Common SQL Data Types

DATA TYPE	FORMAT	COMMENTS
Numeric	NUMBER(L,D) or NUMERIC(L,D)	The declaration NUMBER(7,2) or NUMERIC(7,2) indicates that numbers will be stored with two decimal places and may be up to seven digits long, including the sign and the decimal place (for example, 12.32 or –134.99).
	INTEGER	May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.
	SMALLINT	Like INTEGER but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.
	DECIMAL(L,D)	Like the NUMBER specification, but the storage length is a *minimum* specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.
Character	CHAR(L)	Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as *Smith* and *Katzenjammer* are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes.
	VARCHAR(L) or VARCHAR2(L)	Variable-length character data. The designation VARCHAR2(25) or VARCHAR(25) will let you store characters up to 25 characters long. However, unlike CHAR, VARCHAR will not leave unused spaces. Oracle automatically converts VARCHAR to VARCHAR2.
Date	DATE	Stores dates in the Julian date format.

SQL Date and DateTime Data Types

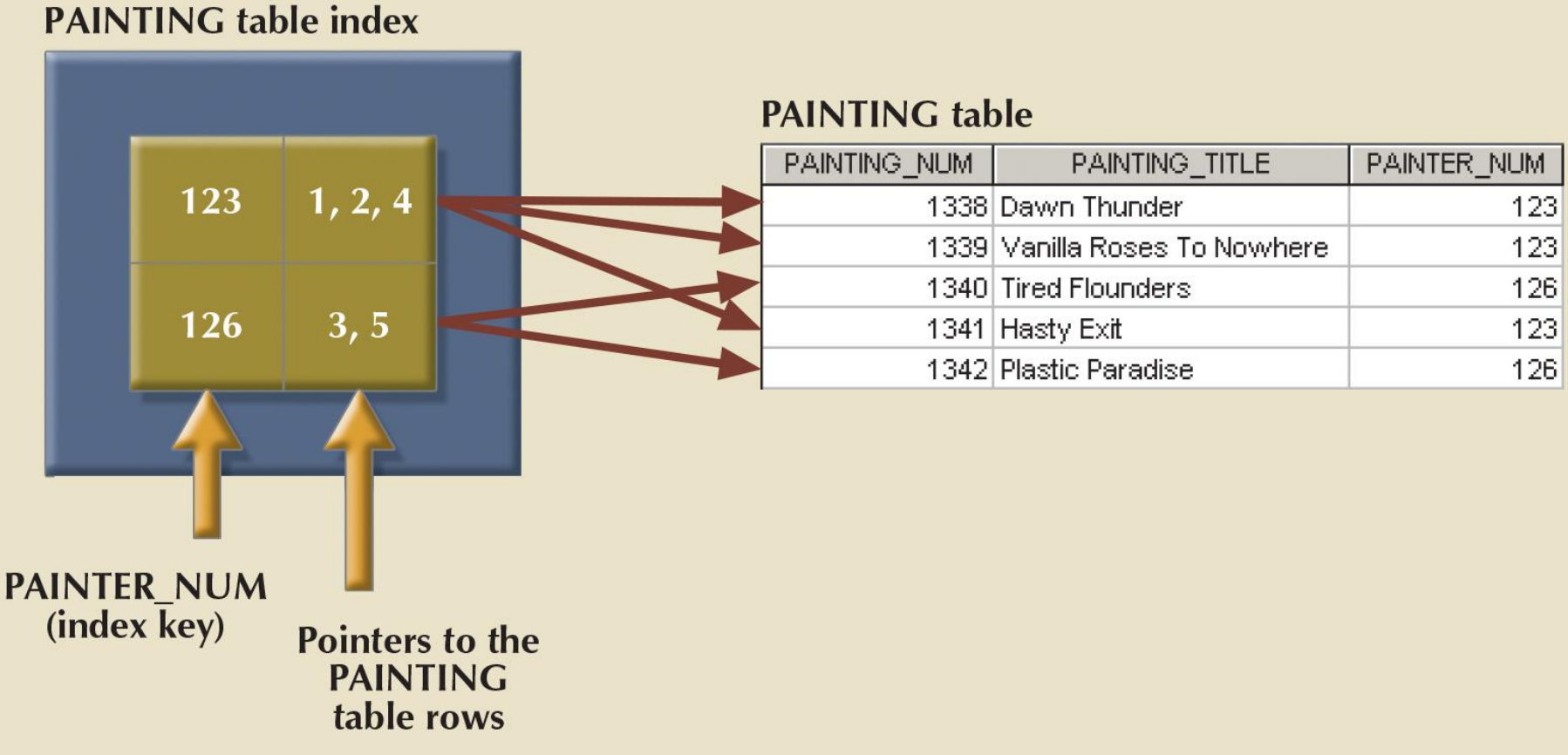
- The DATE type is used for values with a date part but no time part.
  - MySQL retrieves and displays DATE values in 'YYYY-MM-DD' format.
  - The supported range is '1000-01-01' to '9999-12-31'.
- The DATETIME type is used for values that contain both date and time parts.
  - MySQL retrieves and displays DATETIME values in 'YYYY-MM-DD HH:MM:SS' format.
  - The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59‘
- TIMESTAMP is used to record the date and time of an event
  - The time zone used is the server’s time zone
  - The supported range is '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC

Indexes

Indexes

- Indexes are created to provide quick access to data
  - Orderly arrangement to logically access rows in a table
- **Index key**: Index's reference point that leads to data location identified by the key
- **Unique index**: Index key can have only one pointer value associated with it
- Each index is associated with only one table
  - One table can have several indexes
  - Index is automatically created on the primary key column

Indexes


PAINTING table index
123
1, 2, 4
126
3, 5
PAINTER_NUM
(index key)
Pointers to the
PAINTING
table rows
PAINTING table
PAINTING_NUM
PAINTING_TITLE
PAINTER_NUM
1338 Dawn Thunder 123
1339 Vanilla Roses To Nowhere 123
1340 Tired Flounders 126
1341 Hasty Exit 123
1342 Plastic Paradise 126

SQL Indexes

- When primary key is declared, DBMS automatically creates unique index
- The **CREATE INDEX** command can be used to create indexes on the basis of any selected attribute
- **UNIQUE** qualifier prevents a value that has been used before
  - Composite indexes prevent data duplication
- To delete an index use the **DROP INDEX** command

SQL Indexes - Examples

Syntax:

CREATE [UNIQUE] INDEX *indexname*

ON *tablename* (*col1* [, *col2*]);

Examples:

```
CREATE UNIQUE INDEX P_CODEX
    ON PRODUCT (P_CODE);   -- Creates index on column P_CODE
CREATE INDEX PROD_PRICEX
    ON PROD (P_PRICE DESC);  -- Creates index in desc. order
DROP   INDEX   PROD_PRICEX;  -- Deletes index PROD_PRICEX
```

Modifying Table Structure

Modifying Table Structure

- **ALTER TABLE** command: To make changes in the table structure
- Keywords used with the command
  - ADD - Adds a column
  - MODIFY - Changes column characteristics
  - DROP - Deletes a column
- Also used to:
  - Add table constraints
  - Remove table constraints

Changing a Column's Data Type and Data Characteristics

- ALTER used to change data type and characteristics
  - Some RDBMSs do not permit changes to data types unless column is empty
  - Changes in characteristics are permitted if they do not alter the existing data type
- Syntax:
  - Data Type: ALTER TABLE *tablename* MODIFY *(columnname(datatype))*;
  - Data Characteristic: ALTER TABLE *tablename* MODIFY *(columnname(characteristic))*;

Adding and Dropping Columns

- Adding a column
  - Use ALTER and ADD
  - Do not include the NOT NULL clause for new column
- Dropping a column
  - Use ALTER and DROP
  - Some RDBMSs impose restrictions on the deletion of an attribute

ALTER TABLE – Examples

This command adds a new column to the PRODUCT table

```
ALTER TABLE PRODUCT
    ADD (P_SALECODE CHAR(1));
```

This command modifies the column width

```
ALTER TABLE PRODUCT
    MODIFY  P_SALECODE CHAR(2);
```

This command deletes the column

```
ALTER TABLE PRODUCT
    DROP COLUMN P_SALECODE;
```

Deleting a Table from the Database

- **DROP TABLE**: Deletes table from database
  - Syntax - DROP TABLE *tablename*;
- Can drop a table only if it is not the one side of any relationship
  - RDBMS generates a foreign key integrity violation error message if you try to drop a referenced table

SQL’s Data Manipulation Language (DML)

Adding Data to a Table

- Add Table rows using the INSERT command

  INSERT INTO *tablename*

  VALUES (*value1, value2, …, valueN*);

- Example:

  INSERT INTO VENDOR

  VALUES (21225, ‘Bryson, Inc.’, ‘Smithson’, ‘615’, ‘223-3234’, ‘TN’, ‘Y’);

- To view all data in the table, use the following command:

  SELECT * FROM VENDOR

  - More on SELECT later

Adding Rows with Optional Attributes

- All NOT NULL columns need to be included in the INSERT command for adding a table row
- What to do when tables have several optional columns and no data needs to be added yet?
  - Use list of column names to specify what data is being entered
  - Example:

```
INSERT INTO PRODUCT (P_CODE, P_DESCRIPT)
    VALUES ('BRT-345', 'Titanium drill bit');
```

(Note: We are assuming here that only 2 columns are NOT NULL)

Saving Table Changes

- Changes not made permanent until saved in database
  - Power outage may result in loss of data
- Table contents can be saved by using the COMMIT command
- Syntax:
  START TRANSACTION (or BEGIN [WORK])

  COMMIT [WORK];
- COMMIT command permanently saves all changes made to any table in the database.

Restoring Table Contents

- Database can be restored to its previous condition using the ROLLBACK command
  - The changes should not have been permanently stored in the database through the COMMIT command
- Syntax:

  ROLLBACK [WORK];
- COMMIT and ROLLBACK only work with the data manipulation commands that add, modify or delete table rows

Restoring Table Contents

- Example:
  1. CREATE a table called SALES
  2. INSERT 10 rows in the SALES table
  3. UPDATE 2 rows in the SALES table
  4. Execute the ROLLBACK command

- What does the ROLLBACK command do?
  - ROLLBACK will only undo the results of the INSERT and UPDATE commands

Deleting Table Rows

- Syntax:

  DELETE FROM *tablename*

  [WHERE *conditionlist*];

- Examples:
  - To delete all data from the PRODUCT table

    DELETE FROM PRODUCT;
  - To delete all rows with P_MIN = 5

    DELETE FROM PRODUCT

    WHERE P_MIN = 5;

Inserting Table Rows with SELECT

- Select subquery can be used to add multiple rows to a table, using another table as the source of data
  - Subquery is also called a nested query or inner query
- Syntax:

  INSERT INTO *tablename*
  SELECT *columnlist* FROM *tablename*;

  - Example:

    INSERT INTO TMP_PROD
    SELECT P_CODE FROM PRODUCTS;

Listing Table Data – The SELECT statement

- SELECT command is used to list table contents
- Syntax:

SELECT *columnlist* FROM *tablename*;

- Example:
  - Listing all table data

    SELECT * FROM VENDOR;

  - Listing selected columns

    SELECT P_DESCRIPT, P_PRICE FROM PRODUCT;

SELECT with Conditional Restrictions

- Syntax:

  SELECT *columnlist*
  FROM *tablelist*
  [WHERE *conditionlist*];

- Example: Display all products supplied by V_CODE 21344

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
    FROM  PRODUCT
    WHERE V_CODE = 21344
```

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
7.25-in. pwr. saw blade	13-Dec-15	14.99	21344
9.00-in. pwr. saw blade	13-Nov-15	17.49	21344
Rat-tail file, 1/8-in. fine	15-Dec-15	4.99	21344

Comparison Operators

- Adds conditional restrictions on selected character attributes and dates

COMPARISON OPERATORS	
SYMBOL	**MEANING**
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

SELECT with Conditional Restrictions

- Example 2: Display all products not supplied by V_CODE = 21344

```
SELECT      *
    FROM    PRODUCT
    WHERE   V_CODE <> 21344;
```

- Example 3: Display description and price of products with price less than 10

```
SELECT      P_DESCRIPT, P_PRICE
    FROM    PRODUCT
    WHERE   P_PRICE < 10;
```

Logical Operators: AND, OR and NOT

- **OR** and **AND**: Used to link multiple conditional expressions in a WHERE or HAVING clause
  - **OR** requires only one of the conditional expressions to be true
  - **AND** requires all of the conditional expressions to be true
- **NOT** is used to negate the result of a conditional expression

Selected PRODUCT Table Attributes: The Logical OR

Return selected columns for products with V_CODE = 21344 or 24288

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
7.25-in. pwr. saw blade	13-Dec-15	14.99	21344
9.00-in. pwr. saw blade	13-Nov-15	17.49	21344
B&D jigsaw, 12-in. blade	30-Dec-15	109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-15	99.87	24288
Rat-tail file, 1/8-in. fine	15-Dec-15	4.99	21344
Hicut chain saw, 16 in.	07-Feb-16	256.99	24288

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
  FROM    PRODUCT
  WHERE   V_CODE = 21344 OR V_CODE = 24288;
```

Selected PRODUCT Table Attributes: The Logical AND

Return selected columns for products with
P_PRICE < 50 AND P_INDATE > '15-Jan-2016'

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
B&D cordless drill, 1/2-in.	20-Jan-16	38.95	25595
Claw hammer	20-Jan-16	9.95	21225
PVC pipe, 3.5-in., 8-ft	20-Feb-16	5.87	
1.25-in. metal screw, 25	01-Mar-16	6.99	21225
2.5-in. wd. screw, 50	24-Feb-16	8.45	21231

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
    FROM  PRODUCT
    WHERE P_PRICE < 50 AND P_INDATE > ' 2016-01-15';
```

Selected PRODUCT Table Attributes: The Logical AND and OR

Return selected columns for Products
either with P_PRICE < 50 AND P_INDATE > '15-Jan-2016'
OR with V_CODE = 24288

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
B&D jigsaw, 12-in. blade	30-Dec-15	109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-15	99.87	24288
B&D cordless drill, 1/2-in.	20-Jan-16	38.95	25595
Claw hammer	20-Jan-16	9.95	21225
Hicut chain saw, 16 in.	07-Feb-16	256.99	24288
PVC pipe, 3.5-in., 8-ft	20-Feb-16	5.87	
1.25-in. metal screw, 25	01-Mar-16	6.99	21225
2.5-in. wd. screw, 50	24-Feb-16	8.45	21231

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
    FROM  PRODUCT
    WHERE (P_PRICE < 50 AND P_INDATE > '2016-01-15')
          OR V_CODE = 24288;
```

Comparison Operators: Computed Columns and Column Aliases

- SQL accepts any valid expressions/formulas in the computed columns
- **Alias**: Alternate name given to a column or table in any SQL statement to improve the readability
- Computed column, an alias, and date arithmetic can be used in a single query

Arithmetic Operators

- **The Rule of Precedence**: Establish the order in which computations are completed
- Performed in this order:
  - Operations within parentheses
  - Power operations
  - Multiplications and divisions
  - Additions and subtractions
- Remember BODMAS?
  - Bracket, Order, Division, Multiplication, Addition, Subtraction

The Arithmetic Operators

THE ARITHMETIC OPERATORS	
OPERATOR	**DESCRIPTION**
+	Add
-	Subtract
*	Multiply
/	Divide
^	Raise to the power of (some applications use ** instead of ^)

Note: MySQL uses the function Power(a,b) or Pow(a,b) to return a^b.
– e.g. pow(3,2) returns 9

Special Operators

BETWEEN

- Checks whether attribute value is within a range

IS NULL

- Checks whether attribute value is null

LIKE

- Checks whether attribute value matches given string pattern

IN

- Checks whether attribute value matches any value within a value list

EXISTS

- Checks if subquery returns any rows

Examples – BETWEEN and IS NULL

BETWEEN

- SELECT * FROM PRODUCT
  WHERE P_PRICE BETWEEN 50 AND 100;

  is the same as:

- SELECT * FROM PRODUCT
  WHERE P_PRICE >= 50 AND P_PRICE <= 100;

- Note: BETWEEN 100 AND 50 would be the same as
  WHERE P_PRICE <= 50 AND P_PRICE >= 100; this will be incorrect.

IS NULL

- SELECT * FROM PRODUCT WHERE V_CODE IS NULL;

Example - LIKE

- SELECT * FROM VENDOR
  WHERE V_CONTACT LIKE ‘SMITH%’;
  (returns Smith and Smithson from the VENDOR data file shown earlier)

- SELECT * FROM VENDOR
  WHERE V_CONTACT LIKE ‘S%’;
  (returns Singh, Smith and Smithson from the data file shown earlier)

- SELECT * FROM VENDOR
  WHERE V_CONTACT NOT LIKE ‘S%’;
  (returns everyone except Singh, Smith and Smithson from the data file shown earlier)

Wildcards

- % sign = all *following* or *preceding* characters
  - Examples:
    J% => Jim, Jane, Jules, Jones, Johnson, July, …
    Jo% => Jones, Johnson
    %ul% => Jules, July
- _ character = any *one* character
  - Example
    _23-456-6789 => 123-456-6789, 223-456-6789, …
    123-456-6__9 => 123-456-6119, 123-456-6129, …

Example – IN

- SELECT * FROM PRODUCT
  WHERE V_CODE IN (21344, 24288);

- SELECT * FROM VENDOR
  WHERE V_CODE
  IN (SELECT V_CODE FROM PRODUCT)
  - Inner query executed first, returning a list of V_CODEs from the PRODUCT table
  - The IN operator compares the values generated from the sub-query to the V_CODE in the VENDOR table, and select only rows with matching values

Example – EXISTS

- Used to execute a command based on the results of another query
- SELECT * FROM VENDOR
  WHERE EXISTS (SELECT * FROM PRODUCT
  WHERE P_QOH <= P_MIN);
  The main query will execute only if the sub-query returns any rows. It will not be executed otherwise
- SELECT V_CONTACT FROM VENDOR
  WHERE EXISTS (SELECT * FROM PRODUCT
  WHERE P_QOH < P_MIN*2);

Ordering a Listing

- **ORDER BY** clause is useful when listing order is important
- Syntax - SELECT *columnlist*

  FROM *tablelist*

  [WHERE *conditionlist*]

  [ORDER BY *columnlist* [ASC | DESC]];
- **Cascading order sequence**: Multilevel ordered sequence
  - Created by listing several attributes after the ORDER BY clause

Ordering a Listing

- SELECT * FROM PRODUCT
  ORDER BY P_PRICE
  (*lists all products in ascending order of their price*)
- SELECT P_DESCRIPT, P_PRICE FROM PRODUCT
  WHERE P_DESCRIPT LIKE '%hammer%'
  ORDER BY P_PRICE DESC
- SELECT * FROM PRODUCT
  ORDER BY P_DESCRIPT, P_PRICE DESC
  (*lists all products ordered by description in ascending order and then by their price in descending order*)