



# SQL – Data Manipulation

## Chapter Six



# Objectives of SQL



- Ideally, database language should allow user to:
  - create the database and relation structures;
  - perform insertion, modification, deletion of data from relations;
  - perform simple and complex queries.
- Must perform these tasks with minimal user effort and command structure/syntax must be easy to learn.
- It must be portable.
- SQL is a transform-oriented language with 2 major components:
  - A DDL for defining database structure.
  - A DML for retrieving and updating data.
- Until SQL:1999, SQL did not contain flow of control commands. These had to be implemented using a programming or job-control language, or interactively by the decisions of user.
- SQL is relatively easy to learn:
  - it is non-procedural - you specify *what* information you require, rather than *how* to get it;
  - it is essentially free-format.

# Objectives of SQL

- Consists of standard English words:

- 1) CREATE TABLE Staff(staffNo VARCHAR(5),  
                          IName VARCHAR(15),  
                          salary DECIMAL(7,2));
- 2) INSERT INTO Staff VALUES ('SG16', 'Brown', 8300);
- 3) SELECT staffNo, IName, salary  
      FROM Staff  
      WHERE salary > 10000;

- Can be used by range of users including DBAs, management, application developers, and other types of end users.
- An ISO standard now exists for SQL, making it both the formal and *de facto* standard language for relational databases.



# History of SQL

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called 'Structured English Query Language' (SEQUEL).
- A revised version, SEQUEL/2, was defined in 1976 but name was subsequently changed to SQL for legal reasons.
- Still pronounced 'see-quel', though official pronunciation is 'S-Q-L'.
- IBM subsequently produced a prototype DBMS called *System R*, based on SEQUEL/2.
- Roots of SQL, however, are in SQUARE (Specifying Queries as Relational Expressions), which predates System R project.



# History of SQL



- In late 70s, ORACLE appeared and was probably first commercial RDBMS based on SQL.
- In 1987, ANSI and ISO published an initial standard for SQL.
- In 1989, ISO published an addendum that defined an 'Integrity Enhancement Feature'.
- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92.
- In 1999, SQL:1999 was released with support for object-oriented data management.
- In late 2003, SQL:2003 was released.
- In summer 2008, SQL:2008 was released.
- In late 2011, SQL:2011 was released.



# Importance of SQL

- SQL has become part of application architectures such as IBM's Systems Application Architecture.
- It is strategic choice of many large and influential organizations (e.g. X/OPEN).
- SQL is Federal Information Processing Standard (FIPS) to which conformance is required for all sales of databases to American Government.
- SQL is used in other standards and even influences development of other standards as a definitional tool. Examples include:
  - ISO's Information Resource Directory System (IRDS) Standard
  - Remote Data Access (RDA) Standard.

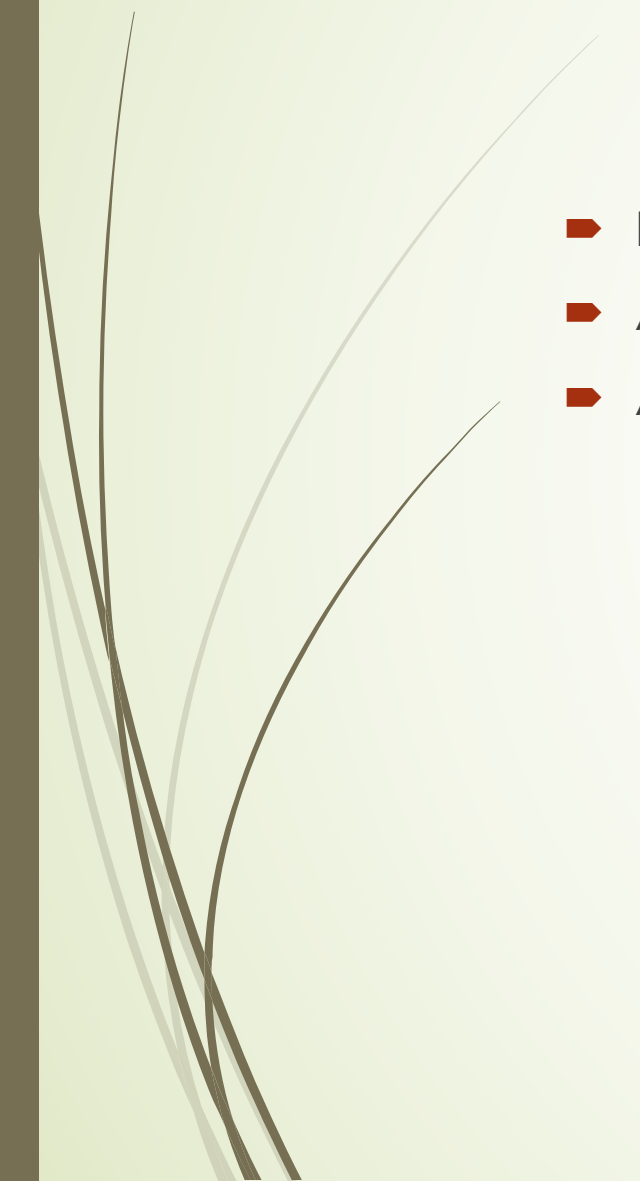
# Writing SQL Commands

- SQL statement consists of *reserved words* and *user-defined words*.
  - Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.
  - User-defined words are made up by user and represent names of various database objects such as relations, columns, views.
- Most components of an SQL statement are *case insensitive*, except for literal character data.
- More readable with indentation and lineation:
  - Each clause should begin on a new line.
  - Start of a clause should line up with start of other clauses.
  - If clause has several parts, should each appear on a separate line and be indented under start of clause.
- Use extended form of BNF notation:
  - Upper-case letters represent reserved words.
  - Lower-case letters represent user-defined words.
  - | indicates a *choice* among alternatives.
  - Curly braces indicate a *required element*.
  - Square brackets indicate an *optional element*.
  - ... indicates *optional repetition* (0 or more).





# Literals

- ▶ Literals are constants used in SQL statements.
  - ▶ All non-numeric literals must be enclosed in single quotes (e.g. 'London').
  - ▶ All numeric literals must not be enclosed in quotes (e.g. 650.00).
- 





# SELECT Statement

```
SELECT [DISTINCT | ALL]
    { * | [columnExpression [AS newName]] [, ...] }
FROM      TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList] [HAVING condition]
[ORDER BY columnList]
```



# SELECT Statement

FROM Specifies table(s) to be used.

WHERE Filters rows.

GROUP BY Forms groups of rows with same  
column value.

HAVING Filters groups subject to some  
condition.

SELECT Specifies which columns are to  
appear in output.

ORDER BY Specifies the order of the output.



# SELECT Statement

- Order of the clauses cannot be changed.
  - Only SELECT and FROM are mandatory.
- 



# SELECT Statement -- Aggregates

- ISO standard defines five aggregate functions:
  - COUNT returns number of values in specified column.
  - SUM returns sum of values in specified column.
  - AVG returns average of values in specified column.
  - MIN returns smallest value in specified column.
  - MAX returns largest value in specified column.
- Each operates on a single column of a table and returns a single value.
- COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG may be used on numeric fields only.
- Apart from COUNT(\*), each function eliminates nulls first and operates only on remaining non-null values.

# SELECT Statement -- Aggregates

- ▶ COUNT(\*) counts all rows of a table, regardless of whether nulls or duplicate values occur.
- ▶ Can use DISTINCT before column name to eliminate duplicates.
- ▶ DISTINCT has no effect with MIN/MAX, but may have with SUM/AVG.
- ▶ Aggregate functions can be used only in SELECT list and in HAVING clause.
- ▶ If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot reference a column out with an aggregate function. For example, the following is illegal:


```
SELECT staffNo, COUNT(salary)
FROM Staff;
```

# SELECT Statement -- Grouping

- Use GROUP BY clause to get sub-totals.
- SELECT and GROUP BY closely integrated: each item in SELECT list must be *single-valued per group*, and SELECT clause may only contain:
  - column names
  - aggregate functions
  - constants
  - expression involving combinations of the above.
- All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function.
- If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.
- ISO considers two nulls to be equal for purposes of GROUP BY.



# Restricted Groupings – HAVING clause

- HAVING clause is designed for use with GROUP BY to restrict groups that appear in final result table.
  - Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
  - Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.
- 





# Subqueries



- ▶ Some SQL statements can have a SELECT embedded within them.
- ▶ A subselect can be used in WHERE and HAVING clauses of an outer SELECT, where it is called a *subquery* or *nested query*.
- ▶ Subselects may also appear in INSERT, UPDATE, and DELETE statements.



# Subqueries Rules



- ORDER BY clause may not be used in a subquery (although it may be used in outermost SELECT).
- Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.
- By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an *alias*.
- When subquery is an operand in a comparison, subquery must appear on right-hand side.
- A subquery may not be used as an operand in an expression.



# Multi-Table Queries

- Can use subqueries provided result columns come from same table.
- If result columns come from more than one table must use a join.
- To perform join, include more than one table in FROM clause.
- Use comma as separator and typically include WHERE clause to specify join column(s).
- Also possible to use an alias for a table named in FROM clause.
- Alias is separated from table name with a space.
- Alias can be used to qualify column names when there is ambiguity.



# Alternative JOIN Constructs

- SQL provides alternative ways to specify joins:

FROM Client c JOIN Viewing v ON c.clientNo = v.clientNo

FROM Client JOIN Viewing USING clientNo

FROM Client NATURAL JOIN Viewing

- In each case, FROM replaces original FROM and WHERE. However, first produces table with two identical clientNo columns.

# Computing a Join

- Procedure for generating results of a join are:
  1. Form Cartesian product of the tables named in FROM clause.
  2. If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition.
  3. For each remaining row, determine value of each item in SELECT list to produce a single row in result table.
  4. If DISTINCT has been specified, eliminate any duplicate rows from the result table.
  5. If there is an ORDER BY clause, sort result table as required.
- SQL provides special format of SELECT for Cartesian product:

```
SELECT  [DISTINCT | ALL]    {*} | columnList  
FROM Table1 CROSS JOIN Table2
```

# OUTER JOIN

- The OUTER JOIN syntax can be used to obtain data that exists in one table without matching data in the other table.

**STUDENT**

StudentPK	StudentName	LockerFK
1	Adams	NULL
2	Buchanan	NULL
3	Carter	10
4	Ford	20
5	Hoover	30
6	Kennedy	40
7	Roosevelt	50
8	Truman	60

**LOCKER**

LockerPK	LockerType
10	Full
20	Full
30	Half
40	Full
50	Full
60	Half
70	Full
80	Full
90	Half

(a) The STUDENT and LOCKER Tables Aligned to Show Row Relationships

# LEFT OUTER JOIN

All rows from STUDENT are shown, even where there is no matching LockerFK=LockerPK value

StudentPK	StudentName	LockerFK	LockerPK	LockerType
1	Adams	NULL	NULL	NULL
2	Buchanan	NULL	NULL	NULL
3	Carter	10	10	Full
4	Ford	20	20	Full
5	Hoover	30	30	Half
6	Kennedy	40	40	Full
7	Roosevelt	50	50	Full
8	Truman	60	60	Half

(c) LEFT OUTER JOIN of the STUDENT and LOCKER Tables



# RIGHT OUTER JOIN

All rows from  
LOCKER are shown,  
even where there is no  
matching  
LockerFK=LockerPK  
value

StudentPK	StudentName	LockerFK	LockerPK	LockerType
3	Carter	10	10	Full
4	Ford	20	20	Full
5	Hoover	30	30	Half
6	Kennedy	40	40	Full
7	Roosevelt	50	50	Full
8	Truman	60	60	Half
NULL	NULL	NULL	70	Full
NULL	NULL	NULL	80	Full
NULL	NULL	NULL	90	Half

(d) RIGHT OUTER JOIN of the STUDENT and LOCKER Tables

# INNER JOIN

Only the rows where  $\text{LockerFK} = \text{LockerPK}$  are shown—Note that some StudentPK and some LockerPK values are not in the results

StudentPK	StudentName	LockerFK	LockerPK	LockerType
3	Carter	10	10	Full
4	Ford	20	20	Full
5	Hoover	30	30	Half
6	Kennedy	40	40	Full
7	Roosevelt	50	50	Full
8	Truman	60	60	Half

(b) INNER JOIN of the STUDENT and LOCKER Tables



# EXISTS and NOT EXISTS

- ▶ EXISTS and NOT EXISTS are for use only with subqueries.
- ▶ Produce a simple true/false result.
- ▶ True if and only if there exists at least one row in result table returned by subquery.
- ▶ False if subquery returns an empty result table.
- ▶ NOT EXISTS is the opposite of EXISTS.
- ▶ As (NOT) EXISTS check only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns.
- ▶ Common for subqueries following (NOT) EXISTS to be of form:

(SELECT \* ...)



# INSERT

INSERT INTO TableName [ (columnList) ]  
VALUES (dataValueList)

- ▶ *columnList* is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- ▶ Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.
- ▶ *dataValueList* must match *columnList* as follows:
  - ▶ number of items in each list must be same;
  - ▶ must be direct correspondence in position of items in two lists;
  - ▶ data type of each item in *dataValueList* must be compatible with data type of corresponding column.



# INSERT...SELECT

- Second form of INSERT allows multiple rows to be copied from one or more tables to another:

```
INSERT INTO TableName [ (columnList) ]  
    SELECT ...
```



# UPDATE

```
UPDATE TableName  
SET columnName1 = dataValue1  
    [, columnName2 = dataValue2...]  
[WHERE searchCondition]
```

- *TableName* can be name of a base table or an updatable view.
- SET clause specifies names of one or more columns that are to be updated.
- WHERE clause is optional:
  - if omitted, named columns are updated for all rows in table;
  - if specified, only those rows that satisfy *searchCondition* are updated.
- New *dataValue(s)* must be compatible with data type for corresponding column.



# DELETE

DELETE FROM TableName  
[WHERE searchCondition]

- *TableName* can be name of a base table or an updatable view.
- *searchCondition* is optional; if omitted, all rows are deleted from table. This does not delete table. If *search\_condition* is specified, only those rows that satisfy condition are deleted.