CSL302 - Database Engineering

Unit 2

Structured Query Language (SQL)

Introduction to SQL

- SQL stands for Structured Query Language
- It is the language for accessing a relational database.
- SQL provides a set of statements for storing and retrieving data to and from a relational database.
- It has gained steadily popularity
- Other languages have been put forth, but SQL is accepted as the standard language for almost all relational database implementations including Oracle.

- SQL is different from other programming languages because it is nonprocedural.
- Unlike programs in other languages, where you specify the sequence of steps to be performed
- A SQL program only expresses the desired result.
- The responsibility for determining how the data will be processed in order to generate the desired result is left to the database management system.
- The nonprocedural nature of SQL makes it easier to access data in application programs.

- SQL is the interface we use to access the data stored in database.
- SQL allows us to create database structures such as tables (to store data), views, and indexes.
- SQL allows us to insert data into the database, and to retrieve that stored data in a desired format
- SQL allows us to modify, delete, and manipulate stored data.

- SQL is the key to everything we do with the database.
- It’s important to know how to get the most out of that interface.
- Mastery over the SQL language is one of the most vital requirements of a database developer or database administrator.

History of SQL

- In 1973, Michael Stonebraker and Eugene Wong (both then at UC Berkeley) made the decision to research relational database systems.
- The project was called INGRES (Interactive Graphics and Retrieval System), and successfully demonstrated a relational model could be efficient and practical.
- INGRES worked with a query language known as QUEL

- IBM developed ‘Structured Query Language’ (SQL) in 1974, which was more advanced
- SQL became ANSI and OSI standards in 1986 and 1987.
- SQL quickly replaced QUEL as the more functional query language.
- The relational model is still in use by many people in the market.
- Most people use the sequel pronunciation just because it flows a bit easier linguistically.

SQL Command Categories

- SQL Statements can be categorized into
- Data Definition Language (DDL) Statements
- Data Manipulation Language (DML) Statements
- Data Control Language (DCL)
- Transaction Control Statements
- Session Control Statements

Data Definition Language (DDL) Statements

- 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

Data Types

- Numeric Data Types
- The NUMBER data type stores fixed and floating-point numbers up to 38 digits of precision
- It can store -
- Positive numbers in the range 1×10^{-130} to $9.99...9 \times 10^{125}$ with up to 38 significant digits
- Negative numbers in the range 1×10^{-130} to $9.99...9 \times 10^{125}$ with up to 38 significant digits
- Zero

Input Data	Specified As	Stored As
7,456,123.89	NUMBER	7456123.89
7,456,123.89	NUMBER(*,1)	7456123.9
7,456,123.89	NUMBER(9)	7456124
7,456,123.89	NUMBER(9,2)	7456123.89
7,456,123.89	NUMBER(9,1)	7456123.9
7,456,123.89	NUMBER(6)	(not accepted, exceeds precision)
7,456,123.89	NUMBER(7,-2)	7456100

- String Data Types - CHAR
- CHAR data type stores fixed-length character strings.
- Need to specify maximum string length between 1 and 2000 bytes
- The default is 1 byte.
- If value to insert is shorter, then the value is blank-padded to the fixed length.
- If a value is too large, Oracle Database returns an error.
- CHAR(20)

- VARCHAR
- VARCHAR data type stores variable-length character strings.
- Need to specify maximum string length between 1 and 2000 bytes
- The default is 1 byte.
- If value to insert is shorter, then allocate only required space and not the fixed amount of space.
- If a value is too large, Oracle Database returns an error.
- VARCHAR(20)

- VARCHAR2
- VARCHAR2 data type stores variable-length character strings.
- Need to specify maximum string length between 1 and 4000 bytes
- The default is 1 byte.
- If value to insert is shorter, then allocate only required space and not the fixed amount of space.
- If a value is too large, Oracle Database returns an error.
- VARCHAR2(20)

- BLOB – Binary Large Objects
- Used for binary data
- videos, images, documents, other

- CLOB – Character Large Objects
- Used for large text data (text)

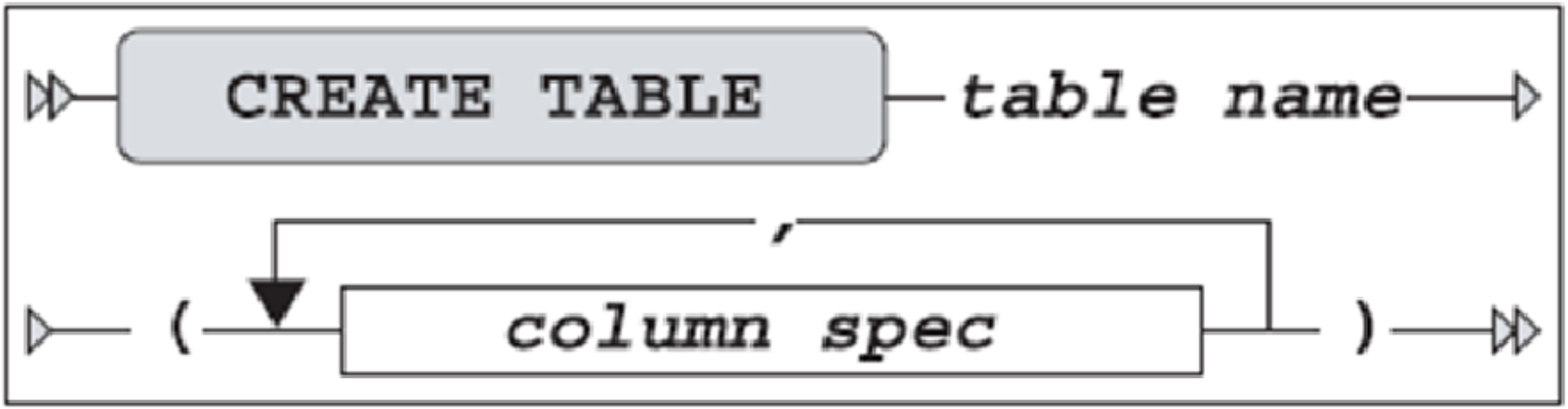
- Maximum size on MySQL 2GB
- Maximum size on Oracle 128TB

Oracle SQL

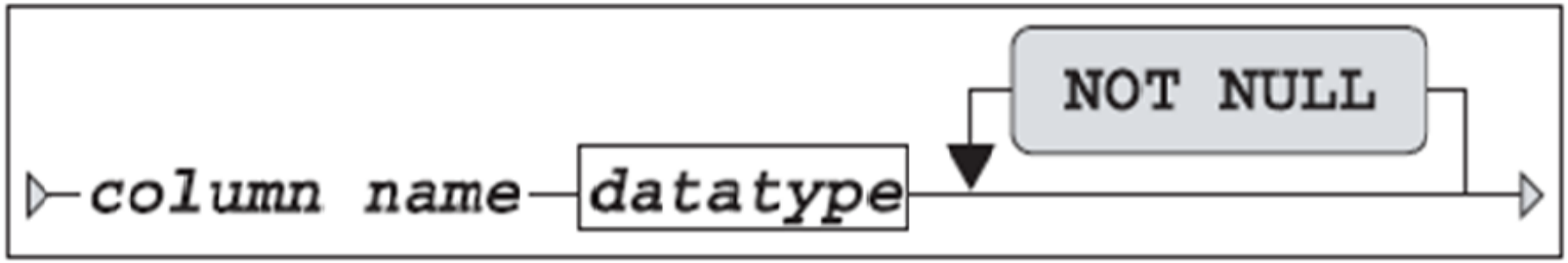
- Create Account
- https://profile.oracle.com/myprofile/account/create-account.jspx

- Login to livesql.oracle.com
- https://livesql.oracle.com/

CREATE



CREATE TABLE basic command syntax



Column specification syntax diagram

CREATE

- Used to create new database object like table, view, index.
- CREATE TABLE *name_of_table (attribute_name data_type [(size)] [constraint],….)*

- CREATE TABLE PERSON (PID NUMBER PRIMARY KEY, PNAME VARCHAR2(50), PADDRESS VARCHAR2(50), AadharID NUMBER(12), MOBILENO NUMBER(10))

- To get the details of table created
- DESCRIBE *name_of_table*

- select * from all_tab_columns where table_name='*name_of_table*'

- SELECT * FROM *name_of_table*

INSERT

- Used to insert data in table

- Insert values for All Attributes
- INSERT INTO *name_of_table* VALUES (*attribute1_value, [attribute2_value,...]*)

- INSERT INTO PERSON VALUES(111, 'ABC','DDD', 123456123456, 9898989898)

INSERT

- Insert values for some Attributes

- INSERT INTO *name_of_table* (*attribute1, [attribute2,...]*)VALUES (*attribute1_value, [attribute2_value,...]*)

- INSERT INTO PERSON(PID, PNAME) VALUES(112, 'ABC')

DROP

- To drop database object
- Syntax:
- DROP <OBJECT> <object name>

- Example:
- DROP TABLE STUDENTS

TRUNCATE

- Remove all records from a table, including all spaces allocated for the records
- Syntax:
- TRUNCATE TABLE <table name>

- Example:
- TRUNCATE TABLE STUDENTS

RENAME

- Rename an database object
- Syntax:
- RENAME <old object name> TO <new object name>

- Example:
- RENAME STUDENTSS to STUDENT

Constraints

- Restrictions on attribute values
- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- PRIMARY KEY
- FOREIGN KEY
- (*attribute_name data_type [(size)] [constraint]*,….)
- (… CONSTRAINT *name_of_constraint type_of_constraint* (*attributes*))

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 constraint can be applied on any number of attributes separately.

- CREATE TABLE STAFF(STID NUMBER NOT NULL, STNAME VARCHAR2(50) NOT NULL)

- INSERT INTO STAFF(STNAME) VALUES ('AAA')

- ORA-01400: cannot insert NULL into ….

UNIQUE Contraint

- 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 STUDENT2 (CLAS VARCHAR2(10), ROLLNO NUMBER UNIQUE, SNAME VARCHAR(50))
- UNIQUE constraint can be applied on any number of attributes separately or in combination

- CREATE TABLE STUDENT1 (CLAS VARCHAR2(10), ROLLNO NUMBER, SNAME VARCHAR(50), CONSTRAINT unique_st1 UNIQUE (CLAS, ROLLNO))

- INSERT INTO STUDENT1 VALUES ('TY', 11, ‘ABC')
- INSERT INTO STUDENT1 VALUES ('TY', 11, 'XYZ')
- ORA-00001: unique constraint (SYSTEM.UNIQUE_ST1) violated

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 CITIZEN (PID NUMBER, CITY VARCHAR2(50) DEFAULT 'ICHALKARANJI')

- INSERT INTO CITIZEN VALUES(1111, ‘SANGLI')
- INSERT INTO CITIZEN (PID) VALUES (2222)
- INSERT INTO CITIZEN VALUES(3333, 'ICHALKARANJI')

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.

- CREATE TABLE CITIZENS (PID NUMBER, CITY VARCHAR2(50), CONSTRAINT chk_citizen CHECK (PID>1000 AND CITY='Ichalkaranji'))

- INSERT INTO CITIZENS VALUES(999, 'ICHALKARANJI')
- INSERT INTO CITIZENS VALUES(9999, 'SANGLI')
- ORA-02290: check constraint (SYSTEM.CHK_CITIZEN) violated

PRIMARY KEY

- If a single attribute forms a primary key
- CREATE TABLE *name_of_table* (*attribute_name data_type* PRIMARY KEY,…)

- CREATE TABLE *name_of_table* (…., CONSTRAINT *name_of_constraint* PRIMARY KEY (*attribute_names*))

PRIMARY KEY

- If a single attribute forms a primary key
- CREATE TABLE PERSON (PID NUMBER PRIMARY KEY, PNAME VARCHAR2(50), PADDRESS VARCHAR2(50), AadharID NUMBER(12), MOBILENO NUMBER(10))
- CREATE TABLE PERSONS (PID NUMBER, PNAME VARCHAR2(50), PADDRESS VARCHAR2(50), AadharID NUMBER(12), MOBILENO NUMBER(10), CONSTRAINT PK_PERSON PRIMARY KEY (PID))

- If more than one attributes form a primary key
- CREATE TABLE PERSONS (PID NUMBER, PNAME VARCHAR2(50), PADDRESS VARCHAR2(50), AadharID NUMBER(12), MOBILENO NUMBER(10), CONSTRAINT PK_PERSON PRIMARY KEY (PID, PNAME))

FOREIGN KEY

- Referential Integrity
- Values of the attributes are restricted to only those values of key attributes in another table
- PERSON – PID
- STUDENT
- FACULTY
- STAFF

- CREATE TABLE *name_of_table* (*attribute_name data_type* REFERENCES *name_of_table_to_refer* (*attribute_name_to_refer)*,…)

- CREATE TABLE *name_of_table* (….., CONSTRAINT *name_of_constraint* FOREIGN KEY (*attribute_names*) REFERENCES *name_of_table_to_refer* (*attribute_name_to_refer*))

- CREATE TABLE PERSON (PID NUMBER PRIMARY KEY, PNAME VARCHAR2(50), PADDRESS VARCHAR2(50), AadharID NUMBER(12), MOBILENO NUMBER(10))
- CREATE TABLE STUDENT (PRN NUMBER PRIMARY KEY, SCLASS VARCHAR2(10), PID NUMBER REFERENCES PERSON(PID))
- CREATE TABLE FACULTY (FID NUMBER PRIMARY KEY, DESIGNATION VARCHAR2(10), PID NUMBER REFERENCES PERSON(PID))

- INSERT INTO STUDENT VALUES(111, 'SY', 1111)

- ORA-02291: integrity constraint (SYSTEM.SYS_C007301) violated - parent key not found

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

```
CREATE TABLE supplier
(       supplier_id       number(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        number(10)      not null,
        supplier_id       number(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 lookup (p VARCHAR2(3),
  CONSTRAINT fk_p_ref1
  FOREIGN KEY (p) REFERENCES ref1(id),
  CONSTRAINT fk_p_ref2
  FOREIGN KEY (p) REFERENCES ref2(id));

ALTER

- The SQL ALTER command is used to alter (modify) the definition of database object.
- ALTER TABLE command can be used to add, delete or modify columns in an existing table.
- ALTER TABLE command can be used to add and drop various constraints on an existing table.

To add a new column in an existing table

- ALTER TABLE *table_name* ADD *column_name datatype*

- CREATE TABLE STUDENT (PRN NUMBER, SNAME VARCHAR2(50))
- ALTER TABLE STUDENT ADD CGPA NUMBER(4,2)
- select * from all_tab_columns where table_name='STUDENT'

To drop a column from an existing table

- ALTER TABLE *table_name* DROP COLUMN *column_name*

- select * from all_tab_columns where table_name='STUDENT'
- ALTER TABLE STUDENT DROP COLUMN CGPA
- select * from all_tab_columns where table_name=‘STUDENT’

To change the data type of a column in an existing table

- ALTER TABLE *table_name* MODIFY *column_name datatype*

- CREATE TABLE STUDENT (PRN NUMBER, SNAME VARCHAR2(50))
- ALTER TABLE STUDENT MODIFY PRN VARCHAR(10)
- select * from all_tab_columns where table_name='STUDENT'

To add NOT NULL constraint to a column in the table

- ALTER TABLE *table_name* MODIFY *column_name* NOT NULL

- ALTER TABLE *table_name* MODIFY *column_name* NOT NULL NOVALIDATE

- CREATE TABLE STUDENT (PRN NUMBER, SNAME VARCHAR2(50))
- INSERT INTO STUDENT (PRN) VALUES ('1111')
- SELECT * FROM STUDENT
- ALTER TABLE STUDENT MODIFY SNAME NOT NULL
- ORA-01442: column to be modified to NOT NULL is already NOT NULL
  ALTER TABLE STUDENT MODIFY SNAME NOT NULL NOVALIDATE

To add UNIQUE constraint to columns in the table

- ALTER TABLE *name_of_table* ADD CONSTRAINT *name_of_constraint* UNIQUE (*attribute_names*)

- ALTER TABLE STUDENT ADD CONSTRAINT un_student UNIQUE (PRN)
- INSERT INTO STUDENT VALUES ('1111','ABC')
- ORA-00001: unique constraint violated

To add CHECK constraint to columns in the table

- ALTER TABLE *name_of_table* ADD CONSTRAINT *name_of_constraint* CHECK (*condition*) [NOVALIDATE]

- CREATE TABLE STUDENTS (PRN NUMBER, SNAME VARCHAR2(50))
- ALTER TABLE STUDENTS ADD CONSTRAINT chk_students CHECK (PRN>1000) NOVALIDATE
- INSERT INTO STUDENT VALUES (999,'ABC')
- ORA-02290: check constraint (SQL_OJYSMHHLZNHPZCYMFWPSMDYWZ. CHK_STUDENT) violated ORA-06512: at "SYS.DBMS_SQL", line 1721
- INSERT INTO STUDENTS VALUES (1111,'ABC')

To add DEFAULT constraint to a column in the table

- ALTER TABLE *name_of_table* MODIFY *column_name* DEFAULT *default_value*

- CREATE TABLE STUDENTSS (PRN NUMBER, SNAME VARCHAR2(50), CITY VARCHAR2(20))
- ALTER TABLE STUDENTSS MODIFY CITY DEFAULT 'ICHALKARANJI'
- INSERT INTO STUDENTSS(PRN) VALUES (2222)
- SELECT * FROM STUDENTSS

To add PRIMARY KEY constraint to a table

- ALTER TABLE *name_of_table* ADD CONSTRAINT *name_of_constraint* PRIMARY KEY (*attributes*)

- ALTER TABLE STUDENTSS ADD CONSTRAINT pk_studentss PRIMARY KEY (PRN)
- select * from all_tab_columns where table_name='STUDENTSS'

To add FOREIGN KEY constraint to a table

- ALTER TABLE *name_of_table* ADD CONSTRAINT *name_of_constraint* FOREIGN KEY (*attribute_names*) REFERENCES *name_of_table_to_refer* (*attribute_name_to_refer*))

- CREATE TABLE STUDENTLIBRARY (PRN NUMBER, BOOKSISSUED VARCHAR2(50))

- ALTER TABLE STUDENTLIBRARY ADD CONSTRAINT fk_studentlibrary FOREIGN KEY (PRN) REFERENCES STUDENTSS(PRN)

To DROP a constraint from a table

- ALTER TABLE *name_of_table* DROP CONSTRAINT *name_of_constraint*

- ALTER TABLE STUDENTLIBRARY DROP CONSTRAINT fk_studentlibrary

Data Manipulation Language (DML) Statements

- INSERT
  - To add records in a table
- UPDATE
  - To modify column values of existing records
- DELETE
  - To remove records from a table
- SELECT
  - To retrieve data from a table

INSERT

- Used to insert data in table

- Insert values for All Attributes
- INSERT INTO *name_of_table* VALUES (*attribute1_value, [attribute2_value,...]*)

- INSERT INTO PERSON VALUES(111, 'ABC','DDD', 123456123456, 9898989898)

INSERT

- Insert values for some of the Attributes

- INSERT INTO *name_of_table* (*attribute1, [attribute2,...]*)VALUES (*attribute1_value, [attribute2_value,...]*)

- INSERT INTO PERSON(PID, PNAME) VALUES(112, 'ABC')

Use of select in insert

- Select data from one table and insert into another table.
- INSERT INTO *name_of_table* SELECT *attribute_names* FROM *name_of_table*

- INSERT INTO STUDENT SELECT PRN, SNAME FROM STUDENTS

- INSERT INTO *name_of_table(attributes)* SELECT *attribute_names* FROM *name_of_table*

- INSERT INTO STUDENT(PRN, SNAME) SELECT PRN, SNAME FROM STUDENTS

UPDATE

- Used to modify column values of existing records
- UPDATE *table_name*

  SET column1 = value1, column2 = value2, ...

  [WHERE *condition*]

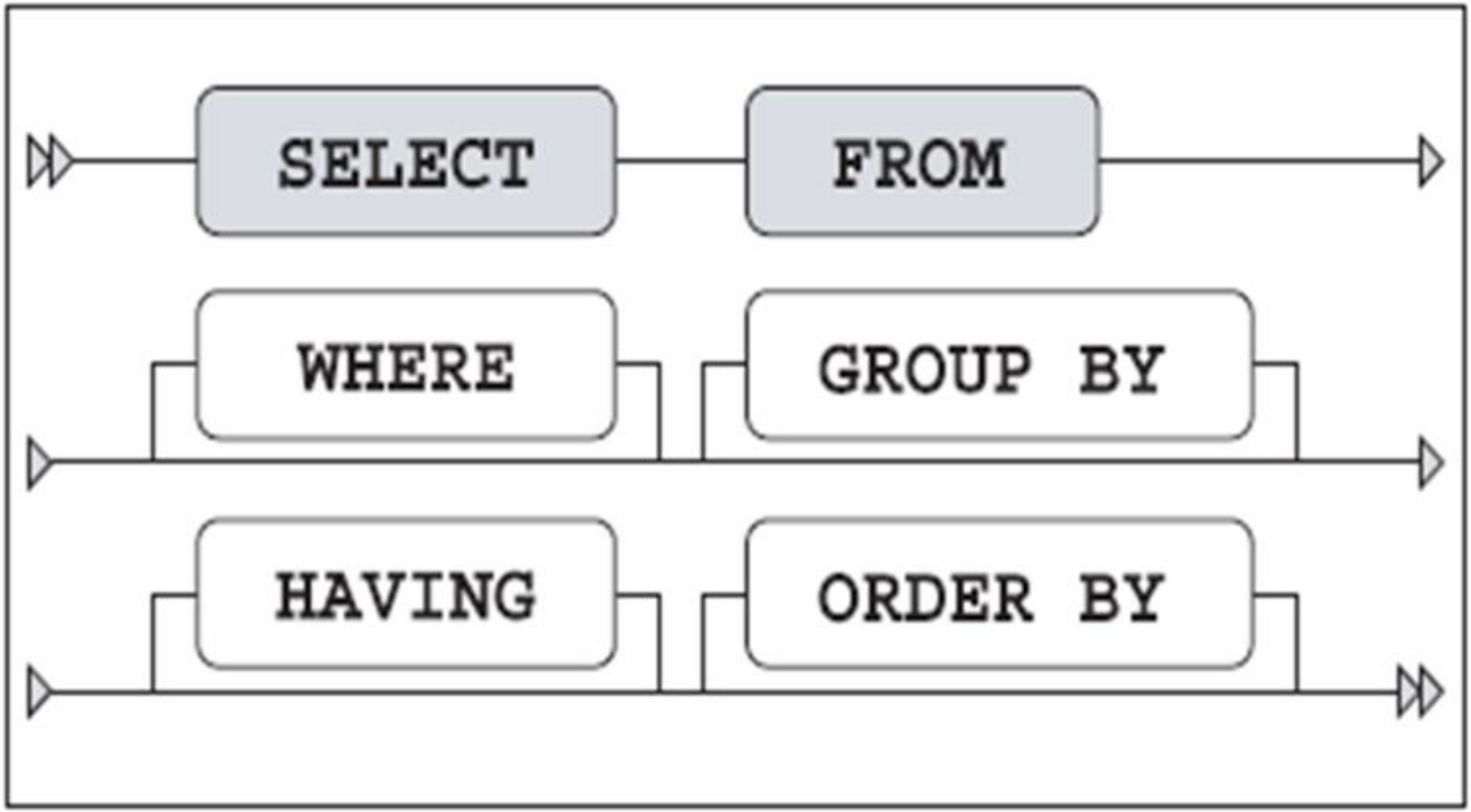
- UPDATE STUDENT SET CGPA=9.0 WHERE CGPA=8.0
- UPDATE STUDENT SET CGPA=9.5 WHERE PRN=222

DELETE

- Used to delete a one or more records from a table.
- DELETE FROM *table_name* [WHERE conditions]
- DELETE FROM STUDENT
- DELETE FROM STUDENT WHERE PRN=333

SELECT

- Used to retrieve data



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

- To retrieve all rows and all columns
- (All Attributes of All Records)

- SELECT * FROM STUDENT

- To retrieve some of the attributes of all records
- SELECT *attribute1[, attribute2,...]* FROM *table_name*
- SELECT PRN, SNAME FROM STUDENT

- Select attributes of all records and display under new headings

- SELECT PRN, SNAME Student_Name FROM STUDENT

- SELECT PRN, SNAME AS Student_Name FROM STUDENT

Conditional Data Retrieval

- Retrieve records (rows) which satisfy certain conditions
- SELECT *list_of_attributes* FROM *list_of_tables* WHERE *conditions*
- Matching certain attribute value
- SELECT * FROM STUDENT WHERE PRN=111
- SELECT * FROM STUDENT WHERE SCITY='ICHALKARANJI'

- Combine multiple conditions
- SELECT *list_of_attributes* FROM *list_of_tables* WHERE *condition1* AND *condition2*

- SELECT * FROM STUDENT1 WHERE PRN=111 AND SCITY='ICHALKARANJI'

- Combine multiple conditions
- SELECT *list_of_attributes* FROM *list_of_tables* WHERE *condition1* OR *condition2*

- SELECT * FROM STUDENT1 WHERE PRN=111 OR SCITY='ICHALKARANJI'

- Data retrieval – matching single value
- SELECT * FROM STUDENT WHERE PRN=111

- SELECT * FROM STUDENT WHERE SCITY=‘ICHALKARANJI’

IN, BETWEEN, <, >

- Data retrieval – matching with any of the given values
- Student (PRN, SNAME, CGPA, SCITY)

- Find the students living in ‘Sangli’ or ‘Miraj’
- SELECT * from STUDENT WHERE SCITY IN ('SANGLI','MIRAJ')

- Find the students having First Class
- SELECT * from STUDENT WHERE CGPA>=6.75
- SELECT * from STUDENT WHERE CGPA BETWEEN 6.75 AND 10.0
- SELECT * from STUDENT WHERE CGPA>=6.75 AND CGPA<=10.0

LIKE

- Find the Students whose name starts with A
- SELECT * from STUDENT WHERE SNAME LIKE ‘A%’
- Find the Students whose name ends with B
- SELECT * from STUDENT WHERE SNAME LIKE '%B‘
- Find the Students whose name consists of 3 characters, starts with A and ends with B
- SELECT * from STUDENT WHERE SNAME LIKE 'A_B'

- This SQL statement would return all students whose name is 5 characters long,
- where the first two characters is 'Sm' and the last two characters is 'th'.
- For example, 'Smith', 'Smyth', 'Smath', 'Smeth', etc.

```
SELECT * FROM STUDENTS
WHERE SNAME like 'Sm_th'
```

Retrieve Data from Multiple Tables – SQL Joins

- Inner Join
- Outer Join
  - Left Join / Left Outer Join
  - Right Join / Right Outer Join
  - Full Join / Full Outer Join
- Natural Join
- Cross Join / Cartesian Product

- Person (PID, PNAME, PADDRESS, PhoneNumber)
- Student (PRN, PID, CLS, CGPA)
- Faculty (FID, PID, Designation, Dept)

Inner Join

- Retrieve the records having same value for similar attributes in two tables
- Use a comparison operator to match rows from two tables
- Retrieve the data based on the values in common columns from each table.

- SELECT <*list_of_attributes*> FROM <*Table1*>
  INNER JOIN <*Table2*>
  ON *table1.attribute=table2.attribute*

- SELECT * FROM PERSON INNER JOIN STUDENT ON PERSON.PID=STUDENT.PID

- SELECT <*list_of_attributes*> FROM <*Table1*> INNER JOIN <*Table2*> USING (<*attribute*>)
- SELECT * FROM PERSON INNER JOIN STUDENT USING (PID)

- The columns listed in the USING clause must be present in both of the two tables being joined.
- The USING clause will be transformed to an ON clause that checks for equality between the named columns in the two tables.

- When to use inner join
- Use an inner join when you want to match values from both tables.
- Use inner join to obtain information from two separate tables and combine that information in one result set.

Outer Join

- Inner join retrieve the records having same value for similar attributes in both tables.
- Outer join retrieve records from one of the table and display records from another table if matches the value of similar attribute.
- Display NULL if similar attribute has no matching value.
  - Left Join / Left Outer Join
  - Right Join / Right Outer Join
  - Full Join / Full Outer Join

Left Join / Left Outer Join

- Retrieve all the rows from left table specified in Join statement.
- And matching row from right table if common attribute value matches.
- 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 selected list columns coming from the right table.

- SELECT <list_of_attributes> FROM <Table1>
  LEFT JOIN <Table2>
  ON table1.attribute=table2.attribute

- SELECT * FROM FACULTY LEFT JOIN
  STUDENT
  ON PERSON.PID=STUDENT.PID

- SELECT <list_of_attributes> FROM <Table1>
  LEFT OUTER JOIN <Table2>
  ON table1.attribute=table2.attribute

- SELECT * FROM FACULTY LEFT OUTER JOIN
  STUDENT
  ON PERSON.PID=STUDENT.PID

- SELECT <list_of_attributes> FROM
  <Table1> LEFT JOIN <Table2>
  USING (<attribute>)

- SELECT * FROM FACULTY LEFT JOIN STUDENT USING (PID)

- SELECT <list_of_attributes> FROM
  <Table1> LEFT OUTER JOIN <Table2>
  USING (<attribute>)

- SELECT * FROM FACULTY LEFT OUTER JOIN STUDENT USING (PID)

Right Join / Right Outer Join

- Retrieve all the rows from right table specified in Join statement.
- And matching row from left table if common attribute value matches.
- When a row in the right table has no matching rows in the left table the associated result set row contains null values for all selected list columns coming from the left table.

- SELECT <list_of_attributes> FROM <Table1>
  RIGHT JOIN <Table2>
  ON table1.attribute=table2.attribute

- SELECT * FROM FACULTY RIGHT JOIN
  STUDENT
  ON PERSON.PID=STUDENT.PID

- SELECT <list_of_attributes> FROM <Table1>
  RIGHT OUTER JOIN <Table2>
  ON table1.attribute=table2.attribute

- SELECT * FROM FACULTY RIGHT OUTER
  JOIN STUDENT
  ON PERSON.PID=STUDENT.PID

- SELECT <list_of_attributes> FROM
  <Table1> RIGHT JOIN <Table2>
  USING (<attribute>)

- SELECT * FROM FACULTY RIGHT JOIN STUDENT USING (PID)

- SELECT <list_of_attributes> FROM
  <Table1> RIGHT OUTER JOIN <Table2>
  USING (<attribute>)

- SELECT * FROM FACULTY RIGHT OUTER JOIN STUDENT USING (PID)

Full Join / Full Outer Join

- Retrieve all the rows from left table and right table specified in Join statement.
- 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 selected list columns coming from the right table.
- When a row in the right table has no matching rows in the left table the associated result set row contains null values for all selected list columns coming from the left table.

- SELECT <list_of_attributes> FROM <Table1>
  FULL JOIN <Table2>
  ON table1.attribute=table2.attribute

- SELECT * FROM FACULTY FULL JOIN
  STUDENT
  ON PERSON.PID=STUDENT.PID

- SELECT <list_of_attributes> FROM <Table1>
  FULL OUTER JOIN <Table2>
  ON table1.attribute=table2.attribute

- SELECT * FROM FACULTY FULL OUTER JOIN
  STUDENT
  ON PERSON.PID=STUDENT.PID

- SELECT <list_of_attributes> FROM
  <Table1> RIGHT JOIN <Table2>
  USING (<attribute>)

- SELECT * FROM FACULTY FULL JOIN STUDENT USING (PID)

- SELECT <list_of_attributes> FROM
  <Table1> RIGHT OUTER JOIN <Table2>
  USING (<attribute>)

- SELECT * FROM FACULTY FULL OUTER JOIN STUDENT USING (PID)

Natural Join

- Similar to Inner Join
- Retrieve the records having same value for similar attributes in two tables.

Difference between Inner Join and Natural Join

- In Inner join takes conditions explicitly.
- Natural join takes condition implicitly.
- Colum names in both tables must be same for natural join.
- In inner join column names can be different as conditions are given explicitly.
- Repeated columns are avoided to display in natural join.

- SELECT <*list_of_attributes*> FROM <*Table1*>
  INNER JOIN <*Table2*>
  ON *table1.attribute=table2.attribute*

- SELECT * FROM FACULTY INNER JOIN STUDENT ON PERSON.PID=STUDENT.PID
- SELECT * FROM FACULTY INNER JOIN STUDENT USING (PID)

- SELECT <*list_of_attributes*> FROM <*Table1*> NATURAL JOIN <*Table2*>

- SELECT * FROM FACULTY NATURAL JOIN STUDENT

Cross Join / Cartesian Product

- Cross Join / Cartesian Product join every row of a table to every row of another table.
- If T1 and T2 are two sets then
- Number of records in T1 CROSS JOIN T2 are
- (No. of records in T1) X (No. of records in T2)
- It does not check for common attribute values.

- SELECT <*list_of_attributes*> FROM <*Table1*>, <*Table2*>…
- SELECT * FROM PERSON, STUDENT

- SELECT <*list_of_attributes*> FROM <*Table1*> CROSS JOIN <*Table2*>
- SELECT * FROM PERSON CROSS JOIN STUDENT

SQL Aggregate Functions

- count
- average
- sum
- min
- max

count

- count returns the number of tuples returned by the query as a number
- SELECT COUNT(*) FROM EMPLOYEE
- SELECT COUNT(*) CNT FROM EMPLOYEE
- SELECT COUNT(*) AS CNT FROM EMPLOYEE
- SELECT COUNT(DISTINCT SALARY) FROM EMPLOYEE
- SELECT COUNT(ALL SALARY) FROM EMPLOYEE

avg

- avg returns the average of numeric values
- SELECT AVG(SALARY) FROM EMPLOYEE
- SELECT AVG(SALARY) AVG FROM EMPLOYEE
- SELECT AVG(SALARY) AS AVG FROM EMPLOYEE
- SELECT AVG(DISTINCT SALARY) FROM EMPLOYEE
- SELECT AVG(ALL SALARY) FROM EMPLOYEE

sum

- count returns the sum of numeric values
- SELECT SUM(SALARY) FROM EMPLOYEE
- SELECT SUM(SALARY) SUM FROM EMPLOYEE
- SELECT SUM(SALARY) AS SUM FROM EMPLOYEE
- SELECT SUM(DISTINCT SALARY) FROM EMPLOYEE
- SELECT SUM(ALL SALARY) FROM EMPLOYEE

min

- Returns minimum value in an expression

- SELECT MIN(SALARY) FROM EMPLOYEE
- SELECT MIN(SALARY) MIN FROM EMPLOYEE
- SELECT MIN(SALARY) AS MIN FROM EMPLOYEE

max

- Returns minimum value in an expression

- SELECT MAX(SALARY) FROM EMPLOYEE
- SELECT MAX(SALARY) MAX FROM EMPLOYEE
- SELECT MAX(SALARY) AS MAX FROM EMPLOYEE

‘group by’ clause

- It 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.
- Normally use a GROUP BY clause in conjunction with an aggregate expression (like SUM, COUNT etc).

- It is necessary to use group attributes in select clause when group by is applied
- E.g. DEPT, CLASS

- Individual attributes can’t be used in select clause when group by is applied
- E.g. PRN, NAME

- Find number of employee working in each department
- SELECT DEPT, COUNT(EMPID) FROM EMPLOYEE GROUP BY DEPT

- Find average salary of employee working in each department
- SELECT DEPT, AVG(SALARY) FROM EMPLOYEE GROUP BY DEPT

- Find minimum salary of employee working in each department
- SELECT DEPT, MIN(SALARY) FROM EMPLOYEE GROUP BY DEPT

- Find maximum salary of employee working in each department
- SELECT DEPT, MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT

- Find total monthly salary of each department
- SELECT DEPT, SUM(SALARY) FROM EMPLOYEE GROUP BY DEPT

‘having’ clause

- 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.

- 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.

- Group by clause can appear without having clause
- But having clause can not appear without group by clause

- Find number of employee working in each department having 3 or more employee
- SELECT DEPT, COUNT(EMPID) FROM EMPLOYEE GROUP BY DEPT HAVING COUNT(EMPID)>3

- Find average salary of employee working in each department having 3 or more employee
- SELECT DEPT, AVG(SALARY) FROM EMPLOYEE GROUP BY DEPT HAVING COUNT(EMPID)>3

- Find minimum salary of employee working in each department having 3 or more employee
- SELECT DEPT, MIN(SALARY) FROM EMPLOYEE GROUP BY DEPT HAVING COUNT(EMPID)>3

- Find maximum salary of employee working in each department having 3 or more employee
- SELECT DEPT, MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT HAVING COUNT(EMPID)>3

- Find total monthly salary of each department having 3 or more employee
- SELECT DEPT, SUM(SALARY) FROM EMPLOYEE GROUP BY DEPT HAVING COUNT(EMPID)>3

‘order by’ clause

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

  [WHERE conditions]

  ORDER BY expression [ASC | DESC];

- Default order is ASC

- Retrieve Employee Name and Salary
- SELECT EMPNAME, SALARY FROM EMPLOYEE ORDER BY SALARY
- SELECT EMPNAME, SALARY FROM EMPLOYEE ORDER BY SALARY ASC
- SELECT EMPNAME, SALARY FROM EMPLOYEE ORDER BY SALARY DESC
- SELECT EMPNAME, SALARY FROM EMPLOYEE ORDER BY SALARY DESC, DEPT ASC

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 four set operations:
- UNION ALL
- UNION
- MINUS
- INTERSECT

<component query>

{UNION | UNION ALL | MINUS | INTERSECT}

<component query>

UNION ALL

- Combines the results of two SELECT statements into one result set.

```
SELECT * FROM CUSTOMER WHERE CUST_ID
IN (SELECT CUST_ID FROM SAVING)
UNION ALL
SELECT * FROM CUSTOMER WHERE CUST_ID
IN (SELECT CUST_ID FROM LOAN)
```

UNION

- Combines the results of two SELECT statements into one result set, and then eliminates any duplicate rows from that result set.

```
SELECT * FROM CUSTOMER WHERE CUST_ID
IN (SELECT CUST_ID FROM SAVING)
UNION
SELECT * FROM CUSTOMER WHERE CUST_ID
IN (SELECT CUST_ID FROM LOAN)
```

MINUS

- Takes the result set of one SELECT statement, and removes those rows that are also returned by a second SELECT statement.

```
SELECT * FROM CUSTOMER WHERE CUST_ID
IN (SELECT CUST_ID FROM SAVING)
MINUS
SELECT * FROM CUSTOMER WHERE CUST_ID
IN (SELECT CUST_ID FROM LOAN)
```

INTERSECT

- Returns only those rows that are returned by each of the two SELECT statements.

```
SELECT * FROM CUSTOMER WHERE CUST_ID
IN (SELECT CUST_ID FROM SAVING)
INTERSECT
SELECT * FROM CUSTOMER WHERE CUST_ID
IN (SELECT CUST_ID FROM LOAN)
```

SQL Query Writing - Examples

- Given the Employee Database composed of following tables.
- Employee (EMPID, EMPName, Street, EMPCity)
- Works (EMPID, CompanyName, Salary)
- Company (CompanyName, CompanyCity)
- Manages (EMPID, ManagerID)

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

- Find the names of all employee who work for ‘ICICI’

```
SELECT EMPName FROM Employee WHERE
EMPID IN (SELECT EMPID FROM Works WHERE
CompanyName='ICICI')

SELECT EMPName FROM Employee INNER JOIN
Works ON Employee.EMPID = Works.EMPID
WHERE Works.CompanyName='ICICI'
```

- Find the names and cities of residence of all employees who work for ‘HDFC’.

```
SELECT EMPName, EMPCity FROM Employee
WHERE EMPID IN (SELECT EMPID FROM Works
WHERE CompanyName=‘HDFC')
```

```
SELECT EMPName , EMPCity FROM Employee
INNER JOIN Works ON Employee.EMPID =
Works.EMPID WHERE
Works.CompanyName=‘HDFC'
```

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

```
SELECT E.EMPName
FROM Employee E, Works W, Company C
WHERE
E.EMPID=W.EMPID and
W.CompanyName=C.CompanyName
and E.EMPCity=C.CompanyCity
```

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

```
SELECT P.EMPName
FROM Employee P, Employee Q, Manages M
WHERE
M.EMPID=P.EMPID and
M.ManagerID=Q.EMPID and
P.Street=Q.Street and
P.EMPCity=Q.EMPCity
```

- Find all employees in the database who do not work for ‘KAIJS’.

```
SELECT EMPName FROM Employee
WHERE EMPID IN
(SELECT EMPID FROM Works WHERE
CompanyName!='KAIJS')

SELECT EMPName FROM Employee
WHERE EMPID IN
(SELECT EMPID FROM Works WHERE
CompanyName <>'KAIJS')
```

- Assume that the companies may be located in several cities.
- Find all companies located in every city in which ‘ICICI’ is located

```
SELECT C.CompanyName FROM Company C
where C.CompanyCity IN
(SELECT CompanyCity FROM Company D
WHERE D.CompanyName='ICICI')
```

- Find the names, street addresses, and cities of residence of all employees who work for ‘HDFC’ and earn minimum 20,000.

```
SELECT EMPName, Street, EMPCity FROM
Employee WHERE EMPID IN (SELECT EMPID
FROM Works WHERE CompanyName='HDFC' and
Salary >=20000)
```

- Modify the database so that ‘BBB’ who was working for ‘HDFC’ now works for ‘ICICI’

```
UPDATE Works SET CompanyName='ICICI'
WHERE EMPID IN (SELECT EMPID FROM
Employee WHERE EMPName='BBB')

UPDATE Manages SET ManagerID=5 WHERE
EMPID IN (SELECT EMPID FROM Employee
WHERE EMPName='BBB')
```

- Given the Academic Institute database composed of following tables.
- PERSON (PID, PNAME, PADDRESS, UIDNO)
- UIDDATA (UIDNO, DOB)
- STUDENT (PRN, PID, CLS, DEPT)
- FACULTY (FID, PID, DEPT)
- STUDENTATTENDANCE (PRN, PERATTENDANCE)
- STUDENTMARKS (PRN, SUBJECT, MARKS)
- STUDENTFEE (PRN, FEEPAID, FEEDUES)
- LECTURES (FID, CLS, SUBJECT)

- Give an expression in SQL for each of the following queries
- Find the faculty who is pursuing PHD in same college
- Find the faculty teaching subject to 'TYCSE‘
- 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 TYCSE 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 123456789105
- Find the topper in subject ‘DBE’.
- Find the subjects in which all the students have passed the examination.
- Find the defaulter list of 'TYCSE‘
- Find the teacher who is teaching a subject in which no defaulter is found.

- Find the faculty who is pursuing Ph.D. in same college

```
SELECT P.PNAME

FROM PERSON P, STUDENT S, FACULTY F

WHERE

S.PID=P.PID AND

F.PID=P.PID AND

S.PID=F.PID AND

S.CLS='PHD'
```

- Find the faculty teaching subject to 'TYCSE'

```
SELECT P.PNAME
FROM PERSON P, FACULTY F, LECTURES L
WHERE
P.PID=F.PID AND
F.FID=L.FID AND
L.CLS='TYCSE'
```

- Find address of students whose attendance is less than 70%

```
SELECT P.PADDRESS
FROM PERSON P, STUDENT S,
STUDENTATTENDANCE A
WHERE
P.PID=S.PID AND
S.PRN=A.PRN AND
A.PERATTENDANCE<70
```

- Find all the students who are eligible for voting

```
SELECT P.PNAME

FROM PERSON P, UIDDATA U, STUDENT S

WHERE

P.PID=S.PID AND

P.UIDNO=U.UIDNO

AND FLOOR((SYSDATE-U.DOB)/365.25)>18
```

- Find the total fee paid by TYCSE students

```
SELECT S.CLS, SUM(E.FEEPAID)
FROM STUDENT S, STUDENTFEE E
WHERE
S.PRN=E.PRN AND
S.CLS='TYCSE'
GROUP BY S.CLS
```

- Find the address of students who have backlog subjects

```
SELECT DISTINCT (P.PADDRESS)
FROM PERSON P, STUDENT S,
STUDENTMARKS M
WHERE
P.PID=S.PID AND
S.PRN=M.PRN AND
M.MARKS<40
```

- Find the list of students who are supposed to attend DBE lectures.

```
SELECT P.PNAME
FROM PERSON P, STUDENT S, LECTURES L
WHERE
P.PID=S.PID AND
S.CLS=L.CLS AND
L.SUBJECT='DBE'
```

- Find all the teachers who teaches to student with uid 123456789106

```
SELECT Q.PNAME
FROM PERSON P, PERSON Q, STUDENT S,
FACULTY F, LECTURES L
WHERE
P.UIDNO=1234567890106 AND
P.PID=S.PID AND
S.CLS=L.CLS AND
F.FID=L.FID AND
Q.PID=F.PID
```

- Find the topper in subject ‘DBE’.

```
SELECT P.PNAME
FROM PERSON P, STUDENT S,
STUDENTMARKS M
WHERE
P.PID=S.PID AND
S.PRN=M.PRN AND
M.SUBJECT='DBE' AND
M.MARKS=(SELECT MAX(MARKS) FROM
STUDENTMARKS WHERE SUBJECT='DBE')
```

- Find the subjects in which all the students have passed the examination.

```
SELECT SUBJECT
FROM STUDENTMARKS
GROUP BY SUBJECT
HAVING MIN(MARKS)>=40
```

- Find the defaulter list of 'TYCSE'

```
SELECT P.PNAME
FROM PERSON P, STUDENT S,
STUDENTATTENDANCE A
WHERE
P.PID=S.PID AND
S.PRN=A.PRN AND
A.PERATTENDANCE<70 AND
S.CLS='TYCSE'
```

- Find the teacher who is teaching a subject in which no defaulter is found.

```
SELECT PNAME
FROM PERSON WHERE PID IN
(SELECT PID FROM FACULTY WHERE FID IN
(SELECT FID FROM
(SELECT FID FROM LECTURES MINUS SELECT
L.FID FROM LECTURES L, STUDENT S,
STUDENTATTENDANCE A
WHERE L.CLS=S.CLS AND
S.PRN=A.PRN AND
A.PERATTENDANCE<70)))
```

- Calculate Percentage Marks of all the Students of 'TYCSE'

```
SELECT M.PRN, AVG(M.MARKS)
FROM STUDENT S, STUDENTMARKS M
WHERE
S.PRN=M.PRN AND
S.CLS='TYCSE'
GROUP BY M.PRN
```

- Calculate Average Marks of 'TYCSE' Students in 'DBE' subject

```
SELECT AVG(M.MARKS)
FROM STUDENT S, STUDENTMARKS M
WHERE
S.PRN=M.PRN AND
S.CLS='TYCSE'
GROUP BY M.SUBJECT
HAVING (M.SUBJECT='DBE')
```

- Calculate Average Marks of 'TYCSE' Students in all subjects

```
SELECT M.SUBJECT, AVG(M.MARKS)
FROM STUDENT S, STUDENTMARKS M
WHERE
S.PRN=M.PRN AND
S.CLS='TYCSE'
GROUP BY M.SUBJECT
```

Thank You