Database Engineering

Unit 2

Structured Query Language (SQL)

Prof. Vivek V. Kheradkar

Getting Started

- Install jdk 1.7
- Install Oracle 11g Express Edition
- Set path C:\jdk1.7\bin
- Set classpath

Set Path

- Start
- Control Panel
- System
- Advanced Tab
- Environment Variables
- Select Path
- Edit
- Add semicolon;
- and then add path at the end – c:\jdk1.7\bin

Set classpath

- Start
- Control Panel
- System
- Advanced Tab
- Environment Variables
- New
- Give name – classpath
- Add semicolon;
- and then add path of the working folder

Getting Started

- Start-oracle11g-
- Get Started-
- Application Express-
- Create Database user and Workspace
- Login
- SQL Workshop

Online SQL Platform

- Online SQL Platform-
- https://livesql.oracle.com/

Banking schemas:

Branch-schema = (*branch-name*, *branch-city*, *assets*)

Customer-schema = (*customer-name*, *customer-street*, *customer-city*)

Loan-schema = (*loan-number*, *branch-name*, *amount*)

Borrower-schema = (*customer-name*, *loan-number*)

Account-schema = (*account-number*, *branch-name*, *balance*)

Depositor-schema = (*customer-name*, *account-number*)

SQL Command Categories

- SQL language is divided into the following command categories:
  - Data definition (Data Definition Language, or DDL)
  - Data manipulation (Data Manipulation Language, or DML)
  - View definition
  - Transaction control
  - Security and authorization
  - Embedded SQL and dynamic SQL.
  - Integrity.

Data Definition Language (DDL)

Statement used to define a database structure or schema

- CREATE - to create a new database object
- ALTER - to change an aspect of the structure of an existing database object
- DROP - to drop (remove) a database object
- TRUNCATE – remove all records from a table, including all spaces allocated for the records are removed
- RENAME - rename an database object

Schema Definition in SQL

Define an SQL relation by using the **create table** command:

$$\textbf{create table } r(A_1 D_1, A_2 D_2, \ldots, A_n D_n, \langle \text{integrity-constraint}_1 \rangle, \ldots, \langle \text{integrity-constraint}_k \rangle)$$

Create Table

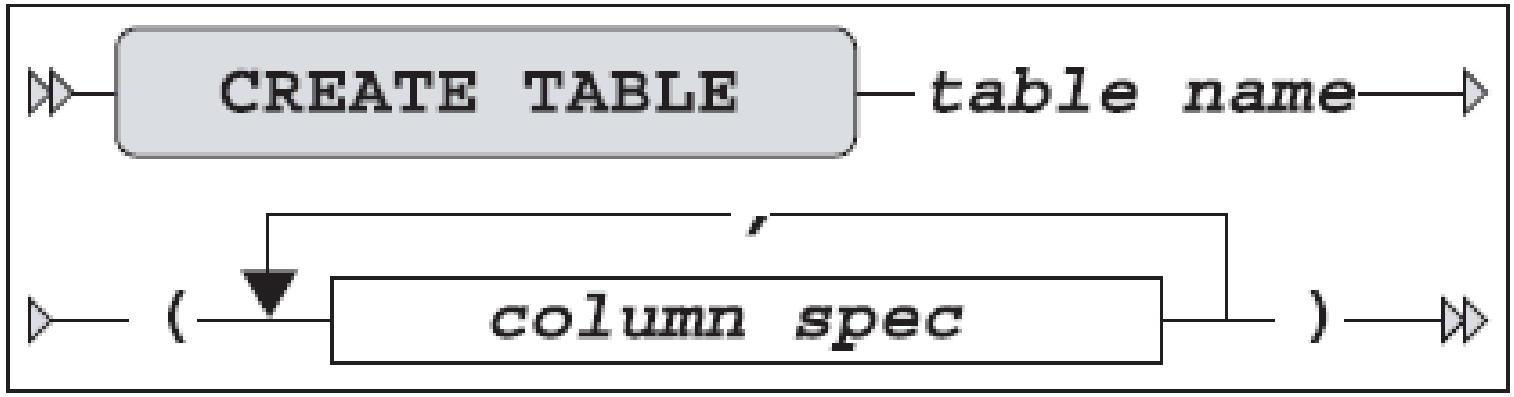


Figure 3-1. *CREATE TABLE basic command syntax diagram*

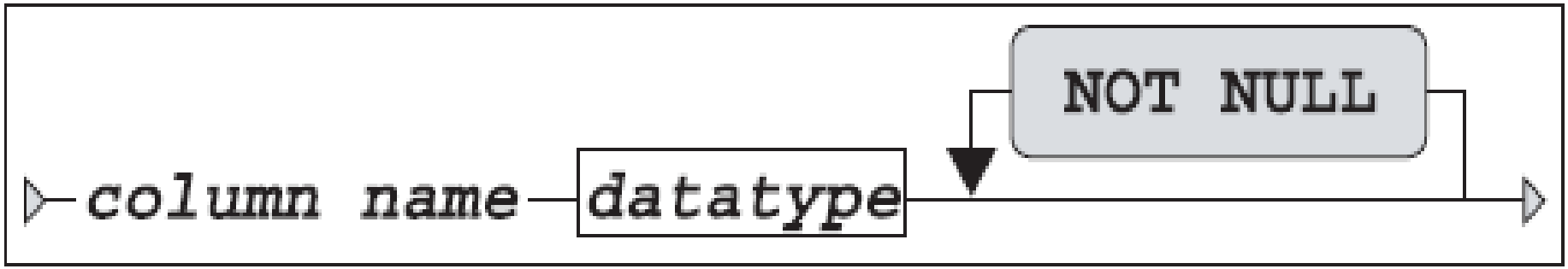


Figure 3-2. *Column specification syntax diagram*

Data Types

char(*n*): A fixed-length character string with user-specified length *n.*

varchar(n): A variable-length character string with user-specified maximum length *n*.

int/ integer: An integer.

numer(p, d): A fixed-point number with user-specified precision. The number consists of *p* digits (plus a sign), and *d* of the *p* digits are to the right of the decimal point.

float(n): A floating-point number.

real, double precision: Floating-point and double-precision floating-point numbers with machine-dependent precision.

date: A calendar date containing a (four-digit) year, month, and day of the month. Format: YYYY-MM-DD.

time: The time of day, in hours, minutes, and seconds. Format: hh:mm:ss.

timestamp: A combination of **date** and **time**. Format: YYYY-MM-DD hh:mm:ss.

```
SQL> create table  EMPLOYEES
  2  ( empno       number(4)    not null
  3  , ename       varchar2(8)  not null
  4  , init        varchar2(5)  not null
  5  , job         varchar2(8)
  6  , mgr         number(4)
  7  , bdate       date         not null
  8  , msal        number(6,2)  not null
  9  , comm        number(6,2)
 10  , deptno      number(2)            );
```

Listing 3-2. *The DEPARTMENTS Table*

```
SQL> create table  DEPARTMENTS
  2  ( deptno      number(2)    not null
  3  , dname       varchar2(10) not null
  4  , location    varchar2(8)  not null
  5  , mgr         number(4)            );
```

SQL data definition for part of the bank database

```
create table customer
    (customer-name      char(20),
     customer-street    char(30),
     customer-city      char(30),
     primary key (customer-name))

create table branch
    (branch-name        char(15),
     branch-city        char(30),
     assets             integer,
     primary key (branch-name),
     check (assets >= 0))
```

```
create table account
    (account-number   char(10),
     branch-name      char(15),
     balance          integer,
     primary key (account-number),
     check (balance >= 0))

create table depositor
    (customer-name    char(20),
     account-number   char(10),
     primary key (customer-name, account-number))
```

DROP

Syntax:

DROP <OBJECT> <object name>

Example:

DROP TABLE Employee;

TRUNCATE

Syntax:

TRUNCATE TABLE <table name>;

Example:

TRUNCATE TABLE Employee;

RENAME

Syntax:

RENAME <old object name> TO <new object name>

Example:

RENAME temporary to new

Alter Table

- The SQL **ALTER TABLE** command is used to add, delete or modify columns in an existing table.
- also use ALTER TABLE command to add and drop various constraints on an existing table.
- **Syntax:**
- The basic syntax of **ALTER TABLE** to add a new column in an existing table is as follows:
- ALTER TABLE table_name ADD column_name datatype;

Alter Table

- The basic syntax of ALTER TABLE to **DROP COLUMN** in an existing table is as follows:
- ALTER TABLE table_name DROP COLUMN column_name;

- The basic syntax of ALTER TABLE to change the **DATA TYPE** of a column in a table is as follows:
- ALTER TABLE table_name MODIFY column_name datatype;

SQL Constraints

- used to specify rules for the data in a table.
- Constraints 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 table. If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

SQL Constraints

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

FOREIGN KEY - Uniquely identifies a row/record in another table

CHECK - Ensures that all values in a column satisfies a specific condition

DEFAULT - Sets a default value for a column when no value is specified

INDEX - Used to create and retrieve data from the database very quickly

Not Null Constraint

- If a column in a table is specified as Not Null,
- then it's not possible to insert a null in such column.
- It can be implemented with create and alter commands.
- When we implement the Not Null constraint with alter command there should not be any null values in the existing table.

Not Null

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
PRIMARY KEY (P_Id)
)
```

Unique Constraint

The unique constraint doesn't allow duplicate values in a column.

If unique constraint encompasses two or more columns, no two equal combinations are allowed

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

DEFAULT Constraint

The DEFAULT constraint is used to insert a default value into a column.

The default value will be added to all new records, if no other value is specified.

```
CREATE TABLE Persons
    (
    P_Id int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255) DEFAULT ‘Ichalkaranji'
    )
```

Primary Key Constraints

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

```
CREATE TABLE books
( book_id NUMBER PRIMARY KEY,
book_name VARCHAR2(30),
author_name VARCHAR2(40),
book_isbn VARCHAR2(20) );
```

```
CREATE TABLE books ( book_id NUMBER,
    book_id_seq NUMBER, book_name
    VARCHAR2(30), author_name VARCHAR2(40),
    book_isbn VARCHAR2(20),
   CONSTRAINT pk_books PRIMARY KEY (book_id,
    book_id_seq));
```

FOREIGN Key Constraints

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

- Foreign Key

```
CREATE TABLE supplier
(     supplier_id          numeric(10)       not null,
      supplier_name        varchar2(50)      not null,
      contact_name         varchar2(50),
      CONSTRAINT supplier_pk PRIMARY KEY (supplier_id)
);

CREATE TABLE products
(     product_id           numeric(10)       not null,
      supplier_id          numeric(10)       not null,
      CONSTRAINT fk_supplier
      FOREIGN KEY (supplier_id)
      REFERENCES supplier(supplier_id)
);
```

```
CREATE TABLE supplier
(       supplier_id         numeric(10)     not null,
        supplier_name       varchar2(50)    not null,
        contact_name        varchar2(50),
        CONSTRAINT supplier_pk PRIMARY KEY (supplier_id, supplier_name)
);

CREATE TABLE products
(       product_id          numeric(10)     not null,
        supplier_id         numeric(10)     not null,
        supplier_name       varchar2(50)    not null,
        CONSTRAINT fk_supplier_comp
        FOREIGN KEY (supplier_id, supplier_name)
        REFERENCES supplier(supplier_id, supplier_name)
);
```

- CREATE TABLE ref1 (id VARCHAR2(3) PRIMARY KEY);

- CREATE TABLE ref2 (id VARCHAR2(3) PRIMARY KEY);

- CREATE TABLE look (p VARCHAR2(3),

  CONSTRAINT fk_p_ref1

  FOREIGN KEY (p) REFERENCES ref1(id),

  CONSTRAINT fk_p_ref2

  FOREIGN KEY (p) REFERENCES ref2(id));

CHECK Constraint

CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

CHECK Constraint

```
CREATE TABLE Persons
(
P_Id int NOT NULL CHECK (P_Id>0),
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

CHECK Constraint

- CREATE TABLE Persons
  (
  P_Id int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255),
  CONSTRAINT chk_Person CHECK (P_Id>0 AND
  City='Ichalkaranji')
)

Alter Table

- basic syntax of ALTER TABLE to add a **NOT NULL** constraint to a column in a table is as follows:
- ALTER TABLE table_name MODIFY column_name datatype NOT NULL;

- The basic syntax of ALTER TABLE to **ADD UNIQUE CONSTRAINT** to a table is as follows:
- ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);

Alter Table

- The basic syntax of ALTER TABLE to **ADD CHECK CONSTRAINT** to a table is as follows:
- ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);

- The basic syntax of ALTER TABLE to **ADD PRIMARY KEY** constraint to a table is as follows:
- ALTER TABLE table_name ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);

Alter Table

- The basic syntax of ALTER TABLE to **DROP CONSTRAINT** from a table is as follows:
- ALTER TABLE table_name DROP CONSTRAINT MyUniqueConstraint;

Alter Table

- The basic syntax of ALTER TABLE to **DROP PRIMARY KEY** constraint from a table is as follows:
- ALTER TABLE table_name DROP CONSTRAINT MyPrimaryKey;

- If you're using MySQL, the code is as follows:
- ALTER TABLE table_name DROP PRIMARY KEY;

- DROP TABLE table_name;

Sample Queries

- Create a table branch with name, city and asset with restriction as city should not null, asset value should greater than equal to 0 and default value is 0 and primary key name.
- Create a relation borrower which is between customer and loan.
- Add a column phoneNo to customer table.
- Change the size of the branch_city to varchar(20).
- Drop the column phoneNo from customer table.

Data Manipulation Language DML)

- INSERT, to add rows to a table
- UPDATE, to change column values of existing rows
- DELETE, to remove rows from a table

INSERT

- Insert data into a relation.
- insert statement is a request to insert one tuple

- Syntax :

  insert into t values (val1, val2, ..., valn);

- Syntax :

  insert into t(A1, A2,…, An) values (val1, val2, ..., valn);

INSERT

- Insert values for All Attributes
- INSERT INTO departments VALUES (280, 'Recreation', 121, 1700);

- Insert values for some Attributes
- INSERT INTO employees (employee_id, last_name, email, hire_date, job_id, salary, commission_pct) VALUES (207, 'Gregory', 'pgregory@oracle.com', sysdate, 'PU_CLERK', 1.2E3, NULL);

- Use of select in insert
- INSERT INTO bonuses SELECT employee_id, salary*1.1 FROM employees WHERE commission_pct > 0.25;

- Sample Queries
- Insert an account A-978245 at the Park Street branch and that is has a balance of Rs. 12000.

- Insert an customer vivek who live in street is main street and city is Ichalkarnji

Update Data

- SQL also has an UPDATE command for modifying existing tuples in a table

- General form:

  UPDATE table
  SET attr1=val1, attr2=val2, ...
  [WHERE condition];

- UPDATE employees SET salary = salary + 1000.0;

Sample Queries

- Change the assests of Perryridge branch to 340000000.
- Add 2% interest to all bank account balances with a balance of $500 or less.
- Transfer the accounts and loans of Perryridge branch to Downtown branch.
- An annual interest payments are being made, interest of 5% to be paid only to accounts with a balance of $1000 or more

Update Data

- SQL provides a case construct, which we can use to perform both the updates with a single update statement, avoiding the problem with order of updates.

- General form:

```
CASE
WHEN condition1 THEN result1
WHEN condition2 THEN result2
. . . . .
WHEN conditionn THEN resultn
ELSE result0
END
```

Update Data

- Transfer Rs. 100 from account A-101 to A-215.

- update account
  set balance = case
  when account_number=’A-101′ then balance-100
  when account_number=’A-215′ then balance+100
  else balance
  end;

delete

- DELETE statement is a used to delete a one or more records from a table.
- DELETE FROM table [WHERE conditions];
- DELETE FROM suppliers;
- DELETE FROM suppliers WHERE supplier_name = 'IBM';

Sample Queries

- Delete the branch Perryridge.
- Remove all the customer who live in “Downtown”
- Remove all the accounts

Retrieving/ Fetching Data from a table

Basic Structure

- SQL queries use the SELECT statement
- The basic structure of an SQL expression consists of three clauses: select, from, and where.
- The select clause corresponds to which column do want to see in result.
- The from clause corresponds to which table(s) is (are) needed for retrieval.
- The where clause corresponds to what condition to filter the rows .

Retrieving/ Fetching Data from a table

General form is:

SELECT A1, A2, ... An

FROM r1, r2, ...

WHERE P;

Where

- ri are the relations (tables)
- Ai are attributes (columns)
- P is the selection predicate

Retrieval

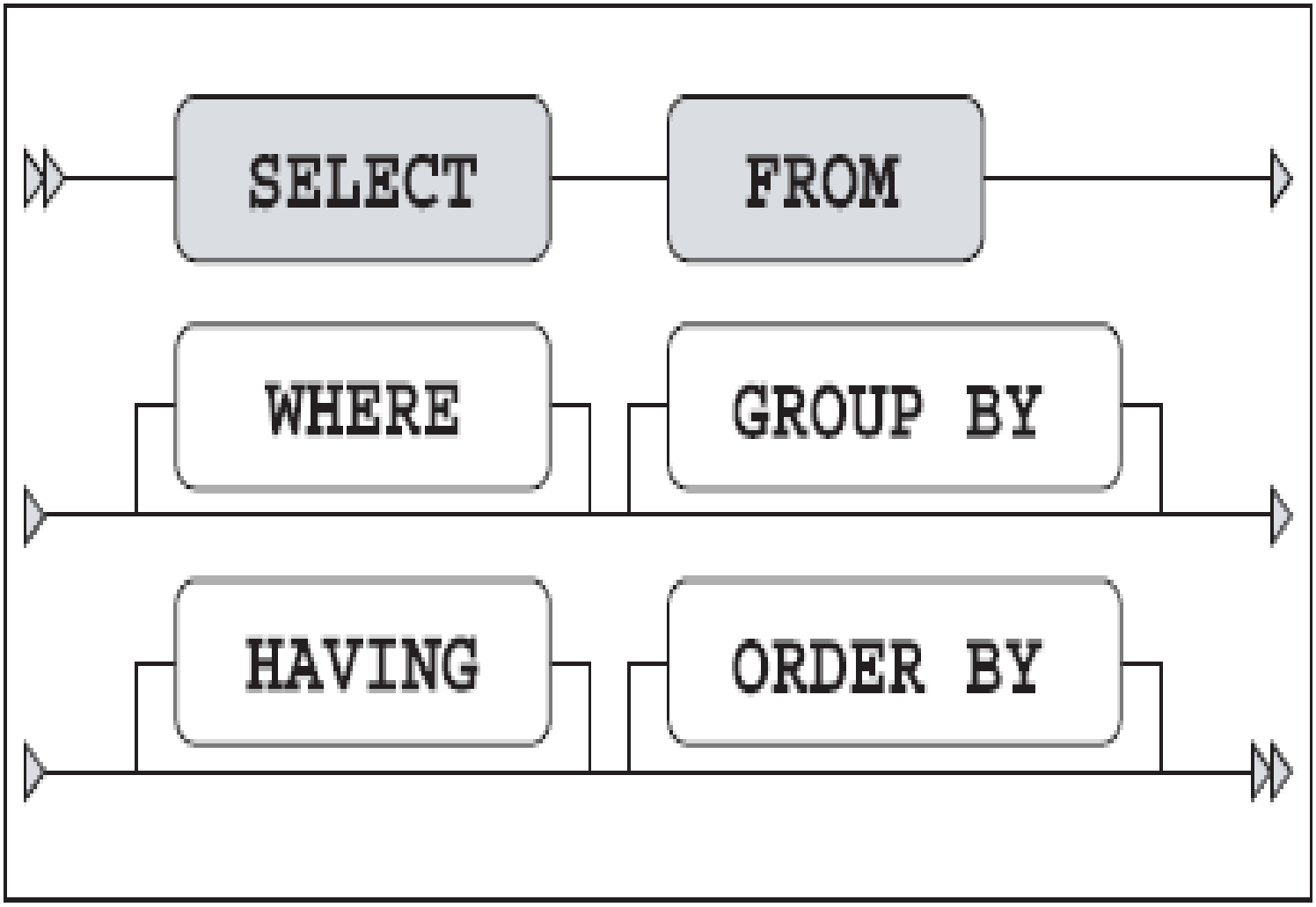


Figure 2-1. *The six main components of the SELECT command*

Table 2-1. *The Six Main Components of the SELECT Command*

Component	Description
FROM	Which table(s) is (are) needed for retrieval?
WHERE	What is the condition to filter the rows?
GROUP BY	How should the rows be grouped/aggregated?
HAVING	What is the condition to filter the aggregated groups?
SELECT	Which columns do you want to see in the result?
ORDER BY	In which order do you want to see the resulting rows?

Select

- List all the employee Details
  - Select * from employees;
  - SELECT FIRST_NAME, LAST_NAME, DEPARTMENT_ID FROM EMPLOYEES;
- List all branch names and their assets
- Displaying Selected Columns Under New Headings
- SELECT FIRST_NAME First, LAST_NAME last, DEPARTMENT_ID Dept FROM EMPLOYEES;

SQL - ALL, DISTINCT

- Find the names of all branches in the loan relation

  SELECT all branch_name

  FROM loan ;

  Or

  SELECT branch_name FROM loan ;

- Get all distinct branches in the loan relation

  SELECT Distinct branch_name FROM loan ;

- The asterisk symbol “ * ” can be used to denote “all attributes.”
- The select clause may also contain arithmetic expressions +, −, ∗, and / operating on constants or attributes of tuples.

- select loan-no, branch-name, amount * 100 from loan

Retrieving a subset of rows- where clause

- **Selecting Data that Satisfies Specified Conditions**
- SELECT FIRST_NAME, LAST_NAME, DEPARTMENT_ID FROM EMPLOYEES WHERE DEPARTMENT_ID = 90;

- Find all loan numbers for loans with loan amounts greater that $1200.

  select loan-no
  from loan
  where amount> 1200

Relational Operator

- SQL uses the Relational operators

  = , < , > , <= , >= , != or < >

- Find the loan number of those loans made at Perryridge branch ,

  SELECT loan-no

  FROM loan

  WHERE branch-name=‘Perryridge’;

Logical Operator

- SQL uses the
  Logical operator: AND, OR, and NOT

Find all loan numbers for loans made at the Perryridge branch with loan amounts greater that $1200.

```
SELECT loan-no
FROM loan
WHERE branch-name='Perryridge';
```

- Find the account number of those account with balance between $20,000 and $100,000,

```
SELECT accountno
FROM account
WHERE balance >=20000 and balance <=100000
```

Retrieval using BETWEEN

- A BETWEEN condition determines whether the value of one expression is in an interval defined by two other expressions.
- SELECT * FROM employees WHERE salary BETWEEN 2000 AND 3000
- Find the loan number of those loan with loan amount between $90,000 and $100,000,

```
SELECT loan-no
FROM loan
WHERE amount between 90000 and 100000
```

- NOT BETWEEN exp1 AND expr2

Retrieval using IN

- An IN determines whether the value of one expression present in the specified list.

- Test-expression [NOT] IN (constant1, constan2……)
- Selecting Data from Specified Departments – IN

- SELECT FIRST_NAME, LAST_NAME, DEPARTMENT_ID FROM EMPLOYEES WHERE DEPARTMENT_ID IN (100, 110, 120);

Retrieval using IN

```
SELECT *
FROM suppliers
WHERE supplier_name in ('Apple', 'HP', 'Microsoft');
```

Above query will return all rows where the supplier_name is either Apple, HP, or Microsoft.

List all the loans taken in SBI or BOI.

```
SELECT * FROM loan
WHERE branch-name='SBI' OR branch-name='BOI';
OR
SELECT * FROM loan
WHERE branch-name IN ('SBI' , 'BOI');
```

Retrieval using LIKE

- SQL specifies strings by enclosing them in single quotes.
- The most commonly used operation on strings is pattern matching using the operator LIKE.
- The **SQL LIKE** operator is used in a WHERE clause to look for a particular pattern in a column.
- column-name [NOT] LIKE pattern;

Retrieval using LIKE

- Patterns are case sensitive;
- Percent (%)- matches any substring
- Underscore(_)- matches any character

- 'Perry%' matches any string beginning with "Perry".
- '%idge%' matches any string containing "idge" as a substring, for example, 'Perryridge', 'Rock Ridge', 'Mianus Bridge', and 'Ridgeway'.
- '_ _ _' matches any string of exactly three characters.
- '_ _ _%' matches any string of at least three characters.

- Find the names of all customers whose street address contain character ‘Main’.”

```
SELECT customer-name
FROM  customer
 WHERE customer-street LIKE ‘%Main%’;
```

- Find the details of all customers whose name start with ‘v’.

```
SELECT *
FROM customer
WHERE customer-name LIKE ‘v%’;
```

- List all Accounts where the Bank_Branch begins with a ‘C’ and has ‘a’ as the second character

```
SELECT *
FROM  account
 WHERE branch-name LIKE ‘Ca%’;
```

- List all Loans where the Loan_Branch name has ‘a’ as the second character and has at least 3 character.

```
SELECT *
FROM loan
WHERE branch-name LIKE ‘_a_%’;
```

- We define the escape character for a like comparison using the escape keyword.
  - like 'ab\%cd%' escape '\' matches all strings beginning with "ab%cd".
  - like 'ab\\cd%' escape '\' matches all strings beginning with "ab\cd".

Retrieval using IS NULL

- An IS NULL determines whether the value present for column is null.
- Indicate absence of information about the value of an attribute.

- column-name [NOT] IS NULL;

- Find all loan numbers that appear in the loan relation with null values for amount.

```
SELECT loan-number
FROM loan
WHERE  amount IS NULL;
```

SQL -Ordered Results (ORDER BY)

- The ORDER BY clause is used to sort the records in result set.
- The ORDER BY clause can only be used with SELECT statements.

- SELECT Column_list FROM table
  [WHERE condition]
  ORDER BY column [ASC | DESC] ;

- Default order is ASC

Find bank accounts with a balance under $700 with order results in increasing order of bank balance.

```
SELECT account_number, balance
FROM account
WHERE balance < 700
ORDER BY balance;
```

Retrieve a list of all bank branch details, ordered by branch city, with each city's branches listed in reverse order of asset holdings

```
SELECT * FROM branch
ORDER BY branch_city ASC, assets DESC;
```

Set Operations

- need to combine the results from two or more SELECT statements.
- SQL enables us to handle these requirements by using set operations.
- The result of each SELECT statement can be treated as a set, and SQL set operations can be applied on those sets to arrive at a final result.
- Oracle SQL supports following five set operations:
- UNION UNION ALL
- INTERSECT INTERSECT ALL
- MINUS

- <component query>
  {UNION | UNION ALL | MINUS | INTERSECT| INTERSECT ALL}
  <component query>

- UNION
- Combines the results of two SELECT statements into one result set, and then eliminates any duplicate rows from that result set.
- Union is like an “OR” operation

- UNION ALL
- Combines the results of two SELECT statements into one result set and it retain all duplicates

- Union - Restrictions
- The SELECT statements must contain the same number of columns

- Data type
  - Each column in the first table must be the same as the data type of the corresponding column in the second table.
  - Data width and column name can differ

- Neither of the two tables can be sorted with the ORDER BY clause.
  - Combined query results can be sorted

- INTERSECT
- Returns only those rows that are returned by each of two SELECT statements.
- It retrieves those tuples which are present in both relation
- An intersection is an AND operation

- MINUS
- Takes the result set of one SELECT statement, and removes those rows that are also returned by a second SELECT statement.
- It retrieves rows which are present in 1 but not in 2

- Find all customers having a loan, an account, or both at the bank

  SELECT customer-name FROM depositor
  UNION
  SELECT customer-name FROM borrower

- Find all customers who have both a loan and an account at the bank

  SELECT customer-name FROM depositor
  INTERSECT
  SELECT customer-name FROM borrower

- Find all customers who have an account but no loan at the bank.

SELECT customer-name FROM depositor
MINUS
SELECT customer-name FROM borrower

Aggregate Functions

- Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value
- Aggregate functions are used in place of column names in the SELECT statement

- MIN
- MAX
- AVERAGE
- SUM
- COUNT

Aggregate function - MIN

- Returns the smallest value that occurs in the specified column.
- Column need not be numeric type.
- MIN ignores any null values.

- SELECT MIN(column name/ expression) FROM Table_name;

Selecting Minimum account balance;

```
SELECT Min(balance) as MinimumBalance
FROM account;
```

Aggregate function - MAX

- Returns the largest value that occurs in the specified column.
- Column need not be numeric type.
- MIN ignores any null values.

- SELECT MAX(column name/ expression) FROM Table_name;

Selecting Maximum loan amount

SELECT MAX(amount) as MaximumAmount
FROM loan;

Aggregate function - average

- Returns the average of all the values in the specified column. SQL AVG() ignores Null Values.
- Column must be numeric data type

- **Syntax of SQL AVG()**
- AVG ([DISTINCT] column-name/ expression)

- **Example 1 of SQL AVG()**

  List the average account balance of customers

  SELECT AVG(balance) as "Average Bal"

  FROM account;

- **Example 2 of SQL AVG()**

  Below is the test_table

Select AVG(Numb) from test_table

Result:
2

Select AVG(distinct Numb) from test_table

Result:
1

ID	Numb
1	1
2	1
3	1
4	2
5	2
6	2
7	3
8	3
9	4
10	1

Aggregate function - sum

- **SQL SUM()** function returns the sum of numeric column.
- *SQL SUM() ignores Null Values*
- **Syntax of SQL SUM()**
- SUM ([DISTINCT] column-name / expression)
- **Example 1 of SQL SUM()**

  Find the sum of loan amount of bank

  SELECT SUM(amount) as

  "Totalamount"

  FROM loan;

- **Example 2 of SQL SUM()**

  Below is the test_table

Select SUM(all Numb) from test_table

Result:
20

Select SUM(distinct Numb) from test_table

Result:
10

ID	Numb
1	1
2	1
3	1
4	2
5	2
6	2
7	3
8	3
9	4
10	1

Aggregate function- count

- count returns the number of tuples returned by the query as a number.
- COUNT([DISTINCT] column-name/ expression)

List the total number of customers

```
SELECT COUNT(*)
FROM customer;
```

- Count(*) = No of rows
- Count(ColumnName) = No. of rows that do not have NULL Value

Aggregate function- count

List the total number of account holder at SBI Branch

```
SELECT COUNT(*)
FROM account
WHERE branch-name='SBI';
```

List the total number of unique customer city

```
SELECT COUNT(DISTINCT customer-city)
FROM customer;
```

SQL- Using GROUP BY

- The aggregate function not only to a single set of tuples, but also to a group of sets of tuples;
- Specifies how to report the output of the query.
- Allows one to define a subset of the values of a particular field and to apply an aggregate function to the subsets.
- Related rows can be grouped together by GROUP BY clause by specifying a column as a grouping column.
- GROUP BY is associated with an aggregate function

SQL Group By

- The attribute or attributes given in the group by clause are used to form groups
- Tuples with the same value on all attributes in the group by clause are placed in one group.
- **Example 1 of SQL Group BY**

  Calculate the Find the average account balance at each branch

accountno	branchname	balance
A101	SBI	$1500
A102	BOI	$2500
A103	HDFC	$3000
A104	SBI	$7000
A105	BOI	$5000

- First, we need to make sure we select the branch name as well as average balance.

```
SELECT branchname, AVG (balance)
FROM account
```

Second, we need to make sure that all the balnce figures are grouped by branchnames.

```
SELECT branchname, AVG (balance)
FROM account
GROUP BY branchname;
```

The Result is:

branchname	AVG(balance)
SBI	$4250
BOI	$3750
HDFC	$3000

- **Example 2 of SQL Group BY**

  SELECT COUNT (*) FROM customer
- The above statement returns the total number of rows in the table.
- We can use **GROUP BY** to count the number of customer in each city.
- With **GROUP BY**, the table is split into groups by city, and COUNT (*) is applied to each group in turn.

  SELECT customercity, COUNT(*)

  FROM customer

  GROUP BY customercity

- we have a table name Orders.

  Orders (O_Id, OrderDate, OrderPrice, Customer)

- we want to find the total sum (total order) of each customer.

  SELECT Customer,SUM(OrderPrice)

  FROM Orders

  GROUP BY Customer

- Returns a list of Department IDs along with the sum of their sales for the date of January 1, 2000.

```
SELECT DeptID, SUM(SaleAmount)
FROM Sales
WHERE SaleDate = '01-Jan-2000'
GROUP BY DeptID
```

- Calculate the total sales for each store

```
SELECT store_name, SUM (Sales)
FROM Store_Information
GROUP BY store_name
```

- **Example 2 of SQL MIN()**

  Selecting Minimum salary for each department:

  SELECT Dept, Min(salary) as "Minimum Salary"
  FROM EMP
  GROUP BY Dept;

- **SQL MIN()** ignores any null values.

  With character data columns, **SQL MIN()** finds the value that is lowest in the sort sequence.

- **Example 2 of SQL MAX()**

  Selecting Maximum salary for each department:

  SELECT Dept, MAX(salary) as "Maximum Salary"
  FROM EMP
  GROUP BY Dept;

  SQL MAX() ignores any null values.

  For character columns, **SQL MAX()** finds the highest value in the collating sequence.

SQL- Retrieval using Having

- The **SQL HAVING** clause allows us to restrict the data that is sent to the GROUP BY clause.
- Group functions cannot be used in the WHERE clause.
- SQL statement can have both a WHERE clause and an HAVING clause.
- WHERE filters data before grouping and HAVING filters the data after grouping.
- Used to specify condition on group
- The Having condition has to be based on some column that appears in the select list

- A WHERE clause is useful in both grouped and ungrouped queries,
- while a HAVING clause should appear only immediately after the GROUP BY clause in a grouped query.
- The HAVING clause can use aggregate functions in its predicate

- SELECT G1,G2,..., F1(A1),F2(A2),...
  FROM r1,r2,...
  WHERE PW
  GROUP BY G1,G2,...
  HAVING PH

- Find those branches where the average loan amount is more than Rs. 120000.

```
SELECT branch_name, avg(amount)
FROM loan
GROUP BY branch_name
HAVING avg(amount) > 120000
```

- Find the average salary for each department that has either more than 1 employee or starts with a “To”:

```
SELECT Dept, AVG(Salary) as avgsal
FROM Employee
GROUP BY Dept
HAVING COUNT(Name) > 1 OR Dept LIKE “To%”
```

- Find average account balance at each branch with average account balance more than 5000.

```
SELECT branch_name, avg(balance)
FROM account
GROUP BY branch_name
HAVING avg(balance) > 5000;
```

- Workforce (workforceno, name, position, salary, email, dcenterno)
- For each distribution center with more than one member of workforce, find the number of workforce working in each of the centers and the sum of their salaries.

```
SELECT dCenterNo, COUNT(workforceNo) AS
totalworkforce,
SUM(salary) AS totalSalary
FROM workforce
GROUP BY dCenterNo
HAVING COUNT(workforceNo) > 1
ORDER BY dCenterNo;
```

- SQL Joins

- Cartesian Product
- Inner Join
- Left outer join
- Right outer join
- Full outer join
- Natural join

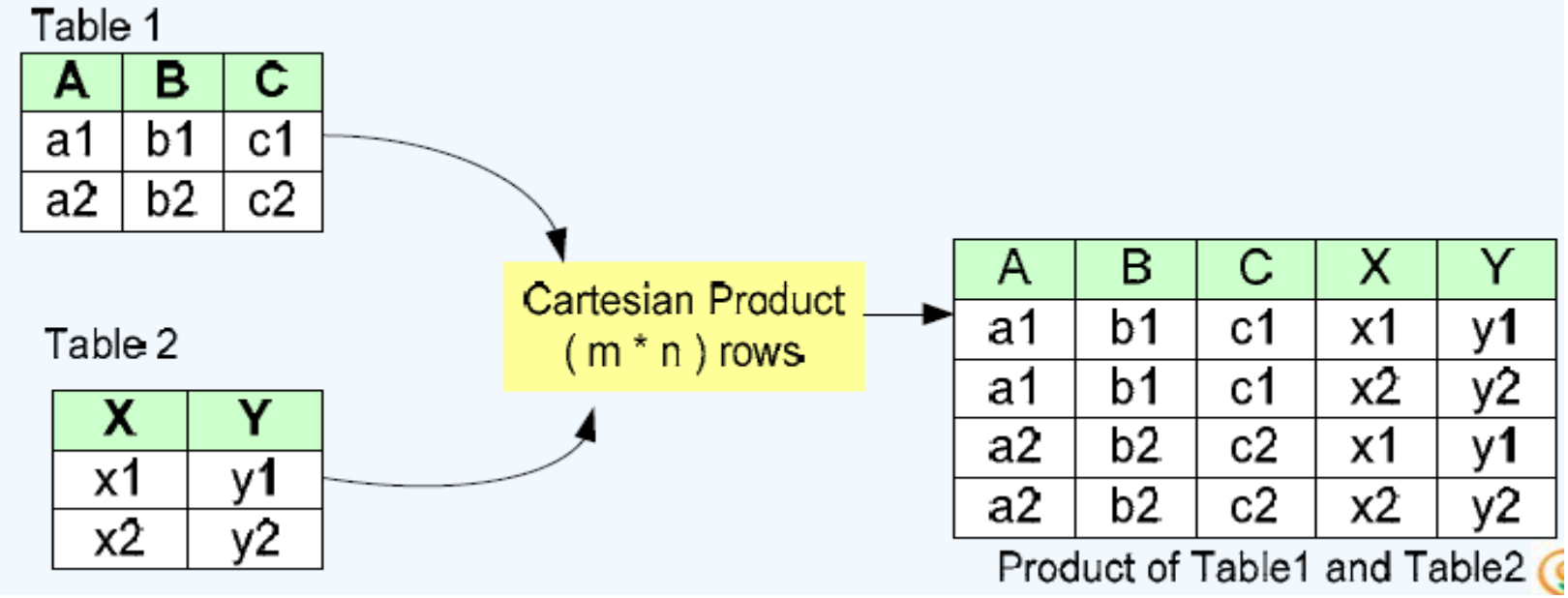
Cartesian Product

- When we join every row of a table to every row of another table we get Cartesian Product
- Returns All rows from first table, Each row from the first table is combined with all rows from the second table
- Cross join – every row of one table is matched with every row of another table.
- Cartesian join and cross join are same
- If T1 and T2 are two sets then
- T1 X T2
- Select * from Table1,Table2;

- SELECT * FROM EMP CROSS JOIN DEPT;

- SELECT * FROM EMP, DEPT;

- In first statement it is specified explicitly
- Second one is implicit



SQL Inner Join – Simple Join

- use a comparison operator to match rows from two tables
- based on the values in common columns from each table.
- An inner join between two (or more) tables is the Cartesian product that satisfies the join condition in the WHERE clause
- For example, retrieving all rows where the student identification number is the same in both the students and courses tables.

The "Consumers" table:

P_Id	LastName	FirstName	Address	City
1	Kumar	Ram	Delhi	AAA
2	Singh	Laxman	Chandigarh	AAA
3	Sharma	Sameer	Ambala	BBB

The "Orders" table

O_Id	OrderNo	P_Id
1	12355	3
2	12356	3
3	12357	1
4	24562	1
5	34764	15

- If we want to list all the Consumers with any orders.
- We use the following SELECT statement:

- SELECT Consumers.LastName, Consumers.FirstName, Orders.OrderNo
  FROM Consumers INNER JOIN Orders
  ON Consumers.P_Id=Orders.P_Id
  ORDER BY Consumers.LastName

- SELECT Consumers.LastName, Consumers.FirstName, Orders.OrderNo FROM Consumers, orders
  WHERE Consumers.P_Id=Orders.P_Id
  ORDER BY Consumers.LastName

- The result-set will look like this

LastName	FirstName	OrderNo
Kumar	Ram	12357
Kumar	Ram	24562
Sharma	Sameer	12355
Sharma	Sameer	12356

When to use inner join

Use an inner join when you want to match values from both tables.

Why to use Inner Joins:

Use inner joins to obtain information from two separate tables and combine that information in one result set.

SQL Outer Join

- Outer joins can be a left, a right, or full outer join
- **LEFT JOIN** or **LEFT OUTER JOIN**
  The result set of a left outer join includes all the rows from the left table specified in the LEFT OUTER clause, not just the ones in which the joined columns match.
- When a row in the left table has no matching rows in the right table,
- the associated result set row contains null values for all select list columns coming from the right table.

The "Consumers" table:

P_Id	LastName	FirstName	Address	City
1	Kumar	Ram	Delhi	AAA
2	Singh	Laxman	Chandigarh	AAA
3	Sharma	Sameer	Ambala	BBB

The "Orders" table

O_Id	OrderNo	P_Id
1	12355	3
2	12356	3
3	12357	1
4	24562	1
5	34764	15

- If we want to list all the Consumers and their orders - if any, from the tables above.
- We use the following SELECT statement:

```
SELECT Consumers.LastName,
Consumers.FirstName, Orders.OrderNo
FROM Consumers
LEFT JOIN Orders
ON Consumers.P_Id=Orders.P_Id
ORDER BY Consumers.LastName
```

- The result-set will look like this:

LastName	FirstName	OrderNo
Kumar	Ram	22456
Kumar	Ram	24562
Sharma	Sameer	77895
Sharma	Sameer	44678
Singh	Laxman	

The LEFT JOIN keyword returns all the rows from the left table (Consumers), even if there are no matches in the right table (Orders).

- **RIGHT JOIN** or **RIGHT OUTER JOIN**
- A **right outer join** is the reverse of a left outer join.
- All rows from the right table are returned.
- Null values are returned for the left table any time a right table row has no matching row in the left table.

- **FULL JOIN** or **FULL OUTER JOIN**
  A **full outer join** returns all rows in both the left and right tables.
- Any time a row has no match in the other table,
- the select list columns from the other table contain null values.
- When there is a match between the tables,
- the entire result set row contains data values from the base tables.

- **FULL JOIN**

Table P

p_id	LastName	FirstName	City
1	Dhall	Sachin	Delhi
2	Gupta	Pankaj	Bangalore
3	Kumar	Sanjeev	Chandigarh

Table O

o_id	OrderNo	p_id
1	111	3
2	222	3
3	333	14
4	444	2
5	555	2

- SELECT P.LastName, P.FirstName, O.OrderNo
  FROM P
  FULL JOIN O
  ON P.P_Id=O.P_Id
  ORDER BY P.LastName

LastName	FirstName	OrderNo
NULL	NULL	333
Dhall	Sachin	NULL
Gupta	Pankaj	444
Gupta	Pankaj	555
Kumar	Sanjeev	111
Kumar	Sanjeev	222

- Natural Join

1 In Inner join we give conditions explicitly

- Natural join takes condition implicitly

2 Colum names in both tables must be same for natural join

- In inner join column names can be different as we give conditions explicitly

3 Repeated columns are avoided to display in natural join

```
SELECT *  FROM foods NATURAL JOIN company;
SELECT * FROM company   INNER JOIN foods ON
company.company_id = foods.company_id;
```

item_id	item_name	item_unit	company_id
1	Chex Mix	Pcs	16
6	Cheez-lt	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15

company_id	company_name	company_city
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

- Inner Join

COMPANY_ID	COMPANY_NAME	COMPANY_CITY	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
15	Jack Hill Ltd	London	6	Cheez-It	Pcs	15
15	Jack Hill Ltd	London	2	BN Biscuit	Pcs	15
17	Foodies.	London	3	Mighty Munch	Pcs	17
15	Jack Hill Ltd	London	4	Pot Rice	Pcs	15

4 rows returned in 0.00 seconds CSV Export

- Natural Join

COMPANY_ID	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_NAME	COMPANY_CITY
16	1	Chex Mix	Pcs	Akas Foods	Delhi
15	6	Cheez-It	Pcs	Jack Hill Ltd	London
15	2	BN Biscuit	Pcs	Jack Hill Ltd	London
17	3	Mighty Munch	Pcs	Foodies.	London
15	4	Pot Rice	Pcs	Jack Hill Ltd	London
18	5	Jaffa Cakes	Pcs	Order All	Boston

6 rows returned in 0.56 seconds

- SQL Examples
- Given the employee database composed of following tables.

employee (employee-name, street, city)
works (employee-name, company-name, salary)
company (company-name, city)
manages (employee-name, manager-name)

- Give an expression in SQL for each of the following queries

- Find the names of all employee who work for First Bank Corporation

- Find the names of all employee who work for First Bank Corporation

select employee-name
from works
where company-name = ’First Bank Corporation’

- Find the names and cities of residence of all employees who work for First Bank Corporation.

- Find the names and cities of residence of all employees who work for First Bank Corporation.

select e.employee-name, city
from employee e, works w
where w.company-name = ’First Bank Corporation’ **and**
w.employee-name = e.employee-name

- Find all employees in the database who live in the same cities as the companies for which they work.

- Find all employees in the database who live in the same cities as the companies for which they work.

```
select e.employee-name
from employee e, works w, company c
where e.employee-name = w.employee-name
and e.city = c.city and
w.company -name = c.company -name
```

- Find all employees in the database who live in the same cities and on the same streets as do their managers.

- Find all employees in the database who live in the same cities and on the same streets as do their managers.

select P.employee-name
from employee P, employee R, manages M
where P.employee-name = M.employee-name **and**
M.manager-name = R.employee-name **and**
P.street = R.street **and** P.city = R.city

- Find all employees in the database who do not work for First Bank Corporation.

- Find all employees in the database who do not work for First Bank Corporation.

```
select employee-name
from works
where company-name = ’First Bank Corporation’
```

- Assume that the companies may be located in several cities.
- Find all companies located in every city in which Small Bank Corporation is located

- Assume that the companies may be located in several cities.
- Find all companies located in every city in which Small Bank Corporation is located

```
select T.company-name
from company T
where (select R.city
from company R
where R.company-name = T.company-name)
contains
(select S.city
from company S
where S.company-name = 'Small Bank Corporation')
```

- Find the names, street addresses, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.

- Find the names, street addresses, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.

```
select *
from employee
where employee-name in
(select employee-name
from works
where company-name = ’First Bank Corporation’ and
    salary >= 10000)
```

- Modify the database so that Jones who was working for City Bank now works for First Bank Corporation.

- Modify the database so that Jones who was working for City Bank now works for First Bank Corporation.

update *works*
set company-name = ’ First Bank Corporation’
where *person-name* = ’Jones’

- SQL Examples
- Given the Academic Institute database composed of following tables.

person (pid, pname, paddress, uid)
UIDData(uid, age)
student (rollno, pid, class, dept)
faculty (fid, pid, dept)
Studentattendance(rollno, perattendance);
Studentmarks(rollno, subject, marks)
Studentfee(rollno, feepaid, feedues)
Lectures(fid, class, subject)

- Give an expression in SQL for each of the following queries

- Find the faculty who is also studying in M.Tech
- Find address of students whose attendance is less than 70%
- Find all the students who are eligible for voting
- Find the total fee paid by T.E.CSE students
- Find the address of students who have backlog subjects
- Find the list of students who are supposed to attend DBE lectures.
- Find all the teachers who teaches to student with uid 123456789101