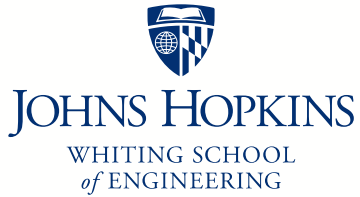
Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 1 - 6

SQL - The Relational DB Language II



Join in SQL

- JOIN
  - Combine rows from two or more relations in a database
  - Conceptually easier and more obvious for the programmer than a correlated subquery
  - Exploit obvious relationships and can be used to discover new relationships

Join in SQL (cont.)

- JOIN (cont.)
  - Can join on any columns in tables
    - if joined columns match data types
    - if join operation makes sense
    - the joined columns don’t need to be key attributes
  - Joined columns are key attributes in general (e.g., PK and FK)
  - Consume system resources (memory and CPU time)

Join in SQL (cont.)

- JOIN (cont.)
  - In COMPANY database, for every project located in 'Stafford', list the project number, the controlling department, and the department manager's last name, address, and birth date

Traditional JOIN

```
SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE Dnum = Dnumber AND
      Mgr_ssn = Ssn AND
      Plocation = 'Stafford';
```

ANSI JOIN ON

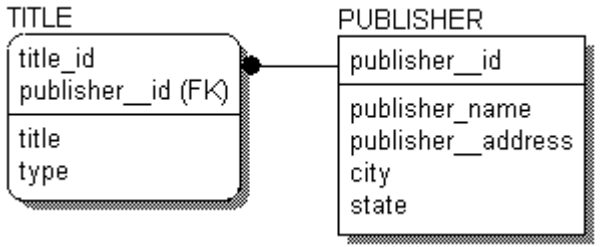
```
SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM PROJECT JOIN DEPARTMENT ON Dnum = Dnumber
     JOIN EMPLOYEE ON Mgr_ssn = Ssn
WHERE Plocation = 'Stafford';
```

Join vs. Subquery

- Join vs. Subquery
  - May use both joins and subqueries to query multiple tables. In general, they can be used interchangeably to solve a given problem.

Example: Retrieve publishers who publish the ‘education’ type titles

```
SELECT publisher_name
FROM   publisher p, title t
WHERE  p.publisher_id = t.publisher_id AND
                t.type = 'education';
```



Join vs. Subquery (cont.)

- Join vs. Subquery (cont.)
  - Subquery can calculate an aggregate value (e.g., max, min, avg, sum, and count) on the fly and feed it back to the outer query for comparison.
  - Subquery is a good choice when you need to compare aggregates to other values.

Example: Retrieve the book title(s) with *a lowest price* in the catalog

```
SELECT title, price
FROM catalog
WHERE price = SELECT MIN(price) FROM catalog;
```

Join vs. Subquery (cont.)

- Join vs. Subquery (cont.)
  - Join operation provides additional options to let you edit the results from two joined tables. Join is a good choice when you want to display results from multiple tables.

Example: Retrieve publishers and authors who live in the same city

1. What is the relationship between publisher and author tables?
2. How to handle a many-to-many relationship?

```
SELECT publisher_name, author_name
FROM publisher P, author A
WHERE P.city = A.city;
```

Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 2

SQL - The Relational DB Language II

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Cross Join

- Cross Join
  - CROSS JOIN returns the Cartesian product of two tables.

    Traditional Way:

    ```
    SELECT ColumnList FROM table1, table2
    ```

    Note: A query for getting information from two tables without a join condition may be a programming mistake.

    SQL Standard Way:

    ```
    SELECT [DISTINCT|ALL] {*|ColumnList}
    FROM table1 CROSS JOIN table2
    ```

```
SQL> SELECT * FROM emp, dept;

SQL> SELECT * FROM emp CROSS JOIN dept;
. . .
56 rows selected. (14 employees and 4 departments)
```

Inner Join

- Inner Join
  - **INNER JOIN** with **ON** returns only the rows that meet the join condition indicated in the ON clause

```
SELECT [DISTINCT|ALL]
{*|ColumnList}
FROM table1 [INNER] JOIN table2 ON
     table1.c1 = table2.c1;
```

```
SQL> SELECT empno, ename, emp.deptno, dname, loc
  2  FROM emp JOIN dept ON emp.deptno = dept.deptno;

     EMPNO ENAME          DEPTNO DNAME          LOC
---------- ---------- ---------- -------------- -----------
      7369 SMITH              20 RESEARCH       DALLAS
      7499 ALLEN              30 SALES          CHICAGO
      7521 WARD               30 SALES          CHICAGO
      7566 JONES              20 RESEARCH       DALLAS
      7654 MARTIN             30 SALES          CHICAGO
      7698 BLAKE              30 SALES          CHICAGO
      7782 CLARK              10 ACCOUNTING     NEW YORK
      7788 SCOTT              20 RESEARCH       DALLAS
      7839 KING               10 ACCOUNTING     NEW YORK
      7844 TURNER             30 SALES          CHICAGO
      7876 ADAMS              20 RESEARCH       DALLAS
      7900 JAMES              30 SALES          CHICAGO
      7902 FORD               20 RESEARCH       DALLAS
      7934 MILLER             10 ACCOUNTING     NEW YORK
```

Inner Join (cont.)

- Inner Join (cont.)
  - Traditional **JOIN** returns only the rows that meet the join condition in the WHERE clause

```
SELECT [DISTINCT|ALL]
{*|ColumnList}
FROM table1, table2
WHERE table1.c1 = table2.c1;
```

```
SQL> SELECT empno, ename, emp.deptno, dname, loc
  2  FROM emp, dept
  3  WHERE emp.deptno = dept.deptno;
     EMPNO ENAME          DEPTNO DNAME          LOC
---------- ---------- ---------- -------------- -----------
      7369 SMITH              20 RESEARCH       DALLAS
      7499 ALLEN              30 SALES          CHICAGO
      7521 WARD               30 SALES          CHICAGO
      7566 JONES              20 RESEARCH       DALLAS
      7654 MARTIN             30 SALES          CHICAGO
      7698 BLAKE              30 SALES          CHICAGO
      7782 CLARK              10 ACCOUNTING     NEW YORK
      7788 SCOTT              20 RESEARCH       DALLAS
      7839 KING               10 ACCOUNTING     NEW YORK
      7844 TURNER             30 SALES          CHICAGO
      7876 ADAMS              20 RESEARCH       DALLAS
      7900 JAMES              30 SALES          CHICAGO
      7902 FORD               20 RESEARCH       DALLAS
      7934 MILLER             10 ACCOUNTING     NEW YORK
```

Inner Join (cont.)

- Inner Join (cont.)
  - **JOIN** with **USING** returns only the rows with matching values in the *common* columns indicated in the USING clause

```
SQL> SELECT empno, ename, emp.deptno, dname, loc
  2  FROM emp JOIN dept USING(deptno);
     EMPNO ENAME          DEPTNO DNAME          LOC
---------- ---------- ---------- -------------- -----------
      7369 SMITH              20 RESEARCH       DALLAS
      7499 ALLEN              30 SALES          CHICAGO
      7521 WARD               30 SALES          CHICAGO
      7566 JONES              20 RESEARCH       DALLAS
      7654 MARTIN             30 SALES          CHICAGO
      7698 BLAKE              30 SALES          CHICAGO
      7782 CLARK              10 ACCOUNTING     NEW YORK
      7788 SCOTT              20 RESEARCH       DALLAS
      7839 KING               10 ACCOUNTING     NEW YORK
      7844 TURNER             30 SALES          CHICAGO
      7876 ADAMS              20 RESEARCH       DALLAS
      7900 JAMES              30 SALES          CHICAGO
      7902 FORD               20 RESEARCH       DALLAS
      7934 MILLER             10 ACCOUNTING     NEW YORK
```

```
SELECT [DISTINCT|ALL]
{*|ColumnList}
FROM table1 JOIN table2 USING (c1);
```

Natural Join

- Natural Join
  - **NATURAL JOIN** returns only the rows with matching values in the matching columns. The matching columns must have the same names and similar data types

```
SELECT [DISTINCT|ALL] {*|ColumnList}
FROM table1 NATURAL JOIN table2
```

  - The NATURAL and USING keywords are mutually exclusive. They cannot be used together.

Natural Join (cont.)

- Natural Join (cont.)
  - With NATURAL JOIN only:

```
SQL> SELECT empno, ename, deptno, dname, loc
  2  FROM emp NATURAL JOIN dept
  3  WHERE deptno = 30;

     EMPNO ENAME          DEPTNO DNAME          LOC
---------- ---------- ---------- -------------- -------------
      7499 ALLEN              30 SALES          CHICAGO
      7521 WARD               30 SALES          CHICAGO
      7654 MARTIN             30 SALES          CHICAGO
      7698 BLAKE              30 SALES          CHICAGO
      7844 TURNER             30 SALES          CHICAGO
      7900 JAMES              30 SALES          CHICAGO

6 rows selected.
```

- Natural Join (cont.)
  - With NATURAL JOIN and WHERE clause:

```
SQL> SELECT *
  2  FROM emp NATURAL JOIN dept
  3  WHERE emp.deptno = 30;
SELECT * FROM emp NATURAL JOIN dept WHERE emp.deptno = 30
                                          *
ERROR at line 1:
ORA-25155: column used in NATURAL join cannot have qualifier
```

Outer Join

- Outer Join
  - **OUTER JOIN** returns not only the rows matching the join condition, but also the rows with unmatched values.
    - **LEFT OUTER JOIN** returns rows with matching values and the rows in table1 without matching values.

```
SELECT *
FROM table1 LEFT [OUTER] JOIN table2 ON
     table1.c1 = table2.c1;
```

    - **RIGHT OUTER JOIN** returns rows with matching values and the rows in table2 without matching values.

```
SELECT *
FROM table1 RIGHT [OUTER] JOIN table2 ON
     table1.c1 = table2.c1;
```

Outer Join (cont.)

- Outer Join (cont.)
  - **FULL OUTER JOIN** returns rows with matching values and the rows in table1 as well as table2 without matching values.

```
SELECT *
FROM table1 FULL [OUTER]
   JOIN table2 ON
      table1.c1 = table2.c1;
```

```
SQL> SELECT *
  2  FROM emp LEFT OUTER JOIN dept ON (emp.deptno = dept.deptno);
...
14 rows selected.

QL> SELECT *
  2 FROM emp RIGHT OUTER JOIN dept oN (emp.deptno = dept.deptno);
...
15 rows selected.

SQL> SELECT *
  2  FROM emp FULL OUTER JOIN dept ON (emp.deptno = dept.deptno);
...
15 rows selected.
```

Self Join

- **SELF JOIN** has a table that joins itself, a recursive relationship.
  - List all employees and their supervisors.

```
SQL> SELECT emp.empno, emp.ename, mgr.empno AS manger_empno, mgr.ename AS mgr_name
  2  FROM emp JOIN emp mgr ON (emp.mgr = mgr.empno);
     EMPNO ENAME      MANGER_EMPNO MGR_NAME
---------- ---------- ------------ ----------
      7369 SMITH              7902 FORD
      7499 ALLEN              7698 BLAKE
      7521 WARD               7698 BLAKE
      7566 JONES              7839 KING
      7654 MARTIN             7698 BLAKE
      7698 BLAKE              7839 KING
      7782 CLARK              7839 KING
      7788 SCOTT              7566 JONES
      7844 TURNER             7698 BLAKE
      7876 ADAMS              7788 SCOTT
      7900 JAMES              7698 BLAKE
      7902 FORD               7566 JONES
      7934 MILLER             7782 CLARK
13 rows selected.
```

Recursive Closure Operation

- Recursive Closure Operation
  - For a recursive relationship, a query to get a hierarchical relation among records.
  - This is a hierarchical query and SQL standard which uses recursive common table expressions: http://msdn.microsoft.com/en-us/library/ms186243.aspx
    Example: Retrieve all employees that work under “JONES” using Oracle:

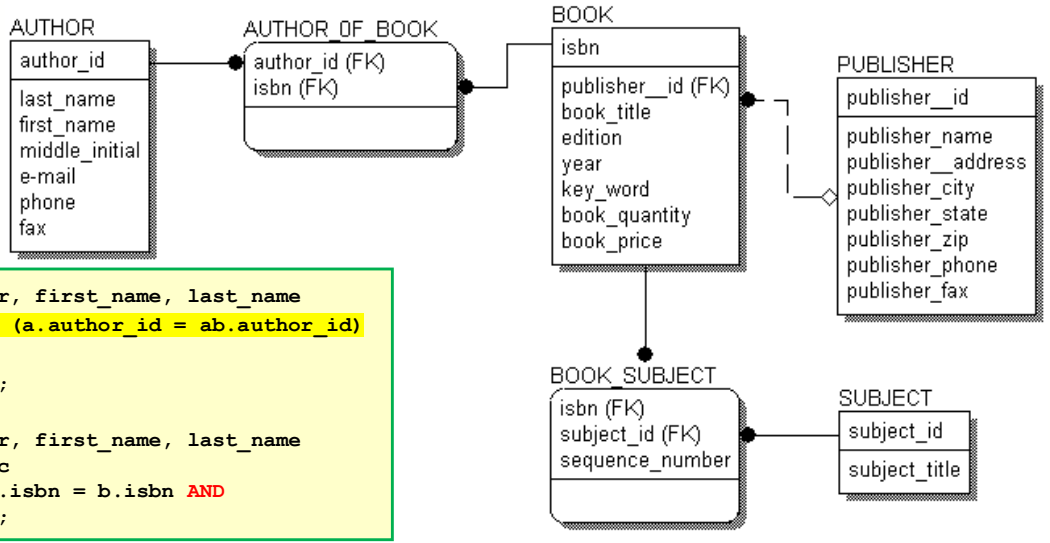
```
SQL> SELECT LEVEL, LPAD (' ', 2 * (LEVEL - 1)) || ename "employee", empno, mgr
     FROM emp START WITH ename = 'JONES'
     CONNECT BY PRIOR empno = mgr;

     LEVEL employee                 EMPNO        MGR
---------- -------------------- ---------- ----------
         1 JONES                      7566       7839
         2   SCOTT                    7788       7566
         3     ADAMS                  7876       7788
         2   FORD                     7902       7566
         3     SMITH                  7369       7902
```

Join for a Many-to-Many relationship

- Join three tables for a M:N relationship

BOOK STORE or LIBRARY ERD (A Subset)




```
SELECT b.isbn, book_tilte, edition, year, first_name, last_name
FROM author a JOIN author_of_book ab ON (a.author_id = ab.author_id)
     JOIN book c ON (ab.isbn = b.isbn)
WHERE book_title = 'One Minute Manager';

SELECT b.isbn, book_tilte, edition, year, first_name, last_name
FROM author a, author_of_book ab, book c
WHERE a.author_id = ab.author_id AND ab.isbn = b.isbn AND
      book_title = 'One Minute Manager';
```

Q1: Find out an author(s) for a book titled 'One Minute Manger' and the book's related information

Q2: Find out an author(s) for all books order by title name and the book's related information

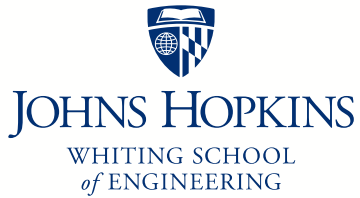
Q3: Find out books by a specific author 'John Smith' and the book's related information

Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 3

SQL - The Relational DB Language II



CASE Expression

- CASE Expression

```
CASE ColumnName
   WHEN condition1 THEN result1
   WHEN condition2 THEN result2
   ...
   ELSE result
END
```

```
SELECT fname, lname, dept_id,
     (CASE title
         WHEN LIKE 'Senior%' THEN '10%'
         WHEN LIKE 'Junior%' THEN '5%'
         ELSE '0%'
      END) AS bonus
FROM employee;
```

```
SELECT fname, lname,
   (CASE dept_id
       WHEN 1 THEN 'Accounting'
       WHEN 2 THEN 'Finance'
       WHEN 3 THEN 'Engineering'
       ELSE 'Not required'
   END) AS department
FROM employee;
```

Data Manipulation - Insertion

- Data Manipulation - Insertion
  - Add a single row to a table

```
INSERT INTO TableName [ColumnList] VALUES (valuelist);
```

```
INSERT INTO emp
VALUES (7954, 'CARTER', 'CLERK', 7698, '7-APR-82', 1000, NULL, 30);

INSERT INTO emp(empno, ename, job, mgr)
VALUES (7980, 'SMITH', 'MANAGER', 3839);
```

  - Add multiple rows from one table to another table

```
INSERT INTO TableName [ColumnList]
SELECT ...;
```

```
INSERT INTO bonus (ename, job, sal, comm)
   SELECT ename, job, sal, comm
   FROM emp
   WHERE job ='MANAGER' OR
         comm > 0.25 * sal;
```

  - Add/copy some or all columns from one table into a new table

```
SELECT {*|ColumnList} INTO NewTable FROM OldTable WHERE condition;
```

Data Manipulation - Modification

- Data Manipulation - Modification
  - Change the contents of existing rows

```
UPDATE TableName
SET ColumnName1 = value1 [, ...]
WHERE ...
```

  - Update a field in one row of a table

```
UPDATE emp
SET sal = 3300
WHERE empno = 7788;
```

  - Update multiple fields of multiple rows in a table

```
UPDATE emp
SET sal = sal x 1.15, comm = comm + 100
WHERE 'SALESMAN' AND deptno = 30;
```

MERGE

- MERGE (also called upsert) statements to INSERT new records or UPDATE existing records depending on whether condition matches.

```
MERGE INTO tablename USING table_reference ON (condition)
   WHEN MATCHED THEN
   UPDATE SET column1 = value1 [, column2 = value2 ...]
   WHEN NOT MATCHED THEN
   INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...]);
```

Tablename is the target and table_reference is the source (table/view/subquery).
Check your RDBMS manual to see whether it supports standard MERGE syntax.

Data Manipulation – Deletion

- Data Manipulation - Deletion
  - Delete rows from a table

```
DELETE FROM TableName
WHERE ...
```

  - Delete one row of a table

```
DELETE FROM emp
WHERE ename = 'WARD';
```

  - Delete multiple rows in a table with or without a condition

```
DELETE FROM bonus;  -- delete all rows in the bonus table

DELETE FROM bonus
WHERE job IN
      (SELECT job
       FROM emp
       WHERE empno = 7566);
```

Data Manipulation – Deletion (cont.)

- Delete rows from a table using TRNCATE TABLE.
  - Is a Data Definition Language (DDL) operation
  - Quickly removes all data from a table, typically bypassing a number of integrity enforcing mechanisms
  - Needs to check RDMBS specific implementations

```
TRNCATE TABLE TableName
```

Note: Use TRUCATE TABLE statement carefully; particularly for a production system. This is a best practice that one may consider doing a backup on a production system before executing TRUNCATE TABLE or DROP TABLE DDLs.

Views in SQL

- A view is a virtual or derived table. The rows of a view do not exist until they are derived from base tables at run time.
- Changes to any of the base tables in the defining query are immediately reflected in the view.
- A view presents current and dynamic information to users regardless of the constantly changing underlying source tables.

Views in SQL (cont.)

- Views can be used for:
  - Restricting access (additional level of table security)
  - Providing referential integrity
  - Presenting tables to users in various forms
  - Pre-joining base tables for easily developing an application
  - Pre-packaging complex queries

Views in SQL (cont.)

- Create a view by embedding a subquery for a subset of columns and/or rows, column expressions from one or more tables or views:

CREATE VIEW ViewName AS subquery

Example: Create a view with employee last name, firstname, and department name

```
CREATE VIEW v_emp_dept
AS SELECT emp.lname, emp.fname, dept.d_name
   FROM emp, dept
   WHERE emp.dept_id = dept.dept_id
```

Views in SQL (cont.)

- Views in practice:
  - Creating a join view with joined tables (e.g., AUTHOR, AUTHER_OF_BOOK, and BOOK) to reduce query complexity
  - Creating views for security and role-based access controls
    - An EMP view excluding sensitive employee attributes
  - Creating views to support multi-user database for customization
    - Views for marketing, sales, customer, administration roles

Views in SQL (cont.)

- Updating a view to a base table may be simple. However, not all views can be updated:
  - Expressions
  - Aggregate functions
  - References to views that are not updatable
  - GROUP BY or HAVING clauses
  - Set operations with multiple tables

View Materialization

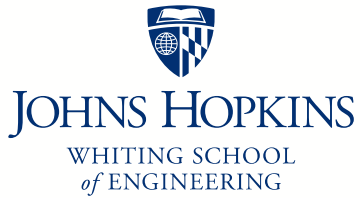
- View Materialization – Materialized View
  - View may cause performance issues. View materialization stores the view as a temporary table.
  - Oracle uses a materialized view (used to called a snapshot) containing the results of a query.
    - A materialized view can be used to store copies of remote data on a local system for replication purposes.
    - A materialized aggregated view or joined view can be used for data warehouse purposes as a fact table.

Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 2

SQL - The Relational DB Language II



Indexing to Improve Query Performance

- Create indexes on a column(s) on database tables. Indexes are an explicitly created schema object. They are stored independently.

  Syntax:

```
CREATE INDEX index_name
ON table_name(column_name)
```

  Example:

```
CREATE INDEX emp_lname ON emp(lname);
```

  Note: These statements have been *removed* from SQL2 because they specify physical access paths - not conceptual or logical concepts.

Indexing to Improve Query Performance (cont.)

- Create unique indexes for rows with unique values in the indexed column(s). A column with unique value normally is important for query and it is good to create a unique index.

  Syntax:

```
CREATE UNIQUE INDEX index_name
ON table_name(column_name)
```

  Example:

```
CREATE UNIQUE INDEX emp_ssn ON emp(ssn);
```

Indexing to Improve Query Performance (cont.)

- Delete or drop an index(es):

  Syntax:

```
DROP INDEX index_name
```

  Example:

```
DROP INDEX emp_lname;
```

Dynamic SQL

- Refers to DDL, DML, and query statements that are constructed, parsed, and executed at runtime:
  - May need input from the users such as the columns they want to see or some elements of the WHERE clause

SQL Injection

- SQL injection is a type of security attack against databases in which the attacker enters SQL code to a Web form input box to maliciously gain access to resources or make changes to data.

  Example:

```
Login: ' OR ''='
Password: Anything' OR '' = '

SELECT username FROM Customer
WHERE username = '' OR '' = ''
  AND password = 'Anything' OR '' = '';
```

SQL Injection (cont.)

- Let's treat emp table storing user login table. You can treat the username column as ename column; and job column as password column.

  Login: **abc**

  Password: **xyz**

```
SELECT * FROM emp
WHERE ename = 'abc'    -- ename is username and job is password
  AND job = 'xyz';
No rows selected
```

  Login: **' OR ''='**

  Password: **Anything' OR '' = '**

```
SELECT * FROM emp
WHERE ename = '' OR '' = ''
  AND job = 'Anything' OR '' = '';
...
14 rows selected
```

  Note: Unexpected input with unexpected output!

SQL Injection (cont.)

- Web applications that use dynamic content without data validations are vulnerable to SQL injection.
  - Illegal access to databases
  - Steal sensitive information
  - Maliciously insert, modify or delete information

- Automated SQL injection programs are now available, and as a result, both the likelihood and the potential damage of an exploit has increased enormously.

SQL Injection (cont.)

- The fundamental issues
  - Programmers may not know secure coding practice.
  - Security is not sufficiently emphasized in software development.
  - Developers focus on the legal values of parameters and how they should be utilized without considering invalid input values.
  - The testers of web applications don’t do in-depth testing to discover invalidated input values and their corresponding unexpected behaviors.

SQL Injection (cont.)

- How to protect the integrity of web sites and applications:
  - Controlling the types and numbers of characters accepted by input boxes (data sanitization).
  - Calling stored procedure to login.
  - Access control over sensitive tables and columns.

- How to identify application vulnerabilities:
  - Using “AppScan” and “Netsparker” to discover vulnerable web pages.
  - Using “Fortify Software” and “Checkmarx” to scan source code.
  - Using “Core Impact” by Core Security, a penetration testing tool to validate discovered vulnerabilities.
  - Manually validating vulnerability findings if necessary.

Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 5

SQL - The Relational DB Language II

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

SQL Programming Language

- Advanced users, such as developers or DBAs, can use vendors' tools to communicate with DBMSs and support complex business processes.
- SQL*Plus, is a tool as a command-line interface with Oracle database (**`sql>`**).
- Common tools for other DBMSs are:
  - "isql" in Sybase and SQL Server
  - "db2" in IBM DB2
  - "psql" in PostgreSQL
  - "mysql" in MySQL
- SQL lacks programming features such as variable, constant declarations, flow controls, exception handling, and modulation.

SQL Programming Language (cont.)

- The server processes SQL statements one at a time. Each SQL statement results in a separate call from the client to the server, causing a overhead, especially across web or a network.

- A high-level language such as Pro*C/C++, Java can also embed SQL and perform database operations.

SQL/Persistent Stored Modules (PSM)

- SQL/PSM is an ANSI SQL extension with procedural programmability.
- Most commercial DBMSs implemented this feature before the standard.
- Procedural Language/SQL (PL/SQL) is Oracle's procedural extension to SQL.
  - Embedding SQL in a Programming Language (PL/SQL) to manipulate complex database operations.
- Similarly, Sybase and Microsoft has Transact-SQL; IBM has SQL PL (Procedural Language)

SQL/Persistent Stored Modules (cont.)

- Example: Oracle PL/SQL Implementation:
  - PL/SQL supports modular programming, declare identifiers, and uses them in a program with procedural language control structures, and error handling.
  - PL/SQL is built in a block construct.

```
[DECLARE TYPE / item / FUNCTION / PROCEDURE declarations] -- optional
BEGIN  -- mandatory
   statements
   [EXCEPTION – optional
    exception handlers]
END; -- mandatory
```

SQL/Persistent Stored Modules (cont.)

- Example: Oracle PL/SQL Implementation (cont.)
  - PL/SQL language can be written using the basic control structures such as SEQUENCE, SELECTION and ITERATION.
    - Conditional IF statement:

```
IF (condition) THEN <sql statements>
[ELSIF (condition) THEN <sql statements>]
[ELSE <sql statements>]
END IF;
```

    - Conditional CASE statement:

```
CASE (operand)
[WHEN (operand list) | WHEN (search condition)
    THEN <sql statements>]
[ELSE <sql statements>]
END CASE;
```

SQL/Persistent Stored Modules (cont.)

- Example: Oracle PL/SQL Implementation (cont.)
  - Iteration LOOP, WHILE, or FOR statements:

```
WHILE (condition)
   LOOP <sql statements>
END LOOP;

FOR iVariable IN lowerbound..upperbound
LOOP
   <sql statements>
END LOOP;
```

  - The set of rows returned by a multi-row query is called the active set. An explicit cursor is equivalent to a *point* to the current row in the active set. This allows a program to process the rows one at a time.

SQL/Persistent Stored Modules (cont.)

- Example: Oracle PL/SQL Implementation (cont.)
  - Embedding SQL in a programming language
  - Use dot notation to reference individual fields

Example: Calculate bonus based on department

```
DECLARE
        CURSOR c1 IS
        SELECT ename, salary, dept_id FROM emp;
        ...;
BEGIN
        . . .
        FOR emp_rec IN c1 LOOP
            . . .
            salary_sum := salary_sum + emp_rec.sarlary;
            ...
        END LOOP;
END;
```

SQL/Persistent Stored Modules (cont.)

- General Constraints
  - Data manipulation to tables may have complex rules beyond common structural constraints such as PKs, FKs, NOT NULL, Unique, Domain, and Check

Example: An employee cannot register two cars for parking permits.

```
CREATE ASSERTION TooManyCars_Constraint
   CHECK (NOT EXIST (SELECT emp_id
                     FROM CAR
                     GROUP BY emp_id
                     HAVING COUNT(*) > 2));
```

SQL/Persistent Stored Modules (cont.)

- General Constraints (cont.)
  - Business processes need for general constraints to support complex rules.
    - Register a course with prerequisite courses and a minimum GPA requirement.
    - Withdraw cash less than the account balance or more than a daily allowed maximum.
    - Assign no more than 10 tasks to an employee or no more than 5 tasks to an employee who has three high-priority tasks.
    - Create an audit trail of all rows inserted into the Customer_Order table.
  - Triggers can be used for complex constraints. Not all DBMSs support triggers. If not, applications need to implement general constraints.

Triggers

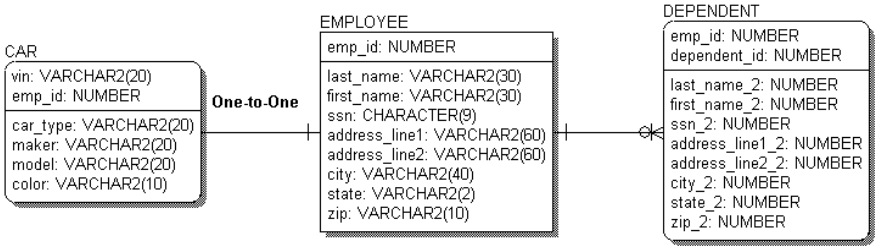
- Triggers:
  - Programs that are executed automatically in response to a change in the database for enforcing constraints or business rules. Trigger can be configured to fire or execute, either before or after the trigger event.
  - Event-Condition-Action:

```
<trigger> ::= CREATE TRIGGER <trigger name>
              ( BEFORE | AFTER ) <triggering events>
              ON <table name>
              [ FOR EACH ROW ]
              [ WHEN <condition> ]
              <trigger actions> ;  -- trigger body

<triggering events> ::=<trigger event>
                       {OR <trigger event> }
<trigger event>::=INSERT | DELETE | UPDATE [ OF <column name> {, <column name>} ]
<trigger action> ::=<PL/SQL block>
```

Triggers (cont.)

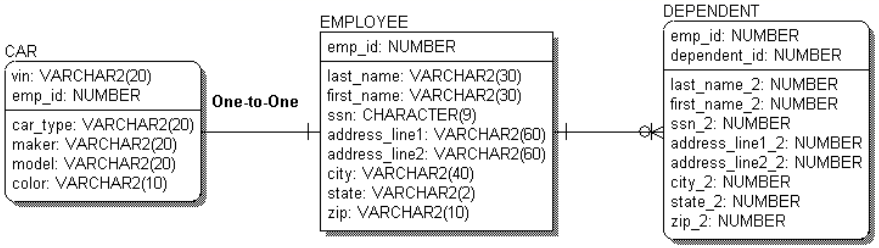
- Triggers (cont.)
  - Example: An Update Trigger




```
-- CREATE TIGGER using Oracle Syntax:
CREATE TRIGGER tU_CAR AFTER UPDATE ON car FOR EACH ROW
-- UPDATE trigger on CAR
DECLARE numrows INTEGER;
BEGIN
  /* EMPLYEE owns CAR on CHILD UPDATE RESTRICT */
  SELECT count(*) INTO numrows
  FROM employee
  WHERE :new.emp_id = EMPLOYEE.emp_id;
  IF ( numrows = 0 ) THEN
     raise_application_error(-20007,
    'Cannot UPDATE "car" because "employee" does not exist.');
  END IF;
END;
```

Triggers (cont.)

- Triggers (cont.)
  - Example: An Insert Trigger




```
-- CREATE TIGGER using Oracle Syntax:
CREATE TRIGGER tI_CAR BEFORE INSERT ON car FOR EACH ROW
-- INSERT trigger on CAR
DECLARE numrows INTEGER;
BEGIN
  /* EMPLOYEE owns only ONE car on CHILD INSERTIION */
  SELECT count(*) INTO numrows
  FROM car
  WHERE :new.emp_id = EMPLOYEE.emp_id;
  IF ( numrows = 1 ) THEN
     raise_application_error(-20008,
     'Cannot INSERT "car" because "employee" can only have one car.');
  END IF;
END;
```

Stored Procedures, Functions and Packages

- Stored Procedures, Functions and Packages:
  - Stored procedures and functions are similar to other high level programming languages to provide modulation, code reusability and maintainability.

```
CREATE OR REPLACE PROCEDURE Procedure_Name [(<parameter-list>)] AS
BEGIN
   <executable-section>
   [exception
      <exception-section>]
END;
```

  - A parameter list can be used as inputs, outputs or both. A procedure can return a set of values through the parameter list.

Stored Procedures, Functions and Packages (cont.)

- Stored Procedures, Functions and Packages (cont.)
  - A function returns a value to the calling program.

```
CREATE OR REPLACE FUNCTION Function_Name [(<parameter-list>)]
return <datatype> AS
BEGIN
   <executable-section>
   [exception
      <exception-section>]
END;
```

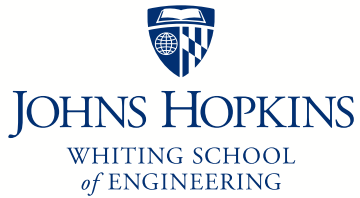
  - A package is a collection of related stored procedures, functions, and variables.

Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 6

SQL - The Relational DB Language II



Object-relational Features in SQL

- Object-relational DB supports
  - Extended datatype
    - Scalar – CHAR, NCHAR (Unicode, national character set), VARCHAR2, NVARCHAR2, NUMBER, DATE, TIMESTAMP, BFILE
    - Collection – VARRAY, TABLE
    - Relationship – REF (for object table only)
  - User-defined data types
  - Objects with attributes and member methods (procedures or functions)
  - Large objects (LOB) such as BLOB, CLOB, NCLOB

Object-relational Features in SQL (cont.)

- Creating Object Types
  - An object type consists of built-in datatypes or object types.
  - Syntax in SQL:
    - **`CREAT TYPE`**
    - **`DROP TYPE`**
    - **`ALTER TYPE`**
    - **`GRANT/REVOKE TYPE`**
  - After a type is created, a constructor method is automatically created.
  - Constructor is used to create specific instances of objects.

Object-relational Features in SQL (cont.)

- Creating Relational Tables with Object Types
  - Creating an object type

```
CREATE TYPE person_type AS OBJECT
(LastName    VARCHAR2(20),
 FirstName   VARCHAR2(20),
 Phone       VARCHAR2(12),
 DOB         DATE, …);
```

  - Creating a table with an object type

```
CREAT TABLE employee
(emp_id      INTEGER PRIMARY KEY,
 emp         person_type);
```

  - Inserting an employee record

```
INSERT INTO employee VALUES (5, person_type('Smith', 'John', '301-420-7700', To_Date(
'12/17/1978', MM/DD/YYYY'), …);
INSERT INTO employee VALUES (6, person_type('Jones', 'Lynn', '410-731-4968', To_Date(
'06/08/1980', 'MM/DD/YYYY'), … );
```

  - Query an employee record using dot notation

```
SELECT emp_id, e.emp.FirstName, e.emp.LastName, e.emp.dob FROM employee e;
```

Object-relational Features in SQL (cont.)

- Creating An Object Table
  - An object table is a table whose rows are all objects with object identifier (OID) values.
  - Store object instances in rows:

```
CREATE TYPE person_type AS OBJECT
(LastName    VARCHAR2(20),
 FirstName   VARCHAR2(20),
 Phone       VARCHAR2(12),
 DOB         DATE, …);

CREATE TABLE emp_table OF person_type;
-- Table emp_table is based on person_type datatype --
```

Object-relational Features in SQL (cont.)

- Accessing An Object in An Object Table
  - An object reference (REF) is a system generated value to locate an object in an object table.
  - A REF consists of the target's object's OID, and object table identifier, and a database identifier.
  - A REF can be used to define relationships between objects, a one-to-one unidirectional association.

Object-relational Features in SQL (cont.)

- Creating A Member Method in An Object
  - An object type can have zero or more member methods and use the object.method to access a member method.

```
CREATE TYPE emp_type AS OBJECT
(emp_id   NUMBER,
 LastName VARCHAR2(20),
 FirstName     VARCHAR2(20),
 Phone    VARCHAR2(12),
 HireDate DATE,
 MEMBER FUNCTION days_joined RETURN NUMBER, …);

CREATE TYPE BODY emp_type
(MEMBER FUNCTION days_joined RETURN NUMBER IS
 BEGIN
     RETURN floor (sysdate – hiredate);
 END; );
```

Object-relational Features in SQL (cont.)

- VARRAY Type
  - A varying-length array (VARRAY) is a collection of similar items
  - The array is an ordered set of items with two attributes
    - Count: Current number of elements
    - Limit: Maximum array size
      Data in a VARRAY stored inline if size < 4Kb
      Index is not supported

Object-relational Features in SQL (cont.)

- VARRAY Example
  - Use the object.method to access a member method

```
CREATE TYPE phone_type AS OBJECT
(Phone       VARCHAR2(12),
 Description VARCHAR2(15) );

CREATE TYPE phone_list_type AS VARRAY(10) OF phone_type;

CREATE TYPE emp_type AS OBJECT
(emp_id      NUMBER,
 LastName    VARCHAR2(20),
 FirstName   VARCHAR2(20),
 Phone_No    phone_list_type,
 HireDate    DATE, …);

CREATE TABLE employee OF emp_type;
```

Object-relational Features in SQL (cont.)

- Nested Tables
  - A table stored within a table is called a nested table.
  - It is suitable a master-detailed relationship or one-to-many relationship.
  - Query is allowed to select nested rows.
  - Nested tables are stored *out-of-line.*
  - DMBS supports indexes for nested tables.

Object-relational Features in SQL (cont.)

- Nested Table Example
  - How to create a nested table and use it

```
CREATE TYPE phone_type AS OBJECT
(Phone          VARCHAR2(12),
 Description    VARCHAR2(15) );

CREATE TYPE phone_nested_type AS TABLE OF phone_type;

CREATE TYPE employee
(emp_id       NUMBER,
 LastName     VARCHAR2(20),
 FirstName    VARCHAR2(20),
 Phone_No     phone_nested_type,
 HireDate     DATE, …)
 NESTED TABLE Phone_No STORE AS emp_phone_nested_table_seg;
(Note: Nested table stored out-of-line in its own segment)
```