UNIT II

SQL

Structured Query Language (SQL) is the standard command set used to communicate with the relational database management systems.

Advantages

- Simple and easy to learn
- can handle complex situations
- Increased acceptance and availability
- Easily ported across systems

DATA TYPES

Predefined Data Types

Char(size)	Fixed length character data, max size 2000 default is 1 byte
Date	Used to store date value as DD-MON-YY, data size is 9 byte
Number	For number column with space for 40 digits plus space for decimal point and sign
Number(size)	Number column of specified size
Number(size,d)	Number column of specified size with d digits after decimal point
Varchar2(size)	Variable length character string having a maximum of size upto 4000 bytes
Varchar(size)	Same as varchar2, usage may be change in future versions of oracle
Float	Same as number
Integer	Same as number does not accept decimal digits as an argument
Long	Character data of variable size upto 2 GB
BLOB	Binary Large Objects upto 4 GB
CLOB	Character Large Objects upto 4 GB

User Defined Data Types

User defined types useful in situations where the same data type is used in several columns from different tables.

The names of the user-defined types provides the extra information.

Syntax

- Creation
  create or replace type <type_name> as object <representation >
- Description:
  desc <type_name>;
- Delete type:
  drop type <typename>;

Types of SQL Commands

1. DDL
2. DML
3. DCL
4. TCL

1. DDL

DDL is abbreviation of **Data Definition Language**. It is used to create and modify the structure of database objects in database.

- Used by the DBA and database designers to define the conceptual schema of a database
- In many DBMSs, the DDL is also used to define the internal and external schemas
- In some DBMSs, separate Storage Definition Language (SDL) and View Definition Language (VDL) are used to define internal and external schemas

DDL Commands

a. create - Creates an object(table) in the database.
b. alter – Modifies the structure of an existing object in various ways. used to add, modify, or drop/delete columns in a table
    - ✓ add
    - ✓ modify
    - ✓ drop
c. rename - allows you to rename an existing table in any schema
d. drop – Deletes an object in database usually that cannot be "ROLL BACK"

a. create :
Syntax:
```
create table <table name> (column name1 datatype1 constraints, column2 datatype2 . . .);
```
Example:
```
create table loan(loan_no varchar2(20) primary key, branch_name varchar2(20), amount number);
```

b. alter:

ADD:
Syntax:
```
alter table <table name> add(column name1 datatype1);
```
Example:
```
alter table loan add(primary key(loan_no));
alter table loan add(roi number);
```
MODIFY:
Syntax:
```
alter table <table name> modify(column name1 datatype1);
```
Example:
```
alter table loan modify(branch_name varchar2(25));
```
DROP:
Syntax:
```
alter table <table name> drop (column name);
```
Example:
```
alter table loan drop(roi);
```

c. rename:
Syntax:
```
rename <old table name> to <new table name>;
```

Example:

```
rename loan to loan1;
```

d. drop:

Syntax:

```
drop table <table name>;
```

Example:

```
drop table loan;
```

2. DML

DML is abbreviation of **Data Manipulation Language**. It is used to retrieve, store, modify, delete, insert and update data in database.

- DML commands can be used as stand-alone (query language) or can be embedded within a general-purpose language
- Procedural DML - allows user to tell system exactly how to manipulate data
- Non-Procedural DML - allows user to state what data is needed rather than how it is to be retrieved

DML Commands

a. insert – Add tuples/attributes to an existing table
b. select - Used to retrieve records from one or more tables
c. update – Modifies a set of existing table tuples/attributes
d. delete- Removes existing tuples/attributes from a table

Syntax:

a. insert:

Single level:

Syntax: Insert into <table name> values (‘attributes1’, ’attributes2’……);

Example: insert into loan values('ln_106','saidapet',2000);

Multilevel:

Syntax: Insert into <table name> values (‘&attributes1’,’&attributes2’….);

Example:

```
SQL> insert into loan values('&loan_no','&branch_name',&amount);

Enter value for loan_no: A101
Enter value for branch_name: chennai-45
Enter value for amount: 50000

old   1 : insert into loan values('&loan_no','&branch_name',&amount);
new  1: insert into loan values('A101','chennai-45',50000)
1 row created.
```

b. select:

Single level:

Syntax: **s**elect <column name> from <table name>;

Example: select branch_name from loan;

Multilevel:

Syntax: **s**elect * from <table name> where <condition>;

Example: Select * from loan order by amount desc;

c. update:

Single level:

Syntax: update <table name> set <column name>='values' where <condition>;

Example: update loan set branch_name='chennai-20' where branch_name='chennai-45';

Multilevel:

Syntax: update <table name> set <column name>='values';

Example: update loan set branch_name='chennai-20'

d. delete:

Single level:

Syntax: delete from <table name> where <column name>='values';

Example: delete from loan where branch_name='chennai-20';

Multilevel:

Syntax: delete from <table name>;

Example: delete from loan;

3. DCL

DCL is abbreviation of **Data Control Language**. It is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

DCL Commands

a. **GRANT -** Used to grant privileges to the user
b. **REVOKE -** Used to revoke privileges from the user

Syntax

a. GRANT

```
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

- **privilege_name** is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.
- **object_name** is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.
- **user_name** is the name of the user to whom an access right is being granted.
- **user_name** is the name of the user to whom an access right is being granted.
- **PUBLIC** is used to grant access rights to all users.
- **ROLES** are a set of privileges grouped together.
- **WITH GRANT OPTION** - allows a user to grant access rights to other users.

b. REVOKE

```
REVOKE privilege_name
ON object_name
FROM {user_name |PUBLIC |role_name}
```

1) System privileges - This allows the user to CREATE, ALTER, or DROP database objects.
2) Object privileges - This allows the user to EXECUTE, SELECT, INSERT, UPDATE, or DELETE data from database objects to which the privileges apply.

4. TCL

TCL is abbreviation of **Transaction Control Language**. It is used to manage different transactions occurring within a database.

TCL Commands

- **COMMIT -** Used to made the changes permanently in the Database.
- **ROLLBACK -** Similar to the undo operation.

Example

```
SQL> delete from branch;
6 rows deleted.
SQL> select * from branch;
no rows selected

SQL> rollback;
Rollback complete.
SQL>  select * from branch;

BRANCH_NAME    BRANCH_CITY           ASSETS
--------------- ------------------- ------------------------------------------
tambaram             chennai-20              50000
adayar               chennai-20              100000
tnagar               chennai-17               250000
saidapet             chennai-15              150000
chrompet             chennai-43              450000
guindy               chennai-32              150000
6 rows selected.
```

SAVE POINT

```
SQL> select * from customer;
   CUSTID   PID      QUANTITY
    ----------   ----------  ----------
      100      1234       10
      101      1235       15
      102      1236       15
      103      1237       10

SQL> savepoint s1;
Savepoint created.

SQL> Delete from customer where custid=103;
   CUSTID  PID      QUANTITY
    ----------  ----------  ----------
    100      1234       10
    101      1235       15
    102      1236       15

SQL> rollback to s1;
Rollback complete.
SQL> select * from customer;
   CUSTID   PID     QUANTITY
   ----------  ---------- ----------
    100      1234       10
    101      1235       15
    102      1236       15
    103      1237       10
SQL> commit;
```

DATABASE OBJECTS

A *database object* is any defined object in a database that is used to store or reference data. Some examples of database objects include tables, views, sequences, indexes, and synonyms.

1. VIEWS

View can be created to retrieve data from one or more tables.

Syntax

```
CREATE VIEW <VIEWNAME> AS SELECT * FROM <TABLENAME> WHERE
<CONDITION>;
```

Example

Create a view using aggregate functions to calculate the age of the customer

```
SQL> create view cust_age as select
CUSTOMER_ID,CUSTOMER_NAME,round((sysdate-CUSTOMER_DOB)/365.25) as
age from customer;

View created.

SQL> select * from cust_age;

CUSTOMER_ID        CUSTOMER_NAME                    AGE
---------- ------------------- ------------------------------------------
cus_101                      suresh             28
cus_102                      selva              26
cus_103                      prem               26
cus_104                      javid              36
cus_105                      pradeep            26
cus_106                      gopal              29
```

```
cus_107                    raja                 27
cus_108                    krishnan             13
cus_109                    mohammed               15

9 rows selected.
```

2. SYNONYMS

A synonym is an alias for a database object (table, view, procedure, function, package, sequence, etc.).

Syntax

CREATE SYNONYM <SYNONYMS NAME> FOR <TABLENAME>;

Example

CREATING A SYNONYM FOR A TABLE

```
CREATE TABLE product (product_name VARCHAR2(25) PRIMARY KEY,
product_price    NUMBER(4,2), quantity_on_hand NUMBER(5,0),
last_stock_date  DATE);

Table created.
```

AFTER INSERTING THE RECORDS TO PRODUCT TABLE

```
 SQL> SELECT * FROM product;

PRODUCT_NAME        PRODUCT_PRICE QUANTITY_ON_HAND LAST_STOC
-------------------------  ------------------------- ------------------------------ ------------------
Product 1                    99              1               15-JAN-03
Product 2                   75              1000            15-JAN-02
Product 3                   50              100             15-JAN-03
Product 4                   25              10000           14-JAN-03
Product 5                   9.95            1234            15-JAN-04
Product 6                   45              1               31-DEC-08

6 rows selected.
SQL> SELECT * FROM prod;
SELECT * FROM prod
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

```
SQL> CREATE SYNONYM prod FOR product;
Synonym created.

SQL> SELECT * FROM prod;
PRODUCT_NAME            PRODUCT_PRICE QUANTITY_ON_HAND LAST_STOC
-------------------------        ------------------------- ------------------------------  ------------------
Product 1                         99              1                15-JAN-03
Product 2                        75              1000             15-JAN-02
Product 3                     50            100           15-JAN-03
Product 4                     25            10000         14-JAN-03
Product 5                     9.95          1234          15-JAN-04
Product 6                     45            1             31-DEC-08

SQL> drop SYNONYM prod;
Synonym dropped.
SQL> drop table product;
Table dropped.
```

3. SEQUENCE

A sequence is a user-defined schema bound object that generates a sequence of numeric values according to the specification with which the sequence was created. The sequence of numeric values is generated in an ascending or descending order at a defined interval and can be configured to restart (cycle) when exhausted.

Syntax

CREATE SEQUENCE<SEQUENCENAME> START WITH <VALUE> MINVALUE <VALUE> INCREMENT BY <VALUE>;

Example

create a sequence and design the student table with the given attributes.

```
SQL> create table student(student_id number, name varchar2(10),result varchar2(10));
SQL> desc student;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 STUDENT_ID                                         NUMBER
 NAME                                               VARCHAR2(10)
 RESULT                                             VARCHAR2(10)
```

Sequence Creation

```
SQL> create sequence student_seq start with 100 minvalue 100 increment by 1;
Sequence created.
SQL> insert into student values(student_seq.nextval,'raja','pass');
1 row created.
SQL> insert into student values(student_seq.nextval,'ravi','pass');
1 row created.
SQL> select * from student;
STUDENT_ID NAME
---------- ---------- ----------
       100 raja       pass
       101 ravi       pass
```

4. INDEXES

An index can be created in a table to find data more quickly and efficiently. The users cannot see the indexes, they are just used to speed up searches/queries.

Syntax

CREATE [UNIQUE] INDEX index_name ON table_name (column_name);

Example

To create an index on the Last Name column of the Employee table

```
SQL> create table Employee(ID VARCHAR2(4) NOT NULL,
First_Name        VARCHAR2(10), Last_Name  VARCHAR2(10),
 Start_Date  DATE, End_Date          DATE, Salary            Number(8,2));

Table created.
SQL> select * from Employee

ID   FIRST_NAME  LAST_NAME  START_DAT END_DATE     SALARY
----- ---------------- ------------------ ----------------- ---------------     -------------------
01       Jason            Martin          25-JUL-96       25-JUL-06   1234.56
02       Alison           Mathews         21-MAR-76   21-FEB-86     6661.78
03       James            Smith           12-DEC-78   15-MAR-90    6544.78
04       Celia            Rice            24-OCT-82   21-APR-99     2344.78
05       Robert           Black           15-JAN-84    08-AUG-98    2334.78
6 rows selected.

SQL>  CREATE INDEX LastNameIndex ON Employee (Last_Name);

Index created.
SQL>  drop index LastNameIndex;
Index dropped.
```

5. SAVE POINT

The SAVEPOINT statement is used to define an SQL savepoint – and therefore the start of a subtransaction within a transaction – and to assign this SQL savepoint a name. A subsequent ROLLBACK TO statement with the SQL savepoint name reverses any modifications that have been made in the meantime, without affecting the database operations that were executed within the transaction before the start of this subtransaction.

Syntax

SAVEPOINT <SAVEPOINT NAME>;

Example

```
SQL> select * from employees;

DEPARTMENT_ID DEPARTMENT_NAME
-------------           ---------------
        101              it
        102             cse
        103             mech
        104             chemical
        105             biotech
        106             eee
6 rows selected.
```

```
SQL> savepoint s1;
Savepoint created.

SQL> insert into employees values(107,'ice');
1 row created.

SQL> savepoint s2;
Savepoint created.

SQL> select * from employees;

DEPARTMENT_ID DEPARTMENT_NAME
------------- ---------------
        101 it
        102 cse
        103 mech
        104 chemical
        105 biotech
        106 eee
        107 ice
7 rows selected.

SQL> ROLLBACK TO SAVEPOINT s1;
Rollback complete.
SQL> select * from employees;

DEPARTMENT_ID DEPARTMENT_NAME
-------------          ---------------
        101             it
        102            cse
        103            mech
        104            chemical
        105            biotech
        106            eee
6 rows selected.
```

CONSTRAINTS

Constraints within a database are rules which control values allowed in columns.

- Rule or restriction concerning a piece of data, enforced at the data level rather than application level
- A constraint clause can constrain a single column or group of column in a table

INTEGRITY

Integrity refers to requirement that information be protected from improper modification.

INTEGRITY CONSTRAINTS

Integrity constraints provide a way of ensuring that changes made to the database by authorized users do not result in a loss of data consistency

- Constraints
    1. Primary key
    2. Referential integrity
    3. Check constraint
    4. Unique Constraint
    5. Not Null/Null

Example Tables

```
create table studentdb(student_no number primary key,student_name varchar2(30) not
null,student_email varchar2(20) unique,student_percentage number
check(student_percentage<=100))

create table coursedb(course_id number primary key,STUDENT_NO NUMBER references
studentdb(STUDENT_NO))
```

1. PRIMARY KEY

The primary key of a relational table uniquely identifies each record in the table

-Unique
-Not null

Example:

<u>**CASE 1:**</u> (Redundant value cannot be accepted by the column)

```
SQL> insert into studentdb values(3123101,'rajan','rajan@gmail.com',76);
1 row created.

SQL> insert into studentdb values(3123101,'rajan','rajan@gmail.com',76);
insert into studentdb values(3123101,'rajan','rajan@gmail.com',76)
*
ERROR at line 1:

ORA-00001: unique constraint (SCOTT.SYS_C006201) violated
```

<u>**CASE 2:**</u> (Null value cannot be accepted by the column)

```
SQL> insert into studentdb values(null,'rajan','rajan@gmail.com',76);
insert into studentdb values(null,'rajan','rajan@gmail.com',76)
*
ERROR at line 1:

ORA-01400: cannot insert NULL into ("SCOTT"."STUDENTDB"."STUDENT_NO")
```

2. REFERENTIAL INTEGRITY

- We ensure that a value appears in one relation for a given set of attribute also appears for a certain set of attribute in another relation.
- The foreign key identifies a column or set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table

Example:

```
SQL> insert into coursedb values(201,3123101);
1 row created.

SQL> insert into coursedb values(202,3123100);
insert into coursedb values(202,3123100)
*
ERROR at line 1:
ORA-02291: integrity constraint (SCOTT.SYS_C006204) violated - parent key not Found
```

3. CHECK CONSTRAINT

A **check constraint** allows you to specify a condition on each row in a table.

- A check constraint can NOT be defined on a **VIEW**.
- The check constraint defined on a table must refer to only columns in that table. It cannot refer to columns in other tables.
- A check constraint can NOT include a **SUBQUERY**.

Example:

```
SQL> insert into studentdb values(3123106,'ragu3','ragu3@gmail.com',101);
insert into studentdb values(3123106,'ragu3','ragu3@gmail.com',101)
*
ERROR at line 1:
ORA-02290: check constraint (SCOTT.SYS_C006200) violated

SQL> insert into studentdb values(3123106,'ragu3','ragu3@gmail.com',100);
1 row created
```

4. UNIQUE CONSTRAINT

- Redundant values will not be accepted
- Accept more than one Null values

Example:

<u>**CASE 1:**</u> (Redundant value cannot be accepted by the column)

```
SQL> insert into studentdb values(3123103,'ragu','ragu@gmail.com',90);
1 row created.

SQL> insert into studentdb values(3123104,'ragu1','ragu@gmail.com',45);
insert into studentdb values(3123104,'ragu1','ragu@gmail.com',45)
*
ERROR at line 1:
```

```
ORA-00001: unique constraint (SCOTT.SYS_C006202) violated
```

<u>**CASE 2:**</u> (More than one null value can be accepted by the column)

```
SQL> insert into studentdb values(3123104,'ragu1',null,67);
1 row created.

SQL> insert into studentdb values(3123105,'ragu2',null,98);
1 row created.
```

5. NOT NULL

-Null value cannot be accepted by the column)

Example:

```
SQL> insert into studentdb values(3123102,'ravi','ravi@gmail.com',80);
1 row created.

SQL> insert into studentdb values(3123103,null,'hai@gmail.com',80);
insert into studentdb values(3123103,null,'hai@gmail.com',80)
*
ERROR at line 1:

ORA-01400: cannot insert NULL into ("SCOTT"."STUDENTDB"."STUDENT_NAME")
```

EMBEDDED SQL

-The SQL standards defines embedding of SQL in variety of programming language such as VB,C,C++,Java.

-A language to which SQL queries are embedded is referred to as a host language and the SQL structures permitted in the language comprise embedded SQL.

Two reasons

- Not all queries expressed in SQL, since SQL does not provide the full expressive power of a general purpose language.
- Non-declarative actions- printing a report, interacting with user.

Embedded SQL in C

```
loop = 1;
while (loop) {
        prompt ("Enter SSN: ", ssn);
        EXEC SQL
                select FNAME, LNAME, ADDRESS, SALARY
                into :fname, :lname, :address, :salary
                from EMPLOYEE where SSN == :ssn;
                if (SQLCODE == 0) printf(fname, …);
                else printf("SSN does not exist: ", ssn);
                prompt("More SSN? (1=yes, 0=no): ", loop);
        END-EXEC
}
```

STATIC SQL

SQL statements in the program are static; that is, they do not change each time the program is run. These statements are compiled when the rest of the program is compiled. Static SQL works well in many situations and can be used in any application for which the data access can be determined at program design time.

For example, an order-entry program always uses the same statement to insert a new order, and an airline reservation system always uses the same statement to change the status of a seat from available to reserve. Each of these statements would be generalized through the use of host variables; different values can be inserted in a sales order, and different seats can be reserved. Because such statements can be hard-coded in the program, such programs have the advantage that the statements need to be parsed, validated, and optimized only once, at compile time. This results in relatively fast code.

DYNAMIC SQL

- The dynamic SQL components of SQL allows programs to construct and submit SQL Queries at runtime.
- In contrast the embedded SQL statement must be completely present at compile time they are compiled by the embedded SQL pre processor
- Using dynamic SQL the programs can create SQL queries at runtime and can either have them executed immediately or have they prepared for subsequent use.

Query optimization

The process of choosing a suitable execution strategy for processing a query is called query optimization.

Two internal representations of a query

- Query Tree
  - a tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as *leaf nodes* of the tree, and represents the relational algebra operations as *internal nodes*.
- Query Graph
  - a graph data structure that corresponds to a relational calculus expression. It does not indicate an order on which operations to perform first. There is only a single graph corresponding to each query.

Steps in processing a high-level query

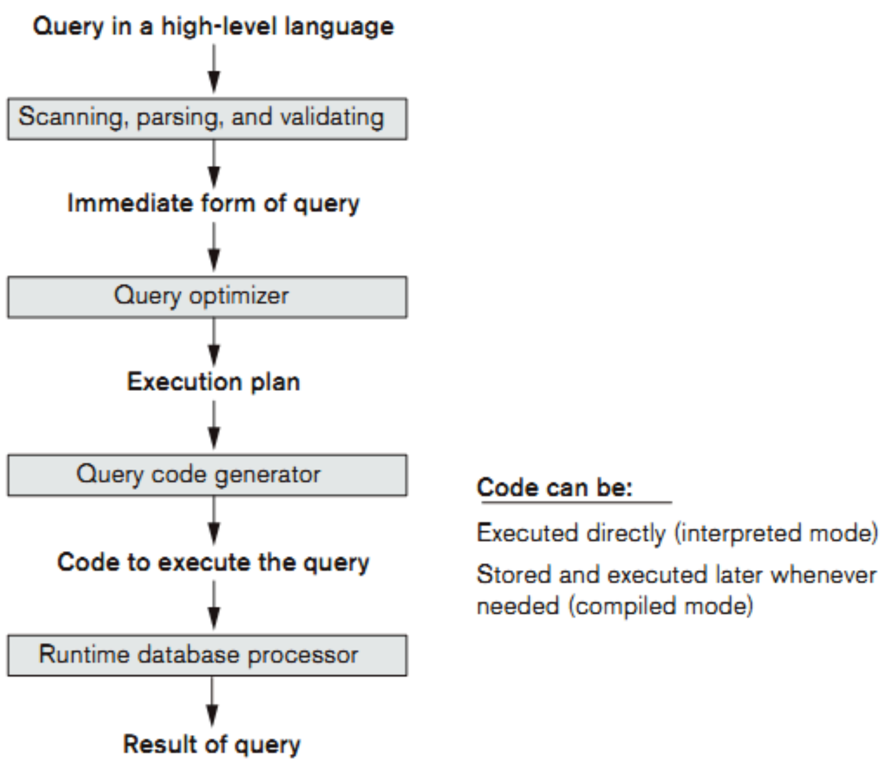

Query in a high-level language
Scanning, parsing, and validating
Immediate form of query
Query optimizer
Execution plan
Query code generator
Code to execute the query
Runtime database processor
Result of query
Code can be:
Executed directly (interpreted mode)
Stored and executed later whenever needed (compiled mode)

Figure 19.1
Typical steps when processing a high-level query.

(i) Using Heuristics in Query Optimization
Process for heuristics optimization

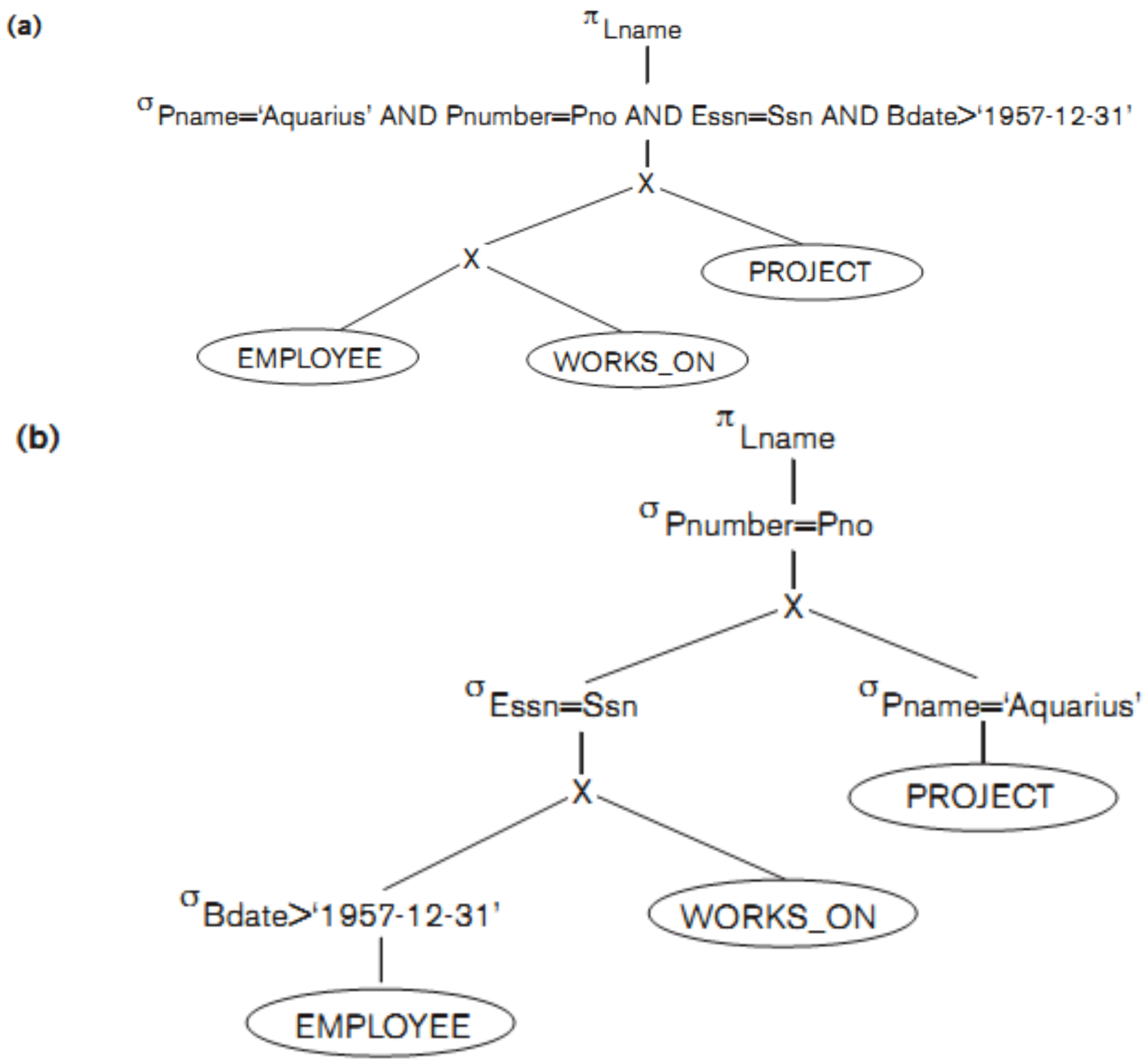
1. The parser of a high-level query generates an *initial internal representation*;
2. Apply heuristics rules to optimize the internal representation.
3. A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.

Example:

```
Q: SELECT LNAME
       FROM          EMPLOYEE, WORKS_ON, PROJECT
       WHERE  PNAME = 'AQUARIUS' AND PNMUBER=PNO
              AND ESSN=SSN AND BDATE > '1957-12-31';
```

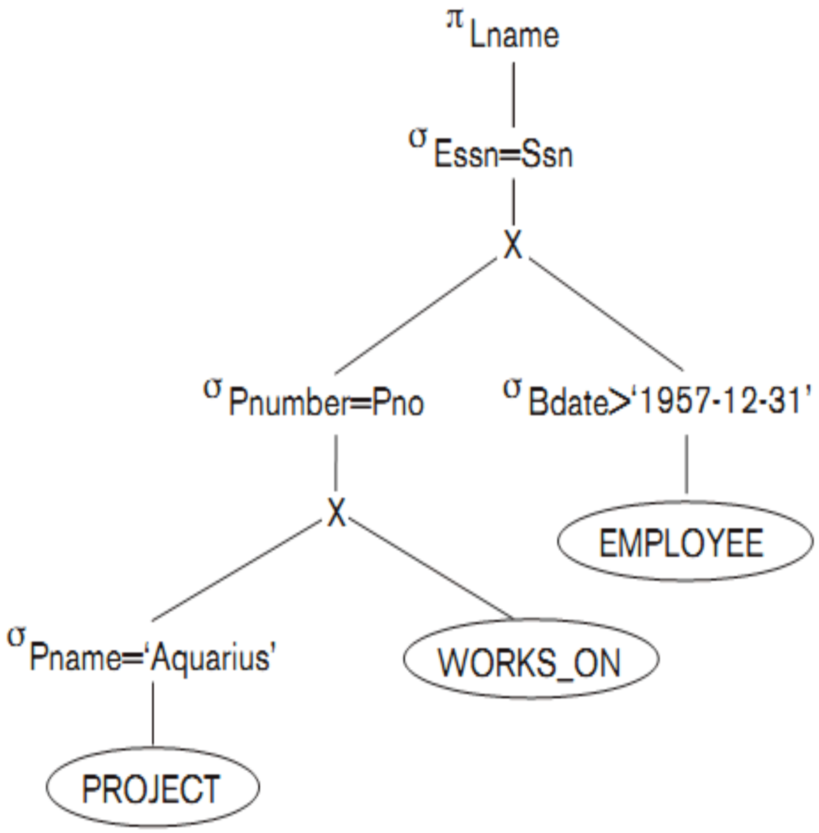
Steps in converting a query tree during heuristic optimization

a) Formulate initial query tree for SQL query
b) Move select operations down the query tree
c) Apply the most restrictive select operation first
d) Replace Cartesian product and select with join operations
e) Move project operations down the query tree

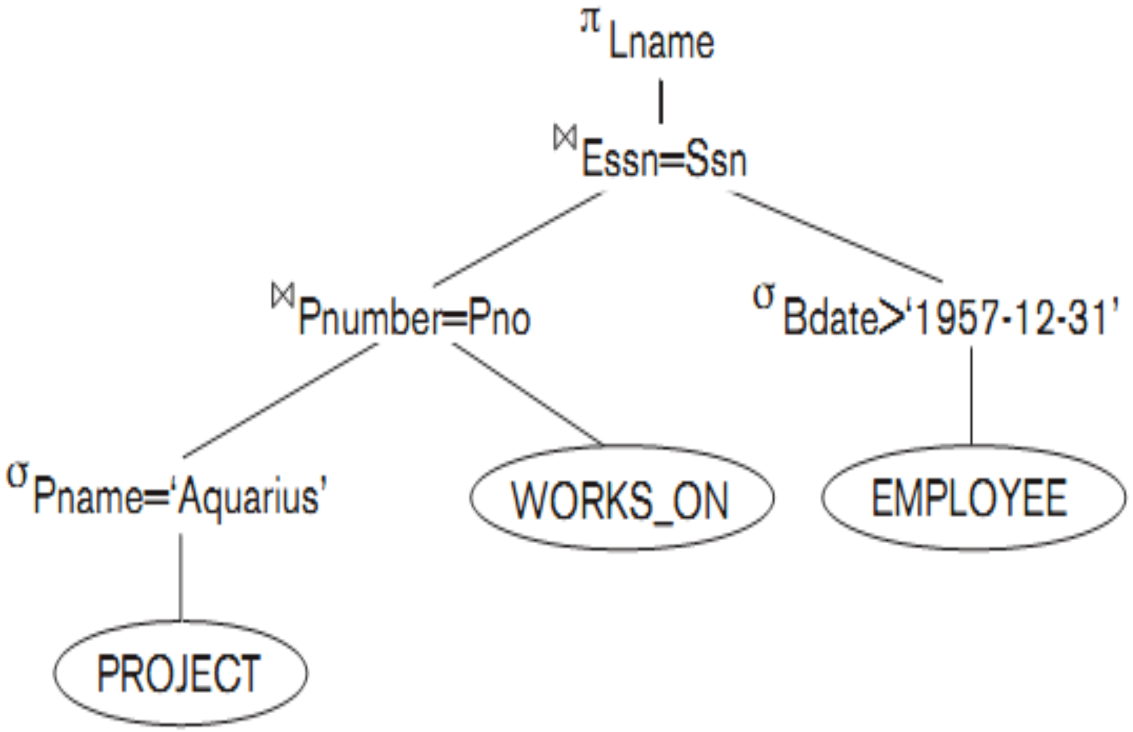


(a) Initial (canonical) query tree for SQL query Q.
(b) Moving SELECT operations down the query tree.

(c)

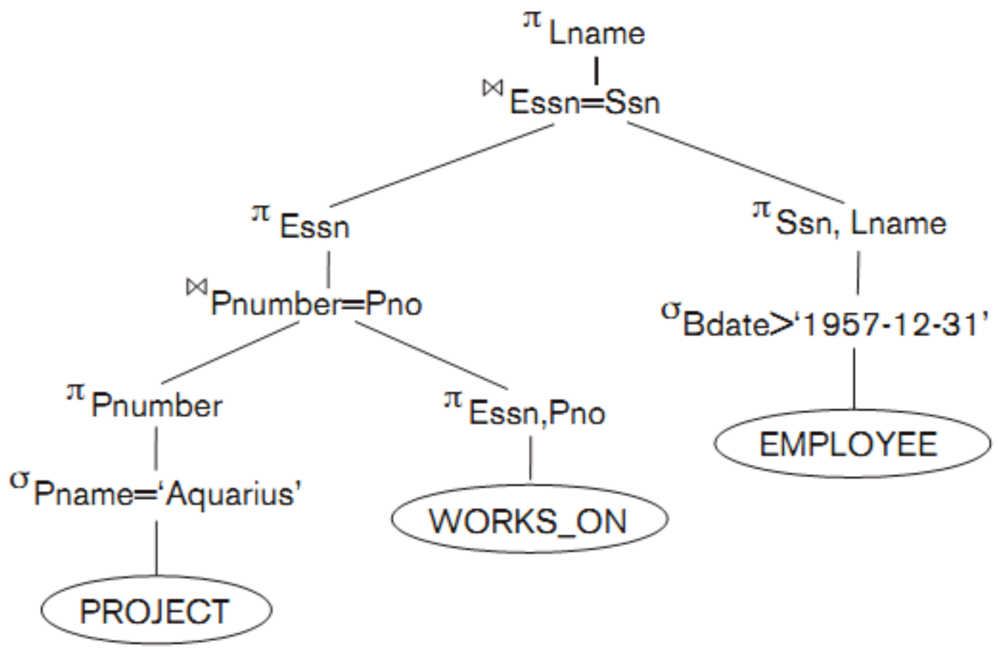


(d)



(c) Applying the more restrictive SELECT operation first.
(d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.

(e)



(e) Moving PROJECT operations down the query tree.

General Transformation Rules for Relational Algebra Operations:

1. **Cascade of σ** A conjunctive selection condition can be broken up into a cascade (that is, a sequence) of individual σ operations:

   $\sigma_{c_1 \text{ AND } c_2 \text{ AND} \ldots \text{AND } c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R))\ldots))$
2. **Commutativity of σ.** The σ operation is commutative:

   $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$
3. **Cascade of π.** In a cascade (sequence) of π operations, all but the last one can be ignored:

   $\pi_{\text{List}_1}(\pi_{\text{List}_2}(\ldots(\pi_{\text{List}_n}(R))\ldots)) \equiv \pi_{\text{List}_1}(R)$
4. **Commuting σ with π.** If the selection condition *c* involves only those attributes $A_1, \ldots, A_n$ in the projection list, the two operations can be commuted:

   $\pi_{A_1, A_2, \ldots, A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1, A_2, \ldots, A_n}(R))$

5. **Commutativity of ⋈ (and ×).** The join operation is commutative, as is the × operation:

   $R \bowtie_c S \equiv S \bowtie_c R$

   $R \times S \equiv S \times R$

6. **Commuting σ with $\bowtie$ (or $\times$).** If all the attributes in the selection condition c involve only the attributes of one of the relations being joined—say, R—the two operations can be commuted as follows:

   $\sigma_c (R \bowtie S) \equiv (\sigma_c (R)) \bowtie S$

7. **Commuting π with $\bowtie$ (or $\times$).** Suppose that the projection list is $L = \{A_1, ..., A_n, B_1, ..., B_m\}$, where $A_1, ..., A_n$ are attributes of R and $B_1, ..., B_m$ are attributes of S. If the join condition c involves only attributes in L, the two operations can be commuted as follows:

   $\pi_L (R \bowtie_c S) \equiv (\pi_{A_1, ..., A_n} (R)) \bowtie_c (\pi_{B_1, ..., B_m} (S))$

8. **Commutativity of set operations.** The set operations $\cup$ and $\cap$ are commutative but $-$ is not.

9. **Associativity of $\bowtie$, $\times$, $\cup$, and $\cap$.** These four operations are individually associative; that is, if θ stands for any one of these four operations (throughout the expression), we have:

   $(R\ \theta\ S)\ \theta\ T \equiv R\ \theta\ (S\ \theta\ T)$

10. **Commuting σ with set operations.** The σ operation commutes with $\cup$, $\cap$, and $-$. If θ stands for any one of these three operations (throughout the expression), we have:

    $\sigma_c (R\ \theta\ S) \equiv (\sigma_c (R))\ \theta\ (\sigma_c (S))$

11. **The π operation commutes with $\cup$.**

    $\pi_L (R \cup S) \equiv (\pi_L (R)) \cup (\pi_L (S))$

12. **Converting a $(\sigma, \times)$ sequence into $\bowtie$.** If the condition c of a σ that follows a $\times$ corresponds to a join condition, convert the $(\sigma, \times)$ sequence into a $\bowtie$ as follows:

    $(\sigma_c (R \times S)) \equiv (R \bowtie_c S)$

Outline of a Heuristic Algebraic Optimization Algorithm:

1. Using rule 1, break up any select operations with conjunctive conditions into a cascade of select operations.
2. Using rules 2, 4, 6, and 10 concerning the commutativity of select with other operations, move each select operation as far down the query tree as is permitted by the attributes involved in the select condition.
3. Using rule 9 concerning associativity of binary operations, rearrange the leaf nodes of the tree so that the leaf node relations with the most restrictive select operations are executed first in the query tree representation.
4. Using Rule 12, combine a cartesian product operation with a subsequent select operation in the tree into a join operation.

5. Using rules 3, 4, 7, and 11 concerning the cascading of project and the commuting of project with other operations, break down and move lists of projection attributes down the tree as far as possible by creating new project operations as needed.
6. Identify subtrees that represent groups of operations that can be executed by a single algorithm.

(ii) Cost-based query optimization

Estimate and compare the costs of executing a query using different execution strategies and choose the strategy with the lowest cost estimate.

Issues

- Cost function
- Number of execution strategies to be considered

Cost Components for Query Execution

1. Access cost to secondary storage
2. Storage cost
3. Computation cost
4. Memory usage cost
5. Communication cost

Catalog Information Used in Cost Functions

- Information about the size of a file
  - number of records (tuples) (r),
  - record size (R),
  - number of blocks (b)
  - blocking factor (bfr)
- Information about indexes and indexing attributes of a file
  - Number of levels (x) of each multilevel index
  - Number of first-level index blocks (bI1)
  - Number of distinct values (d) of an attribute
  - Selectivity (sl) of an attribute
  - Selection cardinality (s) of an attribute. ($s = sl * r$)

Examples of Cost Functions for SELECT

- S1. Linear search (brute force) approach
  - $C_{S1a} = b$;
  - For an equality condition on a key, $C_{S1a} = (b/2)$ if the record is found; otherwise $C_{S1a} = b$.
- S2. Binary search:
  - $C_{S2} = \log_2 b + (s/bfr)\rceil - 1$
  - For an equality condition on a unique (key) attribute, $C_{S2} = \log_2 b$
- S3. Using a primary index (S3a) or hash key (S3b) to retrieve a single record
  - $C_{S3a} = x + 1$; $C_{S3b} = 1$ for static or linear hashing;
  - $C_{S3b} = 1$ for extendible hashing;
- S4. Using an ordering index to retrieve multiple records:

    - For the comparison condition on a key field with an ordering index, $C_{S4} = x + (b/2)$
- S5. Using a clustering index to retrieve multiple records:
    - $C_{S5} = x + \lceil (s/bfr) \rceil$
- S6. Using a secondary (B+-tree) index:
    - For an equality comparison, $C_{S6a} = x + s$;
    - For an comparison condition such as >, <, >=, or <=,
    - $C_{S6a} = x + (b_{I1}/2) + (r/2)$
- S7. Conjunctive selection:
    - Use either S1 or one of the methods S2 to S6 to solve.
    - For the latter case, use one condition to retrieve the records and then check in the memory buffer whether each retrieved record satisfies the remaining conditions in the conjunction.
- S8. Conjunctive selection using a composite index:
    - Same as S3a, S5 or S6a, depending on the type of index.

Examples of Cost Functions for JOIN

- J1. Nested-loop join:
    - $C_{J1} = b_R + (b_R{*}b_S) + ((js{*}\ |R|{*}\ |S|)/bfr_{RS})$
    - (Use R for outer loop)
- J2. Single-loop join (using an access structure to retrieve the matching record(s))
    - If an index exists for the join attribute B of S with index levels x_B, we can retrieve each record s in R and then use the index to retrieve all the matching records t from S that satisfy t[B] = s[A].
    - The cost depends on the type of index.
    - For a secondary index,
        - $C_{J2a} = b_R + (|R| * (x_B + s_B)) + ((js{*}\ |R|{*}\ |S|)/bfr_{RS})$;
    - For a clustering index,
        - $C_{J2b} = b_R + (|R| * (x_B + (s_B/bfr_B))) + ((js{*}\ |R|{*}\ |S|)/bfr_{RS})$;
    - For a primary index,
        - $C_{J2c} = b_R + (|R| * (x_B + 1)) + ((js{*}\ |R|{*}\ |S|)/bfr_{RS})$;
    - If a hash key exists for one of the two join attributes — B of S
        - $C_{J2d} = b_R + (|R| * h) + ((js{*}\ |R|{*}\ |S|)/bfr_{RS})$;
- J3. Sort-merge join:
        - $C_{J3a} = C_S + b_R + b_S + ((js{*}\ |R|{*}\ |S|)/bfr_{RS})$;
        - (CS: Cost for sorting files)