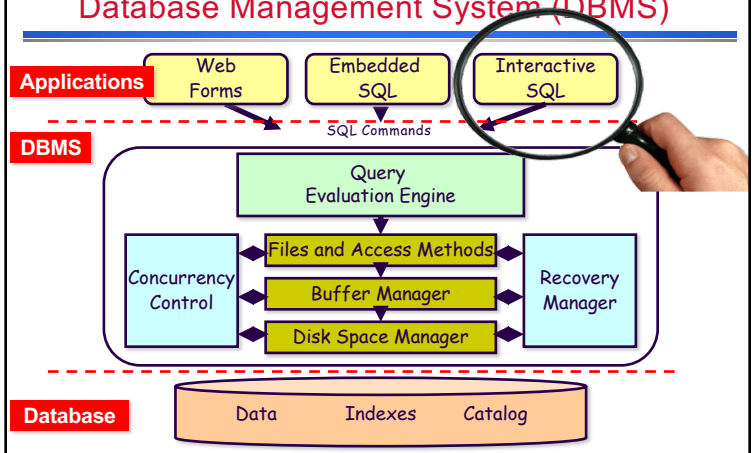
Structured Query Language
SQL – DML

- Relational Operators
- Set Relational Operators
- Retrieving with NULLs
- Nested Operations

Database Management System (DBMS)



Declarative Query

SELECT Title, TeamName

FROM SUPERBOWL

WHERE Rank = 1;

Title	TeamName
Champions	

SQL Select Statement

- Complete form:

```
SELECT [DISTINCT | ALL] attribute-list
FROM   table-list
WHERE selection-condition
GROUP BY grouping-attribute(s)
HAVING grouping-condition
ORDER BY {attribute ASC | DESC} pairs
```

Recall - Preliminaries

- A query is applied to "relation instances" (tables), and the result of a query is also a relation instance (table)
- List-oriented (positional) notation vs. Set-oriented (named-field) notation:
  - Both used in SQL-Select

SQL Insert

STUDENT(SID,Name,Major,QPA)

- **Implicit (list)**:

```
INSERT INTO STUDENT
VALUES (165, 'Susan Jones', 'CS', 0.00);
```

- **Explicit (set)**:

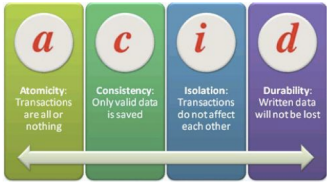
```
INSERT INTO STUDENT (SID, Name)
VALUES (165, 'Susan Jones');

INSERT INTO STUDENT (Name, SID)
VALUES ( 'Susan Jones', 165);
```

- Values-clause may be a list of tuples in some systems

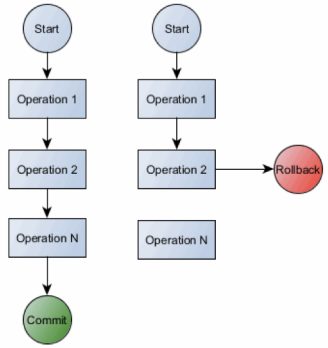
Execution Abstraction

- *A **transaction** is a **logical unit of work** in DBMSs*
  - *It is the execution of a **program segment** that performs some function or task by accessing shared data (e.g., a db)*
  - logical grouping of query and update requests needed to perform a task
- Examples:
  - banking transaction
    - Deposit, withdraw, transfer $
  - airline reservation
    - reserve a seat on a flight
  - inventory transaction
    - Receive, Ship, Update



SQL TRANSACTIONS

- SET TRANSACTION [READ ONLY | READ WRITE] NAME <name>;
- COMMIT ;
- ROLLBACK;
- ROLLBACK default action



Oracle ORA-00054 Error

- ORA-00054: “resource busy and acquire with NOWAIT specified or timeout expired”
- Oracle assume your SQL statements are executed as transactions
  - If you do not start a transaction explicitly, sqlplus starts one implicitly for you
- SO, to avoid getting this error, invoke "commit;"
  - just before exiting sqlplus
  - after an operation that changes the db schema or data
    - i.e., you don't need it for plain queries

ANSI SQL2 Isolation Levels

- SET TRANSACTION READ ONLY | READ WRITE
  [ISOLATION LEVEL READ UNCOMMITTED |
  READ COMMIT |
  REPEATABLE READ |
  SERIALIZABLE]

Relational Operators in SQL

- STUDENT(SID, Name, Major)

- $\Pi_{attribute\_list}(r)$:
  - $\Pi_{SID,Major}(STUDENT)$

```
SELECT  SID, Major
FROM    STUDENT;
```

- $\sigma_{selection\_condition}(r)$
  - $\sigma_{Major='CS'}(STUDENT)$

```
SELECT  *
FROM    STUDENT
WHERE   Major = 'CS';
```


SELECT vs. WHERE

- **In SQL:**
  - **Selection *(σ)*** is expressed by the **`WHERE`** clause
  - **`SELECT`** clause actually does **Projection *(π)***
- It is a historical accident ☺

Basic *SQL*: Single Table Manipulation

```
SELECT   [DISTINCT|ALL] attribute-list | *
FROM     Table1
WHERE    selection-condition
```

- **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates
  - Default is that duplicates are not eliminated! Why?
- *Selection-Condition:* Comparisons
  - *expression* **op** *expression*
  - **op ∈ {<,<=,=,>,>=,<>}**
  - combined using **AND**, **OR** and **NOT**

Aliasing in SQL: The AS Operator

- Renaming attributes in the result of a query:

```
SELECT SID AS Student_ID
 FROM STUDENT;
```

- Table alias can be achieved with the AS operator in the FROM-clause: (Optional the AS)

```
SELECT S.Major
FROM STUDENT AS S
WHERE S.name = 'Ruchi Agrawal';
```

- Renaming of attributes within a query:

```
SELECT *
FROM STUDENT AS S(ID,FN,MJ)
WHERE S.FN = 'Thalia' AND S.MJ = 'COE';
```


Aggregate Functions

- Tuple grouping based on the value of some attributes.

```
SELECT   List of functions F(Attribute)
FROM     Table1
WHERE    selection-condition
```

- *F(B)* = aggregate function on attribute B
- SQL provides five aggregate functions:
  SUM, MAX, MIN, AVG, and COUNT [COUNT(□)]

Aggregate Functions… Example

- LIBRARIAN (SSN, Name, BirthDate, Gender, Salary, SNO);
- Q: Display all the statistics about librarian salaries.

```
SELECT SUM (Salary) AS TotalSalaries,
       MAX (Salary) AS MaxSalary,
       MIN (Salary) AS MinSalary,
       AVG (Salary) AS AvgSalary,
       COUNT (*)  AS Cardinality,
       COUNT(DISTINCT Salary) AS Salarylevels
FROM LIBRARIAN;
```

Note on COUNT

- COUNT (attribute-name) **does not** count NULLs
- COUNT (*) returns cardinality
- COUNT (DISTINCT attribute-name) returns the number of distinct values

Arithmetic Operator

- Arithmetic operators (+; -; *; /) may be applied on numeric values in any expression
- Q1: SELECT 1.1 * SUM (Salary)
  FROM LIBRARIAN;
- Increment (+) and decrement (-) may be applied on data types: date, time and timestamp
- Q2: SELECT Name, (CURRENT_DATE – BirthDate) AS Age
  FROM LIBRARIAN
  WHERE
  (CURRENT_DATE – BirthDate) INTERVAL YEAR > 35;

Grouping of Tuples

- Tuple grouping based on the value of some attributes.

```
SELECT    A-list,  F(B)
FROM      Table1
WHERE     selection-condition
GROUP BY  A-list
HAVING    Pred
```

- ***F(B)*** = aggregate function on attribute B
- *A-list*: The grouping attributes must appear in the SELECT-clause to be meaningful
- *Pred* = a predicate on the tuples of the individual groups

Grouping of Tuples… Example 1

- Example 1:

```
SELECT    DEPT, CLASS, COUNT(*) AS NoStudents
FROM      STUDENT
WHERE     QPA >= 3.5
GROUP BY  DEPT, CLASS;
```

- **WHERE** is evaluated first and then the grouping is done.

Grouping of Tuples… Example 2

```
SELECT    Dept, Class, COUNT(*) AS NoStudents
FROM      STUDENT
WHERE     QPA >= 3.5
GROUP BY  Dept, Class
HAVING    COUNT(*) >= 5;
```


Sorting the Result

- ORDER BY order-list
  - order-list: list of of <attribute,order> pairs.
  - order: ASC (default), DESC
  - attribute relative position is allowed: 2 ASC, 1 DESC
- Q: ?

```
SELECT    *
FROM      STUDENT
WHERE     QPA >= 3.5
ORDER BY  Lname ASC, Fname ASC, MI DESC;
```


Manipulating NULL Values

- NULL values must be considered explicitly
  - IS NULL and IS NOT NULL
- NULL in a condition yields UNKNOWN
- SQL provides operators to test for specific conditions
  - IS FALSE and IS NOT FALSE
  - IS TRUE and IS NOT TRUE
  - IS UNKNOWN and IS NOT UNKNOWN
- Query: ?

SELECT SID, Name
FROM STUDENT AS S
WHERE ((S.Major= 'CS)' and (S.Gender = 'F')) **IS NOT FALSE**;

Truth Tables

X	NOT
TRUE	FALSE
UNK	UNK
FALSE	TRUE

OR	TRUE	UNK	FALSE
TRUE	TRUE	TRUE	TRUE
UNK	TRUE	UNK	UNK
FALSE	TRUE	UNK	FALSE

AND	TRUE	UNK	FALSE
TRUE	TRUE	UNK	FALSE
UNK	UNK	UNK	FALSE
FALSE	FALSE	FALSE	FALSE

SQL CASE Statement & NULL

- Implements if-then-else functionality
- Easy way to handle NULLs
- Simple expression on equality:

```
SELECT SID, CASE Major
           WHEN IS NULL THEN 'undecided'
           WHEN 'CS' THEN 'good choice'
           ELSE 'recruit'
           END AS Strategy
FROM   STUDENT
WHERE CLASS = 'Sophomore';
```

- ELSE is Optional; all SIDs with unmatched Major are shown

SQL CASE Statement & NULL…

- Search, complex expression and beyond equality

```
SELECT  SID, CASE
            WHEN Major = 'CS' THEN 'good choice'
            WHEN Major IS NULL AND QPA > 3.25 THEN 'go after'
            WHEN Major IS NULL AND QPA > 2.75 THEN 'recruit'
            ELSE 'ignore'
            END AS Strategy
 FROM   STUDENT
 WHERE  CLASS = 'Sophomore';
```

- Alias for CASE is optional
- The value of the THEN could be of any type

Basic *SQL*: Two Table Manipulation

```
SELECT   [DISTINCT] attribute-list | *
FROM     table1, table2
WHERE    join-condition & selection-condition
```

- *Cartesian product: table1* X *table2* if no *Joint-Condition*
- *Joint-Condition:* Similar to selection-condition
  - *Expression-table1* **op** *expression-table2*
  - **op** ∈ {<,<=,=,>,>=,<>}
  - combined using **AND**

Relational Operators in SQL

- STUDENT(SID, Name, Major)
- ENROLLS(CID, SID, Term, Grade)
- STUDENT X ENROLLS

```
SELECT  STUDENT.*, ENROLLS.*
FROM    STUDENT, ENROLLS;
```

- STUDENT $\bowtie_{SID=SID}$ ENROLLS

```
SELECT  S.*, E.*
FROM    STUDENT S, ENROLLS E
WHERE   S.SID  = E.SID;
```

Join Operator (SQL2)

- JOIN was introduced for specifying the join conditions in the FROM-clause:

  table1 **JOIN** table2 **ON** join-condition

- Example of Condition-Join:

```
SELECT  SID, S.Name, Term
FROM    (STUDENT S JOIN ENROLLS E ON S.SID=E.SID)
WHERE   E.CID = 1555;
```


Natural Join

- NATURAL JOIN (without ON-clause)

  table1 **NATURAL JOIN** table2

- Use renaming of attribute if there is a need, e.g.,

```
SELECT  *
FROM    (LIBRARIAN NATURAL JOIN SECTION AS
                                S(SNO, SName, Head))
WHERE   SName = 'Children';
```

- Natural join over some attributes: **USING** (attribute-list)

```
SELECT  SID, SName, Term
FROM    (STUDENT NATURAL JOIN ENROLLS USING (SID))
WHERE   ENROLLS.CID = 1555;
```


Other Join Operations

- Outer Join operators:
  - LEFT OUTER JOIN or LEFT JOIN
  - RIGHT OUTER JOIN or RIGHT JOIN
  - FULL OUTER JOIN or FULL JOIN
  - NATURAL LEFT OUTER JOIN or NATURAL LEFT JOIN
  - NATURAL RIGHT OUTER JOIN or NATURAL RIGHT JOIN
  - NATURAL FULL OUTER JOIN or FULL JOIN
- CROSS JOIN: generates a cross product
- UNION JOIN: Outer Union operator

Outer Join Examples

- STUDENT(SID, Name, Class, Major)
  ENROLLS(CID, SID, Term, Grade)
- Q1:

```
SELECT  *
FROM    (STUDENT S LEFT OUTER JOIN ENROLLS E
                                ON S.SID=E.SID)
ORDER BY  S.SID;
```

- Q2:

```
SELECT  SID, S.Name, S. Major
FROM    STUDENT S NATURAL LEFT OUTER JOIN ENROLLS E
WHERE   E.Term IS NULL;
```

Outer Join Q1 Execution

Students

SID	Name	Class	Major
123	John	3	CS
124	Mary	3	CS
999	Newman	1	CS

Enroll

SID	CID	Term	Grade
123	CS1520	Fall 10	3.75
124	CS1520	Fall 10	4
123	CS1555	Fall 10	4
124	CS1555	Fall 10	NULL

Q1 RESULT

S.SID	S.Name	S.Class	S. Major	E.SID	E.CID	E.Term	E.Grade
123	John	3	CS	123	CS1520	Fall 10	3.75
123	John	3	CS	123	CS1555	Fall 10	4
124	Mary	3	CS	124	CS1520	Fall 10	4
124	Mary	3	CS	124	CS1555	Fall 10	NULL
999	Newman	1	CS	NULL	NULL	NULL	NULL

Outer Join Q2 Execution

```
SELECT  SID, S.Name, S. Major
FROM    STUDENT S NATURAL LEFT OUTER JOIN ENROLLS E
WHERE   E.Term IS NULL;
```

Students

SID	Name	Class	Major
123	John	3	CS
124	Mary	3	CS
999	Newman	1	CS

Enroll

SID	CID	Term	Grade
123	CS1520	Fall 10	3.75
124	CS1520	Fall 10	4
123	CS1555	Fall 10	4
124	CS1555	Fall 10	NULL

Q2 RESULT

S.SID	S.Name	S. Major
999	Newman	CS

Old Oracle SQL (+) operator

- (+) operator was introduced to express outer join by annotating the side with the optional table to be ignored
- E.g., STUDENT *left-outer-join* ENROLLS

```
SELECT e.CID, s.SID
FROM STUDENT s, ENROLLS e
WHERE s.SID = e.SID(+) AND term=`Fall 2013';
```


Set Operations

- SQL supports UNION, EXCEPT (difference), INTERSECT (not all vendors)
- UNION ALL retains duplicates
- Tables must be *union-compatible*

```
( SELECT SID
  FROM   STUDENT
  WHERE  Major = 'CS' )
UNION
( SELECT SID
  FROM   STUDENT
  WHERE  Major = 'Math' );
```

```
( SELECT SID
  FROM   STUDENT)
EXCEPT
( SELECT SID
  FROM   STUDENT
  WHERE  Major = 'Math' );
```

Merging Fields in Queries

- String Concatenation is denoted by two vertical bars (||)
  - || merges into a single string, one or more strings
- E.g., Display in a single value (one column) the name of all students in CS 2550 and their phone numbers

```
SELECT  Fname || ' ' || Lname AS Name, PhoneNumber
FROM    STUDENT NATURAL JOIN ENROLLS
WHERE   Dept || CourseNumber = 'CS2550';
```


Range Queries & Range Conditions

- BETWEEN, and its negation NOT BETWEEN, can be used with numeric, character and datetime datatypes
- Simplify the formulation of conjunction expressions
- E.g.,

```
SELECT *
FROM   LIBRARIAN
WHERE  (Salary >= 25000 AND Salary <= 35000);

SELECT *
FROM   LIBRARIAN
WHERE  (Salary BETWEEN 25000 AND 35000);
```


Partial Queries - Pattern Matching

- **LIKE** supports comparisons with partial strings
  - A percent sign `%' indicates a match with an arbitrary number of characters including spaces
    - Note that '*' is not valid
  - An underscore sign '_' matches a single arbitrary character
- Retrieve all students with Pitt phone extension

```
SELECT  Name
FROM    STUDENT
WHERE   Phone  LIKE  '412.62%';
```


Pattern Matching…

- Retrieve all students with *local* phone numbers (any area code) which start with 6 and whose third digit is 3.

```
SELECT  Name
FROM    STUDENT
WHERE   Phone  LIKE  '___6_3%';
```

- Escape defines the escape character that causes SQL to interpret a wildcard char (%) as itself in a string:

```
SELECT  VideoName
FROM    RENTALS
WHERE   Discount  LIKE  '10&%'  ESCAPE  '&';
```

Regular expression functions - Oracle

- **REGEXP_LIKE**(**x**, **pattern** [, **match_option**])
  - True if the source **x** matches the regular expression **pattern**
  - **match_option** can change the default matching:
    - 'c', specifies case sensitive matching (default)
    - 'i', specifies case insensitive matching
    - 'n', allows you to use the match-any-character operator
    - 'm', treats x as multiple line

Regular Expression Meta-characters

Meta-characters	Meaning	Examples
\	the match character is a special character, a literal, or a backreference	\n matches newline, \\ matches \, \(matches (
^	Matches the position at the start of the string	^A matches A if A is the 1st char in the string
$	Matches the position at the end of the string	$B matches B if B is the last char in the string
*	Matches the preceding character zero or more times	ba*rk matches brk, bark, baark, etc.
+	Matches the preceding character one or more times	ba+rk matches bark, baark, etc., but not brk.
?	Matches the preceding character zero or one time	ba?rk matches brk and bark only

Regular Expression Meta-characters…

Meta-characters	Meaning	Examples
{n}	Matches a character exactly n times, where n is an integer	hob{2}it matches hobbit
{n,m}	Matches a character at least n times and at most m times, where n and m are both integers	hob{2,3}it matches hobbit and hobbbit only
.	Matches any single character except null	hob.it matches hobait, hobbit, etc.
(pattern)	A subexpression that matches the specified pattern	anatom(ylies) matches anatomy and anatomies
xly	Matches x or y, where x and y are one or more characters	warlpeace matches war or peace
[abc] or [a-z]	Matches any of the enclosed characters or the specified range	[ab]bc matches abc and bbc; [a-c]bc matches abc, bbc, and cbc

REGEXP_INSTR function - Oracle

- **REGEXP_INSTR**(x, pattern [, start [, occurrence [, return_option [, match_option]]]])
  - Searches for pattern in x and returns the position at which pattern occurs.
  - Options:
    1. start position to begin the search.
    2. occurrence that indicates which occurrence of pattern_exp should be returned.
    3. return_option that indicates what integer to return.
       - 0 specifies the integer to return is the position of the first character in x;
       - non-zero specifies the integer to return is the position of the character in x after the occurrence.
    4. match_option to change the default matching.

Regular expression functions - Oracle

- **REGEXP_REPLACE**(x, pattern [, replace_string [, start [, occurrence [, match_option]]]])
  - Searches x for pattern and replaces it with replace_string
- **REGEXP_SUBSTR**(x, pattern [, start [, occurrence [, match_option]]])
  - Returns a substring of x that matches pattern, which begins at the position specified by start