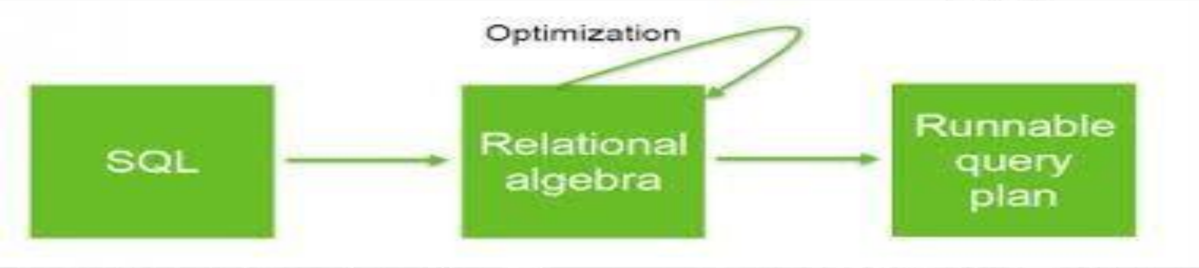
Unit-III

Relational Algebra and Database Programming: Relational Algebra, Basic Operations, Relational calculus: Tuple Calculus, Domain Calculus,
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, SQL DML Queries, Set Operations, Predicates and Joins, Set membership, Grouping and Aggregation, Aggregate Functions, Nested Queries

Relational Algebra



Relational Algebra

- What is “algebra ?

- Mathematical model consisting of:
  - *Operands* --- Variables or values;
  - *Operators* --- Symbols denoting procedures that construct new values from a given values

- **Relational Algebra :** is an algebra whose operands are relations and operators are designed to do the most commons things that we need to do with relations

- **Basic Relational Algebra Operations**:

- Select σ
- Project $\prod$
- Union $\cup$
- Set Difference (or Subtract or minus) $-$
- Cartesian Product X
- Natural Join

Relational Algebra: Select Operation

- **Notation:** $\sigma_p(r)$

p is called the selection predicate

➢ Defined as:

$$\sigma_p(\mathbf{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

➢ Where p is a formula in propositional calculus consisting of terms connected by : ∧ (**and**), ∨(**or**), ¬(**not**)
Each term is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: =, ≠, >, ≥. <. ≤

Example of selection:

Account(account_number, branch_name,balance)

$\sigma_{\text{branch-name}=\text{“Perryridge”}}(\text{account})$

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

Relation *r*

A	B	C	D
α	α	1	7
β	β	23	10

$\sigma_{A=B \wedge D > 5}(r)$

Relational Algebra: Project Operation

- **Notation:** $\prod_{A1, A2, \ldots, Ak}(r)$

where A_1, A_2 are attribute names and r is a relation.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets

E.g. to eliminate the *branch-name* attribute of *account*

$$\prod_{account\text{-}number,\ balance}(account)$$

- If relation Account contains 50 tuples, how many tuples contains $\prod_{account\text{-}number,\ balance}(account)$?

Example of Project Operation :

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

Relation *r*

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

$\prod_{A,C}(r)$

That is, the projection of a relation on a set of attributes is a **set of tuples**

Relational Algebra: Union Operation

Notation: $r \cup s$

- Consider relational schemas:

  Depositor(customer_name, account_number)

  Borrower(customer_name, loan_number)

- For $r \cup s$ to be valid.

  1. r, s must have the same number of attributes

  2. The attribute domains must be *compatible* (e.g., 2nd column of r deals with the same type of values as does the 2nd column of s)

- Find all customers with either an account or a loan

  $$\prod_{customer\text{-}name}(depositor) \cup \prod_{customer\text{-}name}(borrower)$$

Example of Union:

A	B
α	1
α	2
β	1

Relation r

A	B
α	2
β	3

Relation s

A	B
α	1
α	2
β	1
β	3

$r \cup s$

Relational Algebra: Set Difference Operation

- **Notation** : ***r – s***

Set differences must be taken between *compatible* relations.

- *r* and *s* must have the same number of attributes
- attribute domains of *r* and *s* must be compatible

Example of Set Difference:

A	B
α	1
α	2
β	1

Relation r

A	B
α	2
β	3

Relation s

A	B
α	1
β	1

r-s

Relational Algebra: Cartesian Product Operation

- **Notation** : *r x s*

Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).

If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.

Relation r

Relation s

r xs

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Relational Algebra: Natural Join Operation

- **Notation** : $r \bowtie s$
- We can perform a Natural Join only if there is at least one common attribute that exists between two relations
- The common attributes must have the same name and domain.
- Natural join acts on those matching attributes where the values of attributes in both the relations are same.
- **It avoids duplication of columns while providing the result as compared to other joins/cartesian-product.**

A	B
α	1
β	2

Relation r

A	C
α	10
β	30

Relation s

$r \bowtie s$

A	B	C
α	1	10
β	2	30

Relational Algebra Operators

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary}>=85000}(instructor)$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID,\,salary}(instructor)$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
$\bowtie$ (Natural join)	$instructor \bowtie department$
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
$\times$ (Cartesian product)	$instructor \times department$
	Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
$\cup$ (Union)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	Output the union of tuples from the two input relations.

Database Languages and Programming

School of Computer Engineering and technology

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, MySQL, 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).

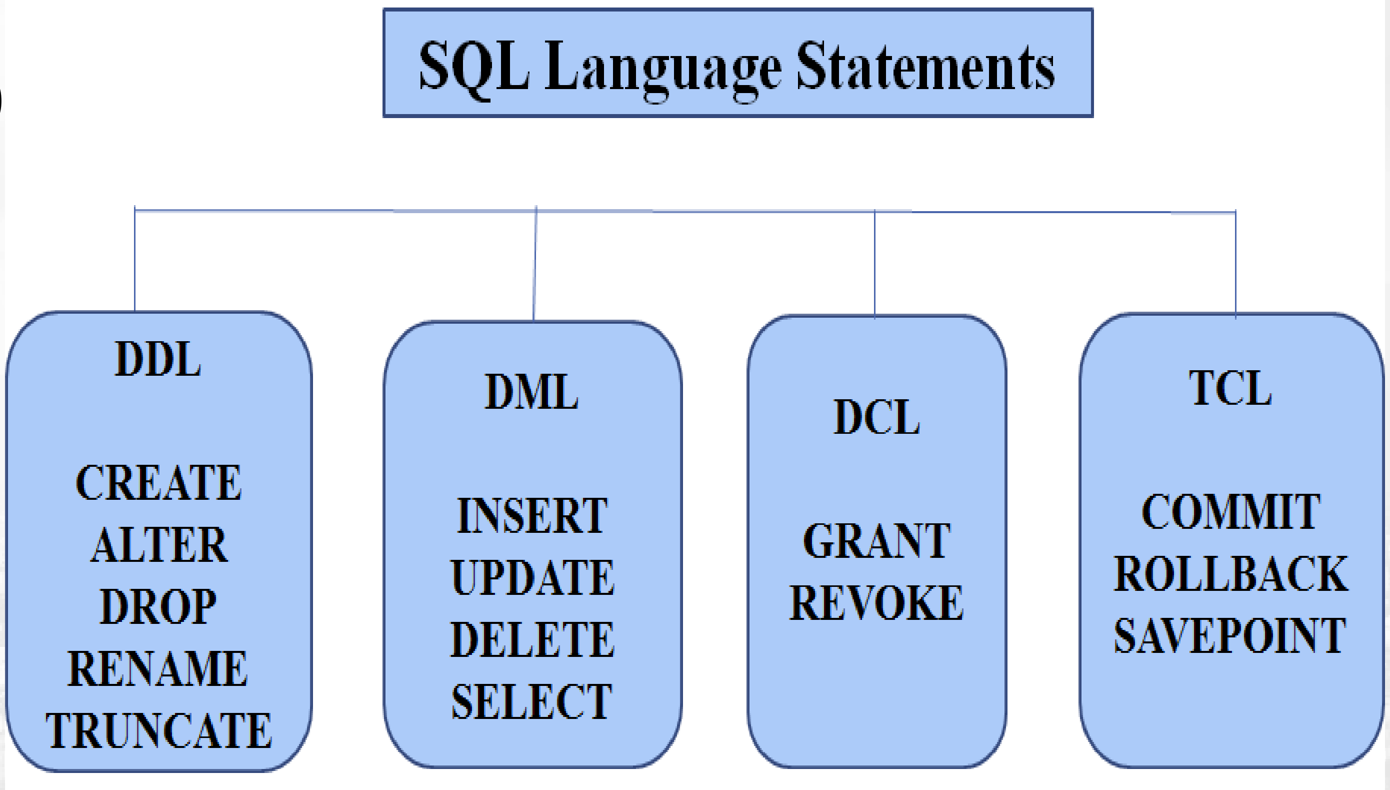
decimal(p,d). Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point. (ex., **decimal(3,1)**, allows 44.5 to be stored exactly, but not 444.5 or 0.32)

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

Create Table Construct

- 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))

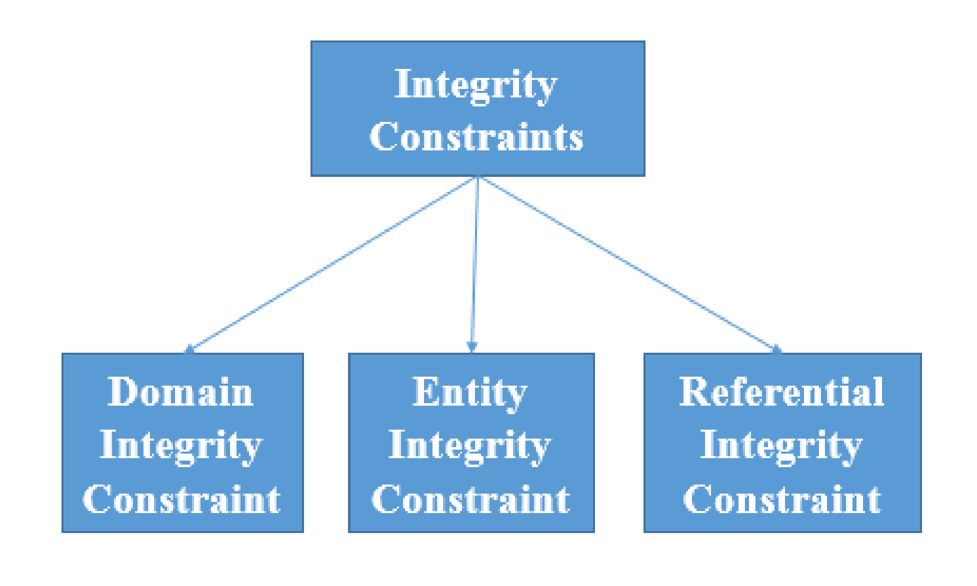
ID	*name*	*dept_name*	*salary*

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 3 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
	AAA

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 sp
   - 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.

Primary Key

Not allowed as Emp_ID is a primary key.

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

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.

(Table 1)

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

Foreign key

Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined

Relationships

(Table 2)

Primary Key

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

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 updated, 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.

5. **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 '2020-07-10');

2. **To modify a column datatype/size.**

> ***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*** *old_column_name* ***TO*** *new_column_name;*

E.g. ALTER TABLE student RENAME 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.
- **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‘
- 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);

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;

Continued…….

- From Root
-
- mysql> REVOKE ALL ON *.* FROM 'finley'@'localhost';
- mysql> REVOKE CREATE,DROP   ON *.*   FROM 'finley'@'localhost';

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,

- Insertion of new tuples into a given relation
- Deletion of tuples from 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;

SELECT Query

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

Select ------ → Mandatory Clause
From
Where
Group by → Optional 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".

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).

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.

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**

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 taught some course and the course_id,

    select name, course_id
    from instructor **as** T , teaches **as** S
    where T.ID = S.ID

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

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

SQL Operators

- SQL Arithmetic Operators
- SQL Comparison Operators
- SQL Logical Operators

Arithmetic Operators

- 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*

SQL Comparison Operators

- *Select * from employee*
  where salary = 90000;

- *Select * from employee*
  where salary <>100000;

- *Select * from employee*
  where salary >=90000 and salary<=100000

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

SQL Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

Logical Operators(Cont..)

- **ALL**

  SELECT * FROM Products WHERE Price > ALL (SELECT Price FROM Products WHERE Price > 500);

- **AND**

  SELECT * FROM Customers WHERE City = "London" AND Country = "UK";

- **ANY**

  SELECT * FROM Products WHERE Price > ANY (SELECT Price FROM Products WHERE Price > 50);

- **BETWEEN AND**

  SELECT * FROM Products WHERE Price BETWEEN 50 AND 60;

- **EXISTS**

  SELECT * FROM Products WHERE EXISTS (SELECT Price FROM Products WHERE Price > 50);

Logical Operators(Cont..)

- **IN**

  SELECT * FROM Customers WHERE City IN ('Paris','London');
- **LIKE**

  SELECT * FROM Customers WHERE City LIKE 's%';
- **NOT**

  SELECT * FROM Customers WHERE City NOT LIKE 's%';
- **OR**

  SELECT * FROM Customers WHERE City = "London" OR Country = "UK";
- **SOME**

SELECT * FROM Products WHERE Price > SOME (SELECT Price FROM Products WHERE Price > 20);

SQL Functions

- **Single Row Functions** : Operate on each row and return one output for each row.
  - Date Functions, String Functions such as length or case conversion functions like UPPER, LOWER.
  - Number functions such as ROUND, TRUNC, and MOD etc.
- **Multi Row Functions** : Aggregate Function/Group Functions : Operates on Group of rows and return output for the complete set of rows. Also known as Group functions.
  - 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
DATE_ADD(date, INTERVAL value addunit)	Adds a specified time interval to a 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.

Single row Function : String (Pattern matching)

- 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:
  - **Percent (%):** The % character matches any substring.
  - **Underscore (_):** The character matches any character in the string.
- ’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.

Pattern matching examples……

- Syntax is

select <column name> from <table_name >where <column_name > ***like/ not like*** *'pattern';*

- To find the records starting with ‘Luck’
  - *SELECT * FROM student WHERE city LIKE 'Luck%';*
- To find the names not starting with ‘Luck’
  - *SELECT name FROM student WHERE city NOT LIKE 'Luck%';*
- To find the names ending with ‘ly’
  - *SELECT * FROM student WHERE city LIKE '%fy’ ;*
- Find names containing a y
  - *SELECT * FROM student WHERE city LIKE '%y%‘;*
- To find names containing exactly five characters
  - *SELECT * FROM student WHERE city LIKE '_____';*

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*

Aggregate Functions

Type	Use	Functions
Single –row functions	Operate on a single column of a relation of single row n the table returning single value as an output	String functions, Date Functions
Multiple –row functions	Act on a multiple row in the relation returning single value as an output	Avg, min, max, sum, count

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

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

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

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

Null Values and Aggregates

- To find the 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

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 $(A_1, A_1, \ldots, A_n)$

v1

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 as a Cartesian product.
- The cross join does not establish a relationship between the joined tables.
- SELECT * FROM `Movies` CROSS JOIN `Artist` OR
- SELECT * FROM `Movies` ,`Artist`;

Movies

Movie_id	Title	Category
1	ASSASSIN'S CREED:	Animations
2	Real Steel(2012)	Animations

Artist

Id	First_name	Last_name	Movie_id
1	Adam	Smith	1
2	Ravi	Kumar	2

Cross Join of 2 tables

Movie_id	Title	Category	Id	First_name	Last_name	Movie_id
1	ASSASSIN'S CREED:	Animations	1	Adam	Smith	1
1	ASSASSIN'S CREED:	Animations	2	Ravi	Kumar	2
2	Real Steel(2012)	Animations	1	Adam	Smith	1
2	Real Steel(2012)	Animations	2	Ravi	Kumar	2

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 `Artist` on movies.movie_id = Artist.movie_id

OR

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

Movie_id	Title	Category	Id	First_name	Last_name	Movie_id
1	ASSASSIN'S CREED:	Animations	1	Adam	Smith	1
2	Real Steel(2012)	Animations	2	Ravi	Kumar	2

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 Artist " 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 Artist " B" USING (`movie_id`)

*Use Using keyword for left and right join queries only not for full outer join queries

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?

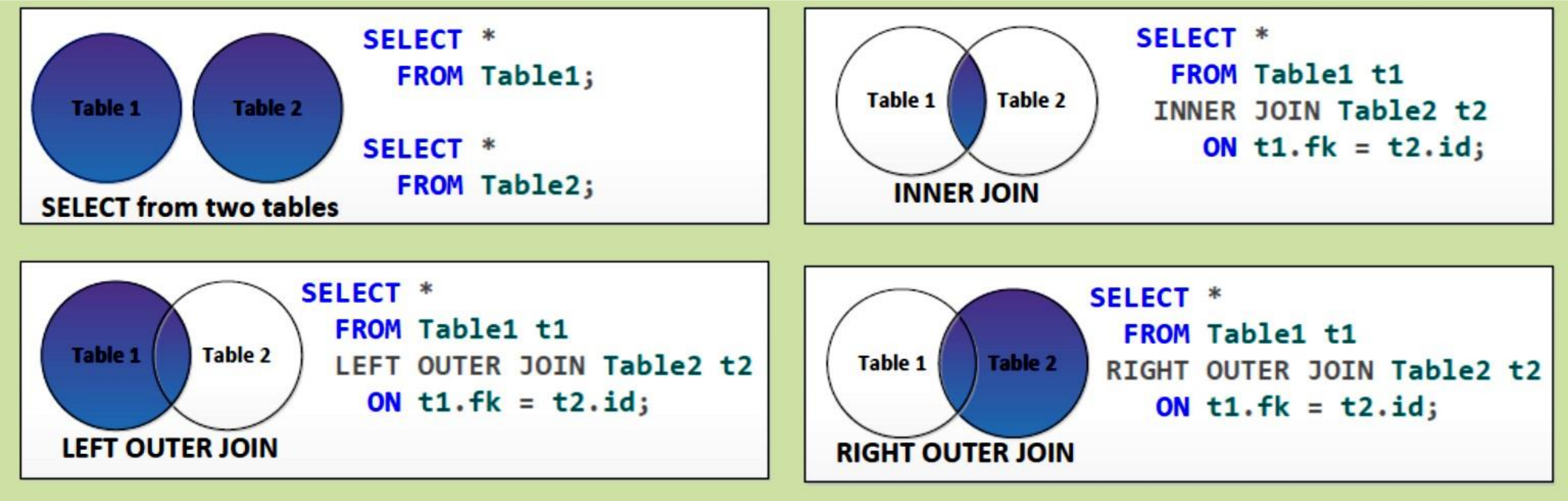
Left outer join Output (contd..)

Movie_id	Title	Category
1	ASSASSIN'S CREED:	Animations
2	Real Steel(2012)	Animations
3	**Jurassic Park**	**Animation**

Id	First_name	Last_name	Movie_id
1	Adam	Smith	1
2	Ravi	Kumar	2

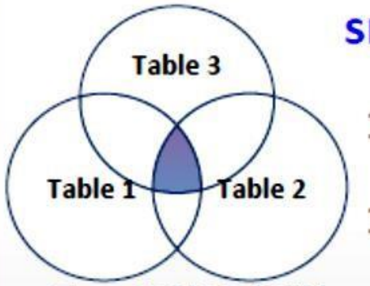
Title	First_name	Last_name
ASSASSIN'S CREED:	Adam	Smith
Real Steel(2012)	Ravi	Kumar
Jurassic Park	Null	Null

SQL Joins - Revision



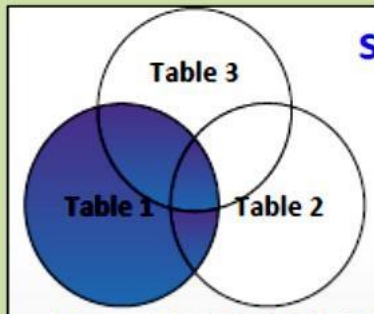
v1

SQL Joins on multiple tables



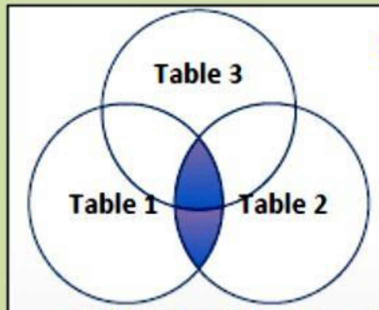

```sql
SELECT *
  FROM Table1 t1
  INNER JOIN Table2 t2
    ON t1.fk = t2.id
  INNER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

Two INNER JOINs




```sql
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;
```

Two LEFT OUTER JOINS




```sql
SELECT *
  FROM Table1 t1
  INNER JOIN Table2 t2
    ON t1.fk = t2.id
  LEFT OUTER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

INNER JOIN and a LEFT OUTER JOIN

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 relation is missing for CS-315 and

  course relation is missing for CS-347

Left Outer Join And Right Outer Join

- *Select * from course* **natural left outer join** *prereq*

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*

- *Select * from 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

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

prereq

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

Full outer Join

- **Full Outer Join is implemented as union of left outer and right outer join in MYSQL.**

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

- *select * from course* **left outer join** *prereq on course.course_id = prereq.course_id*
union
*select * from course* **right outer join** prereq *on course.course_id = prereq.course_id*

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

prereq

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

v6

Inner join Vs. Natural Join

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

course_id	*title*	*dept_name*	*credits*	*prere_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 above query, and a natural join?

- The **difference** is in **natural join** no need to specify condition but in **inner join** condition is mandatory.
- The repeated column is **avoided** in the **output** of natural join.
- ***Select * from course natural join prereq***

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

prereq

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

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.

```
SELECT ProductID,
       Name,
       ListPrice
FROM   production.Product
WHERE  ListPrice > (SELECT AVG(ListPrice)
                    FROM   Production.Product)
```

subquery

Examples of Subquery in DML and Select

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

- *Insert data in new table[table should be existing]*
- 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*)
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

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).

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);

Set Operations

Set operations are **union**, **intersect(inner join)**, and **minus(left join/right join)**

- Each of the above operations automatically eliminates duplicates

To retain all duplicates use the keyword all

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

table1

ID	NAME
1	ABHI
2	SAMEER
3	SAMEER

table2

ID	NAME
3	SAMEER
4	JAVED

ID	NAME
1	ABHI
2	SAMEER
3	SAMEER
4	JAVED

- *Select * from table1* **union** *select * from table 2;*

Set Operations -examples

ID	NAME
1	ABHI
2	SAMEER
3	SAMEER

table1

ID	NAME
3	SAMEER
4	JAVED

table2

- *Select distinct id from t1* ***inner join*** *t2 using(id);(****intersect****)*

ID	NAME
3	SAMEER

- *select id from t1* ***left join*** *t2 using (id) where t2.id is null;* ***(minus)***

ID	NAME
1	ABHI
2	SAMEER

- *Select * from table1* ***union all*** *select * from table 2;*

ID	NAME
1	ABHI
2	SAMEER
3	SAMEER
3	SAMEER
4	JAVED

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);
```

Set Membership (Cont.)

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

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

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

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.

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 , 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 although they are not visible to the user.
- MySQL indices are stored in B-trees.

Example

Syntax:

- CREATE INDEX <*index_name*> ON < *table_name* >(*column1*, *column2*, ...);
- CREATE UNIQUE INDEX <*index_name* > ON <*table_name*> (*column1*, *column2*, ...);
- ALTER TABLE <*table_name*> DROP INDEX <*index_name*;>

Example:

- create table person(pid int primary key , pnm varchar(10));
- create index id1 on person(pnm); *// indexes help to retrieve the data faster*.
- Create unique index id2 on person(pid,pnm) ; *// rows will have unique value*
- alter table person drop index id1;

References

1.Ramakrishnan, R. and Gherke, J., “Database Management Systems”, 3rd Ed., McGraw-Hill.

2. Connally T, Begg C.,”Database Systems”,Pearson Education

References

Text Books:

- Abraham Silberschatz, Henry F. Korth and S. Sudarshan, Database System Concepts 6th Ed, McGraw Hill, 2010.
- Elmasi, R. and Navathe, S.B., “Fundamentals of Database Systems”, 4th
- Ed., Pearson Education.

Reference Books :

- Ramakrishnan, R. and Gherke, J., “Database Management Systems”, 3rd Ed., McGraw-

Hill.

- Connally T, Begg C.,”Database Systems”,Pearson Education

End of Unit 2