SQL-1

SQL Characteristics

- Free form language, statement terminated with semi-colon ;
  - Select empno, empame from employee ..
    vs
  - Select
    empno,
    empname
    from
    employee …. *(preferred style, more readable)*
- Most components are case insensitive
  - SElECT vs select
  - Important exception when searching for a value
    - literal character data must match the exact case in the database
      - SMITH vs smith vs Smith (all different database values)
- Data representation in SQl commands
  - Non numeric data must be enclosed in single quotes - 'Smith'
  - Numeric data must not be enclosed in quotes - 12345

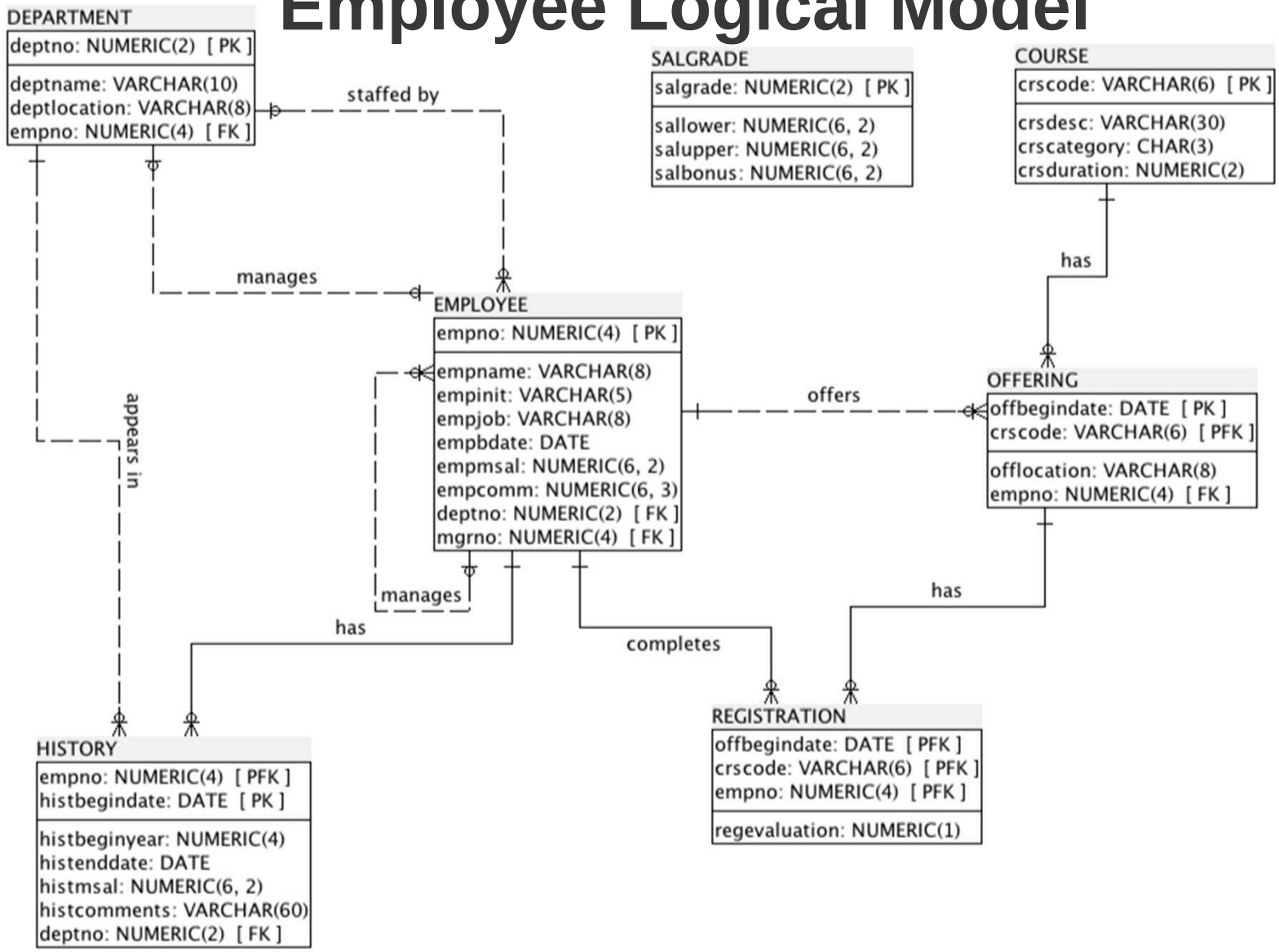
SELECT statement

- Is used to query the database and retrieve selected data that match the criteria that you specify
- In its simplest form a SELECT statement must include the following:
  - A SELECT clause, which specifies the columns to be displayed
  - A FROM clause, which specifies the table containing the columns listed in the SELECT clause

SELECT [DISTINCT] {*, *column* [*alias*],...}

FROM *table;*

Employee Logical Model



SELECT statement

```
SELECT * FROM employee;

SELECT empname, empjob
FROM employee;

SELECT DISTINCT deptno
FROM employee;

SELECT DISTINCT deptno, empjob
FROM employee;

SELECT DISTINCT deptno, empjob, empmsal
FROM employee;

SELECT empname AS "Employee", empmsal  AS "Monthly Salary", empcomm
FROM employee;
```

- note AS is optional (*but recommended*)

WHERE clause

- restricts the rows returned from the query
- contains a condition that must be met
- directly follows the FROM clause
- can compare values in columns, literal values, arithmetic expressions, or functions

SELECT **[DISTINCT] {*, *column* [*alias*],...}**
FROM ***table***
[WHERE ***condition(s)*****]**

- consists of three elements
  - Column name
  - Comparison operator
  - Column name, constant, or list of values

Comparison operators

- Comparison operators
  - are used in conditions that compare one expression to another

Syntax: **… WHERE *expr operator value***

= *Equal*

> *Greater than*

< *Less than*

>=*Greater than or equal*

<=*Less than or equal*

<>*Not equal to*

Comparison operators

```
SELECT empname, mgrno, deptno, empjob
FROM employee
WHERE empjob = 'TRAINER';

SELECT empname, mgrno, deptno, empjob
FROM employee
WHERE empjob = 'Salesrep';
            vs WHERE empjob = 'SALESREP'

SELECT empname, mgrno, deptno, empjob
FROM employee
WHERE deptno = 20;
```

Note in the following two selects empbdate is a date, '31-DEC-1965' and '01/12/1965' are strings:

```
SELECT empname, empbdate, mgrno, deptno, empjob
FROM employee
WHERE empbdate < '31-DEC-1965';

SELECT empname, empbdate, mgrno, deptno, empjob
FROM employee
WHERE empbdate > '01/12/1965';
```

Success depends on Oracle 'standard' date format, unless conversion functions are used - *we will return to this*

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE empmsal >= 1250;
```

Special operators

- LIKE operator
  - Used in the WHERE clause in combination with a search pattern
  - % A percent sign before or after the LIKE operator means zero, one, or more arbitrary characters
  - _ An underscore before or after the LIKE operator means exactly one arbitrary character

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE empjob LIKE '%SALES%';
```

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE empname LIKE 'S%';
```

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE empname LIKE '%N';
```

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE empname LIKE '_A%';
```

Special operators

- IN operator
  - Used in the WHERE clause to compare a column expression against a list of values

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE empjob IN ('TRAINER', 'MANAGER');
```

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE mgrno NOT IN (7698, 7839);
```

Special operators

- IS NULL operator
  - Used in the WHERE clause to check whether an attribute is null
  - Only has one operand: the preceding column name or column expression

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE empcomm IS NULL;
```

Note you cannot use = to test for a null

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE empcomm = NULL;
```

will return no rows, but not an error (it is syntactically correct)

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE empcomm IS NOT NULL;
```

Special operators

- BETWEEN operator
  - Used in the WHERE clause to check whether an attribute value is within a range of values (inclusive)

```
SELECT empname, mgrno, deptno, empjob, empbdate
FROM employee
WHERE empbdate BETWEEN '10-JAN-50' AND '26-NOV-59';
```

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE empmsal NOT BETWEEN 1500 AND 2500;
```

Note values in between clause must be in ascending order:

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE empmsal BETWEEN 1500 AND 1250;
```

will return no rows, but not an error (it is syntactically correct)

Logical operators

- AND operator
  - **requires both conditions to be TRUE**

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE deptno = 30
   AND empjob = 'SALESREP';
```

```
SELECT empname, mgrno, deptno, empjob, empmsal, empbdate
FROM employee
WHERE deptno = 30
   AND empjob = 'SALESREP'
   AND empbdate < '01-JUN-65';
```

Logical operators

- OR operator
  - **requires either condition to be TRUE**

```
SELECT empname, mgrno, deptno, empjob, empmsal
FROM employee
WHERE deptno = 30
   OR empjob = 'SALESREP';

SELECT empname, mgrno, deptno, empjob, empmsal, empbdate
FROM employee
WHERE deptno = 30
   OR empjob = 'SALESREP'
   OR empmsal < 1500;
```

Logical operators

- NOT operator
  - Can be applied to any arbitrary condition to negate that condition

```
SELECT empname, mgrno, deptno, empjob, empmsal, empbdate
FROM employee
WHERE NOT deptno = 30;
```

```
SELECT empname, mgrno, deptno, empjob, empmsal, empbdate
FROM employee
WHERE NOT empbdate < '01-JAN-65';
```

Rules of Precedence

- When writing an SQL query using operators it is important to be aware of how the query is evaluated

ORDER EVALUATED	OPERATOR
1	ALL Comparison Operators
2	NOT
3	AND
4	OR

- To override the order of precedence use parentheses around the clauses – *suggest always use to clarify requirement*

Arithmetic Operators

- Can be applied to NUMBER values and to some extent DATE values

```
SELECT empname, empmsal, empmsal*12 as "Annual Salary"
FROM employee;
```

- If you subtract two DATE values you get the difference between the two dates in days (SYSDATE is an Oracle function which returns the current system date)

```
SELECT empname, empbdate, SYSDATE - empbdate as "Age in
    Days"
FROM employee;
SELECT empname, empbdate, (SYSDATE - empbdate)*24  as "Age in
    Hours"
FROM employee;
```

- You can add a DATE and an INTERVAL value, which results in another date

```
SELECT empname, empbdate, empbdate + INTERVAL '10'
YEAR FROM employee;
```

Arithmetic with Dates

- If you add or subtract a DATE and a NUMBER, the number is interpreted as an interval expressed in days

```
SELECT empname, empbdate, empbdate + 365
FROM employee;
```

- The DATE data type stores date and time information
- Oracle stores dates in an internal numeric format: century, year, month, day, hours, minutes, seconds.
- SYSDATE is a function that returns the current date and time.
- DUAL is a dummy table used to view SYSDATE.

```
SELECT SYSDATE
FROM dual;
```

Nulls in Arithmetic Expressions

- Problem - arithmetic expressions containing a null value evaluate to null:

```
SELECT empname, empmsal, empcomm,
                12*empmsal+empcomm as "Annual Salary"
FROM employee;
```

- NVL function
  - The NVL function forces arithmetic operators to include null values and replace the null with the listed value

```
SELECT empname, empmsal, empcomm,
                12*empmsal+NVL(empcomm,0) as "Annual Salary"
FROM employee;
```

NVL(expr, y)

- if expr is NULL, returns y; otherwise returns expr

Concatenation Operator

- SQl supports one alphanumeric operator that allows you to concatenate string expressions
- Concatenates columns or character strings to other columns
- Is represented by two vertical bars (II)
- Creates a resultant column that is a character expression

```
SElECT empinit II empname
FROM employee;
```

Literal Character Strings

- A literal is a character, expression, or number included in the SElECT list.

```
SElECT 'Employee ' II empname II ' is number ' II empno
FROM employee;
```

- Character literal values must be enclosed within *single* quotation marks.
- Each character string is output once for each row returned.

```
SElECT empname II ' is a ' II empjob
FROM employee;
```

ORDER BY

- The order of rows returned in a query result is undefined
  - Week 3 – property of relational model "Tuples are **unordered** within a relation"
- The ORDER BY clause must ***always*** be used to sort the rows (unless you are sure there is only a single row returned)

```
SELECT      expr
FROM        table
[WHERE      condition(s)]
[ORDER BY   {column, expr} [ASC|DESC]];
```

  - ASC
    - orders the rows in ascending order (this is the default order)
  - DESC
    - orders the rows in descending order

ORDER BY

```
SElECT empname, mgrno,  deptno, empjob, empmsal, empbdate
FROM employee
WHERE deptno = 30
ORDER BY empjob;
```

```
SElECT empname, mgrno,  deptno, empjob, empmsal, empcomm
FROM employee
ORDER BY deptno, empcomm;
```

- How do we treat NUll values when sorting?
  - Always as first values or last values (regardless of sorting order)?
  - As low or high values?
- What is ORAClE's default behaviour for sorting NUlls?

```
SElECT empname, mgrno, deptno, empjob, empmsal, empbdate
FROM employee
ORDER BY deptno, empjob, empmsal DESC;
```